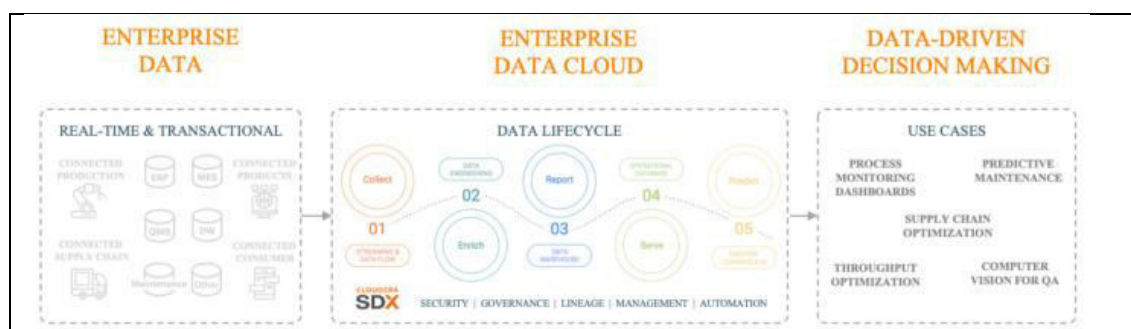


Contents

Chapter-1: Cloudera CDP Platform	1
Chapter-2: CDP Private Cloud	7
Chapter-3: CDP Private Cloud Data Services.....	8
Chapter-4: Cloudera Manager	11
Chapter-5: Apache Atlas	17
Chapter-6: Apache Ranger	18

Chapter-1: Cloudera CDP Platform

- **About Cloudera:** The Cloudera is a Data Company, and it helps people to turn data into clear and actionable insights. You can also say, Cloudera is an Enterprise Data Cloud Company. Cloudera (Leader among all), Hortonworks, MapR made the BigData market. Cloudera adopted new technologies to keep itself afloat like public cloud, public cloud storage, Kubernetes because they really make sense in current time, give benefits to customers. You must focus on things that don't change and since last 100's of years it is a truth that enterprises still want to turn data into insights. And that is what Cloudera do and will continue to do so. It has been a bit harder for BigData company to convince what they do in the BigData. Still Cloudera CDP is the only available platform to analyse petabytes of data. "If I use CDP with Spark running on Kubernetes analyzing data residing in S3, where is Hadoop?" *As long as you use the CDP service, its Hadoop only, because all came from Hadoop only.*
- **Enterprise Data Cloud:** In 2018, Cloudera and Hortonworks announced they would merge to form a single company. This merger completed in January 2019. Its stated goal was to produce the first enterprise data cloud, with a platform to support hybrid and multicloud deployments, and contain 100% open-source components. Originally released as CDP Data Center, CDP Private Cloud Base is the first release from the combined company. It integrates the best of Cloudera and Hortonworks technologies into an on-premises offering.



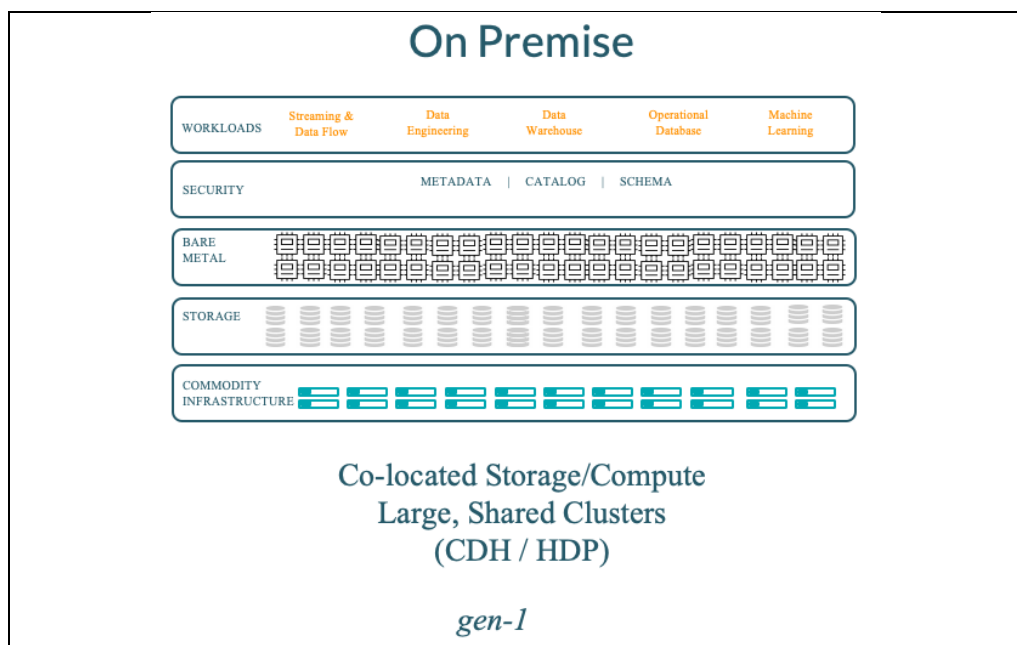
- **Cloudera Data Platform:** CDP can be deployed in a public cloud, in an on-premises data center, and as an on-premises private cloud. CDP is an enterprise data platform that is easy to deploy, manage and use. By simplifying operations, CDP reduces the time to onboard new use cases. CDP manages data in any environment, including multiple public clouds, private cloud, and hybrid cloud. CDP is a new approach to enterprise data, running anywhere from

the Edge to AI. CDP delivers easy-to-use analytics that support the most complex, demanding use cases: Complete: All functions needed to ingest, transform, query, optimize, and make predictions from data are available, eliminating the need for point products Integrated: Unified analytic functions work together eliminating data silos and copies of data. Ensures all workloads on the platform share common security, governance, and metadata. Users can efficiently find, curate, and share data, enabling self-service access to trusted data and analytics. With CDP businesses manage and secure the end-to-end data lifecycle - collecting, enriching, analyzing, experimenting and predicting with their data - to drive actionable insights and data-driven decision making. The most valuable and transformative business use cases require multi-stage analytic pipelines to process enterprise data sets. CDP empowers businesses to unlock value from large-scale, complex, distributed, and rapidly changing data and compete in the age of digital transformation. CDP is just another “distro” (i.e. “unity distro”) of the two parent distros (CDH & HDP).

- **Hybrid Cloud:** CDP Delivers same data management and analytics capabilities seamlessly across private and public clouds. CDP runs analytic workloads on multiple Public Clouds, on-premises Private Clouds or a combination of each for Hybrid Cloud, with a consistent user and IT admin experience. Single cloud providers can't match this flexibility, which is critical for data governance, performance and geographic coverage.
- **CDP Helpful in different scenarios:**
 - **Business Use cases:** Data Privacy regulations i.e., GDPR, Different types of analytics.
 - **Business Objectives:** Speed of deployment, cost efficiency etc.
 - **Technical Considerations:** Source and location of data, elasticity required etc.
- **CDP SDX:** CDP's SDX (Shared Data Experience) technologies ensures enterprise data cloud is secure by design:
 - **Consistency:** Security and governance policies are set once and applied across all data and workloads.
 - **Portability:** Policies stay with the data even as it moves across all supported infrastructures.
 - **SDX ensures** consistent data security, governance, and control across the data lifecycle and across all environments while mitigating risks and costs.
 - SDX provides data access policies and state information and state information (metadata, data lineage, metrics, audit trails and more) as an always-on layer in CDP to meet the needs for ever increasing regulatory compliance, reduce business and security risks in handling sensitive data as well as provide safe and agile self-service access to data and analytics.
 - With SDX, enterprises set data access controls and policies once and they will be automatically and seamlessly enforced across data and analytics in hybrid as well as multi-cloud deployments, even as data and workloads move between them.
 - SDX makes it trivial to setup up fully-secured data-lakes with ABAC (Attribute based access control, this really a good thing to understand if you don't know) and fine-grained policies across data stored in object stores and on-prem HDFS, along-with lineage & provenance for governance and encryption (storage, and on the wire).
 - A shared data experience (SDX) set of pipelines also makes it possible to apply the same security and governance model to any instance of CDE (Cloudera Data Engineering)
 -
- **CDP as SaaS:** Cloudera Data Platform is also available as SaaS on AWS, Azure and Google Cloud. Software as a service is a delivery model for software applications whereby the

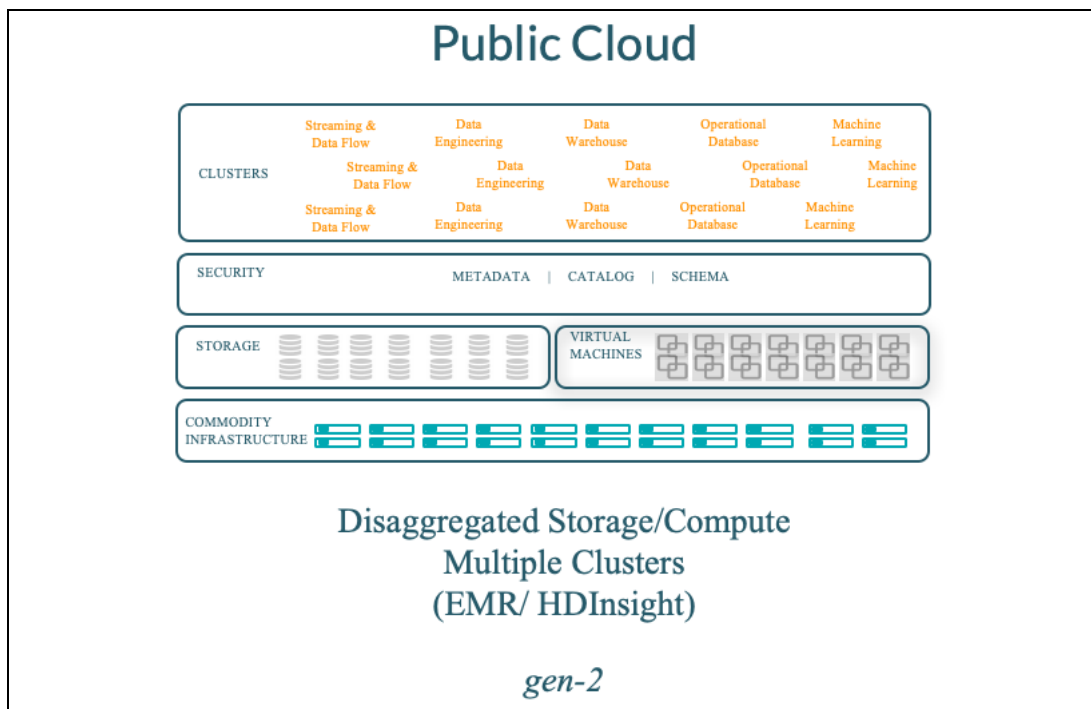
vendor hosts and operates the application over the Internet. Customers pay for using the software without owning the underlying infrastructure. With SaaS Contracts, customers will pay for usage through their AWS bill.

- **CDP for Edge to AI:**
 - The integration of streaming, analytics, and machine learning (the data lifecycle) is now recognized as a prerequisite for nearly every data-driven business use case.
 - By enabling companies to get control of data, across all environments, companies are able to master the data lifecycle to get insights that improve productivity and continue their transformation to being data-driven organizations.
- **Hadoop Philosophy:** Analysing data based on the followings
 - Disaggregate the software stack i.e. storage, compute, security and governance.
 - Build for extremely large-scale using distributed system and commodity infrastructure.
 - Leverage open source for open standards and community scale.
 - Contiguously evolve the ecosystem for innovation at every layer, independently.
- **Hadoop on Premises:** Architecture of Hadoop similar to below in a Data Center



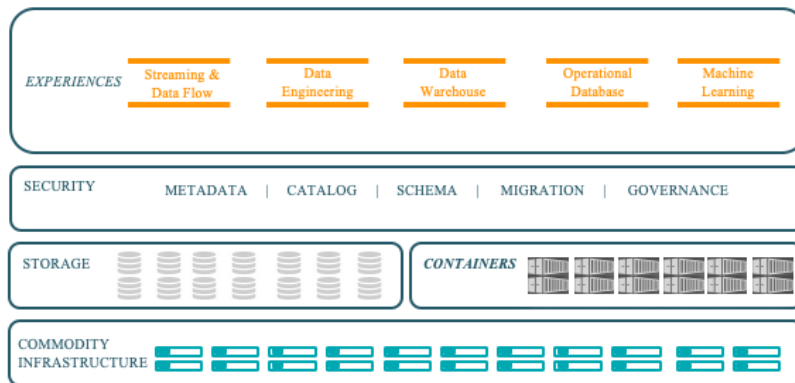
- **Co-located Compute and Storage:** networking (at scale) was expensive and slow relative to data volumes and caching (RAM & SSDs) being expensive (at scale).
- Large-scale, multi-tenant clusters as a shared resource with clusters exceeding 5,000 nodes at the high-end with strong focus on resource management across millions of batch applications (YARN) and nascent attempts at providing multi-tenancy for data-warehousing (Hive-LLAP/Impala), serving (HBase) etc.
- Software to be downloaded and used on shared clusters.
- In on-prem deployments, enterprises were able to use approaches like network perimeter security and physical access control as the key pillars of security. In many cases, customers found such simplistic security enforcement to be sufficient and prioritized simplicity of deployment over more robust security mechanisms.
- **Hadoop gen-1 Challenges:** The big, unfortunate, side-effect of the above was the complexity of upgrading the cluster:

- Large, shared clusters with shrink-wrapped software meant that the upgrade was a big-bang i.e., every tenant had to upgrade at the same time and the exposure was very broad.
 - The organizational effort to coordinate the upgrade across hundreds of tenants and thousands of applications was extremely high.
 - The co-located architecture didn't distinguish between upgrades of the storage layer (risk) and the compute layer (coordination).
- **Hadoop gen-1 in the Public Cloud:** Amazon EMR, Microsoft HDInsight etc.



- Leveraged cloud object stores were decoupled from compute. The community-built connectors to S3, WASB etc. using the Hadoop Compatible Filesystem (HCFS) APIs.
 - Used VMs to spin up compute-only Hadoop clusters which were largely ephemeral; however, the relatively high overhead of spinning up VMs themselves (nearly 10 mins) led to the need to keep clusters up-and-running — an expensive proposition.
 - With the ephemerality of compute clusters, there wasn't a good way to manage long-lived metadata, security policies etc. which also led to expensive long-running clusters.
- **Hadoop Powered Data Cloud (gen-3):** By the end of the first decade, Big Data solution provider needed a fundamental rethink — not just for the public cloud, but also for on-premises. It's also helpful to cast an eye on the various technological forces driving Hadoop's evolution over the next decade:
- Cloud experiences fundamentally changed expectations for easy to use, self-service, on-demand, elastic consumption of software and apps as services.
 - Separation of compute and storage is now practical in both public and private clouds, significantly increasing workload performance.
 - Containers and Kubernetes are ubiquitous as a standard operating environment that is more flexible and agile.
 - The integration of streaming, analytics and machine learning — the data lifecycle — is recognized as a prerequisite for nearly every data-driven business use case.

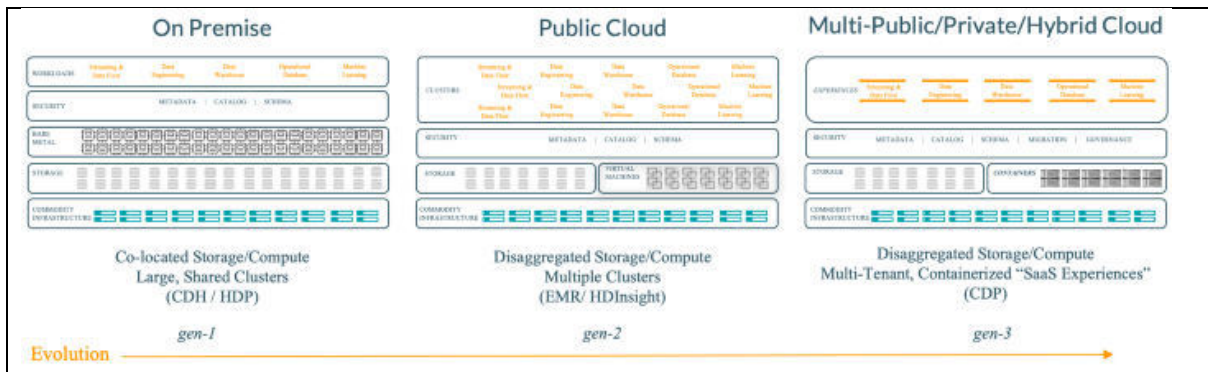
Multi-Public/Private/Hybrid Cloud



Disaggregated Storage/Compute Multi-Tenant, Containerized “SaaS Experiences” (CDP)

gen-3

- **Disaggregated storage**, metadata/security/governance and compute layers. In particular, the ever-widening use of RAM and SSDs for relatively cheap caching means that we can provide interactive performance even when storage is disaggregated i.e., take compute to storage.
- Software in a services form-factor.
- A new approach to multi-tenancy, driven by the emergence of containers, where every tenant can be provisioned as an isolated private service (e.g. warehouse) and leverage Kubernetes for software management.
- Extremely strong focus on security across on-premise and public cloud. No more corporate firewalls for hybrid deployments.
- Increased awareness of data privacy and emergence of stringent regulations (GDPR/EU, CCPA/California, PDPB/India) led to the requirement of richer governance including lineage, provenance and more across data migration (cloud, on-prem etc.) and the entire lifecycle of data including streaming, data engineering, reporting, predicting and serving.
- **Benefits of Hadoop Powered Data Cloud (gen-3):**
 - **Ease of management** due to de-coupling of storage/metadata and compute stacks. Even in situations, such as on-premises, where it might make sense to co-locate data and compute for efficiency purposes they are managed independently.
 - **Ease-of-use** due to the strong focus on ‘it’s a service’ — which leads to persona-focused UX for warehousing, machine learning, data engineering, streaming, and data-flow.
 - **Faster provisioning** with containers and Kubernetes dramatically speeding up provisioning and simplifying management (i.e. eliminates management) of services such as data warehouse, ML, streaming etc.
 - **Strong security and governance** with SDX across the entire data & analytical life-cycle enabling data-driven decision making.
- **Hadoop Power across generations:**



- **Hadoop is dead:** As per Arun C Murthy (If you are in BigData world, you may not need intro of Arun), Hadoop is dead — done, and dusted. Hadoop as a philosophy to drive an ever-evolving ecosystem of open-source technologies and open data standards that empower people to turn data into insights is alive and enduring. As long as there is data, there will be “Hadoop”.
- **Cloudera Runtime:** Cloudera Runtime is the core open-source software distribution within the CDP and foundation of CDP. Cloudera Runtime includes approximately 50 open-source projects that comprises the core distribution of data management tools with the CDP. Cloudera Data Platform is an aggregate of Cloudera Runtime, Cloudera Manager and other configuration. This is available as part of CDP Public Cloud and CDP Private Cloud Base.

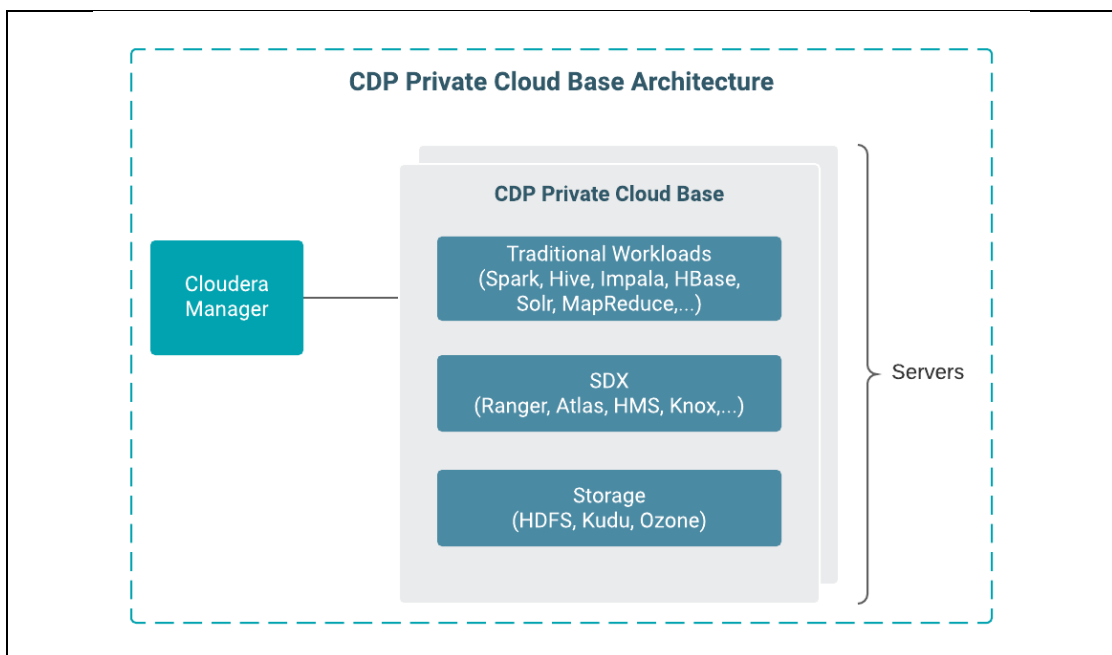


- **Cloudera Runtime Components:** Cloudera Runtime comes with different versions like 7.0 or 7.1, it means which versions are supported underline. There are majorly three categories of the components as below.
 - o **Apache Components:** These are the components which are available from Apache Open source like Apache Arrow, Apache Atlas, Apache HBase, Apache Kafka etc. There are approx. 25 Apache Components included in Cloudera Runtime.
 - o **Other Components:** This includes 10 different components like
 - Cruise Control
 - Data Analytics Studio
 - GCS Connector
 - HBase Indexer
 - Hue
 - Search

- **Schema Registry**
 - **Streams Messaging Manager**
 - **Stream Replication Manager**
- **Connectors and Encryption Components:** This includes 6 components as below.
 - **HBase Connectors**
 - **Hive Meta Store**
 - **Hive on Tez**
 - **Hive Warehouse Connector**
 - **Spark Atlas Connector**
 - **Spark Schema Registry**
- **Cloudera Runtime in Public and Private Cloud:** There are different versions of Cloudera Runtime for Public Cloud (CR-7.2.8) and Private Cloud (7.1.6) as of now.

Chapter-2: CDP Private Cloud

- **CDP Private Cloud:** As name suggests this setup is deployed in private data centers or you can see it's an on-premises version of Cloudera Data Platform. This is a combination of CDP Private Cloud Base and CDP Private Cloud Data Services. CDP Private Cloud Base Architecture is shown in below image.



- **CDP Private Cloud and Hybrid Services:** CDP Private Cloud Base supports a hybrid solutions where compute tasks are segregated from data storage and where data can be accessed from remote clusters, and workloads are created using CDP Private Cloud Data Services.
- **CDP Private Cloud Base:** This was previously known as Cloudera Data Platform (CDP) Data Center. And this is the on-premises version of Cloudera Data Platform. This provides combined capability of Cloudera (CDH: Cloudera Distribution of Apache Hadoop) and Hortonwork's (HDP: Hortonwork's Data Platform) technologies as well many additional features. CDP Private Cloud Base forms a comprehensive data platform that encompasses the entire data life cycle. CDP Private Cloud Base is a foundation for the CDP Private Cloud.

- **CDP Private Cloud Base and Workloads:** CDP Private Cloud Base is comprised of a variety of components such as HDFS, Hive-3, HBase, Impala and many other components. We can select any combination of these services to create clusters that address your business requirements and workloads. Cloudera also provides various pre-configured packages of services for common workloads.
- **CDP Data Center:** This is an old name of CDP Private Cloud Base. And same as on-premise version of Cloudera Data Platform.
- **CDP Private Cloud: (CDP Private Cloud Base + CDP Private Data Cloud Data Services)** form the complete CDP Private Cloud. Hence, any enterprise having either CDH or HDP installed on-premises are requested to upgrade to CDP Private Cloud Base and if needed they can add the Data Services Cluster.
- **Common pre-configured workloads in CDP Private Cloud Base:**
 - **Data Engineering:** This is a use case of processing, developing and serve predictive models. These include services like HDFS, YARN, YARN Queue Manager, Ranger, Atlas, Hive on Tez, Spark, OOZie, Hue, and Data Analytics studio.
 - **Data Mart:** Browse, query and explore your data in interactive way. And this includes services HDFS, Ranger, Atlas, Hive and Hue.
 - **Operational Database:** Real-time insights for modern data-driven business. And include services as HDFS, Ranger, Atlas and HBase.
 - **Custom Services:** In this case, you can choose your own services. When you are installing a CDP Private Cloud Base Cluster, we usually install a single parcel called Cloudera Runtime that contains all of the components.
- **CDP Private Cloud Base and Tools:** In addition to the Cloudera Runtime components, CDP Private Cloud Base includes powerful tools to help manage, govern, and secure your cluster. These includes following tools
 - **Cloudera Manager**
 - **Apache Atlas**
 - **Apache Ranger**
- **CDP Private Cloud Experiences:** CDP Private Cloud Experiences include; CDP Cloud Experience is renamed to CDP Private Cloud Data Services.
 - Management Console
 - Cloudera Data Warehouse (CDW)
 - Cloudera Machine Learning
 - Cloudera Data Engineering

Chapter-3: CDP Private Cloud Data Services

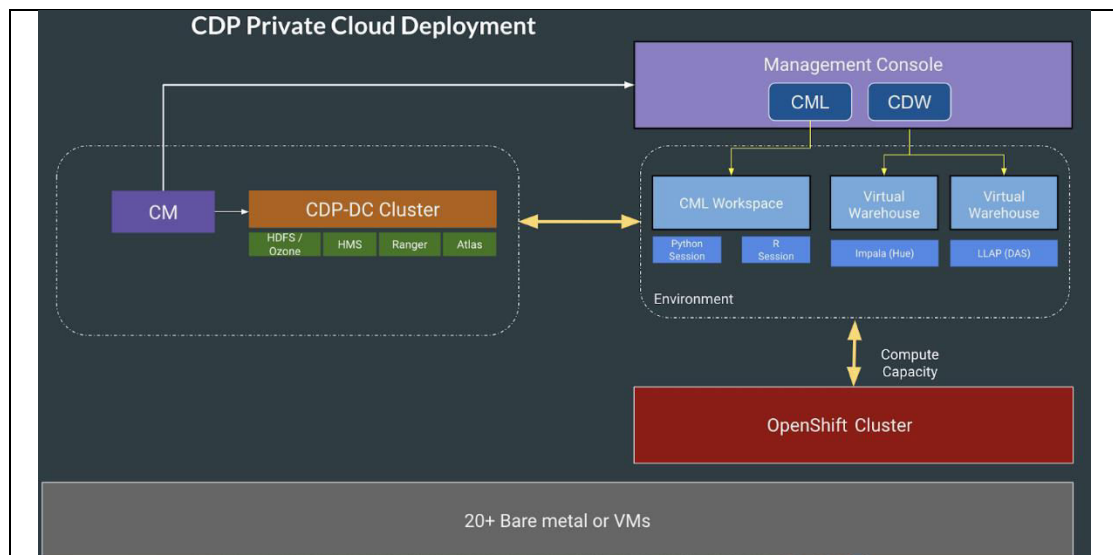
- **CDP Private Cloud Data Services:** This is a separate computing cluster running on a container platform that can be deployed with CDP Private Cloud Base. CDP Private Cloud Data Services always require Cloudera Manager and CDP Private Cloud Base. CDP Cloud Experience is renamed to CDP Private Cloud Data Services. And include following services and these are growing, tomorrow you may see more addition on this.
 - Management Console
 - Cloudera Data Warehouse (CDW)

- Cloudera Machine Learning (CML)
- Cloudera Data Engineering (CDE)

CDP Private Cloud Data Services is a framework which lets you deploy and use the growing collection of Cloudera Data Services mentioned above. You can run CDP Private Cloud Data services on one of the following platforms

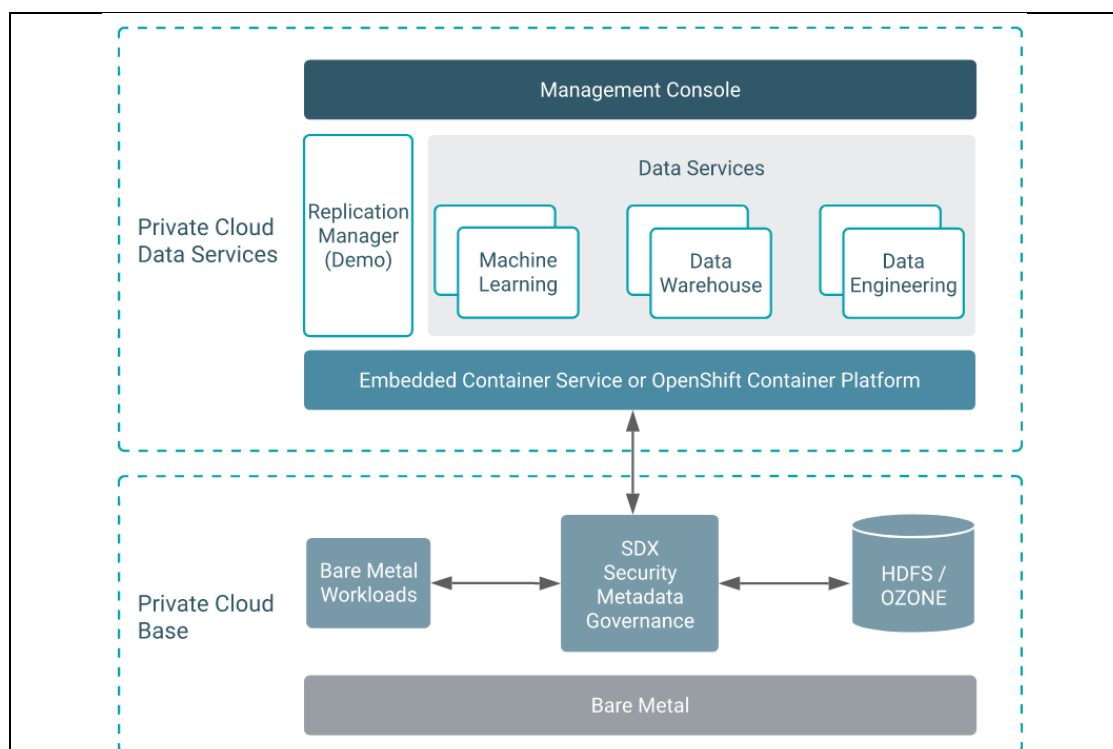
- OpenShift
- Embedded Container Service
- **Experiences Compute Services:** Experiences Compute Services (ECS) is renamed to Embedded Container Service (ECS).
- **Public Cloud, Containers & Cloudera Data Engineering:** Around 2020 Cloudera launched a public cloud edition of CDE that made it possible to deploy Apache Spark Framework for analyzing data on top of Kubernetes clusters.
- **CDP-DE Private Cloud and Containers:** Cloudera Data Engineering in CDP Private Cloud is designed to be deployed in On-premises IT environments. And also, CDE can be deployed on both the Redhat OpenShift platform based on Kubernetes and the Experience Compute Service (ECS) cloud platform provided by Cloudera. IT teams can also burst workloads from an on-premises IT environment to a public cloud whenever additional capacity is needed.
- **Kubernetes and CDE:** Workloads can scale independently of compute and storage and, thanks to Kubernetes, can be managed via the same control plane. Workloads are deployed as containers in virtual clusters that connect to a storage cluster dubbed CDP Base. Collectively, those capabilities make it easier to add additional workloads into a multitenant environment without adversely affecting the performance of the applications already running. As Kubernetes becomes more widely employed, it's becoming clearer that the management of compute, data and applications will converge around a common control plane. It may still require a team of IT professionals to manage those functions but the days when IT teams needed to deploy a separate control plane for each function are coming to an end. That control plane also provides the mechanism to unify DevOps and data operations as the number of stateful applications deployed on Kubernetes clusters continues to steadily increase.
- **Kubernetes, stateless to stateful:**
 - Originally, most of the workloads deployed on Kubernetes clusters were stateless, in that they relied on external system to store data. Increasingly, now organizations are now deploying databases directly on Kubernetes clusters to more efficiently access data stored on the cluster.
 - Kubernetes provides access to storage using persistent volumes (PV). This makes it possible to access data far beyond the lifespan of any given pod. Kubernetes volumes allow users to mount storage units to expand how much data they can share between nodes.
 - Regular volumes will still be deleted if and when the pod hosting that particular volume is shut down. The permanent volume, however, is hosted on its own pod to ensure data remains accessible.
 - Upon creation, the PV is bound to the pod that requested the PVC. IT teams can then manage storage in Kubernetes via a PersistentVolumeClaim (PVC) function to request storage; a PersistentVolume (PV) to manage storage life cycle and a StorageClass function that defines different classes of storage services.
 - The primary reasons IT teams are deploying stateful applications in Kubernetes clusters is to ensure consistency, a desire to standardize on Kubernetes, simplify management and enable developers to self-manage the IT environment.

- Stateful vs. stateless apps are conceptual and technological. More IT settings are adopting Kubernetes clusters in greenfield edge environments without local traditional storage solutions. In other circumstances, IT teams desire to consolidate compute and storage administration to avoid hiring an external storage administrator. Labor costs still dominate IT.
- **Shifting Workload across Public and Private Cloud CDE Solution:** IT teams can now build applications on either platform and then run the, across a hybrid Cloud Computing environment by first shifting workloads and then using tool such as Cloudera Replication Manager to migrate data when required. CDE in CDP Private Cloud makes it easier to isolate noisy data-intensive workloads to ensure service level agreements (SLAs) are met.
- **CDP Private Cloud Management Console:** In CDP Private Cloud Management Console is a service for administering CDP and with this you can manage environments, data lakes, environment resources, and users across all CDP services.
- **Hybrid Data Platform:** CDP Private Cloud is a Hybrid platform, because CDP Private Cloud Base runs on your own Data Center and CDP Private Data Services can run on the Open Shift as a container.
- **OpenShift and Cloudera CDP:**



- With Red Hat OpenShift, CDP Private Cloud will deliver self-service analytics and enterprise-grade performance with the granular security and governance policies Red Hat OpenShift and CDP Private Cloud will help to create an essential hybrid, multi-cloud data architecture.
- Red Hat OpenShift as the preferred container solution for Cloudera Data Platform (CDP) Private Cloud.
- Red Hat OpenShift's position as the market-leading Kubernetes container platform, combined with its 100 per cent open-source nature, make it ideal for CDP Private Cloud.
- CDP Private Cloud, supported by Red Hat OpenShift, creates an enterprise data cloud with a powerful hybrid architecture that separates compute and storage for greater agility, ease of use, and more efficient use of private and public cloud infrastructure.
- **Hybrid, multi-cloud data architecture:**

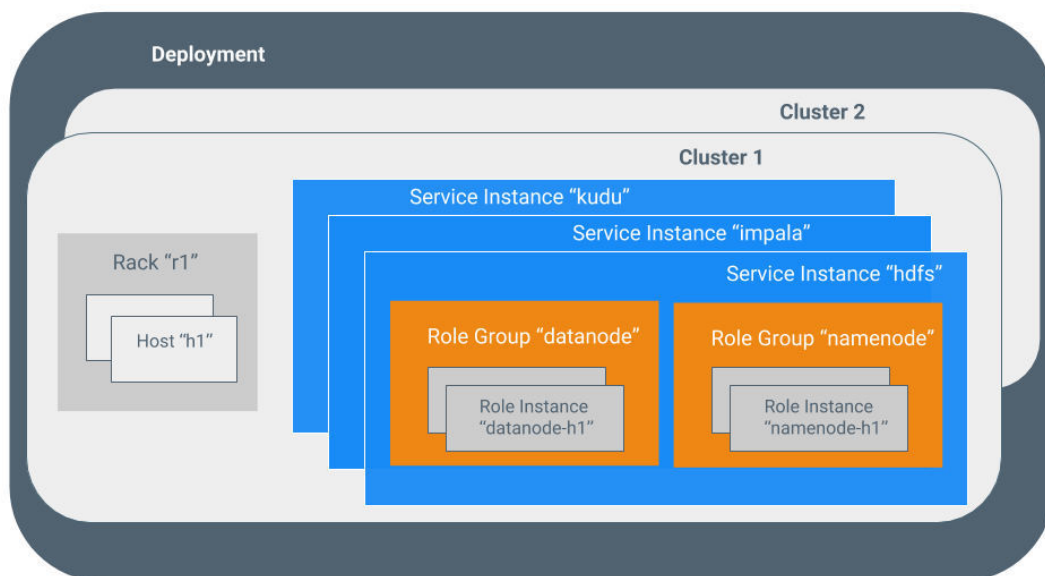
- Red Hat OpenShift and CDP Private Cloud will help to create an essential hybrid, multi-cloud data architecture.
- It will enable teams to rapidly onboard mission-critical applications and run them anywhere, without disrupting existing ones.
- **CDP Stand Alone:** You can deploy CDP Private Cloud Base as stand-alone to run your data analytics. CDP Private Cloud Base is a complete data platform and stand-alone instance of CDP for the on-premises data center that can be deployed on a choice of optimized infrastructure (i.e Dell or Intel or IBM). CDP Private Cloud Base can also be deployed with the CDP Private Cloud Data Services cluster to form the complete CDP Private Cloud.
- **CDP Multi Cluster:** CDP Private Cloud also supports a hybrid or multicluster solution, where compute tasks can be separated from data storage, and where data can be accessed from remote clusters. Where CDP Private Cloud Data Services, a separate computing cluster running on a container platform that can be deployed with CDP Private Cloud Base. This approach provides a foundation for containerized applications by managing storage, table schema, authentication, authorization, and governance in CDP Private Cloud Base. It consists of various components such as Apache HDFS, Apache Hive 3, Apache HBase, and Apache Impala, along with many other components for specialized workloads. You can select any combination of these services to create clusters that address your business requirements and workloads.
- **Rapid Provisioning:** CDP Private Cloud Data Services users can rapidly provision and deploy services such as Cloudera Data Warehouse, Cloudera Machine Learning, and Cloudera Data Engineering using Cloudera Management Console and also this can be scaled up and scale down.
- A Private Cloud Base cluster and container-based clusters are needed to implement CDP Private Cloud Data Services. Use a dedicated RedHat OpenShift cluster or an Embedded Container Service (ECS) for containers.
- Configuring Management Console, registering an environment, and establishing workloads are the steps in Private Cloud deployment.



-

Chapter-4: Cloudera Manager

- **Cloudera Manager:** Cloudera Manager is a component of CDP Private Cloud Data Services. Cloudera Manager is an end-to-end application for managing clusters. With Cloudera Manager, you can easily deploy and centrally operate the complete Cloudera Runtime stack and other managed services. The application automates the installation and upgrade processes and gives you a cluster-wide, real-time view of hosts and running services. The Cloudera Manager Admin Console provides a single, central console where you can make configuration changes across your cluster and incorporates a full range of reporting and diagnostic tools to help you optimize performance and utilization. Cloudera Manager also manages security and encryption functionality. This overview introduces the basic concepts, structure, and functions of Cloudera Manager. A single instance of Cloudera Manager can manage multiple clusters, including older versions of Cloudera Runtime and CDH.
- **Cloudera Manager and CDP Private Cloud Base:**
 - o CDP Private Cloud Base requires the Cloudera Manager to manage one or more clusters and their configurations and to monitor cluster performance.
 - o We also use Cloudera Manager to manage, installations, upgrades, maintenance workflows, encryption, access controls and data replication.
 - o We can use Cloudera Manager to manage Cloudera Enterprise CDH (Previous version of Solutions) clusters.
 - o We can use Cloudera Manager to create a Virtual Private Cluster that allows you to separate compute resources from data storage and to share data storage among compute resources.
- **Cloudera Manager Deployment:** A configuration of Cloudera Manager and all the clusters it manages.



- **Cloudera Manager-Dynamic Resource Pool:** In Cloudera Manager, a named configuration of resources and a policy for scheduling the resources among YARN applications or Impala queries running in the pool.
- **Cloudera Manager-Cluster:** A set of computers or racks of computers that contains an HDFS filesystem and runs MapReduce and other processes on that data. In Cloudera Manager, a logical entity that contains a set of hosts, a single version of Cloudera Runtime installed on the hosts, and the service and role instances running on the hosts. A host can belong to only one cluster. Cloudera Manager can manage multiple clusters; however, each cluster can only be associated with a single Cloudera Manager Server.
- **Cloudera Manager-Host:** In Cloudera Manager, a physical or virtual machine that runs role instances. A host can belong to only one cluster.
- **Cloudera Manager-Rack:** In Cloudera Manager, a physical entity that contains a set of physical hosts typically served by the same switch.
- **Cloudera Manager-Service:** Service is also referred as Service type and this is a managed functionality in Cloudera Manager, which may be distributed or not, running in a cluster and few example of this Hive, HBase, HDFS, YARN and Spark. Also, A Linux command that runs a System V init script in /etc/init.d/ in as predictable an environment as possible, removing most environment variables and setting the current working directory to /.
- **Cloudera Manager-Service Instance:** You can run one or more service instance for example for HDFS service, you can have service instance like HDFS-1, HDFS-2. Hence, there are two service instances are running for service type HDFS.
- **Cloudera Manager-Role:** This represent a category within a service type for example HDFS is a service or service type and it has various roles (category) in it i.e. NameNode, Secondary NameNode, DataNode, and Balancer. This is sometime known as Role type as well.
- **Cloudera Manager-Role Instance:** As name suggests it is an instance of the Cloudera Manager Role. In Cloudera Manager, an instance of a role running on a host. It typically maps to a Unix process. For example: "NameNode-h1" and "DataNode-h1".
- **Cloudera Manager-Role Group:** In Cloudera Manager, a set of configuration properties for a set of role instances.
- **Cloudera Manager-Host Template:** A set of role groups in Cloudera Manager. When a template is applied to a host, a role instance from each role group is created and assigned to that host.
- **Cloudera Manager: Gateway:** A type of role that typically provides client access to specific cluster services. For example, HDFS, Hive, Kafka, MapReduce, Solr, and Spark each have gateway roles to provide access for their clients to their respective services. Gateway roles do not always have "gateway" in their names, nor are they exclusively for client access. For example, Hue Kerberos Ticket Renewer is a gateway role that proxies tickets from Kerberos.

The node supporting one or more gateway roles is sometimes referred to as the gateway node or edge node, with the notion of "edge" common in network or cloud environments. In terms of the Cloudera cluster, the gateway nodes in the cluster receive the appropriate client configuration files when Deploy Client Configuration is selected from the Actions menu in Cloudera Manager Admin Console.

- **Cloudera Manager-Parcel:** This a distribution format in binary which contains the compiled code and meta-information such as a package description, version and dependencies.
- **Cloudera Manager-Static Service Pool:** In Cloudera Manager, a static partitioning of total cluster resources-CPU, memory, I/O weight-across set of services.
- **CDH and HDP:** Both are the previous version of Hadoop Based Solutions.

- **HDP:** Hortonworks Data Platform, It was offered by Hortonworks.
- **CDH:** Cloudera Data Hub was an offering from Cloudera for Hadoop based solution.
- **Previous and new Approaches for Data Processing:** Technology infrastructure formerly demanded the co-location of compute and storage to avoid costly network transfers. Now the needs of high-performance analytics drive a move toward disaggregated compute and storage, where each can be sized and scaled independently.
- **Previous and new user experience:** From a user experience viewpoint, it used to be acceptable to deploy and run in timeframes of weeks, months, or even quarters. Now the expectation is to be able to spin up services in minutes, give users their own clusters, and get insights quickly.
- **Privacy, security and governance in previous and new approaches:** From the privacy, security, and governance perspectives, the primary concerns were formerly about network perimeter and physical access controls. Now, with the entire data life cycle being managed, operators need fine-grained authentication and authorization at the workload and data layers.
- **Why Platform:**
 - The plethora of "apps" available for mobile devices. Applications are ready to provide value almost instantly after installation. Think of something like a mapping application with navigation capability. You install the app, turn on location services, enter an address and you are on the way in less than five minutes.
 - Conversely, platforms are tools for application developers. Platforms do almost nothing for end users after installation. Application developers must first configure and build applications using the platform before end users begin to recognize value.
 - Developers have been using platforms for decades. There are some classes of applications that require core services that are complex to develop but universally useful. In those cases, it makes sense for a group of experienced system developers to build a platform for use by the larger application developer community.
 - Many developers lack the skills to do it on their own. Some of the first and most successful examples are relational database management systems (RDBMS). These systems include IBM DB2, Oracle, and Microsoft SQL Server. The RDBMS category has expanded to include many more platforms over the last several decades. Millions of application developers and billions of end users have benefitted from software applications that are developed using these RDBMS platforms.
 - The most successful data platforms are both robust and flexible. Millions of application developers, who otherwise could not build the scalable foundations that are required to support enterprise class data management, can use them. "Reinventing the wheel" has always been costly and rarely produces superior modes of transportation. Despite that history, many organizations spend many months or years contemplating and prototyping proprietary data platforms.
 - Enterprise developers may be encouraged that most of the hyperscale Internet companies have developed proprietary data platforms to meet their specific industry and scale challenges. Some of these companies include Airbnb, Facebook, LinkedIn, Lyft, Netflix, Twitter, and Uber.
 - These organizations differ from most traditional enterprise organizations in several key ways. They were born "cloud native", meaning the platforms that they have developed constitute the business. They can recruit and retain top talent with the backgrounds that are required to build platforms. Also, they are constantly adding

the already large initial development investments because their data platform is critical to their main value proposition.

- **Data Platform:** Either build your own proprietary data platform or adopt full featured commercial and open-source data platform. There are various Data Platform terms available based on the solution which includes
 - o **Cloudera Data Platform**
 - o **Big Data Platforms**
 - o **Data Management Platforms**
 - o **Data Analytics Platforms**
- **Data Platform and Applications:** Most of the common use case is data pipeline for Data Platform which includes various steps like below
 - o **Collect the Data**
 - o **Apply Data Engineering on the Data**
 - o **Create Reports from the data**
 - o **Running Operations on the Data**
 - o **Apply Machine Learning and AI on the Data**

Below image shows these steps



Data platforms that meet all or most of the needs of your data pipelines simplify the process of getting from raw source data to insights. Whenever data in the pipeline must move between platforms there is a real possibility of introducing complexity both in the development phase and in sustaining operations.

- **Data Management:**
 - o Some file systems are better suited to handling lots of small files while others are better at fewer large files.
 - o For audio and other "stream based" data, data engineers have a choice of buffer size and file creation characteristics that must be matched to the capabilities of the platform and may also impact the complexity of using the data for analysis.
 - o Most digital data has some type of structure or common properties when it is committed to a storage medium. Some examples include:
 - Files have a size property and a filetype (application, text, binary).
 - Text files have an encoding scheme.
 - Images have dimensional size and color depth encoding.
 - Audio has bit rate and frequency range.
 - o These characteristics impact the requirements for a data platform.
 - o If you have more knowledge about the final stages of how your analysis pipelines look, you can build more intelligence into the early stages of data management.

- Although storing high-fidelity data when it is not required for analysis may seem wasteful, think of it as an insurance policy to protect against changing analysis requirements.
- Another aspect of data management, that surprises IT professionals, is the storage that is required to manage multiple copies of data being used for analysis.
- Even the most seasoned data science professionals consume many copies of data that by some appearances are identical. There are several important reasons why this situation is necessary:
 - Both report and model development must be isolated from uncontrolled change. This initial copy is typically a direct copy of source with little or no transformation. This measure ensures that the developers can always return to a ground truth version of the data. That data can be used to compare alternate transformation schemes with repeatability.
 - Managing alternate transformations. One common pattern is grouping and counting events by various factors such as time, geography, market segment, and so on, as well as transformations to clean and regularize the data.
 - Efficiency. Complex data transformation pipelines should get developed in stages. It may be too inefficient to go all the way back to source data for testing an incremental set of tasks late in the pipeline. Data scientists may prefer to stage intermediate steps to reduce the complexity and time investment to run the pipeline from the absolute beginning.
- Another requirement that derives in part from the data copy management challenge is tracking metadata that are associated with transformation logic and history. Creating many copies of the same data may seem reasonable in the heat of shipping a project, but it will be difficult to ascertain why six months later.
- There is a growing interest in platforms that include "feature stores". The concept is to both better track logic and metadata and to promote a more disaggregated approach to data management
- If the only difference between two datasets is how the customer dimension is managed, you should keep two copies of that feature, rather than two copies of the entire dataset.
- Reusing transformation logic to manage frequently used dimensions - like customers and products - independently from all the other features, and all the other analysis datasets in which they are used, could greatly simplify data management.
- **Example Use Cases:**
 - The potential list of use cases that a full-featured data platform can address is nearly limitless.
 - The following list gives some sense of the common use cases
 - Customer 360 Analytics
 - Retail Inventory and Sales Analysis
 - Manufacturing Operational Analysis
 - eCommerce Fraud Prevention
 - Network Security intelligence
 - Data Warehouse Consolidation
 - Discount Pricing Optimization
 - Financial Services
 - Insurance Industry predictive analytics

- Recommendation Engines
 - Social Media analysis and engagement
 - A good business practice is to maintain an active list of potential use cases where the availability of a data platform can enhance development.
 - Assess the list so that you do not tackle too many use cases that are high-priority and high-investment too early.
- **Financial Services use Cases:**
 - Financial services encompass a wide range of business models, for example
 - **Consumer and Commercial Banking**
 - **Individual Wealth Management**
 - **Primary or secondary capital markets**
 - The importance of relationship management is shared across all these businesses and therefore has been a key focus area for analysis.
 - Virtually all mid-sized and larger financial services organizations have one or more data platforms.
 - The intense pressure to compete with other players makes finding, securing, keeping and nurturing relationships with customers a priority that drives profit.
 - There is also a requirement to manage investment risk and assure compliance with all regulatory requirements, which often involve multiple, overlapping jurisdictions.
 - While personal relationships still matter, data-driven modelling and reporting across multiple channels including mobile, online phone or branch agent are a must-have for these organizations.
 - Organizations that build trust by arming the organization with data-driven information increase the confidence of their customers, along with wallet share and lifetime value.
 - To achieve that on a global scale, you must leverage big data and predictive analytics using a proven and modern hybrid data platform.
- **Manufacturing use case:**
 - Industry 4.0 is an emerging term that means smart manufacturing.
 - Advanced technologies are combined with traditional manufacturing and industrial practices to improve operational efficiency across the board.
 - The innovations and documented successes of Industry 4.0 initiatives are encouraging more manufacturing companies to adopt Industrial IoT (IIoT) concepts and technology.
 - Such adoptions transform product development, supply chains, and manufacturing operations.
 - Many recent case studies show that connecting analysis of smart products, design engineering, factory floor operations, and customer experience enable faster time-to-market, improved product quality, and scaling production output while reducing waste and operating costs.
 - Connected products are a key initiative of Industry 4.0.
 - The connectivity these products provide drives customer satisfaction and revenue while reshaping the relationship between people and products.
 - Achieving these benefits requires the abilities to ingest, process, and analyze sometimes massive volumes of IoT data.
 - This data processing scale enables manufacturers the access to near real-time customer feedback to identify product quality issues.

- Another growing area of Industry 4.0 is intelligent supply chain management. Disruptions and delays in a critical supply chain will ripple through an organization from sales to operations.
- Many manufacturers are using near real-time data, analytics, and machine learning to ensure that supply chains are functioning well while risk is managed end-to-end.
- Combined with a modern data platform that supports advanced analytics, including machine learning capabilities, required investments to take advantage of these latest innovations in manufacturing include:
 - Special Purpose Sensors
 - GPS
 - RFID
 - Production Stream Data
- **CDP Data Flow:** IT teams can also programmatically deploy complex pipelines with job dependencies using Apache Airflow, open-source software based on directed acyclic graphs (DAGs), that make it possible to visualize and monitor pipelines running in production environments.
- **DevOps and CDP:** Available Cloudera application programming interfaces (APIs) and command-line interfaces (CLIs) make it simpler to integrate its platform within DevOps workflows.

Chapter-5: Apache Atlas

- **Apache Atlas and CDP Private Cloud Base:**
 - CDP Private Cloud Base includes the Apache Atlas.
 - Apache Atlas is used to provide governance for your data.
 - Atlas serves as a common metadata store that is designed to exchange metadata both inside and outside of the Hadoop stack.
- **Apache Atlas and Apache Ranger:**
 - Close integration of Atlas with Apache Ranger enables us to define, administer, and manage security and compliance policies consistently across all components of the CDP stack.
- **Apache Atlas and Cloudera Navigator:**
 - In previous version of Cloudera Enterprise, Atlas is replacing Cloudera Navigator Metadata server and provides following capabilities
 - Creating Flexible Metadata models.
 - Searching Entity using model attributes, classification (i.e. tags) and free text.
 - Lineage across entities based on processes applied to the entities.

Chapter-6: Apache Ranger

- **Cloudera Private Base and Apache Ranger**
 - Following are the features provided by Apache Ranger (AAA) in CDP Private Base Cluster.
 - Auditing
 - Authentication
 - Authorization
 - Ranger is a centralized framework for collecting and accessing audit history and reporting data even includes filtering on various parameters.
 - Ranger helps in enhancing audit information obtained from CDP components and provides insights through this centralized reporting capability.
 - Ranger controls access control through a user interface to guarantee policy consistency among CDP Private Cloud Base components.
 - Security administrators may set database, table, column, and file security rules and manage LDAP-based group and user permissions.
 - Rules based on dynamic conditions such as time or geolocation can also be added to an existing policy rule.
 - The Ranger authorization model is pluggable and can be easily extended to any data source using a service-based definition.
 - Ranger replaces Sentry and Navigator Audit Server from previous version of Cloudera Enterprise.
- **Apache Ranger and fine-grained access controls:** Apache Ranger provides following capabilities
 - Dynamic Row filtering
 - Dynamic Column Masking
 - Attribute Based Access control
 - SparkSQL fine-grained access control.
- **Apache Ranger and Rich Policy features:** This provides
 - Allow/Deny constructs
 - Custom policy conditions
 - Context enrichers
 - Time bound policies
 - Atlas integration
 - Extensive Access Auditing with rich event metadata

Chapter-1: Apache HDFS

Overview

Hadoop Distributed File System, often known as HDFS, is a file system that is based on Java and offers scalable data storage.

- **NameNode:** The NameNode of an HDFS cluster is responsible for managing the namespace of the cluster.
- **DataNode:** while the DataNodes are used to store data.

HDFS was designed to span huge clusters of commodity systems. The Hadoop Distributed File System (HDFS) serves as the platform's data management layer. YARN is responsible for the administration of the resources, whereas HDFS is in charge of storage.

HDFS is a distributed storage system that is scalable, fault-tolerant, and works closely with a broad range of applications that access data concurrently. By spreading storage and computing among a large number of servers, the total storage resource may expand in a linear fashion in response to increased demand.

Because HDFS can be scaled up or down depending on the requirements, it is truly quite difficult to find any substitute for HDFS that is suitable for storing Big Data. Hadoop is being used by a significant number of the largest corporations in the world. Hadoop is used by several companies, including Amazon, Facebook, Microsoft, Google, Yahoo, IBM, and General Electrics, to store and analyse enormous volumes of data.

HDFS cluster components and their respective roles

The primary components of an HDFS cluster are referred to as the NameNode and the DataNodes respectively.

The NameNode: is responsible for managing the metadata of the cluster, which includes the file and directory hierarchies, rights, changes, and quotas for disc space. The contents of the file are separated into several data blocks, with each block being copied at a number of different DataNodes.

The NameNode keeps monitoring on the total number of blocks that have been replicated. In addition to this, the NameNode stores the full namespace image in RAM and is responsible for maintaining the namespace tree as well as the mapping of blocks to DataNodes.

In order to prevent single point of failure, High-Availability (HA) clusters include a backup NameNode for the current one. And clusters synchronise the active and standby NameNodes by using JournalNodes in the process.

Advantages of using HDFS

The following are some of the advantages provided by HDFS, which are directly responsible for the effective storage and high availability of data inside the cluster:

- **Rack awareness:** refers to the physical location of a node while assigning storage and scheduling jobs.
- **Hadoop reduces the amount of data that has to be moved by moving computational** activities directly to HDFS. Whatever computation Hadoop initiates sends all the computation near to data i.e. on DataNode rather than getting data to the compute node, this is the biggest advantage and reduce the network traffic. This results in a very considerable reduction in the overall amount of network I/O and offers extremely high aggregate bandwidth.
- **Utilities:** Conduct a real-time analysis of the state of the file system's health and rebalance the data throughout the various nodes.
- **Standby NameNode** is a NameNode that offers high availability and provides redundancy (HA).

NameNodes

NameNodes are responsible for maintaining the HDFS namespace tree as well as a mapping of file blocks to the DataNodes that are used to store the data.

Only one main NameNode is necessary for a basic HDFS cluster. This primary NameNode is backed up by a backup NameNode that compresses the NameNode edits log file on a periodic basis. The NameNode edits log file is a list of HDFS metadata alterations. Because of this, the amount of disc space used by the log file on the NameNode is decreased, which in turn results in a shorter length of time required to restart the main NameNode. There are always two NameNodes present in a high-availability cluster: an active and a backup.

DataNodes

NameNode daemon is responsible for managing the data in a Hadoop cluster, and DataNodes are the nodes that hold the data. The data in a file is copied and stored on several DataNodes to ensure its integrity and to facilitate the execution of localised computations in close proximity to the data.

It is important for a cluster's DataNodes to have a consistent appearance. Problems may arise if they are not consistent with one another. For instance, jobs may fail to complete if DataNodes have a lower total amount of memory because they reach capacity more rapidly than DataNodes with a higher total amount of memory.

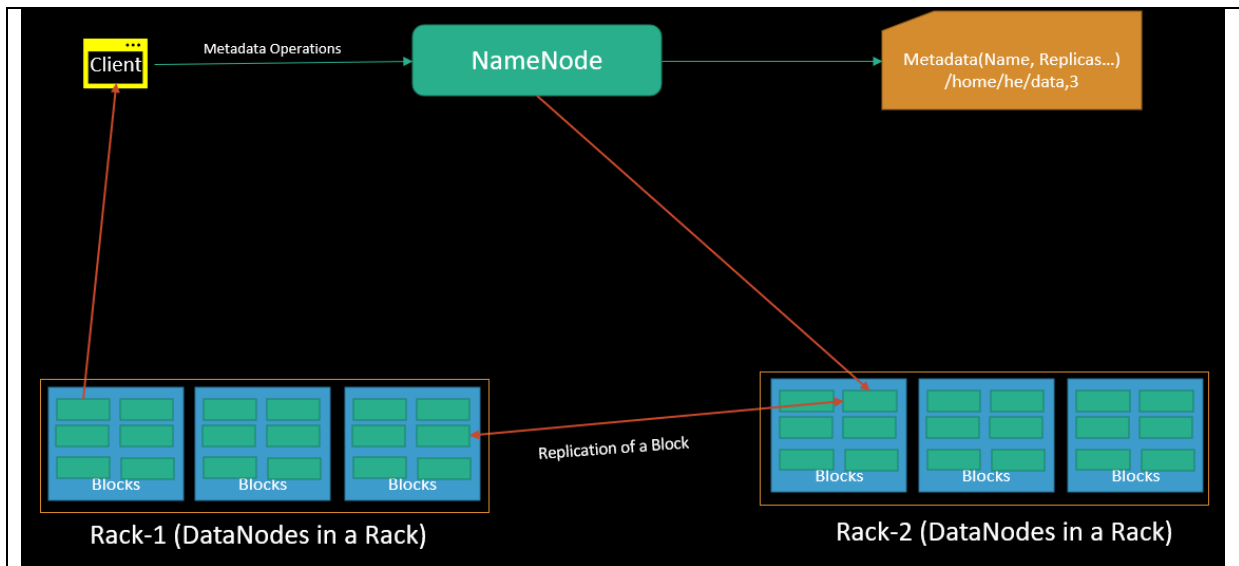
Important: HDFS was set up with a replication factor of three by default. That is, there are always three copies of the data kept on hand at all times. When you have at least three DataNodes, Cloudera recommends that you should not select a lower replication factor than the default value of three. Data loss might result from having a lower replication factor.

JournalNodes

JournalNodes are used to keep active and backup NameNodes in high-availability clusters synchronised with one another. The active NameNode is the one that makes changes to the HDFS namespace information and publishes such "edits" to each JournalNode. Whenever there is a failover, the standby NameNode will promote itself to the active state after first applying all of the updates that were made in the JournalNodes.

The Architecture of HDFS

There is only one NameNode that is responsible for storing metadata, whereas numerous DataNodes are in charge of doing the actual storage operations. In order to offer fault tolerance, the nodes in the cluster are organised into racks. Additionally, copies of data blocks are kept on separate racks within the cluster.



NameNode holds metadata, whereas DataNode contains real data. Due to the fact that NameNode is the central component of the cluster, all interactions between the client and the cluster must take place via NameNode.

Within the cluster, there are a number of DataNodes, each of which has its own local disc on which it stores HDFS data. DataNode will occasionally communicate through a "heartbeat" message to NameNode to let it know that it is still active. In addition to this, it copies the data to additional DataNodes in accordance with the replication factor.

Hadoop File System (HDFS) Features

Let's have a look at some of Hadoop Distributed File System's more interesting features right here in the HDFS lesson.

Storage That Is Distributed: HDFS stores data in a distributed fashion. It then saves each individual piece of data on a distinct DataNode inside the cluster after first dividing the data into smaller chunks. In this approach, the Hadoop Distributed File System provides MapReduce with a means by which it may process a fraction of enormous data sets that have been partitioned into blocks in a manner that is parallel across numerous nodes. Hadoop revolves on the MapReduce programming model, while HDFS is the component that makes all of these other features possible.

Blocks: HDFS breaks down massive files into manageable sections called as blocks. The smallest unit of data that may be stored in a filesystem is called a block. As a client or admin, you do not have any influence on the block's properties, such as its placement. NameNode is the one that makes all of these decisions.

HDFS default block size is 128 MB. We have the ability to adjust the size of the block to meet our specific requirements. This is in contrast to the filesystem used by the operating system, where the block size is 4 KB.

If the data size is less than the HDFS block size, then the block size will be equal to the data size.

If the size of the file, for instance, is 129 megabytes, then there will be 2 blocks produced for it. The default size of one block will be 128 megabytes, while the other block will only be one megabyte since using 128 megabytes would be a waste of space. Hadoop is smart enough to avoid wasting the

remaining 127 megabytes of storage space. Therefore, it is only allocating a 1 MB block for 1 MB worth of data.

A significant benefit of storing data in such a block size is that it reduces the amount of time spent seeking on the disc. Another benefit is that the mapper only processes one block at a time when it performs its operations. So, one mapper handles enormous data at a time.

The Act of Repeating: Hadoop HDFS produces two identical copies of each block it stores. This process is referred to as replication. The data for each block is copied and kept on separate DataNodes distributed across the cluster. It tries to store at least one duplicate on a distinct rack for each original.

What exactly is a Rack?

Racks are used to organise the DataNodes. A single switch connects all of the nodes in a rack. Hence, data may be accessed from another rack even if an individual switch or the whole rack becomes inoperable.

As previously discussed, the default replication factor is 3, but it is possible to alter this to the appropriate values according to the demand by modifying the configuration files (hdfs-site.xml).

High Availability: Data availability can be achieved by replicating data blocks and storing them on numerous nodes distributed across the cluster.

Data Reliability: Data is replicated in HDFS, as we have seen previously. Because of replication, blocks maintain a high availability even in the event that a node or piece of hardware fails. In the event that the DataNode fails, the block will still be available from any other DataNode that has a duplicate of the block. Additionally, even if the rack collapses, the block will still be accessible on the alternative rack. This is how HDFS ensures the integrity of its data storage while also providing fault tolerance and high availability.

Fault Tolerance: Hadoop and the other components of the ecosystem may benefit from the fault-tolerant storage layer that HDFS offers.

HDFS is designed to run on commodity hardware, which refers to systems that have average configurations and a high likelihood of crashing at any given moment. Because of this, the HDFS system duplicates data and stores it in many locations. This helps to ensure that the system as a whole is very resilient to errors.

Scalability: Scalability refers to the capacity to increase or decrease the size of the cluster. There are two methods in which we can grow Hadoop HDFS.

- **Vertical Expansion or scalability:** We are able to install more drives on the data nodes. In order to do this, we need to make changes to the configuration files and add entries that match to the newly inserted drives. Increasing disk size is a vertical scalability.
- **Horizontal Scalability:** is an additional method of scalability that consists of the capability of adding more nodes (Data Nodes) to the cluster dynamically and without incurring any downtime. This technique is referred to as horizontal scaling. We are able to add as many nodes as we like to the cluster at any one moment, in real time, without experiencing any kind of outage.

Access to application data with a high throughput: The Hadoop Distributed File System enables users to access application data with a high throughput. The quantity of work completed in a certain length of time is referred to as the throughput. It is often used as a method for measuring the system's overall performance, and it provides a description of the rate at which data is retrieved from the system.

When we wish to carry out a procedure or an operation using HDFS, the work is partitioned and distributed over a number of different computers. Therefore, each of the systems will independently and simultaneously carry out the duties that have been allocated to them. Because of this, the task will be finished in a relatively short amount of time. Therefore, HDFS provides a strong throughput because of its parallel data reading capabilities.

Read Operations on the HDFS

When a client wants to read any file from HDFS, the client has to communicate with NameNode since NameNode is the only location that keeps metadata about DataNodes. This means that whenever a client wants to read any file from HDFS, the client needs to interact with NameNode. NameNode is responsible for specifying the address of the slaves or the place where the data is kept.

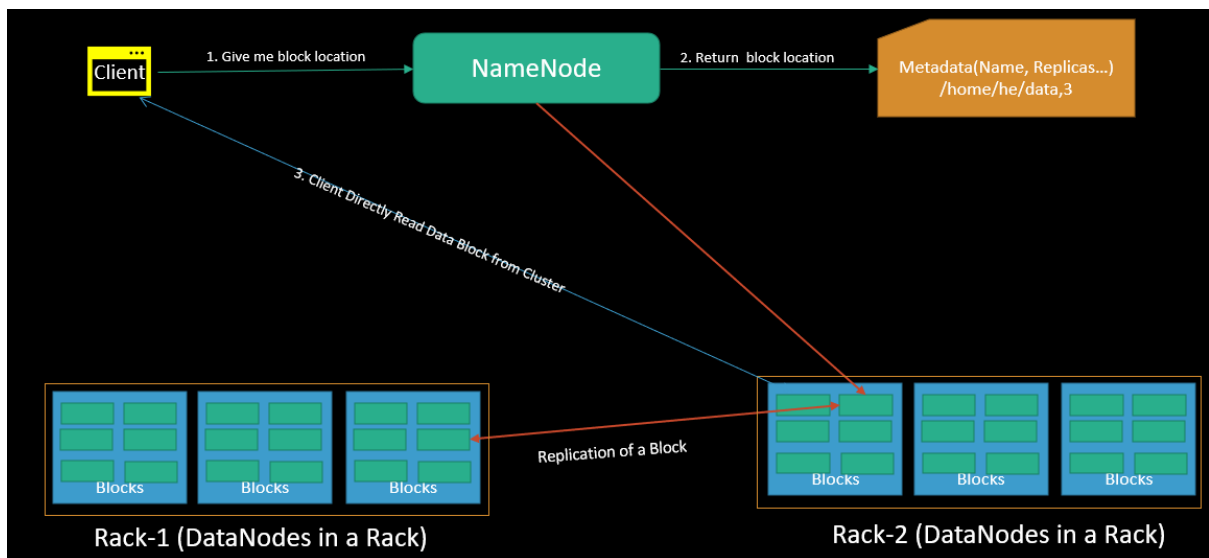
The client will engage in interaction with the DataNodes that have been configured and read the data from those locations. In order to ensure the client's authenticity and safety, the NameNode sends it a token, which the client then presents to the DataNode before beginning to read the file.

In the Hadoop HDFS read operation, the client must first interact with the NameNode in order to read data that is stored in HDFS if the client wishes to read data that is stored in HDFS. Therefore, the client engages in interaction with the API of the distributed file system and submits a request to NameNode to provide the block location. As a result, NameNode examines the client to determine whether or not they have enough credentials to access the data. If the client has the necessary credentials, then the NameNode will provide the address of the DataNode at which the data is kept.

Along with the address, NameNode also provides the client with a security token. In order to access the data, the client is required to provide the security token to DataNode for the purposes of authentication.

When a client travels to DataNode for the purpose of reading the file, DataNode first checks the token, and then it grants permission to the client to read that specific block. Following this step, a client will access the input stream and begin reading data from the DataNodes that have been configured. The client obtains the data in this fashion by reading it straight from the DataNode.

In the event that the DataNode unexpectedly goes down while a file is being read, a client will once again travel to the NameNode, and the NameNode will share another location with the client where that block may be found.

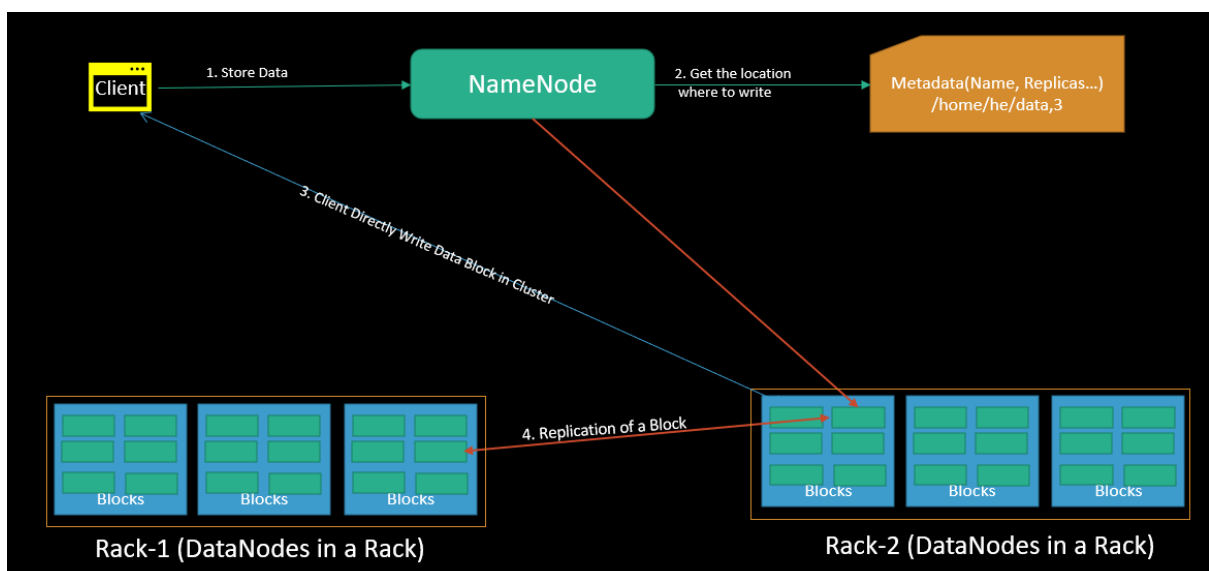


HDFS Operation for Writing

As can be observed when the client is reading a file, it is necessary for the client to interface with NameNode. In a similar way, in order for the client to write a file, they need to communicate with the NameNode.

NameNode gives the client the address of the slaves on which data has to be written in order for the client to complete the transaction.

After the client has completed writing the block, the slave will begin copying the block into another slave, which will then copy the block and send it to the third slave. When the standard replication factor of three is used, this is the result that is obtained. Once all of the necessary replications have been performed, it will send a final acknowledgement to the client.



Any time a client wants to write any data, it is required to communicate with the NameNode in order to do so. The client then communicates with the API for the distributed file system and requests that NameNode transmit a slave location.

The client will then begin writing the data by interacting with the DataNode at where the data has to be written and will begin writing the data via the FS data output stream. Following the completion of writing and replicating the data, the DataNode will send an acknowledgement to the client alerting them that the data has been written in its entirety.

When the client has finished writing to the first block, the first DataNode will immediately duplicate that block to any additional DataNodes that are connected to it. Therefore, after it has received the block, DataNode will begin the process of copying that block to the third DataNode. The third DataNode will send an acknowledgement to the second DataNode, the second DataNode will send an acknowledgment to the first DataNode, and finally, the first DataNode will send the final acknowledgment.

The client only sends one copy of the data regardless of the replication factor, but the DataNodes are responsible for replicating the blocks. Therefore, writing a file on Hadoop's HDFS does not incur any additional costs since many blocks of the file are written in parallel across various DataNodes.

Utilizing Cloudera Manager to relocate the JournalNode, which updates the directory for a role group: You have the ability to modify the location of the edit's directory for each JournalNode that is a part of the JournalNode Default Group, depending on the needs of your organisation.

By using Cloudera Manager, you may move the JournalNode edits directory for a role instance: You are free to adjust the location of the edits directory for one JournalNode instance in accordance with the specifications of your project.

Bringing the contents of JournalNodes into synchronisation: You have the ability to synchronise the data that is included inside the JournalNodes that are part of your CDP Private Cloud Base cluster. When this feature is enabled, it helps to preserve consistency in the contents of all of the JournalNodes that are distributed across the cluster. For instance, if the contents of a JournalNode become inconsistent, it is possible for that JournalNode to automatically duplicate the contents of the other JournalNodes in the cluster in order to restore consistency.

HDFS FAQ

Question-1: How NameNode handles the management of blocks on a DataNode that has failed?

Answer: After a certain amount of time has passed without any heartbeats, a DataNode is said to be dead.

Question-2: To replace a disc on a DataNode host, follow these steps?

Answer: You have the ability to repair defective discs that are hosted on the DataNode in your CDP Private Cloud Base cluster. Before you can replace the malfunctioning disc, all managed services need to be stopped, and the DataNode role instance has to be decommissioned.

Question-3: How do you take out one of the DataNodes?

Answer: Make sure that all of the conditions for deleting a DataNode have been completed before you attempt to remove it.

Question-4: How do you put an end to irregularities in the blocks?

Answer: You may get information on inconsistencies with the HDFS data blocks by using the output of the `hdfs fsck` or `hdfs dfsadmin -report` commands. These inconsistencies include missing,

misreplicated, or underreplicated blocks. You have the flexibility to choose from a variety of approaches to remedy these discrepancies.

Question-5: How do you use the Cloudera Manager to add storage directories?

Answer: Using Cloudera Manager, you may create a new storage directory and choose the kind of storage the directory will use.

Question-6: How do you use Cloudera Manager, get rid of the storage folders?

Answer: Using Cloudera Manager, you are able to delete already existing storage folders and define new directories.

Question-7: How do you set up the storage balancing configuration for DataNodes?

Answer: You have the ability to configure HDFS to distribute writes on each DataNode in a way that maintains a consistent level of available storage across all disc volumes on that DataNode.

Question-8: How do you utilize Cloudera Manager, carry out a disc hot swap on the DataNodes?

Answer: You won't need to restart a DataNode in order to change discs on the CDP Private Cloud Base cluster you're using. The term for this practise is "hot switch."

Question-9: What is HDFS used for?

Answer: Hadoop Distributed File System also known as HDFS is used for storing structure and unstructured data in distributed manner by using commodity hardware.

Question-10: What is Hadoop Distributed File System and what are its components?

Answer: Hadoop HDFS is a distributed file-system that stores data on commodity machines, providing very high aggregate bandwidth across the cluster.

Components of HDFS: HDFS comprises of 3 important components NameNode, DataNode and Secondary NameNode.

HDFS operates on a Master-Slave architecture model where the NameNode acts as the master node for keeping a track of the storage cluster and the DataNode acts as a slave node summing up to the various systems within a Hadoop cluster.

Question-11: What is NameNode and DataNode in HDFS?

Answer: Namenode is the master and DataNodes are slaves NameNode manages the filesystem namespace. It maintains the filesystem tree and the metadata for all the files and directories in the tree.

DataNodes are the workhorses of the filesystem. They store and retrieve blocks when they are told to (by clients or the NameNode), and they report back to the NameNode periodically with lists of Blocks that they are storing. Without the NameNode, the filesystem cannot be used.

Question-12: Why Hadoop uses filesystem for storage?

Answer: HDFS is built to support applications with large data sets, including individual files that reach into the terabytes. File systems are more affordable to handle huge amount of data.

Question-13: What is meant by Data node?

Answer: Data node is the slave deployed in each of the systems and provides the actual storage locations and serves read and writer requests for clients.

Question-14: What is daemon?

Answer: Daemon is the process that runs in background in the UNIX environment. In Windows it is 'services' and in DOS it is 'TSR'.

Question-15: What is meant by heartbeat in HDFS?

Answer: Data nodes and task trackers send heartbeat signals to Name node and Job tracker respectively to inform that they are alive. If the signal is not received it would indicate problems with the node or task tracker.

Question-16: Is it necessary that Name node and job tracker should be on the same host?

Answer: No! They can be on different hosts.

Question-17: What is meant by 'block' in HDFS?

Answer: Block in HDFS refers to minimum quantum of data for reading or writing. Default block size is 128 MB in HDFS.

Question-18: Can blocks be broken down by HDFS if a machine does not have the capacity to copy as many blocks as the user wants?

Answer: Blocks in HDFS cannot be broken. Master node calculates the required space and how data would be transferred to a machine having lower space.

Question-19: How is data replicated in HDFS?

Answer: HDFS is designed to be fault-tolerant. Large HDFS data files are split into smaller chunks called blocks and each block is stored in multiple DataNodes across the cluster. The block size and the replication factor can be configured per file.

HDFS is rack aware for multi-clustered environments, and takes this into consideration when replicating blocks for fault-tolerance. HDFS ensures that the blocks are replicated on DataNodes that are on different racks, so if a rack goes down the data is still available from the DataNode on the other rack.

Question-20: Explain how indexing in HDFS is done?

Answer: Hadoop has a unique way of indexing. Once the data is stored as per the block size, the HDFS will keep on storing the last part of the data which say where the next part of the data will be.

Chapter-2 Apache Ozone

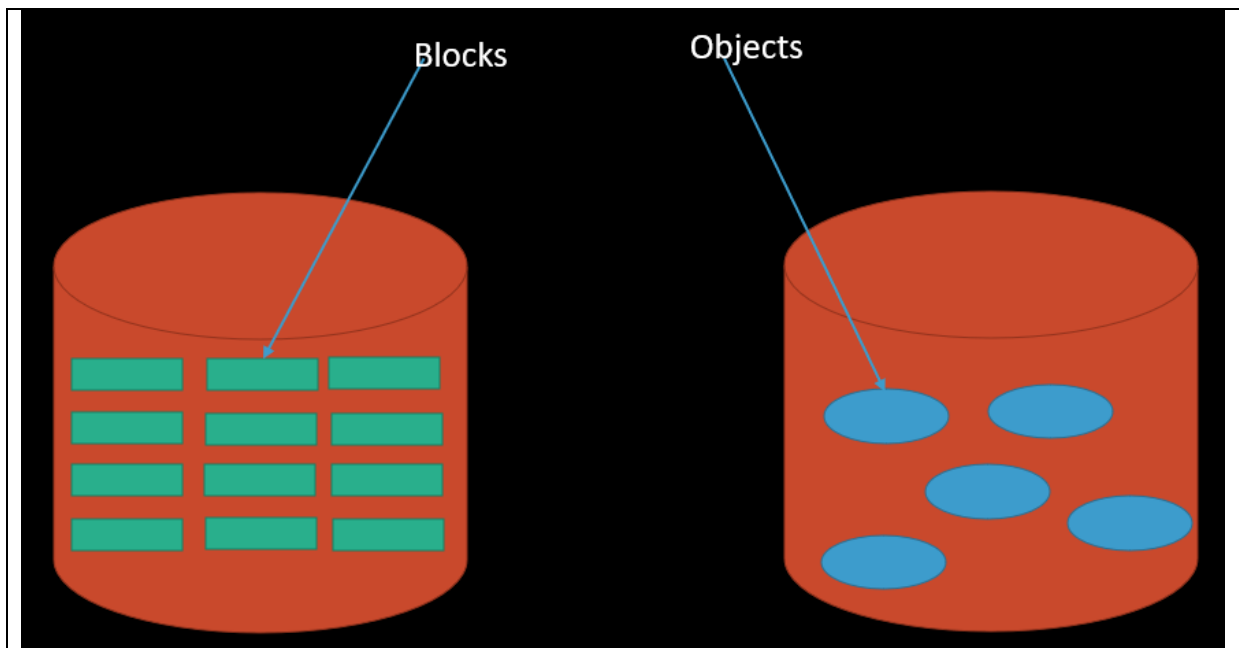
Overview

Apache Ozone is a tool or product that is used to implement Hadoop's Object Storage architecture. As we know that Hadoop Distributed File System (HDFS) is used as a native storage for storing data in Hadoop ecosystem. However, Hadoop Distributed File System (HDFS) is a block storage system that was created by Apache and designed to be used as a storage layer on the Hadoop architecture.

Object storage is the name given to the new storage model that Ozone, that is also a storage engine for Hadoop ecosystem. Within the same Hadoop cluster, it can co-exist with HDFS to offer file store and object store capability. Similar to AWS S3 and Google Cloud storage, you can create object storage for your Hadoop Cluster as well and it can store trillions of files on Ozone, and they can access those files just as if they were stored on HDFS.

Ozone also supports the scalability and tiny file issues that HDFS has. Ozone can be easily integrated into already existing Hadoop installations, and applications like Hive and Spark can run without requiring any adjustments.

Apache has built an object storage system known as Ozone. It is designed to be used inside the architecture in a manner similar to that of HDFS. Specifically, as a storage layer. Below image shows Comparison between Object Storage and Block Storage (HDFS) in graphical form (Ozone).



Block storage: Fragments the data into smaller chunks, which are subsequently stored independently as blocks. The Storage Area Network (SAN) will position the data blocks in the locations where they will be used most effectively. Each block is assigned a unique identification. Because of this, data may be saved wherever it will be more easily accessible, rather than inside the same system. The access to data in block storage does not depend on a single, centralised route. Because of this, the information may be accessed very rapidly.

Storage of Objects: Files are fragmented into smaller parts and dispersed over various devices as part of the object storage system, which has a flat structure. Volumes and containers are used for object storage in place of traditional blocks. The data is stored in volumes, which are modular storage containers that are completely self-contained. Every item included inside it is assigned a distinct identification in addition to the information that details the data. The unique identifier in block storage is made up of two IDs, as opposed to only one. One that tells you which bucket the data is kept on and where it is stored.

And another one that specifies the location inside the bucket where the data is being kept. The metadata may be tailored and specified to the user's specifications. It includes information such as

age, access permissions, and security. Object Storage offers excellent value for the money. You are only responsible for paying for what you use. It is not difficult to scale up.

Large quantities of static and unstructured data are ideal candidates for this sort of storage. Once they have been generated, objects cannot have their properties changed in any way, which is one of the most significant drawbacks. It is required to create a whole new object if you want to make changes to an existing one. Writing objects is a more time-consuming procedure than writing to block storage, which is another reason why standard structured databases do not perform well with object storage.

Apache Ozone is a distributed object store that is scalable, redundant, and optimised for the demands associated with large data. Applications that make use of frameworks such as Apache Spark, Apache YARN, and Apache Hive function natively on Ozone without requiring any changes. This is in addition to Ozone's ability to scale to billions of objects of varied sizes. Ozone offers a Hadoop-compatible file system interface and has native support for the S3 application programming interface. The CDP Private Cloud Base deployment is the common location where Ozone may be found.

Ozone is made up of three essential components that are used for storing information: volumes, buckets, and keys. Each key is a component of a bucket, and the buckets together make up a volume. The creation of volumes is restricted to administrators only. Regular users are able to build buckets in varying quantities, depending on the needs that they have. Within these buckets, Ozone maintains data in the form of keys.

Ozone saves the accompanying data on DataNodes in chunks that are referred to as blocks. This happens whenever a client submits a key. As a result, any key may open anywhere from one to several different chests. Multiple blocks that are not connected to one another may coexist in the same storage container inside of a DataNode.

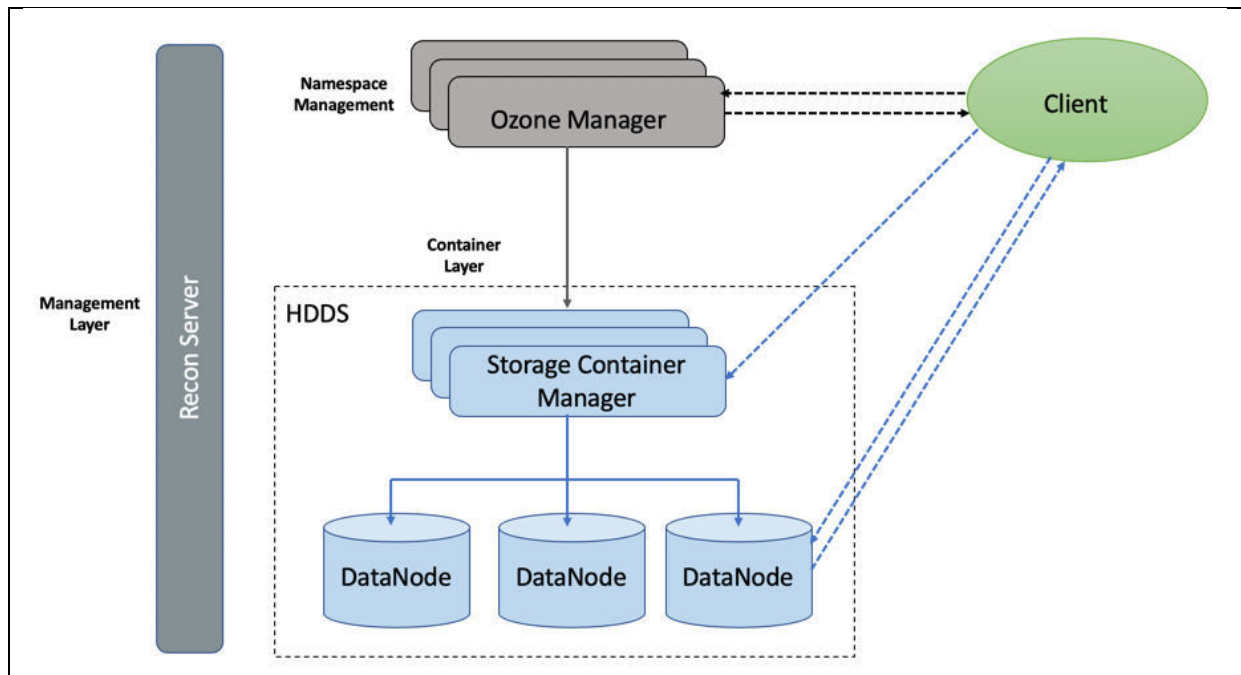
HDFS is the de facto large data file system. It's easy to forget how scalable and reliable HDFS is. Our customers operate clusters with thousands of nodes that serve thousands of concurrent clients.

HDFS operates best with huge files, tens to hundreds of megabytes. HDFS has a modest file capacity and struggles with 400M files. HDFS-like storage that can handle billions of little files is in demand. Ozone can handle tiny and big files. Ozone is an Object Store whereas HDFS is POSIX-like.

Ozone architecture

Ozone can be co-located with HDFS with single security and governance policies for easy data exchange or migration and also offers seamless application portability. Ozone has a scale-out architecture with minimal operational overheads. Ozone separates management of namespaces and storage, helping it to scale effectively. The Ozone Manager (OM) manages the namespaces while the Storage Container Manager (SCM) handles the containers.

The following diagram shows the components that form the basic architecture of Ozone:



Hadoop Distributed Data Store: Ozone is built on a highly available, replicated block storage layer called Hadoop Distributed Data Store (HDDS).

Blocks: Blocks are the basic unit of storage. In Ozone, each block is of 256 MB in size. A collection of blocks forms a storage container. The SCM allocates blocks inside storage containers for the client to store data.

Storage Containers: A storage container is a group of unrelated blocks managed together as a single entity. A container exists in a DataNode and is the basic unit of replication, with a capacity of 2 GB to 16 GB.

The Storage Container Manager performs multiple critical functions for an Ozone cluster. SCM manages the addition and removal of DataNodes, and allocates storage containers and blocks. SCM also manages block collections, ensuring that the blocks maintain the required level of replication. SCM allocates blocks to clients through OM for read and write operations. In addition, SCM executes recovery actions when faced with DataNode or disk failures.

DataNodes: DataNodes contain storage containers comprising of data blocks. The SCM monitors DataNodes through heartbeats.

Ozone Manager: The Ozone Manager (OM) is the metadata manager for Ozone. The OM manages the following storage elements:

- The list of volumes for each user
- The list of buckets for each volume
- The list of keys for each bucket

The OM maintains the mappings between keys and their corresponding Block IDs. When a client application requests for keys to perform read and write operations, the OM interacts with the SCM for information about blocks relevant to the read and write operations, and provides this information to the client. In addition, the OM also handles metadata operations from the clients.

The Ozone Manager (OM) is a highly available namespace manager for Ozone. OM manages the metadata for volumes, buckets, and keys. OM maintains the mappings between keys and their corresponding Block IDs. When a client application requests for keys to perform read and write operations, OM interacts with SCM for information about blocks relevant to the read and write operations, and provides this information to the client. In addition, OM also handles metadata operations from the clients.

Pipelines: Pipelines determine the replication strategy for the blocks associated with a write operation.

Recon Server: Recon is the management interface for Ozone. Recon provides a unified management API for Ozone.

How Ozone manages read operations: The client requests the block locations corresponding to the key it wants to read. The Ozone Manager (OM) returns the block locations if the client has the required read privileges.

The following steps explain how Ozone manages read operations:

1. The client requests OM for block locations corresponding to the key to read.
2. OM checks the ACLs to confirm whether the client has the required privileges, and returns the block locations and the block token that allows the client to read data from the DataNodes.
3. The client connects to the DataNode associated with the returned Block ID and reads the data blocks.

How Ozone manages write operations: The client requests blocks from the Ozone Manager (OM) to write a key. OM returns the Block ID and the corresponding DataNodes for the client to write data.

The following steps explain how Ozone manages write operations:

1. The client requests blocks from OM to write a key. The request includes the key, the pipeline type, and the replication count.
2. OM finds the blocks that match the request from SCM and returns them to the client.

If security is enabled on the cluster, OM also provides a block token along with the block location to the client. The client uses the block token to connect to the DataNodes and send the command to write chunks of data.

1. The client connects to the DataNodes associated with the returned block information and writes the data.
2. After writing the data, the client updates the block information on OM by sending a commit request.
3. OM records the associated key information.

Notes

- a. Keys in Ozone are not visible until OM commits the block information associated with the keys. The client is responsible for sending the key-block information to OM after it has written the blocks on the DNs via a commit request.
- b. If OM fails to commit block information for keys after they have been written, for example, client was unable to send the commit request OM because the write job failed, the keys would not be visible but the data would remain on disk.

Tenets: Ozone's design followed these guidelines:

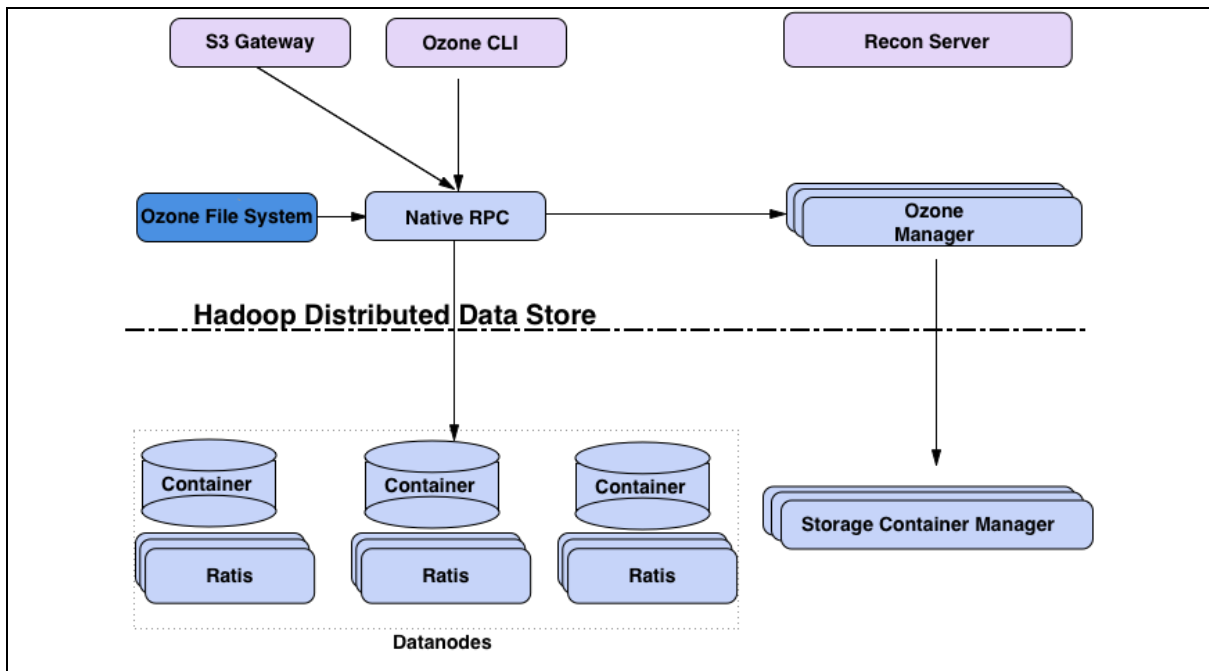
- **Consistent:** Consistency facilitates app development. Ozone is serializable.
- **Simplicity:** A basic architecture is simpler to understand and debug. We've kept Ozone's architecture basic despite its scalability. Ozone scales well. Over 100 billion items may be stored in a single cluster.
- **Layered Architecture:** Ozone is a layered file system for current storage systems. It isolates namespace management from block and node management, allowing scaling on both axes.
- **Pain-free recovery:** HDFS can recover from cluster-wide power outage without losing data or requiring costly recovery methods. Losses in racks and nodes are negligible. Ozone will withstand failures similarly.
- **Apache's Open Source:** Apache Open Source is crucial to Ozone's success. The Apache Hadoop community designs and develops Ozone.
- **Hadoop interoperability:** Ozone should work with current Apache Hadoop applications like Hive, Spark, and MapReduce. Therefore, Ozone:
 - Hadoop FSA (aka OzoneFS). Hive, Spark, etc. may utilise Ozone without modification.
 - Localization. Original HDFS/MapReduce allowed computation operations to be scheduled on the same nodes as the data. Ozone supports application data localization.
 - Deploy HDFS side-by-side. Ozone may share HDFS discs in an existing Hadoop cluster.

Ozone Concepts: Volumes, buckets, and Keys are the component parts that make up ozone.

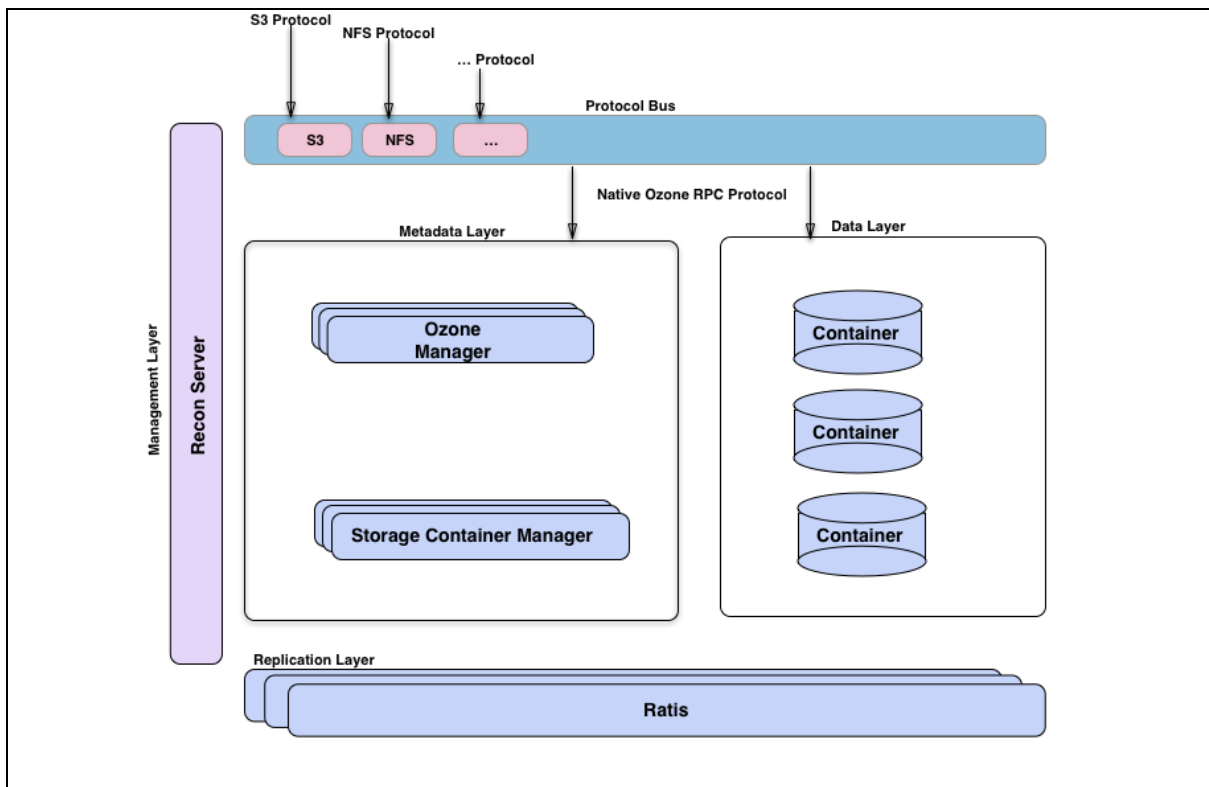
Ozone is an extensible, fault-tolerant, and distributed object storage that is built on top of Hadoop. In addition to being able to handle billions of items of varied sizes, Ozone is also capable of performing well in containerized settings such as those provided by Kubernetes. When Ozone is used, applications like as Apache Spark, Hive, and YARN function normally and do not need any adjustments. It is quite simple to use Ozone due to the fact that it comes equipped with a Java client library, support for the S3 protocol, and a command line interface.

