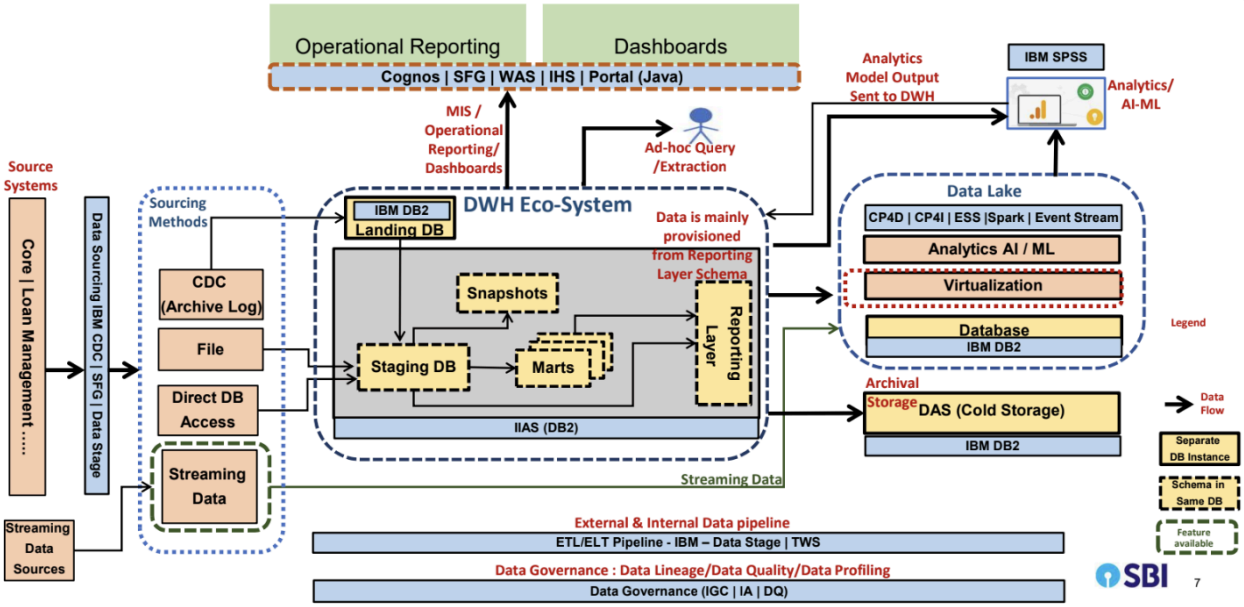
Current Architecture



The above shown architecture represents the current state of SBI's Data Warehouse and Data Lake setups and how the information flow today happens in the bank.

Below are some of the observations.

- The above architecture clearly indicates that there are multiple data stores in the current setup, DWH, Archival & Data Lake
- Current setup does not allow for unified centralized data access patterns as the data is current moved separately to DWH and Data Lake platforms
- The above pattern often leads to duplication of data between the different systems
- It can also cause unavailability of data especially when we need to make combined use of data sitting in DWH & Data Lake
- The Archival solution is also completely isolated, which can cause delays in accessing historical data when needed.

Capabilities for an Integrated Platform

An integrated Data Warehouse and Data Lake platform should have below properties to make best use of the infrastructure and Data for timely actionable insights

- **Unified Data Platform**, All data irrespective of structured or otherwise, Hot, Cold or warm, ingested in batches or via streaming, used for reporting or for Machine Learning should be able to store in one single platform