- **Volumes:** Volumes are comparable to accounts in their function. Administrators are the only people who can create new volumes or remove existing ones. The creation of a volume for an organisation or team will normally be handled by an administrator.
- **Buckets:** A volume may have zero, one, or more buckets inside of it. Buckets in Ozone function in a manner comparable to those in Amazon S3.
- **Keys:** are objects that are exclusive to a certain bucket and are analogous to S3 Objects in their function. Any string may be used for a key name. The data that you store inside these keys is denoted by values, and Ozone does not yet impose any kind of maximum size restriction on key sizes.

Ozone is a redundant, distributed object store optimized for Big data workloads. The primary design point of ozone is scalability, and it aims to scale to billions of objects. Ozone separates namespace management and block space management; this helps ozone to scale much better. The namespace is managed by a daemon called Ozone Manager (OM), and block space is managed by Storage Container Manager (SCM). Ozone consists of volumes, buckets, and keys. A volume is similar to a home directory in the ozone world. Only an administrator can create it. Volumes are used to store buckets. Once a volume is created users can create as many buckets as needed. Ozone stores data as keys which live inside these buckets. Ozone namespace is composed of many storage volumes. Storage volumes are also used as the basis for storage accounting. The block diagram shows the core components of Ozone.



The Ozone Manager is the name space manager, Storage Container Manager manages the physical and data layer and Recon is the management interface for Ozone.



Advanced Concepts

Any distributed system can be viewed from different perspectives. One way to look at Ozone is to imagine it as Ozone Manager as a name space service built on top of HDDS, a distributed block store. Another way to visualize Ozone is to look at the functional layers; we have a metadata data

management layer, composed of Ozone Manager and Storage Container Manager. We have a data storage layer, which is basically the data nodes and they are managed by SCM. The replication layer, provided by Ratis is used to replicate metadata (OM and SCM) and also used for consistency when data is modified at the data nodes. We have a management server called Recon, that talks to all other components of Ozone and provides a unified management API and UX for Ozone.

We have a protocol bus that allows Ozone to be extended via other protocols. We currently only have S3 protocol support built via Protocol bus. Protocol Bus provides a generic notion that you can implement new file system or object store protocols that call into O3 Native protocol.

[Get Full Contents from this link](#)

Chapter-3 Apache Hive

Overview

- Apache Hive is a Hadoop-based data warehouse for searching and analysing massive Hadoop datasets. Hadoop processes structured and semi-structured data.
- Initially, you had to construct sophisticated Map-Reduce tasks to use Hadoop as a Big Data Processing engine, but now you can submit SQL queries. Hive targets SQL-savvy users.
- Hive uses HQL, a SQL-like language. HiveQL converts SQL-like queries to MapReduce jobs.
- Hive simplifies Hadoop. And you don't need to learn java if you know ANSI SQL to work with BigData.
- Your SQL query is transformed into a series of Map Reduce tasks by the Hive client, which typically runs on your gateway/client node and sends them to a Hadoop cluster for execution.
- Hive arranges the Data into tables, which helps for providing an structure to HDFS data.
- Data analysis is performed using Hive, which is a Data Warehousing software that was created on top of Hadoop. Additionally, Hive makes use of a language known as HiveQL (HQL), which automatically converts queries that are similar to SQL into tasks for MapReduce.

Hive Features

- Before Apache Hive, there were several problems with Big Volume of data. Data size are keep growing and making it tough to manage and the typical RDBMS failed to handle this volume.
- Most of the organizations attempted MapReduce to solve this issue. However, writing MapReduce was another programming challenges, everybody has to learn Java and write complex MapReduce jobs.
- Apache Hive helped a lot to overcome above problems.
- BigData Companies are now able to execute the following using Apache Hive:
 - Schema flexibility and evolution
 - Tables can be portioned and bucketed
 - Apache Hive tables are defined directly in the HDFS
 - JDBC/ODBC drivers are available
- For ad hoc needs, Apache Hive spares developers from designing difficult Hadoop MapReduce processes. So, hive offers data summary, analysis, and querying.
- Hive is scalable and very quick. It may be extended greatly. Because Apache Hive and SQL are so similar, learning and using Hive Queries is fairly simple for SQL developers.

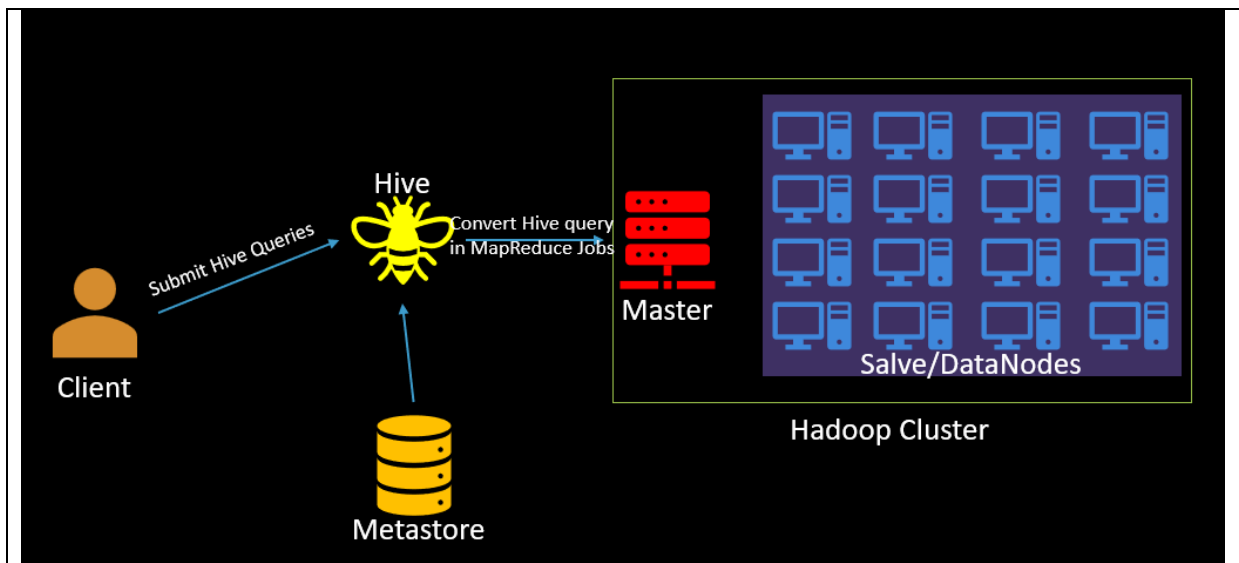
- By giving users a way to submit SQL queries via an interface, Hive lessens the complexity of MapReduce. Therefore, with Apache Hive, business analysts may now experiment with big data and provide insights.
- Additionally, it offers file access to a number of data storage, including HDFS and HBase. The fact that we don't need to understand Java in order to use Apache Hive is its most significant feature.

Hive Architecture

Apache Hive has following major components:

1. **Metastore:**
 - a. It stores metadata for each of the tables like their schema and location.
 - b. Metastore stores the information about table partitions.
 - c. Driver connects to metastore to get the detail about various data sets distributed over the cluster nodes.
 - d. Metastore is always a traditional RDBMS store like MySQL, Oracle RDBMS.
 - e. Hive metadata helps the driver to keep a track of the data and it is highly crucial.
2. **Driver:**
 - a. This is similar to JDBC driver which receives the HiveQL statements.
 - b. The driver starts the execution of the statement by creating sessions.
 - c. It monitors the life cycle and progress of the execution.
 - d. Driver stores the necessary metadata generated during the execution of a HiveQL statement.
 - e. It also acts as a collection point of data or query result obtained after the Reduce operation.
3. **Compiler:**
 - a. Every Hive SQL statement needs to be compiled before it performs the actual query execution.
 - b. Compiler generates the query to an execution plan.
 - c. The execution plan contains the tasks.
 - d. And contains the steps needed to be performed by the MapReduce to get the output as translated by the query.
 - e. The compiler in Hive converts the query to an Abstract Syntax Tree (AST).
 - f. First, check for compatibility and compile-time errors, then converts the AST to a Directed Acyclic Graph (DAG).
4. **Optimizer:**
 - a. It performs various transformations on the execution plan to provide optimized DAG.
 - b. It aggregates the transformations together, such as converting a pipeline of joins to a single join, for better performance.
 - c. The optimizer can also split the tasks, such as applying a transformation on data before a reduce operation, to provide better performance.
5. **Executor:**
 - a. Once compilation and optimization complete, the executor executes the tasks.
 - b. Executor takes care of pipelining the tasks.
6. **CLI, UI, and Thrift Server:**
 - a. CLI (command-line interface) provides a user interface for an external user to interact with Hive.

- b. Thrift server in Hive allows external clients to interact with Hive over a network, similar to the JDBC or ODBC protocols.



Benefits of using Apache Hive

- Hive makes it considerably simpler to do data analysis, queries, and summarization on large amounts of data.
- Since Hive supports external tables, it is feasible to process data without actually storing it in HDFS. This is made possible by the fact that Hive supports external tables.
- Apache Hive is an excellent choice for fulfilling the low-level interface requirement of Hadoop.
- It also enables the division of data at the table level to boost efficiency, and this feature is supported.
- Hive is equipped with a rule-based optimizer with the purpose of improving logical plan performance.
- It is expandable, scalable, and has a user-friendly interface.
- Using HiveQL does not need any prior knowledge of programming languages; all that is required is a fundamental understanding of SQL queries.
- Using Hive, we can simplify the process of processing structured data in Hadoop.
- Because it is so close to SQL, querying in Hive is a pretty straightforward process.
- Through the use of Hive, we can also conduct ad hoc queries for the data analysis.

Problems with Hive

- There is no provision for real-time queries or changes at the row level in Apache Hive.
- Additionally, Hive delivers a latency that is suitable for interactive data browsing.
- It is not beneficial for the processing of online transactions.
- In most cases, Apache Hive queries are characterised by a fairly long latency.

Facts about Apache Hive

- The Apache Hive is a Structural Framework on Hadoop.
- Using Hive, you can query massive datasets that are stored in remote storage might be helpful.
- Apache Hive is a kind of distributed data warehouse.

- HiveQL is a query language that is similar to SQL (HQL).
- HiveQL is a declarative programming language similar to SQL.
- The Hive table structure is analogous to the tables in a relational database.
- Using the Hive-QL query language, several users may concurrently query the data.
- Hive, on the other hand, enables the building of bespoke MapReduce framework processes that may be used to do more in-depth data analysis.
- It is simple to extract, convert, and load (ETL) data using Apache Hive from HDFS.
- Hive provides the framework for a broad range of data types.
- Hive makes it possible to access files that are stored in HDFS.
- In addition to that, Apache Hive enables the conversion of a wide number of formats.
- However, Hive is not intended for use in the processing of online transactions (OLTP). Despite this, we are able to put it to use for online analytical processing (OLAP).
- Apache Hive does not support updates or deletes, but it does support overwriting and acquiring data.
- Hive does not support the use of subqueries, while writing this study material.

FAQ for Apache Hive

Question-1. What is Hive Metastore?

Answer: Hive metastore is a database that stores metadata about your Hive tables (eg. Table name, column names and types, table location, storage handler being used, number of buckets in the table, sorting columns if any, partition columns if any, etc.). When you create a table, this metastore gets updated with the information related to the new table which gets queried when you issue queries on that table.

Question-2: Wherever (Different Directory) I run hive query, it creates new metastore_db, please explain the reason for it?

Answer: Whenever you run the hive in embedded mode, it creates the local metastore. And before creating the metastore it looks whether metastore already exist or not. This property is defined in configuration file hive-site.xml. Property is "javax.jdo.option.ConnectionURL" with default value "jdbc:derby;;databaseName=metastore_db;create=true". So to change the behavior change the location to absolute path, so metastore will be used from that location.

Question-3: Is it possible to use same metastore by multiple users, in case of embedded hive?

Answer: No, it is not possible to use metastore in sharing mode. It is recommended to use standalone "real" database like MySQL or PostGresSQL.

Question-4: Is multiline comment supported in Hive Script ?

Answer: No.

Question-5: If you run hive as a server, what are the available mechanism for connecting it from application?

Answer: There are following ways by which you can connect with the Hive Server:

1. Thrift Client: Using thrift you can call hive commands from a various programming
a. languages e.g. C++, Java, PHP, Python and Ruby.
2. JDBC Driver : It supports the Type 4 (pure Java) JDBC Driver
3. ODBC Driver: It supports ODBC protocol.

Question-6: What is SerDe in Apache Hive?

Answer: A SerDe is a short name for a Serializer Deserializer. Hive uses SerDe (and FileFormat) to read and write data from tables. An important concept behind Hive is that it DOES NOT own the Hadoop File System (HDFS) format that data is stored in. Users are able to write files to HDFS with whatever tools/mechanism takes their fancy("CREATE EXTERNAL TABLE" or "LOAD DATA INPATH,") and use Hive to correctly "parse" that file format in a way that can be used by Hive. A SerDe is a powerful (and customizable) mechanism that Hive uses to "parse" data stored in HDFS to be used by Hive.

Question-7: Which classes are used by the Hive to Read and Write HDFS Files

Answer: Following classes are used by Hive to read and write HDFS files

- TextInputFormat/HiveIgnoreKeyTextOutputFormat: These 2 classes read/write data in plain text file format.
- SequenceFileInputFormat/SequenceFileOutputFormat: These 2 classes read/write data in hadoop SequenceFile format.

Question-8. Give examples of the SerDe classes which hive uses to Serialize and Desterilize data ?

Answer: Hive currently use these SerDe classes to serialize and deserialize data:

- MetadataTypedColumnsetSerDe: This SerDe is used to read/write delimited records like CSV, tab-separated control-A separated records (quote is not supported yet.)
- ThriftSerDe: This SerDe is used to read/write thrift serialized objects. The class file for the Thrift object must be loaded first.
- DynamicSerDe: This SerDe also read/write thrift serialized objects, but it understands thrift DDL so the schema of the object can be provided at runtime. Also it supports a lot of different protocols, including TBinaryProtocol, TJSONProtocol, TCTLSeparatedProtocol (which writes data in delimited records).

Question-9. Can you use Apache Hive for OLTP systems?

Answer: No, it is not suitable for OLTP system because it does not offer insert and update at the row level.

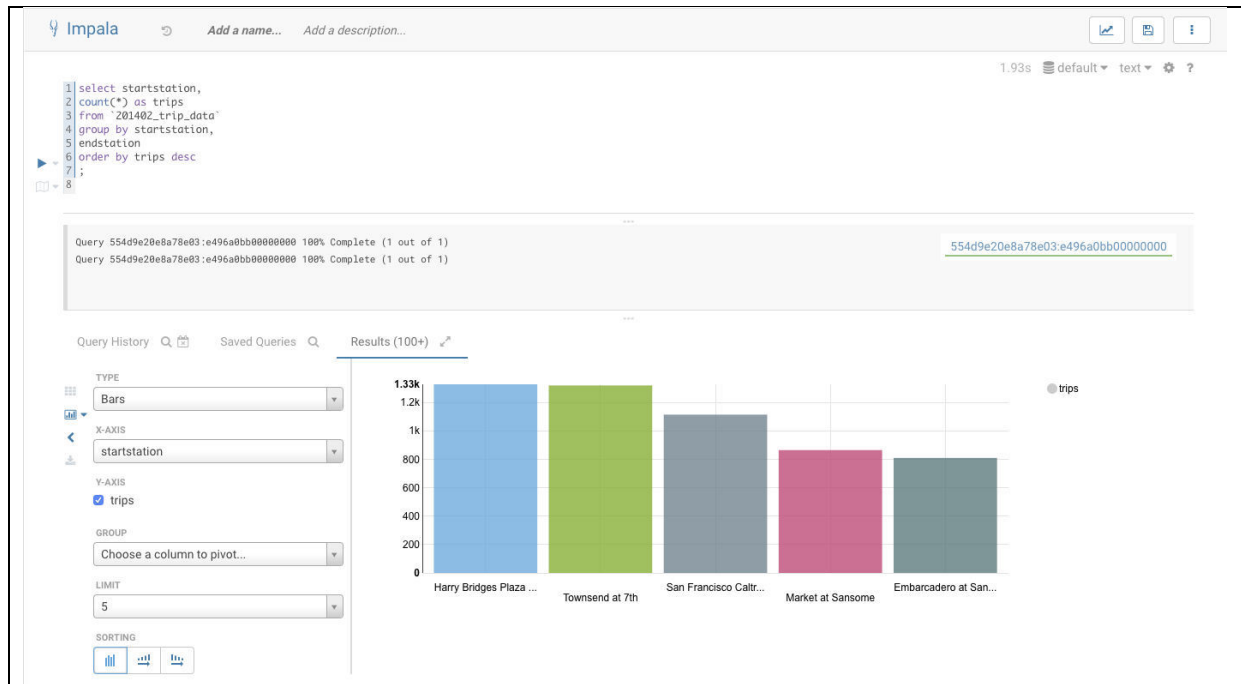
Question-10. Why does Apache Hive not store metadata information in HDFS and instead it needs RDBMS?

Answer: Hive stores metadata information in the metastore which must be an RDBMS, so that it can achieve low latency. Since, HDFS read/write operations are time-consuming processes.

[Get Full Contents from this link](#)

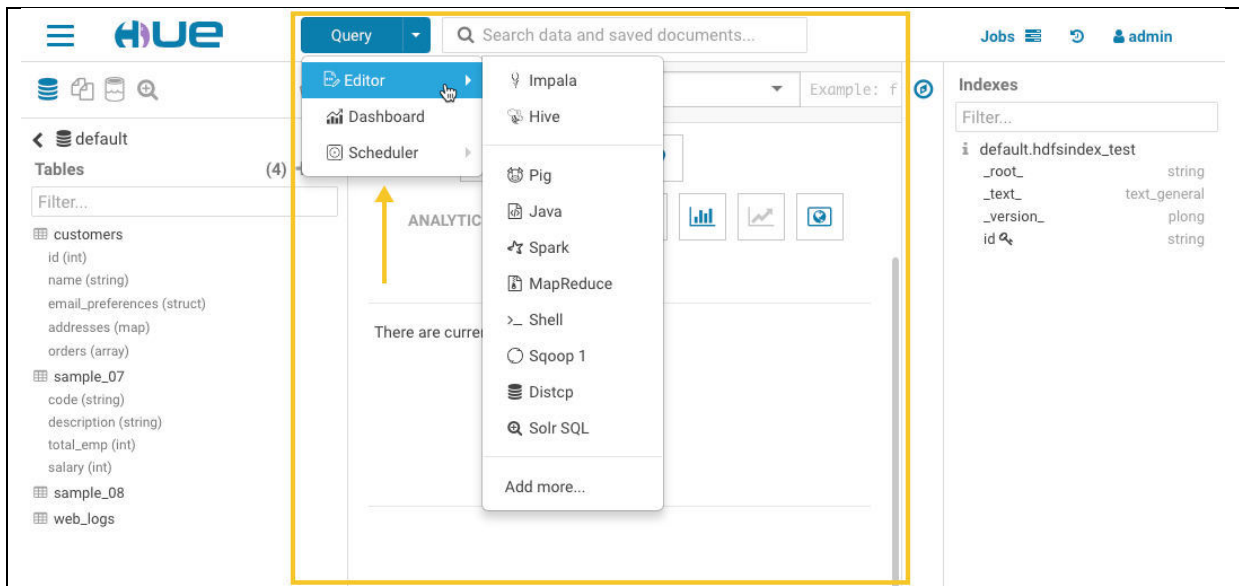
Overview

Hue is an interactive query editor that runs in a web browser and gives users the ability to interact with data warehouses. This is one of the most widely used Query Editor and HadoopExam will also use this heavily in their training programs specially on HDFS, Sqoop, Hive and Impala. Through the use of Hue, you can provide your SQL developers access to the power of business intelligence (BI) and analytics. Hue is an analytics workbench that is open source and was developed for easy and quick data discovery, as well as intelligent query support and seamless collaboration. Create a bridge between information technology and the company in order to provide reliable self-service analytics. For illustration purposes, the following image demonstrates a graphical representation of the results of an Impala SQL query that may be generated with Hue:



Using Hue, you can do following activities

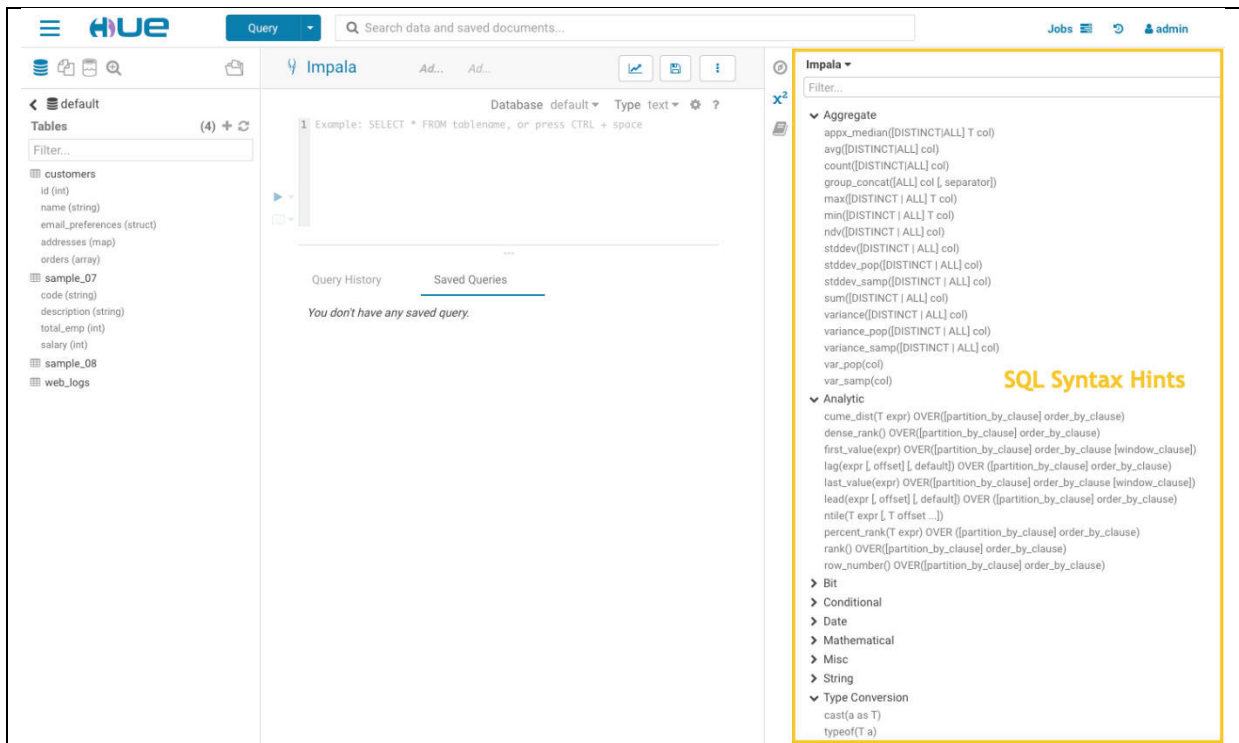
1. **Hive & Impala:** Explore your various databases.
2. **Schema Tables:** Proceed to the specific tables afterward.
3. **Files:** Navigate through the HDFS directories and the cloud storage.
4. **Table Detail:** Find the indexes and tables stored in HBase or Kudu.
5. **Documents:** Find documents



The primary section of the Hue UI has a comprehensive collection of tools, which include the following:

- **Editing environments:** that are flexible and enable the creation of a broad range of scripts.
- **Dashboards:** that may be constructed "on the fly" by dragging and dropping objects into the centre panel of the Hue user interface. There is no need for programming at all. After that, you may study your statistics by using your individualised dashboard.
- **Schedulers:** The tool that allows you to construct schedulers by dragging and dropping, similar to how dashboards are created. Using this feature, you will be able to construct individualised processes and set them to execute on a predetermined schedule at certain intervals. A monitoring interface will provide the current status of the tasks, as well as logs, and will allow you to pause or cancel them.

Assistant Panel: The assistant panel you can find to the right of the main panel, and it offers tips and guidance pertinent to the application that is now being used in the main panel. For instance, in the picture that can be seen above, there is a middle panel that contains Impala SQL tips that may assist in the construction of queries.



Hue Design and Architecture

In general, each Hue server is capable of supporting roughly 25 concurrent users, however this number might vary based on the tasks being carried out by the users. It is not the number of users that causes the majority of scaling problems; rather, it is the actions that users execute that are resource expensive. For instance, downloading a significant amount of query results may have an effect on the availability of resources for other users who are using the same instance of Hue at the same time as the download process. During such period, the users could notice that performance is slowing down. Another typical factor that might bring about observable shifts in performance are sluggish RPC calls made between Hue and another service. When this occurs, the queries that are sent may give the impression that they are "hanging" all of a sudden.

As a general rule, two Hue servers may support up to the following:

- 100 unique users per week
- At peak periods, there are 50 users per hour who can execute up to 100 queries.
- In a typical configuration, there are two Hue servers.

Administrator

- Install a load balancer on the path leading up to Hue.
- Make use of a database of production-quality. Hue Custom Databases may be seen here for further details.
- Make sure that other services, such Impala, Hive, and Oozie, are in good health and are not being negatively affected by the lack of resources. If any of these services become unresponsive, it will have a negative impact on Hue's performance.
- It is a good idea to think about shifting workloads that are governed by service-level agreements (SLAs) or are regarded as "noisy neighbours" to their own computing cluster. Workloads that use the bulk of the available resources and result in performance concerns

are referred to as "noisy neighbours." Look into Virtual Private Clusters and Cloudera SDX if you want to learn more about how to keep your computation and storage functions separate.

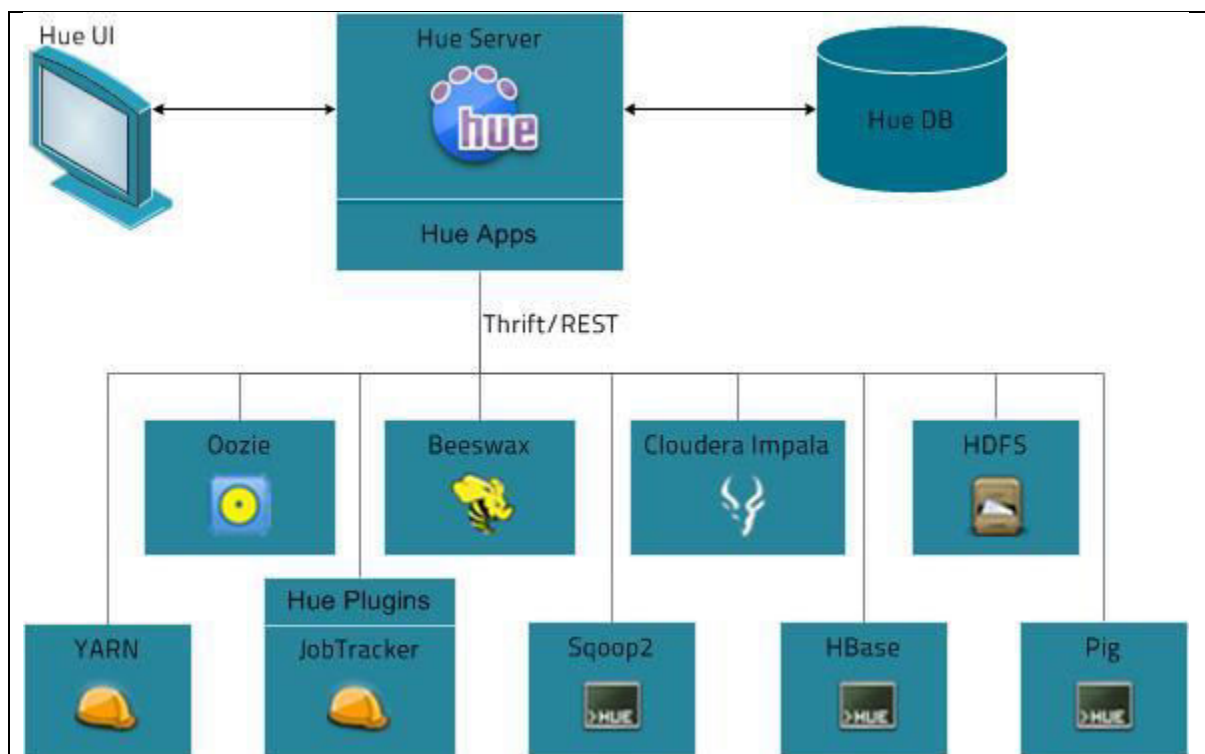
- Set a maximum for the number of rows that will be returned in response to queries.
- Setting a value for the download row limit configuration variable of the Hue Beeswax application is one approach to restrict the number of rows that are retrieved by the application. This property may be set in the Hue Service Advanced Configuration Snippet (Safety Valve), which is located in the Cloudera Manager configuration file for the hue safety valve.ini property.

Major Functionalities of Hue in CDP

Hue is a web-based interface for using Apache Hadoop to do data analysis. It is possible to install it on any computer running any version of Hadoop.

Hue is a collection of apps that allows users to have web-based access to CDH components and functions as a platform for the development of bespoke applications.

The diagram that follows provides an illustration of how Hue works. The Hue Server web application functions as a "container" that bridges the gap between your CDH installation and the browser. It interfaces with a variety of servers that interface with CDH components and hosts the Hue apps.



Hue, which stands for Hadoop User Experience, is a graphical user interface that is open-source, web-based, and designed for use with Apache Hadoop and Cloudera CDP. A flexible user interface is created by Hue, which brings together a number of separate Hadoop ecosystem projects. Cloudera CDP now includes additional customization options that are exclusive to the Hue. Hue functions as a front-end for apps that operate on your cluster. This makes it possible for you to engage with applications using an interface that is maybe more intuitive or well-known to you. It is no longer

necessary to log in to the cluster in order to execute scripts interactively using each application's corresponding shell since Hue's apps, such as the Hive and Pig editors, eliminate this need. It's possible that after a cluster is up and running, you'll communicate only with its apps using Hue or a comparable interface.

Hue supported features:

- Amazon S3 and Hadoop File System (HDFS) Browser
- With the appropriate permissions, you can browse and move data between the ephemeral HDFS storage and S3 buckets belonging to your account.
- Hive—Run interactive queries on your data. This is also a useful way to prototype programmatic or batched querying.
- Pig—Run scripts on your data or issue interactive commands.
- Oozie—Create and monitor Oozie workflows.
- Metastore Manager—View and manipulate the contents of the Hive metastore (import/create, drop, and so on).
- Job browser—See the status of your submitted Hadoop jobs.
- User management—Manage Hue user accounts and integrate LDAP users with Hue.
- To use the Hue Notebook for Spark, you must install Hue with Livy and Spark.
- **Quickly identify relevant data:** For analytics that are both quicker and more reliable, it is possible to search and find relevant tables, views, and columns across all databases, including cloud-native object stores, with ease. Ensure that data can be trusted instantly by incorporating data stewardship into the system and allowing end users to tag data for further classification and organising that is project-focused.
- **Keep your changes safe and iterate across teams:** Through seamless and safe collaboration and sharing, silos of analytics and business intelligence may be eliminated. You can safeguard even the most sensitive data by saving searches and result sets for later use, sharing them with other users or departments, and explicitly setting access rights for those saved items.
- **Intelligent inquiry design and help:** The only SQL editor with usage-enriched intelligence to safeguard against malicious queries and make SQL users more productive. You may explore and do analytics in an efficient and iterative manner by dragging and dropping tables and columns, rapidly designing queries using autocomplete pop-ups, and receiving query suggestions based on use and best practises.
- **Native integration with the cloud:** Browse through all databases, run queries on them, and store the results in both on-premises and cloud-based systems. Apache Hive is used for data preparation, Apache Solr is used for free-text analytics, and Apache Impala is used for high-performance SQL analytics. Hue interfaces with the full of Cloudera's platform, including storage engines, Apache Kudu and Amazon S3 object storage.
- **Include data scientists and analysts in the analytics process:** Cloudera's platform also enables self-service data science with the Cloudera Data Science Workbench, which can be carried out over the same shared data by users who are more familiar with the programming languages R, Python, or Scala. In addition, the platform is compatible with all of the industry's leading business intelligence (BI) and visualisation tools, such as Tableau, Qlik, Zoomdata, and many others. This allows businesses to keep utilising the tools they already rely on while taking advantage of the scalability and flexibility offered by Cloudera.

Hive is a collection of keys and subkeys that are accompanied by a series of supporting files that store backups of the data. To put it simply, the hive is the area where information about the Windows registry is stored. Each hive has a tree, and each tree has a unique key. This key acts as the tree's root, which is the point at which the tree begins or the highest point in the hierarchy inside the register. There are registry keys, registry subkeys, and registry values included inside the registry. The prefix "HKEY" is included at the beginning of every key that belongs to a hive. When all of the other entries in the registry are minimised, a group of keys known as hives will show up on the left-hand side of the screen in the form of folders. One is not possible to establish a Hive, remove one, or rename it. During the early phases of development, Facebook was responsible for launching the hive; however, the Apache Software Foundation eventually took over management of the project.

Both a web user interface that offers a variety of services and a Hadoop framework, Hue is referred to simply as Hue. Hue has a web user interface for browsing HDFS files, in addition to providing the file path. The Job browser, the Hadoop shell, User administrative rights, the Impala editor, the HDFS file browser, the Pig editor, the Hive editor, the Ozzie web interface, and Hadoop API Access are the most essential aspects of Hue. This online user interface style makes it easier for users to browse among the files, in a manner similar to how a typical Windows user would navigate to his or her files on their local PC. Users are able to avoid making syntax mistakes when conducting queries with the help of Hue, which is a convenient tool since it gives a web user interface to programming languages. Hue requires the use of a web browser in order to be installed or configured.

[Get Full Contents from this link](#)

Chapter-5: Cloudera CDP and YARN

Overview

- Apache YARN is the processing layer or execution engine for managing distributed applications that run on multiple machines in a network.
- YARN is also known as MRv2.
- YARN supports MapReduce and legacy (MRv1) MapReduce jobs can run in YARN.
- YARN architecture splits the two primary responsibilities of the JobTracker into
 - Resource management and
 - Job scheduling/monitoring
- YARN will have two separate daemons for each of the above responsibility.
 - A global ResourceManager for resources management and
 - Per-application ApplicationMasters (Assume, each one of your Impala Query or Hive Query or a MapReduce Job is a separate application and for each YARN will create an Application Master).
- YARN enables the use of a variety of data processing engines for batch, interactive, and real-time stream processing of data stored in HDFS or cloud storage such as S3 and ADLS.
- You may run various processing frameworks for distinct use-cases on the same Hadoop cluster with the help of YARN, for example, Hive for SQL applications, Spark for in-memory applications, and Storm for streaming applications.
- YARN is part of Cloudera Runtime.

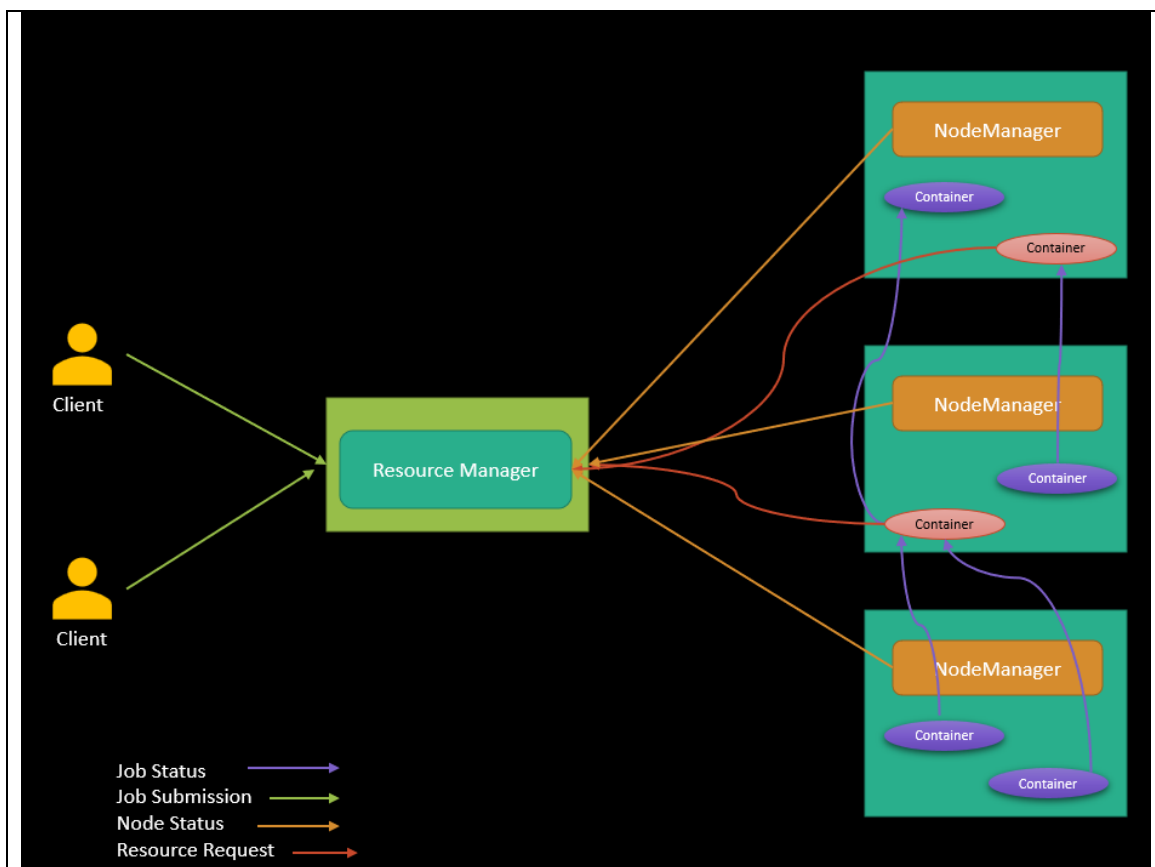
CDP Compute

- Apache YARN is resource management tool for CDP or Hadoop framework.

- Apache YARN manages resources for the applications running on your cluster by allocating resources through scheduling, limiting CPU usage, and partitioning clusters.
- You can use Access Control Lists to use YARN with a secure cluster.
- By using Apache YARN, you can optimize the use of vcores and memory.

YARN architecture and workflow

- YARN has three main components:
 - **ResourceManager:** Allocates cluster resources using a Scheduler and ApplicationManager.
 - **ApplicationMaster:** Manages the life-cycle of a job by directing the NodeManager to create or destroy a container for a job. There is only one ApplicationMaster for a job.
 - **NodeManager:** Manages jobs or workflow in a specific node by creating and destroying containers in a cluster node.



YARN Features

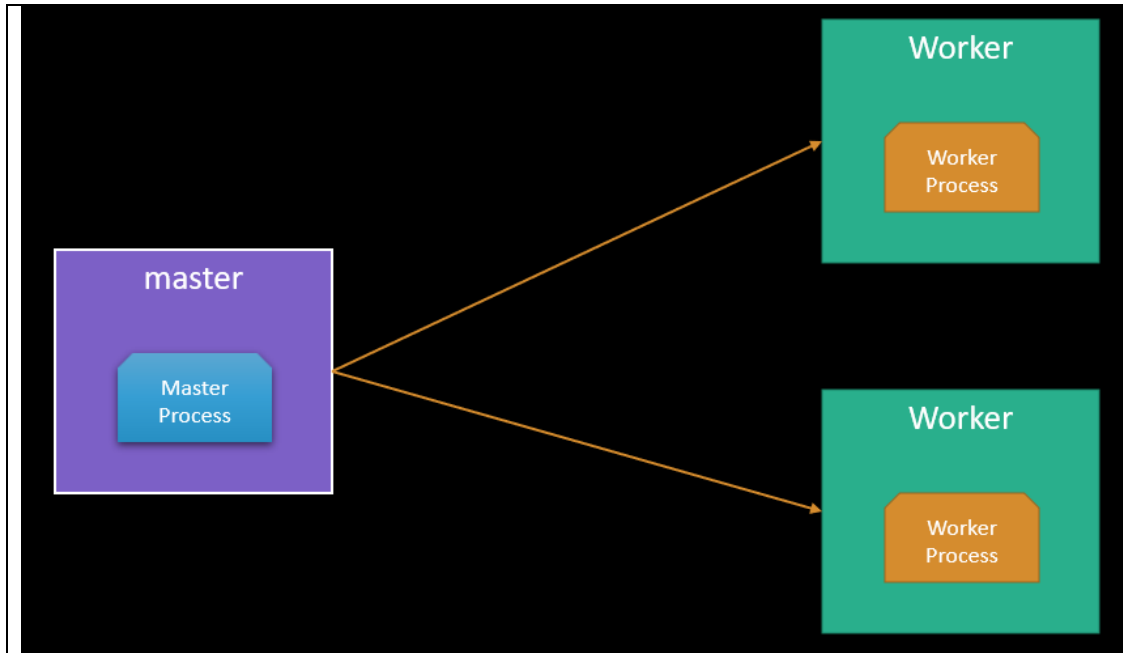
- YARN enables you to manage resources and schedule jobs in Hadoop and has the following features.
- **Multi-tenancy:**
 - You can use multiple open-source and proprietary data access engines for batch, interactive, and real-time access to the same dataset. Multi-tenant data processing improves an enterprise's return on its Hadoop investments.
 - YARN's dynamic resource management allows many engines and workloads to use the same cluster resources. Make your data available to users throughout your whole business environment via batch, interactive, sophisticated, or real-time

processing, all inside the same platform, to get the most out of your Hadoop platform.

- **Cluster utilization:**
 - o You can dynamically allocate cluster resources to improve resource utilization.
 - o Fine-grained settings improve cluster utilisation, allowing you to implement workload SLAs for priority workloads and group-based rules throughout the enterprise. Process more data in more ways while keeping your most vital tasks running smoothly.
- **Multiple resource types:** You can use multiple resource types such as memory, CPU, and GPU.
- **Scalability:**
 - o Significantly improved data center processing power. YARN's ResourceManager focuses exclusively on scheduling and keeps pace as clusters expand to thousands of nodes managing petabytes of data.
 - o YARN is intended to manage scheduling for Hadoop's vast scale, allowing you to add new and bigger workloads while staying on the same platform.
- **Compatibility:** MapReduce applications developed for Hadoop 1 runs on YARN without any disruption to existing processes. YARN maintains API compatibility with the previous stable release of Hadoop.

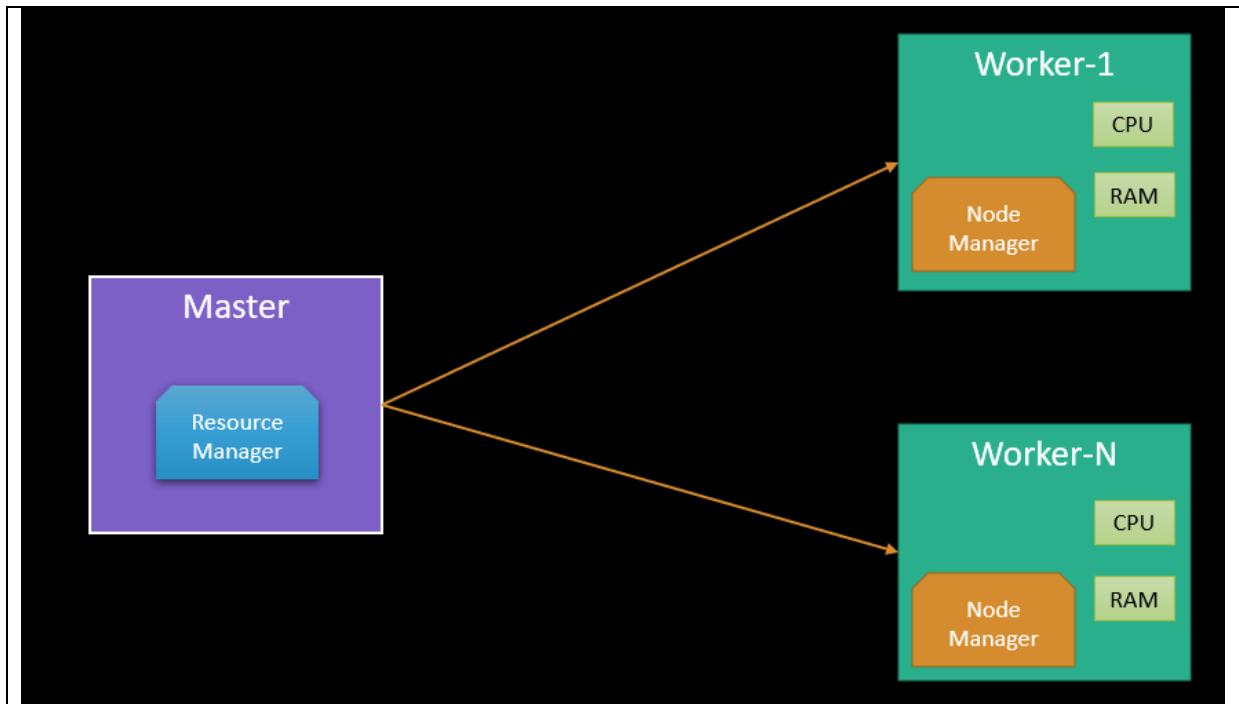
YARN and Cluster Basics (Master and Worker Nodes)

- A host, which is also called a node, in YARN terminology. A cluster is two or more hosts connected by a high-speed local network.
- In Hadoop, there are two types of hosts in the cluster.



- A master host serves as the point of communication for a client programme. The other computers in the cluster, known as worker hosts, get their assignments from the cluster's master host.
- In a YARN cluster, there are two types of hosts:
 - o **The ResourceManager** is the master daemon that communicates with the client, tracks resources on the cluster, and orchestrates work by assigning tasks to NodeManagers.

- **A NodeManager** is a worker daemon that launches and tracks processes spawned on worker hosts.

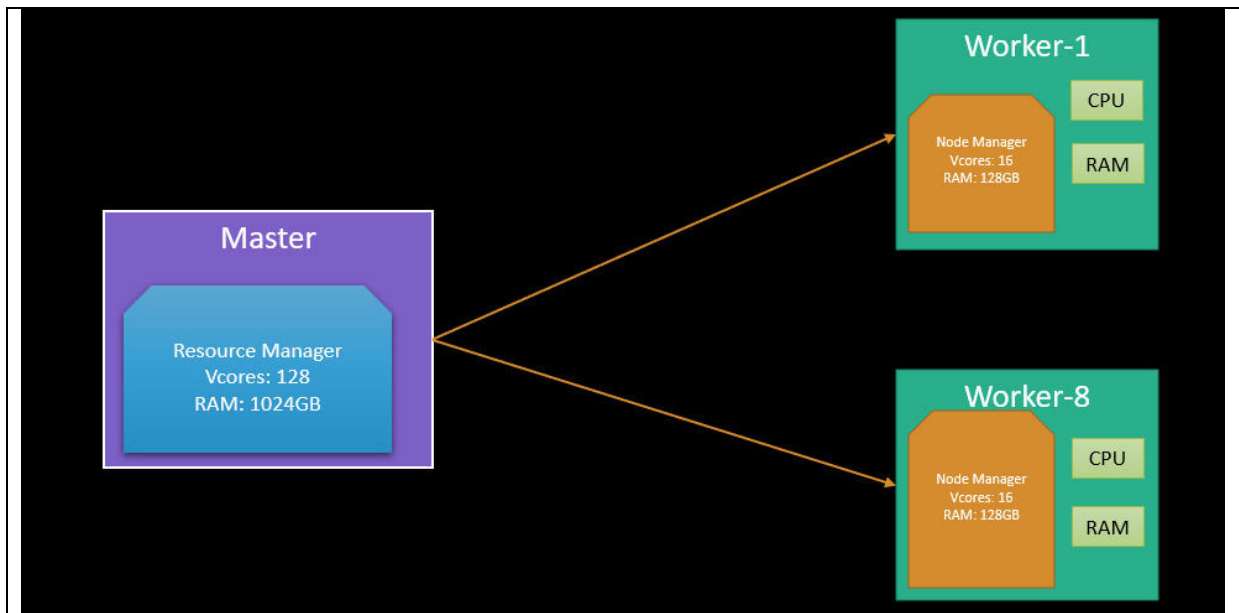


YARN Configuration File

- The YARN configuration file is an XML file that contains properties.
- This file is placed in a well-known location on each host in the cluster and is used to configure the ResourceManager and NodeManager.
- By default, this file is named yarn-site.xml.

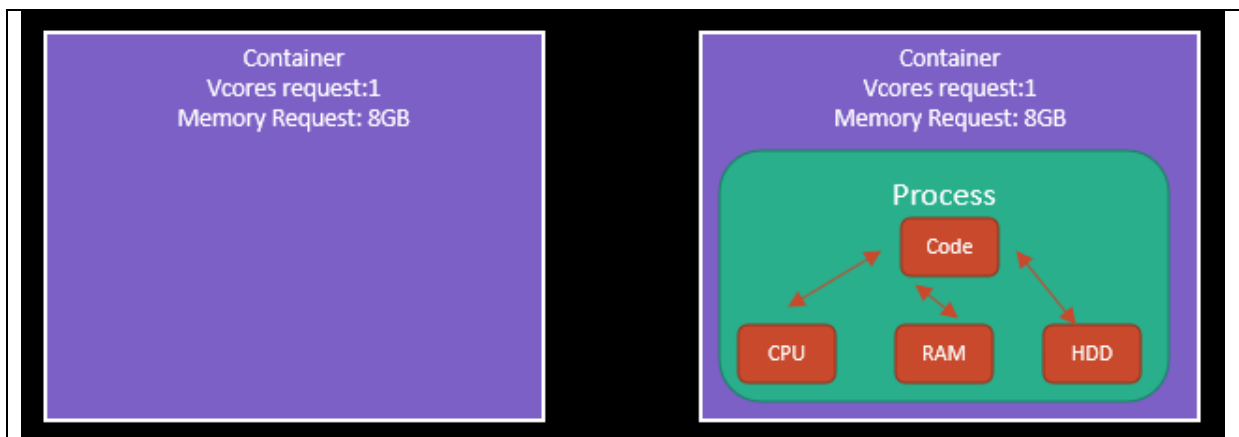
YARN Requires a Global View

- YARN currently defines two resources
 - vcores
 - and memory.
- Each NodeManager tracks its own local resources and communicates its resource configuration to the ResourceManager, which keeps a running total of the cluster's available resources.
- By keeping track of the total, the ResourceManager knows how to allocate resources as they are requested.
- Vcore has a special meaning in YARN. You can think of it simply as a "usage share of a CPU core."
- If you expect your tasks to be less CPU-intensive (sometimes called I/O-intensive), you can set the ratio of vcores to physical cores higher than 1 to maximize your use of hardware resources.)



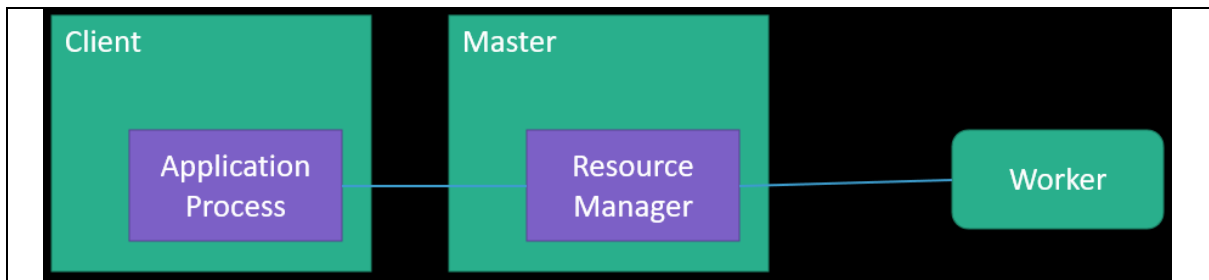
YARN Containers

- Containers are an important YARN concept.
- You can think of a container as a request to hold resources on the YARN cluster.
- Currently, a container hold request consists of vcore and memory, as shown in below
- Container as a hold (left), and container as a running process (right).
- Once a hold has been granted on a host, the NodeManager launches a process called a task.
- The right side of Figure shows the task running as a process inside a container.

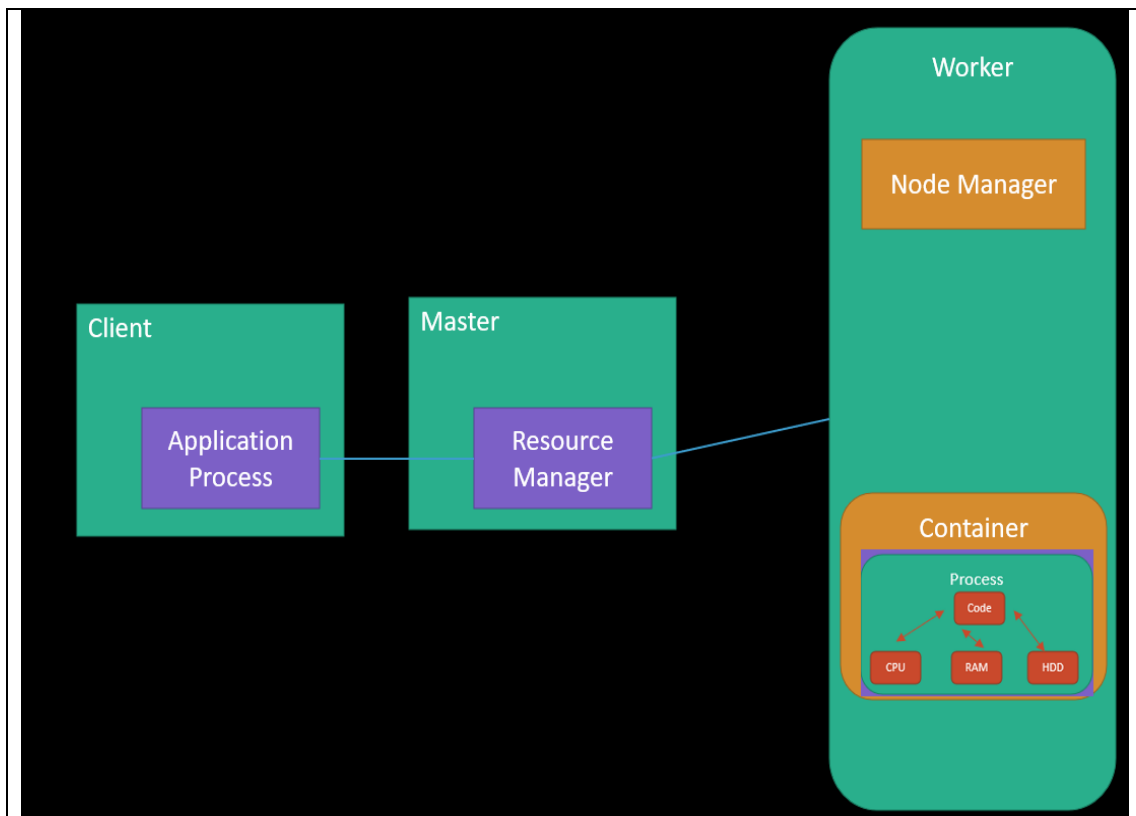


YARN Application Processing on Cluster

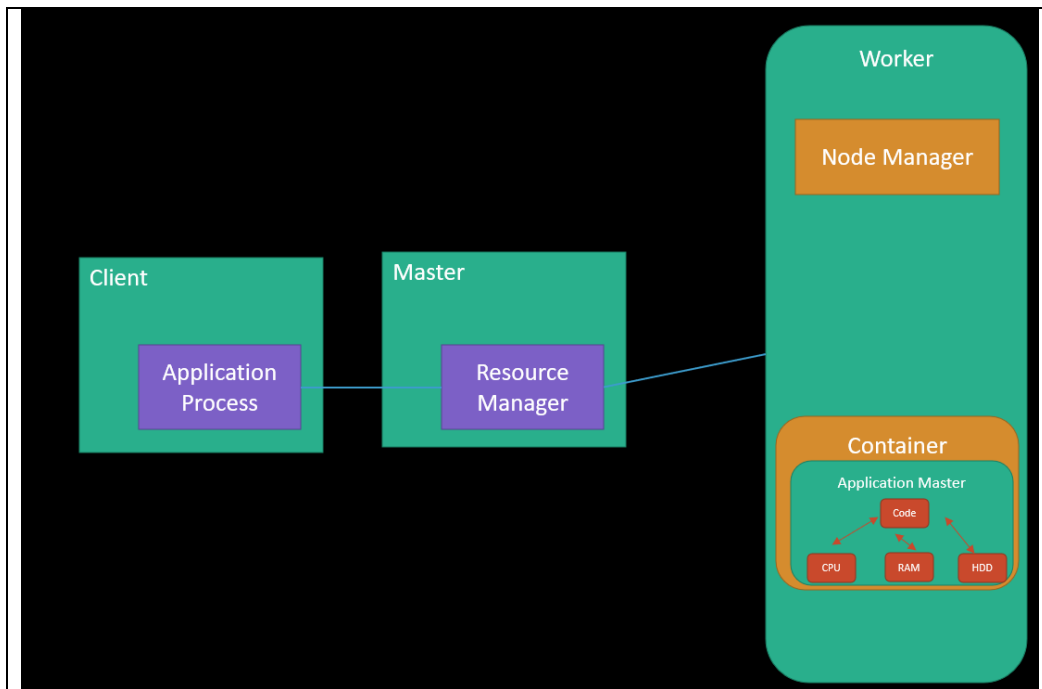
- An application is a YARN client program that is made up of one or more tasks.
- For each running application, a special piece of code called an ApplicationMaster helps coordinate tasks on the YARN cluster.
- The ApplicationMaster is the first process run after the application starts.
- An application running tasks on a YARN cluster consists of the following steps:
- **Step-1:** The application starts and talks to the ResourceManager for the cluster, Application starting up before tasks are assigned to the cluster



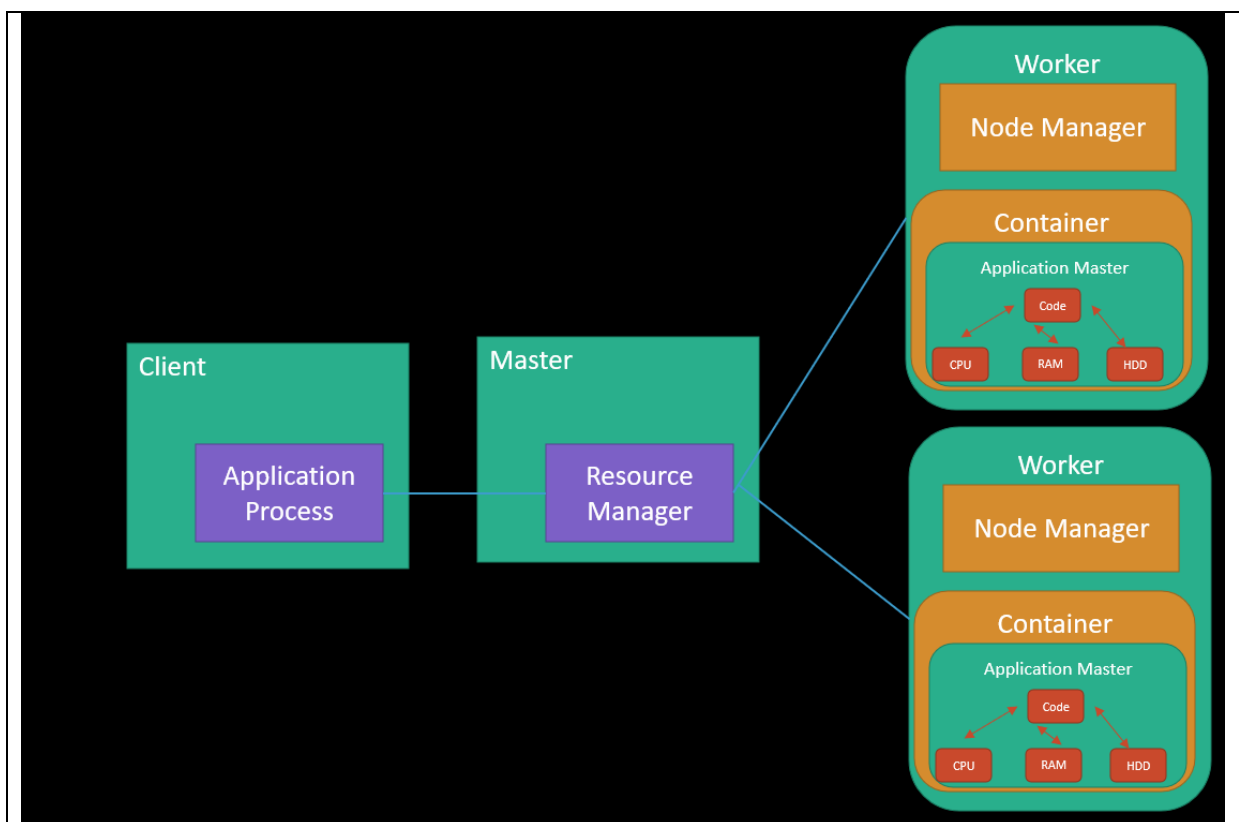
Step-2: The Resource Manager makes a single container request on behalf of the application and allocated container on a cluster.



Step-3: The ApplicationMaster starts running within that container, Application + ApplicationMaster running in the container on the cluster.



Step-4: The ApplicationMaster requests subsequent containers from the ResourceManager that are allocated to run tasks for the application. Those tasks do most of the status communication with the ApplicationMaster allocated in Step 3), as shown below Application + ApplicationMaster + task running in multiple containers running on the cluster.

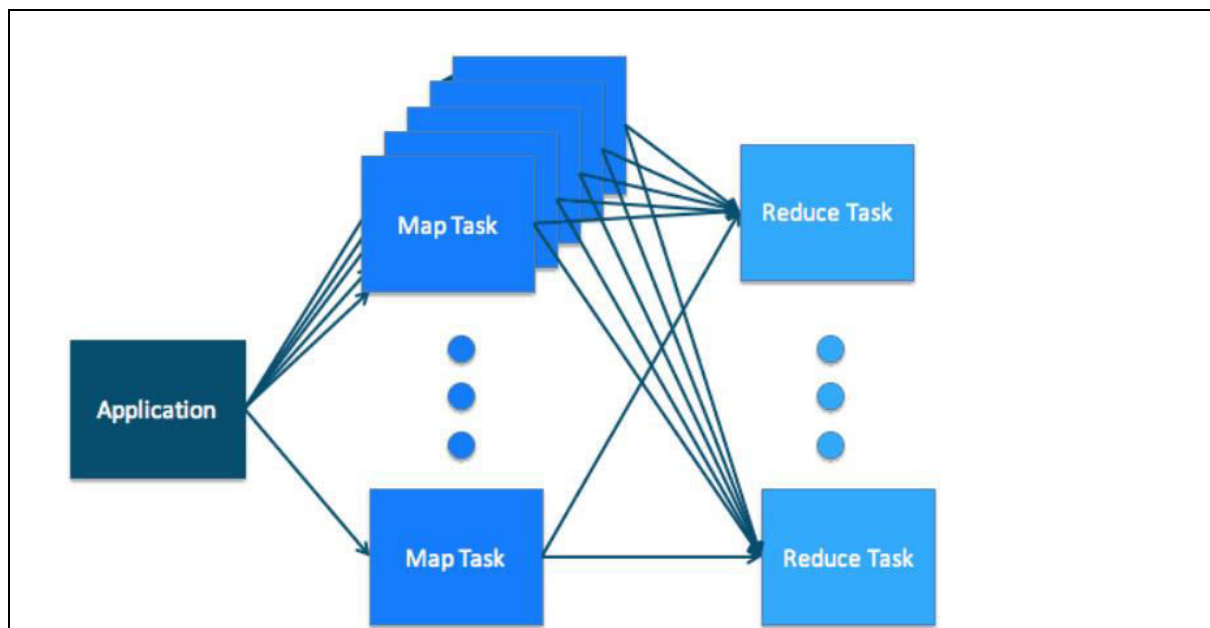


Step-5: Once all tasks are finished, the ApplicationMaster exits. The last container is de-allocated from the cluster.

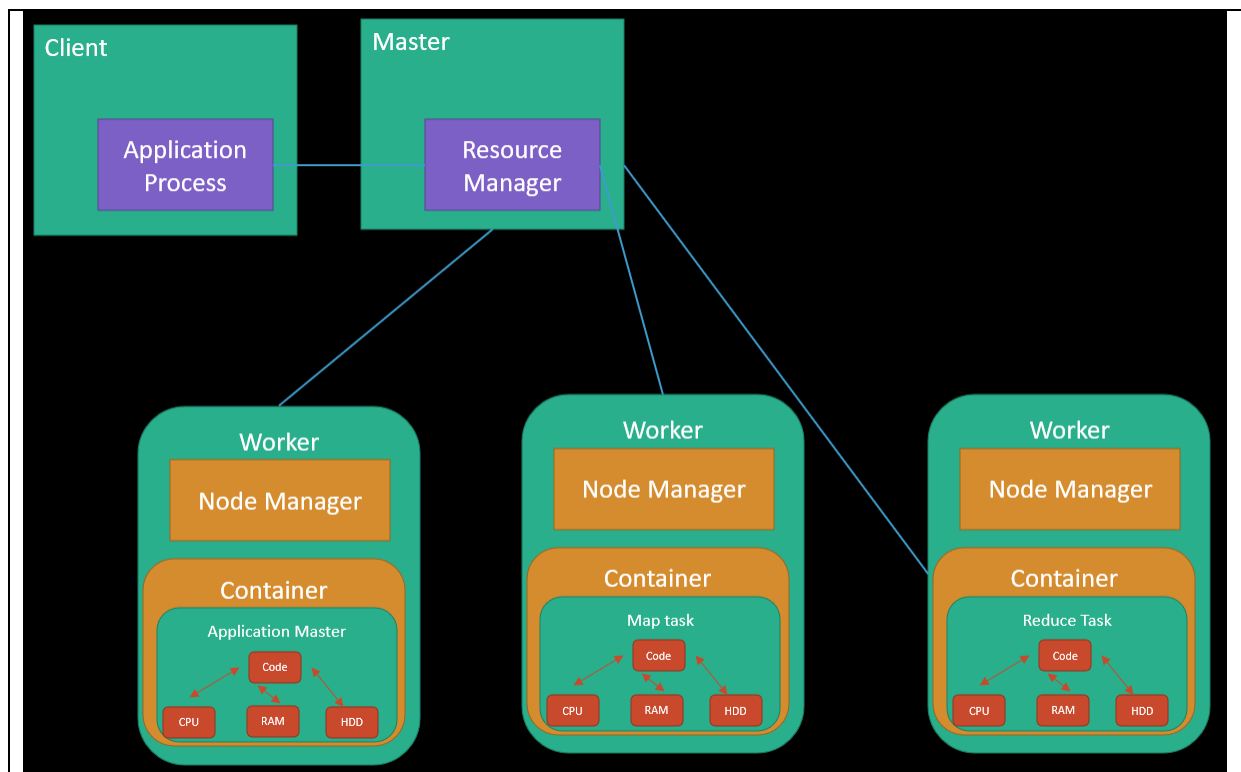
Step-6: The application client exits. (The ApplicationMaster launched in a container is more specifically called a managed AM. Unmanaged ApplicationMasters run outside of YARN's control. Llama is an example of an unmanaged AM.)

MapReduce Fundamental Concepts

- In the MapReduce paradigm, an application consists of Map tasks and Reduce tasks. Map tasks and Reduce tasks align very cleanly with YARN tasks.



- Below image, illustrates how the map tasks and the reduce tasks map cleanly to the YARN concept of tasks running in a cluster.

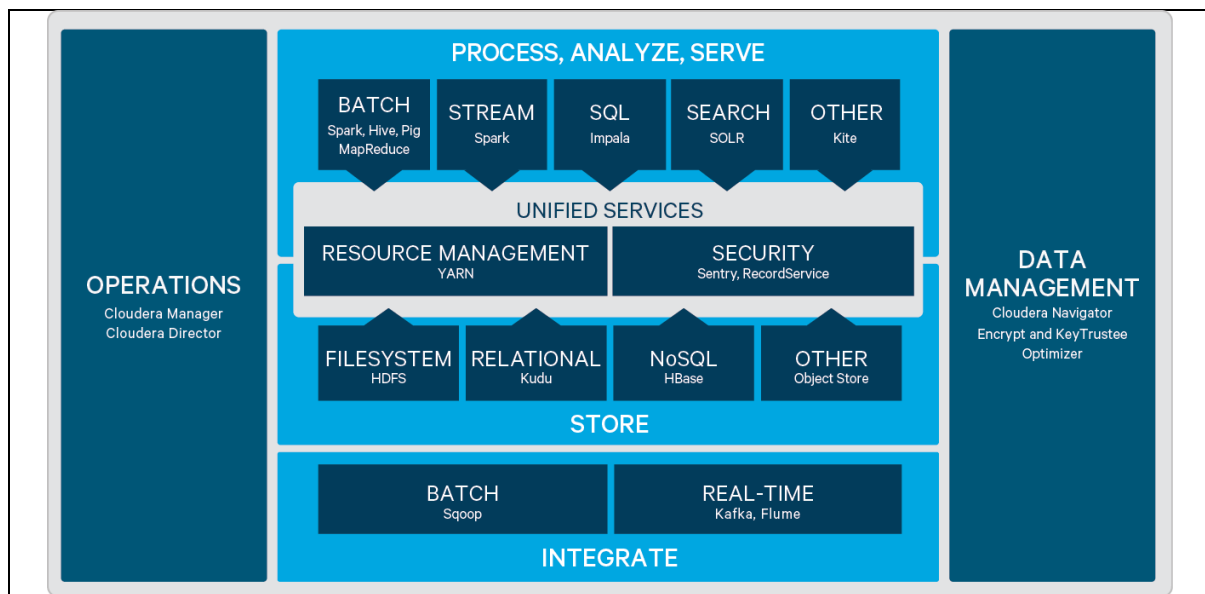


- In a MapReduce application, there are multiple map tasks, each running in a container on a worker host somewhere in the cluster. Similarly, there are multiple reduce tasks, also each running in a container on a worker host.
- Simultaneously on the YARN side, the ResourceManager, NodeManager, and ApplicationMaster work together to manage the cluster's resources and ensure that the tasks, as well as the corresponding application, finish cleanly.

YARN Integrated across the CDP platform

- A CDP cluster is made up of two or more hosts connected by an internal high-speed network.
- Master hosts are a small number of hosts reserved to control the rest of the cluster. Worker hosts are the non-master hosts in the cluster.
- In a cluster with YARN running, the master process is called the ResourceManager and the worker processes are called NodeManagers.
- The configuration file for YARN is named yarn-site.xml. There is a copy on each host in the cluster. It is required by the ResourceManager and NodeManager to run properly. YARN keeps track of two resources on the cluster, vcores and memory. The NodeManager on each host keeps track of the local host's resources, and the ResourceManager keeps track of the cluster's total.
- A container in YARN holds resources on the cluster. YARN determines where there is room on a host in the cluster for the size of the hold for the container. Once the container is allocated, those resources are usable by the container.
- An application in YARN comprises three parts:
 - The application client, which is how a program is run on the cluster.
 - An ApplicationMaster which provides YARN with the ability to perform allocation on behalf of the application.
 - One or more tasks that do the actual work (runs in a process) in the container allocated by YARN.

- A MapReduce application consists of map tasks and reduces tasks.
- A MapReduce application running in a YARN cluster looks very much like the MapReduce application paradigm, but with the addition of an ApplicationMaster as a YARN requirement.
- Cloudera's platform is built on core Hadoop, which includes HDFS, MapReduce, and YARN.
- All platform components have access to the same HDFS data and take part in shared resource management through YARN.
- Hadoop, as part of Cloudera's platform, also benefits from straightforward deployment and administration (through Cloudera Manager) as well as shared compliance-ready security and governance.



YARN Scheduler

- A scheduler determines which jobs run, where and when they run, and the resources allocated to the jobs.
- The YARN (MRv2) and MapReduce (MRv1) computation frameworks support the following schedulers:
 - o **FIFO** - Allocates resources based on arrival time.
 - o **Fair** - Allocates resources to weighted pools, with fair sharing within each pool. When configuring the scheduling policy of a pool, Domain Resource Fairness (DRF) is a type of fair scheduler.
 - o **Capacity** - Allocates resources to pools, with FIFO scheduling within each pool.
- However, Cloudera CDP only supports the Capacity Scheduler.

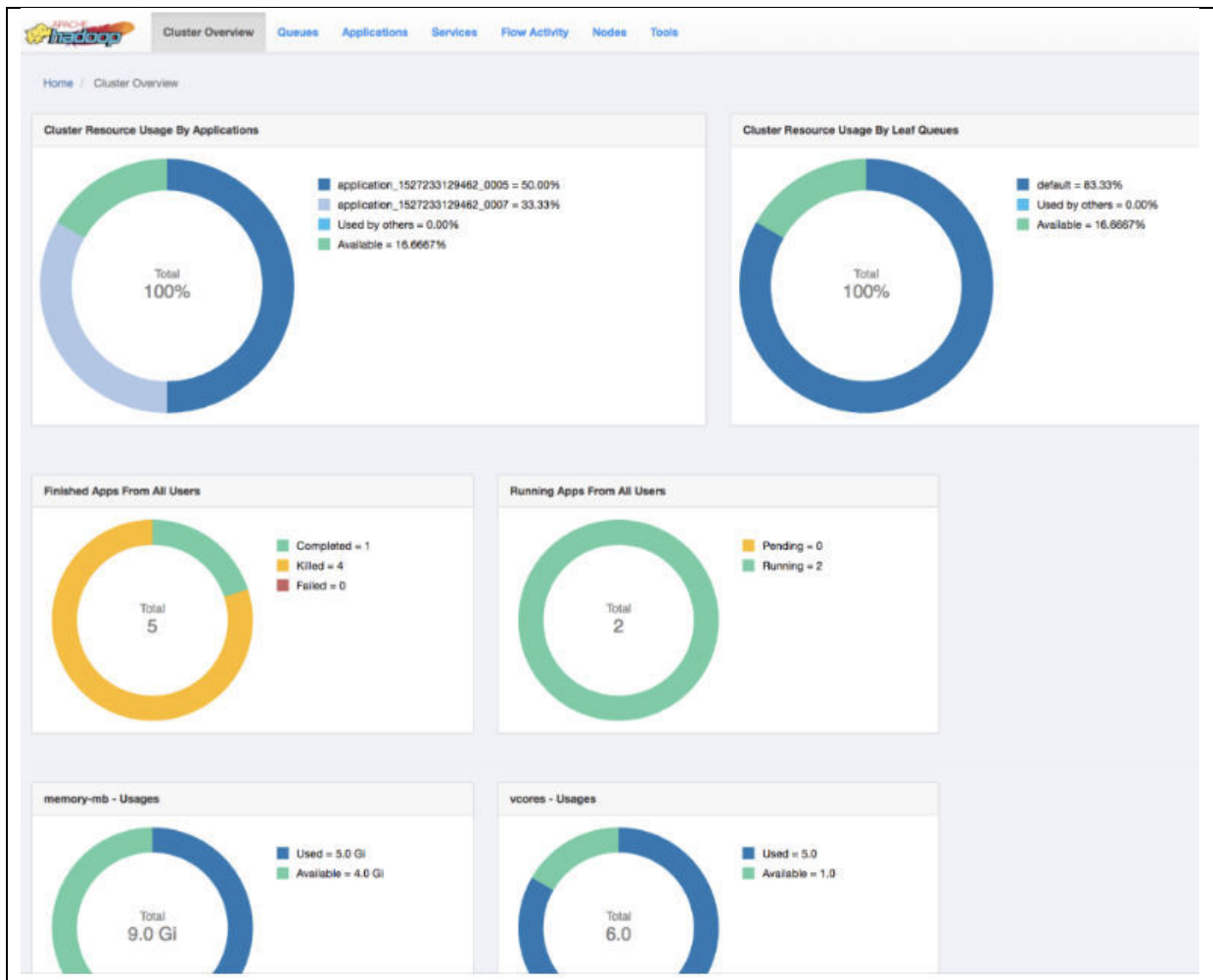
YARN Capacity Scheduler Overview

- The CapacityScheduler is designed to run Hadoop applications as a shared, multi-tenant cluster in an operator-friendly manner while maximizing the throughput and the utilization of the cluster.
- Traditionally each organization has its own private set of compute resources that have sufficient capacity to meet the organization's SLA under peak or near-peak conditions.

- This generally leads to poor average utilization and overhead of managing multiple independent clusters, one per each organization.
 - Sharing clusters between organizations is a cost-effective manner of running large Hadoop installations since this allows them to reap benefits of economies of scale without creating private clusters. However, organizations are concerned about sharing a cluster because they are worried about others using the resources that are critical for their SLAs.
 - The CapacityScheduler is designed to allow sharing a large cluster while giving each organization capacity guarantees.
 - The central idea is that the available resources in the Hadoop cluster are shared among multiple organizations who collectively fund the cluster based on their computing needs.
 - There is an added benefit that an organization can access any excess capacity not being used by others. This provides elasticity for the organizations in a cost-effective manner.
-
- Sharing clusters across organizations necessitates strong support for multi-tenancy since each organization must be guaranteed capacity and safe-guards to ensure the shared cluster is impervious to single rogue application or user or sets thereof.
 - The CapacityScheduler provides a stringent set of limits to ensure that a single application or user or queue cannot consume disproportionate number of resources in the cluster. Also, the CapacityScheduler provides limits on initialized and pending applications from a single user and queue to ensure fairness and stability of the cluster.
 - The primary abstraction provided by the CapacityScheduler is the concept of queues. These queues are typically setup by administrators to reflect the economics of the shared cluster.
 - To provide further control and predictability on sharing of resources, the CapacityScheduler supports hierarchical queues to ensure resources are shared among the sub-queues of an organization before other queues are allowed to use free resources, thereby providing affinity for sharing free resources among applications of a given organization.

YARN Web User Interface

- You can use YARN Web interface to monitor clusters, queues, applications, services, and flow activities.
- **Cluster Overview:**
 - o When you open a Cluster Overview page, it should look like as below.



- And provides the below information
- **Cluster Resource Usage by Applications:**
 - o Displays the percentage of cluster resources in use by applications and the percentage available for usage.
- **Cluster Resource Usage by Leaf Queues:**
 - o Displays the percentage of cluster resources in use by leaf queues and the percentage available for usage.
- **Finished Apps From All Users:**
 - o Displays the number of completed, killed, and failed applications.
- **Monitor Running Apps:**
 - o Displays the number of pending and running applications.
- **memory-mb – Usages:**
 - o Displays the amount of used and available memory.
- **vcores – Usages:**
 - o Displays the number of used and available virtual cores.
- **Monitor Node Managers:**
 - o Displays the status of the Node Managers under the following categories:
 - Active
 - Unhealthy
 - Decommissioning

- Decommissioned

Resource Scheduling and Management

- You can manage resources for the applications running on your cluster by allocating resources through scheduling, limiting CPU usage by configuring cgroups, and partitioning the cluster into subclusters using node labels, and launching applications on Docker containers.
- The CapacityScheduler is responsible for scheduling. The CapacityScheduler is used to run Hadoop applications as a shared, multi-tenant cluster in an operator-friendly manner while maximizing the throughput and the utilization of the cluster.
- The ResourceCalculator is part of the YARN CapacityScheduler. If you have only one type of resource, typically a CPU virtual core (vcore), use the DefaultResourceCalculator. If you have multiple resource types, use the DominantResourceCalculator.
- **YARN resource allocation of multiple resource-types:** You can manage your cluster capacity using the Capacity Scheduler in YARN. You can use the Capacity Scheduler's DefaultResourceCalculator or the DominantResourceCalculator to allocate available resources.
- **Hierarchical queue characteristics:** You must consider the various characteristics of the Capacity Scheduler hierarchical queues before setting them up.
- **Scheduling among queues:** Hierarchical queues ensure that guaranteed resources are first shared among the sub-queues of an organization before any remaining free resources are shared with queues belonging to other organizations. This enables each organization to have control over the utilization of its guaranteed resources.
- **Application reservations:** For a resource-intensive application, the Capacity Scheduler creates a reservation on a cluster node if the node's free capacity can meet the particular application's requirements. This ensures that the resources are utilized only by that particular application until the application reservation is fulfilled.
- **Resource distribution workflow:** During scheduling, queues at any level in the hierarchy are sorted in the order of their current used capacity, and the available resources are distributed among them starting with queues that are currently the most under-served.
- **Use CPU scheduling:** Cgroups with CPU scheduling helps you effectively manage mixed workloads.
- **Use GPU scheduling:** On your cluster, you can configure GPU scheduling and isolation. Currently only Nvidia GPUs are supported in YARN. You can use Cloudera Manager to configure GPU scheduling on your cluster.
- **Use FPGA scheduling:** You can use FPGA as a resource type.
- **Limit CPU usage with Cgroups:** You can use cgroups to limit CPU usage in a Hadoop Cluster.
- **Partition a cluster using node labels:** You can use Node labels to partition a cluster into sub-clusters so that jobs run on nodes with specific characteristics.

Chapter-6: Apache Spark

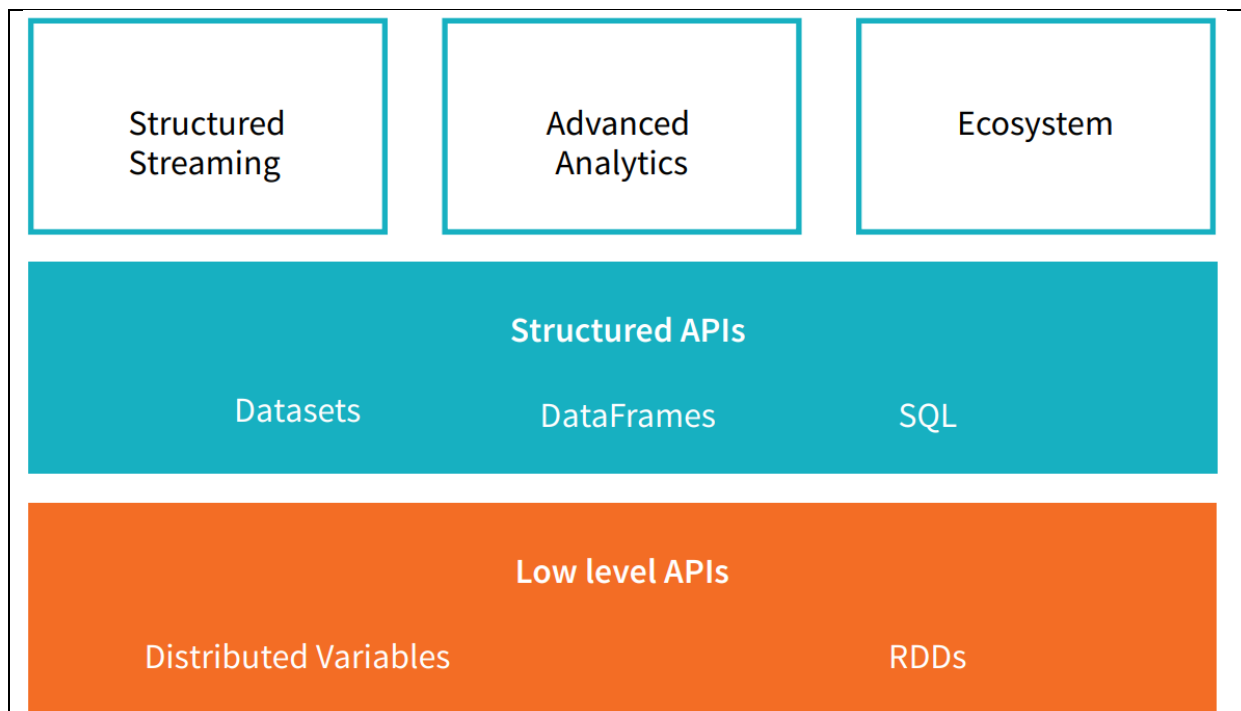
Apache Spark is a broad framework for distributed computing that delivers excellent performance for both batch and interactive processing. It comprises of the Spark core as well as numerous projects that are connected to it and offers application programming interfaces (APIs) for Java, Python, and Scala.

You have the option of using an interactive shell or submitting an application to execute Spark applications locally or distributed over a cluster. Both of these options are available to you. During the data exploration phase as well as for ad hoc analyses, it is usual practise to run Spark applications in an interactive mode.

Spark needs a cluster manager in order to successfully execute applications that are deployed over a cluster. The YARN cluster manager is the only one that Cloudera supports. The YARN ResourceManager and NodeManager roles are responsible for managing Spark application processes while they are being executed on YARN.

Apache Spark is a unified computing engine and a suite of libraries for parallel data processing on computer clusters. Spark has become the tool of choice for any software developer or data scientist that is interested in big data. Spark is capable of running on anything from a laptop to a cluster of thousands of servers, and it offers support for a number of widely used programming languages, including Python, Java, Scala, and R. It also includes libraries that can be used for a variety of tasks, including SQL, streaming, and machine learning. Because of this, it is a simple system to begin with and can easily be expanded to handle massive amounts of data or an extremely vast scale.

An easy-to-understand example of all that Spark has to offer an end user is shown here.



Python, Java, Scala, R, and SQL can utilise Spark. Spark is built in Scala and runs on the Java Virtual Machine (JVM). To run Spark on a laptop or cluster, you require Java 6 or newer. You'll need a Python interpreter to access the API (version 2.6 or newer). You'll need R if you want to utilise it.

Fundamentals of Apache Spark

Unified:

Spark aims to provide a single platform for creating large data applications. Unified means... Spark supports a broad variety of data analytics operations, from basic data loading and SQL queries to machine learning and streaming computation, using the same computational engine and APIs. Real-world data analytics jobs, whether interactive analytics in a Jupyter notebook or conventional software development, integrate several processing kinds and libraries. Spark's cohesive nature makes writing simpler and faster. Spark offers consistent, composable APIs that may be used to create an application from smaller components or existing libraries, and makes it simple to develop your own analytics libraries on top.

Composable APIs aren't enough. Spark's APIs are meant to optimise user programmes' libraries and functions for maximum performance. If you load data using SQL and subsequently analyse a machine learning model using Spark's ML library, the engine may merge both stages into one scan. Spark is a great platform for interactive and production applications because to its broad APIs and high-performance execution.

Spark's emphasis on a single platform mirrors previous software unified platform.

Data scientists use uniform libraries (e.g., Python or R) when modelling, while web developers use united frameworks like Node.js or Django. Before Spark, no open-source system provided a single engine for parallel data processing, therefore users had to piece together an application from disparate APIs and platforms. Spark rapidly became the industry standard.

Spark's built-in APIs have grown to support new workloads. Developers have also refined the project's unifying engine. This book will concentrate on Spark 2.0's "structured APIs" (DataFrames, Datasets, and SQL) to optimise user applications.

Computing Engine:

Spark is a computational engine that aims towards unification. Spark merely loads data from storage systems and performs computations, not permanent storage. Spark may be utilised with Azure Storage, Amazon S3, Apache Hadoop, Apache Cassandra, and Apache Kafka. Spark neither saves nor prioritises long-term data. Most data is already stored in several systems. Spark performs calculations on data wherever it exists since moving it is costly. Spark tries to make user-facing APIs seem comparable so apps don't have to worry about where their data resides. Spark focuses on computing, unlike Apache Hadoop.

Hadoop's storage technology (the Hadoop file system) and computation system (MapReduce) were tightly interwoven. This option makes it difficult to operate one system without the other or create apps that access data elsewhere. Spark operates well on Hadoop storage, but it's also utilised in contexts where Hadoop design doesn't make sense, such as the public cloud (where storage may be rented separately from compute) or streaming applications.

Libraries:

Spark's libraries leverage on its unified engine architecture to offer a single API for data analysis operations. Spark supports both engine-bundled libraries and third-party open source libraries. Today, Spark's standard libraries form the majority of the open source project; the Spark core engine has evolved little since its inception, but the libraries have developed to give greater capability.

Spark comprises SQL and structured data (Spark SQL), machine learning (MLlib), stream processing (Spark Streaming and Structured Streaming), and graph analytics (GraphX). There are hundreds of free source external libraries, from storage interfaces to machine learning techniques. spark-packages.org lists external libraries.

Spark Architecture

When you think of a "computer," you probably picture a desktop device. This computer is great for movies and spreadsheets. As many users know, your computer can't do everything. Data processing is difficult. Single machines can't process large volumes of data (or the user may not have time to wait for the computation to finish).

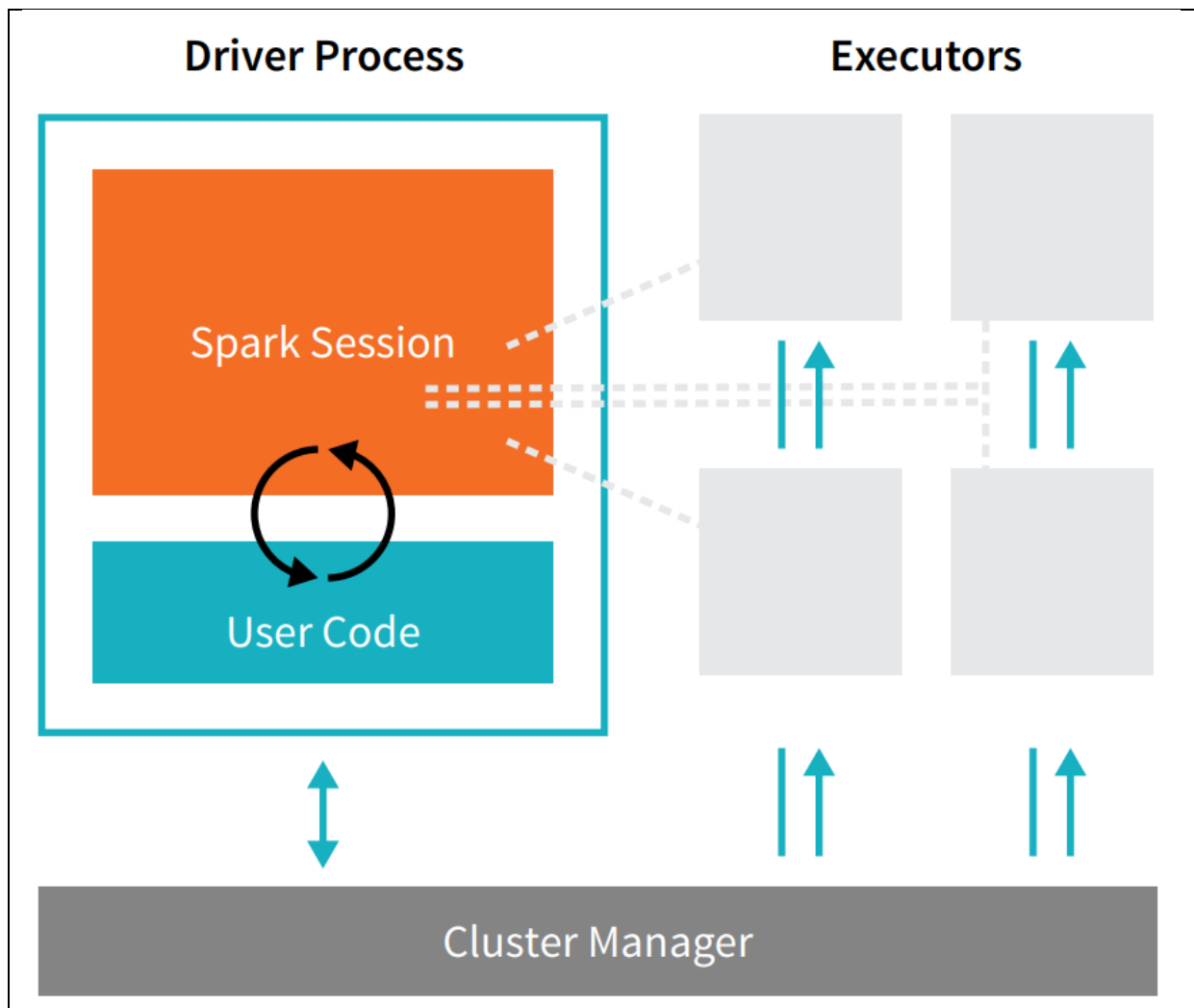
A cluster combines the resources of multiple computers to utilise them as one. A set of machines isn't powerful without a framework to coordinate work. Spark manages and coordinates data tasks across a cluster of computers.

Spark's cluster of computers will be controlled by Spark's Standalone cluster manager, YARN, or Mesos. The cluster administrators then provide Spark applications resources so we may finish our job.

Spark Applications

The components that make up a Spark Application are referred to as the "driver process" and the "executor processes." The driver process is responsible for three things: keeping information about the Spark Application; reacting to a user's programme or input; and analysing, distributing, and scheduling work among the executors. It resides on a node in the cluster and performs your main() function (defined momentarily). The driver process is critically necessary since it is the core of a Spark Application and it is responsible for keeping all of the important information updated during the application's lifetime.

The executors are the ones who are accountable for carrying out the task that has been delegated to them by the driver. This indicates that each executor is solely responsible for two things: running the code that has been given to it by the driver, and reporting back to the driver node the current status of the computation that is taking place on that executor.



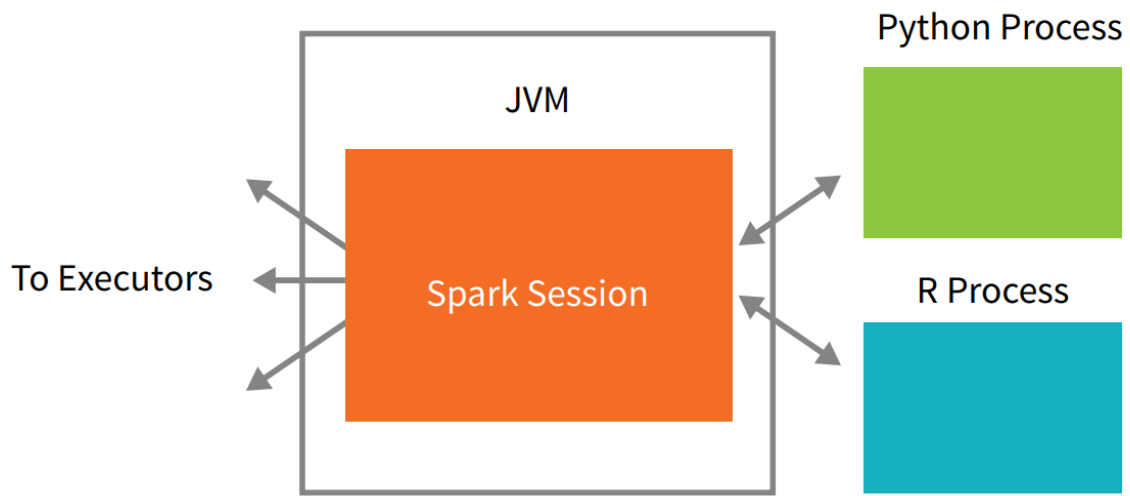
The cluster manager is responsible for managing the underlying hardware and allocating resources to Spark applications. This might be Spark's standalone cluster manager, YARN, or Mesos. Alternatively, it could be one of numerous other fundamental cluster managers. This indicates that a cluster is capable of simultaneously hosting many instances of a Spark application running at the same time. In the next section of this book, titled "Part IV: Production Applications," we will have a more in-depth discussion on cluster managers.

From the preceding example, we can see that our driver is located on the left, while the four executors are located on the right. In this particular design, the idea of cluster nodes has been eliminated. Through the use of settings, the user has the ability to determine how many executors should land on each node.

Spark may operate in both a cluster and a local mode. The cluster mode is the default. Because both the driver and the executors are just processes, it is possible for them to coexist on the same system or on entirely distinct computers. When operating in the local mode, both of them execute (as threads) on your own machine rather than in a cluster. We prepared this book keeping in mind local mode, which means that everything should be able to be executed on a single system.

Spark may operate in both a cluster and a local mode. The cluster mode is the default. Because both the driver and the executors are just processes, it is possible for them to coexist on the same system or on entirely distinct computers. When operating in the local mode, both of them execute (as

threads) on your own machine rather than in a cluster. We prepared this book keeping in mind local mode, which means that everything should be able to be executed on a single system.



Each language's application programming interface (API) will adhere closely to the key notions that we outlined before. Spark code will be executed via the user's `SparkSession`, which is made accessible to the user. The `SparkSession` will serve as the entry point. When using Spark from within Python or R, the user never writes explicit instructions for the JVM; rather, the user writes code in Python and R that Spark will translate into code that Spark can then run on the executor JVMs. This is the case even when the user is using Spark from within Python or R.

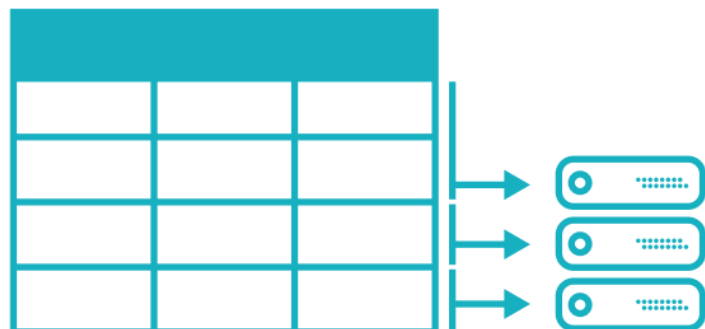
DataFrames

A `DataFrame`, which is the most popular kind of Structured API, is essentially a representation of a data table that has rows and columns. The schema consists of the list of columns as well as the types included inside those columns. A straightforward comparison would be a spreadsheet that has labelled columns. A spreadsheet is stored on a single computer at a single place, but a Spark `DataFrame` may be distributed over thousands of machines. This is the primary distinction between the two. It should be obvious why the data are being stored on several computers: either the data are too massive to fit on a single system or the calculation would simply take too long to complete on a single machine.

Spreadsheet on a single machine



Table or DataFrame partitioned across servers in a data center



The idea of a DataFrame is not one that is exclusive to Spark. R and Python have a number of conceptual similarities. On the other hand, Python/R DataFrames, with a few notable exceptions, are stored on a single system as opposed to numerous workstations. This restricts what you are able to do in Python and R with a particular DataFrame to the resources that are available on that particular system. On the other hand, due to the fact that Spark includes language interfaces for both Python and R, converting Pandas (Python) DataFrames to Spark DataFrames and R DataFrames to Spark DataFrames is a rather simple process (in R).

In addition to Datasets and Dataframes, Spark now supports SQL Tables and Resilient Distributed Datasets as fundamental abstractions (RDDs). Despite the fact that each of these abstractions represents a distinct dispersed collection of data, they nonetheless have unique interfaces for interacting with that data. DataFrames are the most user-friendly and productive option; moreover, they are accessible in every language.

Partitions

Spark divides the data into sections that it calls partitions in order to make it possible for all of the executors to carry out their tasks simultaneously. A partition in our cluster is a set of rows that are hosted on one of our physical machines. The divisions of a DataFrame each represent a different way.

In the course of the execution, the data will be physically dispersed among all of your cluster's devices. Spark will work if you just have one partition. even if you have thousands of executors, you will only have a parallelism of one for the whole process. In the event that you have a single partition while having numerous Because there is just one computing resource, executor Spark will still only have a parallelism of one.

When working with DataFrames, one crucial point to keep in mind is that, for the most part, we do not actively change the divisions of the data. (with regard to each person) In the physical partitions, we only provide the high-level changes of the data, and Spark takes care of the rest. Decides how the task will be carried out in its entirety throughout the cluster. Lower level application programming interfaces are available (through the Resilient Distributed Object interface for datasets), and we go through them in Part III of this book.

Transformations

The fundamental data structures in Spark are immutable, which means that they cannot be altered once they have been generated. If you are unable to modify it in any way, what are you going to do with it then? This is something that may seem unusual at first. In order to "alter" a DataFrame, you will need to inform Spark on the manner in which you would want to transform the DataFrame that you now have into the DataFrame that you desire.

The name for these kinds of instructions is "transformations." Let's carry out a straightforward transformation to discover all even integers included inside the present DataFrame.

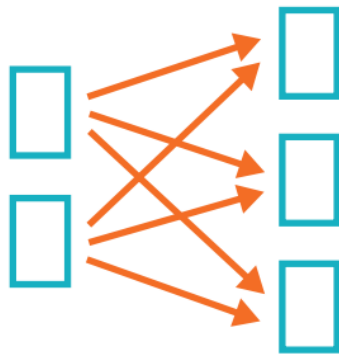
Wide Dependency:

A transformation that follows the broad dependence (or wide transformation) style will have numerous input partitions leading to the production of many output partitions. This process, in which Spark will trade partitions throughout the cluster, is often known to as a shuffle, and you will frequently hear it referred to as such. When narrow transformations are used, Spark will automatically carry out an action known as pipelining on narrow dependencies. This implies that if we define many filters on DataFrames, they will all be carried out in-memory if we use narrow

transformations. One cannot make the same statement about shuffles. When we carry out a shuffle, Spark will save the outcomes of the operation on disc. Because shuffle optimization is such an important issue, you'll likely come across many discussions about it on the internet, but for the time being, all you really need to know is that there are two different types of transformations.

Wide Transformations (shuffles)

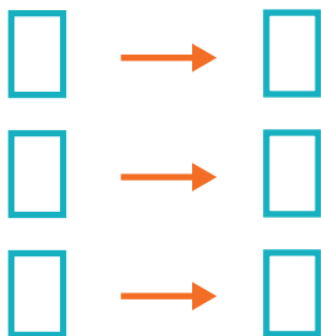
1 to N



Narrow Dependency: Narrow transformations, also known as transformations with narrow dependencies, are ones in which each input partition will only contribute to a single output partition. We'll refer to these transformations as narrow. Our where statement in the previous piece of code establishes a restricted dependence, meaning that just one partition contributes to the maximum of one output partition.

Narrow Transformations

1 to 1



Lazy Evaluation

A graph of processing instructions will not be carried out by Spark until the very last possible second because it uses a technique known as "lazy evaluation." When we do an operation in Spark, rather than instantly affecting the data, we develop a plan of transformations that we would want to apply to our source data. This plan is then executed on our source data. Spark, by delaying the execution

of the code until the very last possible moment, will convert this plan from your raw DataFrame transforms into a physically efficient plan that will run as efficiently as possible throughout the cluster. Because of this, the end user may reap enormous advantages as a result of Spark's ability to optimise the whole data flow from beginning to finish. On DataFrames, there is a feature known as "predicate pushdown" that serves as an illustration of this concept. If we design a huge Spark job but provide a filter at the end that only needs us to get one row from our source data, the most effective approach to carry this out is to access the single record that we needed. This is because the filter only requires us to fetch one row. Spark will actually assist us in optimising this situation by lowering the filter on its own own.

Actions

We are able to construct a more rational transformation plan as a result of transformations. We need to do some action in order to start the calculation. Spark is given the instruction to calculate a result based on a sequence of transformations through an action. The count action is the simplest one, and it provides us with the total number of records included in the DataFrame.

```
divisBy2.count()
```

Now we can see the outcome! It should come as no surprise that there are 500 numbers between 0 and 999 that are divisible by two. Now, counting isn't the only thing you can do. The following are the three categories of actions:

actions to collect data to native objects in the appropriate language; actions to see data in the console; actions to write to output data sources; and actions to view data in the console.

When we were specifying our action, we started a Spark job that runs our filter transformation, which is a narrow transformation. This is followed by an aggregation, which is a wide transformation, that performs the counts on a per-partition basis. Finally, a collect with brings our result to a native object in the appropriate language. All of this is viewable by checking the Spark UI, which is a tool that is included with Spark and gives us the ability to monitor the Spark tasks that are currently executing on a cluster.

DataFrames in addition to SQL

In the last example, we went through the steps of a straightforward example; now, let's go through the steps of a more complicated example while following along in both DataFrames and SQL. Spark the same modifications in precisely the same manner regardless of the language that is being used. Before Spark actually executes your code, you have the option of expressing your business logic in SQL or DataFrames (written in either R, Python, Scala, or Java), and Spark will compile that logic down to an underlying plan (which can be seen in the explain plan). You, as a user of Spark SQL, have the ability to register any DataFrame as a table or view (a temporary table), and then query that DataFrame with standard SQL. There is no difference in performance between writing SQL queries and writing DataFrame code since they both "compile" to the same underlying plan that we define in DataFrame code. This is the case because DataFrame code is compiled.

With one easy method call, a DataFrame may be transformed into a table or view of your choosing.

Chapter-7: Apache Impala

Overview

On data that is stored in common Apache Hadoop file formats, the SQL queries that are provided by the Apache Impala project provide great speed and low latency. Instead of the lengthy batch processes that are typically associated with SQL-on-Hadoop technologies, which are made possible by the slow response times for queries, interactive exploration and fine-tuning of analytic queries are now possible. (You will often come across the word "interactive" being used to different sorts of quick queries with response times on a human-scale.)

Both Apache Hive and Impala are able to exchange their databases and tables because to Impala's integration with the Apache Hive metastore database. Because of Impala and Hive's high degree of interaction with one another and their compliance with the HiveQL syntax, you can use either Impala or Hive to build tables, run queries, load data, and do other similar tasks.

The following is a list of some of the primary benefits offered by the Impala:

- Because Impala is integrated with the preexisting CDH ecosystem, it is possible to store, distribute, and retrieve data by using the many solutions that are included with CDH. This prevents the creation of data silos and lowers the need for costly data transfer.
- Impala allows users to have access to data that is stored in CDH without having them to have the Java expertise that is necessary for MapReduce tasks. The HDFS file system is immediately accessible to Impala, allowing for direct data retrieval. In addition to this, Impala has a SQL front-end that may be used to retrieve data stored in the HBase database system or in the Amazon Simple Storage System (S3).
- Impala queries often provide results within seconds or a few minutes, but Hive queries can take several minutes or even hours to finish. Impala queries also deliver results much more quickly.
- Parquet is a columnar storage architecture that is suited for large-scale queries, which are prevalent in data warehouse applications. Impala is a pioneer in the usage of the Parquet file format, which was developed by Facebook.

Impala enables you to do SQL queries directly on your Apache Hadoop data while it is stored in HDFS, HBase, or the Amazon Simple Storage Service. These searches are both quick and interactive (S3). Impala utilises the same metadata, SQL syntax (referred to as Hive SQL), ODBC driver, and user interface (referred to as the Impala query UI in Hue) as Apache Hive. This is in addition to using the identical unified storage platform.

This offers a platform that is both well-known and unified, and it can be used for real-time or batch-oriented queries.

Impala is a new tool that was recently added to the arsenal of options for querying large amounts of data. Batch processing frameworks like Hive that are based on MapReduce are not rendered obsolete by Impala's introduction. Batch operations that run over an extended period of time, such as those requiring the batch processing of Extract, Transform, and Load (ETL) type processes, are the optimum use case for Hive and other frameworks that are based on MapReduce.

Benefits of using Impala

Impala gives you the following benefits: • A SQL interface that data scientists and analysts are already familiar with.

- The ability to query large amounts of data in Apache Hadoop (also known as "big data").

- Queries that are executed in a decentralised manner inside a cluster setting, which enables simple scalability and the use of commodity hardware that is more cost-effective.
- The capability to exchange data files across multiple components without the need for any steps including copying or exporting and importing the data; for instance, to write with Pig, convert with Hive, and query with Impala. Because Impala can read from and write to Hive tables, it is possible to do analytics on Hive-produced data while using Impala for basic data exchange.
- A unified platform for the processing and analysis of large amounts of data, allowing clients to save unnecessary expenditures on modelling and ETL.

How Impala Works with Hadoop

The following are the components that make up the Impala solution:

- Clients - Impala's ability to interface with external entities is enabled through clients such as Hue, ODBC clients, JDBC clients, and the Impala Shell.
- Typically, queries and administrative actions like connecting to Impala are carried out with the assistance of these interfaces.
- The Hive Metastore is where information about the data that Impala has access to is stored. For instance, the metastore provides Impala with information on the databases that are accessible as well as the layout of those databases. The relevant metadata changes are automatically broadcast to all Impala nodes by the dedicated catalogue service that was introduced in Impala 1.2. This occurs whenever you make changes to schema objects, such as creating, dropping, or altering them; loading data into tables; and so on through Impala SQL statements.
- The Impala process is responsible for query coordination and execution. It is a process that runs on DataNodes. Impala clients' queries may be received, planned, and coordinated by each instance of Impala running. The queries are subsequently divided over the several Impala nodes, which then take on the role of workers to carry out the simultaneous query fragments.
- HBase and HDFS are used for storing data that may be searched later.

Impala and Query Execution

The following procedures are used to handle queries that are run using Impala:

1. User applications use ODBC or JDBC, which both offer standardised querying interfaces, to deliver SQL queries to Impala. Any impalad in the cluster is available for the user application to connect to. This impalad will serve as the query's coordinator going forward.
2. After the query has been parsed and analysed by Impala, it is determined what actions need to be carried out by various impalad instances located across the cluster. Planning goes into the execution to ensure maximum effectiveness.
3. In order to retrieve data, local instances of impalad connect to several services, such as HDFS and HBase.
4. The results of each impalad are sent to the client via the coordinating impalad, which receives the data from each impalad.

Impala and Hive

Impala takes use of a variety of Hadoop ecosystem components that are already well-known to users. Since Impala is capable of exchanging data with other Hadoop components in both a

consumer and producer capacity, it is able to integrate itself into your ETL and ELT pipelines in a variety of different ways.

An important objective of the Impala project is to improve the speed and effectiveness of SQL-on-Hadoop operations to the point where they become appealing to new sorts of users and open up Hadoop to new kinds of use cases. It takes use of preexisting Apache Hive infrastructure whenever it is feasible to do so in order to carry out long-running, batch-oriented SQL queries; this infrastructure is already in place for many Hadoop users.

Impala, in particular, stores the definitions of its tables in a regular MySQL or PostgreSQL database that is referred to as the metastore. This is the same database that Hive uses to store information of this kind. Therefore, Impala is able to access tables that have been created or loaded by Hive, so long as all columns utilise data types, file formats, and compression codecs that are supported by Impala.

As a result of the original emphasis placed on the features and efficiency of queries, Impala is able to read a wider variety of data types using the SELECT statement than it is able to create using the INSERT statement. Hive must be used to load the data before it can be queried via file formats such as Avro, RCFile, or SequenceFile.

The Impala query optimizer has the ability to use table statistics in addition to column statistics. Initially, you acquired this information by using the ANALYZE TABLE command in Hive; however, beginning with version 1.2.2 of Impala and above, you should use the COMPUTE STATS statement instead. The COMPUTE STATS tool needs less setup, is more stable, and does not require the user to move between the Impala shell and the Hive shell at any point throughout the process.

[A Brief Introduction to Impala Metadata and the Metastore](#)

In the section titled "How Impala Works with Hive," located on page 19, it is said that Impala stores information on table definitions in a central database called as the "metastore." Additionally, Impala monitors the following additional information for the low-level properties of data files:

- The actual locations, inside HDFS, of each block in the file system.

When a table has a significant amount of data and/or a big number of partitions, obtaining all of the table's information may be a time-consuming process that, in certain instances, might take several minutes. Therefore, all of this information is stored in the cache of each Impala node so that it may be reused for subsequent searches against the same database.

Before a query can be issued against a table, all of the other Impala daemons in the cluster need to receive the most recent metadata, which will replace any out-of-date cached metadata. This is necessary in the event that either the table definition or the data contained within the table is modified. All DDL and DML statements that are executed via Impala are subject to an automated metadata update, which is handled by the catalogd daemon and begins with version 1.2 of the Impala database. For more information, please refer to page 17 of The Impala Catalog Service.

You should still use the REFRESH statement (when new data files are added to existing tables) or the INVALIDATE METADATA statement (for entirely new tables or after dropping a table, performing an HDFS rebalance operation, or deleting data files) when performing DDL and DML operations through Hive or making changes manually to files in HDFS. The retrieval of metadata for all tables that are monitored by the metastore is performed when the INVALIDATE METADATA command is issued on its own. If you are aware that only some tables have been modified outside of Impala, you may use

the command REFRESH table name for each table that has been impacted to only obtain the most recent metadata for the tables that have been modified.

How Impala Uses HDFS

As its primary form of data storage, Impala makes extensive use of the distributed filesystem HDFS. When it comes to protecting itself from failures in hardware or networks on individual nodes, Impala depends on the redundancy offered by HDFS. The information included inside Impala tables is stored in HDFS in the form of data files, and these files make use of the standard HDFS file formats and compression codecs. Impala will read all of the data files that are present in the directory for a new table, notwithstanding the fact that each file has a different name. Impala assigns new names to existing files before adding new data to them.

How Impala Uses HBase

An alternate storage media for Impala data is HBase, which is an alternative to HDFS. It is a database storage system that is built on top of HDFS, however it does not have any built-in support for SQL. A great number of Hadoop users already have it installed and store enormous data sets in it that are often sparse. You will be able to query the contents of HBase tables using Impala if you first define such tables in Impala and then map them to similar tables in HBase. You may even conduct join queries that include both Impala and HBase tables in the results set.

SQL queries that are run on data that is stored in common Apache Hadoop file formats may be executed by the Apache Impala with high speed and low latency.

The following is an inventory of the components that make up the Impala solution.

Impala: The Impala service is responsible for the coordination and execution of queries that are received from users. The queries are subsequently divided over the several Impala nodes, which then take on the role of workers to carry out the simultaneous query fragments.

The Hive Metastore: is where information on the data that is accessible to Impala is stored. For instance, the metastore provides Impala with information on the databases that are accessible as well as the layout of those databases. Impala's dedicated catalogue service is responsible for automatically broadcasting any relevant metadata changes to all of Impala's nodes whenever you make changes to the schema (such as creating, dropping, or altering objects), load data into tables, or perform any other operation using SQL statements.

Clients: Interactions with Impala may be carried out by a wide variety of entities, including Hue, ODBC clients, JDBC clients, Business Intelligence applications, and the Impala Shell. Typically, queries and administrative actions like connecting to Impala are carried out with the assistance of these interfaces.

Constituent parts of the Impala

The Impala service is a massively parallel processing (MPP) database engine that operates over a distributed environment. It is made up of a variety of daemon processes, each of which operates on a distinct host inside your Hadoop cluster.

The Impala service is comprised of the following groups of processes, which are together referred to as roles.

Impala Daemon

The Impala daemon, which is physically embodied by the `impalad` process, is the essential component of the Impala system. A few of the most important tasks that are carried out by an Impala daemon are as follows:

- Performs reading and writing operations on data files.
- It'll take queries sent through the `impala-shell` command, Hue, JDBC, or ODBC if you choose to use it.
- Performs the queries in parallel and distributes the work evenly among the cluster.
- Sends intermediate query results to the central coordinator for review.
- One of the following strategies might be used to bring about the appearance of Impala daemons:
 - Both HDFS and Impala are hosted on the same physical machine, and each Impala daemon is executed on the same host as a DataNode.
 - Impala is installed independently in a computing cluster, and it retrieves data from HDFS, S3, ADLS, and other locations through remote connections.

StateStore

The Impala StateStore continually monitors the health of all Impala daemons in a cluster. It's a statestored daemon process. One cluster host needs this procedure. If one Impala daemon becomes offline due to hardware failure, network fault, software issue, or other cause, the StateStore notifies all other Impala daemons so subsequent queries may bypass the inaccessible daemon.

StateStore is not necessarily crucial to the regular functioning of an Impala cluster since it helps when things go wrong and broadcasts information to coordinators. If the StateStore is not functioning or unavailable, Impala daemons continue running and distributing tasks as normal with known data. If additional Impala daemons die, the cluster becomes less resilient and metadata becomes less consistent. StateStore reestablishes contact with Impala daemons and continues monitoring and broadcasting when it comes back online.

Queries that access the new object produced by a DDL statement while the StateStore is offline will fail.

Impala Server

The Catalog Server sends Impala SQL statement information updates to all Impala daemons. It's a catalogd daemon process. One cluster host needs this procedure. Because queries are transmitted via StateStore, statestored and catalogd should execute on the same host.

When Impala statements alter metadata, the catalogue service avoids `REFRESH` and `INVALIDATE METADATA` commands. Before running a query on an Impala node after creating a table, loading data, etc. in Hive, you must perform `REFRESH` or `INVALIDATE METADATA`.

Chapter-8: Apache Oozie

Overview

Oozie apps are comparable to executables found in Unix, whereas Oozie jobs are comparable to the processes found in Unix. Users of Oozie create applications, and each individual runthrough of an application is referred to as a job.

Apache Oozie Workflow Scheduler for Hadoop is a workflow and coordination tool for managing Apache Hadoop processes, including the following:

- Directed acyclic graphs (DAGs) of actions are what make up Oozie Workflow tasks. Generally speaking, actions are Hadoop jobs (MapReduce, Streaming, Pipes, Pig, Hive, Sqoop, etc).
- Recurring Workflow tasks may be triggered by Oozie Coordinator jobs depending on the time (or frequency) and the availability of data.
- Oozie Bundle jobs are collections of Coordinator tasks that are treated as a single job for the purposes of management.

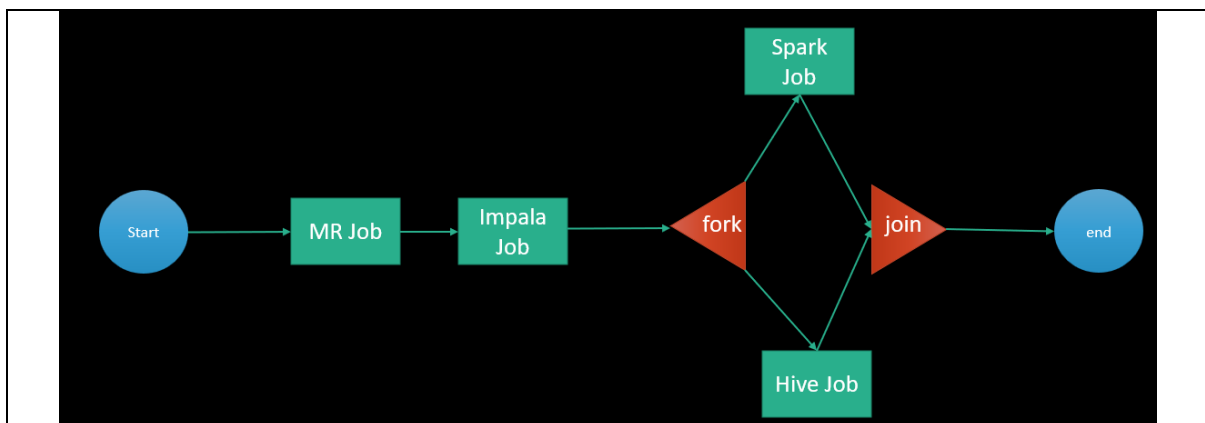
You may utilise Oozie, which is an extensible, scalable, and data-aware service, to coordinate dependencies across tasks that are operating on Hadoop.

Oozie Workflows

A Hadoop task that consists of many stages is known as an Oozie workflow. A workflow is a collection of action nodes and control nodes that are organised in a directed acyclic graph (DAG) that captures control dependence. Each action in a workflow is often a Hadoop job (for example, a MapReduce, Pig, Hive, or Sqoop job), and a DAG records the control dependency. In addition, there may be activities that are not classified as Hadoop jobs (e.g., a Java application, a shell script, or an email notification).

The sequence in which these activities are carried out is determined by the arrangement of the nodes in the process. In a process, an activity will not begin until the action that came before it has completed.

It is up to the control nodes in a process to regulate the order in which activities are carried out. The beginning and the conclusion of a process are denoted by the start control node and the end control node, respectively. The control nodes known as fork and join make it possible to do out tasks simultaneously. The decision control node functions similarly to a switch or case statement in that it may pick a certain execution route inside the work flow by making use of information gleaned directly from the task. An example of a work flow is shown.



Workflows are directed acyclic graphs, which means that they do not support loops in the flow of information.

Oozie is a web-based application written in Java that serves as a workflow manager and coordinator. It is used to manage and coordinate activities in the Hadoop ecosystem. The structure of its work

flow is quite similar to that of a Direct Acyclic Graph (DAG)[1]. Oozie is capable of managing thousands of tasks in a Hadoop cluster because to its scalable design.

There are three standard occupations available in Oozie.

1. Oozie Workflow Jobs: These jobs indicate the order in which actions are to be carried out.
2. Oozie Coordinator Jobs: Coordinates the task and determines when it should be activated based on factors such as time and data availability.
3. The Oozie Bundle is a package consisting of numerous Workflow and Coordinator applications.

Action node and control flow node are both components of the Oozieworkflow.

Action node: An action node is a representation of a workflow activity, such as importing data using Sqoop, putting files into HDFS, importing data using MapReduce, Pig, or Hive tasks, or executing a shell script for a programme that was built in Java. Action nodes are responsible for making decisions on how jobs are carried out.

Control-flow node: A control-flow node directs the flow of work from one action to the next inside a workflow by permitting features such as conditional logic, which allows for the workflow to take one of many possible paths based on the outcome of a previous action node.

There is just one control node, but the number of jobs determines how many action nodes will be created. The number of jobs and the action node's own count will always be equal.

Oozie Architecture

The Oozie server is implemented as a Java web application, and all of the required data is saved in a database. Any kind of database will do, whether it Derby, MySQL, or Oracle, for example. Hadoop cluster serves as the repository for all tasks. Oozie customer contact to the Oozie server, which will be responsible for maintaining and processing the tasks. Following the processing of the data, valuable information is saved in the database. Hadoop cluster serves as the repository for all tasks. The Oozie client makes contact with the Oozie server so that jobs can be managed and processed.

Following the processing of the data, valuable information is saved in the database.

Use-Cases of Apache Oozie

Apache Oozie is used by Hadoop system administrators to run complex log analysis on HDFS. Hadoop Developers use Oozie for performing ETL operations on data in a sequential order and saving the output in a specified format (Avro, ORC, etc.) in HDFS. In an enterprise, Oozie jobs are scheduled as coordinators or bundles.

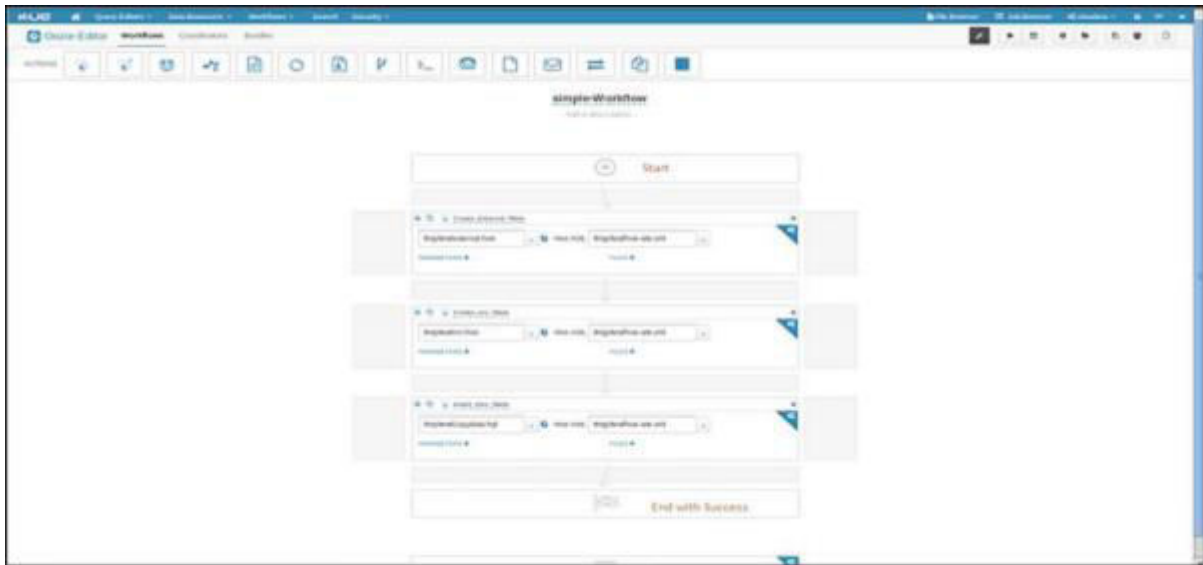
Oozie Editors

Before we dive into Oozie lets have a quick look at the available editors for Oozie. Most of the time, you won't need an editor and will write the workflows using any popular text editors (like Notepad++, Sublime or Atom) as we will be doing in this tutorial.

But as a beginner it makes some sense to create a workflow by the drag and drop method using the editor and then see how the workflow gets generated. Also, to map GUI with the actual workflow.xml created by the editor. The most popular among Oozie editors is Hue.

Hue Editor for Oozie

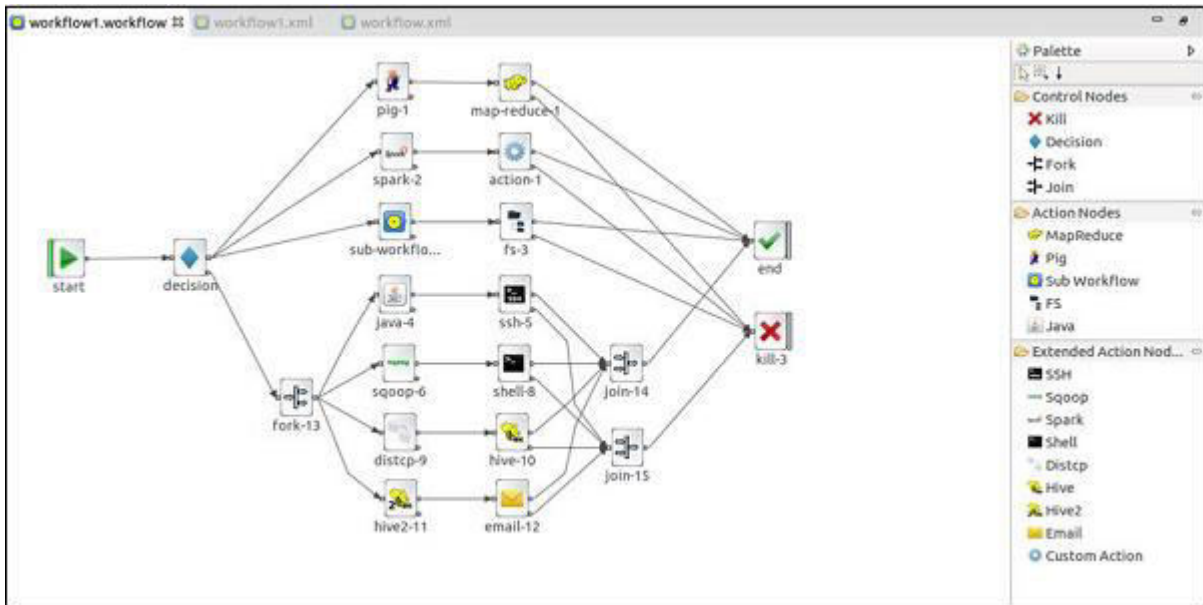
This editor is very handy to use and is available with almost all Hadoop vendors' solutions. The following screenshot shows an example workflow created by this editor.



You can drag and drop controls and actions and add your job inside these actions.

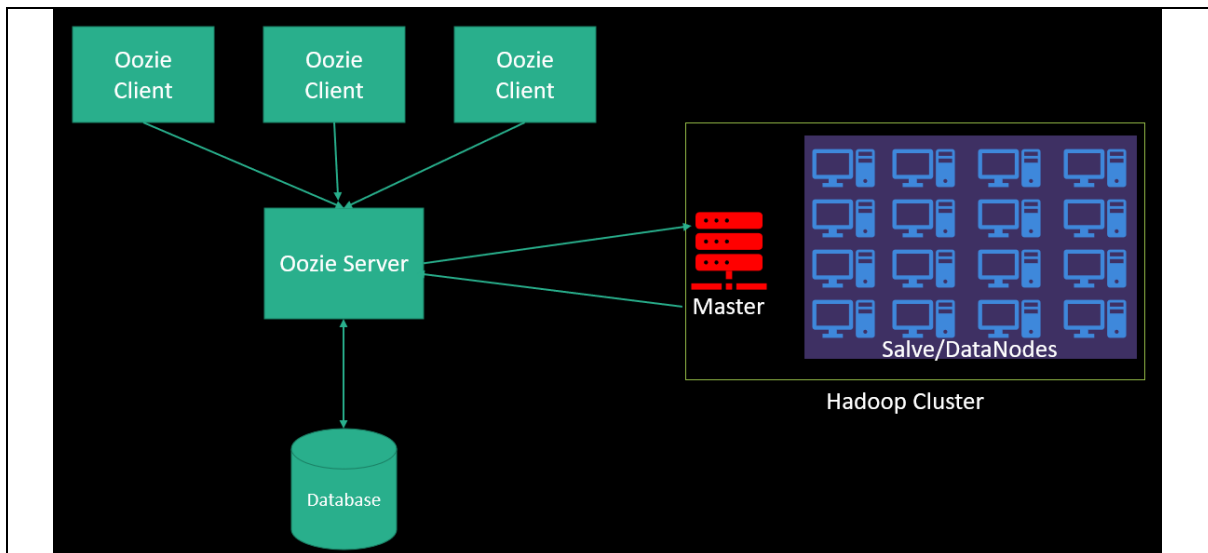
Oozie Eclipse Plugin (OEP)

Oozie Eclipse plugin (OEP) is an Eclipse plugin for editing Apache Oozie workflows graphically. It is a graphical editor for editing Apache Oozie workflows inside Eclipse. Composing Apache Oozie workflows is becoming much simpler. It becomes a matter of drag-and-drop, a matter of connecting lines between the nodes. The following screenshots are examples of OEP.



The bottom screenshot shows the same workflow editor with the 'map-reduce-1' node selected. The Properties panel on the right displays the following configuration:

Property	Value
Archive	Archive
Configuration	
Configuration	[Name: , Value: , Description:]
Description	
Name	
Value	
File	
File	
Job XML	
Job XML	
Misc	
ConfigClass	
Cred	
Jobtracker	job-tracker
Name	map-reduce-1
Namenode	name-node
Pipes	[Map: , Reduce: , Input Format: , Output Format: , Part]
Position	[X: 470, Y: 26]
Prepare	[[Delete: hdfs://lol:1/hadoop], [Mkdir: hdfs://lol:1/had
Retry Interval	
Retry Max	
Streaming	[Mapper: , Reducer: , Record Reader: , [], []]
Type	map-reduce



Oozie Workflow

The Oozie Workflow is a set of actions that are organised in a DAG. Definition of an Oozie process written in the hPDL language. The Oozie process has a set of nodes, including the Start control node, the End control node, the Kill control node, the Decision node, the Fork node, and the Join node.

- a. Start control node: Every Oozie workflow has to have a start control node, and the workflow will always begin execution from the start control node.
- b. End control node: Once the task has been successfully finished, it will proceed to the end control node. When you reach the end control node, it indicates that there were no errors.
- c. Kill control node: If we want to stop the workflow from being executed, then we need to utilise the kill control node. It is possible that there are many kill control nodes.

Chpater-9: Apache Kafka

Overview

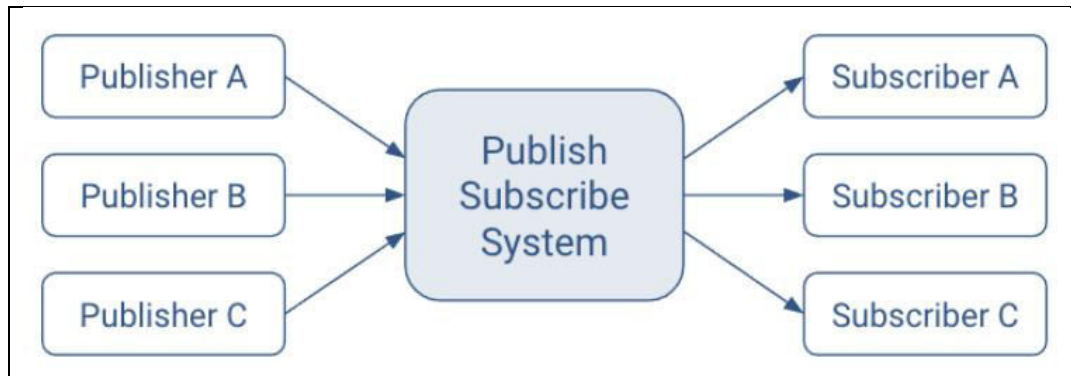
Apache Kafka is a platform for continuously receiving messages. It is built with high performance, high availability, and redundancy in mind from the beginning.

The following are some examples of apps that may leverage such a platform:

- Things Connected to the Internet Telemetry data may be sent back to a server via the Internet from a variety of devices, including televisions, refrigerators, washing machines, dryers, thermostats, and personal health monitors.
- Sensing and Control Networks. It is possible to equip both expansive environments (such as farms, amusement parks, and woods) and intricate machines (such as engines) with a variety of sensors to monitor data or the present state of the environment.
- Positional Data. Sending location data to a centralised platform is possible for delivery vehicles as well as massively multiplayer online games.
- Any Additional Real-Time Data. Satellites and medical sensors both have the ability to transmit data to a centralised location for further analysis.

Pub Sub System

The ideal publish-subscribe system is one that is simple and uncomplicated: the messages published by Publisher A need to find their way to Subscriber A, the messages published by Publisher B need to find their way to Subscriber B, and so on.



- Unlimited Lookback is a perk that should be included in any ideal system. At any given moment in time, a new Subscriber A1 is able to read the stream that is being published by Publisher A.
- Message Retention. No messages are lost.
- No limits on the storage space. Messages can be stored in the publish-subscribe system in an unlimited capacity.
- Absolutely no downtime. The publish-subscribe system never goes offline for maintenance.
- There are no scaling limits. The publish-subscribe system is able to accommodate an unlimited number of publishers and/or subscribers while maintaining a consistent delivery latency for messages.

Kafka Architecture

The design of Kafka deviates from that of the ideal publish-subscribe system, as is the case with all systems that exist in the actual world. Some of the most important distinctions are as follows:

- A replicated and distributed commit log serves as the foundation upon which messaging is built.
- Because the customer now has additional functionalities, they are also responsible for a greater amount.
- Instead of individual messages, batch messaging is tuned to work more efficiently.
- Messages are kept even after they have been eaten, and users have the ability to consume them again.

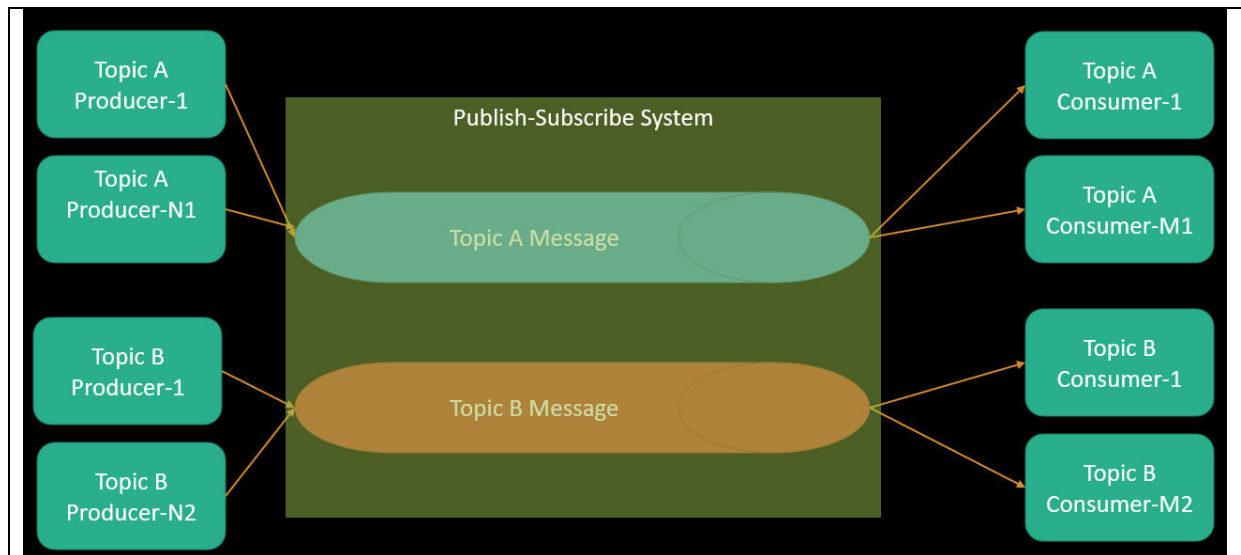
These design choices have led to the following outcomes:

- Extreme horizontal scalability
- Very high throughput
- High availability
- However, distinct semantics and message delivery guarantees are guaranteed.

Topics

In the hypothetical system that was just described, messages sent out by a single publisher would magically make it to the inboxes of each subscriber.

Kafka incorporates the idea of a topic into his writing. The use of a topic makes it easier for publishers and subscribers to find each other.



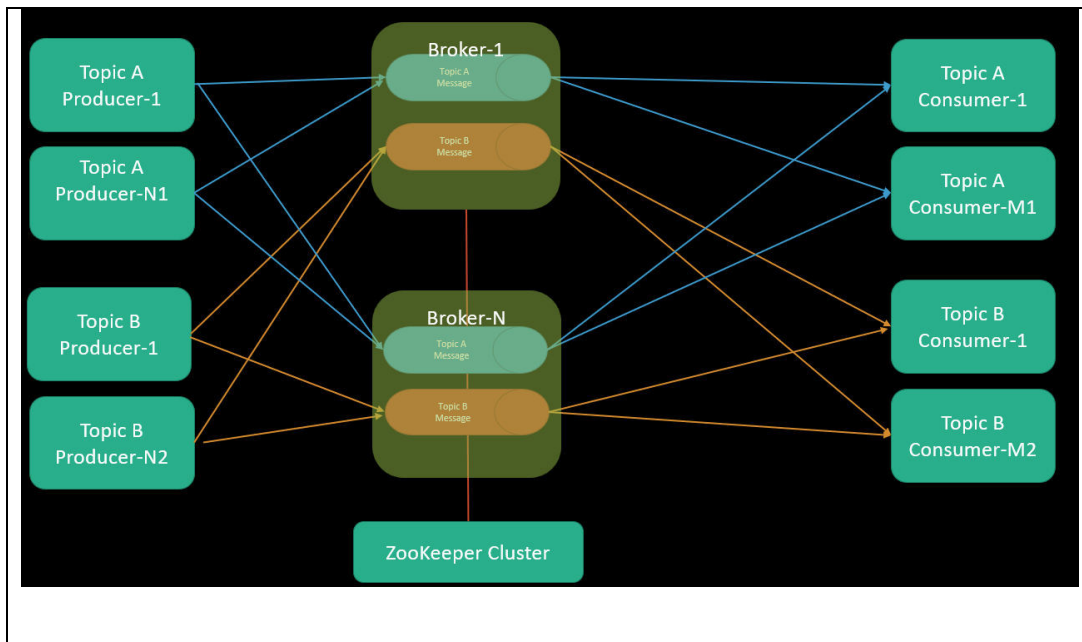
A queue of messages that have been produced by one or more producers and read by one or more consumers is what we mean when we talk about a subject. The name of a subject is what identifies it. This name is a component of a namespace that is used globally by that Kafka cluster.

Only applicable to Kafka: Subscribers are referred to as consumers, while publishers are referred to as producers. When a producer or consumer connects their device to the publish-subscribe system, they are granted the ability to read from or contribute to a particular subject.

Brokers

Kafka is a distributed system that embodies the fundamental elements of the publish-subscribe architecture defined earlier as the ideal.

Each host that makes up the Kafka cluster is responsible for running a server that is known as a broker. This broker stores messages that have been submitted to topics and fulfills consumer requests.



Kafka was developed to operate on numerous hosts simultaneously, with a separate broker running on each server. If one of the hosts stops working for whatever reason, Kafka will do all in its power to keep the others operational. This helps to achieve some of the criteria for the ideal publish-subscribe system, including "No Downtime" and "Unlimited Scaling."

All of the Kafka brokers communicate with Zookeeper in order to provide distributed coordination, which provides further support for the "Unlimited Scaling" requirement of the ideal system.

The same topics are discussed by many brokers. In order to achieve the objectives of "No Downtime," "Unlimited Scaling," and "Message Retention," replication is an essential component.

There is a single broker who is in charge of organising the activities of the cluster. This particular broker is referred to as the controller. As was discussed before, the optimal behaviour of a topic is that of a queue of messages. In point of fact, having just a single queue causes scale problems. The implementation of partitions in Kafka is what contributes to the topics' resilience.

Records

A record is the term used in Kafka to refer to a publish-subscribe message. A record is made up of a key-value pair as well as other information, which may include a timestamp. The key is optional; however, it may be used to distinguish messages coming from the same data source. Arrays of bytes are used for the storage of keys and values in Kafka. Aside from that, it does not care about the format at all.

Headers are something that may be included in the metadata of each record. Key-value pairs may be used to contain application-specific information when using headers.

In the context of the header, the keys are considered to be strings, while the values are considered to be byte arrays.

Partitions

Kafka organises the data it manages into partitions, rather than storing them all in a single log as would be the case with other systems. One way to think of partitions is as a subset of all the records

that pertain to a subject. The concept of "Unlimited Scaling" may be helped along by partitions. Records that are included inside the same partition are organised according to their time of entry.

When a subject is first formed, it is given two attributes to customise it with:

partition count

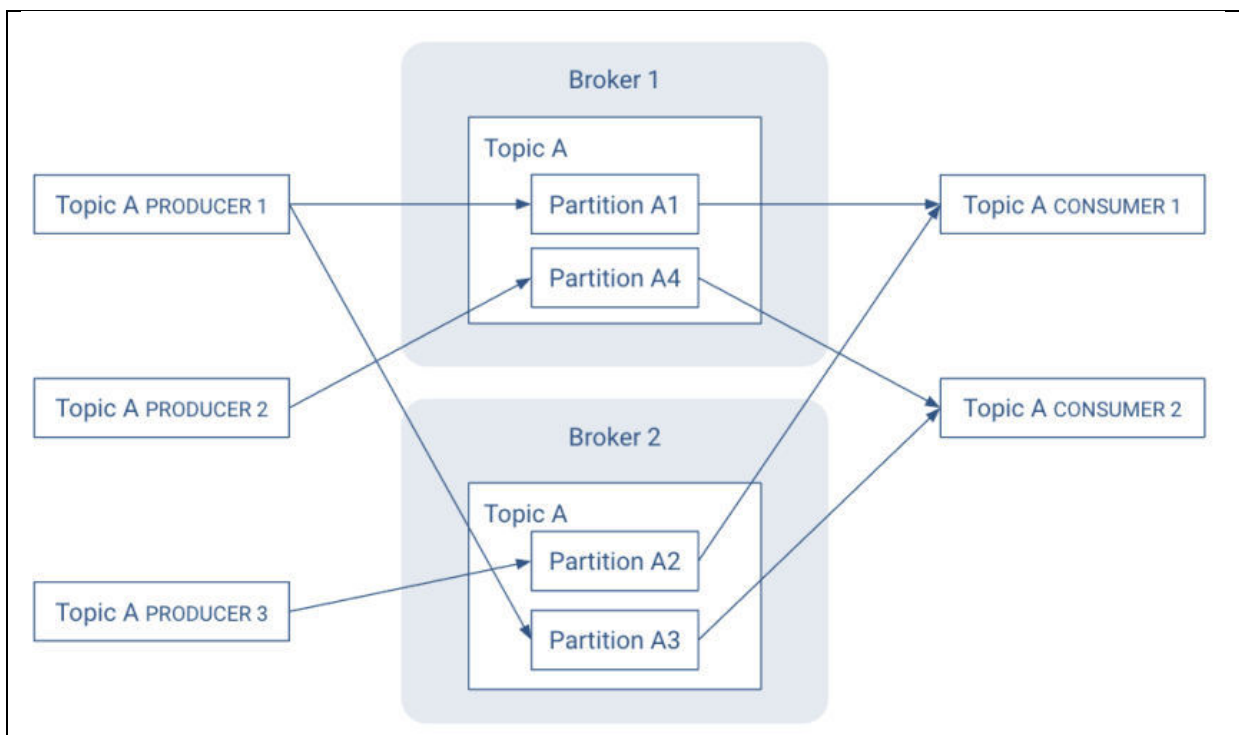
The number of partitions that records for this topic will be spread among.

Replication factor: The number of copies of a partition that are maintained to ensure consumers always have access to the queue of records for a given topic. Each topic has one leader partition. If the replication factor is greater than one, there will be additional follower partitions. (For the replication factor = M, there will be M-1 follower partitions.)

Any Kafka client (a producer or consumer) communicates only with the leader partition for data. All other partitions exist for redundancy and failover. Follower partitions are responsible for copying new records from their leader partitions. Ideally, the follower partitions have an exact copy of the contents of the leader. Such partitions are called in-sync replicas (ISR). With N brokers and topic replication factor M, then

- If $M < N$, each broker will have a subset of all the partitions
- If $M = N$, each broker will have a complete copy of the partitions

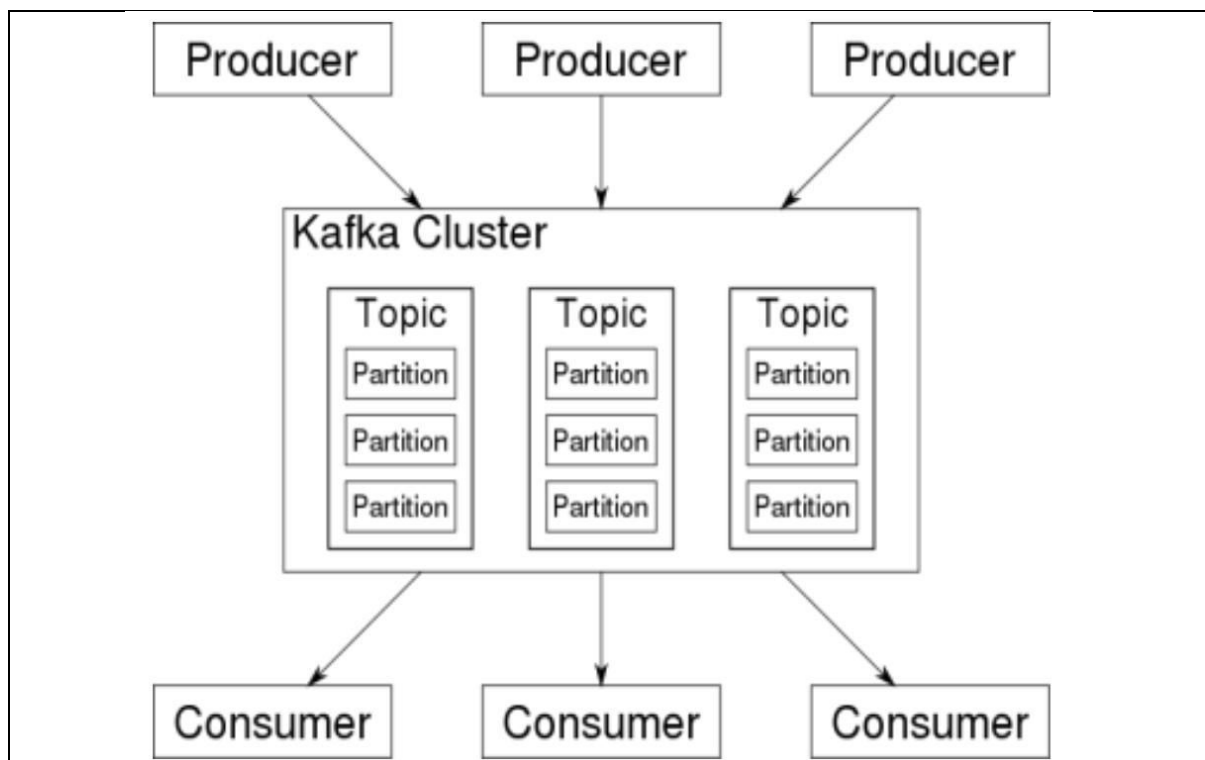
In the following illustration, there are $N = 2$ brokers and $M = 2$ replication factor. Each producer may generate records that are assigned across multiple partitions.



When it comes to maintaining accurate record throughput, partitions are essential. Choosing the appropriate number of partitions and replications for a topic does two things: it ensures that the leader partitions are distributed uniformly throughout the brokers in the cluster, and it ensures that

partitions belonging to the same topic are about the same size. Helps to distribute the workload among the brokers.

In reality, Kafka is a store that is responsible for storing messages that originate from processes, also known as producers. After that, the data or messages are partitioned into a number of distinct partitions inside the respective Topics. The messages are indexed and saved in this Topic's partition in addition to a timestamp for each message. On the other hand, other processes that are referred to as Consumers are able to query messages from these partitions. Kafka, which is working between these producers and consumers, operates on a cluster consisting of one or more servers, and the partitions may be dispersed among different nodes in the cluster. When Apache Storm, Apache HBase, and Apache Spark are used in conjunction with Apache Kafka, the real-time streaming data is processed in an effective and efficient manner. The fundamental structure of Kafka is seen in Figure.



As a result of Kafka being deployed as a cluster on many servers, the complete publish and subscribe messaging system is managed by Kafka with the assistance of four application programming interfaces (APIs), namely the producer API, consumer API, streams API, and connector API. Because of its capacity to provide fault-tolerant delivery of enormous message streams, it has begun to replace some of the more traditional messaging systems, such as JMS, AMQP, and others. Topics, records, and brokers are three of the most important concepts in Kafka's architectural scheme. The stream of records that make up a topic each contain a unique set of information. On the other hand, the responsibility of reproducing the communications falls on the Brokers.

KAFKA REAL TIME APPLICATIONS

Messaging: Kafka works glowing as a substitute message broker which is used for a variety of reasons. Kafka has well throughput and built-in partitioning with replication, and fault tolerance high makes it a good solution for large scale message processing applications. In our experience messaging uses are often comparatively low-throughput, but may require low end-to-end latency and often depend on the strong durability guarantees Kafka provides.

Website Activity Tracking: The original application of Kafka was to be able to rebuild a user activity tracking pipeline as a set of real-time publish-subscribe feeds means site activity like page views, searches, or other actions users may take is published to central topics with one topic per activity type. These feeds are available for subscription for a range of use cases including real-time processing, real-time monitoring, and loading into Hadoop or offline data warehousing systems for offline processing and reporting. Activity tracking is often very high volume as many activity messages are generated for each user page view.

Metrics: Kafka is often used for operational monitoring data. This indulges aggregating statistics from distributed applications to produce centralized feeds of operational data.

Log Aggregation: Kafka is also used as an alternative to log aggregation solution. Log aggregation Combines physical log files for servers and places them in a central processing location. Kafka extracts file details and provides clearer extraction of log or event data as a stream of messages. This allows processing of low latency response time and easier support for multiple data sources and distributed data consumption. Compared with to log-centric systems such as Scribe or Flume, Kafka offers equally good performance, stronger durability guarantees due to replication, and much lower end-to-end latency.

Stream Processing: Kafka users are using Kafka to process data in processing pipelines of multiple stages, and raw input data are put into use in Kafka and then added, enriched or converted into new themes for subsequent consumption or tracking. For example, in order to use news articles, a workstation can scan the content of the article in its RSS content on "articles"; Additional processing can normalize or reduce this content and publish the content of the pure article to a new topic; the last run may try to present this content to users. These processing pipelines create real-time data streams based on individual themes. According to 0.10.0.0, Apache Kafka has a light but powerful streaming library called Kafka Stream to perform data processing as described above. Apart from Kafka Streams, alternative tools for the development of open-source script include Apache Storm and Apache Samza.

Chapter-10: Apache NiFi

Overview

To put it another way, NiFi was developed to automate the transfer of data from one system to another. Although the word "dataflow" is used in a number of different settings, for the sake of this discussion, we will use it to refer to the controlled and automatic flow of information between systems. This issue area has been there since since businesses started using more than one system, with some of those systems producing data and some of those systems using data for their operations. Extensive discussion and articulation have taken place with regard to the challenges and solution patterns that have arisen. The Enterprise Integration Patterns document has a format that is both comprehensive and easy to consume.

Through the years, dataflow has become one of those undesirable aspects of an architecture that are required. However, there are a number of active and quickly developing initiatives that are making dataflow a whole deal more fascinating and a great deal more crucial to the success of any given business. Among them are topics like service-oriented architecture, the proliferation of application programming interfaces (APIs), the internet of things, and big data. In addition, the amount of stringency that is required to ensure compliance, privacy, and security is always increasing.

Even with all of these new ideas being developed, the patterns and requirements of dataflow have remained essentially same for the most part.

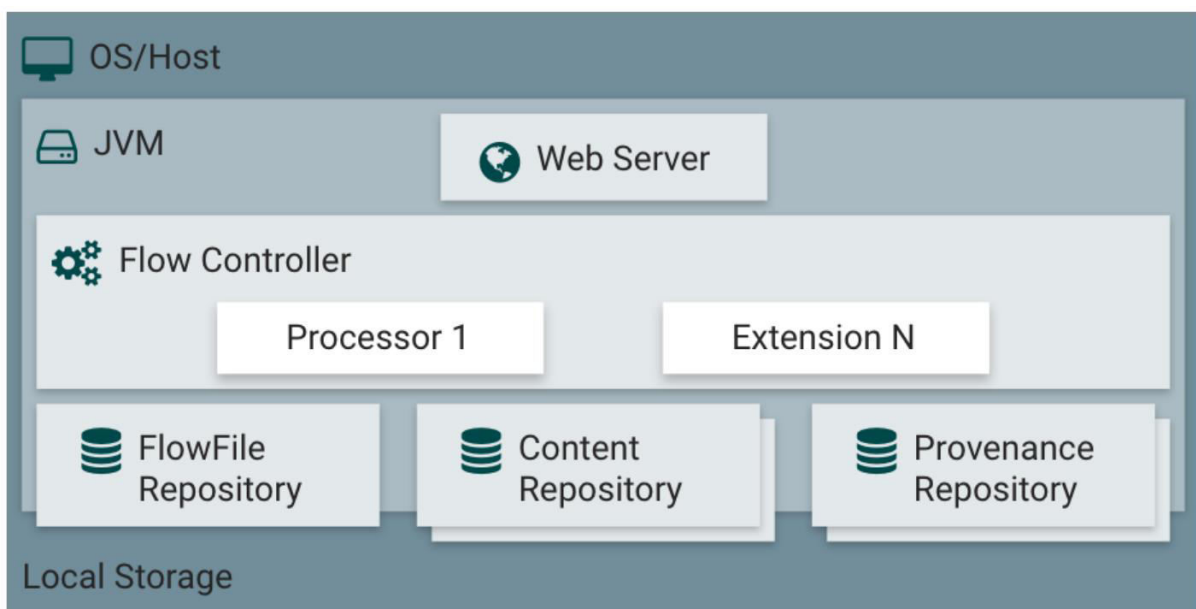
The key distinctions then are the magnitude of the complexity, the velocity of change that must occur in order to adapt, and the fact that, at scale, the exceptional instance becomes the norm. NiFi was developed specifically to assist in overcoming the difficulties associated with current dataflow.

Benefits of NiFi DataFlow

The use of this design paradigm results in a number of advantageous outcomes, which contribute to NiFi's status as a highly efficient platform for the construction of robust and scalable dataflows. A few examples of these advantages are as follows:

- Is inherently asynchronous, which allows for very high throughput and natural buffering even as processing and flow rates fluctuate
- Provides a highly concurrent model without a developer having to worry about the typical complexities of concurrency
- Promotes the development of cohesive and loosely coupled components, which can then be reused in other contexts and promotes testable unidirectional dependencies
- Lends itself well to the visual creation and management of directed graphs of processors
- Error management becomes as natural as the happy route rather than a coarse-grained catch-all due to the resource restricted connections.
- Critical functions such as back-pressure and pressure release become highly natural and intuitive.
- It is possible to quickly understand and monitor the flow of data across the system, as well as the points of entry and departure for the data itself.

NiFi Architecture



NiFi is run on a host operating system inside a Java Virtual Machine (JVM). The following is a list of the key components of NiFi when run on the JVM:

Web Server: The HTTP-based command and control API for NiFi is what the web server will be used to host as its primary function.

Controller of the Flow: The flow controller is the component that acts as the operation's central processing unit. It handles the schedule of when extensions obtain resources to execute and offers threads for them to run on. Extensions may run on these threads.

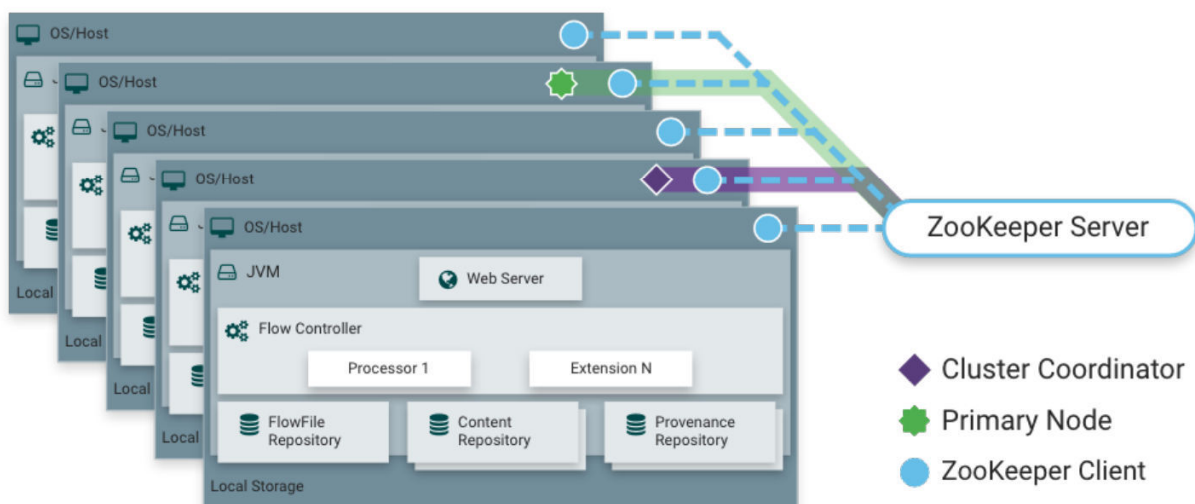
Extensions: Other publications detail the different sorts of NiFi extensions, which may be found in their respective dictionaries. The most important thing to understand here is that extensions work and run inside of the JVM.

FlowFile's File Storage Repository: NiFi maintains a record of the current status of what it knows about a specific FlowFile that is currently being used in the flow in a location known as the FlowFile Repository. It is possible to plug different implementations into the repository. The permanent Write-Ahead Log that is stored on a particular disc partition is the method that is used by default.

Content Repository: The actual content bytes of a specific FlowFile are stored in the Content Repository, which is referred to simply as the Content Repository. It is possible to plug different implementations into the repository. The method that is used by default is a rather straightforward process that saves data in chunks inside the file system. You have the option of specifying more than one file system storage location in order to have various physical partitions activated and so lessen the amount of congestion on any one volume.

Provenance Repository: The Provenance Repository is where all of the data on the provenance of events is housed. The repository structure is pluggable, and the default implementation uses one or more actual storage volumes. The repository may be expanded using additional plug-ins. Event information is indexed and searchable inside each place it is stored in.

NiFi as Cluster



NiFi 1.0 uses Zero-Leader Clustering. Each NiFi node conducts the same functions on various data sets. Apache ZooKeeper automatically selects a Cluster Coordinator and handles failover. The Cluster Coordinator receives heartbeat and status updates from all nodes. Cluster Coordinator reconnects nodes. ZooKeeper elects a Primary Node for each cluster. DataFlow managers may interact with the NiFi cluster using any node's UI. Changes are copied to all cluster nodes, giving numerous access points.

NiFi Features

- **Guaranteed Delivery (Guaranteed Delivery)**

NiFi was founded on the principle that even when operating at a very large scale, assured delivery remains an absolute need.

This is made possible by the efficient use of a purpose-built permanent write-ahead log as well as a content repository. Together, they have been developed in such a manner as to make it possible for extremely high transaction rates, efficient load-spreading, copy-on-write, and to play to the strengths of conventional disc read/write operations.

- **Buffering of Data with Back Pressure and Pressure Release**

NiFi allows for the buffering of any and all data that is queued, as well as the ability to apply back pressure to queues when they reach a certain limit or to "age off" data when it hits a certain age threshold (its value has perished).

- **Queuing Based on Priority**

In order to retrieve data from a queue in the most efficient manner possible, NiFi permits the setup of one or more prioritising algorithms.

The most recent data should be retrieved first by default; however, there are situations in which the data should be fetched in a different order, such as the biggest first or the oldest first.

- **Flow-Specific Quality of Service (latency v throughput, loss tolerance, etc.)**

There are some places along the path of a dataflow when the data is vitally essential and cannot tolerate any loss. There are also situations in which it must be processed and transmitted within a very short amount of time in order for it to be of any use. NiFi makes it possible to configure these concerns in a manner that is fine-grained and flow-specific.

Ease of Operation

- **Command and Control in the Visual Domain**

Dataflows have the potential to become pretty complicated. Having the ability to view those movements and communicate them graphically may be a huge assistance in reducing that complexity and determining which areas need to be reduced. NiFi not only allows the construction of dataflows in a visible manner, but it also does it in real time. It is more comparable to working with clay than it is to being able to "plan and deploy." If you make a modification to the dataflow, that it will take effect as soon as you save it. The changes are quite subtle and are confined to the specific components that were impacted.

It is not necessary to halt a complete flow or collection of flows in order to make a particular alteration.

- **Flow Scheduling Templates**

Dataflows have a tendency to be very pattern-oriented, and while there are often many alternative methods to address an issue, it is really helpful to be able to communicate the best practises that have been developed. The use of templates enables subject matter experts to construct and publish their flow designs, while also allowing others to benefit from and participate on these designs.

- **Data Provenance**

As objects move through the system, NiFi automatically records, indexes, and makes accessible the provenance data for each item. This applies to fan-in, fan-out, transformations, and other operations

as well. When used to support compliance requirements, troubleshooting, optimization, and a variety of other situations, this information takes on an incredibly vital role.

- **Recording a rolling buffer of fine-grained history as part of the recovery process**

The content repository that NiFi uses is intended to perform the function of a rolling buffer of history. Data is never deleted until it has become obsolete in the content repository or more storage space is required. This, in conjunction with the data provenance capabilities, allows for an exceptionally valuable base to enable click-to-content, download of content, and replay, all at a particular moment in the lifespan of an object, which may even span generations.

Protection

- **System to Protection System**

When it comes to dataflow, security is the single most important factor. NiFi provides a safe exchange at every point in a dataflow by making use of protocols that encrypt data, such as 2-way secure socket layer (2-way SSL). In addition, NiFi makes it possible for the flow to encrypt and decrypt material, as well as make use of shared keys or other techniques on either the sender or the receiver side of the equation.

- **User to Operating System**

NiFi supports two-way SSL authentication and offers pluggable authorization so that it may effectively regulate a user's access at a variety of different levels (read-only, dataflow manager, admin). If a user inputs a sensitive property like a password into the flow, it is instantly encrypted on the server side, and it is never again accessible on the client side, not even in its encrypted version. An example of such a property would be a credit card number.

- **Authorization for Multiple Tenants**

Each component is subject to the authority level of a specific dataflow, which enables an administrative user to exercise granular control over the access granted to other components. This indicates that each NiFi cluster has the capacity to meet the needs of one or more enterprises simultaneously. When compared to isolated topologies, multi-tenant authorization makes it possible to implement a self-service model for the management of dataflow. This model gives each group or organisation the ability to manage flows while maintaining full awareness of the portions of the flow to which they do not have access.

Extensible Architecture

- **Extension**

NiFi was designed from the ground up to be extensible, and as such it functions as a platform on which dataflow operations may be carried out and interact with one another in a way that is both dependable and consistent. Processors, Controller Services, Reporting Tasks, Prioritizers, and Customer User Interfaces are all examples of points of extension.

- **Isolation of the Classloader**

Dependency issues may arise very fast in any system that is built using components. This problem is addressed by NiFi, which provides a special class loader mechanism. This approach ensures that each extension bundle is only accessible to a very restricted range of dependencies, which solves the problem. As a consequence of this, extensions may be developed with very little consideration given to the possibility that they would be incompatible with another extension. The idea behind these

extension packages is referred to as "NiFi Archives," and it is covered in the Developer's Guide in further detail.

- **Site-to-Site Communication Protocol**

The NiFi Site-to-Site (S2S) Protocol is the recommended method of communication for using between different instances of NiFi. The use of S2S makes it simple to move data from one instance of NiFi to another in a way that is quick, effective, and safe. It is simple to build NiFi client libraries and incorporate them into other programmes or devices so that they may connect with NiFi via the S2S protocol. It is feasible to include a proxy server into the S2S communication since the socket-based protocol as well as the HTTP(S) protocol are supported in S2S in their capacity as the underlying transport protocol.

Flexible Scaling Model

- **Scaling-out of the (Clustering)**

As was discussed before, NiFi was developed to be able to scale out by using the clustering together of a large number of nodes.

If the provisioning and configuration of a single node is capable of handling hundreds of megabytes per second, then it should be possible to design even a small cluster to handle gigabytes per second. This therefore presents some intriguing issues in the form of load balancing and fail-over between NiFi and the systems from which it obtains its data. It may be beneficial to make use of asynchronous queuing-based protocols such as message services, Kafka, and so forth. Use of NiFi's 'site-to-site' feature is also very effective because it is a protocol that enables NiFi and a client (including another NiFi cluster) to talk to each other, share information about loading, and exchange data on specific authorised ports. This is achieved through the use of NiFi's 'site-to-site' feature, which can be found in the NiFi UI.

- **Scale up or down**

NiFi may also be scaled up or down in a fairly flexible manner due to the way it was created. When setting NiFi, under the Scheduling tab, it is possible to increase the number of concurrent jobs running on the processor, which, from the perspective of the NiFi framework, will result in a higher throughput. This makes it possible for more processes to run at the same time, which results in increased throughput. On the other end of the spectrum, it is possible to scale NiFi down to the point where it is acceptable for running on edge devices. This is useful in situations where a minimal footprint is desirable owing to restricted hardware resources.

The fundamental ideas of NiFi

The core ideas behind NiFi's architecture have a lot in common with the underlying principles behind flow-based programming. The following is a list of some of the most important topics related to NiFi:

Flow File: Each item that is processed by the system is denoted by a FlowFile, and NiFi maintains a map of key/value pair attribute strings and the accompanying content of zero or more bytes for each FlowFile.

FlowFile Processor: Processors are the components that are responsible for carrying out the task. To put it another way, a processor is responsible for some combination of data routing, data transformation, and system-to-system mediation. The properties of a certain FlowFile and the content stream of that file may be accessed by processors. In each given unit of work, processors are

able to perform operations on zero or more FlowFiles and then either commit or rollback the results of those operations.

Connections: Connections are what offer the real connectivity between different processors in a system. These serve as queues, enabling a variety of processes to engage with one another at varying speeds. These queues may have their priorities changed on the fly, and they can also have higher load boundaries, which enables back pressure to be applied.

Flow Controller: The Flow Controller is responsible for retaining the information of how processes relate to one another as well as managing the threads and allocations of those threads that are used by all processes. The Flow Controller performs the role of a broker and makes it possible for processors to trade FlowFiles with one another.

Process Group: Process Group: A Process Group is a particular group of processes and their connections that may accept data via the use of input ports and send data out using output ports. A Process Group is sometimes referred to as a process cluster. In this way, process groups make it possible to generate completely new components only by composing existing components.

The use of this design paradigm results in a number of advantageous outcomes, which contribute to NiFi's status as a highly efficient platform for the construction of robust and scalable dataflows. A few examples of these advantages are as follows:

- Is inherently asynchronous, which allows for very high throughput and natural buffering even as processing and flow rates fluctuate
- Provides a highly concurrent model without a developer having to worry about the typical complexities of concurrency
- Promotes the development of cohesive and loosely coupled components, which can then be reused in other contexts and promotes testable unidirectional dependencies
- Lends itself well to the visual creation and management of directed graphs of processors
- Error management becomes as natural as the happy route rather than a coarse-grained catch-all due to the resource restricted connections.
- Critical functions such as back-pressure and pressure release become highly natural and intuitive.
- It is possible to quickly understand and monitor the flow of data across the system, as well as the points of entry and departure for the data itself.

Chapter-13: Apache Phoenix

Overview

Apache Phoenix is an open source, massively parallel, relational database engine supporting OLTP for Hadoop using Apache HBase as its backing store. Phoenix provides a JDBC driver that hides the intricacies of the NoSQL store enabling users to create, delete, and alter SQL tables, views, indexes, and sequences; insert and delete rows singly and in bulk; and query data through SQL. Phoenix compiles queries and other statements into native NoSQL store APIs rather than using MapReduce enabling the building of low latency applications on top of NoSQL stores. Apache Phoenix OLTP and operational analytics for Apache Hadoop. Apache Phoenix enables OLTP and operational analytics in Hadoop for low latency applications by combining the best of both worlds:

1. The power of standard SQL and JDBC APIs with full ACID transaction capabilities.

About Apache Phoenix

Apache Phoenix is an SQL layer for Apache HBase and provides a programming ANSI SQL interface. Using Apache Phoenix, you can create and interact with tables in the form of typical DDL/DML statements using the Phoenix standard JDBC API. Apache Phoenix is converting HBase into SQL Databases. HBase, is a distributed NoSQL store and if you need OLTP and Analytics over HBase than Phoenix.

Phoenix enables OLTP and operational analytics in Hadoop for low latency applications combining the best of both worlds.

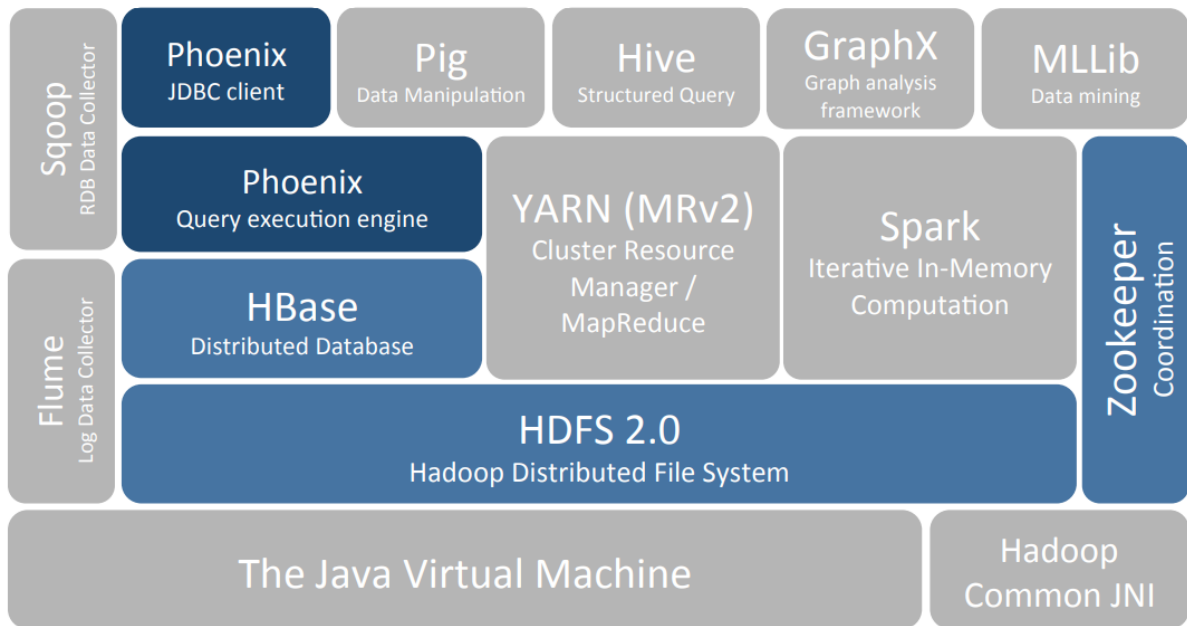
- The Power of standard SQL and JDBC APIs with full ACID transaction capabilities.
- The flexibility of late-bound, schema-on-read capabilities from the NoSQL world by leveraging HBase as its backing store.
- Apache Phoenix has Embedded JDBC Driver which implements the majority of java.sql interfaces, including metadata API's.
- Apache Phoenix allows columns to be modelled as a multi-part row key or key/value cells.
- Full query support with predicate push down and optimal scan key formation.
- DDL support: CREATE TABLE, DROP TABLE, and ALTER TABLE for adding/removing columns.
- Versioned schema repository. Snapshot queries use the schema that was in place when data was written.
- DML support: UPSERT VALUES for row-by-row insertion, UPSERT SELECT for mass data transfer between the same or different tables, and DELETE for deleting rows.

Relational Layer

- Apache Phoenix is a relational layer for Apache HBase
- **Query Engine:**
 - Transform SQL Queries and parses into native HBase API calls. This is in-directly MapReduce.
 - Apache Phoenix pushes as much work as possible onto the cluster for parallel execution.
 - **Metadata Repository:** This is a Phoenix table itself which helps in typed access to data stored in HBase tables. It stores tables, views, sequence definitions, secondary indexes. For your perspective it's a JDBC Driver.

Apache Phoenix Integration with Hadoop

Apache Phoenix is fully integrated with other Hadoop products such as Spark, Hive, Pig Flume and MapReduce. For your perspective Apache Phoenix is just like a JDBC driver.



Apache HBase

Apache HBase is a high performance horizontally scalable datastore engine for BigData, suitable as the store of record for mission critical data.

Phoenix and SQL

- Accessing HBase data with Phoenix can be substantially faster than direct HBase API use.
- Phoenix parallelizes queries based on stats. HBase does not know how to chunk queries beyond scanning an entire region.
- Phoenix pushes processing to the server.
- If you write your own API call, this may not use coprocessors.
- Phoenix has a huge difference for aggregations vs direct HBase API calls.
- Phoenix supports and uses secondary indexes.

Apache Phoenix takes your SQL Query, compiles into a series of HBase scans, and orchestrate the running of those scans to produce regular JDBC result sets. Direct use of the HBase API, along with coprocessors and custom filters, results in performance on the order of milliseconds for small queries, or seconds for tens of millions of rows.

All standard SQL constructs are supported, including SELECT, FROM, WHERE, GROUP BY, HAVING, ORDER BY etc.

Apache Phoenix also supports a full set of DML commands as well as table creation and versioned incremental alterations through our DDL commands.

Phoenix not supported SQL Construct

Below is the list of constructs which are currently not supported.

- Relational Operators: Intersect, Minus
- Miscellaneous Built-In functions:

Phoenix Knobs and Dials

Phoenix provides many different knobs and dials to configure and tune the system to run more optimally on your cluster. The configuration is done through a series of Phoenix-specific properties specified for the most part in your client-side hbase-site.xml file. In addition to these properties, there are of course all the HBase configuration properties.

Cloudera Operational Database

Cloudera Operational Datastore is a real-time auto-scaling operational database powered by Apache HBase and Apache Phoenix. COD is an experience which runs in CDP. Cloudera Operational Database experience allows self-service creation and management of an operational database. You can provision a new database with a single click, build application against it and deploy it on the public cloud without complexity.

Apache Phoenix Use Cases

1. We can use Apache Phoenix for storing data as a basis for measuring activities and generating reports. You should choose Phoenix because it provides the scalability of HBase and the expressiveness of SQL.
2. Phoenix can be used for on Demand Data aggregations. If you have floating time range of

Contents

Chapter-1: Apache HDFS.....	1
Overview	1
HDFS cluster components and their respective roles.....	1
Advantages of using HDFS	2
NameNodes.....	2
DataNodes.....	2
JournalNodes	3
The Architecture of HDFS.....	3
Read Operations on the HDFS.....	5
HDFS Operation for Writing	6
HDFS FAQ	8

Chapter-1: Apache HDFS

Overview

Hadoop Distributed File System, often known as HDFS, is a file system that is based on Java and offers scalable data storage.

- **NameNode:** The NameNode of an HDFS cluster is responsible for managing the namespace of the cluster.
- **DataNode:** while the DataNodes are used to store data.

HDFS was designed to span huge clusters of commodity systems. The Hadoop Distributed File System (HDFS) serves as the platform's data management layer. YARN is responsible for the administration of the resources, whereas HDFS is in charge of storage.

HDFS is a distributed storage system that is scalable, fault-tolerant, and works closely with a broad range of applications that access data concurrently. By spreading storage and computing among a large number of servers, the total storage resource may expand in a linear fashion in response to increased demand.

Because HDFS can be scaled up or down depending on the requirements, it is truly quite difficult to find any substitute for HDFS that is suitable for storing Big Data. Hadoop is being used by a significant number of the largest corporations in the world. Hadoop is used by several companies, including Amazon, Facebook, Microsoft, Google, Yahoo, IBM, and General Electrics, to store and analyse enormous volumes of data.

HDFS cluster components and their respective roles

The primary components of an HDFS cluster are referred to as the NameNode and the DataNodes respectively.

The NameNode: is responsible for managing the metadata of the cluster, which includes the file and directory hierarchies, rights, changes, and quotas for disc space. The contents of the file are separated into several data blocks, with each block being copied at a number of different DataNodes.

The NameNode keeps monitoring on the total number of blocks that have been replicated. In addition to this, the NameNode stores the full namespace image in RAM and is responsible for maintaining the namespace tree as well as the mapping of blocks to DataNodes.

In order to prevent single point of failure, High-Availability (HA) clusters include a backup NameNode for the current one. And clusters synchronise the active and standby NameNodes by using JournalNodes in the process.

Advantages of using HDFS

The following are some of the advantages provided by HDFS, which are directly responsible for the effective storage and high availability of data inside the cluster:

- **Rack awareness:** refers to the physical location of a node while assigning storage and scheduling jobs.
- **Hadoop reduces the amount of data that has to be moved by moving computational** activities directly to HDFS. Whatever computation Hadoop initiates sends all the computation near to data i.e. on DataNode rather than getting data to the compute node, this is the biggest advantage and reduce the network traffic. This results in a very considerable reduction in the overall amount of network I/O and offers extremely high aggregate bandwidth.
- **Utilities:** Conduct a real-time analysis of the state of the file system's health and rebalance the data throughout the various nodes.
- **Standby NameNode** is a NameNode that offers high availability and provides redundancy (HA).

NameNodes

NameNodes are responsible for maintaining the HDFS namespace tree as well as a mapping of file blocks to the DataNodes that are used to store the data.

Only one main NameNode is necessary for a basic HDFS cluster. This primary NameNode is backed up by a backup NameNode that compresses the NameNode edits log file on a periodic basis. The NameNode edits log file is a list of HDFS metadata alterations. Because of this, the amount of disc space used by the log file on the NameNode is decreased, which in turn results in a shorter length of time required to restart the main NameNode. There are always two NameNodes present in a high-availability cluster: an active and a backup.

DataNodes

NameNode daemon is responsible for managing the data in a Hadoop cluster, and DataNodes are the nodes that hold the data. The data in a file is copied and stored on several DataNodes to ensure its integrity and to facilitate the execution of localised computations in close proximity to the data.

It is important for a cluster's DataNodes to have a consistent appearance. Problems may arise if they are not consistent with one another. For instance, jobs may fail to complete if DataNodes have a

lower total amount of memory because they reach capacity more rapidly than DataNodes with a higher total amount of memory.

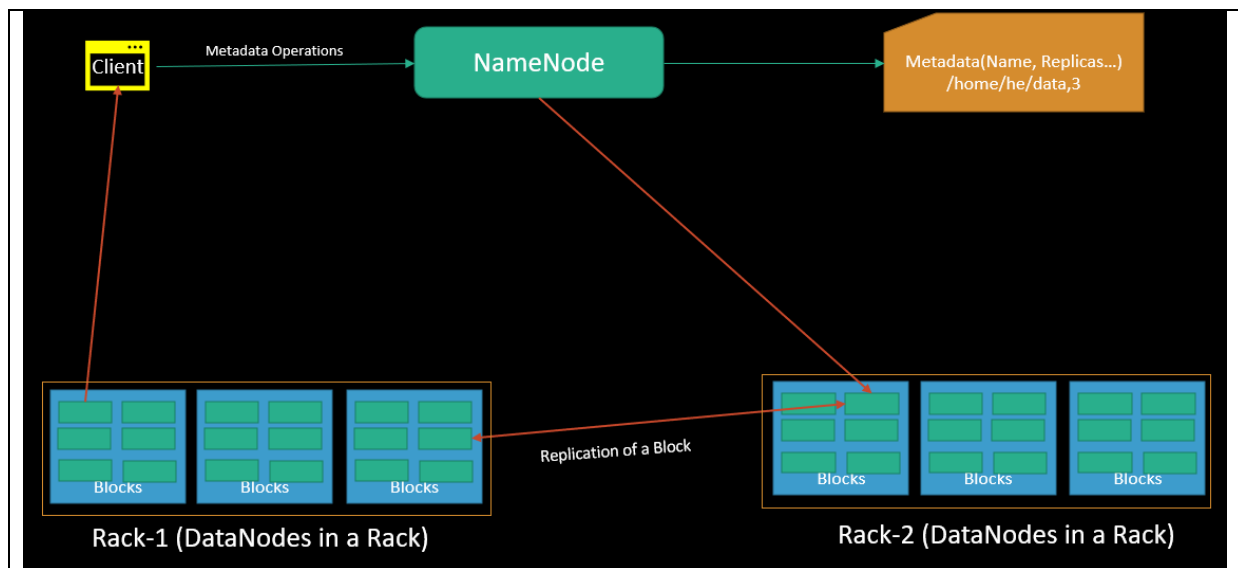
Important: HDFS was set up with a replication factor of three by default. That is, there are always three copies of the data kept on hand at all times. When you have at least three DataNodes, Cloudera recommends that you should not select a lower replication factor than the default value of three. Data loss might result from having a lower replication factor.

JournalNodes

JournalNodes are used to keep active and backup NameNodes in high-availability clusters synchronised with one another. The active NameNode is the one that makes changes to the HDFS namespace information and publishes such "edits" to each JournalNode. Whenever there is a failover, the standby NameNode will promote itself to the active state after first applying all of the updates that were made in the JournalNodes.

The Architecture of HDFS

There is only one NameNode that is responsible for storing metadata, whereas numerous DataNodes are in charge of doing the actual storage operations. In order to offer fault tolerance, the nodes in the cluster are organised into racks. Additionally, copies of data blocks are kept on separate racks within the cluster.



NameNode holds metadata, whereas DataNode contains real data. Due to the fact that NameNode is the central component of the cluster, all interactions between the client and the cluster must take place via NameNode.

Within the cluster, there are a number of DataNodes, each of which has its own local disc on which it stores HDFS data. DataNode will occasionally communicate through a "heartbeat" message to NameNode to let it know that it is still active. In addition to this, it copies the data to additional DataNodes in accordance with the replication factor.

Hadoop File System (HDFS) Features

Let's have a look at some of Hadoop Distributed File System's more interesting features right here in the HDFS lesson.

Storage That Is Distributed: HDFS stores data in a distributed fashion. It then saves each individual piece of data on a distinct DataNode inside the cluster after first dividing the data into smaller chunks. In this approach, the Hadoop Distributed File System provides MapReduce with a means by which it may process a fraction of enormous data sets that have been partitioned into blocks in a manner that is parallel across numerous nodes. Hadoop revolves on the MapReduce programming model, while HDFS is the component that makes all of these other features possible.

Blocks: HDFS breaks down massive files into manageable sections called as blocks. The smallest unit of data that may be stored in a filesystem is called a block. As a client or admin, you do not have any influence on the block's properties, such as its placement. NameNode is the one that makes all of these decisions.

HDFS default block size is 128 MB. We have the ability to adjust the size of the block to meet our specific requirements. This is in contrast to the filesystem used by the operating system, where the block size is 4 KB.

If the data size is less than the HDFS block size, then the block size will be equal to the data size.

If the size of the file, for instance, is 129 megabytes, then there will be 2 blocks produced for it. The default size of one block will be 128 megabytes, while the other block will only be one megabyte since using 128 megabytes would be a waste of space. Hadoop is smart enough to avoid wasting the remaining 127 megabytes of storage space. Therefore, it is only allocating a 1 MB block for 1 MB worth of data.

A significant benefit of storing data in such a block size is that it reduces the amount of time spent seeking on the disc. Another benefit is that the mapper only processes one block at a time when it performs its operations. So, one mapper handles enormous data at a time.

The Act of Repeating: Hadoop HDFS produces two identical copies of each block it stores. This process is referred to as replication. The data for each block is copied and kept on separate DataNodes distributed across the cluster. It tries to store at least one duplicate on a distinct rack for each original.

What exactly is a Rack?

Racks are used to organise the DataNodes. A single switch connects all of the nodes in a rack. Hence, data may be accessed from another rack even if an individual switch or the whole rack becomes inoperable.

As previously discussed, the default replication factor is 3, but it is possible to alter this to the appropriate values according to the demand by modifying the configuration files (hdfs-site.xml).

High Availability: Data availability can be achieved by replicating data blocks and storing them on numerous nodes distributed across the cluster.

Data Reliability: Data is replicated in HDFS, as we have seen previously. Because of replication, blocks maintain a high availability even in the event that a node or piece of hardware fails. In the event that the DataNode fails, the block will still be available from any other DataNode that has a

duplicate of the block. Additionally, even if the rack collapses, the block will still be accessible on the alternative rack. This is how HDFS ensures the integrity of its data storage while also providing fault tolerance and high availability.

Fault Tolerance: Hadoop and the other components of the ecosystem may benefit from the fault-tolerant storage layer that HDFS offers.

HDFS is designed to run on commodity hardware, which refers to systems that have average configurations and a high likelihood of crashing at any given moment. Because of this, the HDFS system duplicates data and stores it in many locations. This helps to ensure that the system as a whole is very resilient to errors.

Scalability: Scalability refers to the capacity to increase or decrease the size of the cluster. There are two methods in which we can grow Hadoop HDFS.

- **Vertical Expansion or scalability:** We are able to install more drives on the data nodes. In order to do this, we need to make changes to the configuration files and add entries that match to the newly inserted drives. Increasing disk size is a vertical scalability.
- **Horizontal Scalability:** is an additional method of scalability that consists of the capability of adding more nodes (Data Nodes) to the cluster dynamically and without incurring any downtime. This technique is referred to as horizontal scaling. We are able to add as many nodes as we like to the cluster at any one moment, in real time, without experiencing any kind of outage.

Access to application data with a high throughput: The Hadoop Distributed File System enables users to access application data with a high throughput. The quantity of work completed in a certain length of time is referred to as the throughput. It is often used as a method for measuring the system's overall performance, and it provides a description of the rate at which data is retrieved from the system.

When we wish to carry out a procedure or an operation using HDFS, the work is partitioned and distributed over a number of different computers. Therefore, each of the systems will independently and simultaneously carry out the duties that have been allocated to them. Because of this, the task will be finished in a relatively short amount of time. Therefore, HDFS provides a strong throughput because of its parallel data reading capabilities.

Read Operations on the HDFS

When a client wants to read any file from HDFS, the client has to communicate with NameNode since NameNode is the only location that keeps metadata about DataNodes. This means that whenever a client wants to read any file from HDFS, the client needs to interact with NameNode. NameNode is responsible for specifying the address of the slaves or the place where the data is kept.

The client will engage in interaction with the DataNodes that have been configured and read the data from those locations. In order to ensure the client's authenticity and safety, the NameNode sends it a token, which the client then presents to the DataNode before beginning to read the file.

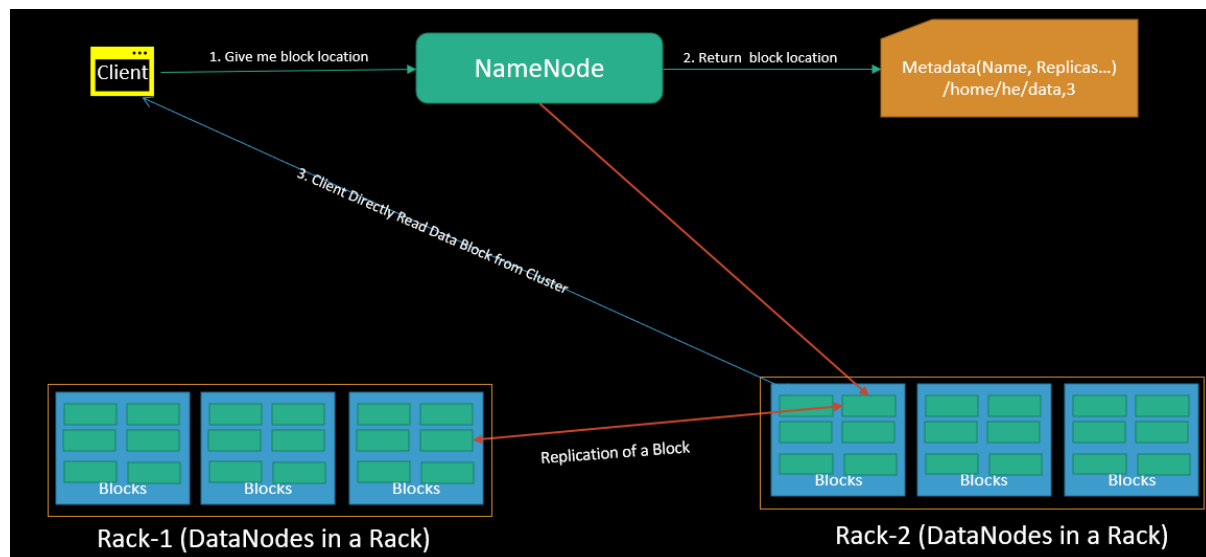
In the Hadoop HDFS read operation, the client must first interact with the NameNode in order to read data that is stored in HDFS if the client wishes to read data that is stored in HDFS. Therefore, the client engages in interaction with the API of the distributed file system and submits a request to

NameNode to provide the block location. As a result, NameNode examines the client to determine whether or not they have enough credentials to access the data. If the client has the necessary credentials, then the NameNode will provide the address of the DataNode at which the data is kept.

Along with the address, NameNode also provides the client with a security token. In order to access the data, the client is required to provide the security token to DataNode for the purposes of authentication.

When a client travels to DataNode for the purpose of reading the file, DataNode first checks the token, and then it grants permission to the client to read that specific block. Following this step, a client will access the input stream and begin reading data from the DataNodes that have been configured. The client obtains the data in this fashion by reading it straight from the DataNode.

In the event that the DataNode unexpectedly goes down while a file is being read, a client will once again travel to the NameNode, and the NameNode will share another location with the client where that block may be found.

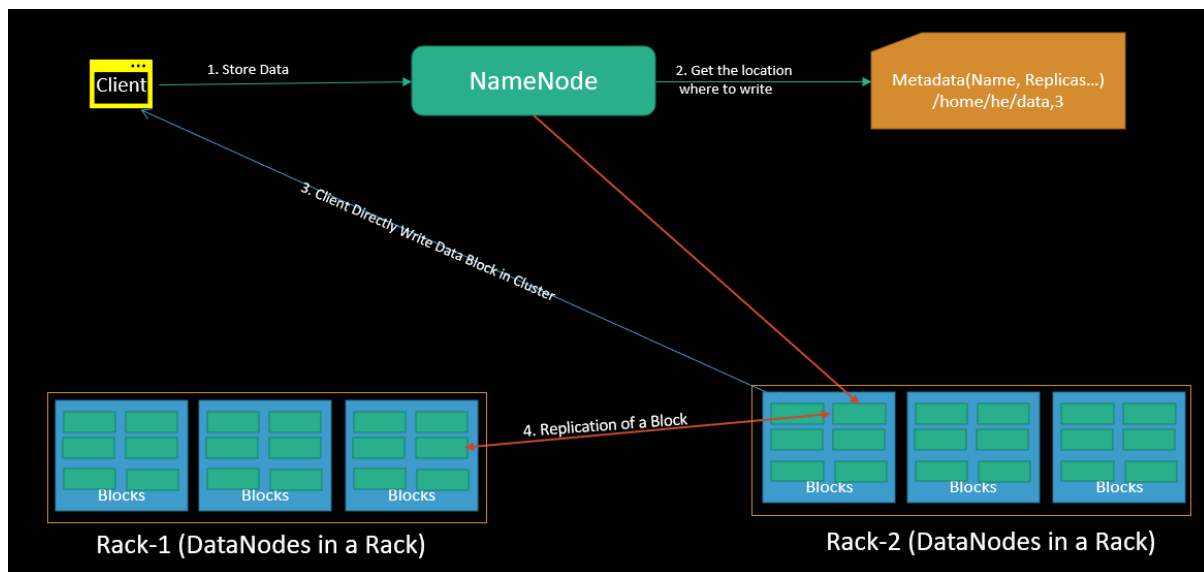


HDFS Operation for Writing

As can be observed when the client is reading a file, it is necessary for the client to interface with NameNode. In a similar way, in order for the client to write a file, they need to communicate with the NameNode.

NameNode gives the client the address of the slaves on which data has to be written in order for the client to complete the transaction.

After the client has completed writing the block, the slave will begin copying the block into another slave, which will then copy the block and send it to the third slave. When the standard replication factor of three is used, this is the result that is obtained. Once all of the necessary replications have been performed, it will send a final acknowledgement to the client.



Any time a client wants to write any data, it is required to communicate with the NameNode in order to do so. The client then communicates with the API for the distributed file system and requests that NameNode transmit a slave location.

The client will then begin writing the data by interacting with the DataNode at where the data has to be written and will begin writing the data via the FS data output stream. Following the completion of writing and replicating the data, the DataNode will send an acknowledgement to the client alerting them that the data has been written in its entirety.

When the client has finished writing to the first block, the first DataNode will immediately duplicate that block to any additional DataNodes that are connected to it. Therefore, after it has received the block, DataNode will begin the process of copying that block to the third DataNode. The third DataNode will send an acknowledgement to the second DataNode, the second DataNode will send an acknowledgment to the first DataNode, and finally, the first DataNode will send the final acknowledgment.