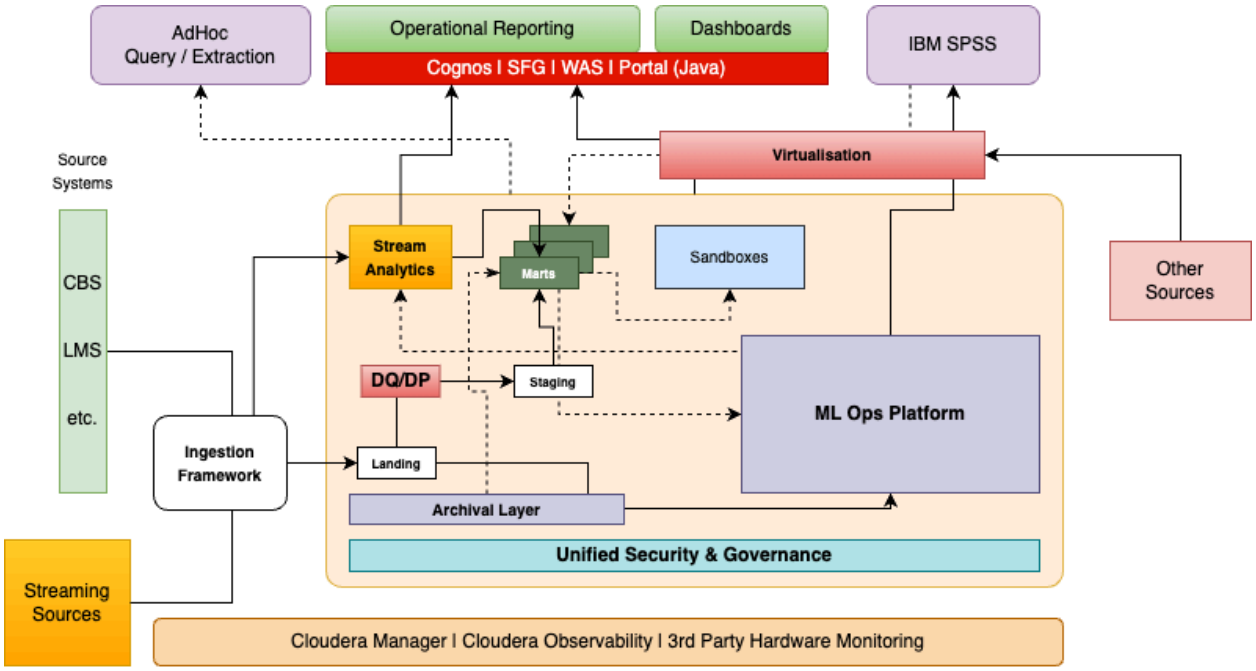
- **Unified Security & Governance**, All data stored in the said unified data platform should also have a unified security and governance framework, meaning the data might be stored, processed and queried using various technologies but should be able to control security (Authorization) from one single app and similarly all metadata management, data asset management should be possible from one single framework

- **Open, Modern & Scalable,** The Platform should be open for integrations, support modern data architectures like data fabric/data mesh and should be scalable not only to support 100's of PBs of data but also the use cases

- **Cloud Native and Hybrid,** The platform should support building and deploying cloud native applications on-premises by taking advantage of latest technology advancements like running workloads on container platforms and should possess the ability to extend to cloud where needed

- **Real-time Ingestion and Analytics,** The platform should support ingestion with high throughput and low latency and also enable event based analytics in real time. The analytics will often need support to join in flight stream data with persisted data to deliver meaningful insights.