The client only sends one copy of the data regardless of the replication factor, but the DataNodes are responsible for replicating the blocks. Therefore, writing a file on Hadoop's HDFS does not incur any additional costs since many blocks of the file are written in parallel across various DataNodes.

Utilizing Cloudera Manager to relocate the JournalNode, which updates the directory for a role group: You have the ability to modify the location of the edit's directory for each JournalNode that is a part of the JournalNode Default Group, depending on the needs of your organisation.

By using Cloudera Manager, you may move the JournalNode edits directory for a role instance: You are free to adjust the location of the edits directory for one JournalNode instance in accordance with the specifications of your project.

Bringing the contents of JournalNodes into synchronisation: You have the ability to synchronise the data that is included inside the JournalNodes that are part of your CDP Private Cloud Base cluster. When this feature is enabled, it helps to preserve consistency in the contents of all of the JournalNodes that are distributed across the cluster. For instance, if the contents of a JournalNode become inconsistent, it is possible for that JournalNode to automatically duplicate the contents of the other JournalNodes in the cluster in order to restore consistency.

HDFS FAQ

Question-1: How NameNode handles the management of blocks on a DataNode that has failed?

Answer: After a certain amount of time has passed without any heartbeats, a DataNode is said to be dead.

Question-2: To replace a disc on a DataNode host, follow these steps?

Answer: You have the ability to repair defective discs that are hosted on the DataNode in your CDP Private Cloud Base cluster. Before you can replace the malfunctioning disc, all managed services need to be stopped, and the DataNode role instance has to be decommissioned.

Question-3: How do you take out one of the DataNodes?

Answer: Make sure that all of the conditions for deleting a DataNode have been completed before you attempt to remove it.

Question-4: How do you put an end to irregularities in the blocks?

Answer: You may get information on inconsistencies with the HDFS data blocks by using the output of the `hdfs fsck` or `hdfs dfsadmin -report` commands. These inconsistencies include missing, misreplicated, or underreplicated blocks. You have the flexibility to choose from a variety of approaches to remedy these discrepancies.

Question-5: How do you use the Cloudera Manager to add storage directories?

Answer: Using Cloudera Manager, you may create a new storage directory and choose the kind of storage the directory will use.

Question-6: How do you use Cloudera Manager, get rid of the storage folders?

Answer: Using Cloudera Manager, you are able to delete already existing storage folders and define new directories.

Question-7: How do you set up the storage balancing configuration for DataNodes?

Answer: You have the ability to configure HDFS to distribute writes on each DataNode in a way that maintains a consistent level of available storage across all disc volumes on that DataNode.

Question-8: How do you utilize Cloudera Manager, carry out a disc hot swap on the DataNodes?

Answer: You won't need to restart a DataNode in order to change discs on the CDP Private Cloud Base cluster you're using. The term for this practise is "hot switch."

Question-9: What is HDFS used for?

Answer: Hadoop Distributed File System also known as HDFS is used for storing structure and unstructured data in distributed manner by using commodity hardware.

Question-10: What is Hadoop Distributed File System and what are its components?

Answer: Hadoop HDFS is a distributed file-system that stores data on commodity machines, providing very high aggregate bandwidth across the cluster.

Components of HDFS: HDFS comprises of 3 important components NameNode, DataNode and Secondary NameNode.

HDFS operates on a Master-Slave architecture model where the NameNode acts as the master node for keeping a track of the storage cluster and the DataNode acts as a slave node summing up to the various systems within a Hadoop cluster.

Question-11: What is NameNode and DataNode in HDFS?

Answer: Namenode is the master and DataNodes are slaves NameNode manages the filesystem namespace. It maintains the filesystem tree and the metadata for all the files and directories in the tree.

DataNodes are the workhorses of the filesystem. They store and retrieve blocks when they are told to (by clients or the NameNode), and they report back to the NameNode periodically with lists of Blocks that they are storing. Without the NameNode, the filesystem cannot be used.

Question-12: Why Hadoop uses filesystem for storage?

Answer: HDFS is built to support applications with large data sets, including individual files that reach into the terabytes. File systems are more affordable to handle huge amount of data.

Question-13: What is meant by Data node?

Answer: Data node is the slave deployed in each of the systems and provides the actual storage locations and serves read and writer requests for clients.

Question-14: What is daemon?

Answer: Daemon is the process that runs in background in the UNIX environment. In Windows it is 'services' and in DOS it is 'TSR'.

Question-15: What is meant by heartbeat in HDFS?

Answer: Data nodes and task trackers send heartbeat signals to Name node and Job tracker respectively to inform that they are alive. If the signal is not received it would indicate problems with the node or task tracker.

Question-16: Is it necessary that Name node and job tracker should be on the same host?

Answer: No! They can be on different hosts.

Question-17: What is meant by 'block' in HDFS?

Answer: Block in HDFS refers to minimum quantum of data for reading or writing. Default block size is 128 MB in HDFS.

Question-18: Can blocks be broken down by HDFS if a machine does not have the capacity to copy as many blocks as the user wants?

Answer: Blocks in HDFS cannot be broken. Master node calculates the required space and how data would be transferred to a machine having lower space.

Question-19: How is data replicated in HDFS?

Answer: HDFS is designed to be fault-tolerant. Large HDFS data files are split into smaller chunks called blocks and each block is stored in multiple DataNodes across the cluster. The block size and the replication factor can be configured per file.

HDFS is rack aware for multi-clustered environments, and takes this into consideration when replicating blocks for fault-tolerance. HDFS ensures that the blocks are replicated on DataNodes that are on different racks, so if a rack goes down the data is still available from the DataNode on the other rack.

Question-20: Explain how indexing in HDFS is done?

Answer: Hadoop has a unique way of indexing. Once the data is stored as per the block size, the HDFS will keep on storing the last part of the data which say where the next part of the data will be.



Contents

Chapter-2 Apache Ozone	1
Overview	1
Ozone architecture	3
Advanced Concepts.....	7

Chapter-2 Apache Ozone

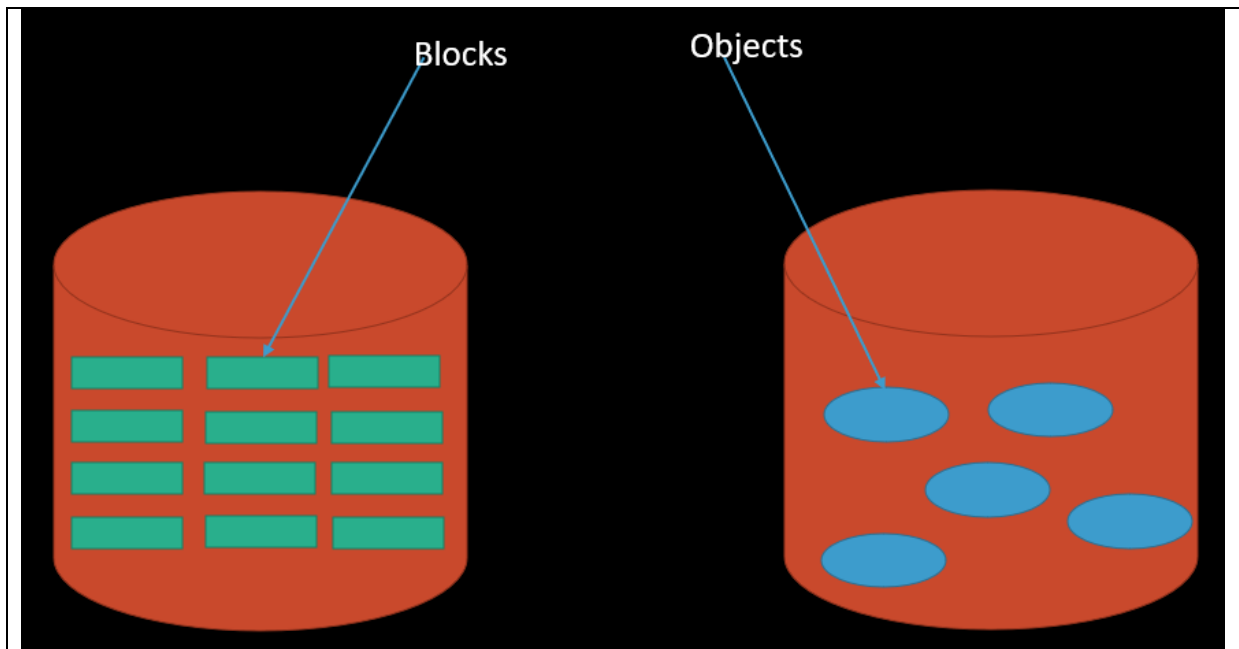
Overview

Apache Ozone is a tool or product that is used to implement Hadoop's Object Storage architecture. As we know that Hadoop Distributed File System (HDFS) is used as a native storage for storing data in Hadoop ecosystem. However, Hadoop Distributed File System (HDFS) is a block storage system that was created by Apache and designed to be used as a storage layer on the Hadoop architecture.

Object storage is the name given to the new storage model that Ozone, that is also a storage engine for Hadoop ecosystem. Within the same Hadoop cluster, it can co-exist with HDFS to offer file store and object store capability. Similar to AWS S3 and Google Cloud storage, you can create object storage for your Hadoop Cluster as well and it can store trillions of files on Ozone, and they can access those files just as if they were stored on HDFS.

Ozone also supports the scalability and tiny file issues that HDFS has. Ozone can be easily integrated into already existing Hadoop installations, and applications like Hive and Spark can run without requiring any adjustments.

Apache has built an object storage system known as Ozone. It is designed to be used inside the architecture in a manner similar to that of HDFS. Specifically, as a storage layer. Below image shows Comparison between Object Storage and Block Storage (HDFS) in graphical form (Ozone).



Block storage: Fragments the data into smaller chunks, which are subsequently stored independently as blocks. The Storage Area Network (SAN) will position the data blocks in the locations where they will be used most effectively. Each block is assigned a unique identification. Because of this, data may be saved wherever it will be more easily accessible, rather than inside the same system. The access to data in block storage does not depend on a single, centralised route. Because of this, the information may be accessed very rapidly.

Storage of Objects: Files are fragmented into smaller parts and dispersed over various devices as part of the object storage system, which has a flat structure. Volumes and containers are used for object storage in place of traditional blocks. The data is stored in volumes, which are modular storage containers that are completely self-contained. Every item included inside it is assigned a distinct identification in addition to the information that details the data. The unique identifier in block storage is made up of two IDs, as opposed to only one. One that tells you which bucket the data is kept on and where it is stored.

And another one that specifies the location inside the bucket where the data is being kept. The metadata may be tailored and specified to the user's specifications. It includes information such as age, access permissions, and security. Object Storage offers excellent value for the money. You are only responsible for paying for what you use. It is not difficult to scale up.

Large quantities of static and unstructured data are ideal candidates for this sort of storage. Once they have been generated, objects cannot have their properties changed in any way, which is one of the most significant drawbacks. It is required to create a whole new object if you want to make changes to an existing one. Writing objects is a more time-consuming procedure than writing to block storage, which is another reason why standard structured databases do not perform well with object storage.

Apache Ozone is a distributed object store that is scalable, redundant, and optimised for the demands associated with large data. Applications that make use of frameworks such as Apache Spark, Apache YARN, and Apache Hive function natively on Ozone without requiring any changes. This is in addition to Ozone's ability to scale to billions of objects of varied sizes. Ozone offers a

Hadoop-compatible file system interface and has native support for the S3 application programming interface. The CDP Private Cloud Base deployment is the common location where Ozone may be found.

Ozone is made up of three essential components that are used for storing information: volumes, buckets, and keys. Each key is a component of a bucket, and the buckets together make up a volume. The creation of volumes is restricted to administrators only. Regular users are able to build buckets in varying quantities, depending on the needs that they have. Within these buckets, Ozone maintains data in the form of keys.

Ozone saves the accompanying data on DataNodes in chunks that are referred to as blocks. This happens whenever a client submits a key. As a result, any key may open anywhere from one to several different chests. Multiple blocks that are not connected to one another may coexist in the same storage container inside of a DataNode.

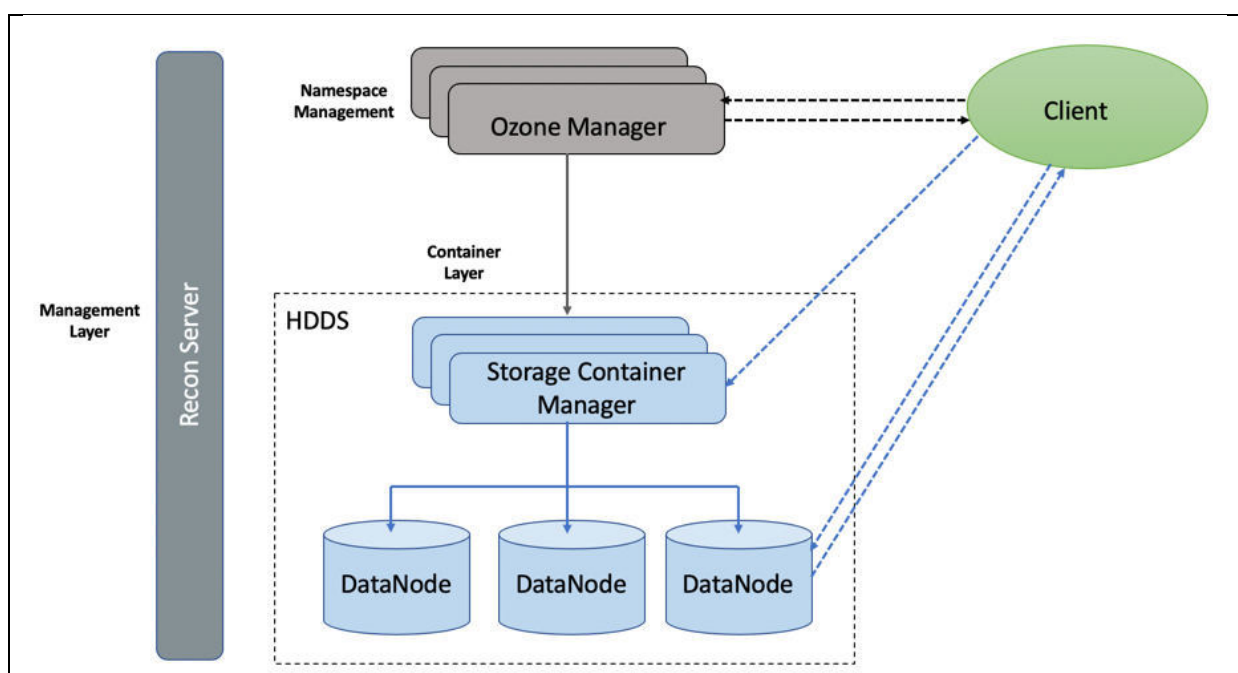
HDFS is the de facto large data file system. It's easy to forget how scalable and reliable HDFS is. Our customers operate clusters with thousands of nodes that serve thousands of concurrent clients.

HDFS operates best with huge files, tens to hundreds of megabytes. HDFS has a modest file capacity and struggles with 400M files. HDFS-like storage that can handle billions of little files is in demand. Ozone can handle tiny and big files. Ozone is an Object Store whereas HDFS is POSIX-like.

Ozone architecture

Ozone can be co-located with HDFS with single security and governance policies for easy data exchange or migration and also offers seamless application portability. Ozone has a scale-out architecture with minimal operational overheads. Ozone separates management of namespaces and storage, helping it to scale effectively. The Ozone Manager (OM) manages the namespaces while the Storage Container Manager (SCM) handles the containers.

The following diagram shows the components that form the basic architecture of Ozone:



Hadoop Distributed Data Store: Ozone is built on a highly available, replicated block storage layer called Hadoop Distributed Data Store (HDDS).

Blocks: Blocks are the basic unit of storage. In Ozone, each block is of 256 MB in size. A collection of blocks forms a storage container. The SCM allocates blocks inside storage containers for the client to store data.

Storage Containers: A storage container is a group of unrelated blocks managed together as a single entity. A container exists in a DataNode and is the basic unit of replication, with a capacity of 2 GB to 16 GB.

The Storage Container Manager performs multiple critical functions for an Ozone cluster. SCM manages the addition and removal of DataNodes, and allocates storage containers and blocks. SCM also manages block collections, ensuring that the blocks maintain the required level of replication. SCM allocates blocks to clients through OM for read and write operations. In addition, SCM executes recovery actions when faced with DataNode or disk failures.

DataNodes: DataNodes contain storage containers comprising of data blocks. The SCM monitors DataNodes through heartbeats.

Ozone Manager: The Ozone Manager (OM) is the metadata manager for Ozone. The OM manages the following storage elements:

- The list of volumes for each user
- The list of buckets for each volume
- The list of keys for each bucket

The OM maintains the mappings between keys and their corresponding Block IDs. When a client application requests for keys to perform read and write operations, the OM interacts with the SCM for information about blocks relevant to the read and write operations, and provides this information to the client. In addition, the OM also handles metadata operations from the clients.

The Ozone Manager (OM) is a highly available namespace manager for Ozone. OM manages the metadata for volumes, buckets, and keys. OM maintains the mappings between keys and their corresponding Block IDs. When a client application requests for keys to perform read and write operations, OM interacts with SCM for information about blocks relevant to the read and write operations, and provides this information to the client. In addition, OM also handles metadata operations from the clients.

Pipelines: Pipelines determine the replication strategy for the blocks associated with a write operation.

Recon Server: Recon is the management interface for Ozone. Recon provides a unified management API for Ozone.

How Ozone manages read operations: The client requests the block locations corresponding to the key it wants to read. The Ozone Manager (OM) returns the block locations if the client has the required read privileges.

The following steps explain how Ozone manages read operations:

1. The client requests OM for block locations corresponding to the key to read.
2. OM checks the ACLs to confirm whether the client has the required privileges, and returns the block locations and the block token that allows the client to read data from the DataNodes.
3. The client connects to the DataNode associated with the returned Block ID and reads the data blocks.

How Ozone manages write operations: The client requests blocks from the Ozone Manager (OM) to write a key. OM returns the Block ID and the corresponding DataNodes for the client to write data.

The following steps explain how Ozone manages write operations:

1. The client requests blocks from OM to write a key. The request includes the key, the pipeline type, and the replication count.
2. OM finds the blocks that match the request from SCM and returns them to the client.

If security is enabled on the cluster, OM also provides a block token along with the block location to the client. The client uses the block token to connect to the DataNodes and send the command to write chunks of data.

1. The client connects to the DataNodes associated with the returned block information and writes the data.
2. After writing the data, the client updates the block information on OM by sending a commit request.
3. OM records the associated key information.

Notes

- a. Keys in Ozone are not visible until OM commits the block information associated with the keys. The client is responsible for sending the key-block information to OM after it has written the blocks on the DNs via a commit request.
- b. If OM fails to commit block information for keys after they have been written, for example, client was unable to send the commit request OM because the write job failed, the keys would not be visible but the data would remain on disk.

Tenets: Ozone's design followed these guidelines:

- **Consistent:** Consistency facilitates app development. Ozone is serializable.
- **Simplicity:** A basic architecture is simpler to understand and debug. We've kept Ozone's architecture basic despite its scalability. Ozone scales well. Over 100 billion items may be stored in a single cluster.
- **Layered Architecture:** Ozone is a layered file system for current storage systems. It isolates namespace management from block and node management, allowing scaling on both axes.
- **Pain-free recovery:** HDFS can recover from cluster-wide power outage without losing data or requiring costly recovery methods. Losses in racks and nodes are negligible. Ozone will withstand failures similarly.
- **Apache's Open Source:** Apache Open Source is crucial to Ozone's success. The Apache Hadoop community designs and develops Ozone.
- **Hadoop interoperability:** Ozone should work with current Apache Hadoop applications like Hive, Spark, and MapReduce. Therefore, Ozone:
 - Hadoop FSA (aka OzoneFS). Hive, Spark, etc. may utilise Ozone without modification.

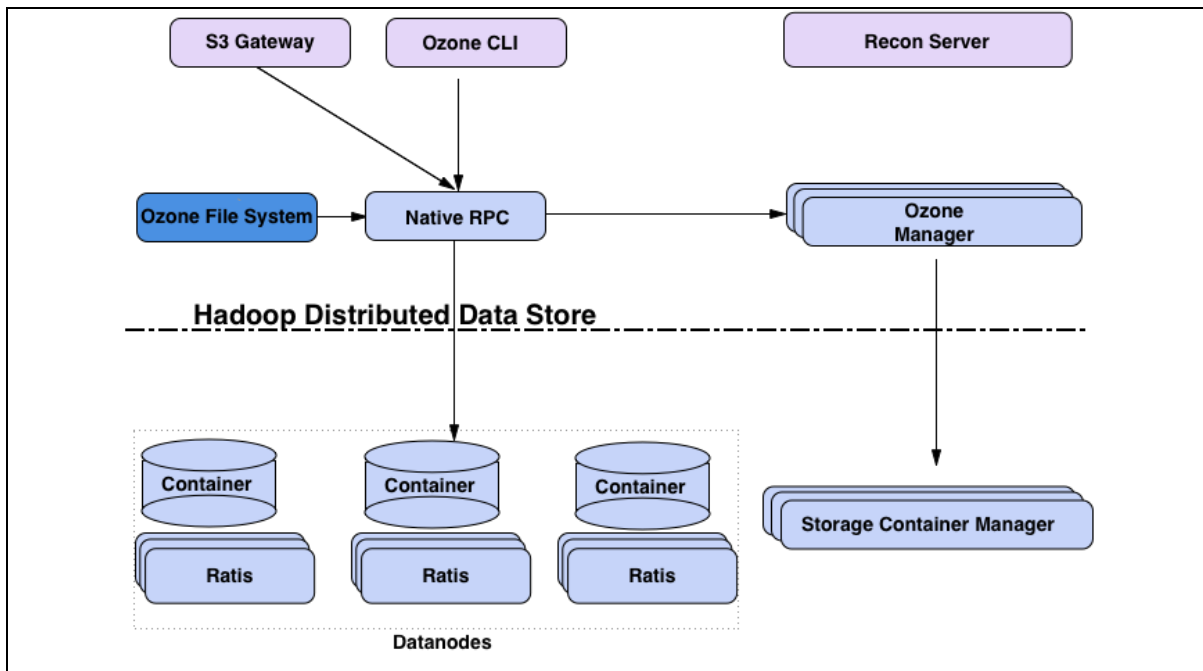
- Localization. Original HDFS/MapReduce allowed computation operations to be scheduled on the same nodes as the data. Ozone supports application data localization.
- Deploy HDFS side-by-side. Ozone may share HDFS discs in an existing Hadoop cluster.

Ozone Concepts: Volumes, buckets, and Keys are the component parts that make up ozone.

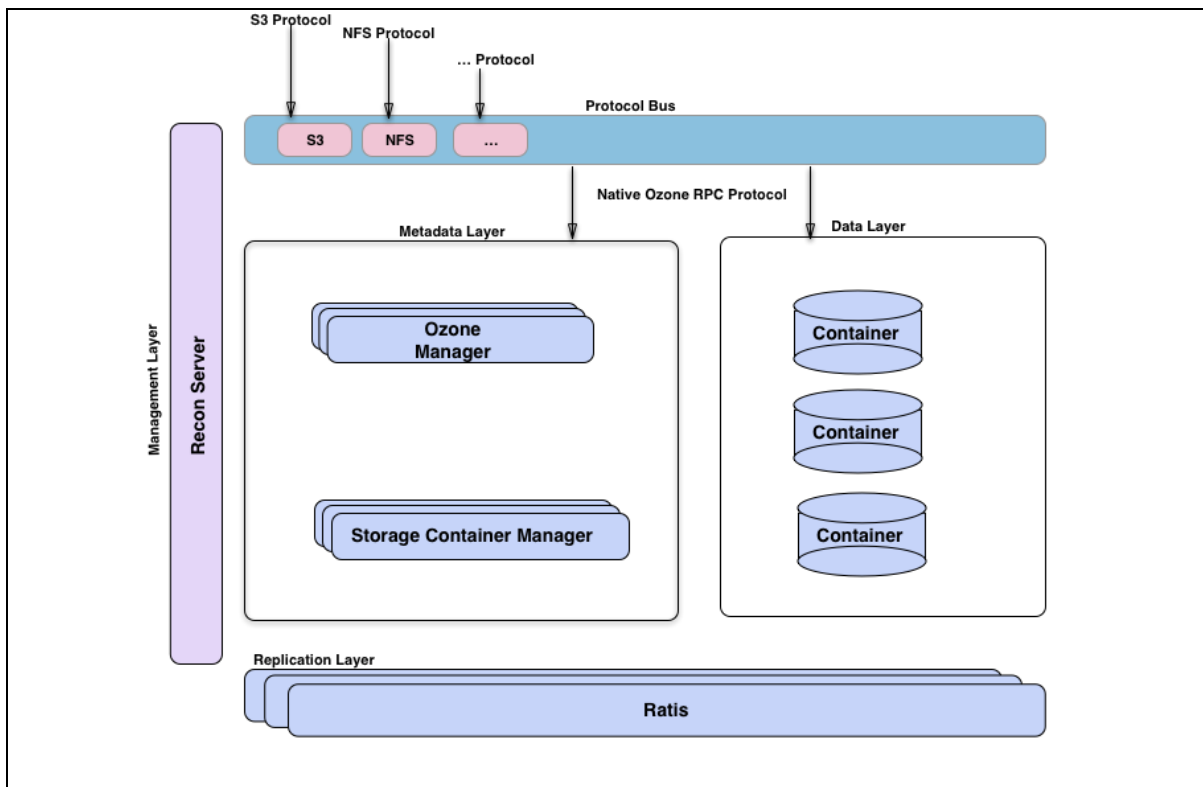
Ozone is an extensible, fault-tolerant, and distributed object storage that is built on top of Hadoop. In addition to being able to handle billions of items of varied sizes, Ozone is also capable of performing well in containerized settings such as those provided by Kubernetes. When Ozone is used, applications like as Apache Spark, Hive, and YARN function normally and do not need any adjustments. It is quite simple to use Ozone due to the fact that it comes equipped with a Java client library, support for the S3 protocol, and a command line interface.

- **Volumes:** Volumes are comparable to accounts in their function. Administrators are the only people who can create new volumes or remove existing ones. The creation of a volume for an organisation or team will normally be handled by an administrator.
- **Buckets:** A volume may have zero, one, or more buckets inside of it. Buckets in Ozone function in a manner comparable to those in Amazon S3.
- **Keys:** are objects that are exclusive to a certain bucket and are analogous to S3 Objects in their function. Any string may be used for a key name. The data that you store inside these keys is denoted by values, and Ozone does not yet impose any kind of maximum size restriction on key sizes.

Ozone is a redundant, distributed object store optimized for Big data workloads. The primary design point of ozone is scalability, and it aims to scale to billions of objects. Ozone separates namespace management and block space management; this helps ozone to scale much better. The namespace is managed by a daemon called Ozone Manager (OM), and block space is managed by Storage Container Manager (SCM). Ozone consists of volumes, buckets, and keys. A volume is similar to a home directory in the ozone world. Only an administrator can create it. Volumes are used to store buckets. Once a volume is created users can create as many buckets as needed. Ozone stores data as keys which live inside these buckets. Ozone namespace is composed of many storage volumes. Storage volumes are also used as the basis for storage accounting. The block diagram shows the core components of Ozone.



The Ozone Manager is the name space manager, Storage Container Manager manages the physical and data layer and Recon is the management interface for Ozone.



Advanced Concepts

Any distributed system can be viewed from different perspectives. One way to look at Ozone is to imagine it as Ozone Manager as a name space service built on top of HDDS, a distributed block store. Another way to visualize Ozone is to look at the functional layers; we have a metadata data

management layer, composed of Ozone Manager and Storage Container Manager. We have a data storage layer, which is basically the data nodes and they are managed by SCM. The replication layer, provided by Ratis is used to replicate metadata (OM and SCM) and also used for consistency when data is modified at the data nodes. We have a management server called Recon, that talks to all other components of Ozone and provides a unified management API and UX for Ozone.

We have a protocol bus that allows Ozone to be extended via other protocols. We currently only have S3 protocol support built via Protocol bus. Protocol Bus provides a generic notion that you can implement new file system or object store protocols that call into O3 Native protocol.



Contents

Chapter-3 Apache Hive	1
Overview	1
Hive Features	1
Hive Architecture	2
Benefits of using Apache Hive.....	3
Problems with Hive	4
Facts about Apache Hive.....	4
FAQ for Apache Hive	4

Chapter-3 Apache Hive

Overview

- Apache Hive is a Hadoop-based data warehouse for searching and analysing massive Hadoop datasets. Hadoop processes structured and semi-structured data.
- Initially, you had to construct sophisticated Map-Reduce tasks to use Hadoop as a Big Data Processing engine, but now you can submit SQL queries. Hive targets SQL-savvy users.
- Hive uses HQL, a SQL-like language. HiveQL converts SQL-like queries to MapReduce jobs.
- Hive simplifies Hadoop. And you don't need to learn java if you know ANSI SQL to work with BigData.
- Your SQL query is transformed into a series of Map Reduce tasks by the Hive client, which typically runs on your gateway/client node and sends them to a Hadoop cluster for execution.
- Hive arranges the Data into tables, which helps for providing an structure to HDFS data.
- Data analysis is performed using Hive, which is a Data Warehousing software that was created on top of Hadoop. Additionally, Hive makes use of a language known as HiveQL (HQL), which automatically converts queries that are similar to SQL into tasks for MapReduce.

Hive Features

- Before Apache Hive, there were several problems with Big Volume of data. Data size are keep growing and making it tough to manage and the typical RDBMS failed to handle this volume.
- Most of the organizations attempted MapReduce to solve this issue. However, writing MapReduce was another programming challenges, everybody has to learn Java and write complex MapReduce jobs.
- Apache Hive helped a lot to overcome above problems.
- BigData Companies are now able to execute the following using Apache Hive:

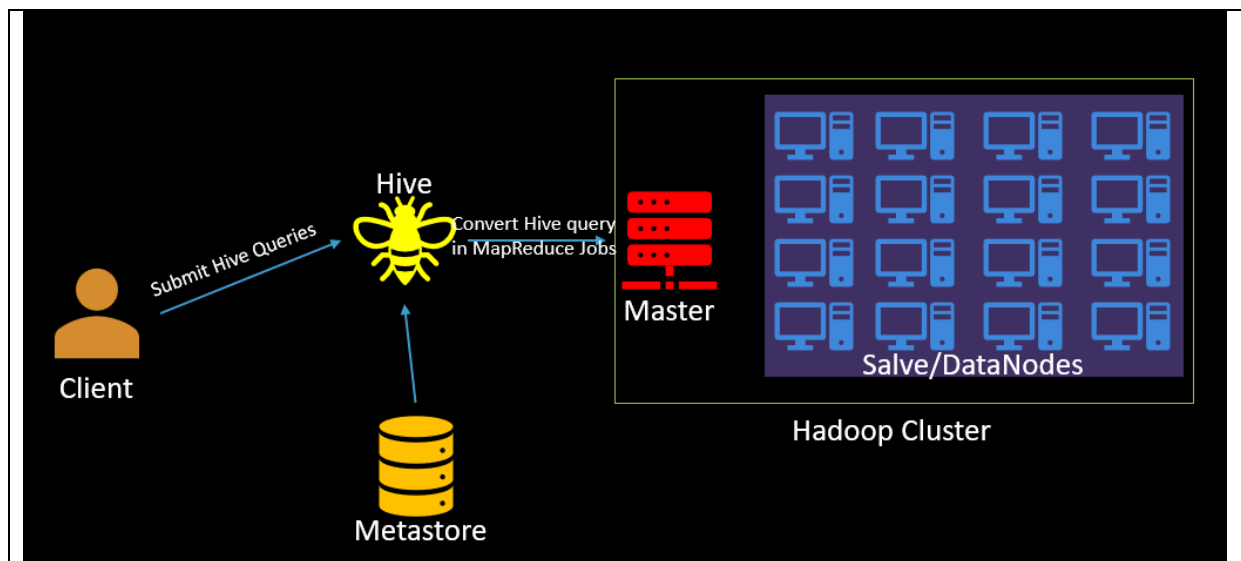
- Schema flexibility and evolution
- Tables can be partitioned and bucketed
- Apache Hive tables are defined directly in the HDFS
- JDBC/ODBC drivers are available
- For ad hoc needs, Apache Hive spares developers from designing difficult Hadoop MapReduce processes. So, hive offers data summary, analysis, and querying.
- Hive is scalable and very quick. It may be extended greatly. Because Apache Hive and SQL are so similar, learning and using Hive Queries is fairly simple for SQL developers.
- By giving users a way to submit SQL queries via an interface, Hive lessens the complexity of MapReduce. Therefore, with Apache Hive, business analysts may now experiment with big data and provide insights.
- Additionally, it offers file access to a number of data storage, including HDFS and HBase. The fact that we don't need to understand Java in order to use Apache Hive is its most significant feature.

Hive Architecture

Apache Hive has following major components:

1. **Metastore:**
 - a. It stores metadata for each of the tables like their schema and location.
 - b. Metastore stores the information about table partitions.
 - c. Driver connects to metastore to get the detail about various data sets distributed over the cluster nodes.
 - d. Metastore is always a traditional RDBMS store like MySQL, Oracle RDBMS.
 - e. Hive metadata helps the driver to keep a track of the data and it is highly crucial.
2. **Driver:**
 - a. This is similar to JDBC driver which receives the HiveQL statements.
 - b. The driver starts the execution of the statement by creating sessions.
 - c. It monitors the life cycle and progress of the execution.
 - d. Driver stores the necessary metadata generated during the execution of a HiveQL statement.
 - e. It also acts as a collection point of data or query result obtained after the Reduce operation.
3. **Compiler:**
 - a. Every Hive SQL statement needs to be compiled before it performs the actual query execution.
 - b. Compiler generates the query to an execution plan.
 - c. The execution plan contains the tasks.
 - d. And contains the steps needed to be performed by the MapReduce to get the output as translated by the query.
 - e. The compiler in Hive converts the query to an Abstract Syntax Tree (AST).
 - f. First, check for compatibility and compile-time errors, then converts the AST to a Directed Acyclic Graph (DAG).
4. **Optimizer:**
 - a. It performs various transformations on the execution plan to provide optimized DAG.

- b. It aggregates the transformations together, such as converting a pipeline of joins to a single join, for better performance.
 - c. The optimizer can also split the tasks, such as applying a transformation on data before a reduce operation, to provide better performance.
5. **Executor:**
 - a. Once compilation and optimization complete, the executor executes the tasks.
 - b. Executor takes care of pipelining the tasks.
6. **CLI, UI, and Thrift Server:**
 - a. CLI (command-line interface) provides a user interface for an external user to interact with Hive.
 - b. Thrift server in Hive allows external clients to interact with Hive over a network, similar to the JDBC or ODBC protocols.



Benefits of using Apache Hive

- Hive makes it considerably simpler to do data analysis, queries, and summarization on large amounts of data.
- Since Hive supports external tables, it is feasible to process data without actually storing it in HDFS. This is made possible by the fact that Hive supports external tables.
- Apache Hive is an excellent choice for fulfilling the low-level interface requirement of Hadoop.
- It also enables the division of data at the table level to boost efficiency, and this feature is supported.
- Hive is equipped with a rule-based optimizer with the purpose of improving logical plan performance.
- It is expandable, scalable, and has a user-friendly interface.
- Using HiveQL does not need any prior knowledge of programming languages; all that is required is a fundamental understanding of SQL queries.
- Using Hive, we can simplify the process of processing structured data in Hadoop.
- Because it is so close to SQL, querying in Hive is a pretty straightforward process.
- Through the use of Hive, we can also conduct ad hoc queries for the data analysis.

Problems with Hive

- There is no provision for real-time queries or changes at the row level in Apache Hive.
- Additionally, Hive delivers a latency that is suitable for interactive data browsing.
- It is not beneficial for the processing of online transactions.
- In most cases, Apache Hive queries are characterised by a fairly long latency.

Facts about Apache Hive

- The Apache Hive is a Structural Framework on Hadoop.
- Using Hive, you can query massive datasets that are stored in remote storage might be helpful.
- Apache Hive is a kind of distributed data warehouse.
- HiveQL is a query language that is similar to SQL (HQL).
- HiveQL is a declarative programming language similar to SQL.
- The Hive table structure is analogous to the tables in a relational database.
- Using the Hive-QL query language, several users may concurrently query the data.
- Hive, on the other hand, enables the building of bespoke MapReduce framework processes that may be used to do more in-depth data analysis.
- It is simple to extract, convert, and load (ETL) data using Apache Hive from HDFS.
- Hive provides the framework for a broad range of data types.
- Hive makes it possible to access files that are stored in HDFS.
- In addition to that, Apache Hive enables the conversion of a wide number of formats.
- However, Hive is not intended for use in the processing of online transactions (OLTP). Despite this, we are able to put it to use for online analytical processing (OLAP).
- Apache Hive does not support updates or deletes, but it does support overwriting and acquiring data.
- Hive does not support the use of subqueries, while writing this study material.

FAQ for Apache Hive

Question-1. What is Hive Metastore?

Answer: Hive metastore is a database that stores metadata about your Hive tables (eg. Table name, column names and types, table location, storage handler being used, number of buckets in the table, sorting columns if any, partition columns if any, etc.). When you create a table, this metastore gets updated with the information related to the new table which gets queried when you issue queries on that table.

Question-2: Wherever (Different Directory) I run hive query, it creates new metastore_db, please explain the reason for it?

Answer: Whenever you run the hive in embedded mode, it creates the local metastore. And before creating the metastore it looks whether metastore already exist or not. This property is defined in configuration file hive-site.xml. Property is "javax.jdo.option.ConnectionURL" with default value "jdbc:derby;;databaseName=metastore_db;create=true". So to change the behavior change the location to absolute path, so metastore will be used from that location.

Question-3: Is it possible to use same metastore by multiple users, in case of embedded hive?

Answer: No, it is not possible to use metastore in sharing mode. It is recommended to use standalone "real" database like MySQL or PostgreSQL.

Question-4: Is multiline comment supported in Hive Script ?

Answer: No.

Question-5: If you run hive as a server, what are the available mechanism for connecting it from application?

Answer: There are following ways by which you can connect with the Hive Server:

1. Thrift Client: Using thrift you can call hive commands from a various programming languages e.g. C++, Java, PHP, Python and Ruby.
2. JDBC Driver : It supports the Type 4 (pure Java) JDBC Driver
3. ODBC Driver: It supports ODBC protocol.

Question-6: What is SerDe in Apache Hive?

Answer: A SerDe is a short name for a Serializer Deserializer. Hive uses SerDe (and FileFormat) to read and write data from tables. An important concept behind Hive is that it DOES NOT own the Hadoop File System (HDFS) format that data is stored in. Users are able to write files to HDFS with whatever tools/mechanism takes their fancy("CREATE EXTERNAL TABLE" or "LOAD DATA INPATH,") and use Hive to correctly "parse" that file format in a way that can be used by Hive. A SerDe is a powerful (and customizable) mechanism that Hive uses to "parse" data stored in HDFS to be used by Hive.

Question-7: Which classes are used by the Hive to Read and Write HDFS Files

Answer: Following classes are used by Hive to read and write HDFS files

- TextInputFormat/HiveIgnoreKeyTextOutputFormat: These 2 classes read/write data in plain text file format.
- SequenceFileInputFormat/SequenceFileOutputFormat: These 2 classes read/write data in hadoop SequenceFile format.

Question-8. Give examples of the SerDe classes which hive uses to Serialize and Desterilize data ?

Answer: Hive currently use these SerDe classes to serialize and deserialize data:

- MetadataTypedColumnsetSerDe: This SerDe is used to read/write delimited records like CSV, tab-separated control-A separated records (quote is not supported yet.)
- ThriftSerDe: This SerDe is used to read/write thrift serialized objects. The class file for the Thrift object must be loaded first.
- DynamicSerDe: This SerDe also read/write thrift serialized objects, but it understands thrift DDL so the schema of the object can be provided at runtime. Also it supports a lot of different protocols, including TBinaryProtocol, TJSONProtocol, TCTLSeparatedProtocol (which writes data in delimited records).

Question-9. Can you use Apache Hive for OLTP systems?

Answer: No, it is not suitable for OLTP system because it does not offer insert and update at the row level.

Question-10. Why does Apache Hive not store metadata information in HDFS and instead it needs RDBMS?

Answer: Hive stores metadata information in the metastore which must be an RDBMS, so that it can achieve low latency. Since, HDFS read/write operations are time-consuming processes.



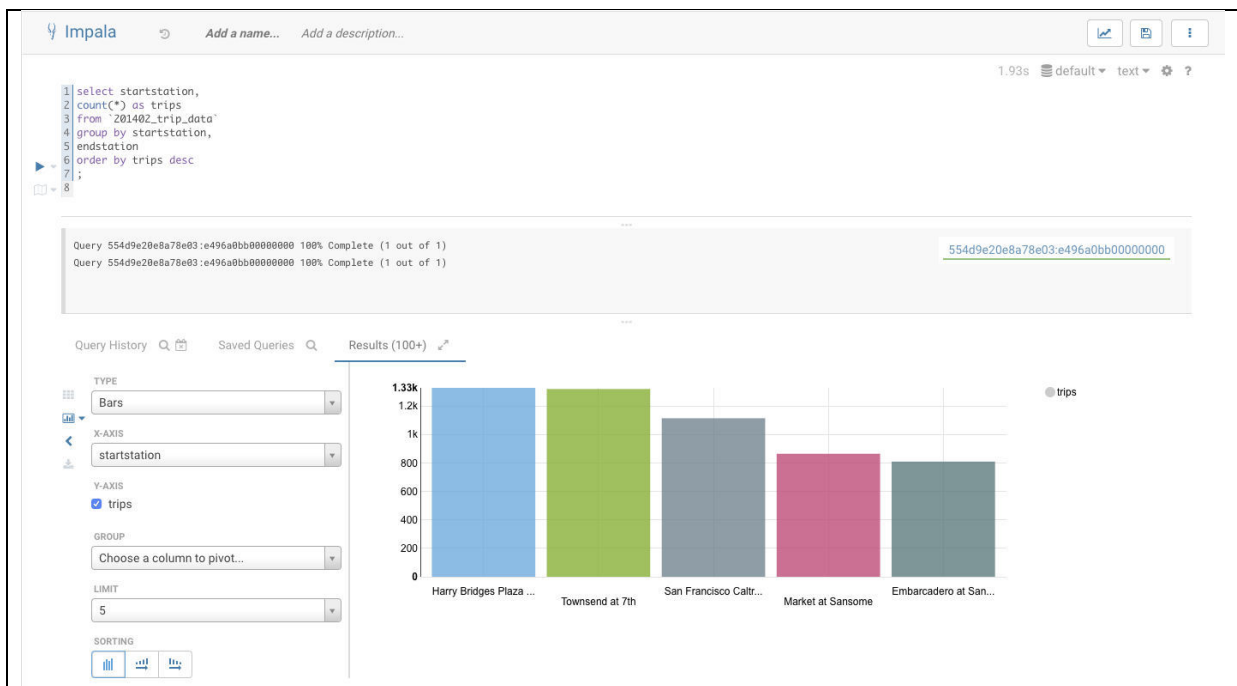
Contents

- Chapter-4: Apache Hue..... 1
 - Overview 1
 - Hue Design and Architecture 3
 - Administrator 3
 - Major Functionalities of Hue in CDP 4
 - Hive vs Hue..... 6

Chapter-4: Apache Hue

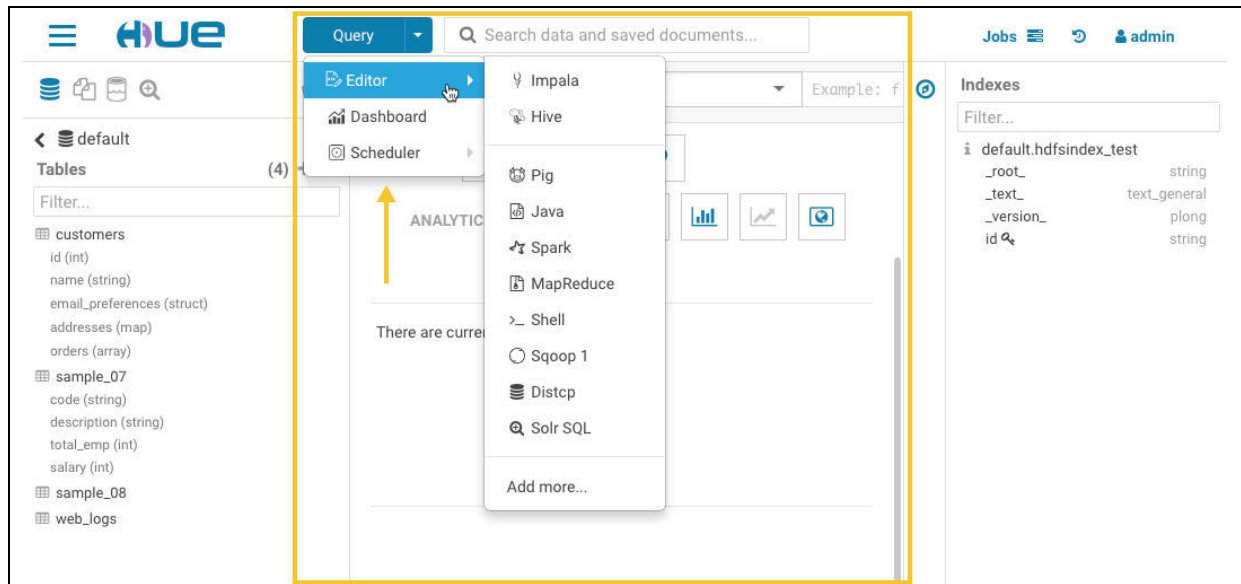
Overview

Hue is an interactive query editor that runs in a web browser and gives users the ability to interact with data warehouses. This is one of the most widely used Query Editor and HadoopExam will also use this heavily in their training programs specially on HDFS, Sqoop, Hive and Impala. Through the use of Hue, you can provide your SQL developers access to the power of business intelligence (BI) and analytics. Hue is an analytics workbench that is open source and was developed for easy and quick data discovery, as well as intelligent query support and seamless collaboration. Create a bridge between information technology and the company in order to provide reliable self-service analytics. For illustration purposes, the following image demonstrates a graphical representation of the results of an Impala SQL query that may be generated with Hue:



Using Hue, you can do following activities

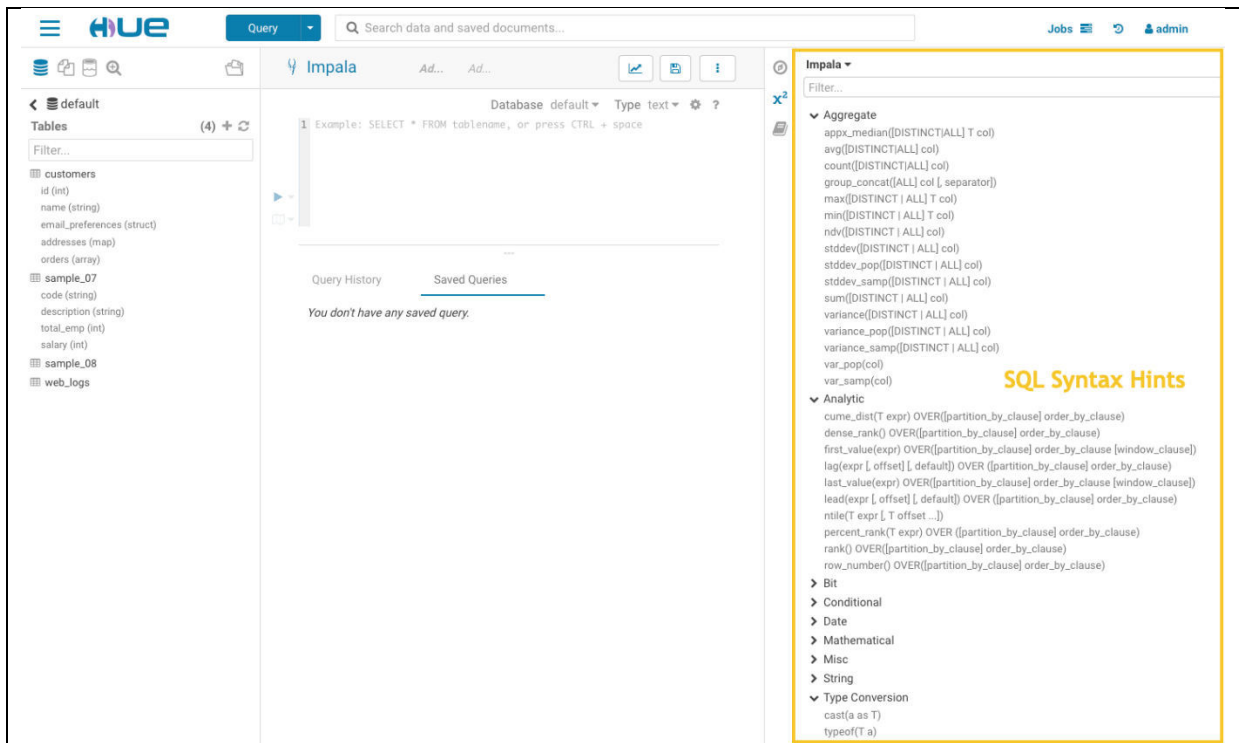
1. **Hive & Impala:** Explore your various databases.
2. **Schema Tables:** Proceed to the specific tables afterward.
3. **Files:** Navigate through the HDFS directories and the cloud storage.
4. **Table Detail:** Find the indexes and tables stored in HBase or Kudu.
5. **Documents:** Find documents



The primary section of the Hue UI has a comprehensive collection of tools, which include the following:

- **Editing environments:** that are flexible and enable the creation of a broad range of scripts.
- **Dashboards:** that may be constructed "on the fly" by dragging and dropping objects into the centre panel of the Hue user interface. There is no need for programming at all. After that, you may study your statistics by using your individualised dashboard.
- **Schedulers:** The tool that allows you to construct schedulers by dragging and dropping, similar to how dashboards are created. Using this feature, you will be able to construct individualised processes and set them to execute on a predetermined schedule at certain intervals. A monitoring interface will provide the current status of the tasks, as well as logs, and will allow you to pause or cancel them.

Assistant Panel: The assistant panel you can find to the right of the main panel, and it offers tips and guidance pertinent to the application that is now being used in the main panel. For instance, in the picture that can be seen above, there is a middle panel that contains Impala SQL tips that may assist in the construction of queries.



Hue Design and Architecture

In general, each Hue server is capable of supporting roughly 25 concurrent users, however this number might vary based on the tasks being carried out by the users. It is not the number of users that causes the majority of scaling problems; rather, it is the actions that users execute that are resource expensive. For instance, downloading a significant amount of query results may have an effect on the availability of resources for other users who are using the same instance of Hue at the same time as the download process. During such period, the users could notice that performance is slowing down. Another typical factor that might bring about observable shifts in performance are sluggish RPC calls made between Hue and another service. When this occurs, the queries that are sent may give the impression that they are "hanging" all of a sudden.

As a general rule, two Hue servers may support up to the following:

- 100 unique users per week
- At peak periods, there are 50 users per hour who can execute up to 100 queries.
- In a typical configuration, there are two Hue servers.

Administrator

- Install a load balancer on the path leading up to Hue.
- Make use of a database of production-quality. Hue Custom Databases may be seen here for further details.
- Make sure that other services, such Impala, Hive, and Oozie, are in good health and are not being negatively affected by the lack of resources. If any of these services become unresponsive, it will have a negative impact on Hue's performance.
- It is a good idea to think about shifting workloads that are governed by service-level agreements (SLAs) or are regarded as "noisy neighbours" to their own computing cluster.

Workloads that use the bulk of the available resources and result in performance concerns are referred to as "noisy neighbours." Look into Virtual Private Clusters and Cloudera SDX if you want to learn more about how to keep your computation and storage functions separate.

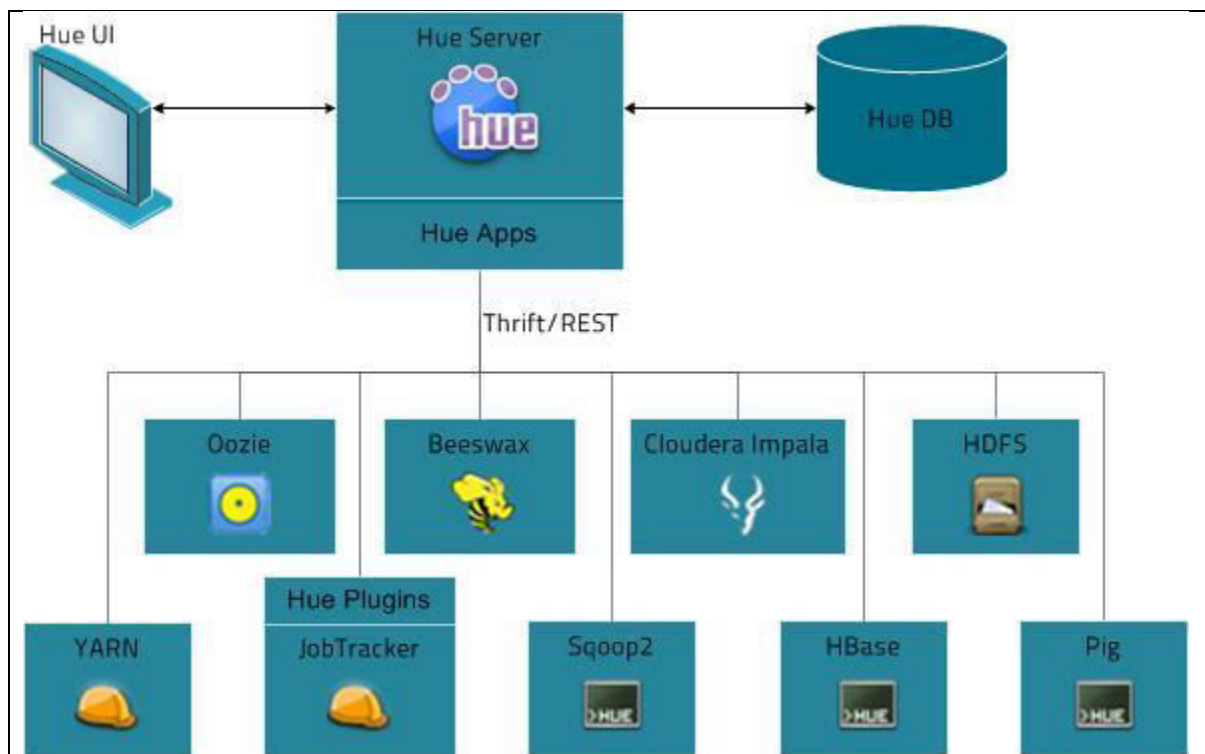
- Set a maximum for the number of rows that will be returned in response to queries.
- Setting a value for the download row limit configuration variable of the Hue Beeswax application is one approach to restrict the number of rows that are retrieved by the application. This property may be set in the Hue Service Advanced Configuration Snippet (Safety Valve), which is located in the Cloudera Manager configuration file for the hue safety valve.ini property.

Major Functionalities of Hue in CDP

Hue is a web-based interface for using Apache Hadoop to do data analysis. It is possible to install it on any computer running any version of Hadoop.

Hue is a collection of apps that allows users to have web-based access to CDH components and functions as a platform for the development of bespoke applications.

The diagram that follows provides an illustration of how Hue works. The Hue Server web application functions as a "container" that bridges the gap between your CDH installation and the browser. It interfaces with a variety of servers that interface with CDH components and hosts the Hue apps.



Hue, which stands for Hadoop User Experience, is a graphical user interface that is open-source, web-based, and designed for use with Apache Hadoop and Cloudera CDP. A flexible user interface is created by Hue, which brings together a number of separate Hadoop ecosystem projects. Cloudera CDP now includes additional customization options that are exclusive to the Hue. Hue functions as a

front-end for apps that operate on your cluster. This makes it possible for you to engage with applications using an interface that is maybe more intuitive or well-known to you. It is no longer necessary to log in to the cluster in order to execute scripts interactively using each application's corresponding shell since Hue's apps, such as the Hive and Pig editors, eliminate this need. It's possible that after a cluster is up and running, you'll communicate only with its apps using Hue or a comparable interface.

Hue supported features:

- Amazon S3 and Hadoop File System (HDFS) Browser
- With the appropriate permissions, you can browse and move data between the ephemeral HDFS storage and S3 buckets belonging to your account.
- Hive—Run interactive queries on your data. This is also a useful way to prototype programmatic or batched querying.
- Pig—Run scripts on your data or issue interactive commands.
- Oozie—Create and monitor Oozie workflows.
- Metastore Manager—View and manipulate the contents of the Hive metastore (import/create, drop, and so on).
- Job browser—See the status of your submitted Hadoop jobs.
- User management—Manage Hue user accounts and integrate LDAP users with Hue.
- To use the Hue Notebook for Spark, you must install Hue with Livy and Spark.
- **Quickly identify relevant data:** For analytics that are both quicker and more reliable, it is possible to search and find relevant tables, views, and columns across all databases, including cloud-native object stores, with ease. Ensure that data can be trusted instantly by incorporating data stewardship into the system and allowing end users to tag data for further classification and organising that is project-focused.
- **Keep your changes safe and iterate across teams:** Through seamless and safe collaboration and sharing, silos of analytics and business intelligence may be eliminated. You can safeguard even the most sensitive data by saving searches and result sets for later use, sharing them with other users or departments, and explicitly setting access rights for those saved items.
- **Intelligent inquiry design and help:** The only SQL editor with usage-enriched intelligence to safeguard against malicious queries and make SQL users more productive. You may explore and do analytics in an efficient and iterative manner by dragging and dropping tables and columns, rapidly designing queries using autocomplete pop-ups, and receiving query suggestions based on use and best practises.
- **Native integration with the cloud:** Browse through all databases, run queries on them, and store the results in both on-premises and cloud-based systems. Apache Hive is used for data preparation, Apache Solr is used for free-text analytics, and Apache Impala is used for high-performance SQL analytics. Hue interfaces with the full of Cloudera's platform, including storage engines, Apache Kudu and Amazon S3 object storage.
- **Include data scientists and analysts in the analytics process:** Cloudera's platform also enables self-service data science with the Cloudera Data Science Workbench, which can be carried out over the same shared data by users who are more familiar with the programming languages R, Python, or Scala. In addition, the platform is compatible with all of the industry's leading business intelligence (BI) and visualisation tools, such as Tableau, Qlik, Zoomdata, and many others. This allows businesses to keep utilising the tools they already rely on while taking advantage of the scalability and flexibility offered by Cloudera.

Hive vs Hue

Hive is a collection of keys and subkeys that are accompanied by a series of supporting files that store backups of the data. To put it simply, the hive is the area where information about the Windows registry is stored. Each hive has a tree, and each tree has a unique key. This key acts as the tree's root, which is the point at which the tree begins or the highest point in the hierarchy inside the register. There are registry keys, registry subkeys, and registry values included inside the registry. The prefix "HKEY" is included at the beginning of every key that belongs to a hive. When all of the other entries in the registry are minimised, a group of keys known as hives will show up on the left-hand side of the screen in the form of folders. One is not possible to establish a Hive, remove one, or rename it. During the early phases of development, Facebook was responsible for launching the hive; however, the Apache Software Foundation eventually took over management of the project.

Both a web user interface that offers a variety of services and a Hadoop framework, Hue is referred to simply as Hue. Hue has a web user interface for browsing HDFS files, in addition to providing the file path. The Job browser, the Hadoop shell, User administrative rights, the Impala editor, the HDFS file browser, the Pig editor, the Hive editor, the Ozzie web interface, and Hadoop API Access are the most essential aspects of Hue. This online user interface style makes it easier for users to browse among the files, in a manner similar to how a typical Windows user would navigate to his or her files on their local PC. Users are able to avoid making syntax mistakes when conducting queries with the help of Hue, which is a convenient tool since it gives a web user interface to programming languages. Hue requires the use of a web browser in order to be installed or configured.



Contents

Chapter-5: Cloudera CDP and YARN	1
Overview	1
CDP Compute	1
YARN architecture and workflow.....	1
YARN Features.....	2
YARN and Cluster Basics (Master and Worker Nodes)	3
YARN Configuration File.....	4
YARN Requires a Global View	4
YARN Containers	5
YARN Application Processing on Cluster.....	5
MapReduce Fundamental Concepts.....	8
YARN Integrated across the CDP platform.....	9
YARN Scheduler.....	10
YARN Capacity Scheduler Overview.....	11
YARN Web User Interface	11
Resource Scheduling and Management	13

Chapter-5: Cloudera CDP and YARN

Overview

- Apache YARN is the processing layer or execution engine for managing distributed applications that run on multiple machines in a network.
- YARN is also known as MRv2.
- YARN supports MapReduce and legacy (MRv1) MapReduce jobs can run in YARN.
- YARN architecture splits the two primary responsibilities of the JobTracker into
 - Resource management and
 - Job scheduling/monitoring
- YARN will have two separate daemons for each of the above responsibility.
 - A global ResourceManager for resources management and
 - Per-application ApplicationMasters (Assume, each one of your Impala Query or Hive Query or a MapReduce Job is a separate application and for each YARN will create an Application Master).
- YARN enables the use of a variety of data processing engines for batch, interactive, and real-time stream processing of data stored in HDFS or cloud storage such as S3 and ADLS.

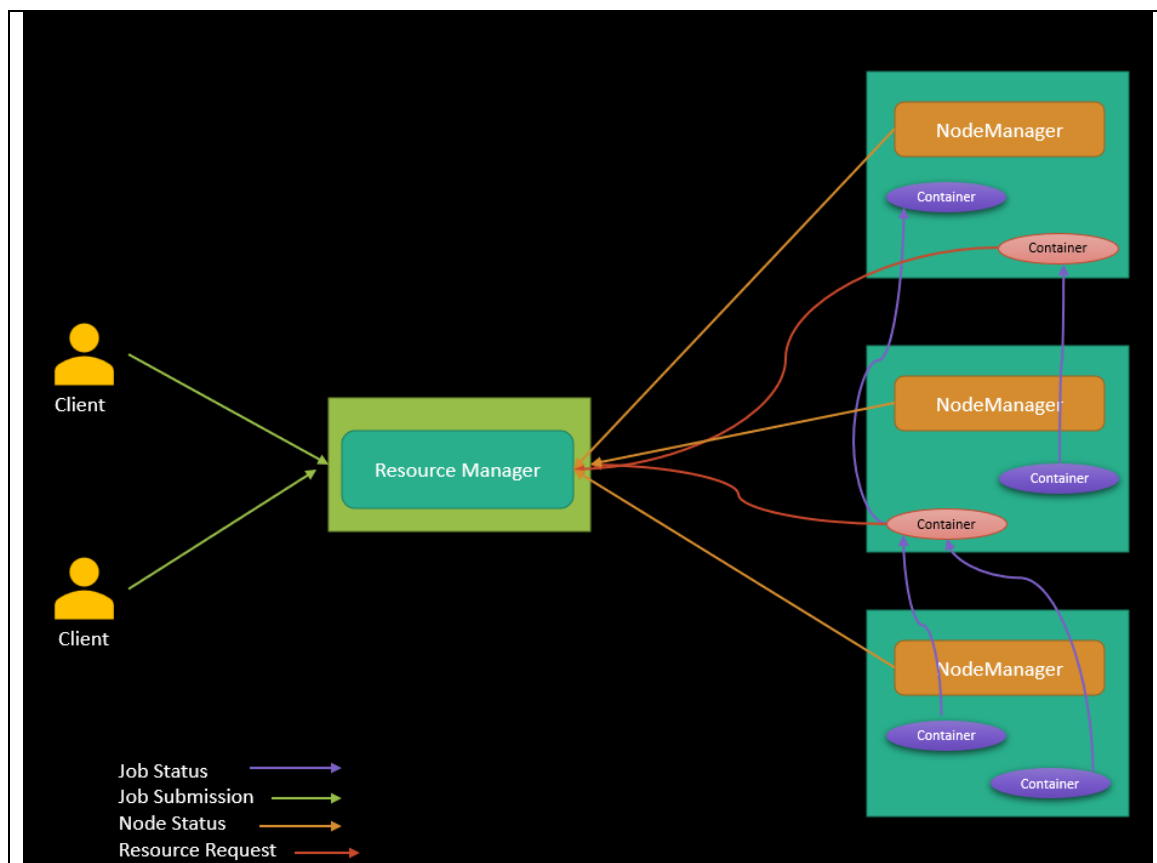
- You may run various processing frameworks for distinct use-cases on the same Hadoop cluster with the help of YARN, for example, Hive for SQL applications, Spark for in-memory applications, and Storm for streaming applications.
- YARN is part of Cloudera Runtime.

CDP Compute

- Apache YARN is resource management tool for CDP or Hadoop framework.
- Apache YARN manages resources for the applications running on your cluster by allocating resources through scheduling, limiting CPU usage, and partitioning clusters.
- You can use Access Control Lists to use YARN with a secure cluster.
- By using Apache YARN, you can optimize the use of vcores and memory.

YARN architecture and workflow

- YARN has three main components:
 - **ResourceManager:** Allocates cluster resources using a Scheduler and ApplicationManager.
 - **ApplicationMaster:** Manages the life-cycle of a job by directing the NodeManager to create or destroy a container for a job. There is only one ApplicationMaster for a job.
 - **NodeManager:** Manages jobs or workflow in a specific node by creating and destroying containers in a cluster node.



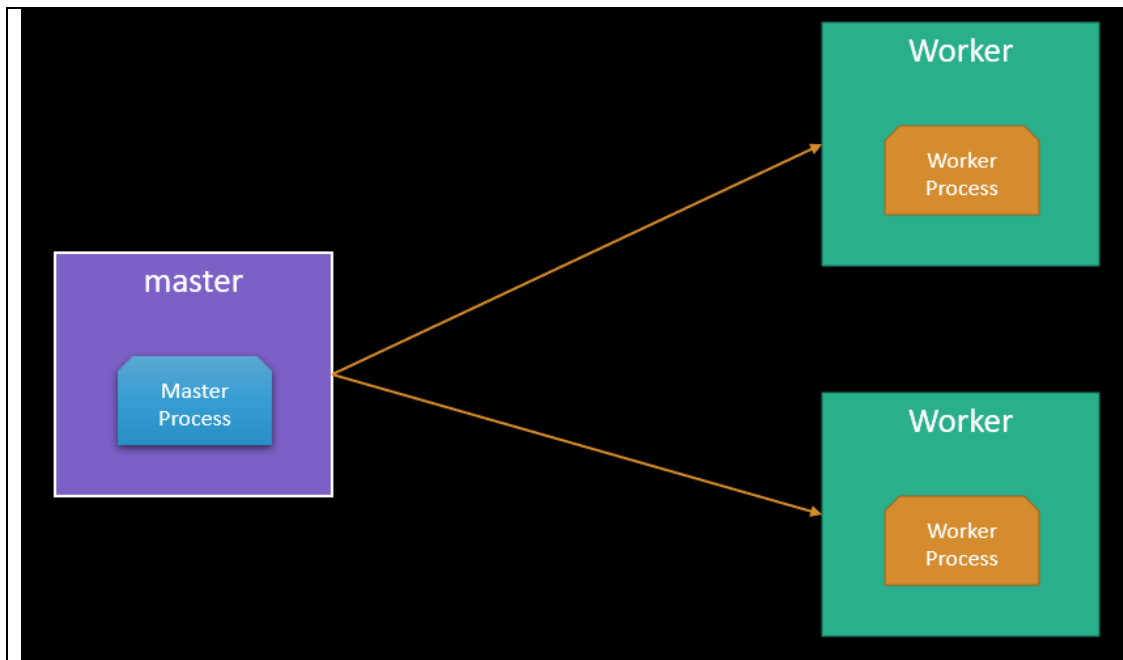
YARN Features

- YARN enables you to manage resources and schedule jobs in Hadoop and has the following features.

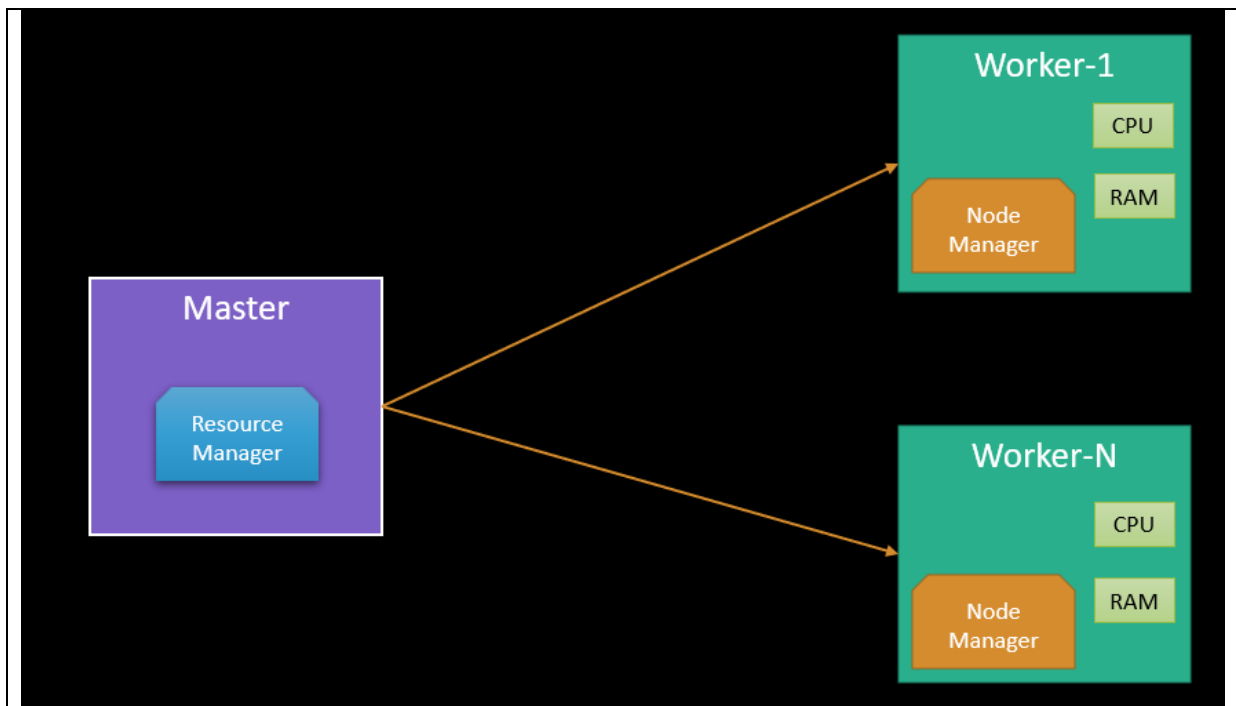
- **Multi-tenancy:**
 - You can use multiple open-source and proprietary data access engines for batch, interactive, and real-time access to the same dataset. Multi-tenant data processing improves an enterprise's return on its Hadoop investments.
 - YARN's dynamic resource management allows many engines and workloads to use the same cluster resources. Make your data available to users throughout your whole business environment via batch, interactive, sophisticated, or real-time processing, all inside the same platform, to get the most out of your Hadoop platform.
- **Cluster utilization:**
 - You can dynamically allocate cluster resources to improve resource utilization.
 - Fine-grained settings improve cluster utilisation, allowing you to implement workload SLAs for priority workloads and group-based rules throughout the enterprise. Process more data in more ways while keeping your most vital tasks running smoothly.
- **Multiple resource types:** You can use multiple resource types such as memory, CPU, and GPU.
- **Scalability:**
 - Significantly improved data center processing power. YARN's ResourceManager focuses exclusively on scheduling and keeps pace as clusters expand to thousands of nodes managing petabytes of data.
 - YARN is intended to manage scheduling for Hadoop's vast scale, allowing you to add new and bigger workloads while staying on the same platform.
- **Compatibility:** MapReduce applications developed for Hadoop 1 runs on YARN without any disruption to existing processes. YARN maintains API compatibility with the previous stable release of Hadoop.

YARN and Cluster Basics (Master and Worker Nodes)

- A host, which is also called a node, in YARN terminology. A cluster is two or more hosts connected by a high-speed local network.
- In Hadoop, there are two types of hosts in the cluster.



- A master host serves as the point of communication for a client programme. The other computers in the cluster, known as worker hosts, get their assignments from the cluster's master host.
- In a YARN cluster, there are two types of hosts:
 - **The ResourceManager** is the master daemon that communicates with the client, tracks resources on the cluster, and orchestrates work by assigning tasks to NodeManagers.
 - **A NodeManager** is a worker daemon that launches and tracks processes spawned on worker hosts.

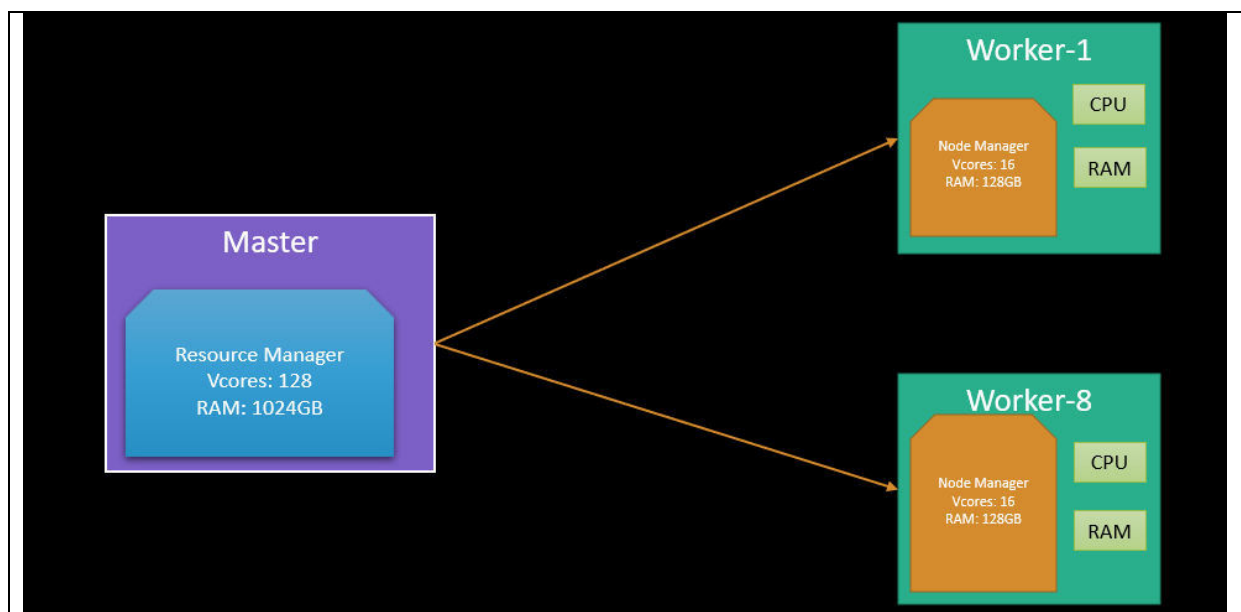


YARN Configuration File

- The YARN configuration file is an XML file that contains properties.
- This file is placed in a well-known location on each host in the cluster and is used to configure the ResourceManager and NodeManager.
- By default, this file is named yarn-site.xml.

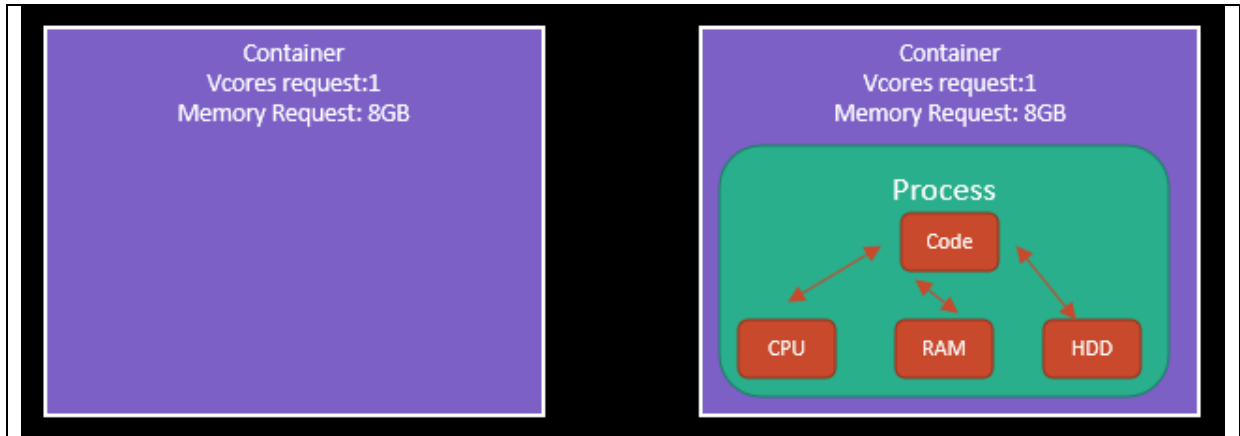
YARN Requires a Global View

- YARN currently defines two resources
 - o vcores
 - o and memory.
- Each NodeManager tracks its own local resources and communicates its resource configuration to the ResourceManager, which keeps a running total of the cluster's available resources.
- By keeping track of the total, the ResourceManager knows how to allocate resources as they are requested.
- Vcore has a special meaning in YARN. You can think of it simply as a "usage share of a CPU core."
- If you expect your tasks to be less CPU-intensive (sometimes called I/O-intensive), you can set the ratio of vcores to physical cores higher than 1 to maximize your use of hardware resources.)



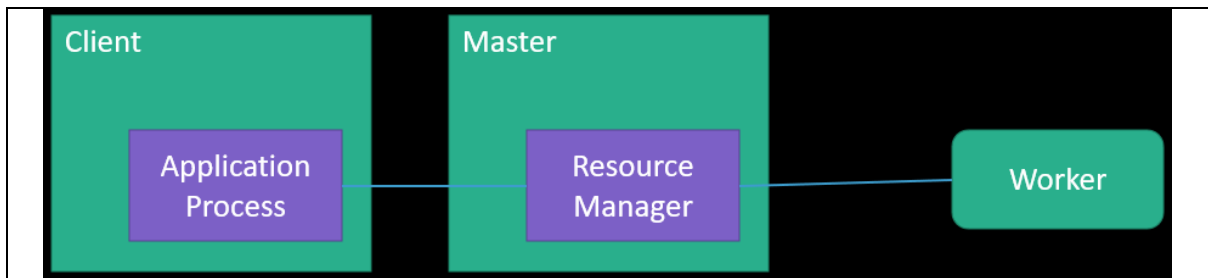
YARN Containers

- Containers are an important YARN concept.
- You can think of a container as a request to hold resources on the YARN cluster.
- Currently, a container hold request consists of vcore and memory, as shown in below
- Container as a hold (left), and container as a running process (right).
- Once a hold has been granted on a host, the NodeManager launches a process called a task.
- The right side of Figure shows the task running as a process inside a container.

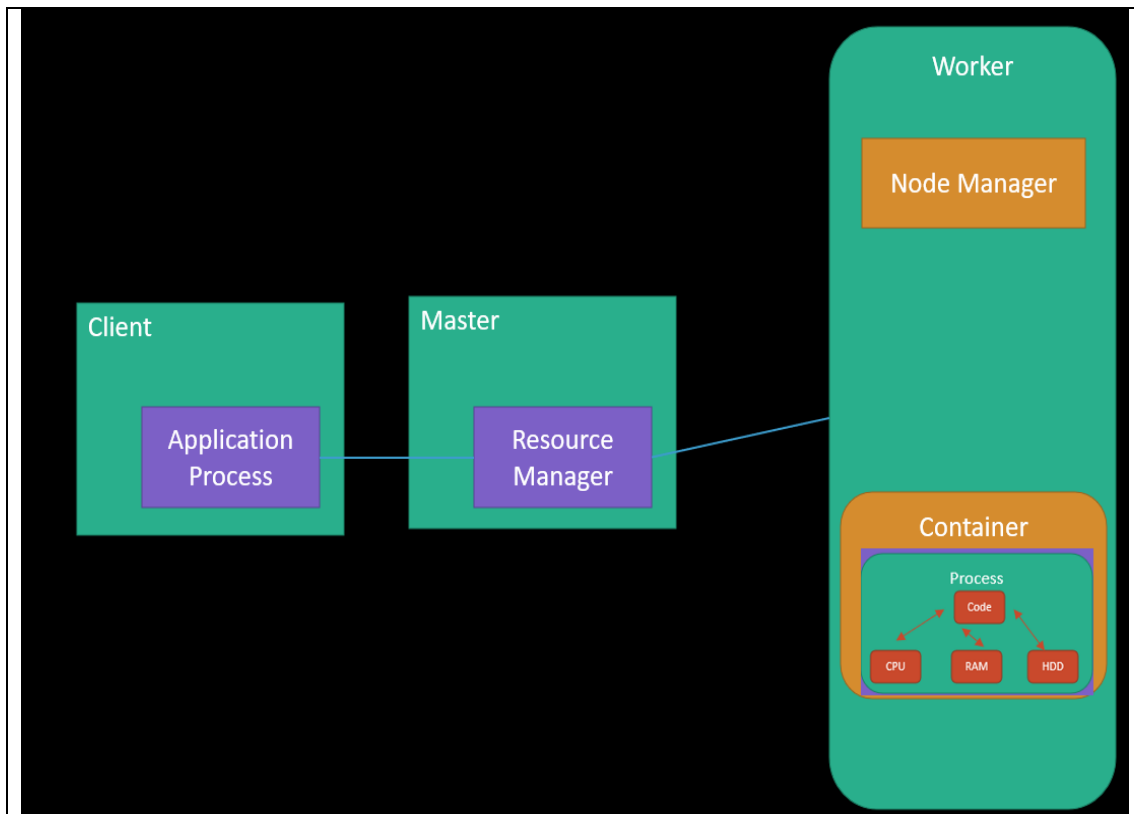


YARN Application Processing on Cluster

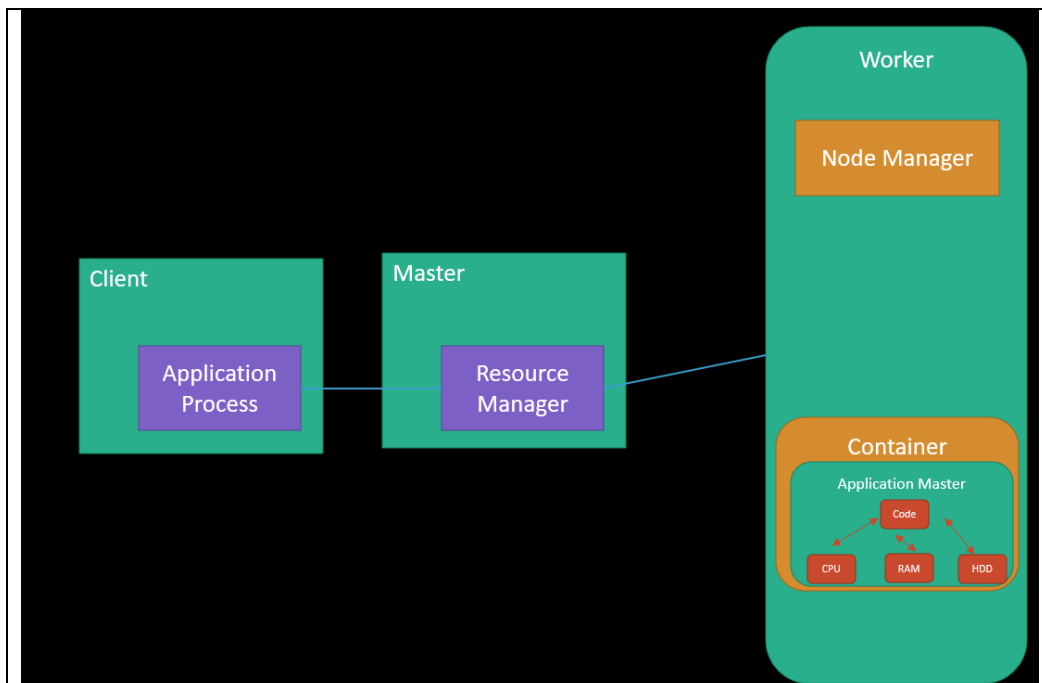
- An application is a YARN client program that is made up of one or more tasks.
- For each running application, a special piece of code called an ApplicationMaster helps coordinate tasks on the YARN cluster.
- The ApplicationMaster is the first process run after the application starts.
- An application running tasks on a YARN cluster consists of the following steps:
- **Step-1:** The application starts and talks to the ResourceManager for the cluster, Application starting up before tasks are assigned to the cluster



Step-2: The ResourceManager makes a single container request on behalf of the application and allocated container on a cluster.

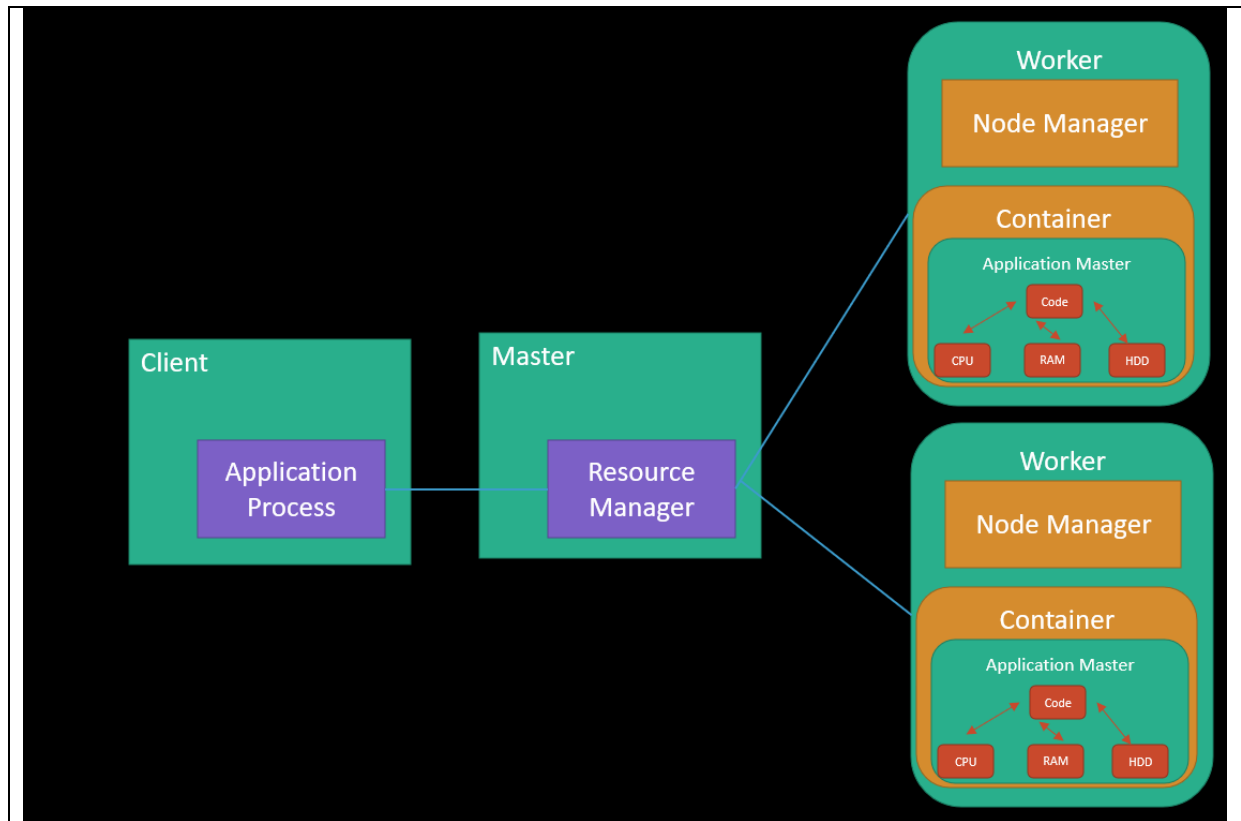


Step-3: The ApplicationMaster starts running within that container, Application + ApplicationMaster running in the container on the cluster.



Step-4: The ApplicationMaster requests subsequent containers from the ResourceManager that are allocated to run tasks for the application. Those tasks do most of the status communication with the

ApplicationMaster allocated in Step 3), as shown below Application + ApplicationMaster + task running in multiple containers running on the cluster.

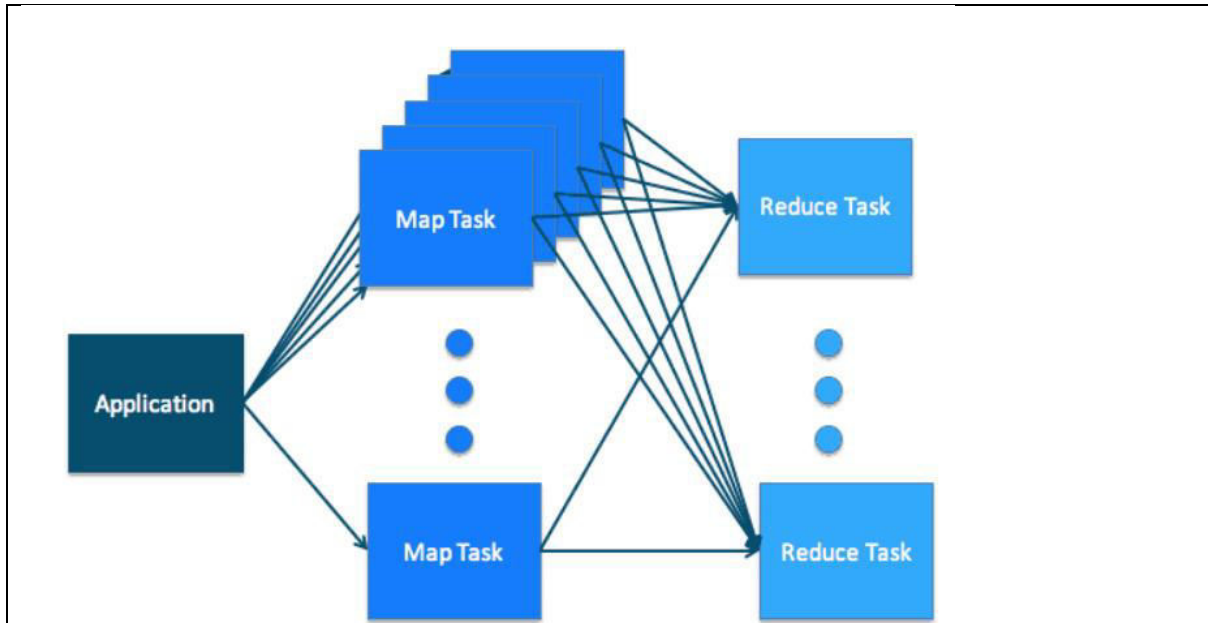


Step-5: Once all tasks are finished, the ApplicationMaster exits. The last container is de-allocated from the cluster.

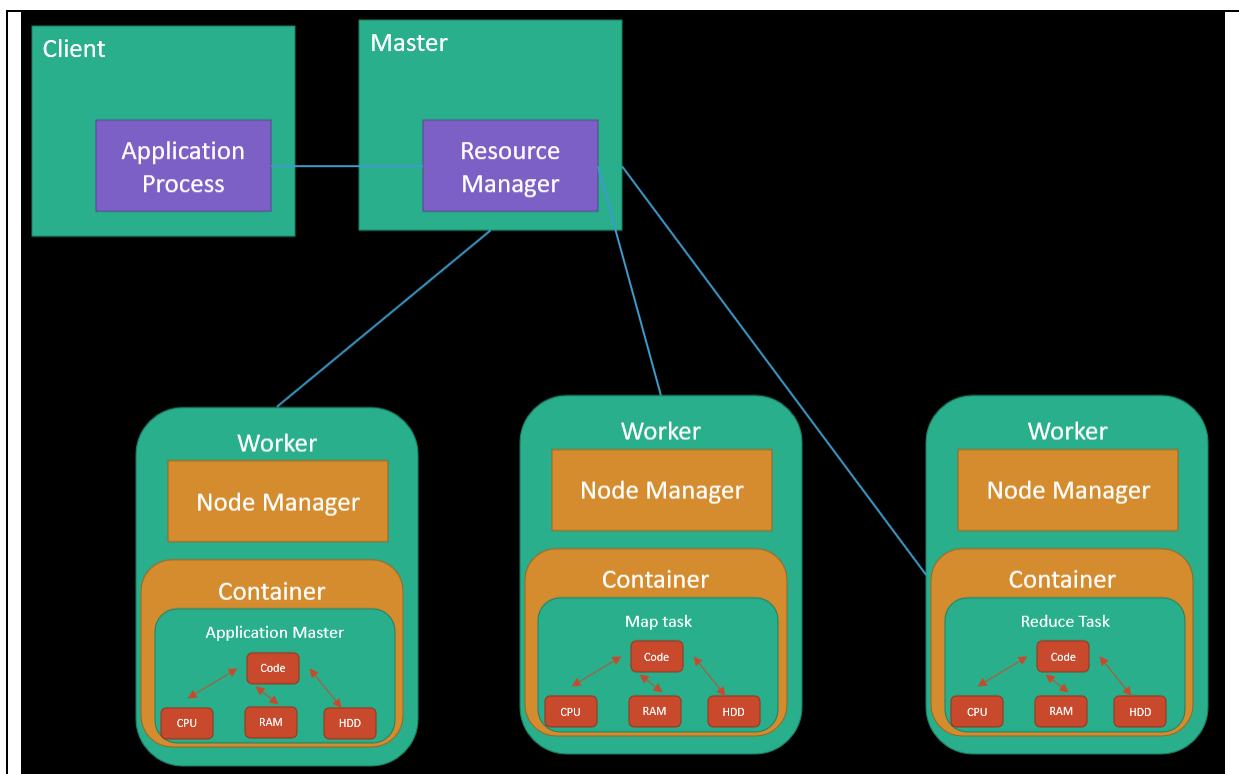
Step-6: The application client exits. (The ApplicationMaster launched in a container is more specifically called a managed AM. Unmanaged ApplicationMasters run outside of YARN's control. Llama is an example of an unmanaged AM.)

MapReduce Fundamental Concepts

- In the MapReduce paradigm, an application consists of Map tasks and Reduce tasks. Map tasks and Reduce tasks align very cleanly with YARN tasks.



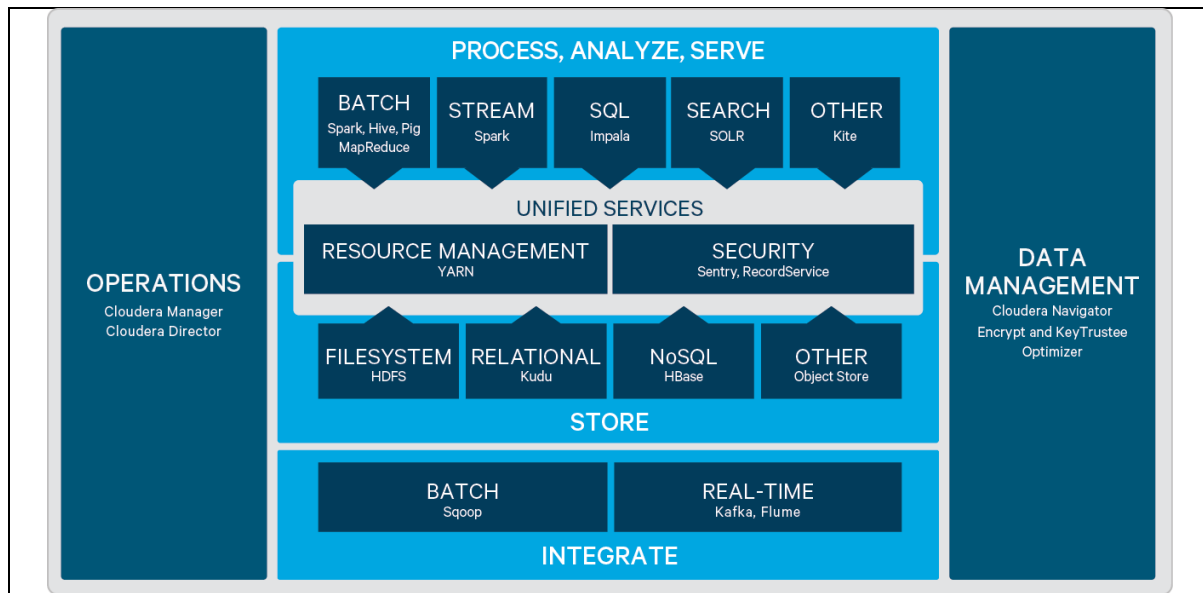
- Below image, illustrates how the map tasks and the reduce tasks map cleanly to the YARN concept of tasks running in a cluster.



- In a MapReduce application, there are multiple map tasks, each running in a container on a worker host somewhere in the cluster. Similarly, there are multiple reduce tasks, also each running in a container on a worker host.
- Simultaneously on the YARN side, the ResourceManager, NodeManager, and ApplicationMaster work together to manage the cluster's resources and ensure that the tasks, as well as the corresponding application, finish cleanly.

YARN Integrated across the CDP platform

- A CDP cluster is made up of two or more hosts connected by an internal high-speed network.
- Master hosts are a small number of hosts reserved to control the rest of the cluster. Worker hosts are the non-master hosts in the cluster.
- In a cluster with YARN running, the master process is called the ResourceManager and the worker processes are called NodeManagers.
- The configuration file for YARN is named yarn-site.xml. There is a copy on each host in the cluster. It is required by the ResourceManager and NodeManager to run properly. YARN keeps track of two resources on the cluster, vcores and memory. The NodeManager on each host keeps track of the local host's resources, and the ResourceManager keeps track of the cluster's total.
- A container in YARN holds resources on the cluster. YARN determines where there is room on a host in the cluster for the size of the hold for the container. Once the container is allocated, those resources are usable by the container.
- An application in YARN comprises three parts:
 - The application client, which is how a program is run on the cluster.
 - An ApplicationMaster which provides YARN with the ability to perform allocation on behalf of the application.
 - One or more tasks that do the actual work (runs in a process) in the container allocated by YARN.
- A MapReduce application consists of map tasks and reduces tasks.
- A MapReduce application running in a YARN cluster looks very much like the MapReduce application paradigm, but with the addition of an ApplicationMaster as a YARN requirement.
- Cloudera's platform is built on core Hadoop, which includes HDFS, MapReduce, and YARN.
- All platform components have access to the same HDFS data and take part in shared resource management through YARN.
- Hadoop, as part of Cloudera's platform, also benefits from straightforward deployment and administration (through Cloudera Manager) as well as shared compliance-ready security and governance.



YARN Scheduler

- A scheduler determines which jobs run, where and when they run, and the resources allocated to the jobs.
- The YARN (MRv2) and MapReduce (MRv1) computation frameworks support the following schedulers:
 - o **FIFO** - Allocates resources based on arrival time.
 - o **Fair** - Allocates resources to weighted pools, with fair sharing within each pool. When configuring the scheduling policy of a pool, Domain Resource Fairness (DRF) is a type of fair scheduler.
 - o **Capacity** - Allocates resources to pools, with FIFO scheduling within each pool.
- However, Cloudera CDP only supports the Capacity Scheduler.

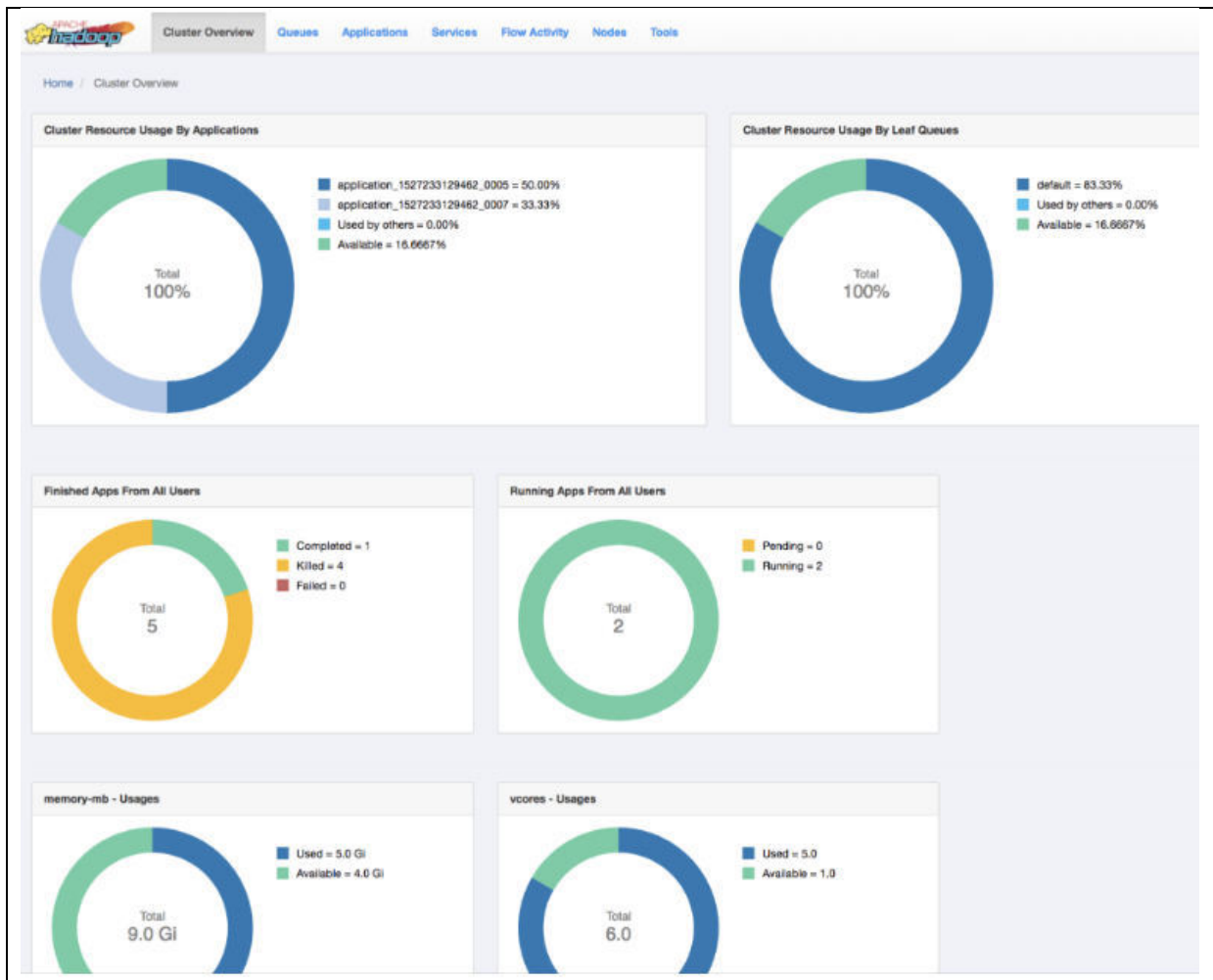
YARN Capacity Scheduler Overview

- The CapacityScheduler is designed to run Hadoop applications as a shared, multi-tenant cluster in an operator-friendly manner while maximizing the throughput and the utilization of the cluster.
- Traditionally each organization has its own private set of compute resources that have sufficient capacity to meet the organization's SLA under peak or near-peak conditions.
- This generally leads to poor average utilization and overhead of managing multiple independent clusters, one per each organization.
- Sharing clusters between organizations is a cost-effective manner of running large Hadoop installations since this allows them to reap benefits of economies of scale without creating private clusters. However, organizations are concerned about sharing a cluster because they are worried about others using the resources that are critical for their SLAs.
- The CapacityScheduler is designed to allow sharing a large cluster while giving each organization capacity guarantees.
- The central idea is that the available resources in the Hadoop cluster are shared among multiple organizations who collectively fund the cluster based on their computing needs.

- There is an added benefit that an organization can access any excess capacity not being used by others. This provides elasticity for the organizations in a cost-effective manner.
- Sharing clusters across organizations necessitates strong support for multi-tenancy since each organization must be guaranteed capacity and safe-guards to ensure the shared cluster is impervious to single rogue application or user or sets thereof.
- The CapacityScheduler provides a stringent set of limits to ensure that a single application or user or queue cannot consume disproportionate number of resources in the cluster. Also, the CapacityScheduler provides limits on initialized and pending applications from a single user and queue to ensure fairness and stability of the cluster.
- The primary abstraction provided by the CapacityScheduler is the concept of queues. These queues are typically setup by administrators to reflect the economics of the shared cluster.
- To provide further control and predictability on sharing of resources, the CapacityScheduler supports hierarchical queues to ensure resources are shared among the sub-queues of an organization before other queues are allowed to use free resources, thereby providing affinity for sharing free resources among applications of a given organization.

YARN Web User Interface

- You can use YARN Web interface to monitor clusters, queues, applications, services, and flow activities.
- **Cluster Overview:**
 - o When you open a Cluster Overview page, it should look like as below.



- And provides the below information
- **Cluster Resource Usage by Applications:**
 - o Displays the percentage of cluster resources in use by applications and the percentage available for usage.
- **Cluster Resource Usage by Leaf Queues:**
 - o Displays the percentage of cluster resources in use by leaf queues and the percentage available for usage.
- **Finished Apps From All Users:**
 - o Displays the number of completed, killed, and failed applications.
- **Monitor Running Apps:**
 - o Displays the number of pending and running applications.
- **memory-mb – Usages:**
 - o Displays the amount of used and available memory.
- **vcores – Usages:**
 - o Displays the number of used and available virtual cores.
- **Monitor Node Managers:**
 - o Displays the status of the Node Managers under the following categories:
 - Active
 - Unhealthy
 - Decommissioning

- Decommissioned

Resource Scheduling and Management

- You can manage resources for the applications running on your cluster by allocating resources through scheduling, limiting CPU usage by configuring cgroups, and partitioning the cluster into subclusters using node labels, and launching applications on Docker containers.
- The CapacityScheduler is responsible for scheduling. The CapacityScheduler is used to run Hadoop applications as a shared, multi-tenant cluster in an operator-friendly manner while maximizing the throughput and the utilization of the cluster.
- The ResourceCalculator is part of the YARN CapacityScheduler. If you have only one type of resource, typically a CPU virtual core (vcore), use the DefaultResourceCalculator. If you have multiple resource types, use the DominantResourceCalculator.
- **YARN resource allocation of multiple resource-types:** You can manage your cluster capacity using the Capacity Scheduler in YARN. You can use the Capacity Scheduler's DefaultResourceCalculator or the DominantResourceCalculator to allocate available resources.
- **Hierarchical queue characteristics:** You must consider the various characteristics of the Capacity Scheduler hierarchical queues before setting them up.
- **Scheduling among queues:** Hierarchical queues ensure that guaranteed resources are first shared among the sub-queues of an organization before any remaining free resources are shared with queues belonging to other organizations. This enables each organization to have control over the utilization of its guaranteed resources.
- **Application reservations:** For a resource-intensive application, the Capacity Scheduler creates a reservation on a cluster node if the node's free capacity can meet the particular application's requirements. This ensures that the resources are utilized only by that particular application until the application reservation is fulfilled.
- **Resource distribution workflow:** During scheduling, queues at any level in the hierarchy are sorted in the order of their current used capacity, and the available resources are distributed among them starting with queues that are currently the most under-served.
- **Use CPU scheduling:** Cgroups with CPU scheduling helps you effectively manage mixed workloads.
- **Use GPU scheduling:** On your cluster, you can configure GPU scheduling and isolation. Currently only Nvidia GPUs are supported in YARN. You can use Cloudera Manager to configure GPU scheduling on your cluster.
- **Use FPGA scheduling:** You can use FPGA as a resource type.
- **Limit CPU usage with Cgroups:** You can use cgroups to limit CPU usage in a Hadoop Cluster.
- **Partition a cluster using node labels:** You can use Node labels to partition a cluster into sub-clusters so that jobs run on nodes with specific characteristics.



Chapter-6: Apache Spark

Overview

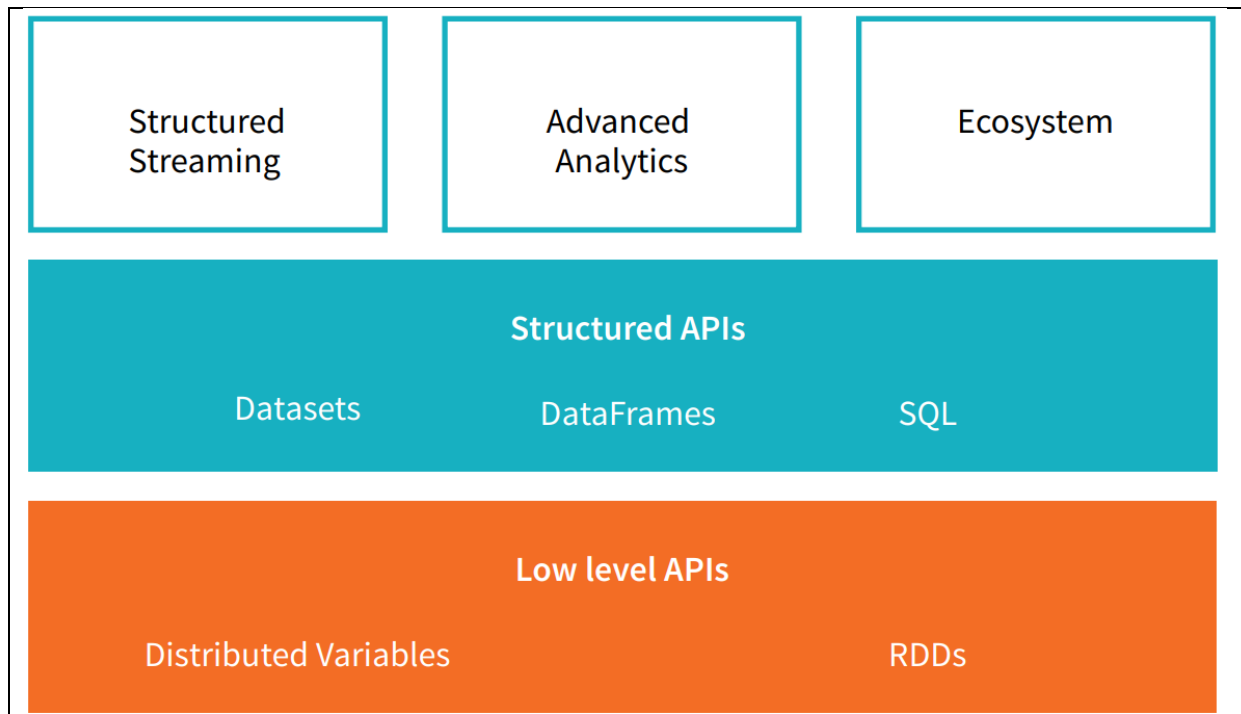
Apache Spark is a broad framework for distributed computing that delivers excellent performance for both batch and interactive processing. It comprises of the Spark core as well as numerous projects that are connected to it and offers application programming interfaces (APIs) for Java, Python, and Scala.

You have the option of using an interactive shell or submitting an application to execute Spark applications locally or distributed over a cluster. Both of these options are available to you. During the data exploration phase as well as for ad hoc analyses, it is usual practise to run Spark applications in an interactive mode.

Spark needs a cluster manager in order to successfully execute applications that are deployed over a cluster. The YARN cluster manager is the only one that Cloudera supports. The YARN ResourceManager and NodeManager roles are responsible for managing Spark application processes while they are being executed on YARN.

Apache Spark is a unified computing engine and a suite of libraries for parallel data processing on computer clusters. Spark has become the tool of choice for any software developer or data scientist that is interested in big data. Spark is capable of running on anything from a laptop to a cluster of thousands of servers, and it offers support for a number of widely used programming languages, including Python, Java, Scala, and R. It also includes libraries that can be used for a variety of tasks, including SQL, streaming, and machine learning. Because of this, it is a simple system to begin with and can easily be expanded to handle massive amounts of data or an extremely vast scale.

An easy-to-understand example of all that Spark has to offer an end user is shown here.



Python, Java, Scala, R, and SQL can utilise Spark. Spark is built in Scala and runs on the Java Virtual Machine (JVM). To run Spark on a laptop or cluster, you require Java 6 or newer. You'll need a Python interpreter to access the API (version 2.6 or newer). You'll need R if you want to utilise it.

Fundamentals of Apache Spark

Unified:

Spark aims to provide a single platform for creating large data applications. Unified means... Spark supports a broad variety of data analytics operations, from basic data loading and SQL queries to machine learning and streaming computation, using the same computational engine and APIs. Real-world data analytics jobs, whether interactive analytics in a Jupyter notebook or conventional software development, integrate several processing kinds and libraries. Spark's cohesive nature makes writing simpler and faster. Spark offers consistent, composable APIs that may be used to create an application from smaller components or existing libraries, and makes it simple to develop your own analytics libraries on top.

Composable APIs aren't enough. Spark's APIs are meant to optimise user programmes' libraries and functions for maximum performance. If you load data using SQL and subsequently analyse a machine learning model using Spark's ML library, the engine may merge both stages into one scan. Spark is a great platform for interactive and production applications because to its broad APIs and high-performance execution.

Spark's emphasis on a single platform mirrors previous software unified platform.

Data scientists use uniform libraries (e.g., Python or R) when modelling, while web developers use united frameworks like Node.js or Django. Before Spark, no open-source system provided a single

engine for parallel data processing, therefore users had to piece together an application from disparate APIs and platforms. Spark rapidly became the industry standard.

Spark's built-in APIs have grown to support new workloads. Developers have also refined the project's unifying engine. This book will concentrate on Spark 2.0's "structured APIs" (DataFrames, Datasets, and SQL) to optimise user applications.

Computing Engine:

Spark is a computational engine that aims towards unification. Spark merely loads data from storage systems and performs computations, not permanent storage. Spark may be utilised with Azure Storage, Amazon S3, Apache Hadoop, Apache Cassandra, and Apache Kafka. Spark neither saves nor prioritises long-term data. Most data is already stored in several systems. Spark performs calculations on data wherever it exists since moving it is costly. Spark tries to make user-facing APIs seem comparable so apps don't have to worry about where their data resides. Spark focuses on computing, unlike Apache Hadoop.

Hadoop's storage technology (the Hadoop file system) and computation system (MapReduce) were tightly interwoven. This option makes it difficult to operate one system without the other or create apps that access data elsewhere. Spark operates well on Hadoop storage, but it's also utilised in contexts where Hadoop design doesn't make sense, such as the public cloud (where storage may be rented separately from compute) or streaming applications.

Libraries:

Spark's libraries leverage on its unified engine architecture to offer a single API for data analysis operations. Spark supports both engine-bundled libraries and third-party open source libraries. Today, Spark's standard libraries form the majority of the open source project; the Spark core engine has evolved little since its inception, but the libraries have developed to give greater capability. Spark comprises SQL and structured data (Spark SQL), machine learning (MLlib), stream processing (Spark Streaming and Structured Streaming), and graph analytics (GraphX). There are hundreds of free source external libraries, from storage interfaces to machine learning techniques. spark-packages.org lists external libraries.

Spark Architecture

When you think of a "computer," you probably picture a desktop device. This computer is great for movies and spreadsheets. As many users know, your computer can't do everything. Data processing is difficult. Single machines can't process large volumes of data (or the user may not have time to wait for the computation to finish).

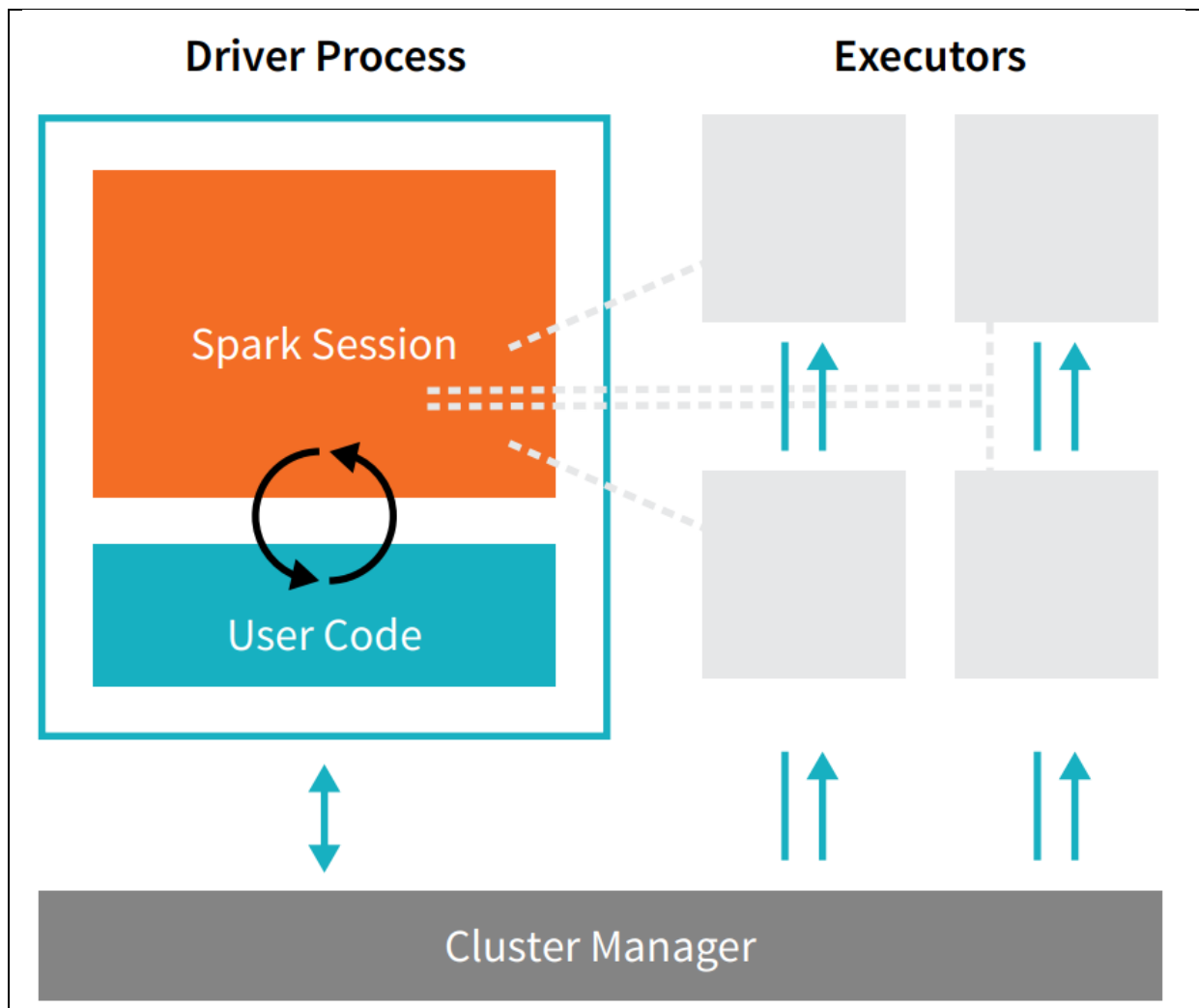
A cluster combines the resources of multiple computers to utilise them as one. A set of machines isn't powerful without a framework to coordinate work. Spark manages and coordinates data tasks across a cluster of computers.

Spark's cluster of computers will be controlled by Spark's Standalone cluster manager, YARN, or Mesos. The cluster administrators then provide Spark applications resources so we may finish our job.

Spark Applications

The components that make up a Spark Application are referred to as the "driver process" and the "executor processes." The driver process is responsible for three things: keeping information about the Spark Application; reacting to a user's programme or input; and analysing, distributing, and scheduling work among the executors. It resides on a node in the cluster and performs your main() function (defined momentarily). The driver process is critically necessary since it is the core of a Spark Application and it is responsible for keeping all of the important information updated during the application's lifetime.

The executors are the ones who are accountable for carrying out the task that has been delegated to them by the driver. This indicates that each executor is solely responsible for two things: running the code that has been given to it by the driver, and reporting back to the driver node the current status of the computation that is taking place on that executor.



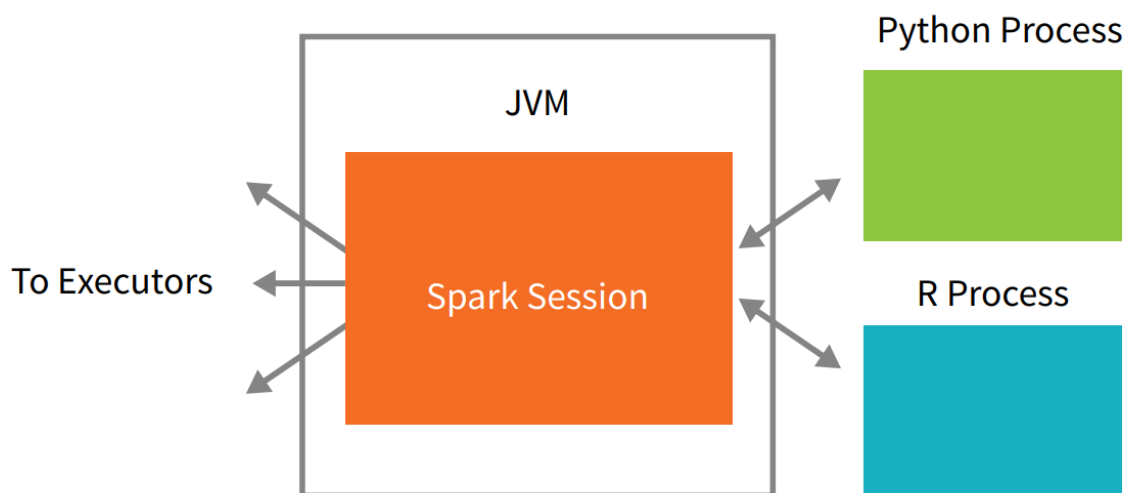
The cluster manager is responsible for managing the underlying hardware and allocating resources to Spark applications. This might be Spark's standalone cluster manager, YARN, or Mesos. Alternatively, it could be one of numerous other fundamental cluster managers. This indicates that a cluster is capable of simultaneously hosting many instances of a Spark application running at the

same time. In the next section of this book, titled "Part IV: Production Applications," we will have a more in-depth discussion on cluster managers.

From the preceding example, we can see that our driver is located on the left, while the four executors are located on the right. In this particular design, the idea of cluster nodes has been eliminated. Through the use of settings, the user has the ability to determine how many executors should land on each node.

Spark may operate in both a cluster and a local mode. The cluster mode is the default. Because both the driver and the executors are just processes, it is possible for them to coexist on the same system or on entirely distinct computers. When operating in the local mode, both of them execute (as threads) on your own machine rather than in a cluster. We prepared this book keeping in mind local mode, which means that everything should be able to be executed on a single system.

Spark may operate in both a cluster and a local mode. The cluster mode is the default. Because both the driver and the executors are just processes, it is possible for them to coexist on the same system or on entirely distinct computers. When operating in the local mode, both of them execute (as threads) on your own machine rather than in a cluster. We prepared this book keeping in mind local mode, which means that everything should be able to be executed on a single system.



Each language's application programming interface (API) will adhere closely to the key notions that we outlined before. Spark code will be executed via the user's SparkSession, which is made accessible to the user. The SparkSession will serve as the entry point. When using Spark from within Python or R, the user never writes explicit instructions for the JVM; rather, the user writes code in Python and R that Spark will translate into code that Spark can then run on the executor JVMs. This is the case even when the user is using Spark from within Python or R.

DataFrames

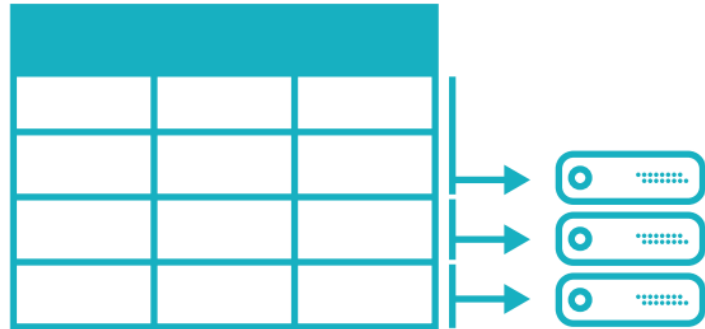
A DataFrame, which is the most popular kind of Structured API, is essentially a representation of a data table that has rows and columns. The schema consists of the list of columns as well as the types included inside those columns. A straightforward comparison would be a spreadsheet that has labelled columns. A spreadsheet is stored on a single computer at a single place, but a Spark DataFrame may be distributed over thousands of machines. This is the primary distinction between

the two. It should be obvious why the data are being stored on several computers: either the data are too massive to fit on a single system or the calculation would simply take too long to complete on a single machine.

Spreadsheet on a single machine



Table or DataFrame partitioned across servers in a data center



The idea of a DataFrame is not one that is exclusive to Spark. R and Python have a number of conceptual similarities. On the other hand, Python/R DataFrames, with a few notable exceptions, are stored on a single system as opposed to numerous workstations. This restricts what you are able to do in Python and R with a particular DataFrame to the resources that are available on that particular system. On the other hand, due to the fact that Spark includes language interfaces for both Python and R, converting Pandas (Python) DataFrames to Spark DataFrames and R DataFrames to Spark DataFrames is a rather simple process (in R).

In addition to Datasets and Dataframes, Spark now supports SQL Tables and Resilient Distributed Datasets as fundamental abstractions (RDDs). Despite the fact that each of these abstractions represents a distinct dispersed collection of data, they nonetheless have unique interfaces for interacting with that data. DataFrames are the most user-friendly and productive option; moreover, they are accessible in every language.

Partitions

Spark divides the data into sections that it calls partitions in order to make it possible for all of the executors to carry out their tasks simultaneously. A partition in our cluster is a set of rows that are hosted on one of our physical machines. The divisions of a DataFrame each represent a different way.

In the course of the execution, the data will be physically dispersed among all of your cluster's devices. Spark will work if you just have one partition. even if you have thousands of executors, you will only have a parallelism of one for the whole process. In the event that you have a single partition while having numerous Because there is just one computing resource, executor Spark will still only have a parallelism of one.

When working with DataFrames, one crucial point to keep in mind is that, for the most part, we do not actively change the divisions of the data. (with regard to each person) In the physical partitions, we only provide the high-level changes of the data, and Spark takes care of the rest. Decides how the task will be carried out in its entirety throughout the cluster. Lower level application

programming interfaces are available (through the Resilient Distributed Object interface for datasets), and we go through them in Part III of this book.

Transformations

The fundamental data structures in Spark are immutable, which means that they cannot be altered once they have been generated. If you are unable to modify it in any way, what are you going to do with it then? This is something that may seem unusual at first. In order to "alter" a DataFrame, you will need to inform Spark on the manner in which you would want to transform the DataFrame that you now have into the DataFrame that you desire.

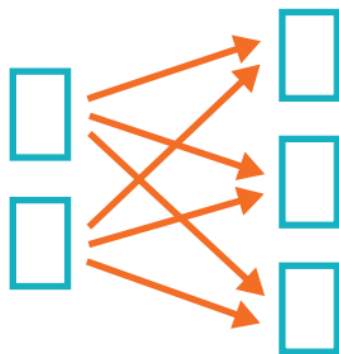
The name for these kinds of instructions is "transformations." Let's carry out a straightforward transformation to discover all even integers included inside the present DataFrame.

Wide Dependency:

A transformation that follows the broad dependence (or wide transformation) style will have numerous input partitions leading to the production of many output partitions. This process, in which Spark will trade partitions throughout the cluster, is often known to as a shuffle, and you will frequently hear it referred to as such. When narrow transformations are used, Spark will automatically carry out an action known as pipelining on narrow dependencies. This implies that if we define many filters on DataFrames, they will all be carried out in-memory if we use narrow transformations. One cannot make the same statement about shuffles. When we carry out a shuffle, Spark will save the outcomes of the operation on disc. Because shuffle optimization is such an important issue, you'll likely come across many discussions about it on the internet, but for the time being, all you really need to know is that there are two different types of transformations.

Wide Transformations (shuffles)

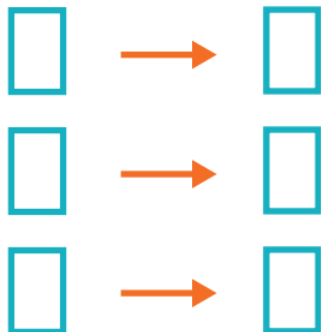
1 to N



Narrow Dependency: Narrow transformations, also known as transformations with narrow dependencies, are ones in which each input partition will only contribute to a single output partition. We'll refer to these transformations as narrow. Our where statement in the previous piece of code establishes a restricted dependence, meaning that just one partition contributes to the maximum of one output partition.

Narrow Transformations

1 to 1



Lazy Evaluation

A graph of processing instructions will not be carried out by Spark until the very last possible second because it uses a technique known as "lazy evaluation." When we do an operation in Spark, rather than instantly affecting the data, we develop a plan of transformations that we would want to apply to our source data. This plan is then executed on our source data. Spark, by delaying the execution of the code until the very last possible moment, will convert this plan from your raw DataFrame transforms into a physically efficient plan that will run as efficiently as possible throughout the cluster. Because of this, the end user may reap enormous advantages as a result of Spark's ability to optimise the whole data flow from beginning to finish. On DataFrames, there is a feature known as "predicate pushdown" that serves as an illustration of this concept. If we design a huge Spark job but provide a filter at the end that only needs us to get one row from our source data, the most effective approach to carry this out is to access the single record that we needed. This is because the filter only requires us to fetch one row. Spark will actually assist us in optimising this situation by lowering the filter on its own own.

Actions

We are able to construct a more rational transformation plan as a result of transformations. We need to do some action in order to start the calculation. Spark is given the instruction to calculate a result based on a sequence of transformations through an action. The count action is the simplest one, and it provides us with the total number of records included in the DataFrame.

```
divisBy2.count()
```

Now we can see the outcome! It should come as no surprise that there are 500 numbers between 0 and 999 that are divisible by two. Now, counting isn't the only thing you can do. The following are the three categories of actions:

actions to collect data to native objects in the appropriate language; actions to see data in the console; actions to write to output data sources; and actions to view data in the console.

When we were specifying our action, we started a Spark job that runs our filter transformation, which is a narrow transformation. This is followed by an aggregation, which is a wide transformation, that performs the counts on a per-partition basis. Finally, a collect with brings our result to a native

object in the appropriate language. All of this is viewable by checking the Spark UI, which is a tool that is included with Spark and gives us the ability to monitor the Spark tasks that are currently executing on a cluster.

DataFrames in addition to SQL

In the last example, we went through the steps of a straightforward example; now, let's go through the steps of a more complicated example while following along in both DataFrames and SQL. Spark the same modifications in precisely the same manner regardless of the language that is being used. Before Spark actually executes your code, you have the option of expressing your business logic in SQL or DataFrames (written in either R, Python, Scala, or Java), and Spark will compile that logic down to an underlying plan (which can be seen in the explain plan). You, as a user of Spark SQL, have the ability to register any DataFrame as a table or view (a temporary table), and then query that DataFrame with standard SQL. There is no difference in performance between writing SQL queries and writing DataFrame code since they both "compile" to the same underlying plan that we define in DataFrame code. This is the case because DataFrame code is compiled.

With one easy method call, a DataFrame may be transformed into a table or view of your choosing.



Contents

Chapter-7: Apache Impala	1
Overview	1
Benefits of using Impala.....	2
How Impala Works with Hadoop	2
Impala and Query Execution.....	2
Impala and Hive	3
A Brief Introduction to Impala Metadata and the Metastore	3
How Impala Uses HDFS	4
How Impala Uses HBase.....	4
Constituent parts of the Impala	5

Chapter-7: Apache Impala

Overview

On data that is stored in common Apache Hadoop file formats, the SQL queries that are provided by the Apache Impala project provide great speed and low latency. Instead of the lengthy batch processes that are typically associated with SQL-on-Hadoop technologies, which are made possible by the slow response times for queries, interactive exploration and fine-tuning of analytic queries are now possible. (You will often come across the word "interactive" being used to different sorts of quick queries with response times on a human-scale.)

Both Apache Hive and Impala are able to exchange their databases and tables because to Impala's integration with the Apache Hive metastore database. Because of Impala and Hive's high degree of interaction with one another and their compliance with the HiveQL syntax, you can use either Impala or Hive to build tables, run queries, load data, and do other similar tasks.

The following is a list of some of the primary benefits offered by the Impala:

- Because Impala is integrated with the preexisting CDH ecosystem, it is possible to store, distribute, and retrieve data by using the many solutions that are included with CDH. This prevents the creation of data silos and lowers the need for costly data transfer.
- Impala allows users to have access to data that is stored in CDH without having them to have the Java expertise that is necessary for MapReduce tasks. The HDFS file system is immediately accessible to Impala, allowing for direct data retrieval. In addition to this, Impala has a SQL front-end that may be used to retrieve data stored in the HBase database system or in the Amazon Simple Storage System (S3).

- Impala queries often provide results within seconds or a few minutes, but Hive queries can take several minutes or even hours to finish. Impala queries also deliver results much more quickly.
- Parquet is a columnar storage architecture that is suited for large-scale queries, which are prevalent in data warehouse applications. Impala is a pioneer in the usage of the Parquet file format, which was developed by Facebook.

Impala enables you to do SQL queries directly on your Apache Hadoop data while it is stored in HDFS, HBase, or the Amazon Simple Storage Service. These searches are both quick and interactive (S3). Impala utilises the same metadata, SQL syntax (referred to as Hive SQL), ODBC driver, and user interface (referred to as the Impala query UI in Hue) as Apache Hive. This is in addition to using the identical unified storage platform.

This offers a platform that is both well-known and unified, and it can be used for real-time or batch-oriented queries.

Impala is a new tool that was recently added to the arsenal of options for querying large amounts of data. Batch processing frameworks like Hive that are based on MapReduce are not rendered obsolete by Impala's introduction. Batch operations that run over an extended period of time, such as those requiring the batch processing of Extract, Transform, and Load (ETL) type processes, are the optimum use case for Hive and other frameworks that are based on MapReduce.

Benefits of using Impala

Impala gives you the following benefits: • A SQL interface that data scientists and analysts are already familiar with.

- The ability to query large amounts of data in Apache Hadoop (also known as "big data").
- Queries that are executed in a decentralised manner inside a cluster setting, which enables simple scalability and the use of commodity hardware that is more cost-effective.
- The capability to exchange data files across multiple components without the need for any steps including copying or exporting and importing the data; for instance, to write with Pig, convert with Hive, and query with Impala. Because Impala can read from and write to Hive tables, it is possible to do analytics on Hive-produced data while using Impala for basic data exchange.
- A unified platform for the processing and analysis of large amounts of data, allowing clients to save unnecessary expenditures on modelling and ETL.

How Impala Works with Hadoop

The following are the components that make up the Impala solution:

- Clients - Impala's ability to interface with external entities is enabled through clients such as Hue, ODBC clients, JDBC clients, and the Impala Shell.
- Typically, queries and administrative actions like connecting to Impala are carried out with the assistance of these interfaces.
- The Hive Metastore is where information about the data that Impala has access to is stored. For instance, the metastore provides Impala with information on the databases that are accessible as well as the layout of those databases. The relevant metadata changes are automatically broadcast to all Impala nodes by the dedicated catalogue service that was

introduced in Impala 1.2. This occurs whenever you make changes to schema objects, such as creating, dropping, or altering them; loading data into tables; and so on through Impala SQL statements.

- The Impala process is responsible for query coordination and execution. It is a process that runs on DataNodes. Impala clients' queries may be received, planned, and coordinated by each instance of Impala running. The queries are subsequently divided over the several Impala nodes, which then take on the role of workers to carry out the simultaneous query fragments.
- HBase and HDFS are used for storing data that may be searched later.

Impala and Query Execution

The following procedures are used to handle queries that are run using Impala:

1. User applications use ODBC or JDBC, which both offer standardised querying interfaces, to deliver SQL queries to Impala. Any impalad in the cluster is available for the user application to connect to. This impalad will serve as the query's coordinator going forward.
2. After the query has been parsed and analysed by Impala, it is determined what actions need to be carried out by various impalad instances located across the cluster. Planning goes into the execution to ensure maximum effectiveness.
3. In order to retrieve data, local instances of impalad connect to several services, such as HDFS and HBase.
4. The results of each impalad are sent to the client via the coordinating impalad, which receives the data from each impalad.

Impala and Hive

Impala takes use of a variety of Hadoop ecosystem components that are already well-known to users. Since Impala is capable of exchanging data with other Hadoop components in both a consumer and producer capacity, it is able to integrate itself into your ETL and ELT pipelines in a variety of different ways.

An important objective of the Impala project is to improve the speed and effectiveness of SQL-on-Hadoop operations to the point where they become appealing to new sorts of users and open up Hadoop to new kinds of use cases. It takes use of preexisting Apache Hive infrastructure whenever it is feasible to do so in order to carry out long-running, batch-oriented SQL queries; this infrastructure is already in place for many Hadoop users.

Impala, in particular, stores the definitions of its tables in a regular MySQL or PostgreSQL database that is referred to as the metastore. This is the same database that Hive uses to store information of this kind. Therefore, Impala is able to access tables that have been created or loaded by Hive, so long as all columns utilise data types, file formats, and compression codecs that are supported by Impala.

As a result of the original emphasis placed on the features and efficiency of queries, Impala is able to read a wider variety of data types using the SELECT statement than it is able to create using the INSERT statement. Hive must be used to load the data before it can be queried via file formats such as Avro, RCFile, or SequenceFile.

The Impala query optimizer has the ability to use table statistics in addition to column statistics. Initially, you acquired this information by using the ANALYZE TABLE command in Hive; however,

beginning with version 1.2.2 of Impala and above, you should use the COMPUTE STATS statement instead. The COMPUTE STATS tool needs less setup, is more stable, and does not require the user to move between the Impala shell and the Hive shell at any point throughout the process.

A Brief Introduction to Impala Metadata and the Metastore

In the section titled "How Impala Works with Hive," located on page 19, it is said that Impala stores information on table definitions in a central database called as the "metastore." Additionally, Impala monitors the following additional information for the low-level properties of data files:

- The actual locations, inside HDFS, of each block in the file system.

When a table has a significant amount of data and/or a big number of partitions, obtaining all of the table's information may be a time-consuming process that, in certain instances, might take several minutes. Therefore, all of this information is stored in the cache of each Impala node so that it may be reused for subsequent searches against the same database.

Before a query can be issued against a table, all of the other Impala daemons in the cluster need to receive the most recent metadata, which will replace any out-of-date cached metadata. This is necessary in the event that either the table definition or the data contained within the table is modified. All DDL and DML statements that are executed via Impala are subject to an automated metadata update, which is handled by the catalogd daemon and begins with version 1.2 of the Impala database. For more information, please refer to page 17 of The Impala Catalog Service.

You should still use the REFRESH statement (when new data files are added to existing tables) or the INVALIDATE METADATA statement (for entirely new tables or after dropping a table, performing an HDFS rebalance operation, or deleting data files) when performing DDL and DML operations through Hive or making changes manually to files in HDFS. The retrieval of metadata for all tables that are monitored by the metastore is performed when the INVALIDATE METADATA command is issued on its own. If you are aware that only some tables have been modified outside of Impala, you may use the command REFRESH table name for each table that has been impacted to only obtain the most recent metadata for the tables that have been modified.

How Impala Uses HDFS

As its primary form of data storage, Impala makes extensive use of the distributed filesystem HDFS. When it comes to protecting itself from failures in hardware or networks on individual nodes, Impala depends on the redundancy offered by HDFS. The information included inside Impala tables is stored in HDFS in the form of data files, and these files make use of the standard HDFS file formats and compression codecs. Impala will read all of the data files that are present in the directory for a new table, notwithstanding the fact that each file has a different name. Impala assigns new names to existing files before adding new data to them.

How Impala Uses HBase

An alternate storage media for Impala data is HBase, which is an alternative to HDFS. It is a database storage system that is built on top of HDFS, however it does not have any built-in support for SQL. A great number of Hadoop users already have it installed and store enormous data sets in it that are often sparse. You will be able to query the contents of HBase tables using Impala if you first define

such tables in Impala and then map them to similar tables in HBase. You may even conduct join queries that include both Impala and HBase tables in the results set.

SQL queries that are run on data that is stored in common Apache Hadoop file formats may be executed by the Apache Impala with high speed and low latency.

The following is an inventory of the components that make up the Impala solution.

Impala: The Impala service is responsible for the coordination and execution of queries that are received from users. The queries are subsequently divided over the several Impala nodes, which then take on the role of workers to carry out the simultaneous query fragments.

The Hive Metastore: is where information on the data that is accessible to Impala is stored. For instance, the metastore provides Impala with information on the databases that are accessible as well as the layout of those databases. Impala's dedicated catalogue service is responsible for automatically broadcasting any relevant metadata changes to all of Impala's nodes whenever you make changes to the schema (such as creating, dropping, or altering objects), load data into tables, or perform any other operation using SQL statements.

Clients: Interactions with Impala may be carried out by a wide variety of entities, including Hue, ODBC clients, JDBC clients, Business Intelligence applications, and the Impala Shell. Typically, queries and administrative actions like connecting to Impala are carried out with the assistance of these interfaces.

Constituent parts of the Impala

The Impala service is a massively parallel processing (MPP) database engine that operates over a distributed environment. It is made up of a variety of daemon processes, each of which operates on a distinct host inside your Hadoop cluster.

The Impala service is comprised of the following groups of processes, which are together referred to as roles.

Impala Daemon

The Impala daemon, which is physically embodied by the `impalad` process, is the essential component of the Impala system. A few of the most important tasks that are carried out by an Impala daemon are as follows:

- Performs reading and writing operations on data files.
- It'll take queries sent through the `impala-shell` command, Hue, JDBC, or ODBC if you choose to use it.
- Performs the queries in parallel and distributes the work evenly among the cluster.
- Sends intermediate query results to the central coordinator for review.
- One of the following strategies might be used to bring about the appearance of Impala daemons:
 - Both HDFS and Impala are hosted on the same physical machine, and each Impala daemon is executed on the same host as a DataNode.
 - Impala is installed independently in a computing cluster, and it retrieves data from HDFS, S3, ADLS, and other locations through remote connections.

StateStore

The Impala StateStore continually monitors the health of all Impala daemons in a cluster. It's a statestored daemon process. One cluster host needs this procedure. If one Impala daemon becomes offline due to hardware failure, network fault, software issue, or other cause, the StateStore notifies all other Impala daemons so subsequent queries may bypass the inaccessible daemon.

StateStore is not necessarily crucial to the regular functioning of an Impala cluster since it helps when things go wrong and broadcasts information to coordinators. If the StateStore is not functioning or unavailable, Impala daemons continue running and distributing tasks as normal with known data. If additional Impala daemons die, the cluster becomes less resilient and metadata becomes less consistent. StateStore reestablishes contact with Impala daemons and continues monitoring and broadcasting when it comes back online.

Queries that access the new object produced by a DDL statement while the StateStore is offline will fail.

Impala Server

The Catalog Server sends Impala SQL statement information updates to all Impala daemons. It's a catalogd daemon process. One cluster host needs this procedure. Because queries are transmitted via StateStore, statestored and catalogd should execute on the same host.

When Impala statements alter metadata, the catalogue service avoids REFRESH and INVALIDATE METADATA commands. Before running a query on an Impala node after creating a table, loading data, etc. in Hive, you must perform REFRESH or INVALIDATE METADATA.



Contents

Chapter-8: Apache OOZie	1
Overview	1
Oozie Workflows	1
Oozie Architecture	3
Use-Cases of Apache Oozie	3
Oozie Editors	3
Hue Editor for Oozie	3
Oozie Eclipse Plugin (OEP)	4
Oozie Workflow	5

Chapter-8: Apache OOZie

Overview

Oozie apps are comparable to executables found in Unix, whereas Oozie jobs are comparable to the processes found in Unix. Users of Oozie create applications, and each individual runthrough of an application is referred to as a job.

Apache Oozie Workflow Scheduler for Hadoop is a workflow and coordination tool for managing Apache Hadoop processes, including the following:

- Directed acyclic graphs (DAGs) of actions are what make up Oozie Workflow tasks. Generally speaking, actions are Hadoop jobs (MapReduce, Streaming, Pipes, Pig, Hive, Sqoop, etc).
- Recurring Workflow tasks may be triggered by Oozie Coordinator jobs depending on the time (or frequency) and the availability of data.
- Oozie Bundle jobs are collections of Coordinator tasks that are treated as a single job for the purposes of management.

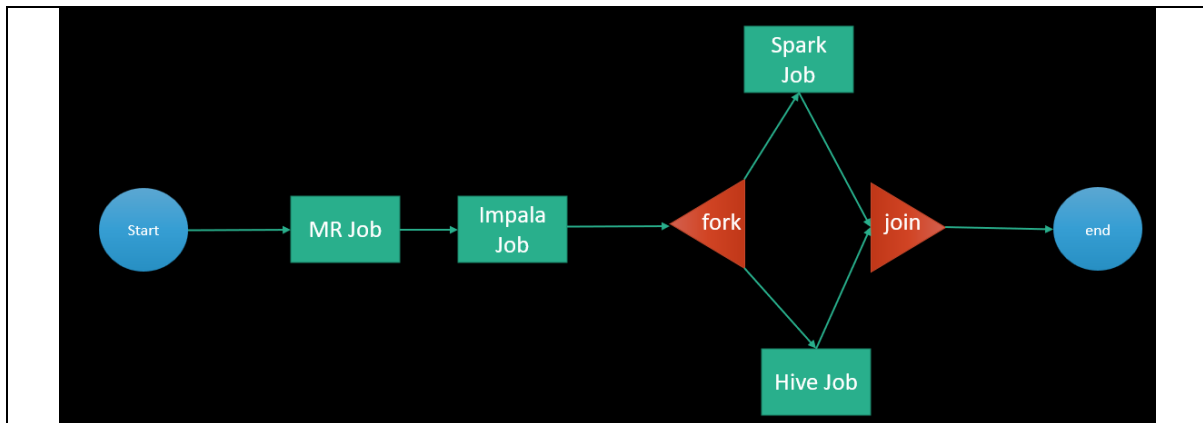
You may utilise Oozie, which is an extensible, scalable, and data-aware service, to coordinate dependencies across tasks that are operating on Hadoop.

Oozie Workflows

A Hadoop task that consists of many stages is known as an Oozie workflow. A workflow is a collection of action nodes and control nodes that are organised in a directed acyclic graph (DAG) that captures control dependence. Each action in a workflow is often a Hadoop job (for example, a MapReduce, Pig, Hive, or Sqoop job), and a DAG records the control dependency. In addition, there may be activities that are not classified as Hadoop jobs (e.g., a Java application, a shell script, or an email notification).

The sequence in which these activities are carried out is determined by the arrangement of the nodes in the process. In a process, an activity will not begin until the action that came before it has completed.

It is up to the control nodes in a process to regulate the order in which activities are carried out. The beginning and the conclusion of a process are denoted by the start control node and the end control node, respectively. The control nodes known as fork and join make it possible to do out tasks simultaneously. The decision control node functions similarly to a switch or case statement in that it may pick a certain execution route inside the work flow by making use of information gleaned directly from the task. An example of a work flow is shown.



Workflows are directed acyclic graphs, which means that they do not support loops in the flow of information.

Oozie is a web-based application written in Java that serves as a workflow manager and coordinator. It is used to manage and coordinate activities in the Hadoop ecosystem. The structure of its work flow is quite similar to that of a Direct Acyclic Graph (DAG)[1]. Oozie is capable of managing thousands of tasks in a Hadoop cluster because to its scalable design.

There are three standard occupations available in Oozie.

1. Oozie Workflow Jobs: These jobs indicate the order in which actions are to be carried out.
2. Oozie Coordinator Jobs: Coordinates the task and determines when it should be activated based on factors such as time and data availability.
3. The Oozie Bundle is a package consisting of numerous Workflow and Coordinator applications.

Action node and control flow node are both components of the Oozieworkflow.

Action node: An action node is a representation of a workflow activity, such as importing data using Sqoop, putting files into HDFS, importing data using MapReduce, Pig, or Hive tasks, or executing a shell script for a programme that was built in Java. Action nodes are responsible for making decisions on how jobs are carried out.

Control-flow node: A control-flow node directs the flow of work from one action to the next inside a workflow by permitting features such as conditional logic, which allows for the workflow to take one of many possible paths based on the outcome of a previous action node.

There is just one control node, but the number of jobs determines how many action nodes will be created. The number of jobs and the action node's own count will always be equal.

Oozie Architecture

The Oozie server is implemented as a Java web application, and all of the required data is saved in a database. Any kind of database will do, whether it Derby, MySQL, or Oracle, for example. Hadoop cluster serves as the repository for all tasks. Oozie customer contact to the Oozie server, which will be responsible for maintaining and processing the tasks. Following the processing of the data, valuable information is saved in the database. Hadoop cluster serves as the repository for all tasks. The Oozie client makes contact with the Oozie server so that jobs can be managed and processed.

Following the processing of the data, valuable information is saved in the database.

Use-Cases of Apache Oozie

Apache Oozie is used by Hadoop system administrators to run complex log analysis on HDFS. Hadoop Developers use Oozie for performing ETL operations on data in a sequential order and saving the output in a specified format (Avro, ORC, etc.) in HDFS. In an enterprise, Oozie jobs are scheduled as coordinators or bundles.

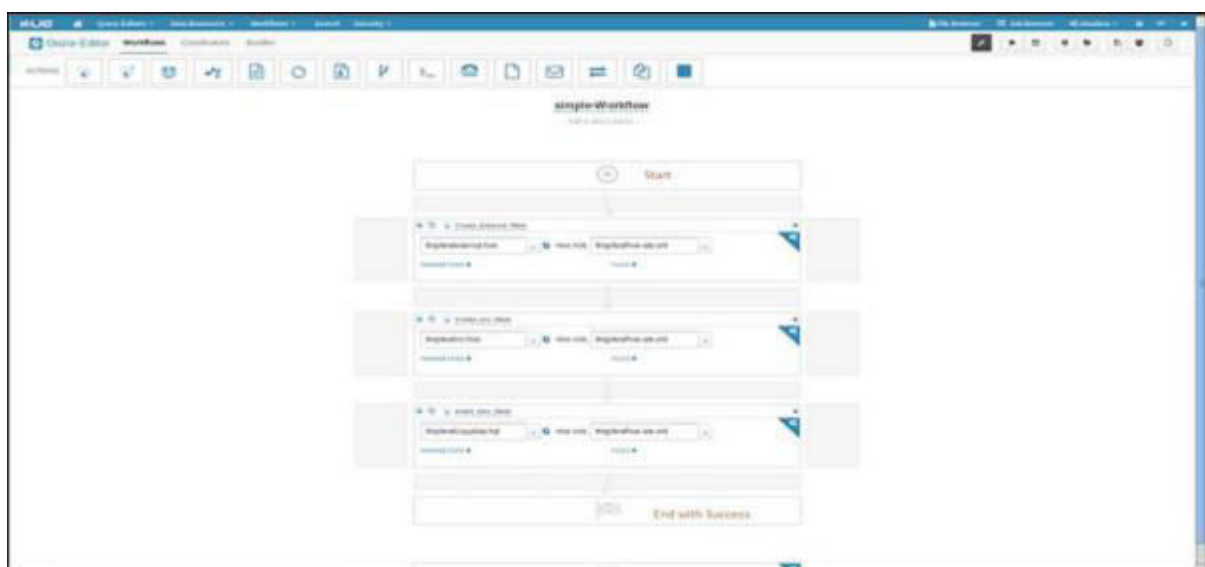
Oozie Editors

Before we dive into Oozie lets have a quick look at the available editors for Oozie. Most of the time, you won't need an editor and will write the workflows using any popular text editors (like Notepad++, Sublime or Atom) as we will be doing in this tutorial.

But as a beginner it makes some sense to create a workflow by the drag and drop method using the editor and then see how the workflow gets generated. Also, to map GUI with the actual workflow.xml created by the editor. The most popular among Oozie editors is Hue.

Hue Editor for Oozie

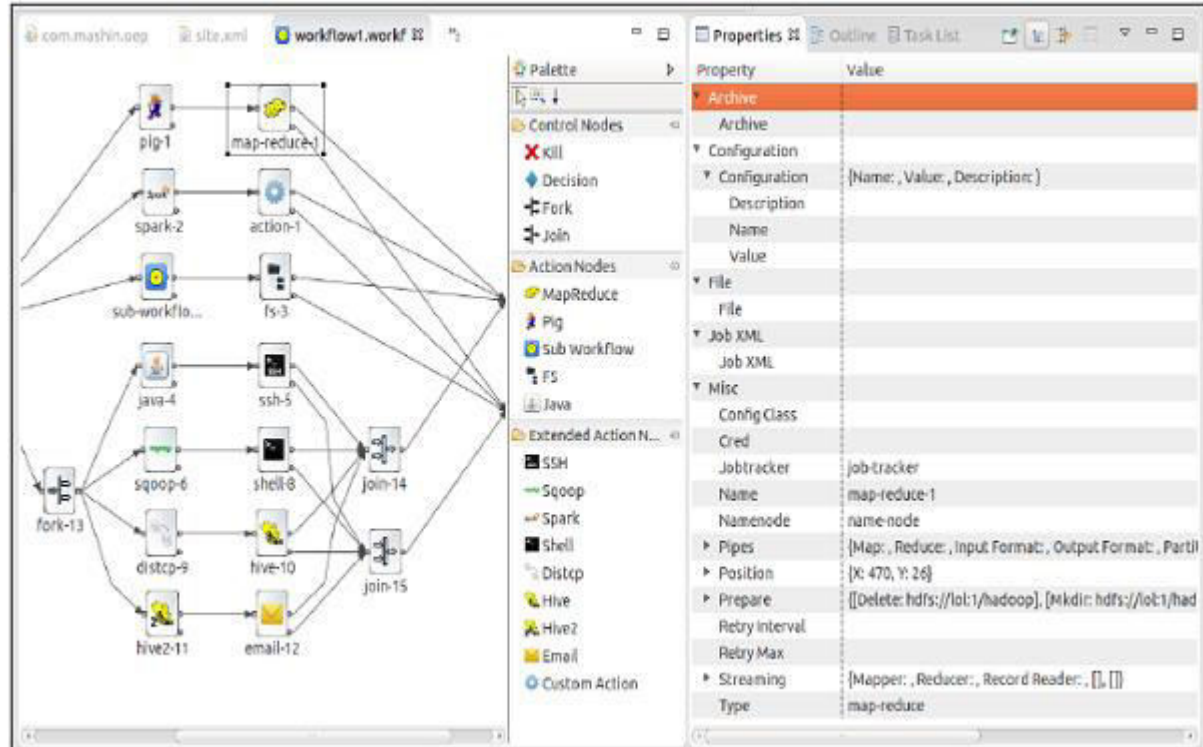
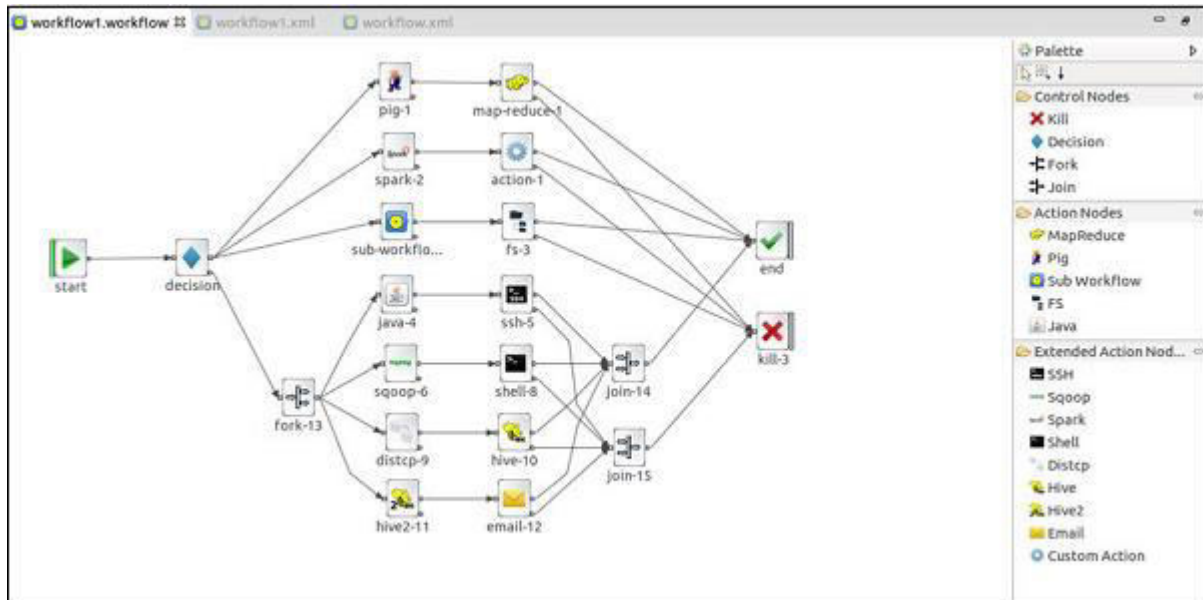
This editor is very handy to use and is available with almost all Hadoop vendors' solutions. The following screenshot shows an example workflow created by this editor.

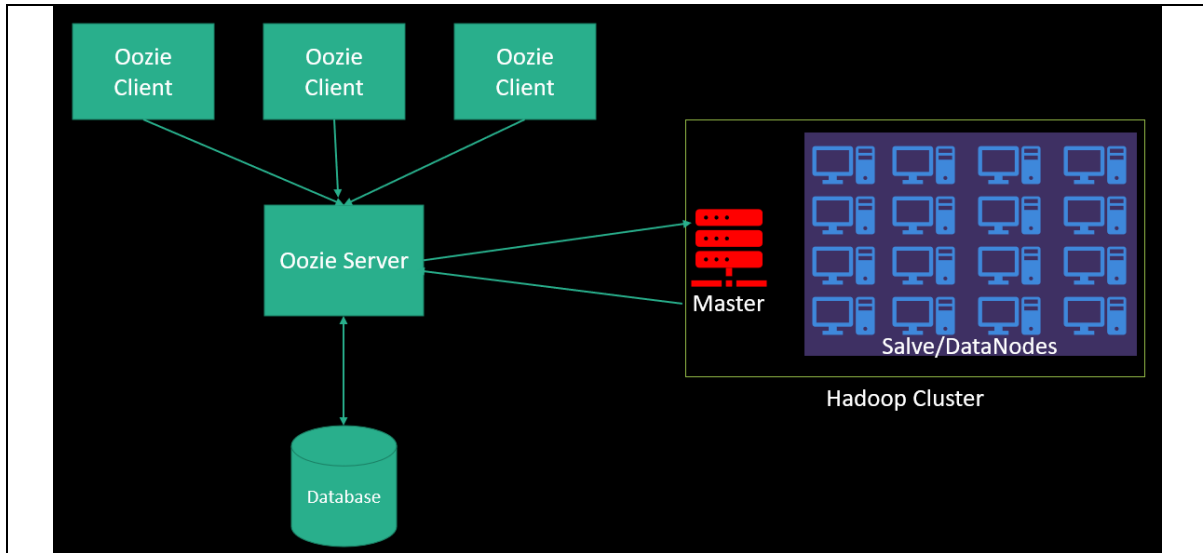


You can drag and drop controls and actions and add your job inside these actions.

Oozie Eclipse Plugin (OEP)

Oozie Eclipse plugin (OEP) is an Eclipse plugin for editing Apache Oozie workflows graphically. It is a graphical editor for editing Apache Oozie workflows inside Eclipse. Composing Apache Oozie workflows is becoming much simpler. It becomes a matter of drag-and-drop, a matter of connecting lines between the nodes. The following screenshots are examples of OEP.





Oozie Workflow

The OOZIE Workflow is a set of actions that are organised in a DAG. Definition of an Oozie process written in the hPDL language. The Oozie process has a set of nodes, including the Start control node, the End control node, the Kill control node, the Decision node, the Fork node, and the Join node.

- a. Start control node: Every Oozie workflow has to have a start control node, and the workflow will always begin execution from the start control node.
- b. End control node: Once the task has been successfully finished, it will proceed to the end control node. When you reach the end control node, it indicates that there were no errors.
- c. Kill control node: If we want to stop the workflow from being executed, then we need to utilise the kill control node. It is possible that there are many kill control nodes.



Contents

Chapter-9: Apache Kafka	1
Overview	1
Pub Sub System.....	1
Kafka Architecture	2
Topics	2
Brokers	3
Records.....	4
Partitions.....	4
KAFKA REAL TIME APPLICATIONS	6

Chapter-9: Apache Kafka

Overview

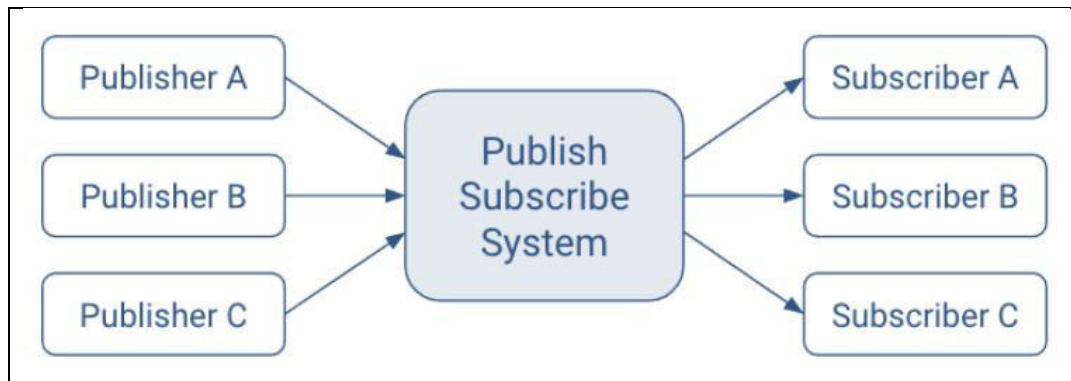
Apache Kafka is a platform for continuously receiving messages. It is built with high performance, high availability, and redundancy in mind from the beginning.

The following are some examples of apps that may leverage such a platform:

- Things Connected to the Internet Telemetry data may be sent back to a server via the Internet from a variety of devices, including televisions, refrigerators, washing machines, dryers, thermostats, and personal health monitors.
- Sensing and Control Networks. It is possible to equip both expansive environments (such as farms, amusement parks, and woods) and intricate machines (such as engines) with a variety of sensors to monitor data or the present state of the environment.
- Positional Data. Sending location data to a centralised platform is possible for delivery vehicles as well as massively multiplayer online games.
- Any Additional Real-Time Data. Satellites and medical sensors both have the ability to transmit data to a centralised location for further analysis.

Pub Sub System

The ideal publish-subscribe system is one that is simple and uncomplicated: the messages published by Publisher A need to find their way to Subscriber A, the messages published by Publisher B need to find their way to Subscriber B, and so on.



- Unlimited Lookback is a perk that should be included in any ideal system. At any given moment in time, a new Subscriber A1 is able to read the stream that is being published by Publisher A.
- Message Retention. No messages are lost.
- No limits on the storage space. Messages can be stored in the publish-subscribe system in an unlimited capacity.
- Absolutely no downtime. The publish-subscribe system never goes offline for maintenance.
- There are no scaling limits. The publish-subscribe system is able to accommodate an unlimited number of publishers and/or subscribers while maintaining a consistent delivery latency for messages.

Kafka Architecture

The design of Kafka deviates from that of the ideal publish-subscribe system, as is the case with all systems that exist in the actual world. Some of the most important distinctions are as follows:

- A replicated and distributed commit log serves as the foundation upon which messaging is built.
- Because the customer now has additional functionalities, they are also responsible for a greater amount.
- Instead of individual messages, batch messaging is tuned to work more efficiently.
- Messages are kept even after they have been eaten, and users have the ability to consume them again.

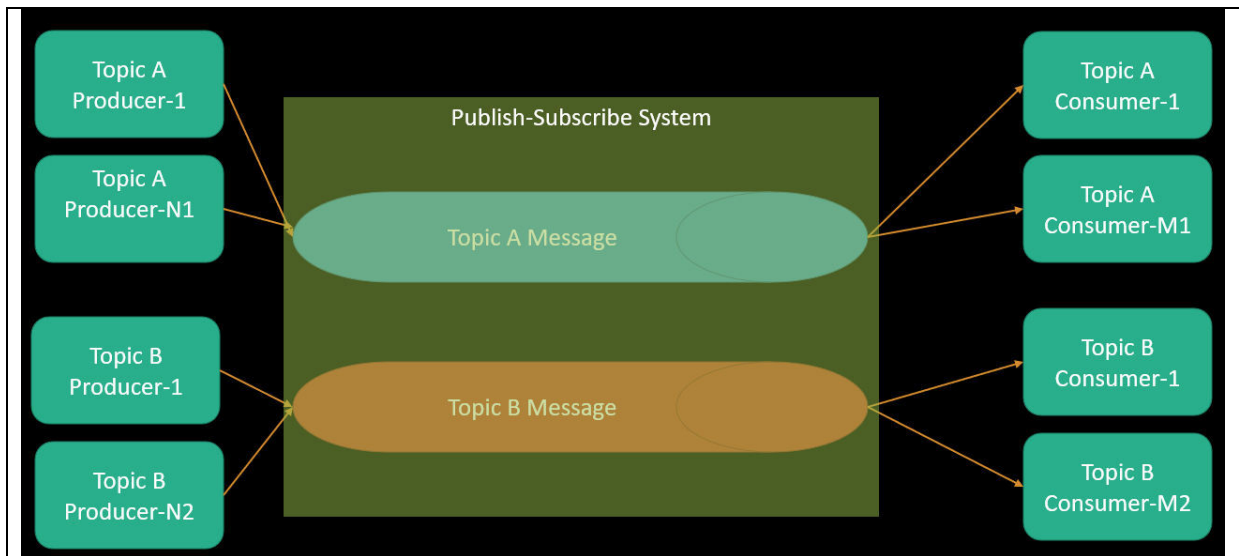
These design choices have led to the following outcomes:

- Extreme horizontal scalability
- Very high throughput
- High availability
- However, distinct semantics and message delivery guarantees are guaranteed.

Topics

In the hypothetical system that was just described, messages sent out by a single publisher would magically make it to the inboxes of each subscriber.

Kafka incorporates the idea of a topic into his writing. The use of a topic makes it easier for publishers and subscribers to find each other.



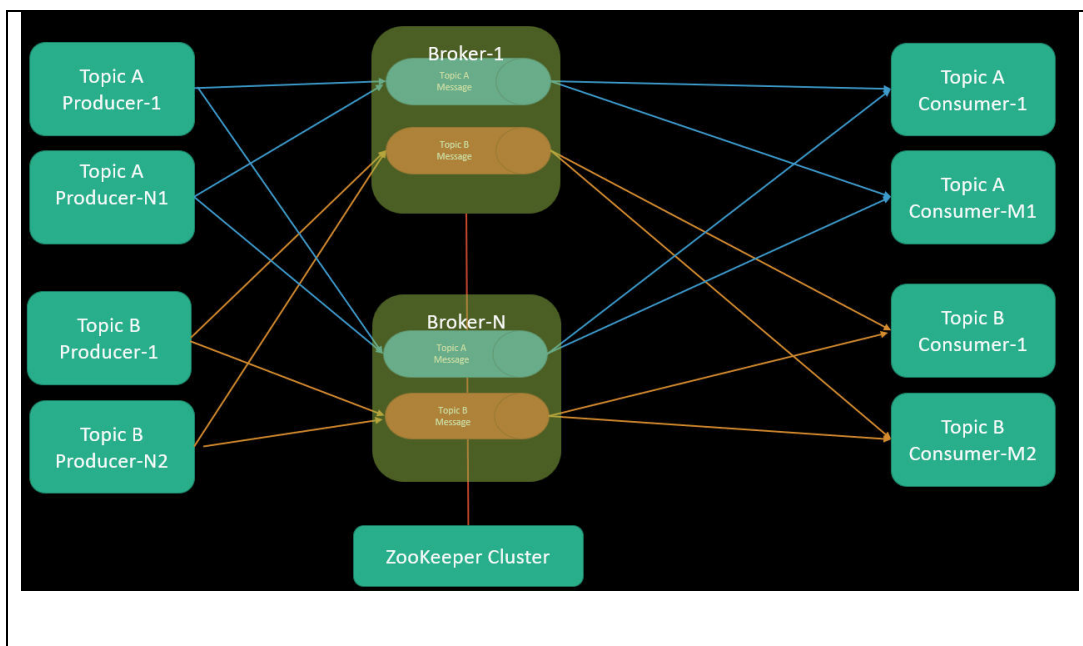
A queue of messages that have been produced by one or more producers and read by one or more consumers is what we mean when we talk about a subject. The name of a subject is what identifies it. This name is a component of a namespace that is used globally by that Kafka cluster.

Only applicable to Kafka: Subscribers are referred to as consumers, while publishers are referred to as producers. When a producer or consumer connects their device to the publish-subscribe system, they are granted the ability to read from or contribute to a particular subject.

Brokers

Kafka is a distributed system that embodies the fundamental elements of the publish-subscribe architecture defined earlier as the ideal.

Each host that makes up the Kafka cluster is responsible for running a server that is known as a broker. This broker stores messages that have been submitted to topics and fulfils consumer requests.



Kafka was developed to operate on numerous hosts simultaneously, with a separate broker running on each server. If one of the hosts stops working for whatever reason, Kafka will do all in its power to keep the others operational. This helps to achieve some of the criteria for the ideal publish-subscribe system, including "No Downtime" and "Unlimited Scaling."

All of the Kafka brokers communicate with Zookeeper in order to provide distributed coordination, which provides further support for the "Unlimited Scaling" requirement of the ideal system.

The same topics are discussed by many brokers. In order to achieve the objectives of "No Downtime," "Unlimited Scaling," and "Message Retention," replication is an essential component.

There is a single broker who is in charge of organising the activities of the cluster. This particular broker is referred to as the controller. As was discussed before, the optimal behaviour of a topic is that of a queue of messages. In point of fact, having just a single queue causes scale problems. The implementation of partitions in Kafka is what contributes to the topics' resilience.

Records

A record is the term used in Kafka to refer to a publish-subscribe message. A record is made up of a key-value pair as well as other information, which may include a timestamp. The key is optional; however, it may be used to distinguish messages coming from the same data source. Arrays of bytes are used for the storage of keys and values in Kafka. Aside from that, it does not care about the format at all.

Headers are something that may be included in the metadata of each record. Key-value pairs may be used to contain application-specific information when using headers.

In the context of the header, the keys are considered to be strings, while the values are considered to be byte arrays.

Partitions

Kafka organises the data it manages into partitions, rather than storing them all in a single log as would be the case with other systems. One way to think of partitions is as a subset of all the records that pertain to a subject. The concept of "Unlimited Scaling" may be helped along by partitions. Records that are included inside the same partition are organised according to their time of entry.

When a subject is first formed, it is given two attributes to customise it with:

partition count

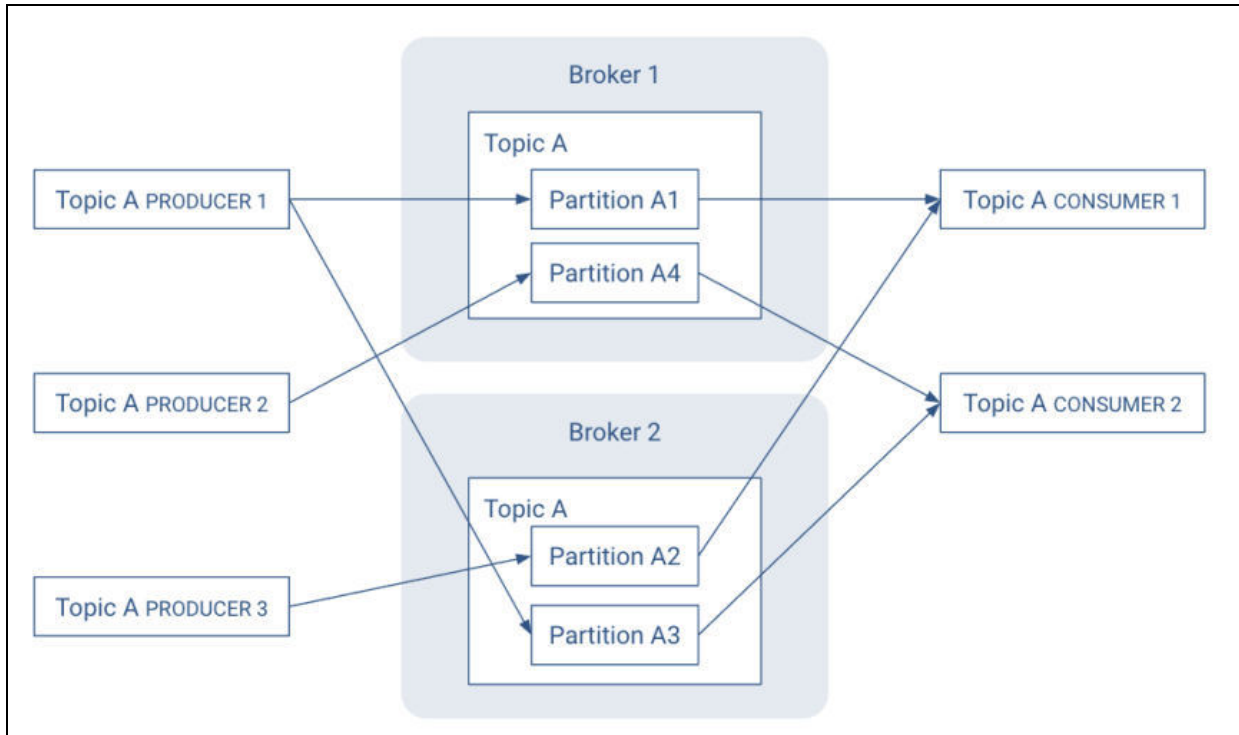
The number of partitions that records for this topic will be spread among.

Replication factor: The number of copies of a partition that are maintained to ensure consumers always have access to the queue of records for a given topic. Each topic has one leader partition. If the replication factor is greater than one, there will be additional follower partitions. (For the replication factor = M , there will be $M-1$ follower partitions.)

Any Kafka client (a producer or consumer) communicates only with the leader partition for data. All other partitions exist for redundancy and failover. Follower partitions are responsible for copying new records from their leader partitions. Ideally, the follower partitions have an exact copy of the contents of the leader. Such partitions are called in-sync replicas (ISR). With N brokers and topic replication factor M , then

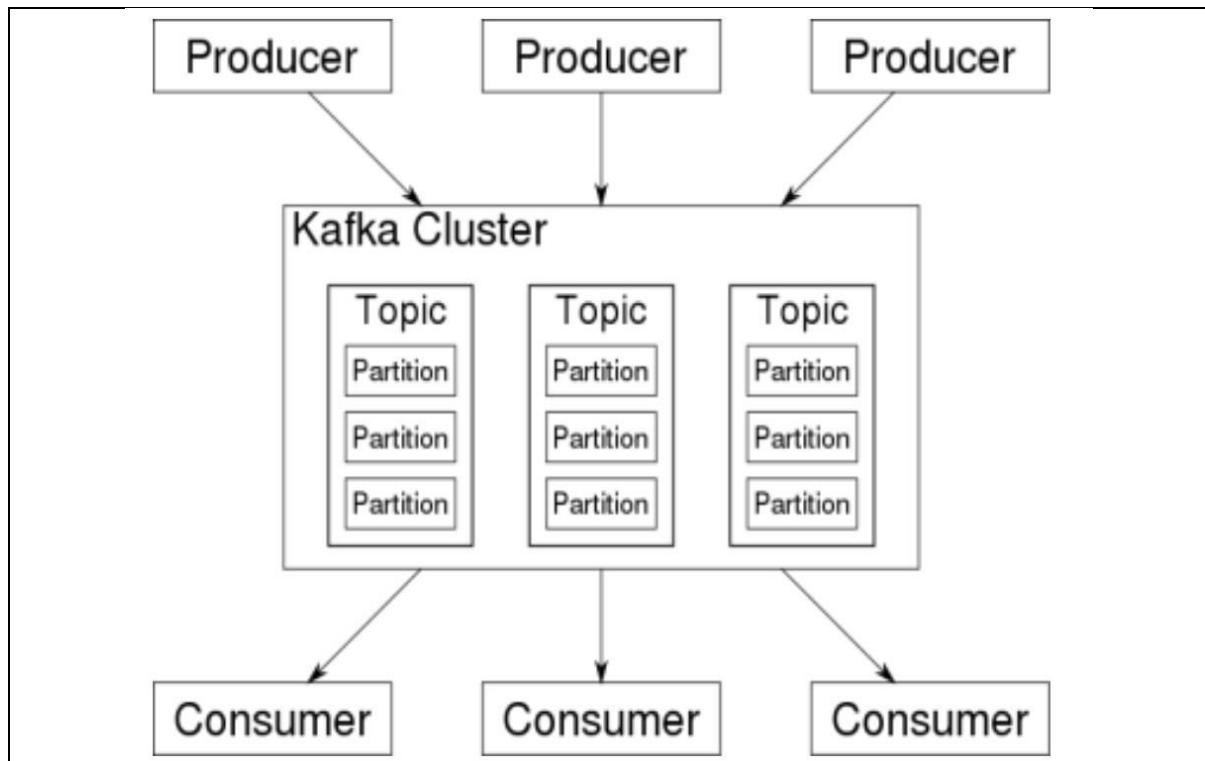
- If $M < N$, each broker will have a subset of all the partitions
- If $M = N$, each broker will have a complete copy of the partitions

In the following illustration, there are $N = 2$ brokers and $M = 2$ replication factor. Each producer may generate records that are assigned across multiple partitions.



When it comes to maintaining accurate record throughput, partitions are essential. Choosing the appropriate number of partitions and replications for a topic does two things: it ensures that the leader partitions are distributed uniformly throughout the brokers in the cluster, and it ensures that partitions belonging to the same topic are about the same size. Helps to distribute the workload among the brokers.

In reality, Kafka is a store that is responsible for storing messages that originate from processes, also known as producers. After that, the data or messages are partitioned into a number of distinct partitions inside the respective Topics. The messages are indexed and saved in this Topic's partition in addition to a timestamp for each message. On the other hand, other processes that are referred to as Consumers are able to query messages from these partitions. Kafka, which is working between these producers and consumers, operates on a cluster consisting of one or more servers, and the partitions may be dispersed among different nodes in the cluster. When Apache Storm, Apache HBase, and Apache Spark are used in conjunction with Apache Kafka, the real-time streaming data is processed in an effective and efficient manner. The fundamental structure of Kafka is seen in Figure.



As a result of Kafka being deployed as a cluster on many servers, the complete publish and subscribe messaging system is managed by Kafka with the assistance of four application programming interfaces (APIs), namely the producer API, consumer API, streams API, and connector API. Because of its capacity to provide fault-tolerant delivery of enormous message streams, it has begun to replace some of the more traditional messaging systems, such as JMS, AMQP, and others. Topics, records, and brokers are three of the most important concepts in Kafka's architectural scheme. The stream of records that make up a topic each contain a unique set of information. On the other hand, the responsibility of reproducing the communications falls on the Brokers.

KAFKA REAL TIME APPLICATIONS

Messaging: Kafka works glowing as a substitute message broker which is used for a variety of reasons. Kafka has well throughput and built-in partitioning with replication, and fault tolerance hich makes it a good solution for large scale message processing applications. In our experience messaging uses are often comparatively low-throughput, but may require low end-to-end latency and often depend on the strong durability guarantees Kafka provides.

Website Activity Tracking: The original application of Kafka was to be able to rebuild a user activity tracking pipeline as a set of real-time publish-subscribe feeds means site activity like page views, searches, or other actions users may take is published to central topics with one topic per activity type. These feeds are available for subscription for a range of use cases including real-time processing, real-time monitoring, and loading into Hadoop or offline data warehousing systems for offline processing and reporting. Activity tracking is often very high volume as many activity messages are generated for each user page view.

Metrics: Kafka is often used for operational monitoring data. This indulges aggregating statistics from distributed applications to produce centralized feeds of operational data.

Log Aggregation: Kafka is also used as an alternative to log aggregation solution. Log aggregation Combines physical log files for servers and places them in a central processing location. Kafka extracts file details and provides clearer extraction of log or event data as a as a stream of messages. This allows processing of low latency response time and easier support for multiple data sources and distributed data consumption. Compared with to log-centric systems such as Scribe or Flume, Kafka offers equally good performance, stronger durability guarantees due to replication, and much lower end-to-end latency.

Stream Processing: Kafka users are using Kafka to process data in processing pipelines of multiple stages, and raw input data are put into use in Kafka and then added, enriched or converted into new themes for subsequent consumption or tracking. For example, in order to use news articles, a workstation can scan the content of the article in its RSS content on "articles"; Additional processing can normalize or reduce this content and publish the content of the pure article to a new topic; the last run may try to present this content to users. These processing pipelines create real-time data streams based on individual themes. According to 0.10.0.0, Apache Kafka has a light but powerful streaming library called Kafka Stream to perform data processing as described above. Apart from Kafka Streams, alternative tools for the development of open-source script include Apache Storm and Apache Samza.



Contents

Chapter-10: Apache NiFi	1
Overview	1
Benefits of NiFi DataFlow.....	1
NiFi Architecture	2
NiFi as Cluster.....	3
NiFi Features	3
The fundamental ideas of NiFi	6

Chapter-10: Apache NiFi

Overview

To put it another way, NiFi was developed to automate the transfer of data from one system to another. Although the word "dataflow" is used in a number of different settings, for the sake of this discussion, we will use it to refer to the controlled and automatic flow of information between systems. This issue area has been there since since businesses started using more than one system, with some of those systems producing data and some of those systems using data for their operations. Extensive discussion and articulation have taken place with regard to the challenges and solution patterns that have arisen. The Enterprise Integration Patterns document has a format that is both comprehensive and easy to consume.

Through the years, dataflow has become one of those undesirable aspects of an architecture that are required. However, there are a number of active and quickly developing initiatives that are making dataflow a whole deal more fascinating and a great deal more crucial to the success of any given business. Among them are topics like service-oriented architecture, the proliferation of application programming interfaces (APIs), the internet of things, and big data. In addition, the amount of stringency that is required to ensure compliance, privacy, and security is always increasing.

Even with all of these new ideas being developed, the patterns and requirements of dataflow have remained essentially same for the most part.

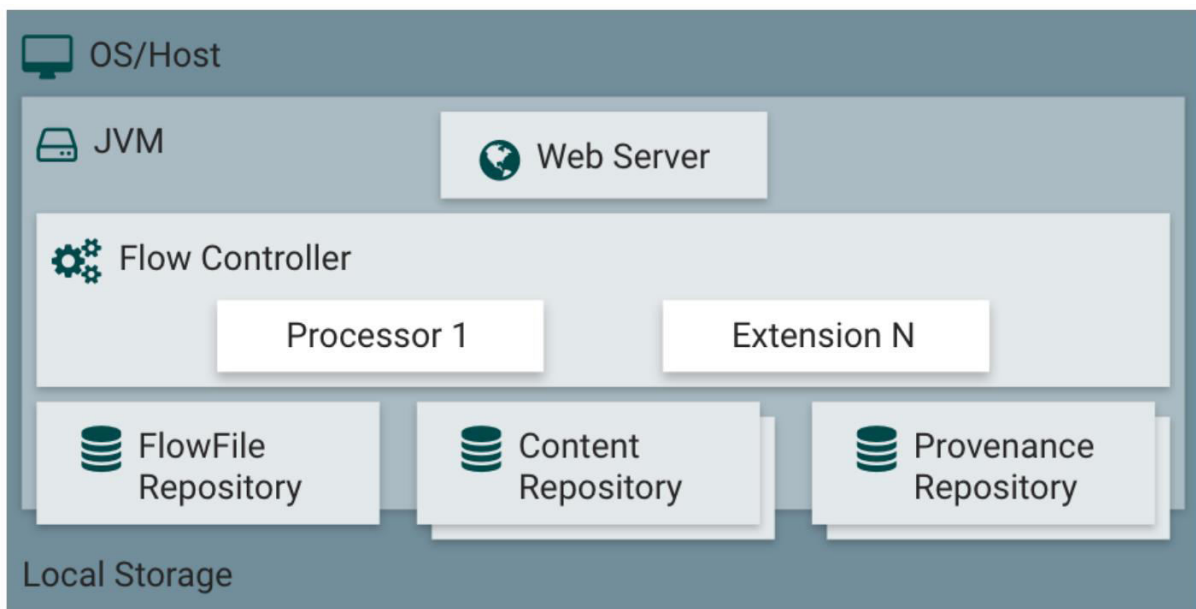
The key distinctions then are the magnitude of the complexity, the velocity of change that must occur in order to adapt, and the fact that, at scale, the exceptional instance becomes the norm. NiFi was developed specifically to assist in overcoming the difficulties associated with current dataflow.

Benefits of NiFi DataFlow

The use of this design paradigm results in a number of advantageous outcomes, which contribute to NiFi's status as a highly efficient platform for the construction of robust and scalable dataflows. A few examples of these advantages are as follows:

- Is inherently asynchronous, which allows for very high throughput and natural buffering even as processing and flow rates fluctuate
- Provides a highly concurrent model without a developer having to worry about the typical complexities of concurrency
- Promotes the development of cohesive and loosely coupled components, which can then be reused in other contexts and promotes testable unidirectional dependencies
- Lends itself well to the visual creation and management of directed graphs of processors
- Error management becomes as natural as the happy route rather than a coarse-grained catch-all due to the resource restricted connections.
- Critical functions such as back-pressure and pressure release become highly natural and intuitive.
- It is possible to quickly understand and monitor the flow of data across the system, as well as the points of entry and departure for the data itself.

NiFi Architecture



NiFi is run on a host operating system inside a Java Virtual Machine (JVM). The following is a list of the key components of NiFi when run on the JVM:

Web Server: The HTTP-based command and control API for NiFi is what the web server will be used to host as its primary function.

Controller of the Flow: The flow controller is the component that acts as the operation's central processing unit. It handles the schedule of when extensions obtain resources to execute and offers threads for them to run on. Extensions may run on these threads.

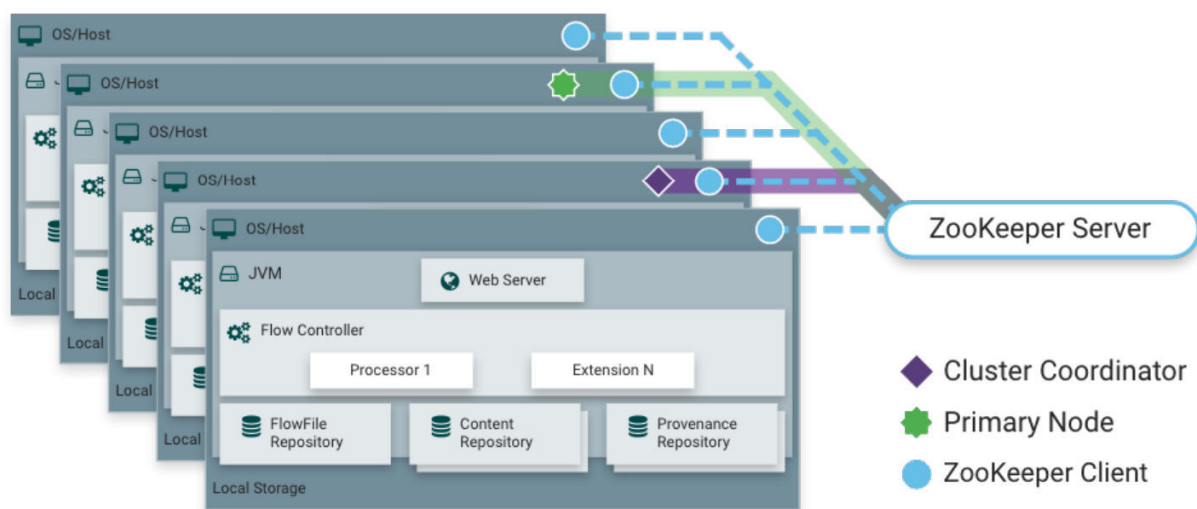
Extensions: Other publications detail the different sorts of NiFi extensions, which may be found in their respective dictionaries. The most important thing to understand here is that extensions work and run inside of the JVM.

FlowFile's File Storage Repository: NiFi maintains a record of the current status of what it knows about a specific FlowFile that is currently being used in the flow in a location known as the FlowFile Repository. It is possible to plug different implementations into the repository. The permanent Write-Ahead Log that is stored on a particular disc partition is the method that is used by default.

Content Repository: The actual content bytes of a specific FlowFile are stored in the Content Repository, which is referred to simply as the Content Repository. It is possible to plug different implementations into the repository. The method that is used by default is a rather straightforward process that saves data in chunks inside the file system. You have the option of specifying more than one file system storage location in order to have various physical partitions activated and so lessen the amount of congestion on any one volume.

Provenance Repository: The Provenance Repository is where all of the data on the provenance of events is housed. The repository structure is pluggable, and the default implementation uses one or more actual storage volumes. The repository may be expanded using additional plug-ins. Event information is indexed and searchable inside each place it is stored in.

NiFi as Cluster



NiFi 1.0 uses Zero-Leader Clustering. Each NiFi node conducts the same functions on various data sets. Apache ZooKeeper automatically selects a Cluster Coordinator and handles failover. The Cluster Coordinator receives heartbeat and status updates from all nodes. Cluster Coordinator reconnects nodes. ZooKeeper elects a Primary Node for each cluster. DataFlow managers may interact with the NiFi cluster using any node's UI. Changes are copied to all cluster nodes, giving numerous access points.

NiFi Features

- **Guaranteed Delivery (Guaranteed Delivery)**

NiFi was founded on the principle that even when operating at a very large scale, assured delivery remains an absolute need.

This is made possible by the efficient use of a purpose-built permanent write-ahead log as well as a content repository. Together, they have been developed in such a manner as to make it possible for extremely high transaction rates, efficient load-spreading, copy-on-write, and to play to the strengths of conventional disc read/write operations.

- **Buffering of Data with Back Pressure and Pressure Release**

NiFi allows for the buffering of any and all data that is queued, as well as the ability to apply back pressure to queues when they reach a certain limit or to "age off" data when it hits a certain age threshold (its value has perished).

- **Queuing Based on Priority**

In order to retrieve data from a queue in the most efficient manner possible, NiFi permits the setup of one or more prioritising algorithms.

The most recent data should be retrieved first by default; however, there are situations in which the data should be fetched in a different order, such as the biggest first or the oldest first.

- **Flow-Specific Quality of Service (latency v throughput, loss tolerance, etc.)**

There are some places along the path of a dataflow when the data is vitally essential and cannot tolerate any loss. There are also situations in which it must be processed and transmitted within a very short amount of time in order for it to be of any use. NiFi makes it possible to configure these concerns in a manner that is fine-grained and flow-specific.

Ease of Operation

- **Command and Control in the Visual Domain**

Dataflows have the potential to become pretty complicated. Having the ability to view those movements and communicate them graphically may be a huge assistance in reducing that complexity and determining which areas need to be reduced. NiFi not only allows the construction of dataflows in a visible manner, but it also does it in real time. It is more comparable to working with clay than it is to being able to "plan and deploy." If you make a modification to the dataflow, that it will take effect as soon as you save it. The changes are quite subtle and are confined to the specific components that were impacted.

It is not necessary to halt a complete flow or collection of flows in order to make a particular alteration.

- **Flow Scheduling Templates**

Dataflows have a tendency to be very pattern-oriented, and while there are often many alternative methods to address an issue, it is really helpful to be able to communicate the best practises that have been developed. The use of templates enables subject matter experts to construct and publish their flow designs, while also allowing others to benefit from and participate on these designs.

- **Data Provenance**

As objects move through the system, NiFi automatically records, indexes, and makes accessible the provenance data for each item. This applies to fan-in, fan-out, transformations, and other operations as well. When used to support compliance requirements, troubleshooting, optimization, and a variety of other situations, this information takes on an incredibly vital role.

- **Recording a rolling buffer of fine-grained history as part of the recovery process**

The content repository that NiFi uses is intended to perform the function of a rolling buffer of history. Data is never deleted until it has become obsolete in the content repository or more storage space is required. This, in conjunction with the data provenance capabilities, allows for an

exceptionally valuable base to enable click-to-content, download of content, and replay, all at a particular moment in the lifespan of an object, which may even span generations.

Protection

- **System to Protection System**

When it comes to dataflow, security is the single most important factor. NiFi provides a safe exchange at every point in a dataflow by making use of protocols that encrypt data, such as 2-way secure socket layer (2-way SSL). In addition, NiFi makes it possible for the flow to encrypt and decrypt material, as well as make use of shared keys or other techniques on either the sender or the receiver side of the equation.

- **User to Operating System**

NiFi supports two-way SSL authentication and offers pluggable authorization so that it may effectively regulate a user's access at a variety of different levels (read-only, dataflow manager, admin). If a user inputs a sensitive property like a password into the flow, it is instantly encrypted on the server side, and it is never again accessible on the client side, not even in its encrypted version. An example of such a property would be a credit card number.

- **Authorization for Multiple Tenants**

Each component is subject to the authority level of a specific dataflow, which enables an administrative user to exercise granular control over the access granted to other components. This indicates that each NiFi cluster has the capacity to meet the needs of one or more enterprises simultaneously. When compared to isolated topologies, multi-tenant authorization makes it possible to implement a self-service model for the management of dataflow. This model gives each group or organisation the ability to manage flows while maintaining full awareness of the portions of the flow to which they do not have access.

Extensible Architecture

- **Extension**

NiFi was designed from the ground up to be extensible, and as such it functions as a platform on which dataflow operations may be carried out and interact with one another in a way that is both dependable and consistent. Processors, Controller Services, Reporting Tasks, Prioritizers, and Customer User Interfaces are all examples of points of extension.

- **Isolation of the Classloader**

Dependency issues may arise very fast in any system that is built using components. This problem is addressed by NiFi, which provides a special class loader mechanism. This approach ensures that each extension bundle is only accessible to a very restricted range of dependencies, which solves the problem. As a consequence of this, extensions may be developed with very little consideration given to the possibility that they would be incompatible with another extension. The idea behind these extension packages is referred to as "NiFi Archives," and it is covered in the Developer's Guide in further detail.

- **Site-to-Site Communication Protocol**

The NiFi Site-to-Site (S2S) Protocol is the recommended method of communication for using between different instances of NiFi. The use of S2S makes it simple to move data from one instance

of NiFi to another in a way that is quick, effective, and safe. It is simple to build NiFi client libraries and incorporate them into other programmes or devices so that they may connect with NiFi via the S2S protocol. It is feasible to include a proxy server into the S2S communication since the socket-based protocol as well as the HTTP(S) protocol are supported in S2S in their capacity as the underlying transport protocol.

Flexible Scaling Model

- **Scaling-out of the (Clustering)**

As was discussed before, NiFi was developed to be able to scale out by using the clustering together of a large number of nodes.

If the provisioning and configuration of a single node is capable of handling hundreds of megabytes per second, then it should be possible to design even a small cluster to handle gigabytes per second. This therefore presents some intriguing issues in the form of load balancing and fail-over between NiFi and the systems from which it obtains its data. It may be beneficial to make use of asynchronous queuing-based protocols such as message services, Kafka, and so forth. Use of NiFi's 'site-to-site' feature is also very effective because it is a protocol that enables NiFi and a client (including another NiFi cluster) to talk to each other, share information about loading, and exchange data on specific authorised ports. This is achieved through the use of NiFi's 'site-to-site' feature, which can be found in the NiFi UI.

- **Scale up or down**

NiFi may also be scaled up or down in a fairly flexible manner due to the way it was created. When setting NiFi, under the Scheduling tab, it is possible to increase the number of concurrent jobs running on the processor, which, from the perspective of the NiFi framework, will result in a higher throughput. This makes it possible for more processes to run at the same time, which results in increased throughput. On the other end of the spectrum, it is possible to scale NiFi down to the point where it is acceptable for running on edge devices. This is useful in situations where a minimal footprint is desirable owing to restricted hardware resources.

The fundamental ideas of NiFi

The core ideas behind NiFi's architecture have a lot in common with the underlying principles behind flow-based programming. The following is a list of some of the most important topics related to NiFi:

Flow File: Each item that is processed by the system is denoted by a FlowFile, and NiFi maintains a map of key/value pair attribute strings and the accompanying content of zero or more bytes for each FlowFile.

FlowFile Processor: Processors are the components that are responsible for carrying out the task. To put it another way, a processor is responsible for some combination of data routing, data transformation, and system-to-system mediation. The properties of a certain FlowFile and the content stream of that file may be accessed by processors. In each given unit of work, processors are able to perform operations on zero or more FlowFiles and then either commit or rollback the results of those operations.

Connections: Connections are what offer the real connectivity between different processors in a system. These serve as queues, enabling a variety of processes to engage with one another at varying speeds. These queues may have their priorities changed on the fly, and they can also have higher load boundaries, which enables back pressure to be applied.

Flow Controller: The Flow Controller is responsible for retaining the information of how processes relate to one another as well as managing the threads and allocations of those threads that are used by all processes. The Flow Controller performs the role of a broker and makes it possible for processors to trade FlowFiles with one another.

Process Group: Process Group: A Process Group is a particular group of processes and their connections that may accept data via the use of input ports and send data out using output ports. A Process Group is sometimes referred to as a process cluster. In this way, process groups make it possible to generate completely new components only by composing existing components.

The use of this design paradigm results in a number of advantageous outcomes, which contribute to NiFi's status as a highly efficient platform for the construction of robust and scalable dataflows. A few examples of these advantages are as follows:

- Is inherently asynchronous, which allows for very high throughput and natural buffering even as processing and flow rates fluctuate
- Provides a highly concurrent model without a developer having to worry about the typical complexities of concurrency
- Promotes the development of cohesive and loosely coupled components, which can then be reused in other contexts and promotes testable unidirectional dependencies
- Lends itself well to the visual creation and management of directed graphs of processors
- Error management becomes as natural as the happy route rather than a coarse-grained catch-all due to the resource restricted connections.
- Critical functions such as back-pressure and pressure release become highly natural and intuitive.
- It is possible to quickly understand and monitor the flow of data across the system, as well as the points of entry and departure for the data itself.



Contents

Chapter-13: Apache Phoenix	1
Overview	1
Apache Phoenix History	1
Apache Phoenix & SQL Support	1
Apache Phoenix JDBC Connection	1
About Apache Phoenix.....	2
Relational Layer.....	2
Apache Phoenix Integration with Hadoop.....	3
Apache HBase	3
Phoenix and SQL	3
Phoenix not supported SQL Construct.....	3
Phoenix Knobs and Dials	4
Cloudera Operational Database.....	4
Apache Phoenix Use Cases	4

Chapter-13: Apache Phoenix

Overview

Apache Phoenix is an open source, massively parallel, relational database engine supporting OLTP for Hadoop using Apache HBase as its backing store. Phoenix provides a JDBC driver that hides the intricacies of the NoSQL store enabling users to create, delete, and alter SQL tables, views, indexes, and sequences; insert and delete rows singly and in bulk; and query data through SQL. Phoenix compiles queries and other statements into native NoSQL store APIs rather than using MapReduce enabling the building of low latency applications on top of NoSQL stores. Apache Phoenix OLTP and operational analytics for Apache Hadoop. Apache Phoenix enables OLTP and operational analytics in Hadoop for low latency applications by combining the best of both worlds:

1. The power of standard SQL and JDBC APIs with full ACID transaction capabilities.
2. And the flexibility of late-bound, schema-on-read capabilities from the NoSQL world by leveraging HBase as its backing store

Apache Phoenix is fully integrated with other Hadoop products such as Spark, Hive, Pig, Flume, and Map Reduce.

Apache Phoenix History

Phoenix began as an internal project by the company salesforce.com out of a need to support a higher level, well understood, SQL language. It was originally open-sourced on GitHub on 28 Jan 2014 and became a top-level Apache project on 22 May 2014. Apache Phoenix is included in the

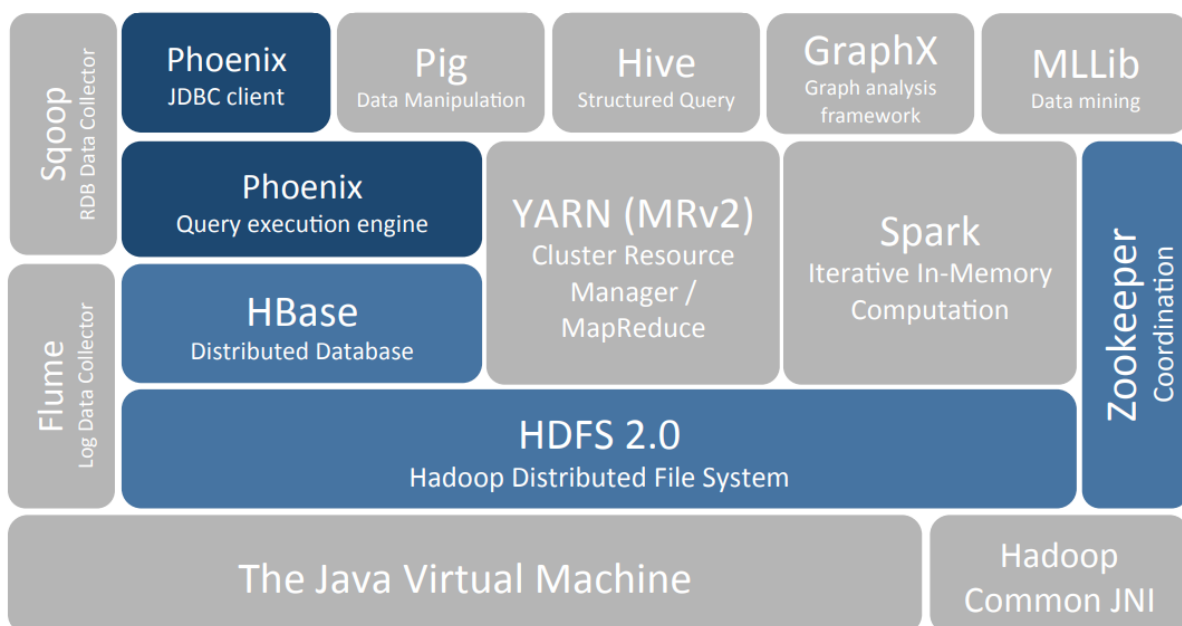
- The Power of standard SQL and JDBC APIs with full ACID transaction capabilities.
- The flexibility of late-bound, schema-on-read capabilities from the NoSQL world by leveraging HBase as its backing store.
- Apache Phoenix has Embedded JDBC Driver which implements the majority of java.sql interfaces, including metadata API's.
- Apache Phoenix allows columns to be modelled as a multi-part row key or key/value cells.
- Full query support with predicate push down and optimal scan key formation.
- DDL support: CREATE TABLE, DROP TABLE, and ALTER TABLE for adding/removing columns.
- Versioned schema repository. Snapshot queries use the schema that was in place when data was written.
- DML support: UPSERT VALUES for row-by-row insertion, UPSERT SELECT for mass data transfer between the same or different tables, and DELETE for deleting rows.

Relational Layer

- Apache Phoenix is a relational layer for Apache HBase
- **Query Engine:**
 - Transform SQL Queries and parses into native HBase API calls. This is in-directly MapReduce.
 - Apache Phoenix pushes as much work as possible onto the cluster for parallel execution.
 - **Metadata Repository:** This is a Phoenix table itself which helps in typed access to data stored in HBase tables. It stores tables, views, sequence definitions, secondary indexes. For your perspective it's a JDBC Driver.

Apache Phoenix Integration with Hadoop

Apache Phoenix is fully integrated with other Hadoop products such as Spark, Hive, Pig Flume and MapReduce. For your perspective Apache Phoenix is just like a JDBC driver.



Apache HBase

Apache HBase is a high performance horizontally scalable datastore engine for BigData, suitable as the store of record for mission critical data.

Phoenix and SQL

- Accessing HBase data with Phoenix can be substantially faster than direct HBase API use.
- Phoenix parallelizes queries based on stats. HBase does not know how to chunk queries beyond scanning an entire region.
- Phoenix pushes processing to the server.
- If you write your own API call, this may not use coprocessors.
- Phoenix has a huge difference for aggregations vs direct HBase API calls.
- Phoenix supports and uses secondary indexes.

Apache Phoenix takes your SQL Query, compiles into a series of HBase scans, and orchestrate the running of those scans to produce regular JDBC result sets. Direct use of the HBase API, along with coprocessors and custom filters, results in performance on the order of milliseconds for small queries, or seconds for tens of millions of rows.

All standard SQL constructs are supported, including SELECT, FROM, WHERE, GROUP BY, HAVING, ORDER BY etc.

Apache Phoenix also supports a full set of DML commands as well as table creation and versioned incremental alterations through our DDL commands.

Phoenix not supported SQL Construct

Below is the list of constructs which are currently not supported.

- Relational Operators: Intersect, Minus
- Miscellaneous Built-In functions:

Phoenix Knobs and Dials

Phoenix provides many different knobs and dials to configure and tune the system to run more optimally on your cluster. The configuration is done through a series of Phoenix-specific properties specified for the most part in your client-side hbase-site.xml file. In addition to these properties, there are of course all the HBase configuration properties.

Cloudera Operational Database

Cloudera Operational Datastore is a real-time auto-scaling operational database powered by Apache HBase and Apache Phoenix. COD is an experience which runs in CDP. Cloudera Operational Database experience allows self-service creation and management of an operational database. You can provision a new database with a single click, build application against it and deploy it on the public cloud without complexity.

Apache Phoenix Use Cases

1. We can use Apache Phoenix for storing data as a basis for measuring activities and generating reports. You should choose Phoenix because it provides the scalability of HBase and the expressiveness of SQL.
2. Phoenix can be used for on Demand Data aggregations. If you have floating time range of



Contents

Chapter-12: Apache Kudu for Cloudera CDP-0011 Certification	1
Overview	1
Comparison with other storage engines.....	2
Kudu's Design and its benefits	2
HDFS vs Kudu	2
Kudu Example Use Cases	3
Kudu tables, schemas.....	3
Kudu and Write Operations	4
Kudu and Read Operations	4
Kudu and API.....	4
Kudu and Consistency Model.....	4
Kudu and Timestamps.....	5

Chapter-12: Apache Kudu for Cloudera CDP-0011 Certification

Overview

- Kudu is an open-source structured data storage engine with low-latency random and analytical access.
- Kudu uses horizontal partitioning and Raft consensus to provide reduced mean-time-to-recovery and tail latencies.
- Kudu is part of the Hadoop ecosystem and supports Cloudera Impala, Apache Spark, and MapReduce.
- Static data sets are usually stored on HDFS using Apache Avro or Apache Parquet for structured storage.
- HDFS and these formats don't allow changing individual records or random access.
- Semi structured stores like Apache HBase or Cassandra store mutable data sets. These systems provide low-latency record-level reads and writes but trail static file formats in sequential read speed for SQL-based analytics or machine learning.
- The gap between static HDFS data sets' analytic performance and HBase and Cassandra's low-latency row-level random access requires sophisticated structures when both access patterns are needed in a single application.
- Many organisations implemented data pipelines with streaming input and changes in HBase, followed by periodic operations to export tables to Parquet for subsequent analysis. Such architectures have drawbacks:
 - o Application architects must create complicated code to control data flow and synchronisation.
 - o Operators must maintain numerous backups, security controls, and monitoring.
 - o The resultant architecture may have a large latency between fresh data entering the HBase "staging area" and being accessible for analyses.

- Real-world systems must accommodate late-arriving data, revisions on old records, or privacy-related removals on transferred data. Rewriting, shifting partitions, and manual intervention may be needed.
- Kudu is a novel storage system aimed to bridge the gap between sequential-access systems like HDFS and random-access systems like HBase or Cassandra. Kudu provides a "middle ground" solution that simplifies the design of many typical workloads. Kudu provides a straightforward API for row-level inserts, updates, and deletes while offering table scans at Parquet-like throughputs.
- Apache Hadoop comes with its own storage layer which is called HDFS (Hadoop Distributed File System).
- The Hadoop distributed file system (HDFS) is a distributed, scalable, and portable file system written in Java for the Hadoop framework.
- Some consider it to instead be a data store due to its lack of POSIX compliance, but it does provide shell commands and Java application programming interface (API) methods that are similar to other file systems.
- A Hadoop instance is divided into HDFS and MapReduce. HDFS is used for storing the data and MapReduce is used for processing data.
- Based on your data needs you can choose Apache Kudu instead of HDFS.
- Because Kudu is a good solution for the time-series data which are good use case for collecting data from IoT sensors, works well with Spark.
- Apache Kudu column-oriented data store of the Apache Hadoop ecosystem.
- It provides completeness to Hadoop's storage layer to enable fast analytics on fast data.
- Apache Kudu began as internal project at Cloudera.

HadoopExam.com

Comparison with other storage engines

- Kudu was designed and optimized for OLAP workloads.
- Like HBase, it is a real-time store that supports key-indexed record lookup and mutation.
- Kudu differs from HBase since Kudu's Datamodel is a more traditional relational model, while HBase is schemaless.
- Kudu's "on-disk representation is truly columnar and follows an entirely different storage design than HBase/Bigtable".

Kudu's Design and its benefits

- Fast processing of OLAP workloads.
- Strong but flexible consistency model, allowing you to choose consistency requirements on a per-request basis, including the option for strict-serializable consistency.
- Structured data model.
- Strong performance for running sequential and random workloads simultaneously.
- Tight integration with Apache Impala, making it a good, mutable alternative to using HDFS with Apache Parquet.
- Integration with Apache NiFi and Apache Spark.
- Integration with Hive Metastore (HMS) and Apache Ranger to provide fine-grain authorization and access control.
- Authenticated and encrypted RPC communication.

- Automatic fault detection and self-healing: to keep data highly available, the system detects failed tablet replicas and re-replicates data from available ones, so failed replicas are automatically replaced when enough Tablet Servers are available in the cluster.
- Location awareness (a.k.a. rack awareness) to keep the system available in case of correlated failures and allowing Kudu clusters to span over multiple availability zones.
- Logical backup (full and incremental) and restore.
- Multi-row transactions (only for INSERT/INSERT_IGNORE operations as of Kudu 1.15 release).

HDFS vs Kudu

- By combining all of above properties, Kudu targets support for families of applications that are difficult or impossible to implement using Hadoop storage technologies, while it is compatible with most of the data processing frameworks in the Hadoop ecosystem.
- A few examples of applications for which Kudu is a great solution are:
 - o Reporting applications where newly-arrived data needs to be immediately available for end users.
 - o Time-series applications that must simultaneously support:
 - queries across large amounts of historic data
 - granular queries about an individual entity that must return very quickly
- Applications that use predictive models to make real-time decisions with periodic refreshes of the predictive model based on all historic data

Kudu Example Use Cases

Use Case-1: Streaming Input with Near Real Time Availability

New data arrives rapidly and regularly, yet the same data must be available in practically real time for readings, scans, and updates. Kudu allows real-time analytics on a single storage layer by combining fast inserts, updates, and columnar scans.

Use Case-2: Time-series application with widely varying access patterns

A time-series schema arranges and indexes data points by time. This allows for investigating measurement performance over time or forecasting future behaviour based on past data. Time-series customer data may be used to store click-stream history, predict future purchases, and for customer service. These data may be used elsewhere. During analysis, insertions and mutations may occur individually or in bulk and become quickly available to read data. Kudu can manage all of these access patterns simultaneously and efficiently.

Kudu handles time-series workloads well for several reasons. With Kudu's hash-based partitioning and native support for compound row keys, it's easy to build up a distributed table without the danger of "hotspotting," which is common with range partitioning. In contrast, range partitioning poses a risk. Many time-series applications only read a few columns, not the whole row, hence Kudu's columnar storage engine is extremely useful in this context.

Previously, you may have needed many data stores to handle data access patterns. This strategy makes your software and its procedures more sophisticated and duplicates your data, doubling the required storage space. Kudu manages all of these access patterns natively, removing the need to outsource.

Use Case-3: Predictive Modelling

Data scientists build predictive learning models from large data sets. As new information is obtained or the modelled situation changes, the model and data may need to be updated or revised. The researcher may also adjust the model to examine its impact over time. Each HDFS file must be completely rebuilt to bring a large volume of data up to date. Kudu updates are near real-time. The researcher may update the graph in seconds or minutes, rather than hours or days. Batch or incremental algorithms may be run on the data at any moment with almost real-time results.

Use Case-4: Combining Data in Kudu with Legacy Systems

Companies produce and store data from many sources and formats. Your data may be saved in Kudu, RDBMS, and HDFS. Impala can query all these sources and formats without changing old systems.

Kudu tables, schemas

- Kudu stores structured data tables for users.
- A Kudu cluster may have unlimited number of well-defined tables with finite column counts.
- The table's primary key is a subset of those columns. The primary key maintains uniqueness and is the only index by which rows can be changed or removed effectively.
- This data architecture is familiar to relational database users but different from Cassandra, MongoDB, Riak, BigTable, etc.
- The user must specify a table's schema when creating it, like with a relational database.
- Undefined columns and primary key uniqueness violations cause errors. Primary key columns cannot be omitted when using the modify table command.
- We decided to explicitly declare column types instead of utilising NoSQL-style "everything is bytes" for two reasons.
 - o Explicit types enable columnar encodings like bit-packing for integers.
 - o Explicit types offer SQL-like information to other systems, such as BI or data exploration tools.
- Kudu lacks secondary indexes and uniqueness requirements beyond the main key, unlike other relational databases.
- Kudu needs every table to have a primary key, although future versions will generate surrogate keys automatically.

Kudu and Write Operations

- User mutates table using Insert, Update, and Delete APIs.
- The user must completely define a primary key, predicate-based deletions or modifications must be handled by a higher-level access mechanism.
- Kudu supports Java, C++, and Python.
- APIs offer accurate batching and asynchronous error handling to reduce round trips while processing huge data such as data loads or large updates.
- Kudu does not support multi-row transactional APIs, each mutation runs as its own transaction, albeit being batched for speed.
- Single-row modifications are always atomic across columns.

Kudu and Read Operations

- Kudu's single table operation is Scan.
- A scan may be filtered by any number of predicates.

- We provide only column-constant value comparisons and composite primary key ranges.
- The client API and server understand these predicates to reduce disc and network data transfers.
- The user may define a scan projection in addition to predicates.
- A projection consists of retrieved columns.
- Because Kudu's on-disk storage is columnar, defining a subset may increase analytic speed.

Kudu and API

- Kudu client library exposes more than data path APIs.
- Hadoop's performance is boosted by data locality scheduling.
- Kudu offers APIs to let Spark, MapReduce, or Impala schedule data ranges.

Kudu and Consistency Model

- Kudu has two consistency modes.
 - o **Snapshot consistency is default:** A scan guarantees a snapshot without causality-violating anomalies. It ensures consistency from a single client's read-your-writes.
 - o **Kudu doesn't ensure external consistency by default:** If a client writes, then connects with another through an external mechanism (e.g. a message bus), the causal dependency between the two writes is not recorded. A third reader may only see the second write in a snapshot.
- Based on our experience supporting other systems like HBase, this is adequate for many use cases.
- Kudu has the possibility to manually propagate timestamps across clients:
 - o After a write operation, the user may ask the client library for a timestamp token. This token may be sent to another client over an external channel and then to the Kudu API, keeping the causal link between writes from both clients.

Kudu and Timestamps

- Kudu utilises timestamps internally for concurrency management
- However, the user can't set a write's timestamp.
- Cassandra and HBase consider a cell's timestamp as a first-class data model element.
- While skilled users may effectively utilise the timestamp dimension, most users find it perplexing and a cause of user mistake, particularly with back-dated insertions and removals.
- The user may define a read timestamp. This enables users to run point-in-time searches in the past and ensures that dispersed processes in a single "query" (e.g., Spark or Impala) read a consistent snapshot.

Contents

Chapter-14: Cloudera Shared Data Experience	1
SDX Overview	1
SDX and Security	1
BigData Challenges: security and governance	3
Cloudera SDX (Shared Data Experience).....	3
Data Catalog.....	8

Chapter-14: Cloudera Shared Data Experience

SDX Overview

SDX is a critical part of the CDP platform from Cloudera. SDX is an integrated set of security and governance technologies built on metadata and delivering consistent context across all analytics and public as well as private cloud. SDX reduces security risk and operational costs by delivering consistent data context across deployments.

Using SDX you can set multi-tenant data access and governance policies only once, and automatically enforced across the data lifecycle in hybrid as well as multi-clouds.

SDX and Security

SDX data context architecture ensures CDP is secured by design, unlike the approach taken by other vendors where security is an afterthought or added on top of existing infrastructure. Because of SDX, enterprise can make new data available at speed and without compromise.

SDX and Deployment

- **Corporate Identities:** with SDX you can use existing corporate identities and groups with the multi-tenant clusters.
- **Networks & encryptions:** With SDX you can configure Kerberos based authentication, and TLS wire encryption, DNS proxies for web interfaces.
- **Storage and encryption:** Using SDX, you can enable encrypted data at rest across the platform.

SDX and Data Access

- **Single Sign On:** Using SDX you can have LDAP based authentication and authorization for services and web UIs.
- **Authorization:** You can use tag-based policies for Authorization using SDX.
- **Lineage, non-repudiation & audit:** You can have audit of the data access as well as entire lineage of the data. i.e., what is the source of the data and who all are using it.

SDX Key features

- **Insightful metadata:** Trusted, reusable data assets and efficient deployments need more than just technical and structural metadata. CDP's Data Catalog provides a single pane of glass to administer and discover all data, profiled and enhanced with rich metadata that includes the operational, social and business context, and turns data into valuable information.
- **Powerful security:** Eliminate business and security risks, and ensure compliance by preventing unauthorized access to sensitive or restricted data across the platform with full auditing. SDX enables organizations to establish multi-tenant data access with ease through standardization and seamless enforcement of granular, dynamic, role- and attribute-based security policies on all clouds and data centers.
- **Full encryption:** Enjoy ultimate protection as a fundamental part of your CDP installation. Clusters are deployed and automatically configured to use Kerberos and for encrypted network traffic with Auto-TLS. Data at rest, both on-premises and in the cloud, is protected with enterprise-grade cryptography, supporting best practice tried and tested configurations
- **Hybrid Control:** Meet the ever-changing business needs to balance performance, cost and resilience. Deliver true infrastructure independence. SDX enables it all with the ability to move data, together with its context, as well as workloads between CDP deployments. Platform operational insight into aspects like workload performance deliver intelligent recommendations for optimal resource utilization.
- **Data Governance:** Prove compliance and manage the complete data lifecycle from the edge to AI and from ingestion to purge with data management across all analytics and deployments. Identify and manage sensitive data, and effectively address regulatory requirements with unified, platform-wide operations, including data classification, lineage, and modelling.

SDX and automatic Data profiling

- With data volumes of any type exploding, quickly making new data available to end users lets organizations capitalize on the flood of information. The limiting factor is the time taken to understand the data. SDX automatically profiles new data as it arrives to build insight and classify sensitive information, triggering the appropriate access policies and ensuring corporate standards and compliance are always met.

Easy, secure data access

- Innovation and digital transformation require organizations to uncover insight and value from their data at scale. The key enabler is providing all users access to all data and analytics to experiment and implement new use cases. SDX delivers consistent security and governance for multi-tenant data access across all deployments, marrying free access with complete safety and compliance.

Regulatory compliance

- Regulatory compliance (e.g. GDPR and CCPA) demands a modern data architecture that decreases business and security risks stemming from ever-changing data privacy requirements. SDX lets organizations identify and manage sensitive data for compliance without disruption to business processes whilst also providing consistent security and governance transparently across all data and deployments.

BigData Challenges: security and governance

- Sharing Data Across Workloads
 - o Requires multiple copies of data need to be created.
 - o Each with its own set of data context
- Burdensome admin effort
 - o Multiple Clusters=multiple places to administer.
- Missing permissions: One missing permission in one copy of the data can lead to significant financial and reputation risk.
- Data Sharing: Difficult to share data safely for new Analyses.
- New Regulations: New regulations makes things even more challenging.

Cloudera SDX (Shared Data Experience)



Question: What is Cloudera Shared Data Experience or SDX?

Answer: Cloudera SDX is a suite of technologies that make it possible for an enterprise to pull all of their data into one place. SDX enables you to share these data with different teams and services in a secured and governed manner.

Data Security, governance, and control policies are set once and consistently enforced everywhere, reducing operational costs and business risks while also enabling complete infrastructure choice and flexibility.

Question: Which of the services are involved within SDX technologies?

Answer: There are four discrete services within SDX technologies







- **Data Lake:** These are set of functionalities for creating safe, secured and governed data lakes that provide a protective ring around the data wherever it is stored, be that in cloud object storage or HDFS (Hadoop Distributed File System).
- **Data Catalog:** This is a service for searching, organizing, securing and governing data within enterprise data cloud. The Data Catalog service enables you to understand, manage, secure, and govern data assets across multiple clusters and CDP environments. It will help you understand how data is created, modified, secured, and protected.
- **Replication Manager:** A service for copying, migrating, snapshotting, and restoring data between environments within the enterprise data cloud. This service is used by administrators and data stewards to move, copy, backup, replicate, and restore data in or between data lakes.
- **Workload Manager:** A service for analysing and optimizing workloads within the enterprise data cloud.

Question: What all can be achieved using Shared Data Experience?

Answer: You can achieve following from the Shared Data Experience

- **Data Catalog:** Data Catalog is a comprehensive Catalog of all data sets, spanning on-premises, Cloud Object stores, structured, unstructured, and semi-structured. Which includes technical schemas from the Hive Metastore, as well as business glossary definitions, classifications and usage guidance.
- **Security:** Role Based Access Control applied consistently across the platform. This also includes full stack encryption and key management.
- **Governance:** SDX provides enterprise grade auditing, lineage and other governance capabilities applied universally across the platform with rich extensibility for partner integrations.
- **Data Lifecycle management:** Comprehensive ingest-to-purge management of data set life cycle activities.
- **Control plane:** Using this you can have multi-environment cluster provisioning, deployment, management, and troubleshooting.

EXAMPLE CLUDERA SDX USE CASES

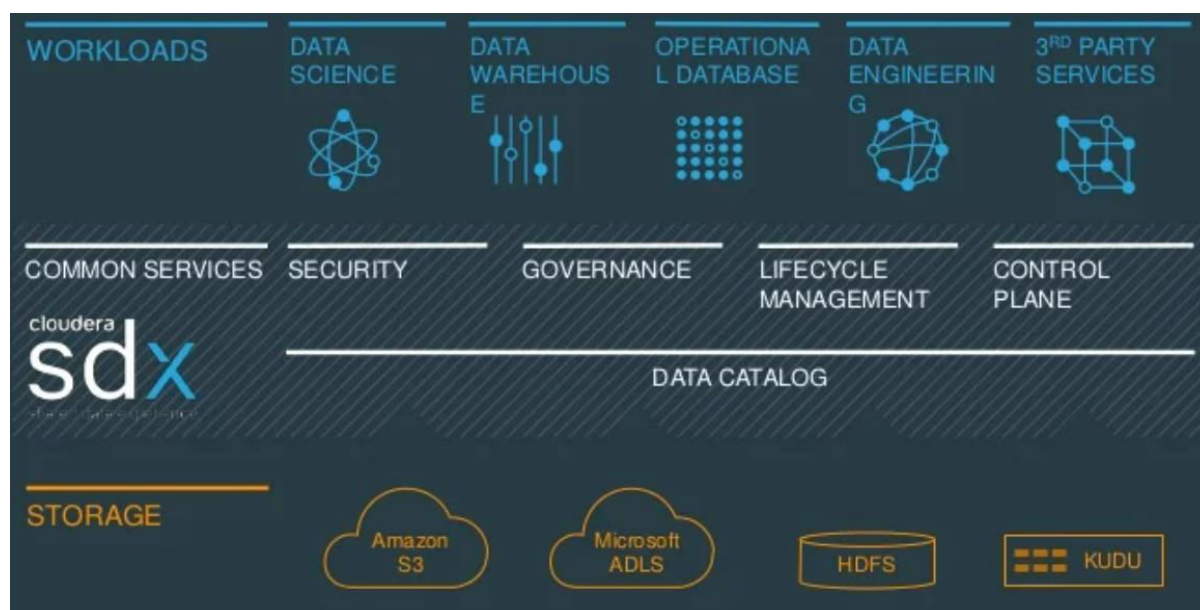
<p>Cloudera SDX makes it easy for administrators, BI users, data scientists to work together on a common data set, with consistent data context</p>	 + 	<ul style="list-style-type: none"> Run ETL with Spark, MapReduce, or any number of partner tools Assign permissions and classifications once Data, along with all data context, is immediately available in the analytics database
<p>Partner tools can use and enrich data context automatically</p>	 + 	<ul style="list-style-type: none"> Run specialized transient workloads for security profiling, data preparation, ETL, etc. Partner tools can have dedicated clusters Data, along with all data context, is immediately available to all partner tools
	 + 	<ul style="list-style-type: none"> Run ETL with Spark, MapReduce, or any number of partner tools Assign permissions and classifications once Data, along with all data context, is immediately available for data science and machine learning

Overview

- Cloudera's Shared Data experience delivers an integrated set of security and governance technologies built on metadata and delivers persistent context across all analytics as well as public and private clouds.

Benefits for IT Infra and Ops

- Central control and security
- Find value from source of truth.
- Bring the best tools for each job.



Cloudera Shared Data eXperience (SDX) Cloudera Shared Data eXperience (SDX) provides a powerful set of data and metadata shared services for security, governance and insight. Policies are set once and they are automatically enforced across data and analytics in hybrid, private and multi-public clouds across all analytics and machine learning. SDX is used to: – Achieve and maintain regulatory compliance (e.g. GDPR and CCPA) across all data and environments and lets you identify and manage sensitive data. – Ensure metadata and policies are part of every data replication or migration, regardless of its purpose, and across on-premises, private and public cloud infrastructures. – Deliver unified data management with security, governance and control policies that are set once and consistently enforced seamlessly across multiple analytic workloads running against the same diverse data sets on fluid multi and hybrid cloud infrastructures. SDX creates a trusted, business-ready foundation with consistent governance, protection, and compliance regardless of the location or analysis of the data.

A core layer of CDP Private Cloud Base is Cloudera Shared Data Experience (SDX), with uniform capabilities of Data, Schema, Replication, Security, and Governance. Cloudera SDX Shared Data Experience includes the following capabilities:

- Schema Automatic capture and storage of all schema and metadata definitions as platform workloads use and create them.
- Replication Deliver data copies and data policies that the enterprise requires to work, with complete consistency and security.
- Security Role-based access control applied consistently across the platform, including full stack encryption and key management.
- Governance Enterprise-grade auditing, lineage, and governance capabilities applied across the platform with rich extensibility for partner integrations.

Shared Data Experience (SDX)

Beyond all the integration between the three tenets of CDF, the most important element that makes CDF a true platform is Cloudera Data Platform's SDX. A powerful data fabric for complete security, governance and control across infrastructures, providing ultimate deployment choice and flexibility. Since all the components of CDF integrate tightly with SDX, you get a unified experience for security (with Apache Ranger), governance (with Apache Atlas) and data lineage from edge-to-cloud.

Cloudera's SDX is the shared data experience that provides the security, governance, lineage, management, and automation to streamline this data journey for the public sector. It's a single pane of glass that manages all of those experiences – wherever employees are. An agency can set the right governance and provide access to the right users and the right business unit. There is no other solution in the industry that can offer this management today, and it is certified to run on Red Hat OpenShift. Any existing Red Hat customer today running a containerized environment can take advantage of these experiences for a more holistic data strategy

Meeting Enterprise Operational Needs with a Modern Platform

SDX is a set of shared open platform services built for multi-functional, multi-tenant, and/or multi-disciplinary analytics that have been optimized for the cloud. This means that Cloudera Enterprise offers a unified security model that helps protect sensitive data with a consistent set of controls, and that it offers a consistent governance model that enables self-service secure access to all of your relevant data. Not just one type of data, really to all of it, increasing your ability to be compliant, particularly in a regulatory environment.

Catalog

Common pain points across analytics functions include users not being able to find relevant data, or trust what they've found, or be able to access it without IT help. Sometimes they don't know the lineage or history, sometimes the data is missing business context. Working with data in transient environments in the cloud can be particularly challenging, as this information can be lost and will need to be re-created again. A common scenario might be, "I see 10 tables called 'Customer Accounts' - which one should I use?" SDX help you identify the table that is most popular, used by all your team members, or characterized by other important attributes. The SDX shared data Catalog helps to define and preserve the structure and the business context of all your data, regardless of where it happens to reside, spanning on-premises, cloud object stores, structured, unstructured, and semi-structured data. Business Catalog services (not just a Hive metastore) span all enterprise data sets, schemas, collaborative tags, and business classifications, and are targeted for each type of user. This is enhanced by key features such as technical metadata separation, typing, and validation, and automated policybased definitions. Persisting this information, even for temporary cloud environments makes everyone's life easier.

Security and Governance

Another set of problems with traditional and alternative approaches is around security and governance. Pain points here include incomplete or inconsistent controls, which lead to significant financial and reputation risk. Trying to solve these through administrator effort alone is burdensome, and likely won't meet industry and government regulations like GDPR, PCI DSS, or HIPAA. Security should be an enabler, good security makes it easier to share, and avoid producing copies of data that are either stripped down or unsafe. You might be thinking, "when I add a new security policy, I need it to immediately take effect for all workloads because my users use a mix of Apache Impala, Hive, and Spark." Great, SDX provides that, too. Alternately, if security is too hard to configure, my users just won't use it - they'll find a way to turn it off. SDX provides automatic configuration for encryption at rest and wire encryption. You can keep key management safe in your own facility. Audit logs are complete, immutable, and preserved, unlike a cloud provider that does them at 5 minute intervals and discards after two weeks, or a Hadoop distribution that allows them to be disabled. Cludera Enterprise has a full complement of security features for compliance including encryption at rest and in motion, authorization by role, audit logs as noted, visibility to classify data by sensitivity, and full record updates or erasure upon request.

Lifecycle

Lifecycle management, often alongside partners, increases user productivity and boosts job predictability, and includes functions like flexible data ingest and replication. This is supported by a control plane handling multi-environment and multi-tenant cluster provisioning, deployment, management, and troubleshooting, described below. As data silos cause so many problems already detailed, SDX is really a core piece of how Cludera separates from the legacy and unintegrated competition.

Control Plane

Understanding that any piece of technology has requirements for administration, Cludera Enterprise is designed to minimize this tax on value. It includes many advanced features to monitor, provision, and manage resources and workloads. Cludera Manager in fact can handle up to 2,500 nodes in a single view. Cludera Director facilitates managing cloud infrastructure. Cludera Altus automates many functions entirely, letting you focus on the jobs at hand. All of these include support for rolling upgrades, automation of tasks, and normal housekeeping.

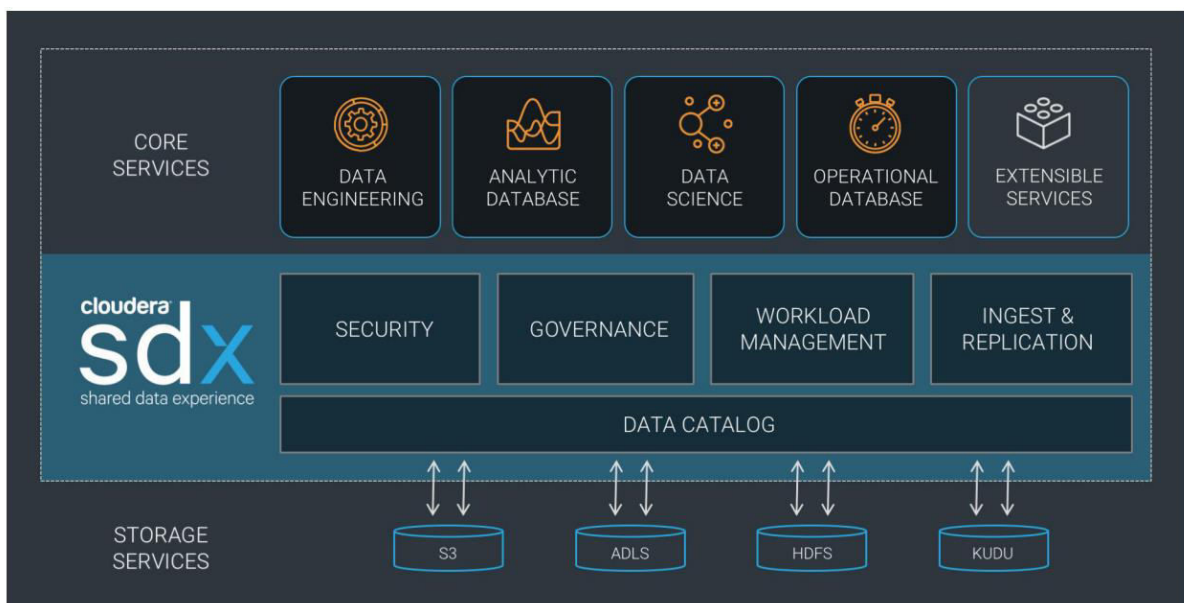
Cludera SDX: Shared Data Experience for On-Premise, Cloud and Hybrid Deployments Cludera Shared Data Experience (SDX) creates a seamless integration of all analytic disciplines with full security, governance and administration across any set of deployments: on-premise, cloud and hybrid

SDX – The data access control layer that sits on top of the backend object store and provides coherent data security and governance for all the applications running with the environment

Cloudera SDX: Shared Data Experience

Cloudera SDX offers a modular software framework that ensures a shared data experience across all deployment types, including multiple public clouds, private cloud, hybrid cloud, and bare metal configurations. SDX applies stateful, centralized, and consistent data context services making it possible for hundreds of different workloads to run against shared or overlapping sets of data. SDX makes multi-disciplinary data applications easier to develop, less expensive to deploy and increasingly important in today's environment, more consistently secure. SDX is comprised of five discrete functions that together solve a really hard problem — providing a shared data experience for a platform that supports a diverse set of workloads and user interaction models.

- **Shared Governance** provides the ability to govern the data in a unified manner so users can easily discover new data, understand where that data came from, and track how it has been modified.
- **Shared Security** implements consistent, granular authentication, authorization, encryption, and compliance controls in a unified manner across the entire platform.
- **Shared Workload Management** enables administrators to create, manage and optimize workloads individually or as a collection and to allocate resources and assign workload priority based on business requirements.
- **Shared Ingest & Replication** provide the ability to ingest data once and make it available to all applications and users without additional ingest pipelines or copies of data, and to replicate data on demand to remote locations or directly to the cloud.
- **Shared Data Catalog** provides a common catalog of schema and lineage metadata to each workload and user accessing the platform for maximum efficiency and productivity.



Data Catalog

- Data Catalog is a service within Cloudera Data Platform that enables you to understand, manage, secure, and govern data assets across the enterprise. Data Catalog helps you understand data across multiple clusters and across multiple CDP environments. You can search to locate relevant data of interest based on various parameters. Using Data Catalog,

you can understand how data is interpreted for use, how it is created and modified, and how data access is secured and protected.

Ranger

- Security policies across CDP can be created, modified and destroyed using Ranger Web Interface.

Data Catalog enables data stewards across the enterprise to work with data assets in the following ways:

- Organize and curate data globally
 - o Organize data based on business classifications, purpose, protections needed, etc.
 - o Promote responsible collaboration across enterprise data workers
- Understand where relevant data is located
 - o Catalog and search to locate relevant data of interest (sensitive data, commonly used, high risk data, etc.)
 - o Understand what types of sensitive personal data exists and where it is located
- Understand how data is interpreted for use
 - o View basic descriptions: schema, classifications (business cataloging), and encodings
 - o View statistical models and parameters
 - o View user annotations, wrangling scripts, view definitions etc.
- Understand how data is created and modified
 - o Visualize upstream lineage and downstream impact
 - o Understand how schema or data evolve
 - o View and understand data supply chain (pipelines, versioning, and evolution)
- Understand how data access is secured and protected, and audit use
 - o Understand who can see which data and metadata (for example, based on business classifications) and under what conditions (security policies, data protection, anonymization)
 - o -View who has accessed what data from a forensic audit or compliance perspective
 - o -Visualize access patterns and identify anomalies

[Get Full Version Contents from this link](#)



Contents

Chapter-17: Private Cloud Base HDFS Transparent Encryption.....	1
Overview	1
Architecture Overview	2
Encrypted data access.....	2
EDEKs, KeyProvider, KeyManagementServer	3
Private Cloud Base and Encryption	3
Design.....	8
KEY MANAGEMENT SERVER.	9
ENCRYPTED DATA ACCESS	9

Chapter-17: Private Cloud Base HDFS Transparent Encryption

Overview

HDFS Transparent Encryption safeguards Hadoop disc data. When a cluster's encryption is enabled, write and read operations on encrypted zones (HDFS folders) are encrypted and decrypted automatically. This method is "transparent" to the data-using programme. HDFS Transparent Encryption doesn't effect user access to Hadoop data but may influence performance. The cluster where you want to use HDFS Transparent Encryption must have Kerberos enabled. Cluster creation requires Security Setup. On the Security page of the Create Cluster wizard, pick Security Setup: Enabled. After creating a cluster, you can't activate Kerberos.

HDFS uses end-to-end encryption. Data read from and written to specific HDFS folders is transparently encrypted and decrypted once setup. End-to-end encryption means only the client can encrypt and decode data. HDFS never stores unencrypted data or keys. This meets both at-rest and in-transit encryption needs (e.g. when data is travelling over the network).

Different levels of a standard data management software/hardware stack may be encrypted. Each encryption layer has pros and cons.

- **APP encryption.** This method is secure and adaptable. The programme controls what's encrypted and may represent user needs. Application writing is difficult. Existing programmes without encryption can't use this feature.
- **Encrypting databases.** Comparable to application-level encryption. Database suppliers provide encryption. Performance difficulties might arise. Indexes aren't encryptable.
- **Encrypting filesystems.** This alternative is high-performing and straightforward to implement. Some application-level regulations can't be modelled. Multi-tenant apps may encrypt by user. Each column in a database file may need separate encryption settings.
- **Encrypting discs.** High-performance and easy to install, yet inflexible. Physical theft only.

Government, financial, and regulatory agencies demand data encryption. Health care has HIPAA, card payments have PCI DSS, and the government has FISMA. HDFS's transparent encryption helps enterprises comply with requirements.

By adding encryption into HDFS, existing applications may use encrypted data without modifications. This architecture integrates encrypted file semantics with HDFS functionality. HDFS-level encryption is between database-level and filesystem-level. It's beneficial. Existing Hadoop applications can operate transparently on encrypted HDFS data. HDFS gives policymakers more context than conventional filesystems.

HDFS-level encryption avoids filesystem- and OS-level attacks. Since HDFS encrypts the data, the OS and disc only communicate with encrypted bytes.

Architecture Overview

HDFS's encryption zone provides transparent encryption. The contents of an encryption zone are transparently encrypted when written and decrypted when read. Each encryption zone is generated with a single key. Each encrypted file has its own key (DEK). HDFS doesn't handle DEKs. HDFS exclusively handles encrypted key data (EDEK). After decrypting an EDEK, clients read and write data using a DEK. HDFS datanodes view encrypted bytes.

Encrypting all files in a filesystem is a key use case of encryption. HDFS provides stacked encryption zones so that various encryption zone keys may be used in different portions of the filesystem. After creating an encryption zone (e.g. on /), a user may build additional with different keys on descendent directories (e.g. /home/alice). The file's EDEK is produced using its ancestor's encryption zone key.

Hadoop Key Management Server manages encryption keys (KMS). KMS has three main roles in HDFS encryption:

1. Encryption zone key storage
2. Creating new NameNode encryption keys
3. Decrypting HDFS client encryption keys

Below, we'll explain the KMS.

Encrypted data access

NameNode requests a new EDEK encrypted using the encryption zone's key when creating a new file. EDEK is saved as file metadata on NameNode. When reading a file in an encryption zone, the NameNode supplies the client with the file's EDEK and encryption zone key version. The client queries the KMS to decode the EDEK, which requires validating authorization to access the encryption zone key version. If it works, the client decrypts the file using the DEK.

All of the aforementioned read and write route stages happen automatically via DFSClient, NameNode, and KMS interactions. HDFS filesystem permissions govern access to encrypted data and

metadata. If HDFS is breached, a malicious user only has access to ciphertext and encrypted keys. Since KMS and key store permissions govern access to encryption zone keys, this is not a security risk.

EDEKs, KeyProvider, KeyManagementServer

The KMS acts as a proxy between HDFS daemons and clients and a key store. Both the key store and KMS implement Hadoop's KeyProvider API. For more, see KMS documentation. Each encryption key in KeyProvider has a unique name. Each key version has its unique key material since keys may be rolled (the actual secret bytes used during encryption and decryption). A key may be obtained by name, which returns the newest version, or by version.

The KMS can create and decode encrypted encryption keys (EEKs). KMS creates and decrypts EEKs. The client requesting EEK generation or decryption never touches the key. The KMS produces a random key, encrypts it with the provided key, and provides it to the client. KMS verifies if the user has the encryption key, decrypts the EEK, and returns it.

EEKs are encrypted data encryption keys (EDEKs) utilised by HDFS to encrypt and decode file data. Typically, the key store only lets end users access DEK-encrypting keys. The HDFS user won't have access to unencrypted EDEKs, thus they may be securely stored and managed.

Configuration

KMS instance and key store are required. After setting up a KMS and configuring the NameNode and HDFS clients, an admin may use the `hadoop key` and `hdfs crypto` command-line tools to produce encryption keys and encryption zones. Using technologies like `distcp`, existing data may be encrypted.

Private Cloud Base and Encryption

Encrypting data at rest is a desired or sometimes essential requirement for data platforms in HealthCare, Financial, and Government enterprises. The feature protects sensitive data from internal and external attacks. Apache Ranger manages HDFS data. Administrators oversee HDFS activity through regulations and audit logs. Any HDFS admin or root user on cluster nodes might mimic "hdfs" and access sensitive data in plain text. HDFS Encryption protects transparent text. This method ensures the safety of sensitive and personal data that, if disclosed in an accidental or deliberate breach, would harm both people (customers, workers, partners) and the company as a whole.

HDFS Encryption encrypts data at rest transparently end-to-end. Only the client encrypts and decrypts end-to-end data. Until it reaches the HDFS client, data is encrypted. Each HDFS file is encrypted. Encryption keys are kept in file metadata to avoid performance bottlenecks caused by managing millions of keys. The file encryption key is encrypted using another "encryption zone key" for further protection.

Easy configuration. Ranger's key management rules govern decryption access to HDFS data. HDFS Native encryption works with Protegrity Tokenization, which tokenizes and detokenizes encrypted HDFS data according on Protegrity ESA server settings. Ranger's dynamic column masking capabilities include redacting, hashing, and masking data on top of encrypted HDFS data for further protection.

HDFS encryption paired with column masking capabilities by Ranger and/or Protegrity protects data at rest, across the network, and via permission controls.

Encryption/Decryption Flow:

Documentation and publications describe how HDFS encrypts data. Here's the fundamental flow:

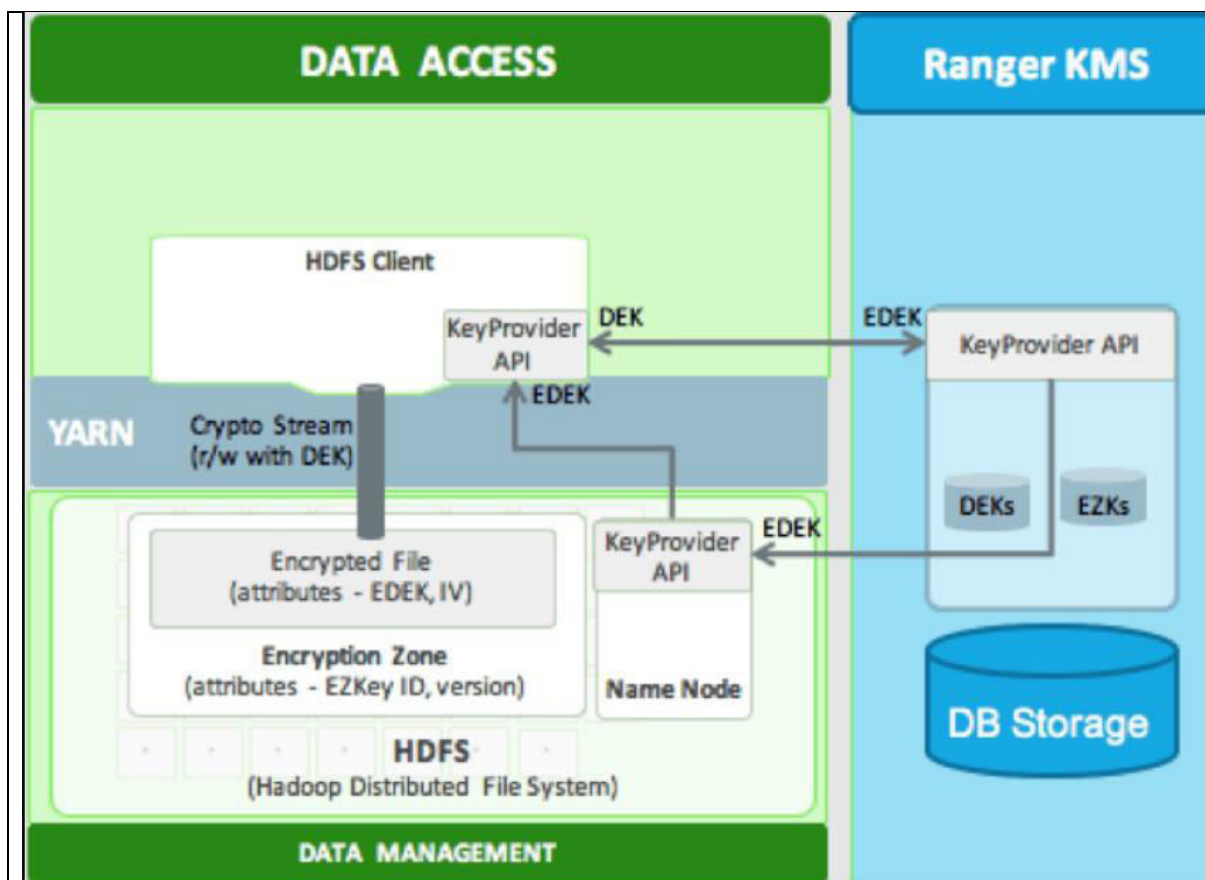
Encryption:

1. To encrypt HDFS files, build an EZK.
2. This is an empty HDFS folder attached with an EZK.
3. Every HDFS encrypted file gets a data encryption key (DEK).
4. DEK encrypts the file.
5. EZK encrypts DEK to create an encrypted data encryption key (EDEK).
6. Each file's metadata has an EDEK.

Decryption:

1. Accessing an encrypted file needs "DECRYPT" on the appropriate EZK.
2. "hdfs dfs -cat" generates a KMS API request to check "DECRYPT" access.
3. EZK decrypts EDEK if the user has access.
4. The DEK decrypts the file to show the user.

The following figure demonstrates how HDFS utilises Key provider API to decode EDEK and access file contents:



The graphic below depicts the relationship between EZK, DEK, and EDEK.



Installing:

HDFS Native Encryption depends on Ranger KMS to create encryption zone keys and permission rules to provide ENCRYPT, DECRYPT access. Ranger KMS must be installed and configured to allow HDFS Native Encryption.

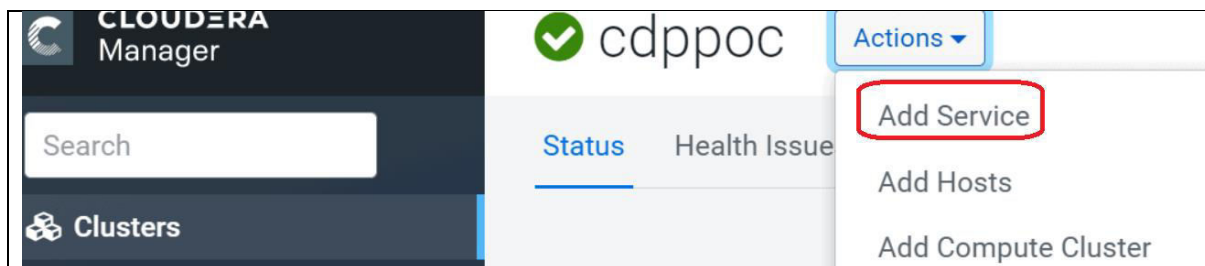
Ranger KMS requires a backend to hold encryption zone keys. Cloudera Manager provides two ways to install Ranger KMS:

1. RDBMS-supported Ranger KMS.
2. KTS backs Ranger KMS.

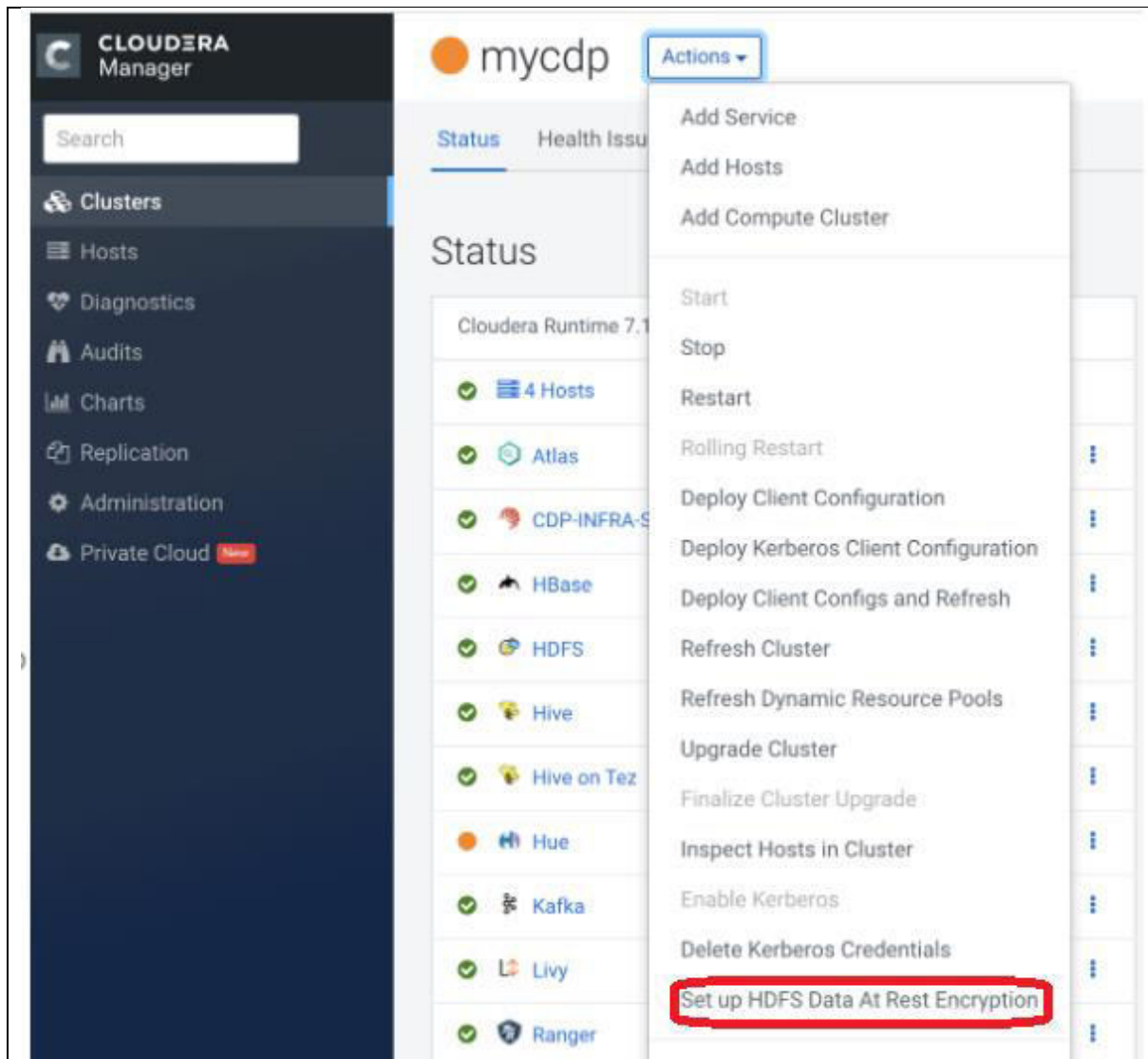
Cloudera Manager simplifies configuring both.

Encrypt HDFS at Rest:

In Cloudera Manager (CM), click "Clusters" and then "mycdp"



Click "Actions" and "Set up HDFS Data at Rest Encryption"



"Setup HDFS Data At Rest Encryption" in Cloudera Manager's UI offers three options:

- Ranger KMS/RDBMS.
- Ranger KMS/KTS
- Keystore file-based.

This screenshot displays all three options:

Set up HDFS Data At Rest Encryption for mycdp

HDFS Encryption implements transparent, end-to-end encryption of data read from and written to HDFS, without requiring changes to application code. Because the encryption is end-to-end, data is encrypted and decrypted only by the client. HDFS does not store or have access to unencrypted data or encryption keys. [Read the Cloudera documentation before enabling encryption](#)

The root of trust for encryption keys can either be:

Ranger Key Management Service backed by Key Trustee Server

Ranger Key Management Service backed by Key Trustee Server is a Hadoop Key Management Service implementation that sources encryption zone keys from a backing Key Trustee Server. For HSM integration please refer to documentation.

Ranger Key Management Service backed by Database

Ranger Key Management Service backed by Database is a Hadoop Key Management Service implementation that sources encryption zone keys from a backing database. For HSM integration please refer to documentation.

A file-based password-protected Java KeyStore

The file-based Java KeyStore may not be sufficient for large enterprises where a more robust and secure key management solution is required. It is **not suitable** for production use.

After the root of trust is chosen, a new service called the Hadoop **Key Management Server** (KMS) must be added to your cluster.

Choosing the first option in the screenshot, "Ranger Key Management Service powered by Key Trustee Server," invites Cloudera Manager to fulfil a few requirements and set up KTS infrastructure. The following should be done before activating HDFS Data at Rest Encryption:

- Activate kerberos.
- Activate TLS/SSL.

KTS infrastructure may be established two ways:

- Add KTS cluster (helps manage KTS infrastructure outside the cluster, and it is a best practise as well).
- Parcelize KTS (it requires parcels to be downloaded from archive.cloudera.com, and configure into CM).

Once KTS is in place,

- Select "Add service" on Cloudera manager's UI to add KTS.
- Select "Add service" on Cloudera Manager UI to add Ranger KMS with Key Trustee Server.

Set up HDFS Data At Rest Encryption for mycdp

HDFS Encryption implements transparent, end-to-end encryption of data read from and written to HDFS, without requiring changes to application code. Because the encryption is end-to-end, data can be encrypted and decrypted only by the client. HDFS does not store or have access to unencrypted data or encryption keys. [Read the Cloudera documentation before enabling encryption](#).

The root of trust for encryption keys can either be:

Ranger Key Management Service backed by Key Trustee Server

Ranger Key Management Service backed by Key Trustee Server is a Hadoop Key Management Service implementation that sources encryption zone keys from a backing Key Trustee Server. For HSM integration please refer to documentation.

Ranger Key Management Service backed by Database

Ranger Key Management Service backed by Database is a Hadoop Key Management Service implementation that sources encryption zone keys from a backing database. For HSM integration please refer to documentation.

A file-based password-protected Java KeyStore

The file-based Java KeyStore may not be sufficient for large enterprises where a more robust and secure key management solution is required. It is **not suitable** for production use.

After the root of trust is chosen, a new service called the Hadoop **Key Management Server (KMS)** must be added to your cluster.

The following steps are required to set up HDFS Encryption. Click the links below to complete each step.

Note: This workflow will not encrypt data automatically. You must manually create encryption keys and encryption zones and move data into them.

Step	Status	Notes
1 Enable Kerberos	✔ Completed	
2 Enable TLS/SSL View Documentation		Strongly Recommended. Otherwise, all of your encryption keys will be transmitted in plain text.
3 Add a dedicated cluster for the Key Trustee Servers		Optional if an external Key Trustee Server cluster (not managed by this instance of Cloudera Manager) is used.
4 Install Key Trustee Server binary using packages or parcels		Add a Key Trustee Server cluster to enable this step.
5 Add Key Trustee Server Service		Add a Key Trustee Server cluster to enable this step.
6 Add Ranger KMS with Key Trustee Server Service		
7 Restart stale services and redeploy client configuration		

HDFS encryption can:

- HDFS clients may encrypt/decrypt data.
- HDFS doesn't handle keys. HDFS can't read unencrypted data or keys. HDFS and key administration are independent user roles (HDFS administrator, Key Administrator), guaranteeing no one user has full access to both data and keys.
- The OS and HDFS communicate utilising encrypted HDFS data, minimising OS and file-system vulnerabilities.
- HDFS employs AES-CTR encryption. AES-CTR offers a 128-bit encryption key (default) or 256-bit when maximum strength JCE is enabled.
- HDFS encryption uses the AES-NI instruction set, a hardware-based encryption accelerator, thus setting encryption shouldn't effect cluster performance. The AES-NI instruction set is quicker than software implementations. To utilise the acceleration technique, you may need to upgrade HDFS and MapReduce cryptography libraries.

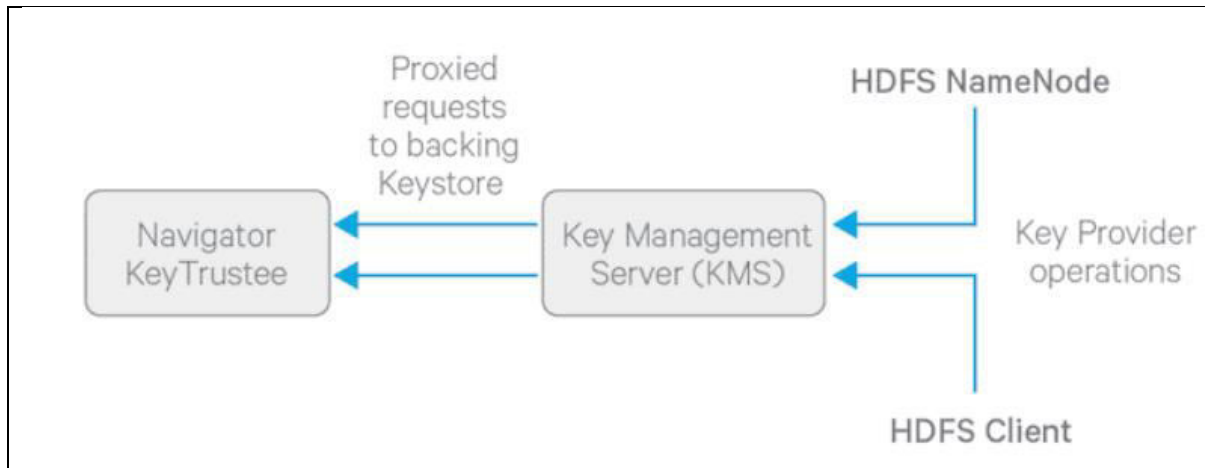
Design

HDFS encryption has special issues. Distributed filesystems prioritise performance and scalability. Transferred data must be encrypted. As HDFS is a multi-user system, we must be cautious not to disclose sensitive information to other users, especially administrators with HDFS superuser access or cluster root shell access.

All of the following must happen without changing user application code for transparent encryption. Encryption must support WebHDFS, HTTPFS, FUSE, and NFS.

KEY MANAGEMENT SERVER.

HDFS integration with Cloudera Navigator Key Trustee was a design objective. Most keystores aren't built for Hadoop workloads and lack a consistent API. We created Hadoop Key Management Server for these reasons (KMS).



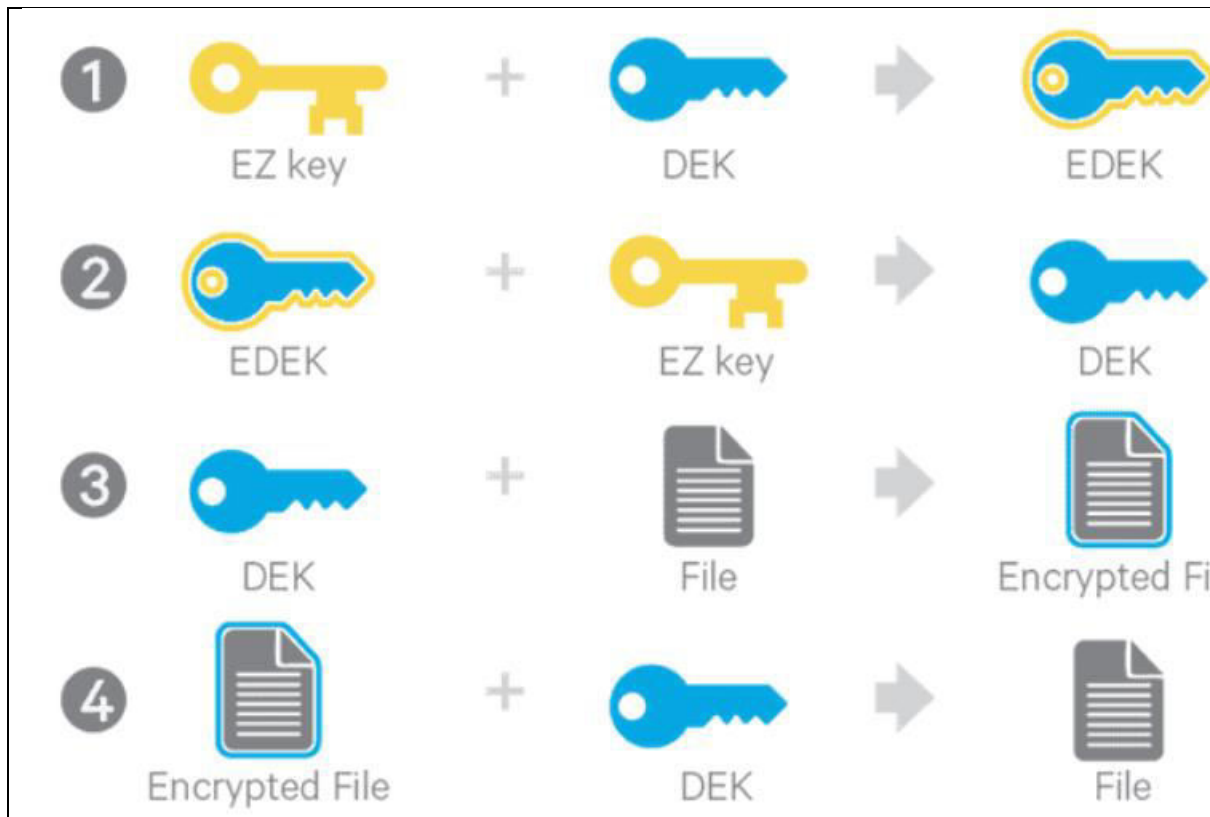
The KMS operates as a proxy between cluster clients and a keystore, offering Hadoop's KeyProvider interface through REST API. Any keystore with the needed capabilities may be included into the KMS.

KMS doesn't store keys (other than temporarily in its cache). The corporate keystore must be the official key storage and guarantee that keys are never lost, since a missing key destroys data. For production, install two or more enterprise key stores.

The KMS enables ACLs that granularly regulate key access and actions. This functionality may be used to limit NameNode and DataNode's access to keys.

ENCRYPTED DATA ACCESS

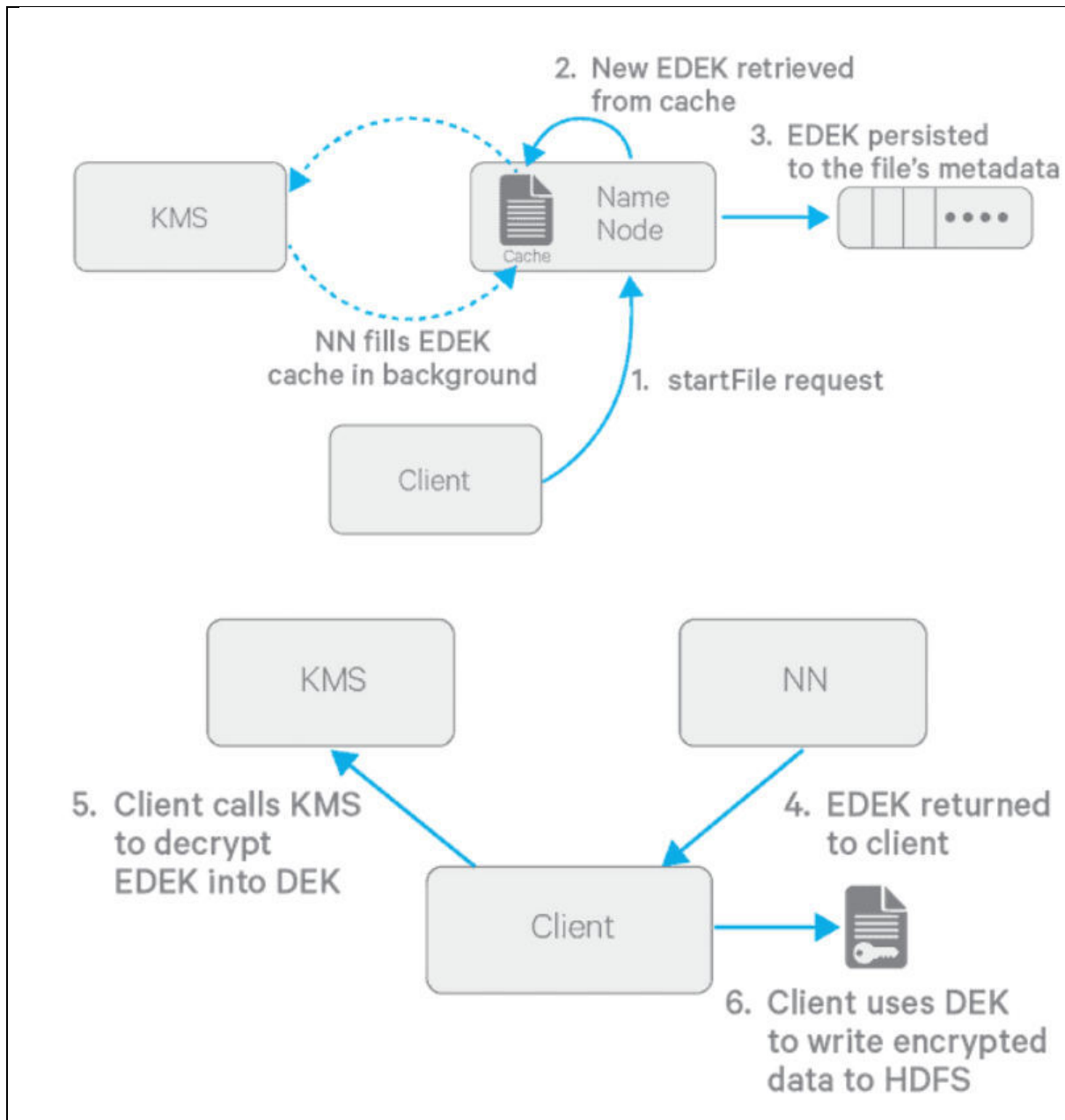
The new design provides an encryption zone (EZ), a directory in HDFS whose contents are automatically encrypted on write and decrypted on read. Encryption zones start off empty and cannot be renamed into or out of. A EZ's contents are always encrypted.



When creating an EZ, the administrator provides a key from the backing keystore. The EZ Key encrypts the file encryption keys (DEKs). DEKs are encrypted with the EZ key to produce EDEK, which is saved on the NameNode through a file attribute (1).

The client obtains a new EDEK from the NameNode and requests the KMS to decode it using the EZ key. This creates a DEK (2) that the client may use to encrypt data (3).

To decrypt a file, the client must decode its EDEK using the EZ key (2). The client decrypts encrypted data using DEK (4).



The graphics above explain how to encrypt a file. Important is the background-populated per-EZ EDEK cache on the NameNode. This avoids calling the KMS for each creation call. Note that HDFS never handles the EZ key directly; KMS generates and decrypts EDEKs.



Contents

Chapter-23: Cloudera Data Engineering for CDP-0011 Certification	2
CDP Data Engineering (CDE)	2
Cloudera Data Engineering	3
Components for CDE Services	5
Apache Airflow and Data Pipeline	9
Cloudera Data Engineering Use Cases	9
Data Engineering at Scale	10
Data Engineering Lifecycle	10
Data Collection & Acquisition Steps.....	11
Data Acquisition	11
Data Aggregation	11
Data Cleansing Steps.....	12
Data Sampling and Selection	12
CDP Airflow	12

HadoopExam.com

Chapter-23: Cloudera Data Engineering for CDP-0011 Certification

CDP Data Engineering (CDE)

Data Engineering provides a complete data processing solution, powered by Apache Spark and Apache Hive. Spark and Hive enable fast, scalable, fault-tolerant data engineering and analytics over petabytes of data.

This Data Engineering template includes a standalone deployment of Spark and Hive, as well as Apache Oozie for job scheduling and orchestration, Apache Livy for remote job submission, and Hue and Apache Zeppelin for job authoring and interactive analysis.

CDP Data Engineering is a powerful Apache Spark service on Kubernetes and includes key productivity enhancing capabilities typically which are not available without basic data engineering services. This provides

- GUI based monitoring, performance tuning and troubleshooting for faster problem resolution.
- It also has native Apache Airflow and robust APIs for orchestrating and automating job scheduling and delivering complex data pipeline anywhere.
- It provides resource isolation and UI based job management.
- It provides CDP data lifecycle integration and SDX security and governance.

Cloudera Data Engineering (CDE) in CDP Private Cloud is intended to be installed and used in locally hosted information technology settings. Additionally, the business revealed that CDE is now compatible with the Amazon Elastic Compute Service (ECS) cloud platform in addition to the Red Hat OpenShift platform, which is based on Kubernetes.

On top of Kubernetes clusters, the Cloudera public cloud version of CDE is used to install the Apache Spark framework for the purpose of doing data analysis. IT and software development teams can now construct applications on either a public or a private platform, and then run those applications

across a hybrid cloud computing environment. This is accomplished by first shifting workloads, and then utilizing tools such as Cloudera Replication Manager to migrate data when it is necessary.

IT & Software Developer teams can programmatically deploy complex pipelines with job dependencies using Apache Airflow, open source software based on directed acyclic graphs (DAGs) that makes it possible to visualize and monitor pipelines running in production environments.

It is becoming more apparent, as Kubernetes gains more widespread use, that the administration of compute, data, and applications will eventually converge around a single control plane. It is possible that those tasks will still need to be managed by a group of IT experts, but the days in which IT teams were required to install a distinct control plane for each function are drawing to a close. As the number of stateful apps that are deployed on Kubernetes clusters continues to gradually expand, this control plane offers not only the means to combine DevOps and data operations but also provides the control itself.

The CDP Data Engineering service is the only cloud-native offering that was developed specifically for the needs of business data engineering teams. Data Engineering is an all-inclusive data engineering toolset that is built on Apache Spark. It enables orchestration automation with Apache Airflow, advanced pipeline monitoring, visual troubleshooting, and comprehensive management tools to streamline ETL processes across enterprise analytics teams. Data Engineering is built on Apache Spark.

Data Engineering is completely integrated with Cloudera Data Platform, allowing end-to-end visibility and security with SDX and smooth interactions with CDP services like Data Warehouse and Machine Learning. CDP Data Engineering enables uniform, repeatable, and automated data engineering procedures wherever.

Cloudera Data Engineering

Cloudera Data Engineering or CDE is a serverless service for CDP using which we can submit Spark jobs to an auto-scaling cluster, you don't have to worry about, how cluster to be created in Cloud. This service is available in both the form factor whether is public cloud or private cloud deployment.

CDE is a service in CDP Private Cloud Data Services allows you to submit jobs to auto-scaling clusters. In fact, you can create, manage and schedule Apache Spark jobs without the overhead of creating and maintaining Apache Spark clusters. With the CDE, you define virtual clusters with a range of CPU and memory resources and the virtual cluster scales up and down as needed to run your Spark Workloads.



Data Engineers who are using Apache Spark, CDP CDE offers an inclusive toolset that enables data pipeline orchestration, automation, advanced monitoring, visual profiling and comprehensive management toolsets for streamlining ETL processes and making complex data actionable across your analytics.

Apache Airflow-based pipeline orchestration for Cloudera Data Platform (CDP) with scalable Spark and Hive transformations. DE, DW, and ML practitioners may build curated datasets for downstream applications using Spark and Hive. A managed Airflow service schedules and orchestrates pipelines without the expense of implementing an external scheduling service. Customers solely pay for computational infrastructure.

Customers may now configure their data pipeline using a simple python configuration file. The steps may be a combination of hive and spark operators that conduct tasks on CDW and CDE, with SDX providing security and governance. Job sequencing may involve retries, dependencies, and conditional branching. Customers may now design modular data transformation stages that are more reusable and simpler to debug, which can be coordinated with glueing logic at the pipeline configuration level instead of buried in code. With CDW, we've created a new processing mode that mimics Hive on Tez for ETL workloads, combining containerization and dedicated resources.

As data pipelines expand in complexity, size, and breadth, being agile depends on the solution's strength and adaptability. Most production data pipelines have one or more of these flaws.

- Hard to scale, particularly multi-stage conversions.
- Lack of automation to achieve SLAs with high-quality data sets.
- Limited insight into data pipeline health and progress.
- Difficult to diagnose and solve pipeline faults.
- Harmonize data pipeline and downstream application security standards.
- Incomplete lineage of source-to-target data pipelines.

CDE is the first cloud-native solution developed for corporate data engineering teams building complicated, dependable data pipelines at scale and across LOBs. CDE is an all-inclusive data

engineering solution that automates orchestration, monitors pipelines, provides visual debugging, and streamlines ETL procedures for business analytics teams. CDE addresses the problems noted previously, notably in end-to-end workflows, by combining Apache Spark, Apache Hive, Apache Airflow, and Apache Atlas to enable:

- Complex data processing using data frames, SQL, or low-level distributed data sets ensures dependable multi-stage transformations at scale.
- Built-in monitoring and troubleshooting tools for pipeline concerns.
- Robust orchestration with Airflow provides flexible flows of Spark and Hive integration and transformation processes leveraging recent advancements such as big query isolation in CDP's Data Warehouse service.
- Reliably and swiftly implement end-to-end LOB processes with CDP's Data Flow, Data Warehousing, and Machine Learning services.
- Integrated with CDP common services like as SDX and Workload Management to deliver unified security standards, lineage traceability for end-to-end processes, and optimum data pipeline health.
- Spark can integrate with several 3rd party data sources to create a large library for CDE. Common databases (redshift, snowflake, mongo, hbase) and file formats (avro, parquet, ORC, csv) are included here.
- ISV interaction through CDE APIs (latest partner integration blog).

Components for CDE Services

- **Environment**

This is a logical subset of your private cloud deployment, including a datalake and multiple compute resources.

- **CDE Service:**

CDE Service is a subset of the long-running Kubernetes cluster and services that manage the virtual clusters. The CDE service must be enabled on an environment before you can create any virtual clusters.

- **Virtual Cluster**

An individual auto-scaling cluster with defined CPU and memory ranges. Virtual Clusters in CDE can be created and deleted on demand. Jobs are associated with Clusters.

- **Job**

This is your application code, configuration and resources. Jobs can be run on demand or scheduled. An individual Job execution is called a Job-run.

Resources: A resource in CDE is a named collection of files which used by a Job. Resources can include

- Application Code
- Configuration files.
- Custom Docker Image
- Python virtual environment specifications (requirements.txt)

Resources are associated with virtual clusters. A resource can be used by multiple jobs., and a single job can use multiple resources. The resource type which are supported by CDE include files and python-env.

Files: These are the files which a job can refer. This includes application code, configuration files and supporting libraries. You can upload and remove new files as needed.

Python-env: This is a defined virtual Python environment that a job runs in. The only file you can upload in python-env is requirement.txt file. When you associate a python-env resource with a job, the job runs within a Python virtual environment built according to the requirement.txt specifications.

Custom-runtime-image: This is a Docker container image, when you run a job using a custom-runtime-image resource, the executors that are launched use your custom image.

- **Job run**

As discussed above this is an individual job run.

CDE Prerequisites

If you want to use CDE on CDP Private Cloud, you must have followed requirement fulfilled.

Apache Ozone: The CDP Private Cloud Base cluster must have Apache Ozone service enabled, if you want to use CDE in CDP Private Cloud Base.

OpenShift Container: For CDP Private Cloud running on Red Hat OpenShift Container Platform (OCP), you must configure a route admission policy.

CDE Supported Jobs

- In Cloudera Public Cloud, currently CDE supports the Scala, Java and Python Jobs.
- When you create a Cloudera Data Engineering (CDE) service, you specify an instance type (size) and auto-scale range of instances. Virtual clusters associated with the CDE service use CPU and memory resources as needed to run jobs. When more resources are required, virtual machines of the specified instance type are started. When resources are no longer required, instances are terminated.
- CDE allows you to create, manage, and schedule Apache Spark jobs without the overhead of creating and maintaining Spark clusters.
- In addition to this, you can define virtual clusters with a range of CPU and memory resources, and the cluster scales up and down as needed to execute your Spark workloads, helping control your cloud costs.

CDE Jobs Example

Example-1: You can read network access logs stored in S3, extract the whole data and inserts into an Hive table and then this data you can use in your Downstream process.

CDE Job Scheduling & Deployment

- You can run job ad-hoc basis or schooled basis.
- If you have already built your Spark code in your laptop, this can be deployed as CDE job with just a single click.

- You can use a simple wizard where you can define all the key configuration of your job.
- DE supports, Scala, Java and Python Jobs.
- Cloudera kept in mind that, there should be less number of fields required to run a job and exposed all the typical configurations data engineers needed or expected. For example
 - o Runtime Arguments
 - o Overriding Default Configurations
 - o Including Dependencies (this can be Jars, config files or python egg files). Resources are managed as single entity and called resource.
 - o Behind the scenes resources has proper versioning to ensure that whenever the job run, the correct dependencies are available.
 - o Resources are automatically mounted and available to all Spark executors alleviating the manual work of copying files on all the nodes.
 - o Applying Resource Parameters.
- As you can see below

The screenshot displays the 'Jobs / Create Job' configuration page in Cloudera Data Engineering. The left sidebar shows navigation options: Job Runs, Jobs, Resources, and Schedules. The main content area is divided into several sections:

- Job Details:** Name (Ingestion ETL Job), Spark Application File (insurance-cde-app-1.1-sdx.jar), Main Class (org.cloudera.cde.app.Application).
- Arguments (Optional):** targetDB
- Configurations (Optional):** spark.yarn.access.hadoopFileSystems
- Advanced Options:** Includes a toggle for 'Advanced Options' and a section for 'Jar Files' with an 'Upload' button and 'sdx.conf' dependency.
- Schedule:** Set to 'Every year on every day of every month'.
- Resource Allocation:** Sliders for Executors (32), Driver Cores (8), Executor Cores (4), Driver Memory (8 GB), and Executor Memory (16 GB).

Apache Airflow

- Once Job is defined CDE job, its execution is handled by Apache Airflow.
- Cloudera Data Engineers introducing a completely new orchestration service backed by Apache Airflow.
- Airflow allows you defining pipelines using Python code that are represented as entities called DAGs.

- CDE automatically takes care of generating the Airflow python configuration using the custom DE operator. As you can see below

The screenshot shows the Cloudera Data Engineering (CDE) interface. On the left, a sidebar contains navigation options: Job Runs, Jobs, Resources, and Schedules. The main area is titled 'Schedule' and shows a list of DAGs (Directed Acyclic Graphs) under the heading 'DAGs'. The list includes DAGs like 'analytics-pyspark-job1', 'demo-etl-job', 'demo-job-1', and 'ingestion-ETL-job-1'. The 'ingestion-ETL-job-1' DAG is selected, and its configuration is displayed in a code editor on the right. The code defines a DAG with a single task 'ingestion-ETL-job-1' using a custom operator 'CDEJobRunOperator'.

```

1 from dateutil import parser
2 from datetime import timezone
3 from airflow import DAG
4 from cde_job_run_operator.operator import CDEJobRunOperator
5
6 default_args = {
7     'owner': 'Airflow',
8     'depends_on_past': False,
9     'wait_for_downstream': False,
10    'start_date': parser.isoparse('2020-08-04T15:12:47.262Z').replace(tzinfo=timezone.utc),
11    'end_date': parser.isoparse('2020-08-05T15:12:47.262Z').replace(tzinfo=timezone.utc),
12    'connection_id': 'cde_runtime_api',
13    'job_name': 'ingestion-ETL-job-1',
14    'user': 'sahmadian',
15 }

```

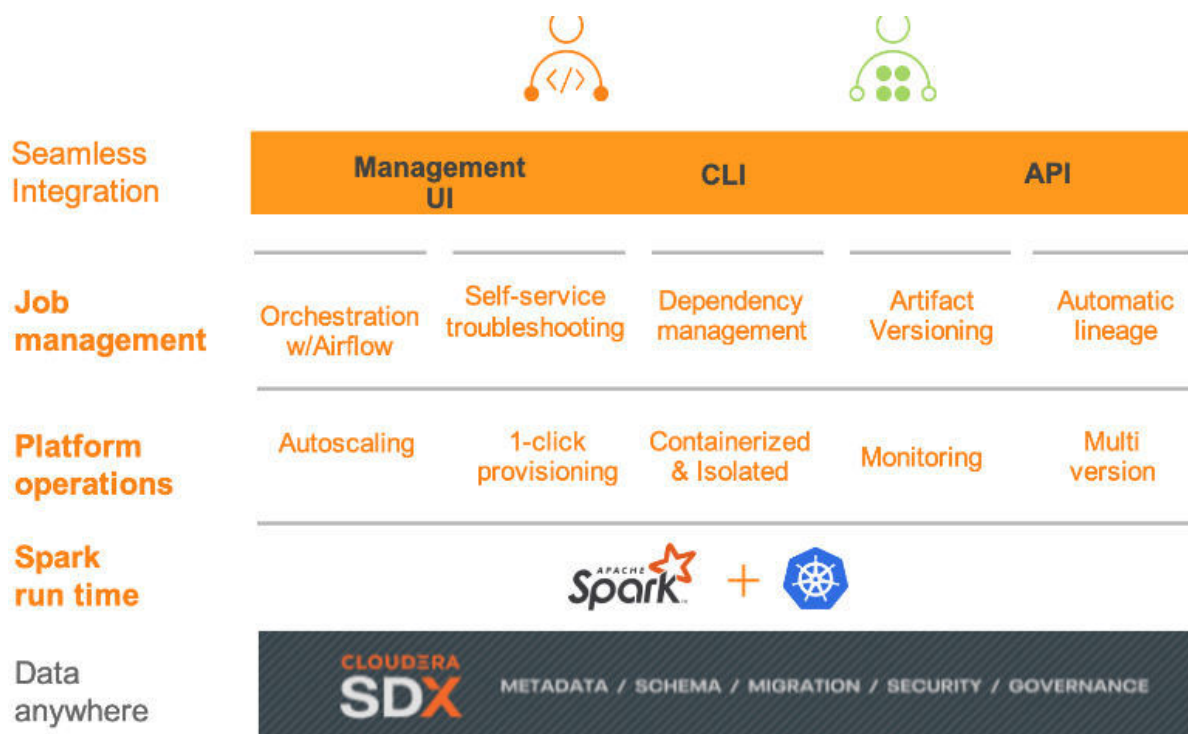
- With Apache Airflow, Data Engineers can use many of the hundreds of communities contributed operators to define their own pipeline.
- This will allow defining of custom DAGs and scheduling of jobs based on certain event triggers like an input file showing up in an S3 bucket.

Apache Spark

- Apache Spark has become the de-facto processing framework for ETL and ELT workflows.
- However, we need to optimize it correctly, since it can become challenging and resource intensive.
- Usually, Spark typical workload in Hadoop context is Hadoop clusters with YARN running on VM or physical servers.
- Alternative deployment is Apache Spark on Kubernetes, which also requires considerable effort to setup, manage and optimize performance.
- Usually, Data Engineers, who want to deploy Apache Spark in public cloud are looking for ephemeral compute resources that auto scale based on demand. CDP Public Cloud has this option available via Data Hub Clusters, which provide Hadoop cluster form-factor that can then be used to run ETL jobs using Spark.
- **Data Hub Clusters:** It was observed that the majority of the time the Data Hub clusters are short lived, running for less than 10 minutes.
- Apache Spark clusters has the ability to scale up and down on-demand and well suited for containerization based on Kubernetes. In fact, they are also portable across cloud providers and hybrid deployments.
- CDP DE is architected with this keeping in mind, which offering a fully managed and robust serverless Spark service for operating data pipelines at scale.

Kubernetes

- Leveraging Kubernetes to fully containerize workloads , DE provides a built-in administration layer that enables one click provisioning of autoscaling resources with guardrails, as well as a comprehensive job management interface from streamlining pipeline delivery. Which is depicted as below



Because modularity and portability are two of the most important aspects of CDE, and its primary emphasis was on developing a service that is fully managed and production ready for Spark on Kubernetes. Because of this, CDP is able to create storage and computation layers that were disaggregated and could scale independently according to the needs of the workload.

Apache Airflow and Data Pipeline

Data pipelines are made up of a number of different phases, each of which may have dependencies or triggers. A versatile orchestration platform, such as Apache Airflow, that provides simpler automation, dependency management, and customization is required to satisfy the ever-evolving requirements of businesses of all sizes. While alleviating the traditional operational management overhead of security and availability, packaging Apache Airflow and exposing it as a managed service inside CDE enables data engineers to plan and monitor multi-step pipelines using a task management API.

Cloudera Data Engineering Use Cases

1. To fulfil the requirements for mission-critical analytics, you may orchestrate complicated data transformation processes supported by Apache Airflow and including hundreds of operators.
2. Containerized, scalable, and portable, with separated workload environments and guardrails, Data Engineering enables safe pipeline management with on-demand elastic computing to satisfy business SLAs in the most cost-effective manner possible.
3. During the debugging process, it is helpful to see performance metrics like as CPU, memory, and I/O across all phases of your Spark operations. This will allow you to locate performance bottlenecks and find the needle in the haystack.

4. Utilize a powerful task management interface by means of a command-line interface (CLI) and REST APIs to easily automate and connect with pre-existing processes such as CI/CD pipelines and third-party applications.
5. A fully integrated Spark on Kubernetes service is provided by Data Engineering. This service automates and optimises artefact management, security, and resource scheduling by using Apache Yunikorn to provide FIFO and GANG scheduling.
6. Platform administrators are able to control access and security from a single interface, then swiftly provision new workloads while simply monitoring capacity and visualising resource utilisation over time. SDX also offers complete lifecycle lineage tracing, which makes it possible to discover the origin of data and determine its destination.

Data Engineering at Scale

- Apache Spark is the extract/transform/load (ELT) framework of choice, but there's a catch for commercial users. Apache Spark helps data engineers handle enormous data volumes in real time. Its ability to expedite data intake, exploration, modelling, curation, and categorization allows users develop batch or streaming pipelines fast. Spark's processing power isn't without physical labor. ETL tasks are resource- and time-intensive and might affect analytical procedures. When a data pipeline is ready for deployment, you must allocate resources, plan for it, and schedule it. Even after deployment, you must verify the proper dependencies are in production and monitor the task for errors.
- Debugging or tuning the work will cost time, resources, and headaches. To uncover a bottleneck or underlying problem, you must manually acquire and examine logs. When it's time to update Spark, the whole cluster must be brought down, halting operations.
- Spark can handle massive data volumes quickly, but it challenges with good data engineering at scale, which hurts advanced analytics projects.
- Building, implementing, and maintaining data pipelines requires a simplified, safe approach to data integration, modelling, optimization, quality, governance, security, and reusability.
- CDP Data Engineering (DE) simplifies data pipelines for business analytics and machine learning using Spark. DE streamlines your data pipeline management lifecycle, speeding corporate data from input to insight.
- DE can process data at any scale since it's containerized and multi-cloud portable. Autoscaling workload resources eliminates manual provisioning. This regulates expenses and guarantees work resources when demand rises.

Data Engineering Lifecycle

Data preparation and curation are equally important data lifecycle steps. Data preparation solutions for machine learning ensure that the data used to train models is clean, accurate, comprehensive, and relevant. AI-relevant data preparation includes:

- **Select Data:** You will want to check that the data chosen are accurate. Be careful to provide a reason for including or excluding certain pieces of data in your analysis.
- **Clean Data:** In most cases, the data that is compiled from a wide variety of sources does not arrive in an organized way. Data cleaning comprises updating or removing inaccurate data, anonymizing data, decreasing all types of data noise, and standardizing formats across various data sources.
- **Construct Data:** Record Attributes Derived from Generated Records

- **Integrate Data:** You need to combine the data that has been gathered from a variety of different sources.
- **Format Data:** Standardizing file formats across various data sources is something that has to be done in order for machine learning algorithms to correctly utilise the data that you provide (data types, fields, matched formats, currency or metric conversions, etc.).
- **Dataset:** You need to make sure that your datasets include up-to-date and correct information rather than outdated, archaic, or irrelevant information that might potentially contaminate the ensuing models. If you are going to employ a sub-section of your data, ensure that you have a description of the dataset that was applied to the analysis.

Data Collection & Acquisition Steps

- Big data drives AI. Big data and the know-how and infrastructure to handle it are driving the AI renaissance. In the past, AI progress stagnated due to restricted data sets, lack of representative sample data, and inability to cope with lots of data. Today's corporations have real-time, infinite data with machine learning capacity.
- Big data has taught us the 4 V's of data:
 - o **Volume:** How to manage terabytes of resting data.
 - o **Velocity:** Here's how to swiftly handle streaming data.
 - o **Variety:** How to manage organised, unstructured, text, multimedia, and more data.
 - o **Veracity:** How to handle inconsistent, partial, unclear, late, and approximated data.

Data Acquisition

Data engineering is a group of tools for moving, manipulating, and operating on massive data collections. Data engineering technologies stem from decades-old Extract-Transform-Load (ETL) techniques. AI-relevant ETL features include:

- Extracting useful data from existing data stores, data lakes, data warehouses, and other structured and unstructured information repositories for machine learning model training data sets or support.
- Transforming data using rules or combination logic to build, create, and maintain machine learning models.
- Loading converted data into the proper repository and format for machine learning model development, training, and maintenance.

Many firms process huge data in the cloud. ETL data and real-time streaming data are delivered to the cloud rather than being stored locally. This improves a company's flexibility, agility, dependability, and security.

Data Aggregation

Combining data from numerous sources is a fundamental stage in data collecting for training machine learning models. This entails collecting and integrating structured and unstructured data from numerous sources.

Key data aggregation steps include:

- Identifying structured and unstructured data sources to support machine learning.
- Determining missing information that can be merged from multiple sources.
- Determining varying data quality levels from multiple data sources and merging rules.
- Creating data pipelines to facilitate data aggregation and merging.

Data Cleansing Steps

Data purification is required for training machine learning models. When cleaning data, take these steps:

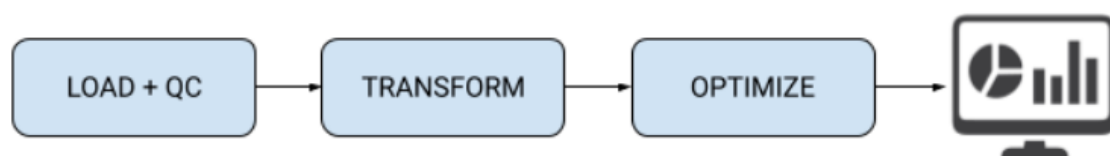
- **Formatting:** Standardizing data formats (data types, fields, matched formats, currency or metric conversions, etc.)
- **Replacing incorrect data:** During the cleaning step, erroneous information is removed.
- **Enhancing / Augmenting Data:**
 - o Add precalculated dimensions. Gather essential data.
 - o Add third-party data.
 - o "Multiply" image-based training data sets.
- **Removing extraneous information and de-duplication:** To enhance training, remove irrelevant data. Pixelating pictures.
- **Noise reduction and Disambiguation:** Reduce information, visual, and aural noise.
- **Data anonymization:** Remove PII before feeding models.
- **Normalizing Data:** Standardize data across areas to improve training.
- **Data sampling:** Extract a large representative subsample for ML training from huge data sources.

Data Sampling and Selection

- Two types of data filtering are required for machine learning. ML-related information. First, Data Sampling, especially for large Big data. Data Attribute trimming reduces size and complexity.
- **Data Sampling:** For huge data sets, extract a representative subsample for ML training. You don't require a Terabyte, Petabyte, or larger dataset for training. You should choose a representative, large sample that is data-balanced and free of unintended or purposeful informational biases.
- **Data Attribute Pruning:** You don't require every record's data field, attribute, or metadata. Remove extraneous data that bloats the data set and hampers ML model training.

CDP Airflow

Take a typical BI reporting use case. Users must consume data, convert it using quality checks, and improve visual analytics tool querying.



Each stage may be a separate task, enabling modular development. Data loading and quality checks may be combined into one Spark operation. The optimize phase might use Hive's materialized view to speed up BI reporting. Airflow's modular design lets us maximize the CDP platform. We created unique Airflow operators to exploit CDP's CDE and CDW analytical skills.

Users may create Airflow pipelines in their IDE of choice using a simple python setup file and then submit them to a DE Virtual Cluster using the same APIs via CLI, REST, or UI (VC). Users may execute Hive tasks in CDW or Spark jobs in CDE utilizing two CDP operators. Using the new CDP operators

only a few basic setup lines that abstract away traditional complications like security while offering critical schedule management features like retries, SLA, and alarms.

HadoopExam.com

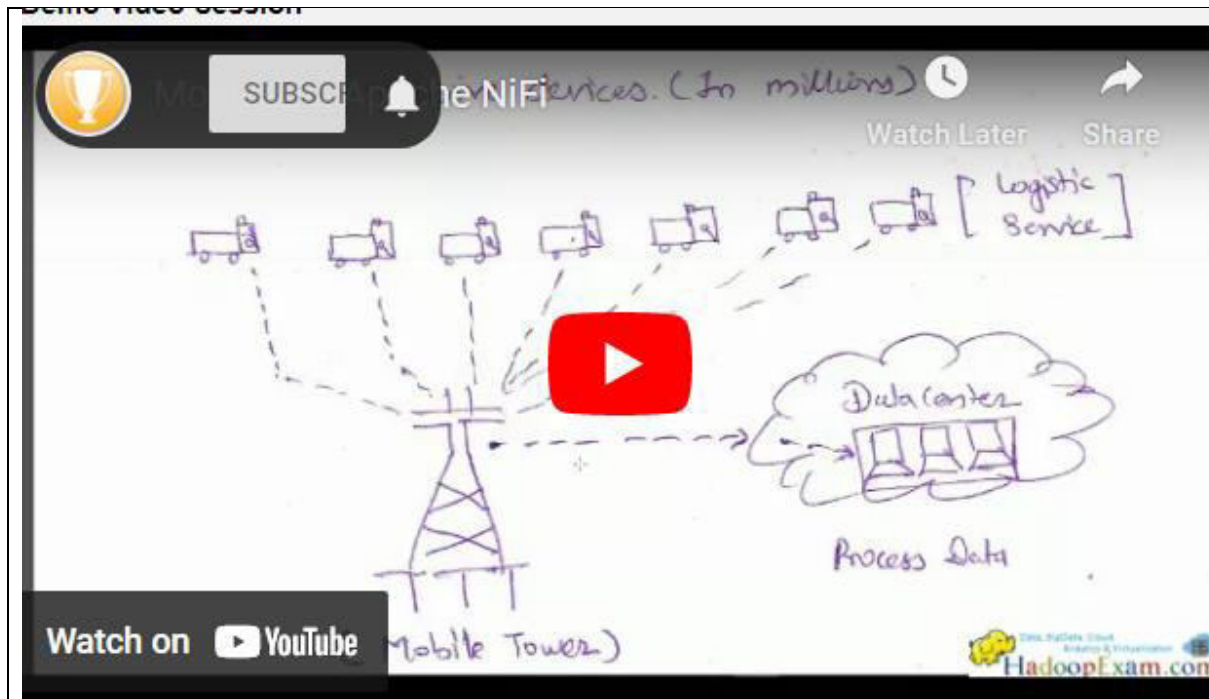
Contents

Chapter-26: Cloudera DataFlow	1
Overview	1
Cloudera DataFlow Key characteristics.....	3
DataFlow Cloudera principles.....	4
Use Case.....	5

Chapter-26: Cloudera DataFlow

Overview

Watch Below Video



Cloudera DataFlow (CDF), previously Hortonworks DataFlow (HDF), ingests, curates, and analyses data for insights and actionable information. Cloudera DataFlow Deployments offers a cloud-native runtime for Apache NiFi flows on auto-scaling Kubernetes clusters with centralised monitoring and alerting. DataFlow Functions offers a cloud-native runtime to execute Apache NiFi flows as functions on AWS Lambda, Azure Functions, and Google Cloud Functions, targeting use cases that don't need always-running NiFi flows.

Cloudera DataFlow for the Public Cloud (CDF-PC) connects to any data source, processes it, and delivers it to any destination using Apache NiFi. Design streaming ingest pipelines that send edge data to Snowflake, Confluent, and Cloudera Data Platform (CDP).

CFM is a no-code Apache NiFi data intake and management solution. CFM provides scalable data transportation, transformation, and administration using NiFi's graphical interface and processors. CFM has two parts:

NiFi-Apache: The main data intake engine provides processors for connection, transformation, and content routing.

Apache NiFi Registry: NiFi's companion for DevOps-style flow file creation and deployment. It provides flow versioning, flow deployment, and flow promotion.

CDF-PC is a cloud-native universal data distribution service powered by Apache NiFi that lets you connect to any data source, analyse, and transmit data to any destination.

Cloudera DataFlow for the Public Cloud (CDF-PC) uses a two-tier architecture where product capabilities like the Dashboard, Catalog, and Environment management are hosted on the CDP Control Plane and flow deployments processing your data are provisioned in a CDP environment representing infrastructure in your cloud provider account. Find out how CDF-service PC's architecture helps users reach their objectives.

DataFlow establishes a Kubernetes cluster, Kubernetes Operators, and the DataFlow workload application in your cloud account when you activate DataFlow for one of your registered CDP environments. Users may install Flow Definitions once DataFlow is enabled. Flow Definition deployment establishes a specialised NiFi cluster on Kubernetes so NiFi flows may be treated as independent deployments.

Flow installations handle cloud data using NiFi flow logic. Flow deployment data doesn't cross the CDP Control Plane. Flow installations transmit heartbeats carrying health and performance data to the Control Plane.

DataFlow Functions lets you deploy DataFlow Catalog NiFi flows as AWS Lambda, Azure Functions, or Google Cloud Functions. DataFlow Functions don't need CDP DataFlow. DataFlow functions communicate with the Control Plane to obtain flow definitions and deliver monitoring data. CDF-PC lets you execute NiFi flows on Kubernetes clusters, enabling efficient scalability. In certain circumstances, DataFlow Functions are preferable. Apache NiFi offers a no-code UI for constructing and executing functions quickly using DataFlow Functions, a CDF-PC extension.

DataFlow Deployments offers a cloud-native runtime for Apache NiFi flows on Kubernetes. It enables centralised monitoring and alerting, improving developers' SDLC. DataFlow requires a Kubernetes cluster (or CDF environment) and at least one always-running virtual machine. Additional resources will be added based on your needs and flow use. TCO includes cloud provider and Cloudera charges. The cloud provider's expenses include operating Kubernetes via its native service and are based on virtual machines. Cloudera's prices are based on Concurrent User (CCU) pricing for operating flows, which is time-based.

DataFlow Functions is a cloud-native runtime for Apache NiFi flows on three serverless cloud providers (AWS Lambda, Azure Functions, and Google Cloud Functions). It's especially strong when

the flow doesn't need up-and-running NiFi resources. Use cases include event-driven object store processing, serverless microservices, IoT data processing, asynchronous API gateway request processing, batch file processing, and task automation using cron/timer scheduling. For these use situations, NiFi flows require a start and finish. A file landing in an object storage, a cron event starting, a gateway endpoint being called, etc. triggers the start. The cloud provider only allocated resources for the flow's trigger event processing period. The TCO includes the cloud provider's expenses based on the quantity and duration (in milliseconds) of resources supplied with function a and Cloudera's charges based on the number of function invocations and execution time if processing a single event takes more than one second.

Cloudera DataFlow Key characteristics

CDF-PC is a cloud-native universal data distribution service powered by Apache NiFi that lets you connect to any data source, analyse, and transmit data to any destination. See below for characteristics and functions.

Flow and resource isolation

CDF isolates data flows and guarantees resources for each without needing additional NiFi clusters. CDF generates an auto-scaling NiFi cluster for each flow deployment using shared Kubernetes resources. Flow deployments may grow independently, letting you isolate them and distribute resources as required. Flow isolation may guarantee resources for a particular data flow or separate failure zones.

Auto-scaling flow deployments

CDF scales Apache NiFi data flows. Flow deployments scale depending on CPU use within the deployment wizard's limits. CDF scales flow deployments by adding or deleting NiFi pods on the Kubernetes cluster as required and scaling the cluster within DataFlow-specified constraints.

ReadyFlows speed up flow deployment.

ReadyFlows may be easily deployed with little setup. ReadyFlows simplify typical data flow use cases.

DataFlow-powered serverless NiFi flows.

DataFlow Functions lets you deploy NiFi flows as serverless functions on AWS Lambda, Azure Functions, and Google Cloud Functions. DataFlow Functions addresses use cases that don't need continually running NiFi flows, enabling developers to concentrate more on business logic, and offers a serverless pay for value paradigm.

Central dashboard, KPIs.

One dashboard lets you monitor flow deployments across environments and cloud providers. Define KPI alerts for your flow deployments to monitor performance data.

Interconnectivity.

NiFi's broad processor library lets you connect to any data source or destination, including on-premise data sources, cloud data storage, cloud data warehouses, log data sources, and cloud data analytics services.

Role-based security.

By giving CDP people or groups preset roles like Flow Administrator or Flow User, you can determine who may activate the data service in an environment, make new flow deployments, or monitor current flow deployments.

Secure connectivity.

Any application may submit data to flow installations using easily provisioned safe, robust, and scalable endpoints.

Continuous deployment/integration (CD).

DataFlow is automated. Any UI operation may be automated using CLI. One CLI command deploys a new NiFi flow.

DataFlow Cloudera principles.

Describe flow.

A flow definition represents Apache NiFi data flow logic exported using the Download Flow Definition action on a process group or root canvas. Parameterization makes flow definitions transferable across development and production NiFi environments. To operate a NiFi data flow in CDF, export it as a flow specification and upload it.

Catalog.

In the CDF Catalog, you may import, version, and delete flow definitions. Initiate fresh deployments from the Catalog.

Deploying flow.

A flow deployment is a Kubernetes-based NiFi cluster executing a flow specification. When you start flow deployment using the CDF Catalog, a wizard lets you deploy a flow definition. Using the wizard, define your flow deployment's environment, configuration parameters, auto-scaling, and KPIs.

Deployment Manager

The Deployment Manager lets you change flow deployment parameters, size and scaling settings, KPIs, and alerts. It lets you update NiFi, access your flow deployments' NiFi canvas, and terminate them. To access Deployment Manager, click Manage Deployment under Deployment Details.

Function.

A function is a DataFlow Catalog flow that serverless cloud provider services may perform.

Environment.

CDP environments use CDF. CDP-supported environments may use DataFlow. The CDF enablement procedure develops Kubernetes infrastructure for each environment. Once DataFlow is enabled, you may deploy flow definitions.

ReadyFlow.

ReadyFlow is a preconfigured data flow that can be installed with a few settings.

Gallery ReadyFlow.

All ReadyFlows are in the Gallery. Add a ReadyFlow from the Gallery to the Catalog to establish a Flow Deployment.

KPI.

Apache NiFi monitors memory, CPU, data flow, and other system information using numerous metrics. KPIs represent NiFi measurements in Cloudera DataFlow. They provide a real-time perspective of data flow performance.

Dashboard.

The Dashboard displays all flow deployments across environments in CDF. The Deployment Details pane includes KPIs, system metrics, events, and alarms for each flow deployment.

Use Case

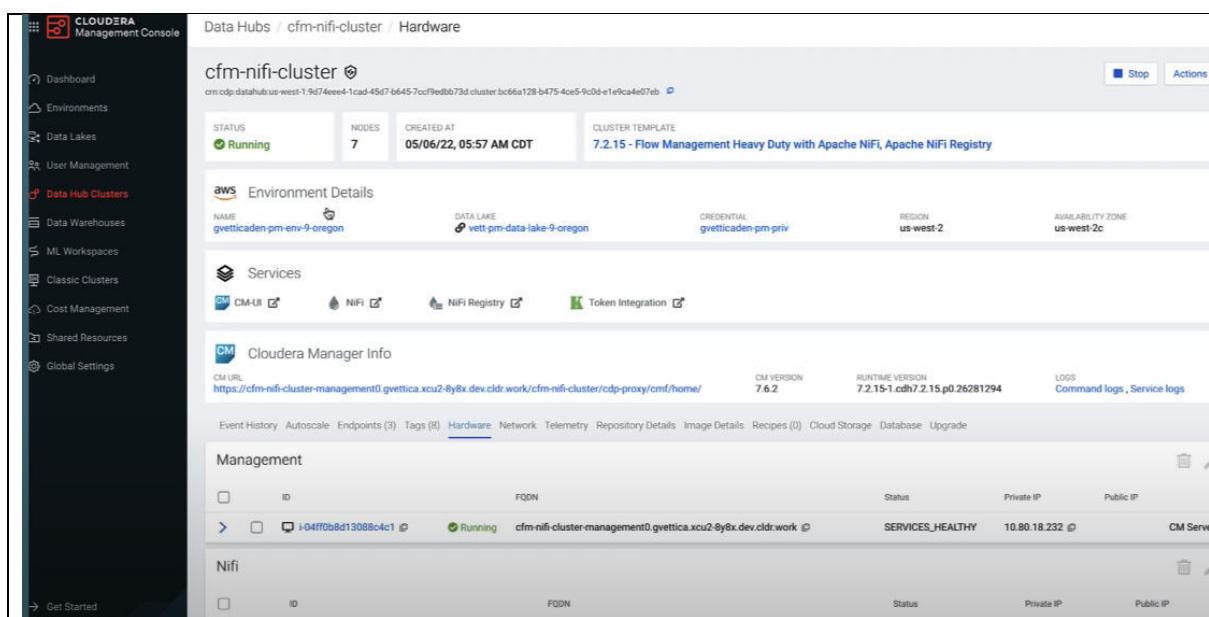
A CDF is multi-cloud universal data distribution service let's first set up the use case context a multinational retail company wants to collect data from point-of-sale systems

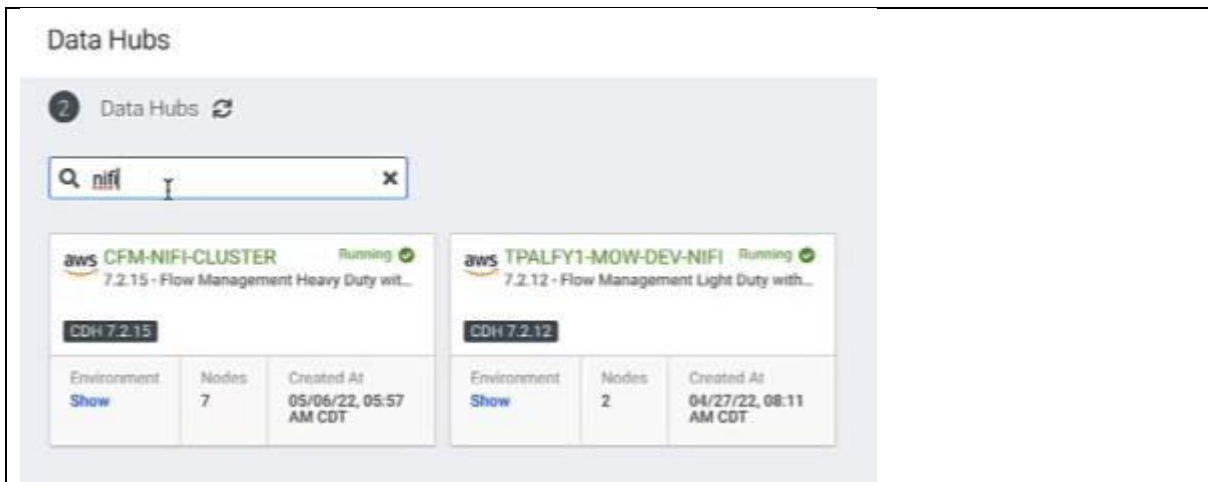
across the globe and distribute them to multiple cloud services they have six key requirements, which you can implement using CDF.

1. The company has thousands of point-of-sale systems and need a scalable way to collect data in streaming mode
2. Developers need an agile low code approach to developing edge data collection flows in different regions and easily deploy them to thousands of point-of-sale systems.
3. Data residency requirements the pos data and the processing of that data cannot occur outside the region of origination until that data has been redacted based on local geo rules.
4. These different geo rules require the use of different cloud providers and the need to be able to process data in different regions.
5. Given the distributive nature of the requirements centralized monitoring across regions and cloud providers is critical
6. Lastly requirement number six need the ability to deliver data to diverse destinations and services including cloud provider analytics services snowflake and kafka without requiring multiple point solutions.



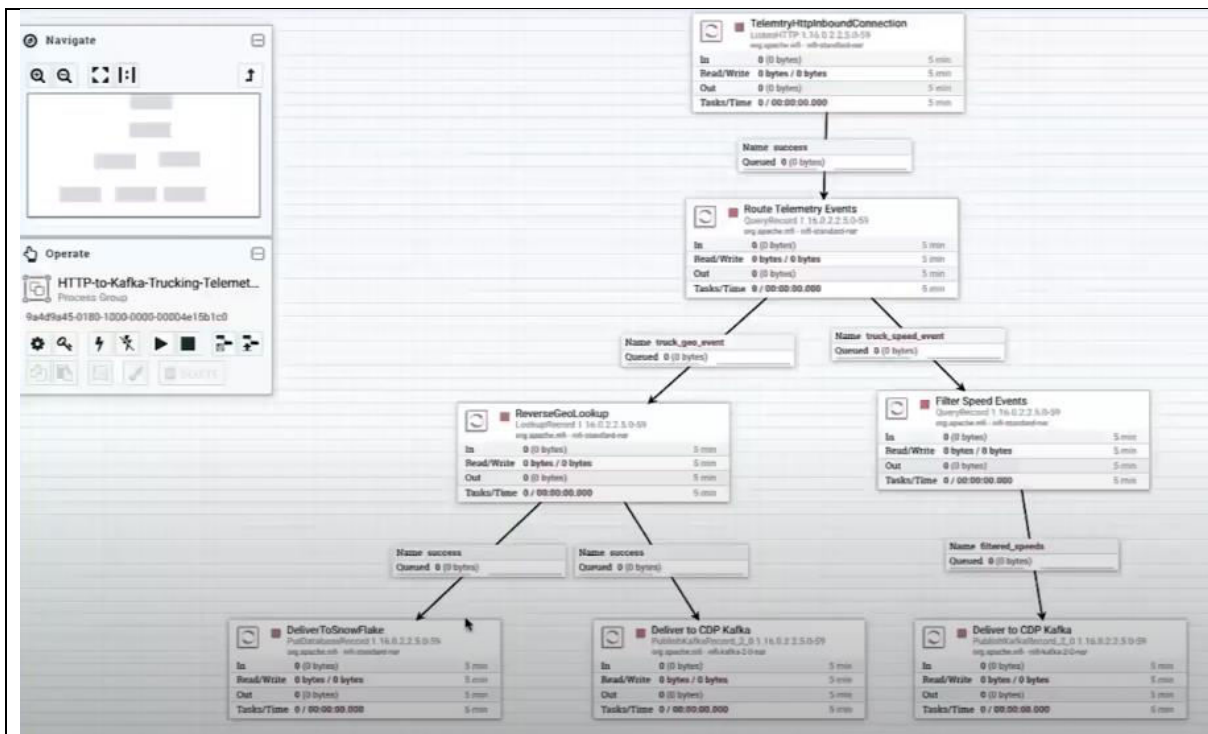
Data flow cloudera Start at cdp public cloud's homepage. The first step is to develop a nifi data distribution flow using the data hub service in cdp public cloud. Let's navigate to the nifi ui on the nifi canvas, as shown below. The start of the flow is the inbound gateway using the listen http processor that pos clients can send data to. Open up the options for the snowflake insert database record processor in the current cdf public cloud release and provided a new connection pooling controller service for snowflake. In the setup, the user id and password to snowflake are parameterized. The controller service will automatically download the snowflake driver and optimise the connection pool to make it simple to transmit data to snowflake. The flow also sends data to cdp kafka. As you can see, we're delivering data to parametrized topics. Once we've constructed the flow, we'll download it, put it into the catalogue, and deploy it on a containerized nifi cluster.





Let's go to the dataflow service before deploying the application we just studied. The ready complete gallery can link to cdp data sources like kudu or kafka, as well as third-party data providers like azure event hub confluent cloud and snowflake.

We won't use the ready for gallery but rather import the flow we developed in data hub into the catalogue. Let's give that flow a name, a description, and upload the flow that we downloaded earlier. Once the file is uploaded, let's import that into the catalogue. Now that it's in the catalogue, we can start the deployment process of actually deploying this onto a kubernetes nifi cluster.



This is just for your understanding and good enough for CDP-0011 certification exam.



Contents

Chapter-30: Local System Requirement for Private Cloud Base.....	1
Cloudera Manager Overview	1
Cloudera Runtime	1
CDP Private Cloud Base and Trial Version.....	1
CDP Private Cloud and Hardware	2
Operating System Requirement and setup.....	3

Chapter-30: Local System Requirement for Private Cloud Base

Cloudera Manager Overview

Production-use Cloudera Manager binaries need authentication. To access the binaries, you must have a valid subscription agreement, a licencing key file, and login credentials (username and password).

Cloudera sends new licensees an email with the licencing key file and login credentials. You may not have gotten an email if you have a CDP Private Cloud Base Edition licence. The licence key file contains the authentication credentials. Contact your account representative if you need a licence key.

Note authentication credentials. During installation, you may require them to configure a remote parcel repository or install Cloudera Manager packages using YUM, APT, or other tools. Use your login credentials to build the URL to the Cloudera Manager repository in the Cloudera Archive.

Cloudera Runtime

Production Cloudera Runtime packages need authentication. To access the packages below, you must have a subscription agreement, licence key file, and login credentials (username and password). Cloudera sends new licensees an email with the licencing key file and login credentials. You may not have gotten an email if you have a CDP Private Cloud Base Edition licence. The licence key file contains the authentication credentials. Contact your account representative if you need a licence key.

Use your account credentials to build the URL to the Cloudera Archive Runtime repository. During installation, Cloudera Manager may download Runtime parcels.

CDP Private Cloud Base and Trial Version

Try Cloudera Data Platform's CDP Private Cloud Base Edition for 60 days without a licence key. Visit the CDP Private Cloud Base Trial Download page, click Try Now, and follow the download

instructions. Installing CDP Private Cloud Base without a licence key performs a trial installation with an integrated PostgreSQL database that is not suited for a production setting. Trial installation documentation has further details. A licence may permanently allow a 60-day CDP Private Cloud Base Edition trial.

CDP Private Cloud and Hardware

As you build your cluster, you'll need to assign Cloudera Manager and Runtime responsibilities to hosts to optimise resources. Cloudera offers recommendations for assigning cluster host roles. Installation in a production environment is covered here.

- Installing: Before installing Cloudera Manager, Cloudera Runtime, and other managed services in production, examine the Cloudera Data Platform Private Cloud Base Requirements and Supported Versions.
- Cloudera Manager, Runtime, and Managed Services Installation: This method is recommended for production Cloudera Manager and Runtime installations. Installing CDP Private Cloud Base Trial is for non-production use.
- Configure Cloudera Manager's Repository: RHEL-compatible computers install Cloudera Manager using yum. Installing software requires access to repositories. Cloudera maintains Internet-accessible Runtime and Cloudera Manager repositories.
- Install JDK: You may install OpenJDK or Oracle JDK directly from Oracle on all hosts for CDP Private Cloud Base.
- Install Cloudera Manager Server: Install Cloudera Manager packages on Cloudera Manager Server.
- Installing and configuring databases: Cloudera Manager saves configuration information, system health, and job progress in databases and datastores.
- Configure Cloudera Manager's database: scm prepare database.sh may build and setup a database in Cloudera Manager Server.
- Install software and runtime: Start Cloudera Manager Server and log in to the Admin Console after setting up the database. Install using the wizard.
- Setup a cluster using the wizard: Add Cluster - Configuration wizard begins immediately after Add Cluster - Installation. Each wizard page is explained here.
- (Recommended) AUTO-TLS: Auto-TLS simplifies cluster TLS encryption management.
- Apache Ranger Extras: Apache Ranger must be installed after Cloudera Manager and a cluster.
- Ubuntu Knox: This guide explains how to install Apache Knox with CDP Private Cloud Base.
- HDFS data-at-rest encryption: This section explains how to encrypt HDFS data end-to-end. HA is supplied for best performance.
- Encrypting Cloudera Navigator: Learn about installing Navigator Encrypt, setting up TLS certificates on a client, and entropy requirements.
- Cloudera Key HSM installation: Cloudera Navigator Key HSM translates between the target HSM platform and Cloudera Navigator Key Trustee Server. Navigator Key HSM lets you utilise a Key Trustee Server to securely store and recover encryption keys and other secure items.
- Installing Ranger RMS translates Hive access controls to HDFS.

Operating System Requirement and setup

There are some requirements which needs to be followed for setting up the Operating System before installation of the “Cloudera Data Platform”. Let’s follow the below steps for the same.

- Create a User on the operating System which has following detail.
 - **Username:** hadoopexam1
 - **Password:** hadoopexam
- As the default user created is **hadoopexam** in my case. In your case it would be something else, this is usually the email Id which you have used.
- To create new user type below command.

```
sudo useradd hadoopexam1
```

- And we need to set the password for this new user(hadoopexam1). Hence, type the below command.

```
sudo passwd hadoopexam1
```

- Please provide the value of the password as “hadoopexam” (**Note:** Without quote). Your screen should look like something below.

```
[hadoopexam@hemain ~]$  
[hadoopexam@hemain ~]$  
[hadoopexam@hemain ~]$  
[hadoopexam@hemain ~]$ sudo useradd hadoopexam1  
[hadoopexam@hemain ~]$ sudo passwd hadoopexam1  
Changing password for user hadoopexam1.  
New password:  
BAD PASSWORD: The password contains the user name in some form  
Retype new password:  
passwd: all authentication tokens updated successfully.  
[hadoopexam@hemain ~]$
```

Type passwod
hadoopexam here

Now you have "hadoopexam1" user and "hadoopexam" as password.

- Now create an entry in the “/etc/sudoers” file. This file is used to assign System rights to the System users. This is more of permission file, what a user can do.

```
sudo vi /etc/sudoers
```

Add the below line

```
hadoopexam1  ALL=(ALL)  NOPASSWD: ALL
```

```

# Prior to version 1.8.15, groups listed in sudoers that were not
# found in the system group database were passed to the group
# plugin, if any. Starting with 1.8.15, only groups of the form
# %:group are resolved via the group plugin by default.
# We enable always_query_group_plugin to restore old behavior.
# Disable this option for new behavior.
Defaults    always_query_group_plugin

Defaults    env_reset
Defaults    env_keep = "COLORS DISPLAY HOSTNAME HISTSIZE KDEDIR LS_COLORS"
Defaults    env_keep += "MAIL PS1 PS2 QTDIR USERNAME LANG LC_ADDRESS LC_CTYPE"
Defaults    env_keep += "LC_COLLATE LC_IDENTIFICATION LC_MEASUREMENT LC_MESSAGES"
Defaults    env_keep += "LC_MONETARY LC_NAME LC_NUMERIC LC_PAPER LC_TELEPHONE"
Defaults    env_keep += "LC_TIME LC_ALL LANGUAGE LINGUAS _XKB_CHARSET XAUTHORITY"

#
# Adding HOME to env_keep may enable a user to run unrestricted
# commands via sudo.
#
# Defaults    env_keep += "HOME"

Defaults    secure_path = /sbin:/bin:/usr/sbin:/usr/bin

## Next comes the main part: which users can run what software on
## which machines (the sudoers file can be shared between multiple
## systems).
## Syntax:
##
##     user    MACHINE=COMMANDS
##
## The COMMANDS section may have other options added to it.
##
## Allow root to run any commands anywhere
root    ALL=(ALL)    ALL

## Allows members of the 'sys' group to run networking, software,
## service management apps and more.
# %sys ALL = NETWORKING, SOFTWARE, SERVICES, STORAGE, DELEGATING, PROCESSES, LOCATE, DRIVERS

## Allows people in group wheel to run all commands
%wheel  ALL=(ALL)    ALL

## Same thing without a password
# %wheel    ALL=(ALL)    NOPASSWD: ALL
hadoopexam1    ALL=(ALL)    NOPASSWD: ALL

## Allows members of the users group to mount and unmount the
## cdrom as root
# %users    ALL=/sbin/mount /mnt/cdrom, /sbin/umount /mnt/cdrom

```

- Now configure the SSH Password. As we need to SSH most of the time. And there are multiple ways to do the SSH like SSH Key based authentication, username and password-based authentication. We wanted to use the Password based authentication to login on this instance using the “**PasswordAuthentication**” mechanism.

```
sudo vi /etc/ssh/sshd_config
```

```

[hadoopexam@hemain ~]$ sudo vi /etc/ssh/sshd_config
[hadoopexam@hemain ~]$

```

Now set the value as below. This key-value is already present in the file. We just need to uncomment it and save the file.

```

#LoginGraceTime 2m
PermitRootLogin no
#StrictModes yes
#MaxAuthTries 6
#MaxSessions 10

#PubkeyAuthentication yes

# The default is to check both .ssh/authorized_keys and
# but this is overridden so installations will only check
AuthorizedKeysFile      .ssh/authorized_keys

#AuthorizedPrincipalsFile none

#AuthorizedKeysCommand none
#AuthorizedKeysCommandUser nobody

# For this to work you will also need host keys in /etc
#HostbasedAuthentication no
# Change to yes if you don't trust ~/.ssh/known_hosts
# HostbasedAuthentication
#IgnoreUserKnownHosts no
# Don't read the user's ~/.rhosts and ~/.shosts files
#IgnoreRhosts yes

# To disable tunneled clear text passwords, change to
PasswordAuthentication yes
#PermitEmptyPasswords no
PasswordAuthentication no

# Change to no to disable s/key passwords
#ChallengeResponseAuthentication yes
ChallengeResponseAuthentication no

# Kerberos options
#KerberosAuthentication no
#KerberosOrLocalPasswd yes
#KerberosTicketCleanup yes
#KerberosGetAFSToken no
#KerberosUseKuserok yes

# GSSAPI options
GSSAPIAuthentication yes
GSSAPICleanupCredentials no
#GSSAPIStrictAcceptorCheck yes
#GSSAPIKeyExchange no
#GSSAPIEnablek5users no

# Set this to 'yes' to enable PAM authentication, account
# and session processing. If this is enabled, PAM auth
# will be allowed through the ChallengeResponseAuthentication

```

- To be effective this setting we need to restart the ssh service using below command.

```
sudo systemctl restart sshd.service
```

```
[hadoopexam@hemain ~]$  
[hadoopexam@hemain ~]$ sudo systemctl restart sshd.service  
[hadoopexam@hemain ~]$
```

- Now Disable the Security Enhanced Linux aka SELinux". This is again one of the ways to controlling the access on the Linux Operating System. However, we wanted this to be disabled for CDP installation. Use the below command to open the file.

```
sudo vi /etc/sysconfig/selinux
```

And set the key-value as "SELINUX=disabled". Once done save the file.

```
# This file controls the state of SELinux on the system.  
# SELINUX= can take one of these three values:  
#   enforcing - SELinux security policy is enforced.  
#   permissive - SELinux prints warnings instead of enforcing.  
#   disabled - No SELinux policy is loaded.  
SELINUX=disabled  
# SELINUXTYPE= can take one of three values:  
#   targeted - Targeted processes are protected,  
#   minimum - Modification of targeted policy. Only selected processes are protected.  
#   mls - Multi Level Security protection.  
SELINUXTYPE=targeted
```

- Now we need to set the hostname for this instance as below. However, this command is effect is only temporary. As soon as you restart this VM instance this setting would be lost.

You can remove --static argument

```
sudo hostnamectl set-hostname main.hadoopexam1.com --static  
hostname -f
```

```
[hadoopexam@hemain ~]$ sudo hostnamectl set-hostname main.hadoopexam1.com --static  
[hadoopexam@hemain ~]$ hostname -f  
main.hadoopexam1.com  
[hadoopexam@hemain ~]$
```

- To make this permanent, we need to run some script as soon as instance started. For that we can create a new script file named "usr/sbin/heonstart.sh" and add the below scripting code in it. This would few things as mentioned in the script comments.

```
#Create a custom script file, which can be executed when host starts  
sudo touch /usr/sbin/heonstart.sh  
sudo chmod 755 /usr/sbin/heonstart.sh  
sudo vi /usr/sbin/heonstart.sh
```

```
[hadoopexam@hemain ~]$ sudo touch /usr/sbin/heonstart.sh  
[hadoopexam@hemain ~]$ sudo chmod 755 /usr/sbin/heonstart.sh  
[hadoopexam@hemain ~]$ sudo vi /usr/sbin/heonstart.sh
```

- Now update the “/etc/rc.d/rc.local” for running this script on the startup.

```
#Lets make rc.local file as an executable
sudo chmod +x /etc/rc.d/rc.local
```

```
sudo systemctl enable rc-local
sudo systemctl start rc-local
```

```
[hadoopexam@hemain ~]$ sudo chmod +x /etc/rc.d/rc.local
[hadoopexam@hemain ~]$
[hadoopexam@hemain ~]$
[hadoopexam@hemain ~]$ sudo systemctl enable rc-local
[hadoopexam@hemain ~]$ sudo systemctl start rc-local
```

```
sudo vi /etc/rc.d/rc.local
```

```
[hadoopexam@hemain ~]$
[hadoopexam@hemain ~]$ sudo vi /etc/rc.d/rc.local
```

Now add the below line in this file.

```
sudo sh /usr/sbin/heonstart.sh
```

```
#!/bin/bash
# THIS FILE IS ADDED FOR COMPATIBILITY PURPOSES
#
# It is highly advisable to create own systemd services or udev rules
# to run scripts during boot instead of using this file.
#
# In contrast to previous versions due to parallel execution during boot
# this script will NOT be run after all other services.
#
# Please note that you must run 'chmod +x /etc/rc.d/rc.local' to ensure
# that this script will be executed during boot.

touch /var/lock/subsys/local
sudo sh /usr/sbin/heonstart.sh
```

- Update Network Configuration file.

```
sudo vi /etc/sysconfig/network
```

```
[hadoopexam@hemain ~]$
[hadoopexam@hemain ~]$ sudo vi /etc/sysconfig/network
```

And add the below lines in the file.

```
NETWORKING=yes
HOSTNAME=main.hadoopexam1.com
```

```
# Created by anaconda
NETWORKING=yes
HOSTNAME=main.hadoopexam1.com
```

- Now disable the firewall.

First, we need to create a firewall.rules file as below. We are doing by copying existing iptables.

```
sudo iptables-save > ~/firewall.rules
```

```
[hadoopexam@hemain ~]$ sudo iptables-save > ~/firewall.rules
[hadoopexam@hemain ~]$
```

Now using the below command, disable the firewall.

```
sudo systemctl disable firewalld
sudo systemctl stop firewalld
```

```
[hadoopexam@hemain ~]$ sudo systemctl disable firewalld
Removed symlink /etc/systemd/system/multi-user.target.wants/firewalld.service.
Removed symlink /etc/systemd/system/dbus-org.fedoraproject.FirewallD1.service.
[hadoopexam@hemain ~]$ sudo systemctl stop firewalld
[hadoopexam@hemain ~]$
```

- Configure the swappiness.

Swapping is a technique where data in Random Access Memory (RAM) is written to a special location on your hard disk—either a swap partition or a swap file—to free up RAM.

```
sudo vi /etc/sysctl.conf
```

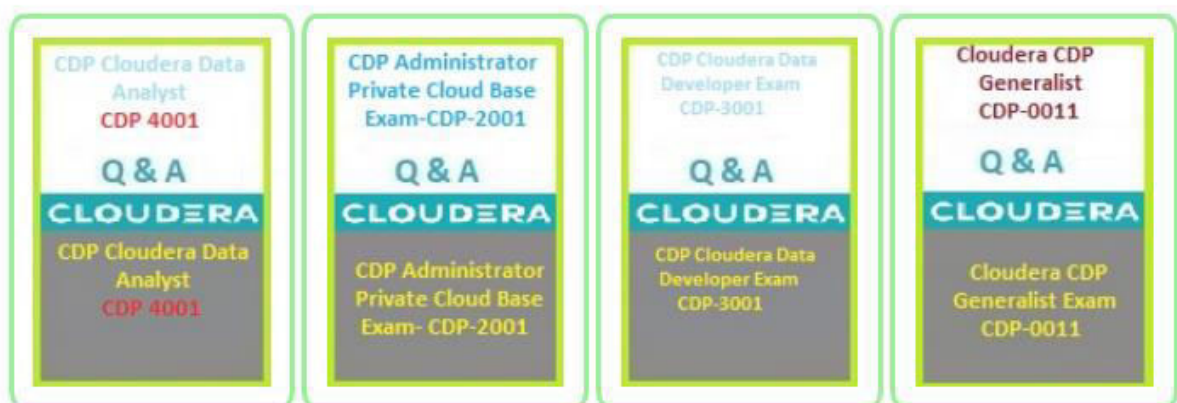
```
[hadoopexam@hemain ~]$
[hadoopexam@hemain ~]$ sudo vi /etc/sysctl.conf
```

Now add the following line in this file.

```
vm.swappiness=0
```

```
ssh.cloud.google.com/projects/moonlit-web-271805/zones/asia-south1-c/
# sysctl settings are defined through files in
# /usr/lib/sysctl.d/, /run/sysctl.d/, and /etc/sysctl.d/.
#
# Vendors settings live in /usr/lib/sysctl.d/.
# To override a whole file, create a new file with the same in
# /etc/sysctl.d/ and put new settings there. To override
# only specific settings, add a file with a lexically later
# name in /etc/sysctl.d/ and put new settings there.
#
# For more information, see sysctl.conf(5) and sysctl.d(5).
vm.swappiness=0
```

[Get Full Version Contents from this link](#)



Contents

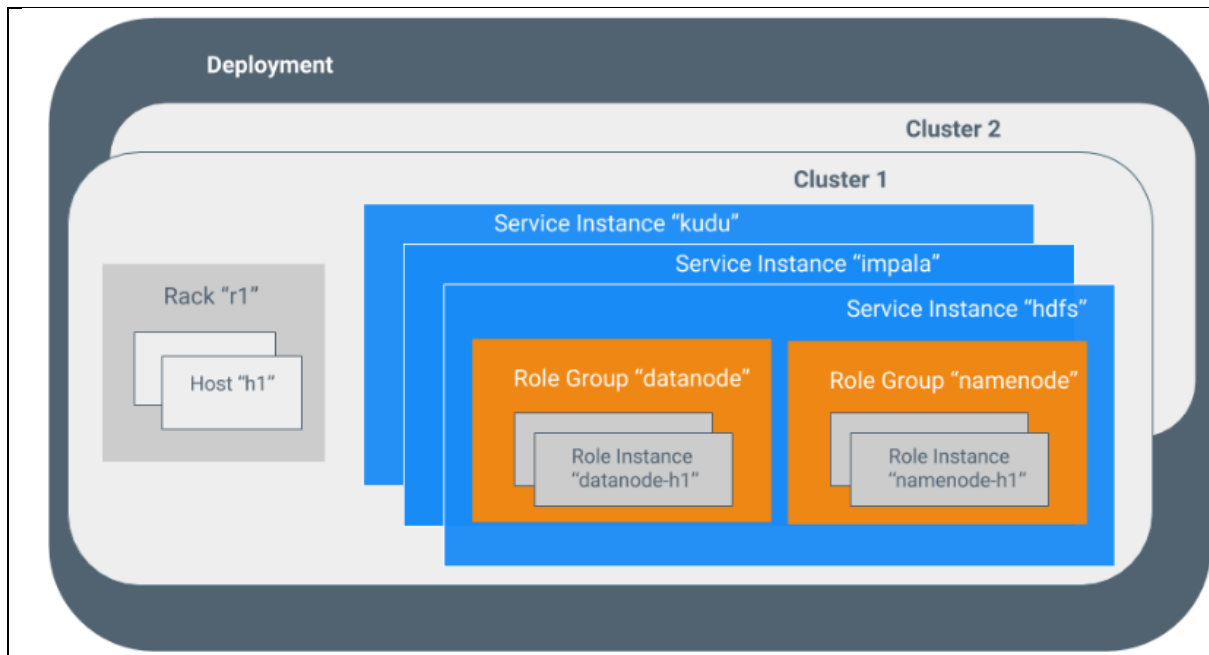
Chapter-31: Cloudera Manager	1
Overview	1
Cloudera Manager Architecture	3
Cloudera Manager and Heartbeating	4
Cloudera Manager and State management.....	4
Cloudera Manager and Distributing Software	5
Cloudera Manager and Process Management	5
Cloudera Manager and Host Management	5
Cloudera Manager Agents	6
Cloudera Manager and Resource Management.....	6
Cloudera Manager and User Management	7
Cloudera Manager and Security Management	7
Cloudera Manager and Monitoring	8
Cloudera Manager and Health Test	9
Cloudera Manager and Metric Collection and Display	9
Cloudera Manager Alerts and triggers.....	10

Chapter-31: Cloudera Manager

Overview

Cloudera Manager manages CDP clusters and Cloudera Runtime services. The Cloudera Manager server controls one or more clusters using Cloudera Manager Agents on each cluster host. The Cloudera Manager Admin Console manages CDP Private Cloud Base installations. You may start and stop the cluster and individual services, configure and add new services, manage security, and update the cluster using Cloudera Manager Admin Console. You may programmatically manage Cloudera using the API. Cloudera Manager is separate from CDP Versions.

Cloudera Manager terminology. Cloudera Manager language must be understood before usage. The words are related and defined below.



Sometimes service and role relate to both types and instances, which is confusing. Cloudera Manager and this section interchange type and instance. Home > Status and Clusters > ClusterName list service instances. This is comparable to computer languages, where "string" might refer to a type or an instance ("hi there"). "Type" is used to indicate a type, while "instance" indicates an instance.

Deployment: Cloudera Manager's settings and clusters.

Resource pool: A named configuration of resources and a strategy for scheduling them in Cloudera Manager.

Cluster: A group of computers or racks that conduct MapReduce and other algorithms on HDFS data. In Cloudera Manager, a logical entity that includes hosts, Cloudera Runtime, and service and role instances. One cluster per host. Each cluster may only be linked with one Cloudera Manager Server. Physical or virtual computer that executes role instances in Cloudera Manager. One cluster per host.

Rack: In Cloudera Manager, a collection of physical hosts serviced by the same switch.

Service: Linux tool that executes a System V init script in /etc/init.d/ in a predictable environment, eliminating most environment variables and setting the working directory to /. A Cloudera Manager category of distributed or non-distributed cluster-running capability. Service kind. Hive, HDFS, YARN, and Spark.

Instance service: In Cloudera Manager, a cluster-running service. "HDFS-1" and "yarn" A service instance spans roles.

Role: A Cloudera Manager service category. NameNode, SecondaryNameNode, DataNode, and Balancer are HDFS roles. Role kind.

Instance: A role instance in Cloudera Manager. It's a Unix process. "NameNode-h1" and "DataNode-h1"

Grouping: In Cloudera Manager, role instance configuration properties.

Host-template: Cloudera Manager roles. Each role group's instance is generated and assigned to a host when a template is deployed.

Gateway: A position that grants access to cluster services. HDFS, Hive, Kafka, MapReduce, Solr, and Spark each have gateway roles for their clients. Gateway roles aren't necessarily called "gateway" or just for client access. Hue Kerberos Ticket Renewer proxies Kerberos tickets.

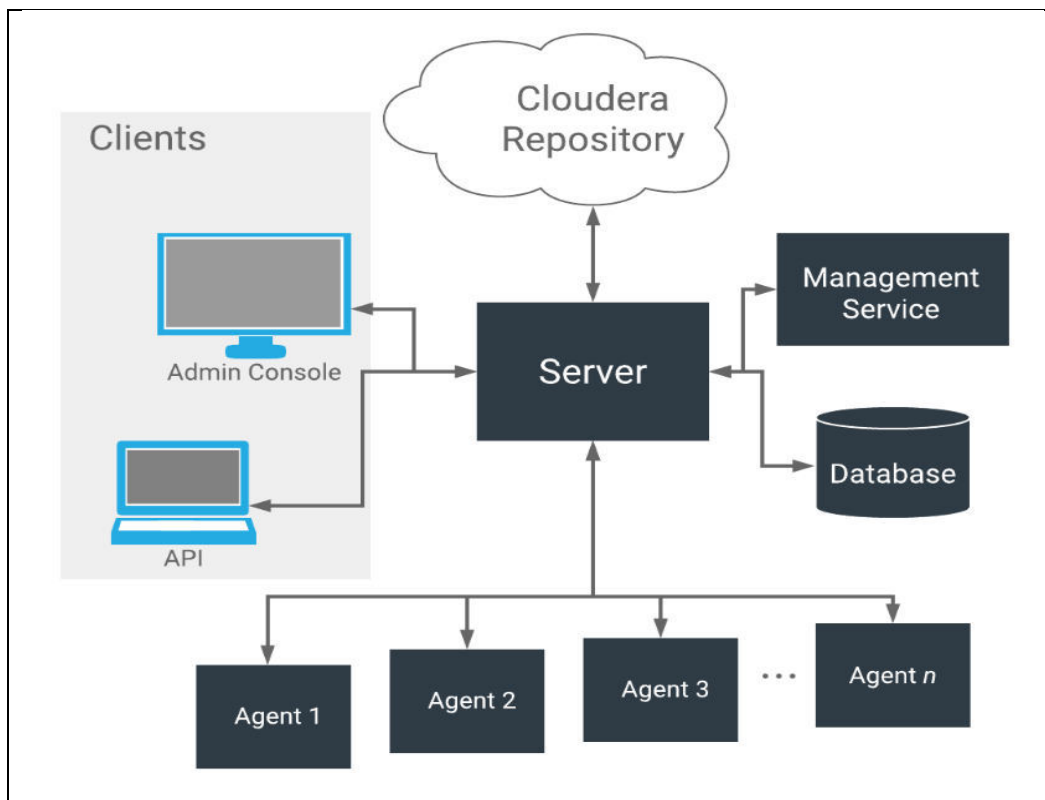
Gateway nodes or edge nodes provide one or more gateway functions in network or cloud environments. When Deploy Client Configuration is chosen in Cloudera Manager Admin Console, the gateway nodes get the relevant client configuration files.

Parcel: Binary distribution format including compiled code and meta-information like package description, version, and dependencies.

Dynamic pool: Static partitioning of CPU, memory, and I/O weight among services in Cloudera Manager.

Cloudera Manager Architecture

Cloudera Manager Server is at its core. The Server hosts the Cloudera Manager Admin Console, API, and application logic and installs, configures, starts, and stops services and manages the cluster.



Cloudera Manager Server interacts with other parts.

- Every host has agent. The agent starts and stops processes, unpacks configurations, and triggers installs.
- Management Service – a collection of roles that monitor, alert, and report.
- Database contains monitoring and configuration data. One or more database servers operate numerous logical databases. Cloudera Manager Server and monitoring utilise independent logical databases.
- Cloudera Repository is Cloudera Manager's software repository.
- Clients are server interfaces.
 - Administrators utilise Cloudera Manager Admin Console to manage clusters and Cloudera Manager.
 - Developers utilise Cloudera Manager API to construct bespoke apps.

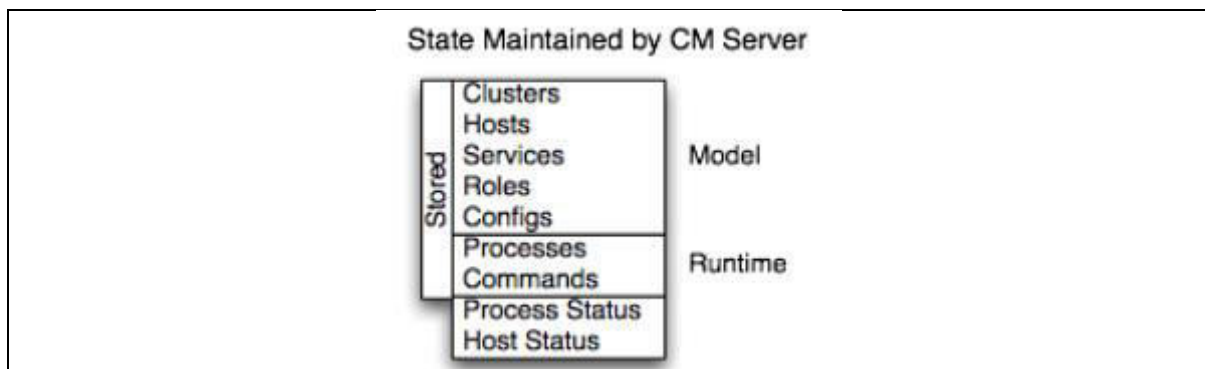
Cloudera Manager and Heartbeating

Cloudera Manager uses heartbeats for communication. Agents transmit heartbeats to Cloudera Manager Server every 15 seconds. State changes increase frequency to decrease user latency.

Agent tells Cloudera Manager Server of its activity during heartbeat exchange. Cloudera Manager Server replies with Agent actions. Agent and Cloudera Manager Server reconcile. If you start a service, the Agent tries to start the required processes; if a process fails to start, Cloudera Manager Server classifies the start command as unsuccessful.

Cloudera Manager and State management

Cloudera Manager Server manages cluster state. This state is recorded in Cloudera Manager Server's database as "model" and "runtime." Cloudera Manager represents cluster roles, settings, and interdependencies. Model state captures what runs where and how. Model state captures that a cluster has 17 hosts that each operate a DataNode. Cloudera Manager Admin Console settings screens, API, and "Add Service" interact with the model.



Runtime state is what processes are executing where and what actions (such as rebalance HDFS or perform a Backup/Disaster Recovery schedule) are running. Runtime state contains process configuration files. When you choose Start in the Cloudera Manager Admin Console, the server collects all necessary service and role settings, verifies it, produces configuration files, and saves them in the database.

When you edit a configuration, you update the model state. If Hue is running, it uses the old port. When this happens, the job is considered "outdated." Restarting the role resynchronizes (which triggers the configuration re-generation and process restart).

Cloudera Manager models most settings, but some need specific management. Cloudera Manager allows you add attributes directly to configuration files to work around bugs or investigate unsupported alternatives.

Cloudera Manager and Distributing Software

Cloudera Manager installs Cloudera Runtime and other managed services. Cloudera Manager supports packages and parcels.

A package is a binary distribution format that comprises compiled code, meta-information, and dependencies. Package management systems use meta-information to facilitate package searches, execute upgrades, and fulfil dependencies. Cloudera Manager installs and upgrades using each OS's native package manager. A parcel is a binary format that contains Cloudera Manager's application files and information.

Cloudera Manager and Process Management

Cloudera Manager starts/stops processes: You can only start or terminate role instance processes in a Cloudera Manager-managed cluster. Cloudera Manager employs supervisor, an open source process management tool, to start processes, reroute log files, inform of process failure, and set the caller process's effective user ID. Cloudera Manager restarts crashed processes. If a role instance's process crashes regularly after startup, it will be flagged as unhealthy. Stopping the Cloudera Manager Server and Agents won't affect running role instances.

Start-up init.d starts the Agent. It queries Cloudera Manager Server to decide which processes to launch. Cloudera Manager monitors the Agent's host. If the Agent stops beating, the host is considered sick. Agents initiate and stop processes. When the Agent discovers a new process from the Server heartbeat, it unpacks the configuration in /var/run/cloudera-scm-agent. It contacts supervisor to begin. Cloudera Manager processes never travel alone, as these activities show. A process involves more than simply exec() parameters; it also includes configuration files, directories, and other information.

Cloudera Manager and Host Management

Cloudera Manager manages hosts in clusters. When you initially launch Cloudera Manager Admin Console, you may search for hosts to add to the cluster and map host responsibilities. Cloudera Manager delivers JDK, Cloudera Manager Agent, Impala, Solr, etc. to managed hosts automatically.

Once the services are installed and operating, the Admin Console's Hosts tab reveals the managed hosts' status. The information comprises the host's Cloudera Runtime version, cluster, and number of roles. Cloudera Manager manages host lifecycles and adds/deletes hosts. Cloudera Management Service Host Monitor runs health checks and gathers host metrics to monitor host health and performance.

Cloudera Manager Agents

Cloudera Manager Agent manages role instance processes with Cloudera Manager Server. You can only start or terminate role instance processes in a Cloudera Manager-managed cluster. Cloudera Manager employs supervisord, an open source process management tool, to start processes, reroute log files, inform of process failure, and set the caller process's effective user ID. Cloudera Manager restarts crashed processes. If a role instance's process crashes regularly after startup, it will be flagged as unhealthy.

Start-up init.d starts the Agent. It queries Cloudera Manager Server to decide which processes to launch. Cloudera Manager monitors the Agent's host. If the Agent stops beating, the host is considered sick.

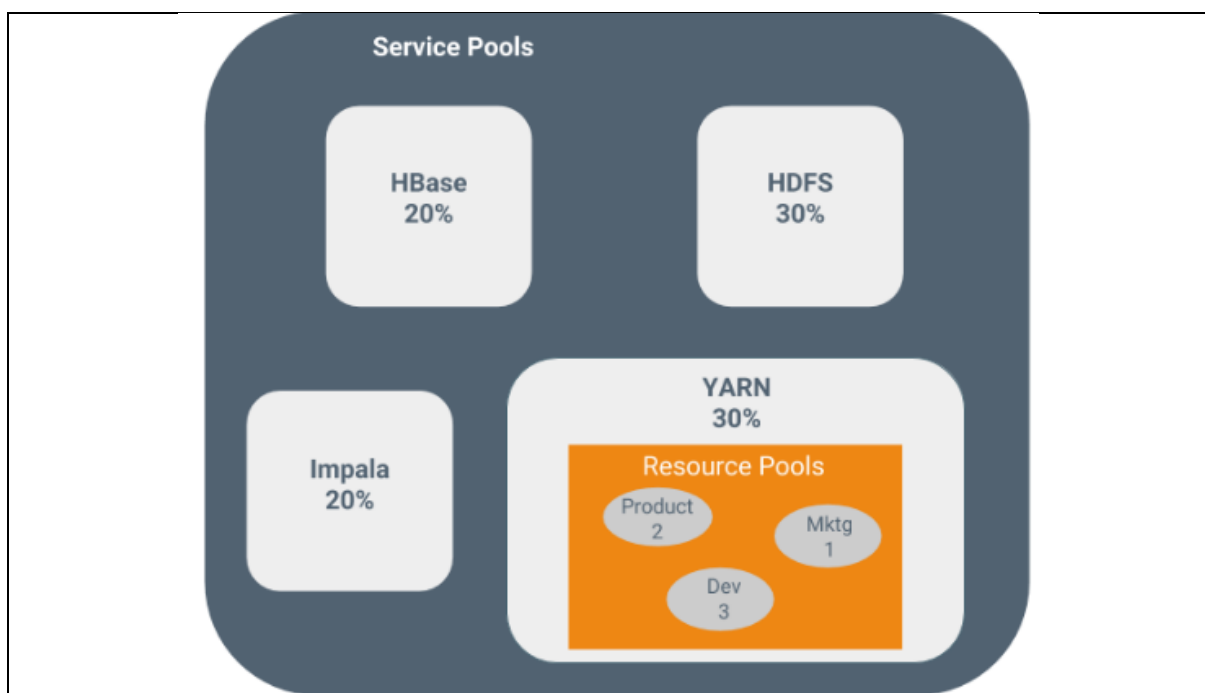
Agents initiate and stop processes. When the Agent discovers a new process from the Server heartbeat, it unpacks the configuration in `/var/run/cloudera-scm-agent`. It contacts supervisord to begin.

Cloudera Manager and Resource Management

Resource management helps forecast cluster behaviour by describing service impacts.

- Guarantee timely completion of key tasks using resource management.
- Support equitable cluster scheduling amongst user groups.
- Prevent users from blocking cluster access.

A static service pool wizard may statically allocate resources using cgroups. The wizard configures cgroups based on your service allocation. The next graphic shows static pools for HBase, HDFS, Impala, and YARN services with 20%, 30%, 20%, and 30% of cluster resources.



Using dynamic resource pools, you may dynamically apportion YARN and Impala resources. Depending on your Cloudera Runtime version, Cloudera Manager's dynamic resource pools enable the following scenarios:

- **YARN** controls virtual cores, memory, running applications, undeclared child pool resources, and pool scheduling policies. In the figure, three dynamic resource pools—Dev, Product, and Mktg—are established for YARN. If an application is allocated to the Product pool while other apps use the Dev and Mktg pools, the Product pool gets 10% of the total cluster resources. YARN Product pool gets 30% of cluster resources if neither Dev nor Mktg pools are in use.
- **Impala** maintains memory for query pools and limits ongoing and pending queries.

Cloudera Manager and User Management

User accounts control Cloudera Manager functionalities. A user account defines how a user is authenticated and their rights. Cloudera Manager has many authentication methods. Cloudera Manager can authenticate users against its database or an external service. The external authentication service might be an LDAP server or another external service. Cloudera Manager supports SAML single sign-on.

Cloudera Manage and Security Management

Authentication: Users and services must authenticate their identity to access a system resource via authentication. Organizations handle user identification and authentication using time-tested technologies, such as LDAP for identity, directory, and group management, and Kerberos for authentication.

Cloudera supports both technologies. Organizations having existing LDAP directory services such as Active Directory (included in Microsoft Windows Server's Active Directory Services) may utilise existing user accounts and group listings instead of establishing new accounts across the cluster. Cloudera Navigator's user role authorisation method requires Active Directory or OpenLDAP.

Cloudera supports Kerberos and Active Directory authentication. Active Directory supports Kerberos in addition to LDAP for identity management and directory capabilities.

Both systems exist. Microsoft Active Directory is an LDAP directory service that offers Kerberos authentication, and Kerberos credentials may be stored and managed in LDAP. Kerberos authentication may be used by Cloudera Manager Server, cluster nodes, and Apache Hive, Hue, and Impala.

Authorization: Who or what has access to a resource or service is authorised. Since Hadoop combines the capabilities of numerous, previously distinct IT systems as an enterprise data centre, it needs different permission rules with differing granularity. Hadoop administration solutions ease setup and maintenance by tying all users to existing LDAP or AD groups. Role-based access control for batch and interactive SQL queries. Cloudera Runtime's access control includes:

- Traditional Each directory and file has a single owner and group with POSIX-style permissions. Each assignment has a basic set of permissions: read, write, and execute for files, and access to child directories for directories.
- HDFS ACLs let you establish various permissions for individual users or groups.
- Apache HBase employs ACLs to permit column, column family, and column family qualifier activities. Users and groups may get HBase ACLs.
- Apache Ranger's role-based access control.

Encryption: At several cluster technology tiers, data at rest and in transit are encrypted.

- **Application Encryption:** HDFS Transparent Encryption allows you encrypt HDFS folders. Cloudera recommends utilising Key Trustee Server with HDFS encryption to store encryption keys. CDP components (Impala, MapReduce, YARN, or HBase) may encrypt data stored temporarily outside HDFS.
- **Operating System Encryption:** Encryption may be applied to a whole volume under Linux.
- **Network Level Encryption:** Using TLS/SSL, HTTP, RPC, or TCP/IP client-server connections may be encrypted.

Cloudera Manager and Monitoring

Cloudera Manager monitors the health and performance of cluster components (hosts, service daemons) and task performance and resource needs.

Cloudera Manager monitors:

- **Monitoring Cloudera Runtime** Services explains how to observe service and role instance health testing. Charts showing measurements assist diagnose problems. Health tests suggest what to do if a component's health deteriorates. You may inspect a service or role's activity history and a configuration audit log.
- **Monitoring Hosts** shows how to examine information about all the hosts on your cluster: which hosts are up or down, current resident and virtual memory usage for a host, what role instances are executing on a host, etc. You may examine a summary of all hosts in your cluster or dig down for more information on an individual server, including charts of critical data.
- **Activities** - Describes how to observe the activities operating on the cluster, both in real time and via dashboards that show previous activity, and gives various data on the resources utilised by particular processes. Compare the performance of related jobs and individual task efforts to identify behaviour or performance issues.
- **Events** - Describes how to observe and alert on cluster-wide events and search their history. Time range, service, host, keyword, etc. may filter events.
- **Cloudera Manager** may be configured to create alerts from particular occurrences. Configure event thresholds, activate and disable them, and set up email or SNMP trap alerts for crucial events. You may temporarily silence notifications for particular roles, services, hosts, or the whole cluster for system maintenance/troubleshooting.
- **Lifecycle and Security Auditing** shows how to observe service, role, and host lifecycle activities such as establishing a role or service, making configuration updates, decommissioning and recommissioning hosts, and performing Cloudera Manager

management service commands. Time range, service, host, phrase, etc. may filter audit event entries.

- **Charting Time-Series** Data explains how to search metric data, build charts, organise data, and save charts to user-defined dashboards.
- **Logs** - How to retrieve logs based on the current context. When monitoring a service, you may click a single link to examine log entries using the same UI. When examining a user's activities, you may simply examine the appropriate log entries from the job's hosts.
- **Reports** - View historical disc use by user, user group, and directory and cluster job activity by user, group, or job ID. These aggregated reports (hourly, daily, weekly, etc.) may be exported as XLS or CSV files. You may search and set quotas for HDFS directories.
- **Troubleshooting Cluster Configuration and Operation** - Describes how to utilise Cloudera Manager log and notification features to troubleshoot issues.

Cloudera Manager and Health Test

Cloudera Manager uses health checks to monitor cluster services, roles, and hosts. Cloudera Management Service offers role health checks. Default settings allow role-based health checks. A basic health test is if every NameNode data directory has adequate space. A more complex health test may compare the latest HDFS checkpoint to a threshold or if a DataNode is linked to a NameNode. In a distributed system like HDFS, it's usual to have a few DataNodes offline (assuming dozens of servers), therefore we enable setting limits on what proportion of hosts should colour the whole service unavailable.

Tests might be good, concerning, or bad. If a test falls below a warning level, it's Concerning. If a test fails, it returns Bad. A service or role instance's overall health is determined by its health tests. If any health test is Concerning (but none are Bad), the role or service's health is Concerning. In Cloudera Manager Admin Console, health checks are color-coded: Good, Concerning, and Bad.

Monitoring and setup are often separated. Monitoring is enabled without further settings or tools (for example, Nagios). By having a comprehensive configuration model, Cloudera Manager knows which folders to monitor, which ports to utilise, and what credentials to use. When you install Cloudera Manager, all monitoring is enabled.

Cloudera Manager and Metric Collection and Display

Cloudera Manager uses health checks to monitor cluster services, roles, and hosts. Cloudera Management Service offers role health checks. Default settings allow role-based health checks. A basic health test is if every NameNode data directory has adequate space. A more complex health test may compare the latest HDFS checkpoint to a threshold or if a DataNode is linked to a NameNode. In a distributed system like HDFS, it's usual to have a few DataNodes offline (assuming dozens of servers), therefore we enable setting limits on what proportion of hosts should colour the whole service unavailable.

Tests might be good, concerning, or bad. If a test falls below a warning level, it's Concerning. If a test fails, it returns Bad. A service or role instance's overall health is determined by its health tests. If any health test is Concerning (but none are Bad), the role or service's health is Concerning. In Cloudera Manager Admin Console, health checks are color-coded: Good, Concerning, and Bad.

Monitoring and setup are often separated. Monitoring is enabled without further settings or tools (for example, Nagios). By having a comprehensive configuration model, Cloudera Manager knows which folders to monitor, which ports to utilise, and what credentials to use. When you install Cloudera Manager, all monitoring is enabled.

Cloudera Manager Alerts and triggers

An event records something interesting that has happened, such as a service's health changing or a log message being written. Many default-enabled events are specified.

A notable incident triggers an alert. Alerts are badged in event lists. Alert Publisher may deliver email or SNMP traps to a trap receiver.

A trigger describes an action to be executed when a service, role, role configuration group, or host meets certain requirements. The criteria are represented as a tsquery statement, and the action is to alter the service, role, role configuration group, or host health to Concerning (yellow) or Bad (red).

[Get Full Version Contents from this link](#)

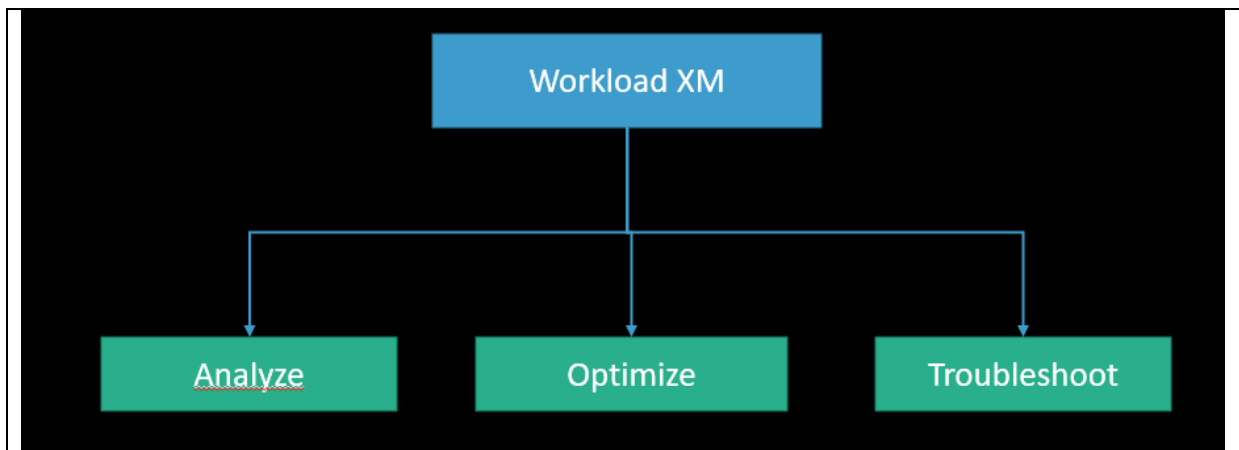


Contents

Chapter-32: Describe the use and major functions of Workload XM.....	1
Overview	1
Recent features added in Workload XM are.....	2
Introduction to Workload XM.....	2
Telemetry Publisher	3
Enabling Workload XM	3
How metrics are collected?	4
From Telemetry Publisher to Workload XM	4
Data Types collected by Telemetry Publisher Service	4
Removing Sensitive information from Metrics.....	5
Workload XM Web Interface	5

Chapter-32: Describe the use and major functions of Workload XM

Overview



How do you get to know, how much workload is being handled by your Cloudera CDP platform? For example, you want to know that

- Your workload is processed by which cluster.
- Which Service was involved in processing your workload?
- What data is processed as part of your workload.

These all things you can understand using Workload XM. Other things for your CDP, Workload XM can help is

- Troubleshooting failed jobs.
- Optimizing slow jobs
- Workload XM displays metrics about a Job's performance and compares the current Job execution with previous execution by creating baselines.

Cloudera Workload XM is a complete workload-centric solution that proactively optimises workloads, application performance, and infrastructure capacity for

- Data Warehousing
- Data Engineering
- Machine Learning settings.

It does this by monitoring and analysing all aspects of the workload. Customers and end users can get an interactive understanding of their workloads and improve their systems with the help of Workload XM.

Following are the main features of Workload XM and help you in

- To better understanding of their workloads, clusters, and resources.
- Analyses available on demand, together with instruction and suggestions for improvement.
- Detection and health checks at early stages, as well as daily reporting.
- Multiple data kinds originating from the platform's primary engines known as Cloudera Enterprise.
- Quicker incorporation of new use cases and applications as they become available.

Recent features added in Workload XM are

Analyze your cluster, job, and query costs with the Workload XM Chargeback feature: Chargeback lets you specify cost centres. Once established, Workload XM visibly shows cluster charges. This new functionality lets you define Workload XM cost centres for workload clusters based on CPU and memory use. Once formed, you may plan and estimate budgets and future workload situations utilising Workload cluster cost insights.

Workload XM Auto Actions - Notifications triggers job and query actions: Based on your criteria, this new functionality instantaneously initiates an action event. The event of your action is triggered when a task or query satisfies your action's requirements and its conditions are met. For instance, nodes or tasks may crash as a result of memory depletion. You may take action before a possible issue arises if you are aware when the amount of accessible memory is approaching a certain level. You may create an action using the Auto Actions feature that alerts you through email when a task is using too much memory so that you can take action to resolve a possible issue.

Utilize the Workload XM Purge Event to delete unused HDFS files: By customising and timing the new Workload XM purge event, bottlenecks between Telemetry Publisher and Workload XM may be lessened. With the use of this functionality, you can get rid of old HDFS data that can pile up or prevent Telemetry Publisher from absorbing and providing more recent data to Workload XM. The file's data group and the data group's retention limit determine the purge event.

Resource Consumption by Services: shows how much CPU and memory are used by each service throughout the given time period. To see how much CPU or memory is used by each of the cluster's services, as a percentage, move your mouse over the time line.

Resource Consumption by Nodes: shows the CPU and memory usage for every cluster node. The proportion of CPU or memory used by each node and its services is shown as you move your cursor over the time line.

Introduction to Workload XM

Cloudera's Workload XM helps you understand your workloads, clusters, and resources. Its analytics and health checks let you detect and troubleshoot current and future issues, and its suggestions help you promptly solve and improve them. Telemetry Publisher, a Cloudera Manager Management Service role, sends Workload XM diagnostic information after a workload completes.

- Metrics and health checks to discover and fix problems.
- Prescriptive advice and solutions to solve issues swiftly.
- Performance baselines and historical analysis to detect and fix issues.

Workload XM also:

- Visualize your workload cluster's current and historical expenditures to plan and anticipate budgets, workload environments, and user groups and resources.
- Trigger real-time actions across operations and queries to prevent issues.
- Enable daily email delivery of cluster data to track, compare, and monitor without logging in.
- Break down workload measurements to meet business needs and examine particular workload criteria. You may examine how queries that access a given database or resource pool perform versus SLAs. Or you may observe how a user's queries execute on your cluster.

Telemetry Publisher

Telemetry Publisher is a role, in a Cloudera Manager Management Service. When any jobs get completed, then information about the job and the cluster that processed the job is sent to Workload XM with Telemetry Publisher.

Enabling Workload XM initiates the Telemetry Publisher role. Telemetry Publisher sends Impala, Oozie, Hive, YARN, and Spark task metrics, configuration, and log files to Workload XM. Telemetry Publisher gathers measurements for Workload XM clusters.

Enabling Workload XM

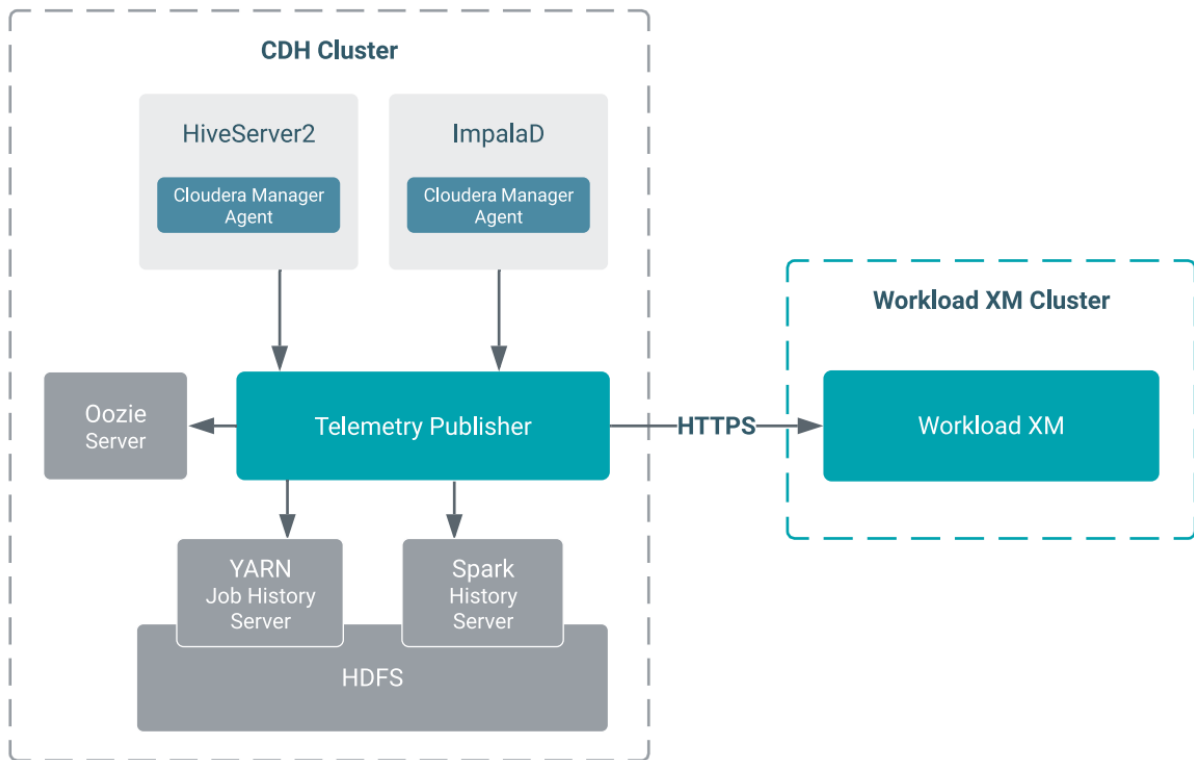
As soon as you enable the Workload XM, Cloudera Management Service will start the Telemetry Publisher Role as well. For the following jobs

- Impala
- Oozie
- Hive
- YARN
- Spark

Telemetry Publisher collects and transmits

- Metrics data
- Configuration Data
- Log files

For all of the above-mentioned services to Workload XM. You can have multiple servers configured in your CDP and Telemetry Publisher Role collects metrics for all the clusters that use Workload-XM environments. For a single cluster how, that is performed is shown below



How metrics are collected?

There are two mechanisms by which metrics can be collected

- **Pull:** As name suggests, Telemetry Publisher pulls diagnostic metrics from Oozie, YARN and Spark periodically. Default configuration is once per minute.
- **Push:** To push metrics data, agent must be installed for respective service. In case of Hive and Impala, Cloudera Manager Agent pushes, metrics data to the Telemetry Publisher within every 5 seconds after a job finishes.

From Telemetry Publisher to Workload XM

- Once data reaches to Telemetry Publisher, this would be stored under its own data directory for temporarily and periodically (once per minute) exported to the Workload XM.

Data Types collected by Telemetry Publisher Service

Telemetry Publisher Service collects and sends the following metrics data to Workload XM.

1. **YARN MapReduce Jobs:** As you know all the MapReduce jobs are executed by YARN, are saved in YARN Job History Server. So, Telemetry Publisher Pulls the configuration and jhist file.

- Jhist:
 - This is a Job History File.
 - Contains the job and task counters.
 - It is created in HDFS.
 - MapReduce Task logs:
 - MapReduce task logs are also created in HDFS.
 - Be default it is disabled to fetch MapReduce Task logs.
 - To send MapReduce Task logs, you need to enable it then only it can send data to Workload XM.
2. **Spark Applications:** Again, in this case as well Telemetry Publisher has to Polls
 - Applications which are completed, stored in Spark History Server.
 - Event logs for Spark can also be stored in HDFS.
 - Telemetry Publisher collects event logs from HDFS and send them back to Workload XM.
 - By default, for Spark Application Data Collection is not enabled.
 3. **Oozie Workflow:**
 - Similarly, Telemetry Publisher polls Oozie servers for recently completed Oozie workflows and send the metrics data to Workload XM.
 4. **Hive Queries:**
 - HlveServer2 creates a query detail file after query completed.
 - Cloudera Manager Agent periodically searches for the query details files.
 - Agent send these files to Telemetry Publisher.
 - To get these query files Hive Query Audits must be enabled.
 5. **Impala Queries:**
 - Impala creates Query profiles for recently completed queries.
 - Cloudera Manager Agent sends this query profile to Telemetry Publisher.

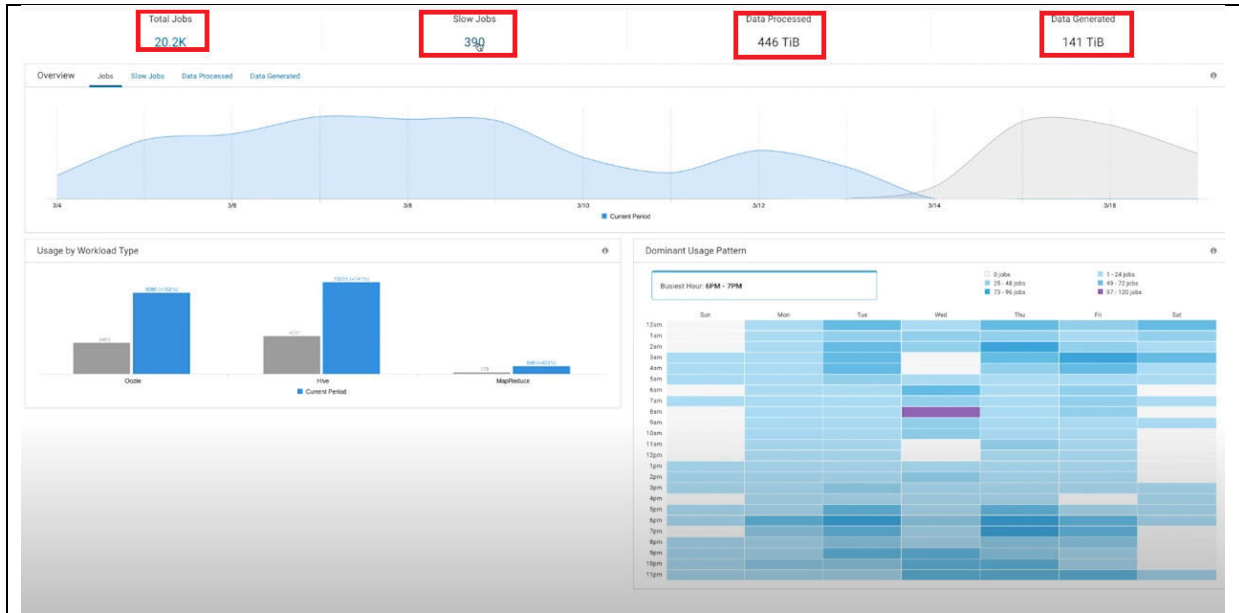
Removing Sensitive information from Metrics

- It is possible that applications you are running using Hive, Impala, MapReduce, Spark or Oozie have some sensitive information in Diagnostic Data.
- So, it is necessary to configure the redaction.
- It is recommended that even if you are not sending metrics or diagnostic data to Telemetry Publisher.
- Job configurations of logs can have sensitive information and that needs to redacted.
- Following are the list of data and resources which can be configured to redacting sensitive data before sending it to Telemetry publisher.
 - **Log & query redaction:** You have to create regular expression for filtering out the data. This needs to be done on the query and logs which are collected by Telemetry Publisher.
 - **YARN MapReduce Job properties:** As you know, Telemetry publisher pull job configuration data from the HDFS. Hence, before storing job configuration information in HDFS, you have to redact sensitive information.
 - **Spark Event logs & Spark executor logs:** Again, this can be filtered using regular expression for Spark2 jobs only. This can filter both event and executor logs.

- Be default this is enabled. However, you can override by using safety valves in Cloudera Manager or in the Spark application itself.

Workload XM Web Interface

The Workload XM UI presents graphical data on workload health, performance, and status. Its dashboard components include data, performance, health, and recommendations.



Reduce Stage

Task Duration Skew

This stage had **poor parallelization** as 4 (out of 643) tasks ran for very long time.



This health check compares the amount of time tasks take to finish their processing. A healthy status indicates that successful tasks took less than two standard deviations and less than five minutes from the average for all tasks. If the status is not healthy, try to configure the job so that processing is distributed evenly across tasks as a starting point.

Troubleshooting reason for long running failed query

Jobs
select count(*) from bdahd01p_dlar... Actions Details Analytics

Health Checks Execution Details Baseline Trends 3/5/2018 7:46 AM 13h 25m pi94210 root.bda.high

Expand All Collapse All

07:46	HV	select count(*) from bdahd01p...	13h 25m
07:46	MR	Stage-1	13h 25m
07:46		Map Stage	22s
07:46		Reduce Stage	13h 24m

Reduce Stage / Failed Reduce Tasks
task_1520179468024_5644_r_000408

Attempt	Host	Start Time	Duration
Attempt 0		7:46 AM	N/A
Attempt 1	attempt_1520179468024_5644_r_000408_0	7:48 AM	13h 22m
Error Message			
Error: java.lang.RuntimeException: Hive Runtime Error while closing operators: null			
Attempt 2	lbdp38nbd.prod.pncint.net	9:10 PM	4s
Error Message			
File does not exist: hdfs://nameservice1/tmp/hive/hive/66b39f32-abbe-4751-ace5-1d9a8ed72e6			
Attempt 3	lbdp38jbd.prod.pncint.net	9:10 PM	4s
Error Message			
File does not exist: hdfs://nameservice1/tmp/hive/hive/66b39f32-abbe-4751-ace5-1d9a8ed72e6			



Contents

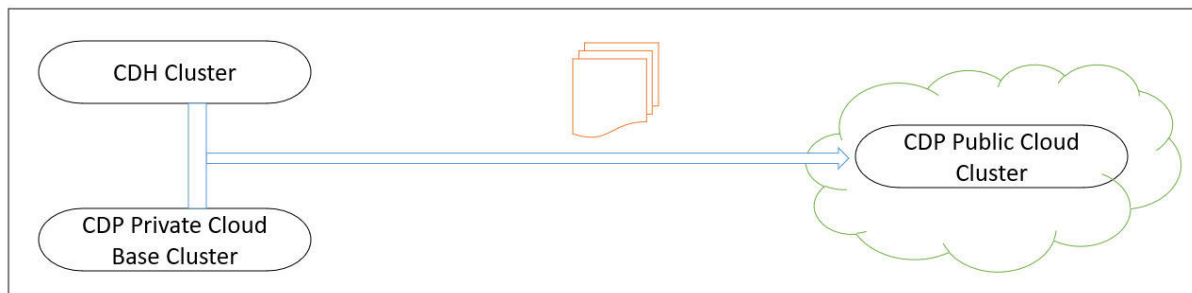
Chapter-33: Describe the use and major functions of Replication Manager.	2
Learn Some Terminology for Replication Manager	2
Policies for Some Components	3
Accessing Replication Manager	4
HBase backups	5
HDFS Replication.....	5
Hive/Impala replication.....	5
Sentry to Ranger replication	5
Verifying the Replication on CDP	5

Chapter-33: Describe the use and major functions of Replication Manager.

About Replication Manager

As name suggests, it is for replicating data to Public Cloud Clusters, which are managed using CDP. Replication Manager is a service in a CDP. That support copying data from HDFS, Hive and HBase to CDP Public Cloud clusters. However, there are different versions dependency what it supports or not. For that you need to look into Support Matrix for Replication Manager on CDP Public Cloud.

Replication Manager can be used to replicate data from CDH Clusters and CDP Private Cloud Base Cluster to CDP Public Cloud Clusters.



As mentioned, replication manager cannot support every possible use case and you may have to use alternate method for replicating data like Hive External tables and HBase, when Replication Manager is not able to support. For an alternate replication for HBase, you can use the Operation Database Replication plugin. Using this plugin, you can securely enable data replication for HBase data in Data Hub and CDP Operational Database. That plugin is not available openly, you need to connect to Cloudera Account team for getting the same. Replication Manager: service to copy and migrate data from CDH clusters to CDP Public Cloud.

- HDFS replication
- Hive metadata replication
- Hive external table replication
- Table-level replication

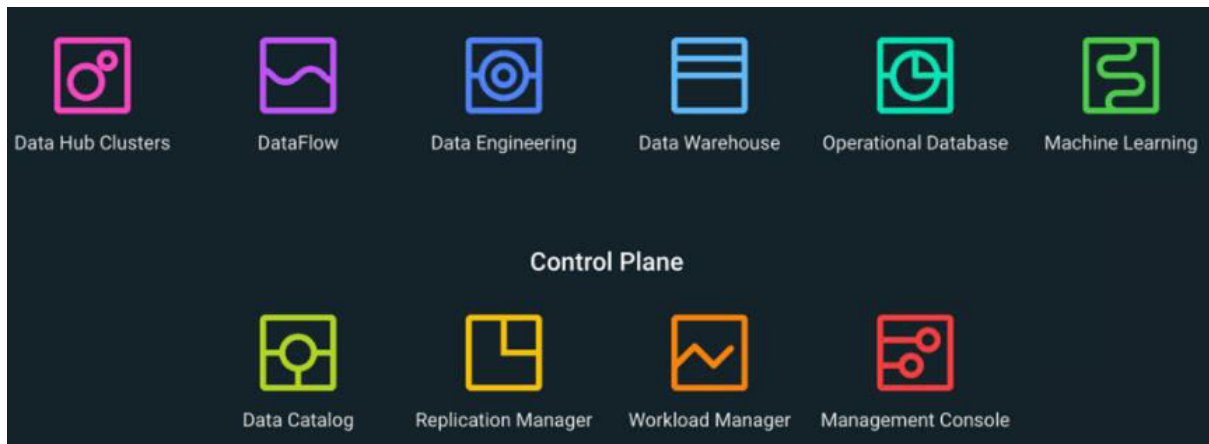
Many Cloudera clients are moving from on-prem to cloud by backing up their data or performing multi-functional analytics on CDP Public cloud like AWS or Azure. Replication Manager allows catastrophe recovery and data movement. Replication Manager allows users to effortlessly migrate tables/structured data and files/unstructured data to their chosen CDP cloud using easy-to-define rules.

Policies may enable periodic replication, allowing end-user control over workload transfer to the cloud. Replication Manager migrates Apache Hive, Apache Impala, and HDFS from CDH to CDP Public Cloud.

Learn Some Terminology for Replication Manager

Web UI

Replication Manager Service can be accessed using WebUI, which runs on the same host where Cloudera Data Platform runs. As you can see on the Replication Manager under Control Plane.



Policy

These are set of rules, that would be applied during replication. For example, which cluster is a source and destination, the type of data to replicate, schedule for replication etc.

Job

This represent an instance of a policy that can be in running state or already completed.

Data Lake

You can have a CDP cluster on Cloud, using virtual machines, and keeping data retained on Cloud storage. A Cloud data lake requires minimal services for metadata and governance, such as Hive MetaStore, Ranger and Atlas.

Cloud storage

You can have data in public cloud like AWS, Azure etc. So, if your storage is in cloud account like AWS S3 of AWS or Azure then that is called cloud storage.

Policies for Some Components

1. HDFS

To replicate data from HDFS, you must create schedules to replicate data incrementally.

2. Hive

You can replicate following data

- Data stored in Hive tables.
- MetaData for Hive tables.
- Data stored in Hive Metastore.
- Impala Metadata i.e. Catalog Server Metadata, which is associated with Impala tables registered in the Hive metastore.
- Replicating Hive external tables.
- Tables which are managed and ACID tables in Hive are converted to external tables after replication.
- You can do table level replication as well.
- You can also migrate Sentry Permissions to the Ranger.

3. HBase

For HBase all below mentioned replications are supported

- From CDP Private Cloud Base cluster to Data Lake Cluster.
- From CDH cluster to Data Lake Cluster

- From CDH Cluster to Cloudera Operational Database (COD) cluster.
- From COD Cluster to COD Cluster.

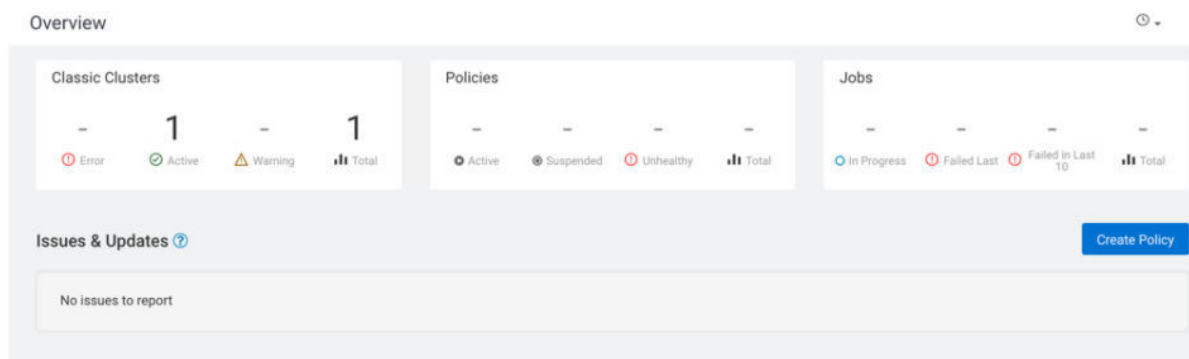
You can use HBase replication policies to perform an active-active disaster recovery with conflict resolution (enabling other disaster recovery use cases which provides an efficient utilization of resources), or to replicate HBase data in CDP Private Cloud Base clusters, CDH clusters, or COD. You can copy or replicate HBase data between different environments within a Virtual Private Cloud (VPC) using HBase replication policies. Any data change in the source cluster is pushed to the target cluster automatically without user intervention.

Accessing Replication Manager

As you can see replication Manager is under Control Plane. Once you open a Replication Manager there are in total 4 different web pages would be shown to you as below

1. Overview Page

Overview page provides a snapshot view of the Replication Manager service and insights into issues and updates related to various entities and resources through dashboards like Classic Clusters, Replication Policies, Notifications etc.



2. Classic Clusters Page

It is basically a status page for clusters, which specifies

- Total number of clusters enabled for Replication Manager
- Total number of clusters are in error state.
- Total number of clusters which are active.
- Total number of clusters for which a warning is issued.

3. Cloud Credentials page

This shows the registered cloud credentials for Replication Manager. If you are using Cloud Storage like S3 then you must provide credentials to access those cloud storage. Currently supported Cloud storage are S3 and Azure Blob Filesystem (ABFS). Hence, to add, update & delete cloud credentials you have to use Cloud Credentials page.

4. Replication Policies page

This page shows the policies which are in

- Active state
- Suspended state
- Error state
- Total number of policies which are available in Replication Manager.

HBase backups

HBase snapshots offer point-in-time table backups without data copies and minimum RegionServer effect. These policies take frequent snapshots of HDFS and HBase.

Snapshots are backups that may be used to restore an HDFS directory or HBase table to a prior version or another place on the same service. Replication uses snapshots. First-run replication strategy replicates all data and metadata from directories. Subsequent replication policy runs replicate updated data using HDFS snapshot diffs.

HDFS Replication

HDFS replication lets you transfer your HDFS data from one HDFS service to another, syncing the data sets based on a replication strategy. The destination service must be controlled by the Cloudera Manager Server setting up the replication, while the source service may be administered by the same server or a peer. Specifying source and destination folders lets you duplicate HDFS data inside a cluster. Remote Replication Manager replicates HDFS information with files. Locally backed up HDFS metadata.

These rules replicate HDFS data and metadata from CDH (5.10+) to CDP Private Cloud Base (7.0.3+) clusters.

Use HDFS replication policies for:

- Copying data from on-premises systems to Amazon S3 or Microsoft ADLS Gen2 (ABFS) cloud buckets or vice versa.
- Replicating data to another cluster to conduct load-intensive operations optimises the main cluster.
- Enterprise backup-restore solution.

Hive/Impala replication

Hive/Impala replication copies your Hive metastore and data from one cluster to another and synchronises them depending on a replication strategy. These policies replicate HDFS, Hive external tables (without manual translation of Hive datasets to HDFS datasets, or vice versa), Hive metastore data, Impala metadata (catalogue server metadata) associated with Impala tables registered in the Hive metastore, Impala data, and Sentry permissions to Ranger from CDH (5.10 and higher) to CDP Private Cloud Base (7.0.3 and higher) clusters. In this case, apps that rely on Hive's external table definitions update both replica and source.

Use replication policies to:

- Archive legacy or cold data.
- Run analytics on replicated or moved cloud data.
- Completely backup and restore data.

Sentry to Ranger replication

You may migrate Sentry rules for replicated Hive objects, Impala data, and URLs when creating or editing a replication policy. In the target cluster, the Replication Manager translates Sentry policies to Ranger policies. To replicate Sentry policies to Ranger, Cloudera Manager 6.3.1 is necessary.

Verifying the Replication on CDP

Once the Hive replication policy has successfully executed, the admin can perform the following validations to ensure that replication was indeed successful:

- HDFS data replication – The admin can enumerate the cloud storage path (S3 bucket path) to verify whether the data was successfully copied in the specified bucket.
- Hive metadata replication – The admin can verify whether the specified source database, along with tables, partitions, UDFs and column stats are indeed present in the Data Lake HMS instance. For this, the admin can use a Data Hub cluster and run the corresponding queries either via Hue or beeline.
- Ranger policies – Finally, the admin can query the Ranger policies to ensure that the Sentry policies are properly mapped as Ranger policies for the right users and groups.



Contents

Chapter-19: Cloudera Navigator Encrypt, Files System Encryption.....	1
Overview	1
Managing encryption keys	2

Chapter-19: Cloudera Navigator Encrypt, Files System Encryption

Overview

Many companies use SSL encryption so customers may access Hadoop data. With this integration, encrypted wire communication may be observed. Integrating Navigator makes Kerberos and LDAP authentication simpler as no keytab setting is needed. Cloudera Navigator enables audits, information management, and policy enforcement for Hadoop. Cloudera Navigator Encrypt encrypts and protects data at rest without changing apps and with minimum performance latency. Cloudera Navigator Key Trustee Server and Navigator Encrypt's process-based access controls help enterprises satisfy compliance laws and prevent unauthorised parties from accessing encrypted data. Navigator has following main characteristics:

- Key Trustee Server stores encryption keys to isolate them from encrypted data. Without the key, encrypted data is meaningless.
- Data is encrypted and decrypted transparently, with no performance effect and no programme changes.
- Individual processes may access encrypted data. Changing the process prevents malicious users from utilising customised application binaries to evade access restrictions.
- Navigator Encrypt's performance is increased by the Intel AES-NI cryptographic accelerator.

- Navigator Encrypt complies with HIPAA-HITECH, PCI-DSS, FISMA, EU Data Protection Directive, and other standards.
- Debian, Ubuntu, RHEL, CentOS, and SLES are supported.
- Easy setup: Navigator Encrypt has RPM, DEB, and SLES KMP packages.
- Multiple mountpoints: Each has its own encryption key.

Navigator Encrypt can encrypt several types of data, including:

- Databases.
- Backups & Temporary File (YARN containers, spill files, and so on).
- Reports & Log Files.
- Directories.
- File settings.

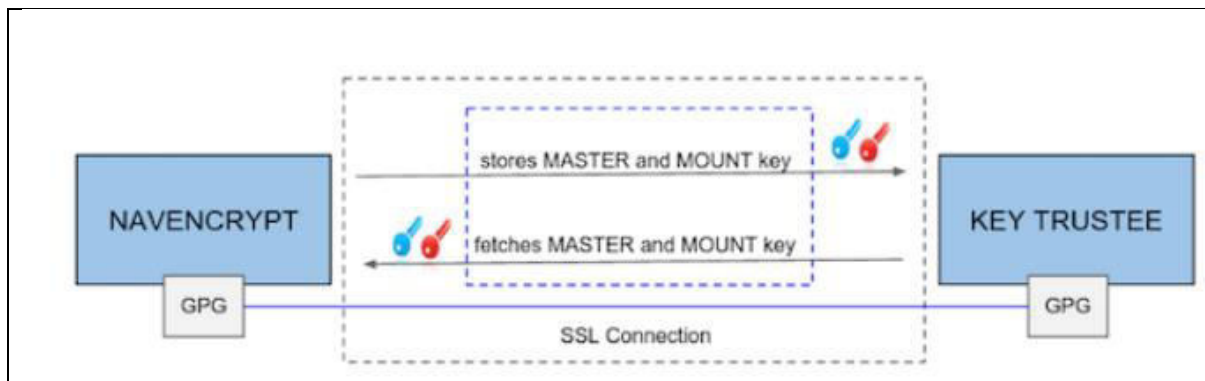
Navigator Encrypt employs several encryption keys.

- **Master Key:** The master key is a single, dual, or RSA key file. Key Trustee Server caches the master key. This key is used for Key Trustee Server registration and Navigator Encrypt client administration.
- **Mount Encryption Key:** Navigator Encrypt generates this key using openssl rand by default, although it may also use /dev/urandom. A new mount point generates this key. MEKs are at each mount point. Key Trustee Server receives this key.
- **Device Encryption Key:** Navigator Encrypt or Key Trustee Server don't handle dmccrypt DEK. dmccrypt stores it in the device's header.

Managing encryption keys

Key Trustee Server stores the master and mount encryption keys. Locally stored MEKs provide offline recovery and quick access. Master key encrypts locally-stored MEKs. TLS/SSL certificates secure Navigator Encrypt and Key Trustee Server.

The diagram below shows Navigator Encrypt and Key Trustee Server connection.



Local GPG key encrypts master key. It's encrypted again using the Key Trustee Server GPG key before being saved. When Navigator Encrypt needs the master key, Key Trustee Server decrypts it using its server GPG key and delivers it to the client, which decrypts it with its local GPG key. TLS encrypts all communications.



Contents

Chapter-20: SSL and TLS Implementation on Cloudera CDP	1
Overview	1
About SSL Certificates	1
Renew expiring certificates.....	2
Truststores and Keystores.....	2
Manual vs. auto-TLS.....	5
SANs	7
Cloudera Manager Auto-TLS TLS Encryption	7
Manual TLS setup.....	7

Chapter-20: SSL and TLS Implementation on Cloudera CDP

Overview

Transport Layer Security (TLS) 1.2 is a collection of industry-standard cryptographic protocols for safeguarding network connections. TLS is the successor of Secure Sockets Layer (SSL). The actual protocol utilised is TLS. SSL is not implemented inside Cloudera software.

HDFS and HBase use remote procedure calls to transport data (RPCs). To safeguard this transmission, RPC encryption must be enabled.

About SSL Certificates

TLS/SSL encrypts packets sent between endpoints to protect privacy and data integrity (ports on a host, for example). Configuring TLS/SSL requires establishing a private key and public key for server and client processes to negotiate an encrypted connection at runtime. TLS/SSL may employ certificates to validate the trustworthiness of negotiation keys to avoid spoofing and other security concerns.

Using TLS/SSL involves producing a private key, public key, and keystore, among other duties. Adding a certificate to the keystore may be the final step, but the lead time depends on the sort of certificate you want to use.

A certificate is digitally signed by a certificate authority (CA), which indirectly validates the public key given during negotiation. Table shows three methods to sign certificates.

Type	Usage Note
Public CA-signed certificates	This certificate is signed by Symantec or Comodo. Public CAs are trustworthy third-parties whose certifications may be validated publicly. This kind of certificate simplifies deployment since the Java JDK and its default truststore incorporate security infrastructure, such as root CAs.

Type	Usage Note
Internal CA-signed certificates	Internal CA signs this certificate. This certificate may be used by OpenSSL, Microsoft Active Directory, or another internal CA system.
Self-signed certificates	Production deployments discouraged. Self-signed certificates are suitable for non-production deployments like proof-of-concept.

During TLS/SSL cluster configuration, you receive a certificate for each host and re-use it for the host's services (daemon roles). Cloudera cluster components enable wildcard domains and SubjectAlternateName certificates instead of generating individual certificates for each server.

SAN and Wildcard Domain Certificates

CDP and Cloudera Manager support wildcard and SAN certifications. A wildcard certificate (*.example.com) may be used for any number of first-level subdomains inside a single domain. Wildcard certificates may be used with host-1.example.com, host-2.example.com, etc.

Wildcard certificates minimise the cost of public CA certificates. Using wildcard certificates makes it simpler to encrypt transitory and growing clusters, as the same certificate and keystore may be utilised. Wildcard domain certificates are insecure. Because all nodes share the same certificate, a compromise on one may affect others.

Wildcard Certificates	All domain hosts may utilise wildcard certificates. Using wildcard certificates for all cluster servers reduces expenses but increases risk.
SubjectAlternativeName Certificates	SAN certificates are tied to DNS names. All or a subset of cluster hosts may utilise a single SAN certificate. Cloudera Manager HA uses SAN certificates.

Renew expiring certificates

Public or internal CA-signed certificates expire. Most cluster operations fail with expired certificates. Cloudera Manager Agent hosts cannot verify the Cloudera Manager Server host and cannot deploy cluster nodes. When deploying certificates to cluster nodes, administrators should notice expiry dates and establish reminders to renew.

Truststores and Keystores

Java Keystore and Truststore

All clients in a TLS/SSL-configured Cloudera Manager cluster require access to the truststore to verify certificates. The certificates guarantee the client or server that the issuing authority is authentic.

The default Oracle Java JDK truststore (cacerts) contains Symantec's root certificate. Cloudera prefers jssecacerts over the default truststore. Copying cacerts to that filename creates the alternative truststore (jssecacerts). Additional roles or services may add certificates to this truststore. Startup Hadoop daemons load this truststore.

For Cloudera clusters, jssecacerts must start as a copy of cacerts since cacerts includes the default certificates required to create the chain of trust during the TLS/SSL handshake. After jssecacerts is formed, the cluster adds public and private root CAs. Configure TLS encryption for Cloudera Manager on each cluster host. The private keys are maintained in the keystore.

As setup for Cloudera Manager Server and CDP clusters, the keystore and truststore are separate files. Each server in a Cloudera Manager Server cluster needs its own keystore but may share a truststore.

Keystore	Truststore
Used by the server side of a TLS/SSL client-server connection.	Used by the client side of a TLS/SSL client-server connection.
Typically contains 1 private key for the host system.	Contains no keys of any kind.
Contains the certificate for the host's private key.	Contains root certificates for well-known public certificate authorities. May contain certificates for intermediary certificate authorities.
Password protected. Use the same password for the key and its keystore.	Password-protection not needed. However, if password has been used for the truststore, never use the same password as used for a key and keystore.
Password stored in a plaintext file read permissions granted to a specific group only (OS filesystem permissions set to 0440, hadoop:hadoop).	Password (if there is one for the truststore) stored in a plaintext file readable by all (OS filesystem permissions set to 0440).
No default. Provide a keystore name and password when you create the private key and CSR for any host system.	For Java JDK, cacerts is the default unless the alternative default jssecacerts is available.
Must be owned by hadoop user and group so that HDFS, MapReduce, YARN can access the private key.	HDFS, MapReduce, and YARN need client access to truststore.

The table above contains facts about the Java KeyStore (JKS) format, which is utilised by Java-based cluster services such as Cloudera Manager Server, Cloudera Management Service, and many CDP components and services.

CDP Services as TLS/SSL Servers and Clients

Cluster services are TLS/SSL servers, clients, or both.

Component	Client	Server
HBase	⊘	✓
HDFS	✓	✓
Hive	✓	✓
Hue (Hue is a TLS/SSL client of HDFS, MapReduce, YARN, HBase, and Oozie.)	✓	⊘
MapReduce	✓	✓
Oozie	⊘	✓
YARN	✓	✓
ZooKeeper	✓	✓

Start-up TLS/SSL daemons load keystores. When a client connects to a TLS/SSL server daemon, the server sends the client its starting certificate, and the client uses its truststore to verify it.

Cluster Components (JKS, PEM)

Many CDP services employ JKS keystores and certificates, including Cloudera Manager Server and Cloudera Management Service. Cloudera Manager Agent, Hue, Key Trustee Server, Impala, and other Python or C++ services need PEM certificates and keystores. PEM certificates comply to PKCS #8, which needs Base64-encoded certificate and private key files. The table lists many certificate kinds.

Component	JKS	PEM
HBase	✓	⊘
HDFS	✓	⊘
Hive (Hive clients and HiveServer 2)	✓	⊘
Hue	⊘	✓
Impala	⊘	✓
MapReduce	✓	⊘
Oozie	✓	⊘
Solr	✓	⊘
YARN	✓	⊘

Component	JKS	PEM
ZooKeeper	✓	

Cloudera Manager keystores and truststores should be:

- Separate each host's keystore. Each keystore should have a name that identifies the host type—server or agent. Password-protect the keystore containing the private key.
- Create a cluster-wide truststore. This truststore comprises root and intermediate CAs required to authenticate TLS/SSL certificates. Truststore isn't password-protected.

Manual vs. auto-TLS

Transport Layer Security (TLS) is the most used wire encryption protocol. TLS encrypts packets between endpoints to offer authentication, privacy, and data integrity. Applications employ REST APIs or Thrift to interface with Hadoop clusters.

TLS manual configuration

Step-1: Get Certificates

- Each host should generate a public/private keypair.
- Create CSRs for all hosts.
- Internal Certificate Authority should sign the CSR (CA).
- Generate keystore and truststore on all cluster hosts.

Step-2: Cluster Configuration

- Set TLS keystore and truststore for each service.
- Before enabling TLS for another service, restart the affected components.
- Change the cluster manager's UI (like setting up truststore, Enabling Knox SSL, etc.)

Step-3: Upkeep

- New service installation requires configuring the keystore and truststore. Start affected services.
- Each new cluster host must follow the "Get Certificates" procedures (only for the new hosts).
- Pre-expiring certificates are cycled.

Cloudera Manager's Auto-TLS

In big deployments, the aforementioned method might be time-consuming and cause operational problems. Auto-TLS automates cluster-level TLS encryption. Using Auto-TLS, you may let Cloudera administer the cluster's Certificate Authority (CA) or utilise the company's CA. In most

circumstances, Cloudera Manager's UI enables all essential actions. The following procedures are automated.

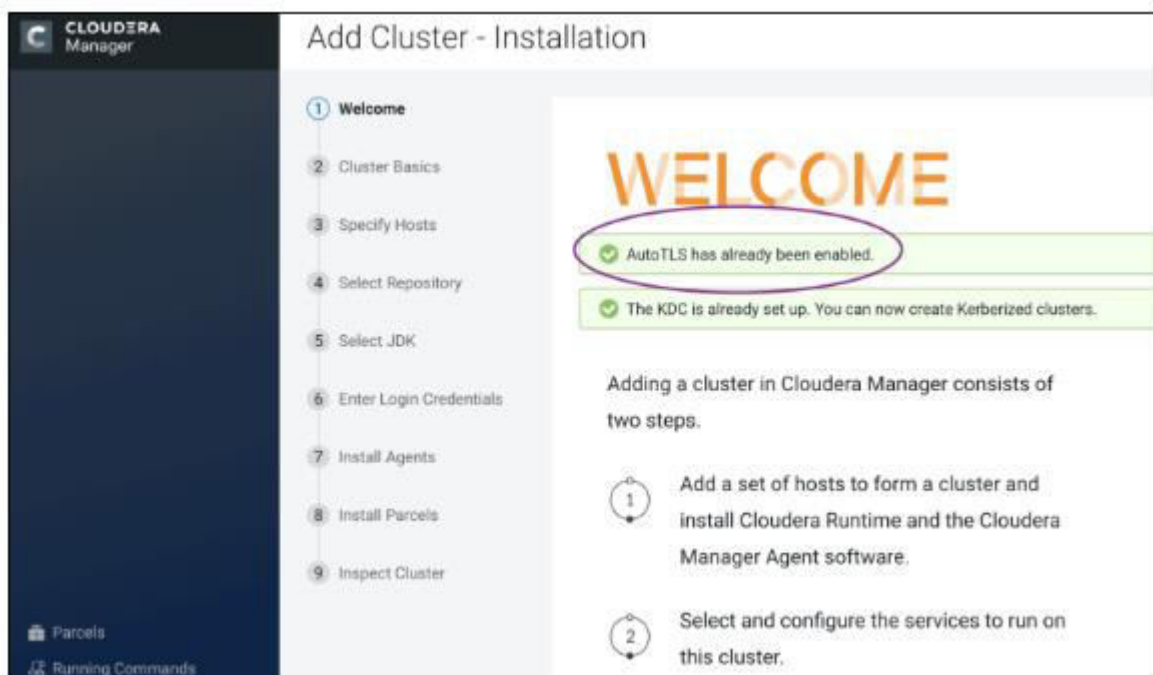
Cloudera Manager as a CA:

- Creates the root Certificate Authority or a Certificate Signing Request (CSR) for generating an intermediate Certificate Authority (CA)
- Signs host CSRs automatically.

Always do these

- Creates hosts' keystore and truststore.
- Deploys the cluster's certificates, keystore, and truststore.
- Configuring the keystore and truststore from a role instance specific directory enables TLS for all cluster services.
- Cloudera Manager server and agents get TLS.
- After this initial configuration, TLS is automatically activated on new services, hosts, and compute clusters.
- Automates certificate rotation.

When starting a new cluster on a Cloudera Manager with Auto-TLS enabled, you may use one of these options. The following notice should appear when you begin the wizard to build a new cluster. When you install the cluster, all services will be encrypted.



Auto-TLS speeds up initial wire encryption setup and automates cluster TLS settings. The table below compares blog choices. Auto-TLS minimises the TLS administration overhead of your cluster, offering improved security with decreased operational overhead and let you concentrate on customers and workloads.

Steps	HDP/EDH (manual)	CDP Private Cloud use case 1 - Using Cloudera Manager to generate an internal CA and corresponding certificates	CDP Private Cloud use case 2 - Enabling Auto-TLS with an existing Root CA	CDP Private Cloud use case 3 - Enabling Auto-TLS with Existing Certificates
Generate CSR	Manual	Automated	Automated	Manual
CSR Signed by CA	Manual	Automated	One-time	Manual
Deploy certificate to all hosts	Manual	Automated	Automated	Automated
Configuration for each service	Manual	Automated	Automated	Automated
Cluster restarts	Multiple	Once	Once	Once
Configuration steps	Manual	Automated	Automated	Automated
New Service steps	Manual	Automated	Automated	Automated
New Host cert. generation	Manual	Automated	Automated	Manual

SANs

A SubjectAlternativeName (SAN) certificate employs the SubjectAlternativeName extension to include multiple host names. Auto-TLS only supports SAN certificates with custom certificates. Internal Cloudera Manager Certificate Authority doesn't support SAN certificates.

Cloudera Manager Auto-TLS TLS Encryption

Auto-TLS simplifies Cloudera Manager TLS configuration. There are three different use cases, we can use to configure Auto TLS as described one by one below.

Manual TLS setup

Auto-TLS is recommended for Cloudera Manager and CDP TLS encryption. Auto-TLS automates cluster-level TLS encryption. Using Auto-TLS, you may let Cloudera administer the cluster's Certificate Authority (CA) or utilise the company's CA. In most circumstances, Cloudera Manager's UI enables all essential actions.

Cloudera Manager Server distributes Kerberos keytabs and password-protected configuration files to cluster hosts when you establish authentication and authorisation. Cloudera Manager Server and all cluster hosts must use TLS encryption to safeguard this transmission. HTTPS client connections to Cloudera Manager Admin Interface employ TLS encryption.

Cloudera Manager supports TLS. Without certificate authentication, a malicious user may add a host to Cloudera Manager by installing and configuring Cloudera Manager Agent. Install certificates on each agent host and configure Cloudera Manager Server to trust them.

[Get Full Version Contents from this link](#)



Contents

Chapter-21_Kerberos_Authentication	1
Overview	1
About Kerberos	2
Kerberos Principals	2
About Kerberos Keytabs	3
About Delegation Tokens.....	3
Understanding Token Format	3
Kerberos Authentication Process.....	3
Kerberos and Token Renewal	4
Kerberos CDP configuration.....	4

Chapter-21_Kerberos_Authentication

Overview

Users and services must authenticate their identity to access a system resource via authentication. Organizations handle user identification and authentication using time-tested technologies including pluggable authentication modules (PAM), Lightweight Directory Access Protocol (LDAP) for identity, directory, and other services, and Kerberos for authentication.

Cloudera clusters support these technologies. Organizations having existing LDAP directory services such as Active Directory (included in Microsoft Windows Server's Active Directory Services) may utilise existing user accounts and group listings instead of establishing new accounts across the cluster. User role authorisation may be supported via Active Directory or OpenLDAP.

Pluggable authentication modules (PAM) are standard Linux modules for external authentication. PAM setup in Cloudera Manager is easier than LDAP, and administrators may install additional PAM modules to provide new authentication methods. Cloudera Manager does not allow multiple LDAP servers, but you may set PAM to synchronise with them using SSSD. Cloudera Manager may be configured to utilise Apache Knox for authentication and PAM for LDAP lookups.

Cloudera supports Kerberos and Active Directory authentication. Active Directory supports Kerberos in addition to LDAP for identity management and directory capabilities. Kerberos offers robust authentication, which means cryptographic procedures are utilised instead of passwords.

Both systems exist. Microsoft Active Directory is an LDAP directory service that offers Kerberos authentication, and Kerberos credentials may be stored and managed in LDAP. Kerberos authentication may be used with Cloudera Manager Server, CDP nodes, and Apache Hive, Hue, and Impala. Cloudera does not implement Kerberos but may utilise MIT Kerberos or Microsoft Server Active Directory and its KDC for authentication. Configuring the cluster to utilise Kerberos needs

administrator access to the KDC (KDC). The approach may need troubleshooting Cloudera Manager and KDC difficulties.

AllowTgtSessionKey must be deactivated to utilise Microsoft Active Directory as a KDC (set to 0). Despite the "Successful" message at the conclusion of configuration/integration, users and credentials are not generated if this registry entry is enabled. Before setting Active Directory as a KDC, set AllowTgtSessionKey to 0 if required.

Cloudera Server and CDP services install local Linux user:group accounts on each node's host OS. Local user:group accounts are translated to LDAP-compliant directory service user accounts and groups to apply per-node authentication and authorisation across all cluster nodes.

Cloudera suggests utilising SSSD or Centrify Server Suite to authenticate each cluster node to the LDAP directory.

About Kerberos

Cloudera Clusters employ principals, keytabs, and delegation tokens. Cloudera uses Kerberos for authentication since native Hadoop authentication only checks for legitimate user:group membership in HDFS, not across all network resources. The Kerberos protocol authenticates a user or service for a limited duration, and each service the user wants to access needs the right Kerberos artefact. This section shows how Cloudera clusters employ Kerberos principals and keytabs for user authentication and how delegation tokens authenticate tasks on behalf of authorised users during runtime.

Kerberos Principals

Each user and service that requires Kerberos authentication needs a principal, which uniquely identifies them across numerous Kerberos servers and subsystems. A principal has up to three identifiers, beginning with the user or service name (called a primary). The main element of the principal is usually the user account name from the operating system, such as jcarlos for a Unix account or hdfs for a Linux account connected with a service daemon on a cluster node.

User principals consist of the main and Kerberos realm name. The realm is a logical collection of principles related to the same Key Distribution Center (KDC) with similar attributes, such as encryption techniques. Large businesses may utilise realms to delegate management for particular users or functions and distribute authentication-processing activities across different servers.

As illustrated in this user principal pattern, utilise your organization's domain name as the Kerberos realm name (in all capital letters).

```
username@REALM.HADOOPEXAM.COM
```

Primary and realm names identify users. amit@REALM1.HADOOPEXAM.COM and amit@REALM2.HADOOPEXAM.COM may be the same person. For service role instance IDs, the primary is the Hadoop daemons' Unix account name (hdfs, mapred, etc.) followed by the host's instance name. An HDFS instance's principal is

hdfs/hostname.fqdn.example.com@REALM2.HADOOPEXAM.COM. Forward slash (/) separates main and instance names.

```
service-name/hostname.fqdn.example.com@REALM.HADOOPEXAM.COM
```

HTTP is the main principle for Hadoop web service interfaces, not Unix. An instance name helps identify administrators. Principals amit@REALM2.HADOOPEXAM.COM and amit/admin@REALM2.HADOOPEXAM.COM have separate passwords and rights. Example principle for HDFS service role instance operating on a cluster in an organisation with realms for each geographic location:

```
hdfs/hostname.fqdn.example.com@MUMBAILAND.HADOOPEXAM.COM
```

The service name is the Unix account name used by the service role instance, such as hdfs or mapred. The HTTP principal for Hadoop web authentication has no Unix account, hence HTTP is the main.

About Kerberos Keytabs

A keytab includes a principal's encrypted key. Each host's keytab file for a Hadoop daemon includes the hostname. This file authenticates a host's principal to Kerberos without a password or human intervention. Access to keytab files should be securely guarded since they enable acting as a principal. They should be viewable by a small number of users, saved on local disc, and not included in host backups unless backup access is as secure as local host access.

About Delegation Tokens

A keytab includes a principal's encrypted key. Each host's keytab file for a Hadoop daemon includes the hostname. This file authenticates a host's principal to Kerberos without a password or human intervention. Access to keytab files should be securely guarded since they enable acting as a principal. They should be viewable by a small number of users, saved on local disc, and not included in host backups unless backup access is as secure as local host access.

Understanding Token Format

NameNode generates tokens using a random masterKey. All current tokens' expiration dates are maintained in memory (maxDate). Delegation tokens may expire when the expiration date is reached or be cancelled by the owner. Deleted tokens are expired or cancelled. The sequenceNumber identifies tokens. The next section explains Delegation Token authentication.

```
TokenID = {ownerID, renewerID, issueDate, maxDate, sequenceNumber}  
TokenAuthenticator = HMAC-SHA1(masterKey, TokenID)  
Delegation Token = {TokenID, TokenAuthenticator}
```

Kerberos Authentication Process

The client delivers the TokenID to NameNode to begin authentication. NameNode utilises TokenID and masterKey to construct the Delegation Token. If NameNode discovers the token in memory and the current time is less than the token's maxDate, the token is valid. If legitimate, the client and NameNode will authenticate each other using their TokenAuthenticator secret key and MD5

protocol. Since the client and NameNode don't exchange TokenAuthenticators, tokens aren't jeopardised if authentication fails.

Kerberos and Token Renewal

The authorised renewer renews delegation tokens regularly (renewerID). Authenticating to the NameNode is required if a NodeManager is the specified renewer. It sends the token to NameNode for authentication. Before renewing the token, NameNode validates the following:

- The requested NodeManager matches the token's renewerID.
- The NameNode's created TokenAuthenticator matches the one it previously stored.
- maxDate must be later than now.

NameNode sets the new expiration date to $\min(\text{current time} + \text{renew period}, \text{maxDate})$ if the token renewal request is successful. NameNode will lose all memory tokens if it's restarted. In this situation, the token is kept with a new expiration date. After a restart and before relaunching unsuccessful tasks, authorised renewers must renew all tokens with the NameNode.

A specified renewer may revive an expired or cancelled token if the current time doesn't exceed maxDate. Only the masterKey stays in memory, thus the NameNode cannot distinguish the difference between a cancelled or expired token and one deleted by a restart. Update masterKey often.

Kerberos CDP configuration

Kerberos configuration options for Cloudera Data Platform (CDP).

	A. Direct integration of Cloudera Manager with AD/MIT/IPA KDC	B. Keytab retrieval through "keytab retrieval script"	C. Manual deployment / configuration of keytab for services
Brief description of functionality	Cloudera Manager may create/delete service principals directly in the KDC and automatically retrieve keytabs.	Cloudera Manager calls an external script to get a principal's keytab.	Cloudera Manager will keep keytab/kerberos related settings for A or B column services, but we will override the default keytab location using a "Safety Valve" option.
Example	Default/recommended configuration. <ul style="list-style-type: none"> • Cloudera Manager may construct missing principals/keytabs 	<ul style="list-style-type: none"> • Internal restrictions/policy prevent direct AD/KDC integration. • Integration of unsupported KDC 	External variables may need customising keytabs.

	<p>after adding new services/role instances.</p> <ul style="list-style-type: none"> • Cloudera Manager can rebuild existing principals/keytabs. 	types with Cloudera Manager.	
Principal naming convention	<p>CDP Kerberos principals are service/fqdn@REALM.</p> <p>Most services utilise a single username for all roles (such "hdfs" or "hive"). A single service will likely require many principals, where the username is the same and the hostname matches one of the hosts where this service is executing. Hadoop Users (user:group) and Kerberos Principals list default service usernames.</p>		

Cloudera clusters may be manually setup to utilise Kerberos or configured via the Cloudera Manager Admin Console configuration wizard. The wizard automates numerous configuration and deployment chores. Enabling Kerberos for the cluster via the wizard enables Kerberos for all CDP components.

Configure TLS/SSL before enabling Kerberos authentication. Kerberos keytabs are sent unencrypted if not.

Cloudera Manager's Kerberos wizard verifies the cluster's Kerberos instance. Before utilising the wizard, obtain all Kerberos service data or ask the administrator for assistance. The wizard pages need numerous Kerberos information.

The wizard needs a functional MIT, FreeIPA, or Active Directory KDC. Before beginning the wizard, make sure KDC is functional. Wizard prompts need administrator-level Kerberos access. If you lack these credentials, the Kerberos administrator must help you.

Cloudera Manager has many authentication methods. Cloudera Manager can authenticate users against its database or an external service. The external authentication service might be an LDAP server or another external service. Cloudera Manager supports SAML single sign-on.

If you're using LDAP or another external service, you may set Cloudera Manager to utilise both kinds of authentication (internal database and external service) and the order in which it searches. Full Administrators may always authenticate against the Cloudera Manager database using an external authentication method. This avoids locking out everyone if authentication settings are wrong, such a

faulty LDAP URL. External authentication lets you limit login access to specified groups and provide their members Full Administrator access to Cloudera Manager.

Cloudera Manager appears in the User Type field for database users. External is the User Type field for LDAP and other external authentication users.



Contents

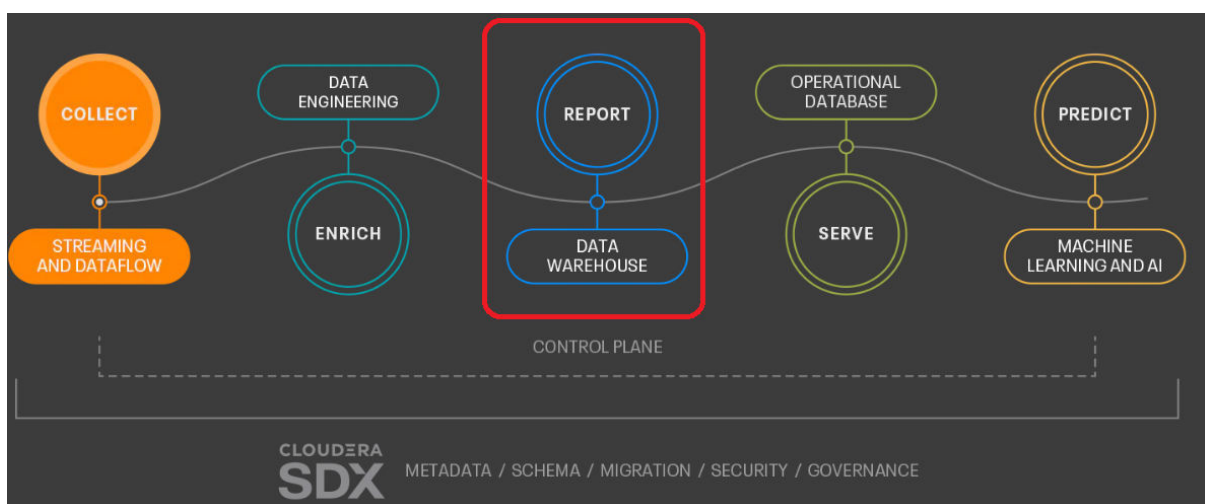
Chapter-23: Cloudera CDP DataWarehouse	1
Overview	1
Data Warehouse Cloudera Service	2
Architecture of Cloudera Data Warehouse.....	2
Analytics Experiences.....	4
Key Features of Cloudera Data Warehouse.....	5

Chapter-23: Cloudera CDP DataWarehouse

Overview

Data Warehouse is a CDP Cloud data service for establishing autonomous, self-service data warehouses and data marts that autoscale. The Data Warehouse service offers isolated compute instances for each data warehouse/mart, automated optimization, and cost-saving SLAs. Data Warehouse clients using CDP Public Cloud have a separate runtime. Apache Hive, Apache Impala SQL, and the Hue interactive SQL editor for testing queries and sampling data, included in Data Warehouse. Turn any data into business insights. CDP Data Warehouse allows IT to provide BI analysts with cloud-native self-service analytics in minutes. It outperforms rival data warehouses on all sizes and kinds of data, including structured and unstructured.

Data Warehouse on CDP integrates streaming, data engineering, and machine learning analytics. It has a unified architecture that protects and governs data and information on private, public, or hybrid clouds.



Deliver business insights based on huge amounts of verified data to thousands of users quickly and on a large scale, without sacrificing compliance or going over budget. Cloudera Data Warehouse outperforms shadow IT by keeping up with changing business needs and meeting SLAs with self-

service access to reports, dashboards, and advanced analytics. It does this by moving workloads from on-premises to any cloud in a smooth and secure way.

Data Warehouse Cloudera Service

Cloudera Data Warehouse (CDW) Data Service is a containerized application for establishing highly performant, autonomous, self-service data warehouses in the cloud.

Cloudera Data Platform (CDP) lets you build Data Mesh, Data Fabric, and Data Lakehouse. CDP provides a Data Lakehouse architecture by pre-integrating and unifying Data Warehouses and Data Lakes to assist data engineering, BI, and machine learning. Cloudera's support for an open data lakehouse focused on CDW simplifies data management for data practitioners and administrators.

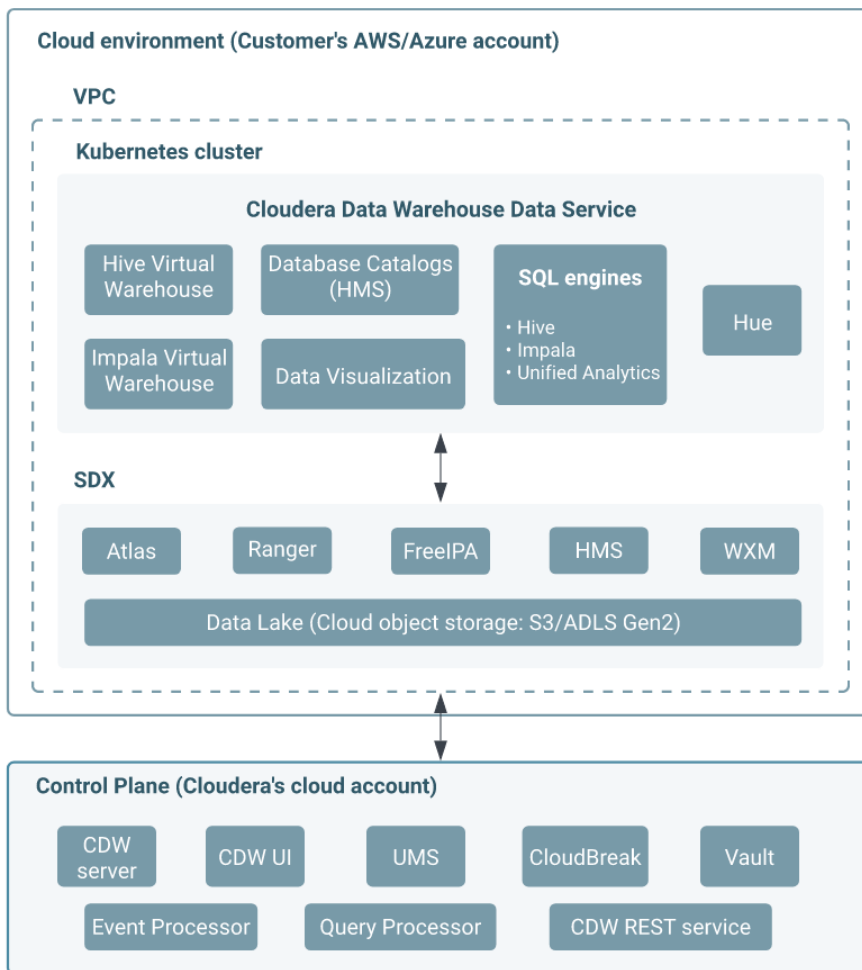
CDW integrates with Cloudera's data ingestion, engineering, machine learning, and visualisation services. CDW uses Apache Iceberg, Apache Impala, Hive ACID, and Hive table format to support wide workloads.

CDW enables rapid reaction times, high concurrency, simple data exploration, and business insight on Data Lake corporate data. CDW streamlines application development via open standards, file formats, and APIs. By isolating tenants' duties, CDW can fulfil report deadlines and reduce expenditures.

CDW streamlines multi-tenancy management and decreases cloud expenditures for admins. Virtual Warehouses may be deployed on-demand and de-provisioned when idle. Administrators may update Virtual Warehouses and Database Catalogs individually. CDW lets you choose the version of Hive, Impala, and Hue.

Architecture of Cloudera Data Warehouse

Administrators and IT teams may see how Cloudera Data Warehouse (CDW) service components fit within the CDP architecture. The CDW service consists of decoupled Database Catalogs (storage for a Virtual Warehouse) and Virtual Warehouses (compute environments that may access a Database Catalog). Multiple Virtual Warehouses of different sizes and kinds may run on the same Database Catalog, providing workload variety and isolation.



Database Catalogue

A Database Catalog contains table and view metadata, security permissions, and other data. Each Database Catalog has a Hive metastore (HMS) that stores data definitions. All your environment's data is in a secure object storage. A Database Catalog provides temporary Virtual Warehouse user and workload contexts and governance artefacts for auditing. Database Catalog is queried by many Virtual Warehouses. Multiple Database Catalogs may exist.

A default Database Catalog is established when you activate an environment from the Data Warehouse. The default Database Catalog shares HMS with Data Hub. You may access Data Mart or Data Engineering items or data sets from CDW Virtual Warehouses and vice versa. Queries and query history kept in Hue are not destroyed when a Virtual Warehouse is removed. When you install a non-default Database Catalog, Hue may import demo data.

Virtual Warehouses

Virtual Warehouses are Kubernetes instances that perform queries. Access Data Lake tables and views from a Virtual Warehouse. Virtual Warehouses bind computing and storage by running approved database queries. Virtual Warehouses can grow automatically and assure good performance. Querying the virtual warehouse requires JDBC/ODBC-compliant tools. Virtual Warehouses offer HS2-compatible endpoints for Beeline, Impala-Shell, and Impala.

Data Visualization

CDW incorporates Data Visualization for developing data visualisations, dashboards, and visual applications based on CDW data or other data sources you connect to. Authorized users may study CDP data using pie charts and histograms. Dashboards allow collaborative examination of graphics.

Analytics Experiences

Task-1: Exploring a Data Lake

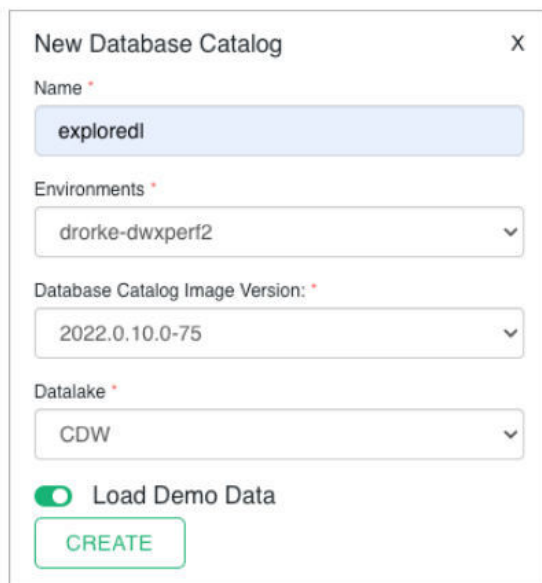
Description: Cloudera Data Warehouse's Virtual Warehouse lets you examine airline database tables in your data lake. You load airline information. You query tables. Set up a simple Virtual Warehouse to examine a data lake. Data lake exploration doesn't need auto-scaling or extra features. After exploring, remove Virtual Warehouse.

Step-1: Navigate to Data Warehouse > Database Catalogs > Add New.

Step-2: In New Database Catalog, in Name, specify a Database Catalog name.

Step-3: In Environments, select the name of your environment activated from Cloudera Data Warehouse.

Step-4: Accept default values for the image version and data lake type (CDW). For example:



New Database Catalog

Name *

exploredl

Environments *

drorke-dwxperf2

Database Catalog Image Version: *

2022.0.10.0-75

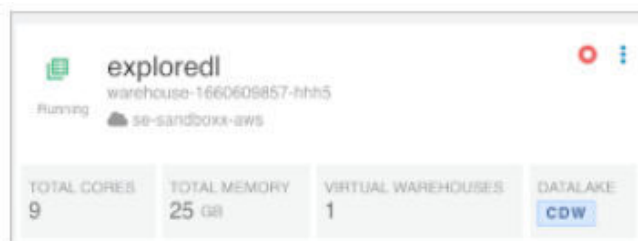
Datalake *

CDW

Load Demo Data

CREATE

Step-5: Turn on Load Demo Data to explore sample airline data from Hue, and click Create. For example,



exploredl

warehouse-1660609857-nhh5

Running

se-sandbox1-aws

TOTAL CORES	TOTAL MEMORY	VIRTUAL WAREHOUSES	DATALAKE
9	25 GB	1	CDW

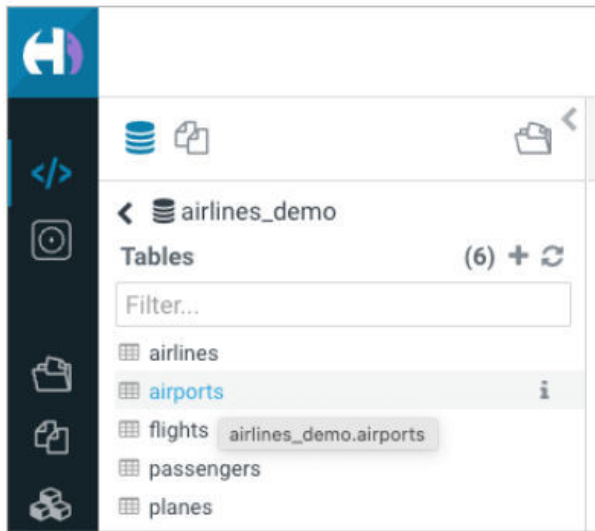
Step-6: Click Virtual Warehouses > Add New.

Step-7: Set up the Virtual Warehouse:

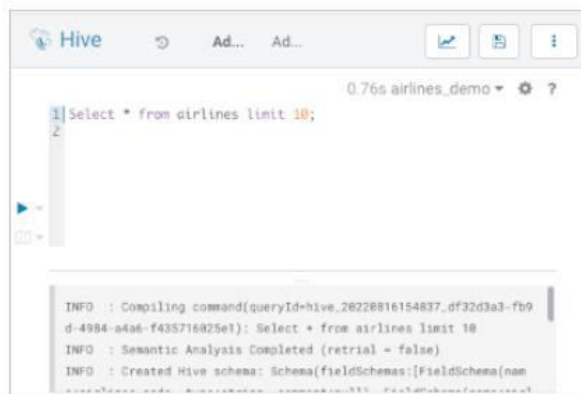
- Specify a Name for the Virtual Warehouse.
- In Type, click the SQL engine you prefer: Hive or Impala.
- Select your Database Catalog and User Group if you have been assigned a user group.
- In Size, select the number of executors, for example xsmall-2Executors.
- Accept default values for other settings.

Step-8: Click Create.

Step-9: After your Virtual Warehouse starts running, click Hue, and expand Tables to explore available data.



Step-10: Explore data lake contents by running queries. For example, select all data from the airlines table.



Key Features of Cloudera Data Warehouse

Self-service provisioning/administration: Start examining datasets from an intuitive data catalogue in minutes. Auto-scaling and auto-suspend provide zero-touch provisioning and management of data warehouses.

Unmatched scale and volume: High-performance SQL engines like Impala and Hive LLAP provide sub-second query response times for huge datasets (150PB+). Workload separation and optimization unblock hundreds of people and thousands of use cases on the same data.

Real-time data kinds: Add machine log, event stream, IoT sensor, media, and sentiment data to standard datasets. Make all data available in a single catalogue for dashboards, reporting, ad-hoc, and exploratory analytics.

Tools for queries and optimization: A set of tools, like Data Visualization, Hue, and Workload XM, that make it easy to explore, visualise, and query datasets and optimise workload health for maximum efficiency.

Using analytics for the whole lifecycle of data: Unlock the potential of data for thousands of users and hundreds of thousands of use cases. Workload isolation and optimization, auto-scaling, and easy-to-use web-based self-service tools make sure that everyone can do their work on the same data without stepping on each other's toes.

Data visualisation: CDP Data Visualization lets users build and publish custom dashboards and analytic applications in minutes. This makes it easy and quick to make interactive dashboards and share insights across your business. Use easy-to-use tools and instant sharing of insights across your business to help teams work together well.



Contents

Chapter-24: Cloudera Operational Database	1
Overview	1
Cloudera Operational Database Benefits & Characteristics	2
Practical Examples of COD	2
Deep Understanding of CDP OpDB	4
Cloudera OpDB and NoSQL	5
Operation Database Security	6

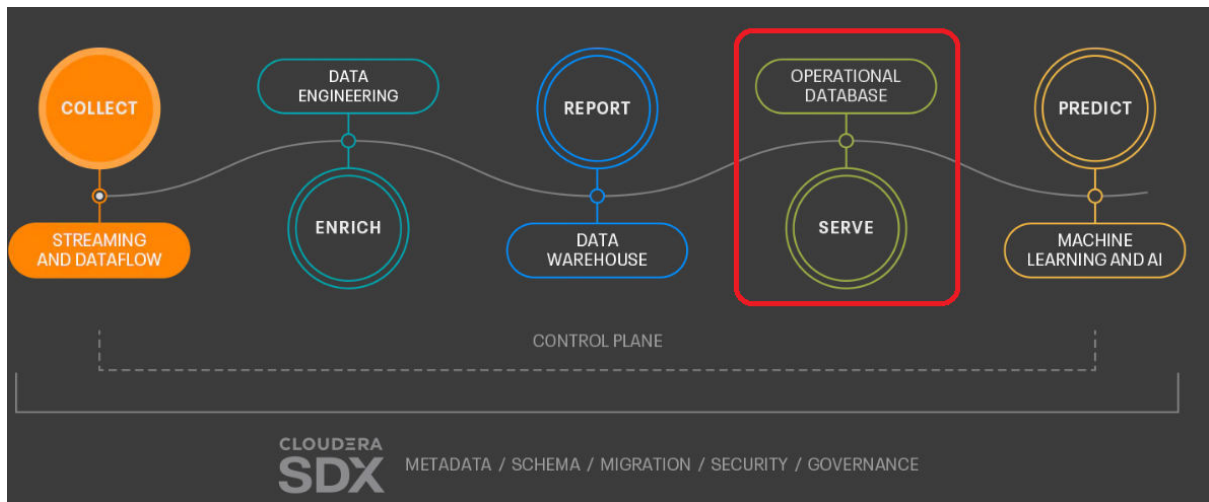
Chapter-24: Cloudera Operational Database

Overview

Cloudera Operational Database serves both structured and unstructured data on a single end-to-end open-source platform. This makes sure that decisions are based on stream processing and real-time analytics on data that is always changing. Users can serve real-time data at scale with high concurrency and low latency, and they can do data science at scale to make it easy to build, score, and put machine learning models into production.

COD is a Cloudera Data Platform service (CDP). COD creates a new operational database with one click and auto-scales depending on workload. Cloudera Operational Database is a real-time, always-available, scalable operational database that handles structured and unstructured data.

Apache HBase and Apache Phoenix power Cloudera ODBC. Apache HBase is used as a datastore in Cloudera Operational Database, using HDFS and/or S3 as storage. You may use Apache HBase API or Apache Phoenix to access data. Apache Phoenix is an ANSI SQL layer. It operates on Apache HBase and allows SQL queries and Apache Phoenix commands to manage data. Cloudera Operational Database is cloud- and on-premises-ready.



You set up COD on a public cloud infrastructure, which gives you features and flexibility that your hardware on-premises can't always give you. Using a CDP environment that is already set up, you can quickly make a working database with just one click. You can start a database with the same reliable and consistent storage technology you may already be familiar with if you have used CDH or HDP, but without any of the legacy complexity.

Cloudera Operational Database Benefits & Characteristics

You instal COD on a public cloud architecture, which offers features and flexibility that on-premises hardware can't. You may construct an operational database with a single click in an existing CDP system. You may start a database using CDH or HDP's durable and consistent storage technology, but without the legacy complexities.

Data Access: Apache HBase Java API, Phoenix JDBC driver, or Phoenix thin client JDBC driver may access COD data. CDP components and experiences may also aid data ingress.

SDX Cloudera support: Cloudera SDX Data Lake gives COD standard security, auditing, and lineage features. COD instances for R&D environments may be paused and resumed to save cloud costs.

Auto-scaling: Your database's capabilities may expand to manage rising traffic and decrease to minimise expenses without affecting availability. COD monitors services and accumulates metrics. By evaluating gathered metrics, COD achieves latency requirements. COD improves with time. COD analyses underlying services and automatically scales them to meet latency and RPC metrics. COD lets you construct a cluster rapidly and features auto-scaling for varied workloads.

Auto-healing: COD monitors the clusters and fixes them if a problem occurs. COD monitors the CDP cluster instances and recreates damaged or missing instances.

Practical Examples of COD

Complete circle of care for the customer:

- To counteract the far-reaching consequences of the change in consumer expectations, your company should enable a comprehensive picture of your consumers across all of your goods, systems, devices, and interaction channels.
- Make sure the experience you provide is consistent, tailored to the user, and relevant to their current situation.
- Models for predicting customer turnover may be used to hone in on potentially vanishing clients and implement preventative retention measures.

Putting Things Together:

- Make sure there is enough capacity for all users to access data with minimal latency and high concurrency.
- Create data-driven apps to share specialised, easily understood data across departments.

Time-series:

- Decision-making touchpoints should include real-time data and analysis.

Technology geared for the end user:

- Enable direct delivery of analytics to users of mobile and web apps.
- Applications may make use of this data structure as a key-value store.
- Model-based scoring and serving should be put into operation.
- Models for operational data that may be used for prevention, optimization, prescription, and prediction can be built and scored.
- Raising the success rate of cross-selling and upselling efforts.
- Accurately estimate a customer's creditworthiness and lifetime value.

Management of fraud and security risks:

- Implement serving and detecting fraud models.

Internet of Things - Managing Operations and Generating Revenue:

- Use the Internet of Things to improve your company's operations and model.
- Through continuous monitoring, alerting, and troubleshooting in real time, you can always know how your fleet is doing.
- Create monetary benefit by opening up opportunities for novel business configurations.
- Raising the success rate of cross-selling and upselling efforts.

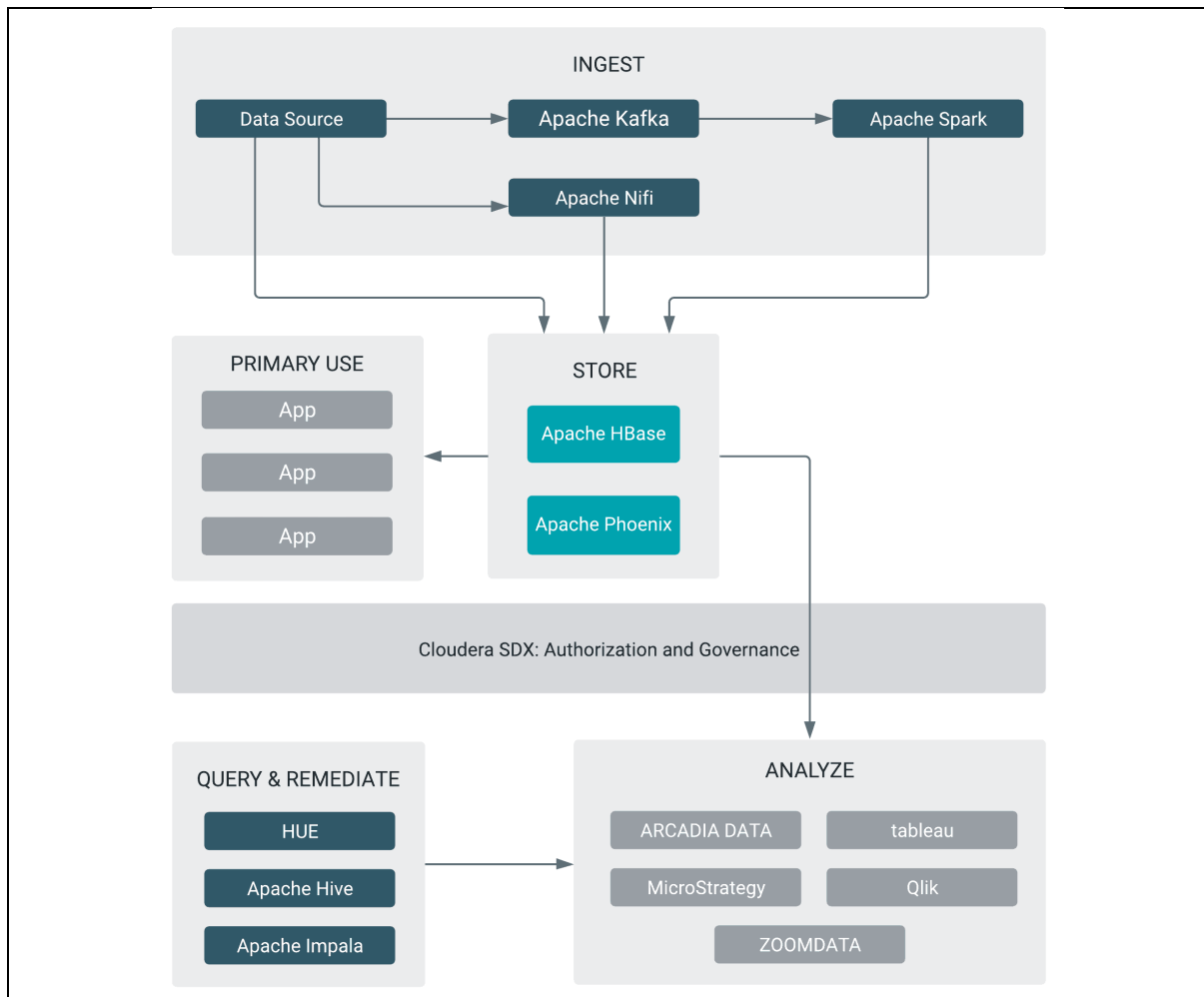
Superiority in operation:

- Total cost of ownership (TCO), efficiency, and threat mitigation are all ways to move toward operational excellence.
- By actively collecting and monitoring data from the network, downtime may be reduced via preventative measures.

- Maximize productivity while minimising overhead by analysing data in real time from connected devices.

Deep Understanding of CDP OpDB

When it comes to serving both old structured data and new unstructured data inside a unified Operational and Warehousing platform, Cloudera's Operational Database (OpDB) in CDP delivers. Within a single, open-source framework, Cloudera feeds both conventional structured data and emerging unstructured data from its operational database.



The usefulness of the operational database is best shown by the following.

- Apply machine learning and AI in the real world to alter industries like healthcare, public services, and more.
- Provide web-scale, real-time content.
- Facilitate the application of big data analytics in offline settings.
- It may be used as a reliable data repository.

Both a completely secure, semi-managed service in CDP Public Cloud - Data Hub and a fully configurable offering in CDP Data Center are already available, with the latter being identical to what

is previously offered in CDH and HDP. Depending on your desired deployment method and OpDB requirements, a suitable format may be selected. Apache HBase writes its data files, or HFiles, to an object store like Amazon S3, while its write-ahead logs, or WALs, are saved to HDFS in the production database.

These parts make up CDP's operating database:

- SQL interface for Apache HBase, known as Phoenix.
- With Apache HBase, you can store an infinite quantity of data on a single platform and accommodate rising needs for data serving because of its built-in scalability.
- Apache ZooKeeper functions as a name registry, a synchronisation service, and a distributed configuration service.
- With the help of Apache Knox Gateway, businesses can easily open their networks to new users without worrying about breaching any security measures.
- The Apache HBase WALs are written using Apache HDFS.
- The Apache HBase HFiles are kept in an object storage, such Amazon S3 or Microsoft ADLS Gen2.
- Data security and management tools are provided by Shared Data Experience (SDX). All data and workloads share the same set of security and governance regulations.
- IDBroker is a Representational State Transfer (REST) API that was developed for use with Apache Knox's authentication features. It enables a verified user to trade in their credentials or token for access tokens from a cloud provider.

Cloudera OpDB and NoSQL

Document Store: The Operational Database (OpDB) from Cloudera is multi-model because it can work with a wide variety of object models out of the box. Key-value, wide-column, and relational databases are all available, and users may even provide their own object model.

Nifi and Hive may be used to convert and store many models, including JSON and XML, while Hive can be used to store models natively as key-value pairs and query them. You may add support for JSON and XML with JSONRest's custom implementation features.

Object Store: Direct support for durable object storage like Azure Data Lake Store and S3 is available in Cloudera's OpDB (AWS native and implementations like Ceph). HBase's store files, which contain the majority of the database, may be backed up to an object store.

HDFS-integration: Cloudera's significant presence in Hadoop enables close interaction with HDFS. Snapshots, Export from operating systems, or moving underlying files (HFiles on HDFS) offline might export data.

Spark-integration: OpDB supports Spark. Spark may access tables as external data sources or sinks via many integrations. Spark-SQL works with DataFrames and DataSets.

DataFrame and DataSet support all catalytic optimization approaches. This achieves data locality, partition pruning, predicate pushdowns, scanning, and BulkGate. Co-locating Spark worker nodes on the cluster enables data locality. OpDB read/write is supported.

Each table needs a catalogue. This catalogue comprises the row key, columns with data type and preset column families, and column-to-table schema mapping. The catalogue is json.

An HBase DataFrame is a Spark DataFrame that can interface with Hive, ORC, Parquet, JSON, etc. Avro, Phoenix, and PrimitiveType are Java's primitive serdes.

Streaming: Cloudera's OpDB includes streaming data processing frameworks and tools.

DataFlowCloudera (CDF): Cloudera DataFlow is a scalable, real-time streaming data platform that gathers, curates, and analyses data.

Flow management: CFM is a no-code Apache NiFi data intake and management solution. It provides enterprise-scale data transport, transformation, and administration. Nifi automates system-to-system data transfer.

Analytical streaming: Cloudera Streaming Analytics driven by Apache Flink gives real-time streaming analytics. CSA delivers a low-latency, scalable streaming solution. It enables connections based on sources and sinks, such as HBase Streaming. Cloudera Streaming Analytics

Streaming: Apache Kafka is the fundamental stream processing engine for Cloudera Stream Processing (CSP). It manages streams. Cloudera Stream Processing has details.

Spark: Spark Streaming is a micro-batched stream processing framework. HBase and Spark Streaming are perfect partners because HBase can aid Spark Streaming:

- A quick source of reference or profile data
- A location to store counts or aggregates to fulfil Spark Streaming's once-processing guarantee.

Operation Database Security

Encrypting data-at-rest: HDFS' Transparent Data Encryption (TDE) capability encrypts data-at-rest.

- Data end-to-end encryption
- cryptographic and administrative functions separated
- Lifecycle management features are mature

Our cloud installations for cloud-native storage include encryption key escrow with AWS KMS or Azure Key Vault.

Encryption in Transit: OpDB's wire encryption employs TLS. It gives networked applications authentication, privacy, and data integrity. OpDB's Auto-TLS functionality simplifies TLS encryption on your cluster. Apache Phoenix, HBase (Web UIs, Thrift Server, REST Server) enable Auto-TLS.

Ranger Key Management: Ranger KMS stores encryption zone keys (EZKs) needed to decode data encryption keys to view decrypted files. Through RangerKMS, users may segregate key and data access controls. EZKs are kept in a KMS database. This database may be installed on cluster nodes in a secure way.

The EZKs are encrypted using a master key that is externalised in HSMs. Key rotation and versioning are enabled through configuration and policy management interfaces. Access audits in Apache Ranger monitor access keys.

Decryption: Decryption only happens at the client, and during decryption, no zone key leaves the KMS.

Due to the separation of duties (for example, platform operators cannot get access to encrypted data at rest), it is possible to control who can access decrypted content and under what circumstances in a very fine-grained way. Apache Ranger automatically takes care of this separation by using fine-grained policies to limit operator access to decrypted data. Ranger KMS has the same management interface that can be used to rotate and roll over keys.

User Authentication: Cloudera's platform supports the following forms of user authentication:

- Kerberos
- LDAP username/password
- SAML
- OAuth (using Apache Knox)

Authorization:

Attribute-based access control:

Cloudera's OpDBMS includes Apache Ranger for RBAC and ABAC. Cell, column family, table, namespace, and global authorization are possible. This supports creating roles as global administrators, namespace admins, table admins, or any mix of these scopes. Apache Ranger centralises the definition, administration, and management of large data security rules. ABAC-based policies may incorporate the subject (user), action (create or edit), resource (table or column family), and environmental attributes to produce a fine-grained authorization policy.

Apache Ranger offers sophisticated features including security zones, Deny policies, and policy expiry (setting up a policy which is enabled only for limited time). These properties, together with those outlined above, help establish effective, scalable, and manageable OpDBMS security policies.

Descriptive characteristics may be used to manage OpDBMS access in large-scale applications with a limited number of access control rules. Descriptions: AD groups Apache Atlas-based tags, geo-location, and other topic, resource, and environment information.

Apache Ranger rules may be exported/imported into another OpDBMS system with minimum effort. This technique allows compliance employees and security administrators to specify GDPR-required security rules finely.

Admin database access: Apache Ranger allows fine-grained database management via rules or grant-and-revoke procedures. It gives users and groups fine-grained permission mapping. This allows DBAs to have just the needed rights for certain resources (columns, tables, column families, etc.).

Administrators or operators might be restricted from decrypting TDE-encrypted HDFS data. Specific key access restrictions ensure that administrative users cannot see or update encrypted data because they lack key access.

Blocking unlawful use: Several of Cloudera's query engines employ variable binding and query compilation to avoid SQL injections. Our platform undergoes dynamic penetration testing and static code scanning for every customer-facing release to identify SQL injection and other vulnerabilities.

Apache Ranger's extensive security system can prohibit unauthorised use.

Least Privileges Model: Ranger's OpDB default is deny. A user is denied access to a resource if no policy expressly grants authorization. Policies must permit explicit privileges. Specific roles map privileged users and actions. Apache Ranger has delegated administration facilities to enable specified permission for certain resource groups.

[Get Full Version Contents from this link](#)



Contents

Chapter-27: DESCRIBE REQUIREMENTS TO DEPLOY CDP PUBLIC CLOUD ON MAJOR CLOUD INFRASTRUCTURE: AWS.....	1
Overview	1
CDP On Public Cloud	2
CDP in Public Cloud Examples.....	3
CDP Services in Public Cloud	3
Data Services.....	4
Security and Governance	4
Accessing CDP on Public Cloud	4
CDP On AWS.....	5
AWS resources created for Data Hub	8
AWS resources created for Data Engineering	9
AWS resources created for DataFlow	10
AWS resources created for Data Warehouse.....	11
AWS resources created for Machine Learning	12
AWS resources created for Operational Database	13

Chapter-27: DESCRIBE REQUIREMENTS TO DEPLOY CDP PUBLIC CLOUD ON MAJOR CLOUD INFRASTRUCTURE: AWS

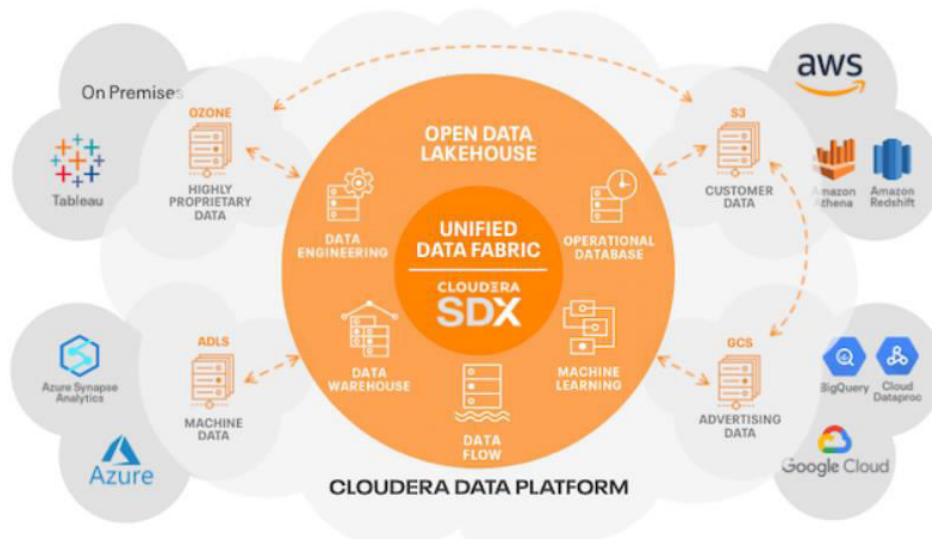
Overview

Cloudera Data Platform (CDP) allows users to pick any cloud, analytics, or data. CDP has public and private clouds. Cloudera Data Platform securely moves applications, data, and users across the data centre and several public clouds.

- A single data fabric orchestrates data sources across several clouds and on-premises.
- An open data lakehouse offers multi-function analytics on streaming and stored data in hybrid multi-cloud.
- A scalable data mesh eliminates data silos by spreading responsibility to cross-functional teams.

CDP delivers enterprise-level security and control with Cloudera SDX. Cloudera SDX combines enterprise-grade centralised security, governance, and administration with shared metadata and a

data catalogue, avoiding expensive data silos, proprietary format lock-in, and resource contention. Now users and admins may exchange data.

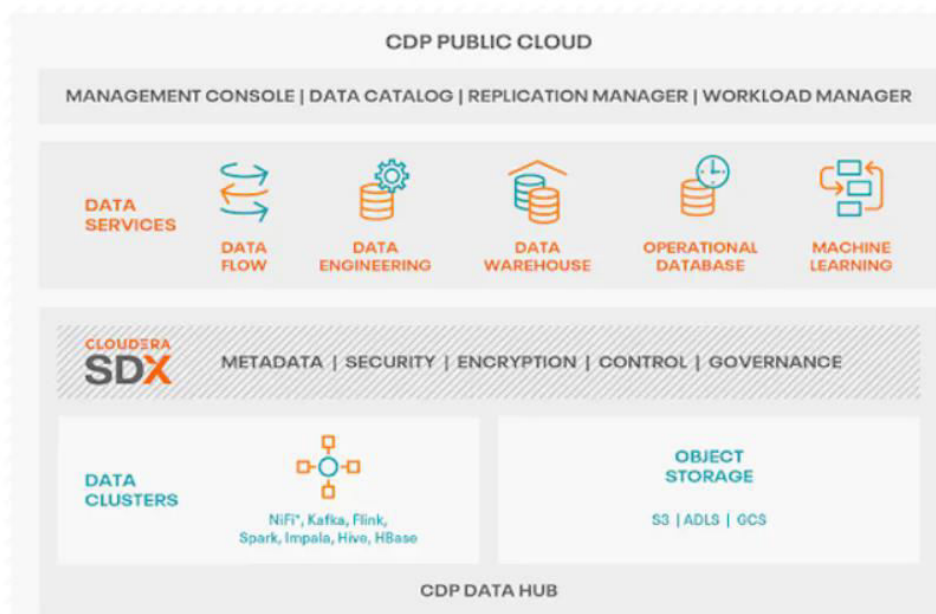


CDP On Public Cloud

Create safe data lakes, self-service analytics, and machine learning services without data platform software. Cloudera manages CDP Public Cloud services, but your data will always stay under your control in your workloads and cloud account. AWS, Azure, and Google Cloud host CDP.

You can:

- Control cloud expenditures by automatically starting workloads when required, scaling them as demand changes, and stopping them when done.
- Isolate and regulate user, workload, and priority workloads.
- Centralize customer and operational data across multi-cloud and hybrid systems.

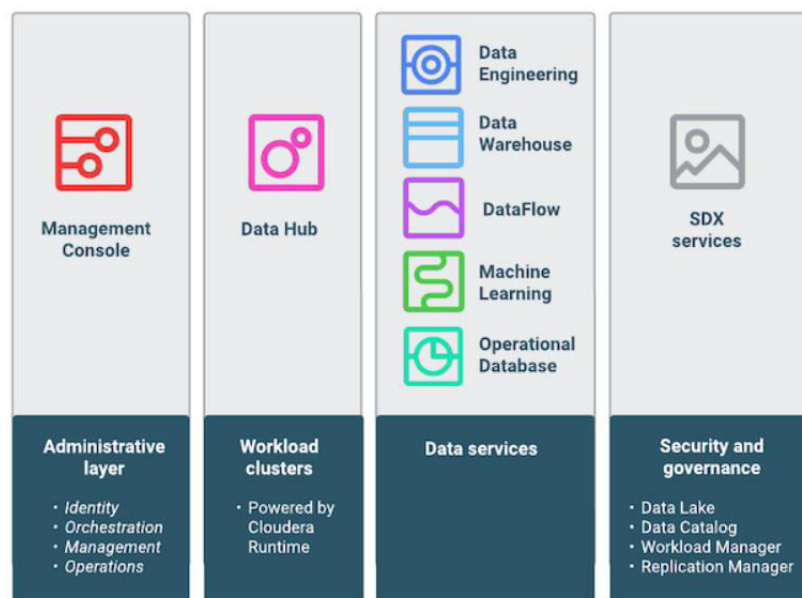


CDP in Public Cloud Examples

- Streaming solutions may enhance data sets and select production data for downstream practitioners. Manage data transport between sources and destinations with real-time data flow control.
- Leverage object stores as centralised storage to bring together disparate datasets, enhance and analyse utilising CDP analytical engines, and create a customer 360 use case.
- Collect measurements from process and discrete manufacturing systems to record, model, and notify on deviations before it's too late.
- Run workloads at the proper time and place to decrease computing and storage expenses and prevent cloud lock-in.

CDP Services in Public Cloud

CDP Public Cloud offers cloud services for business data use cases. This contains Cloudera Data Hub, data services (Data Warehouse, Machine Learning, Data Engineering, and DataFlow), the administrative layer (Management Console), and SDX services (Data Lake, Data Catalog, Replication Manager, and Workload Manager).



Administrative layer

Management Console is used by CDP administrators to administer, monitor, and coordinate all CDP services across all environments. You can create, monitor, provision, and delete services for data centre and public cloud deployments in one location.

Workload Cluster

Data Hub is a service for deploying and maintaining Cloudera Runtime-powered workload clusters. This contains cloud-optimized templates for typical workload kinds and capabilities for considerable customisation. Data Hub enables total workload isolation and full elasticity so any task, application, or department may have its own cluster with a separate software version, configuration, and infrastructure. These speeds up development. Data Hub clusters are straightforward to deploy and have an automated lifetime, so you can construct them on demand and return the resources to the cloud when you're done.

Data Services

- **Data Engineering** is an all-inclusive data engineering solution built on Apache Spark that automates ETL procedures using Apache Airflow, enhanced pipeline monitoring, visual debugging, and extensive management features.
- **DataFlow** is a cloud-native, Apache NiFi-powered universal data distribution service that allows developers connect to any data source, analyse it, and distribute it to any destination. It provides a flow-based low-code development methodology that corresponds with how developers build data pipelines.
- **Data Warehouse** helps data analysts create and manage self-service data warehouses. This service makes it simple to setup a new data warehouse and share a subset with a team or department. The ephemeral service allows you to swiftly construct and end data warehouses.
- **Machine Learning** creates and manages self-service workplaces. This allows data scientists to construct, test, train, and deploy machine learning models using business and cloud data.
- **Operational Database** allows self-service database construction. Operational Database uses Apache HBase and Apache Phoenix. You may utilise the same storage and access layers from CDH and HDP.

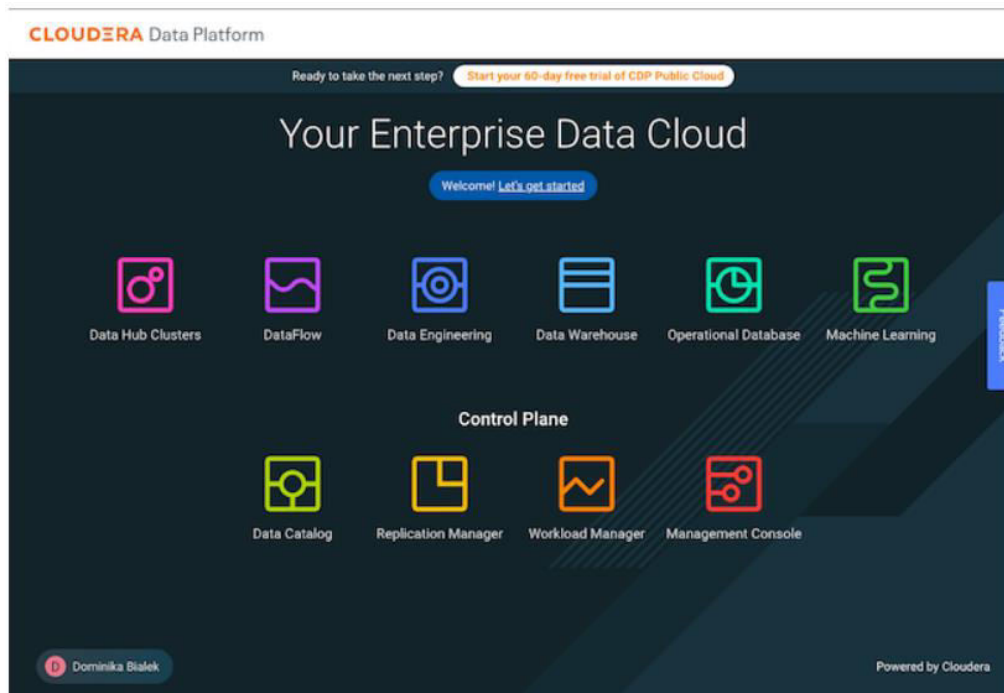
Security and Governance

- **Shared Data Experience (SDX)** is a package of technologies that allows organisations to bring all their data into one place for safe and managed sharing. Data Lake, Data Catalog, Replication Manager, and Workload Manager are SDX services.
- **Data Lake** offers a protective ring around data saved in cloud object storage or HDFS. Management Console and Cloudera Runtime absorb Data Lake capability (Ranger, Atlas, Hive MetaStore).
- **Data Catalog** organises, secures, and governs business cloud data. Data Catalog is used by data stewards to browse, search, and tag a data lake's content, develop and maintain permission rules (by file, table, column, row, etc.), and track a data set's history.
- **Replication Manager** copies, migrates, snapshots, and restores corporate data cloud data. This service moves, copies, backups, replicates, and restores data in or across data lakes. For backup, disaster recovery, migration, or dev/test in another virtual environment.
- **Workload Manager** optimises business cloud workloads. Administrators use this service to diagnose, analyse, and optimise workloads to enhance performance and/or cost.

Accessing CDP on Public Cloud

Web interface, CLI client, and SDK are the main methods to utilise CDP Public Cloud.

Web Interface: CDP Public Cloud's web interface is graphical. As an admin user, you may register environments, manage users, and supply CDP service resources for end users. End users may utilise the web console to access CDP service web interfaces for data engineering or analytics.



Command Line Interface

You may download and setup the CDP client to use the CDP CLI tool in a terminal window. CDP CLI lets you do web console activities. It also automates cluster construction.

SDK (Software Development Kit)

CDP SDK for Java integrates CDP services into applications. Connect to CDP services, construct and manage clusters, and perform tasks from your Java application or other data integration tools.

CDP On AWS

CDP Control Plane Regions: Cloudera-operated service including Management Console, Workload Manager, Replication Manager, and Data Catalog. These services interface with your Amazon Web Services (AWS), Microsoft Azure, and Google Cloud accounts to deploy and manage computing infrastructure for managing cloud data lifecycles. The Control Plane can also enable hybrid cloud deployments by integrating with on-premises and Private Cloud infrastructure.

Each Control Plane and account operates in an area. If your company needs your CDP account and its data to stay in a specified area, consider the regions below to setup your CDP account and Control Plane.

Every CDP account uses the CDP Control Plane. CDP Control Plane and its services were formerly hosted at us-west-1.

Certain nations may restrict or prohibit sending or storing certain kinds of data beyond their borders. Cloudera has so added additional Control Plane areas. By picking a region other than us-west-1, you may prevent sensitive metadata from leaving its area or nation.

During CDP account creation, your administrator will pick the Control Plane area. Your CDP account, users, groups, and CDP resources like environments, Data Lakes, and Data Hubs are related to their Control Plane area. Control Plane and CDP accounts cannot be changed after creation.

Regions	us-west-1	eu-1	ap-1
Location	United States	Germany	Australia
Year Opened	2019	2021	2021

As additional Control Plane areas are introduced, not every CDP feature or data service will be accessible, but they will be supported for every region. Refer to the following support matrix to identify whether CDP data services and significant features are available in your CDP Control Plane's region:

CDP service	us-west-1 (USA)	eu-1 (Germany)	ap-1 (Australia)
Data Engineering	GA	GA	GA
Data Hub	GA	GA	GA
Data Warehouse	GA	GA	GA
DataFlow	GA	N/A	N/A
Machine Learning	GA	GA	GA
Operational Database	GA	GA	GA
Replication Manager	GA	GA	GA
Workload Manager	GA	N/A	N/A
Data Catalog	GA	N/A	N/A

CDP feature	us-west-1 (USA)	eu-1 (Germany)	ap-1 (Australia)
Auditing	GA	GA	GA
CCMv1	GA	N/A	N/A
CCMv2	GA	GA	GA
Classic Clusters	GA	GA	GA
Unified Diagnostics	GA	N/A	N/A
Metering	GA	GA	GA

Though there are few observable variations across Control Plane regions, it's necessary to set up outbound network access effectively. Control Plane API and UI endpoints vary by area.

The name of a Control Plane's region is used in CRNs (the ID CDP uses to designate a resource, such as an environment, user, or cluster), API endpoints, and more, however CDP documentation commonly uses "us-west-1."

Required Permissions on AWS

As a CDP administrator, you may do the following in AWS:

- Create policies and roles in IAM.
- Create and administer VPCs, subnets, security groups, and S3 buckets.
- Perform administrative tasks in various AWS services such as EC2 and CloudFormation.
- AWS Administrator credentials allow creating CDP resources.

The following AWS resources are used by CDP and CDP services.

- AWS resources created for a CDP environment
- When a CDP environment is created, a FreeIPA cluster and a Data Lake cluster are created.

Resource	Description
Virtual Private Cloud (VPC)	If during environment creation you select to have a new VPC and subnets created, then the new VPC and subnets are created on your AWS account. Alternatively, you can provide your own existing VPC and subnets. In both cases, all the resources that CDP provisions for the environment are provisioned into this specific VPC. For example, the EC2 instances provisioned for Data Hub or Data Warehouse are provisioned into that VPC.
Identity and Access Management (IAM)	The cross-account IAM policy that you provided as your credential allows CDP to obtain an access and secret key from AWS, allowing CDP to create resources for your environment and for CDP services such as Data Hub, Data Warehouse, and Machine Learning on your AWS account.
CloudFormation	During environment creation, CloudFormation stack is provisioned for FreeIPA to create required resources. This generates an AWS stack which links and describes the resources of your FreeIPA server. Multi-AZ deployments do not use a CloudFormation template for VM creation. Neither autoscaling groups or launch templates are created. The cluster resources are managed individually using AWS native components (for example, EC2 instances).
Elastic Compute Cloud (EC2)	During environment creation, two or three m5.large EC2 instances are provisioned for the FreeIPA HA server by default. The number of instances depends on the selected Data Lake type. Furthermore, security groups with the rules specified during environment creation are provisioned to define inbound and outbound access to the instances.

In addition, the following resources are created for each Data Lake (one per environment):

Resource	Description
CloudFormation	A CloudFormation stack is provisioned for your Data Lake to create instances, disks, and RDS required. This generates an AWS stack which links and describes the resources of your Data Lake cluster. Multi-AZ deployments do not use a CloudFormation template for VM creation. Neither autoscaling groups or launch templates are created. In a multi-AZ setup the cluster resources are managed individually using AWS native components (for example, EC2 instances).
Elastic Compute Cloud (EC2)	<p>EC2 instances with attached storage are provisioned for the Data Lake nodes:</p> <ul style="list-style-type: none"> • Light duty: Two instances are provisioned: One <code>t3.medium</code> instance (IDBroker) and one <code>m5.2xlarge</code> instance (Data Lake Master node). • Medium duty: Ten instances are provisioned: Two <code>t3.medium</code> instances (IDBroker), three <code>m5.xlarge</code> instances (two Data Lake Master nodes and one Auxiliary node), and five <code>m5.2xlarge</code> instances (three DataLake Core nodes and two Gateway nodes). <p>Furthermore, security groups with the rules specified during environment creation are provisioned to define inbound and outbound access to the instances.</p>
Relational Database Service (RDS)	An RDS instance (<code>db.m5.large</code>) is provisioned for the Data Lake. This RDS instance is used for Cloudera Manager, Ranger, and Hive MetaStore.
Simple Storage Service (S3)	The existing S3 that you provide during environment creation for the Data Lake is used for Data Lake log storage and workload data storage.
DynamoDB (Only when using Runtime older than 7.2.2)	<p>S3 storage is eventually consistent, so for example file listing on S3 might miss entries that were only created very recently. To work around the eventual consistency issues, CDP uses S3Guard, a hadoop extension that stores file names in a DynamoDB table. S3Guard can therefore return the expected file listings, without really having to query the S3 content. One DynamoDB table is provisioned for S3Guard during Data Lake provisioning.</p> <p>As of June 21 2021, S3Guard is no longer enabled when using Runtime 7.2.2 or newer, so any environments deployed after this date with Runtime 7.2.2 or newer don't require creating a DynamoDB table.</p>

AWS resources created for Data Hub

The following AWS resources are created for the Data Hub service:

Resource	Description
CloudFormation	A CloudFormation stack is created for each Data Hub cluster to create instances and disks. This generates an AWS stack which links and describes the resources of your Data Hub cluster. Multi-AZ deployments do not use a CloudFormation template for VM creation. Neither autoscaling groups or launch templates are created. In a multi-AZ setup the cluster resources are managed individually using AWS native components (for example, EC2 instances).
Elastic Compute Cloud (EC2)	An EC2 instance is created for each cluster node. The instance type varies depending on what you selected during Data Hub cluster creation. For each instance, attached storage is provisioned. The storage size and type varies depending on what you selected during cluster creation. Furthermore, security groups with the rules specified during environment creation are provisioned to define inbound and outbound access to the instances. For a list of supported EC2 instance types.
Relational Database Service (RDS)	Data Hub connects to the Hive MetaStore database on the RDS instance provisioned for the Data Lake.
Simple Storage Service (S3)	The existing S3 bucket that you provided for the Data Lake to use for workload data storage can be accessed from Data Hub clusters via the S3A connector.
Auto Scaling	Data Hub uses the Auto Scaling service for upscaling clusters, except in a multi-AZ deployment.
Key Management Service (KMS)	Data Hub uses KMS for encrypting your disks if during Data Hub cluster creation, you select to use disk encryption.

AWS resources created for Data Engineering

The following AWS resources are created for the Cloudera Data Engineering (CDE) service:

Resource	Description
CloudFormation	The initial deployment of services such as the EKS cluster is orchestrated through CloudFormation. This generates an AWS stack which links and describes the resources of your CDE cluster.
Elastic Compute Cloud (EC2)	CDE uses EC2 instances as cluster nodes. For a list of supported EC2 instance types.
Auto Scaling	CDE uses AWS AutoScaler to add or remove EC2 instances to the kubernetes cluster. Whenever the kubernetes cluster is running low on resources, new EC2 instances are provisioned and jointed into the EKS cluster. Whenever the

Resource	Description
	AutoScaler detects an over-provisioning of resources, it removes and suspends EC2 instances.
Elastic Kubernetes Service (EKS)	EKS is the AWS implementation of the kubernetes stack. All PODs are running within an EKS cluster (one per environment).
ELB Classic Load Balancer	CDE uses Classic Load Balancers for redirecting traffic to EC2 instances.
Key Management Service (KMS)	CDE uses KMS for encrypting your disks if you select to use disk encryption.
Elastic Block Store (EBS)	CDE uses EBS for persistent instance storage.
Elastic File System (EFS)	CDE uses EFS for persistent service and virtual cluster storage.
Relational Database Service (RDS)	CDE uses RDS for provisioning relational databases.

AWS resources created for DataFlow

The following AWS resources are created for the DataFlow (DF) service:

Resource	Description
CloudFormation	The initial deployment of services such as the EKS cluster is orchestrated through CloudFormation. This generates an AWS stack which links and describes the resources of your DataFlow cluster.
Elastic Compute Cloud (EC2)	DataFlow uses EC2 instances as cluster nodes. For a list of supported EC2 instance types.
Auto Scaling	DataFlow uses AWS AutoScaler to add or remove EC2 instances to the kubernetes cluster. Whenever the kubernetes cluster is running low on resources, new EC2 instances are provisioned and jointed into the EKS cluster. Whenever the AutoScaler detects an over-provisioning of resources, it removes and suspends EC2 instances.
Elastic Kubernetes Service (EKS)	EKS is the AWS implementation of the kubernetes stack. All PODs are running within an EKS cluster (one per environment).

Resource	Description
ELB Classic Load Balancer	DataFlow uses Classic Load Balancers for redirecting traffic to EC2 instances.
Elastic Block Store (EBS)	DataFlow uses EBS for persistent instance storage.
Relational Database Service (RDS)	DataFlow uses RDS for provisioning relational databases.

AWS resources created for Data Warehouse

The following AWS resources are created for the Cloudera Data Warehouse (CDW) service:

Resource	Description
Identity and Access Management (IAM)	During CDW cluster provisioning, the DW service creates an IAM role that defines access to S3 and other provisioned resources. Such role is then attached to the EC2 instance profile to grant PODs within the kubernetes environment access to these resources.
Certificate Manager	CDW creates, stores, and maintains a certificate in the AWS certificate manager. This certificate is used to allow HTTPS connections to the external facing endpoints (i.e. for JDBC or the DAS UI). The certificate is signed by a trusted certificate authority, therefore external consumers and browser can securely connect to DW services without having to deal with untrusted CA or self-signed certificates.
CloudFormation	The initial deployment of services such as the EKS cluster, the CDW-specific RDS database, and S3 buckets is orchestrated through CloudFormation. This generates an AWS stack which links and describes the resources of your DW cluster.
Elastic Compute Cloud (EC2)	CDW uses EC2 instances as cluster nodes. Two different EC2 instance types (through two different auto scaler groups) are used to support shared services and compute requirements within the cluster: m5.2xlarge for always on components, and r5d.4xlarge for compute nodes (Hive and Impala executors). Furthermore, security groups with the rules specified during environment creation are provisioned to define inbound and outbound access to the instances. For a list of supported EC2 instance types.
Simple Storage Service (S3)	CDW creates its own S3 buckets (separate from the environment's S3 bucket(s)) for storing data and logs.

Resource	Description
Auto Scaling	CDW uses AWS AutoScaler to add or remove EC2 instances to the kubernetes cluster. Whenever the kubernetes cluster is running low on resources, new EC2 instances are provisioned and jointed into the EKS cluster. Whenever the AutoScaler detects an over-provisioning of resources, it removes and suspends EC2 instances.
Elastic File System (EFS)	EFS is used as shared filesystem across PODs to persist data (i.e. result cache).
Elastic Load Balancing (ELB)	All inbound traffic is routed through ELB towards the ingress controller of the kubernetes cluster. The ELB is provisioned as a result of the kubernetes ingress controller, which is the single point of entry for services, running in the kubernetes cluster.
Managed Kubernetes Service (EKS)	EKS is the AWS implementation of the kubernetes stack. All DW-deployed PODs are running within an EKS cluster (one per environment).
Key Management Service (KMS)	CDW encrypts data at rest in S3. This requires an encryption key to be generated and stored in KMS. The key is completely under the control of AWS and cannot be exported or otherwise extracted. The S3 buckets are directly referencing the key within KMS, using it to encrypt the stored data.
Relational Database Service (RDS)	During cluster provisioning, DW provisions an RDS instance to be used as backend database system for metadata, managed and stored by the HMS instances, represented by "DB Catalogs". Each DB Catalog is implemented as separate database within this single RDS instance.
Security Token Service (STS)	STS is used to generate access tokens (based on roles) to access the resources within the environment's VPC.

AWS resources created for Machine Learning

The following AWS resources are created for the Cloudera Machine Learning (CML) service:

Resource	Description
Identity and Access Management (IAM)	CML creates additional IAM roles and policies for each cluster. Such roles are then attached to the EC2 instance profile.

Resource	Description
Elastic Block Store (EBS)	CML uses EBS as block storage.
Elastic Load Balancer (ELB)	CML uses Classic Load Balancers for redirecting traffic to EC2 instances.
Key Management Service (KMS)	CML uses KMS for encrypting your disks if you select to use disk encryption.
Elastic File System (EFS)	EFS is used for project file storage.
Elastic Compute Cloud (EC2)	CML uses EC2 instances as cluster nodes. Three different EC2 instance types (through three different auto scaler groups) are used to support CML infra and compute requirements within the kubernetes cluster. Furthermore, security groups with the rules specified during environment creation are provisioned to define inbound and outbound access to the instances. For a list of supported EC2 instance types.
Auto Scaling	CML uses AWS AutoScaler to add or remove EC2 instances to the kubernetes cluster. Whenever the kubernetes cluster is running low on resources, new EC2 instances are provisioned and jointed into the cluster. Whenever the AutoScaler detects an over-provisioning of resources, it removes and suspends EC2 instances.
Simple Storage Service (S3)	CML uses S3 as the primary store for data and logs.
Security Token Service (STS)	STS is used to generate access tokens (based on roles) to access the resources within the environment's VPC.
Managed Kubernetes Service (EKS)	EKS is the AWS implementation of the kubernetes stack. All PODs are running within an EKS cluster (one per environment).

AWS resources created for Operational Database

The following AWS resources are created for the Cloudera Operational Database (COD) service:

Resource	Description
CloudFormation	A CloudFormation stack is created for each COD database to create instances and disks. This generates an AWS stack which links and describes the resources of your COD database.

Resource	Description
Elastic Compute Cloud (EC2)	An EC2 instance is created for each node. The instance type, storage size, and storage type is determined automatically by COD. Furthermore, security groups with the rules specified during environment creation are provisioned to define inbound and outbound access to the instances.
Simple Storage Service (S3)	The existing S3 bucket that you provided for the Data Lake to use for workload data storage can be accessed from COD database via the S3A connector.

AWS Resources and Services used for CDP

AWS resources used by CDP Follow these steps to verify CDP has access to your AWS account's resources and that your account has all the CDP-required resources:

AWS Region

Choose an AWS region before registering an environment. CDP deploys all AWS resources into a single VPC in a given region for each AWS environment registered in CDP.

You should install clusters in the same area as your S3 input and output buckets. Consider where your data is while choosing a region. CDP needs the S3 storage location to be in the same region as the environment. Numerous areas need multiple settings, one per region.

VPC and subnets

You must pick a VPC and two or more subnets when registering an AWS environment in CDP.

Two choices:

- Provision CDP resources using your VPC and subnets.
- Create a VPC and subnets using CDP. This new VPC and subnets will house all CDP resources.

New VPC and Subnets:

If you choose to allow CDP to create a new VPC, six subnets will be created automatically. One subnet is created for each availability zone assuming three AZs per region; If a region has two AZs instead of three, then still three subnets are created, two in the same AZ.

You will need to specify a valid CIDR in IPv4 range that will be used to define the range of private IPs for EC2 instances provisioned into these subnets. Default is 10.10.0.0/16. Consider changing the IP range to correspond to corporate policies for standardized IP address ranges. The CIDR must match the <network mask>/16 pattern.

By default, CDP creates 6 subnets (3 private and 3 public) and divides the address space as follows:

- 3 x /19 private subnets for FreeIPA, Data Lake, Data Hub, Data Warehouse, Machine Learning
- 3 x /24 public subnets

You can disable creating private subnets, in which case only 3 public subnets will be created.

By default, when creating a new network, CDP uses public endpoints. But during environment registration you can optionally select the “Create Private Endpoints” option to use private endpoints instead of public endpoints.

Security

Security groups control incoming and outgoing CDP traffic. To grant your organization's users access to CDP resources, employ security group settings. Security groups control incoming and outgoing CDP traffic. To grant your organization's users access to CDP resources, employ security group settings.

Two choices:

- Utilize security groups (recommended for production)
- Create CDP security groups

If you want CDP to construct security groups for you, offer a CIDR range for your organization's incoming EC2 traffic. CDP provides numerous security groups for Data Lake, FreeIPA, DataFlow, Data Hub, Data Warehouse, and Machine Learning clusters.

SSH keys

You must supply a matching SSH public and private key when registering an environment. 2048-bit SSH keys are minimal. When registering an environment, you will be asked to provide an SSH public key for which you have a matching private key. The minimum SSH key size is 2048 bits. The SSH public key will be used for root-level access to Data Lake and Data Hub instances. Only those users who have the corresponding private key would be able to login as an admin user.

EC2 Instances

CDP creates Data Lake, FreeIPA, and computing clusters using EC2 instances. CDP creates Data Lake, FreeIPA, and computing clusters using EC2 instances. Verify the quantity and kind of EC2 instances in your AWS account to establish CDP environments and clusters. CDP supports Amazon EC2 reserved instances; if you buy them, CDP automatically utilizes them per AWS rules. Using IMDSv1 or IMDSv2, you may retrieve EC2 instance information on AWS. CDP supports IMDSv1 but not IMDSv2, thus don't activate IMDSv2 on CDP EC2 instances.

CDP uses default images for all provisioned VMs, however you may use custom images for Data Lake, FreeIPA, and Data Hub. You may need a custom image for compliance or security (a "hardened" image) or to pre-install monitoring tools or applications.

Create a cross-account access IAM role

In your AWS account and add CDP as a trusted principal by giving an AWS account and external ID. Create a cross-account access IAM role in your AWS account and add CDP as a trusted principal by giving an AWS account and external ID.

The cross-account access IAM role policy must have the below-linked permissions. AWS account ID and external ID must be referenced in the IAM role.

S3 bucket, IAM roles, backups, and data storage

CDP needs an S3 bucket for workload data and logs. You must also build IAM roles and rules that provide S3 bucket access.

The S3 bucket is used for:

- **Storage location base** - Workload data storage and Ranger audits
- **Log's location base** - Service logs, FreeIPA logs
- **Backup location base** - FreeIPA and Data Lake backups

The S3 bucket must be in the same region as the environment.

DynamoDB

You must pick a DynamoDB table for S3Guard when enrolling an AWS environment in CDP.

Keys Managed by customers

By default, Data Lake and FreeIPA's Amazon EBS volumes and RDS are encrypted using Amazon's KMS, but you may specify Customer Managed Keys (CMK). Data Hubs inherit the environment's

Amazon lets you encrypt EBS volumes and RDS instances using Amazon's KMS or a customer-managed KMS. By default, Data Lake and FreeIPA are encrypted using Amazon's KMS default key in the environment's region, but you may supply a customer-managed KMS key instead. Block and root devices are encrypted. When encryption is set for a cluster, it's immediately applied to any new VM instances added due to scaling or repair.

AWS restrictions

AWS restricts your resources when you establish an account. Region-specific constraints apply.

Accessing to Workload UIs in AWS

If you have limited DNS or networking, ensure sure *.cloudera.site can be resolved from your network so employees can use workload UIs.

Subdomains under cloudera.site host CDP UI endpoints (including Data Lake) (Cloudera Manager, Ranger, Knox, Hue and so on). When a Data Lake, Data Hub, or other workload (such as Virtual Warehouse in CDW) is formed, CDP automatically supplies these endpoints and sets up routing so you can access them from your network.

```
<endpoint-name>.env-truncated-name>.customer-workload-subdomain>.regional-  
subdomain>.cloudera.site
```

CDP CIDR

CDP CIDR includes the following IP ranges, when creating your own security groups for CDP, you must open required ports to all of these IP ranges.

Control Plane Region	IP Ranges
us-west-1	35.80.24.128/27, 35.166.86.177/32, 52.36.110.208/32, 52.40.165.49/32
eu-1	3.65.246.128/27
ap-1	3.26.127.64/27

CDP Public Cloud reference network architecture for AWS

Customers may build up cloud Data Lakes and compute workloads on AWS, Azure, and Google Cloud. It bridges a cloud account to the environment where compute workload clusters (Data Hubs) and data services (such as Cloudera Data Engineering (CDE), Cloudera Data Warehouse (CDW), Cloudera Machine Learning (CML), Cloudera Operational Database (COD), Cloudera DataFlow (CDF)) are launched. For Data Lakes, computing workload clusters, and data services to perform properly, numerous cloud architecture aspects must be configured: access rights, networking configuration, cloud storage, etc. This may be done in two ways:

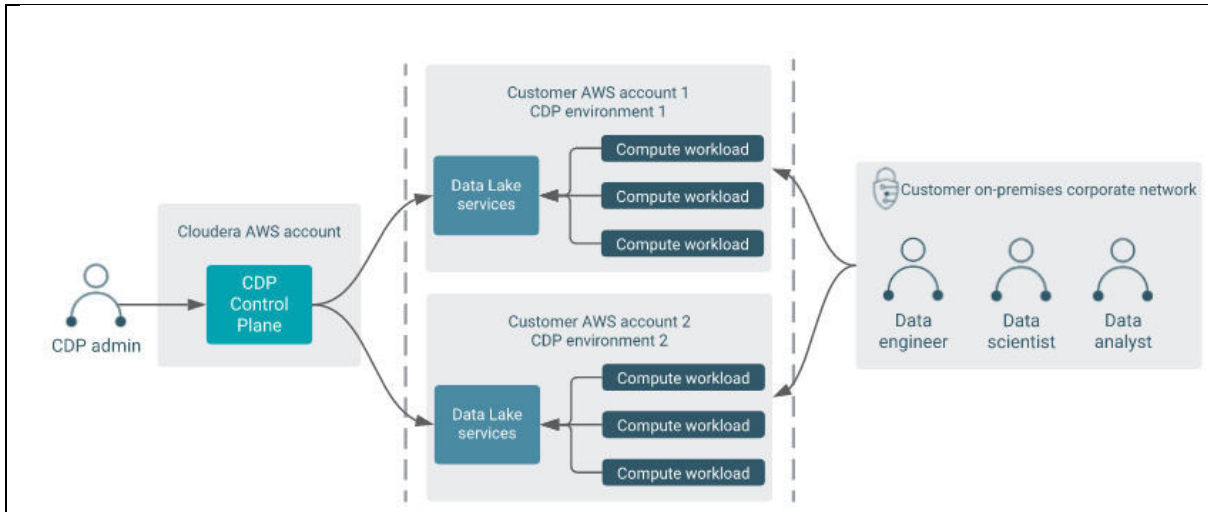
1. **CDP may build up these consumer elements:** This methodology helps easily build up a CDP environment. Many business clients want specialised cloud setups for Infosec or compliance. Setting up networking and cloud storage required clearance, but they wouldn't actively block a third-party provider like Cloudera from doing so.
2. **Customer-created components may be used in CDP:** In this concept, cloud creation flows Data Lakes uses pre-created cloud settings to deploy applications. This approach meets business needs. However, the arrangement may not meet CDP criteria. As a consequence, clients may have trouble deploying CDP workloads, and getting to a functional environment may take longer and entail numerous tiresome contacts between Cloudera and customer cloud teams.

CDP Public Cloud lets businesses handle data in a safe, controlled Data Lake utilising Data Hub or data services. Typical workload lifecycle:

- A CDP admin sets up their cloud environment. This creates a cloud Data Lake cluster, FreeIPA cluster, and identity provider for this environment. CDP admin may need to engage with a cloud administrator to generate cloud provider resources (including networking resources).
- Then, connected computing workload clusters may be launched. Each workload cluster has a distinct function, such data intake, analytics, or machine learning.
- Data engineers, analysts, and scientists access these computational clusters. This is why CDP is on the public cloud.
- These computational workload clusters might be long-running or transitory.

Two sorts of CDP consumers utilise it for distinct reasons:

- **CDP administrators:** deploy and maintain the cloud environment, Data Lake, Data Hubs, FreeIPA, and CDP data services. They manage the infrastructure using a Cloudera AWS Management Console.
- **Data consumers:** utilise Data Hubs and data services to process data. They connect directly with cloud-based computational workloads (Data Hubs and data services). They might access them via their business networks (through VPN) or other cloud networks their company controls.



CDP Cloudera Data Analyst
CDP 4001
 Q & A
CLUSTERA
 CDP Cloudera Data Analyst
CDP 4001

CDP Administrator Private Cloud Base
 Exam-CDP-2001
 Q & A
CLUSTERA
 CDP Administrator Private Cloud Base
 Exam- CDP-2001

CDP Cloudera Data Developer Exam
 CDP-3001
 Q & A
CLUSTERA
 CDP Cloudera Data Developer Exam
 CDP-3001

Cloudera CDP Generalist
CDP-0011
 Q & A
CLUSTERA
 Cloudera CDP Generalist Exam
 CDP-0011