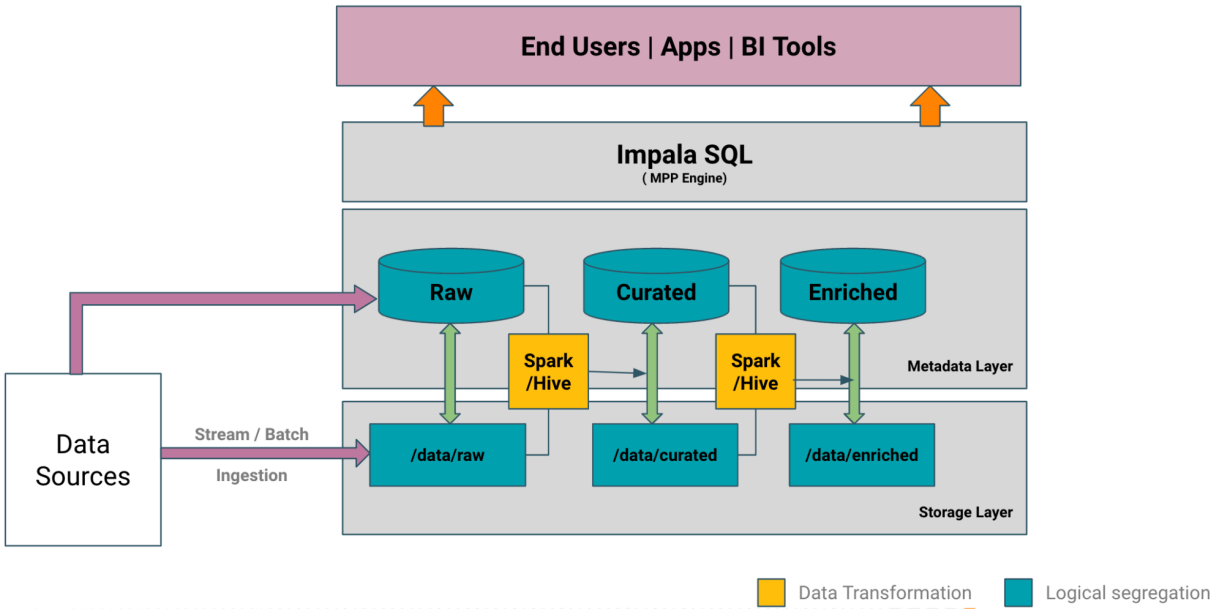
Proposed Architecture



The proposed architecture combines three important core functionalities Data Warehouse, Data Lake and Archival Database from existing architecture into one single central platform. The objective is to have them part of the same platform but have them logically separated so as to reuse data that was written as and when required and also have a single security & Governance framework for all data.

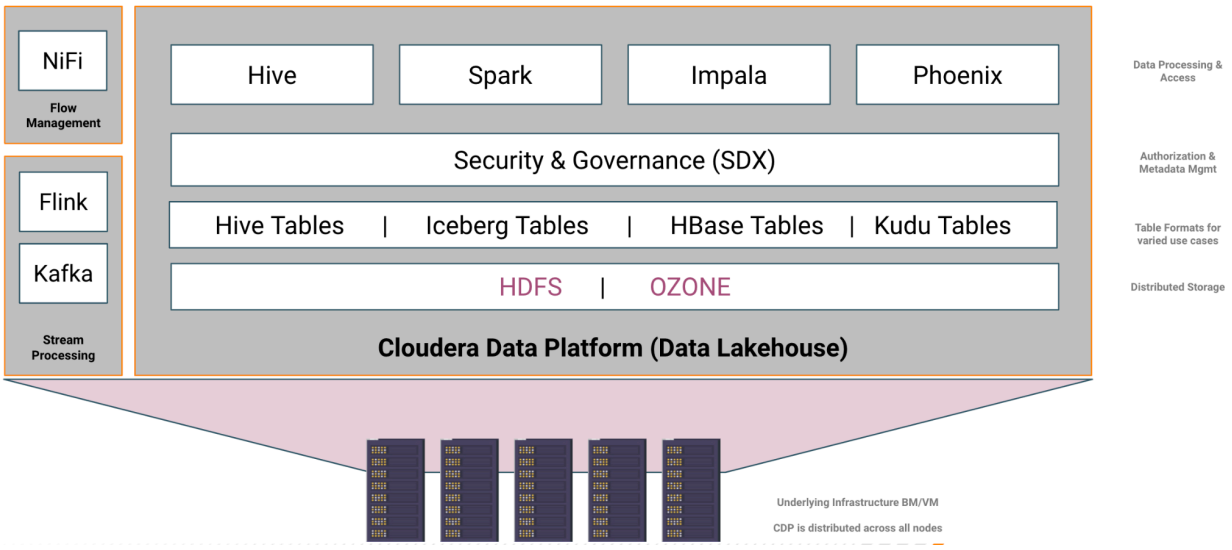
This approach allows for unified and instance access to all kinds of data through a common query framework.

Below is a representation of how we can have logical layers within the storage technology layer and address the landing,staging and datamart layers.



Cloudera Platform Architectures

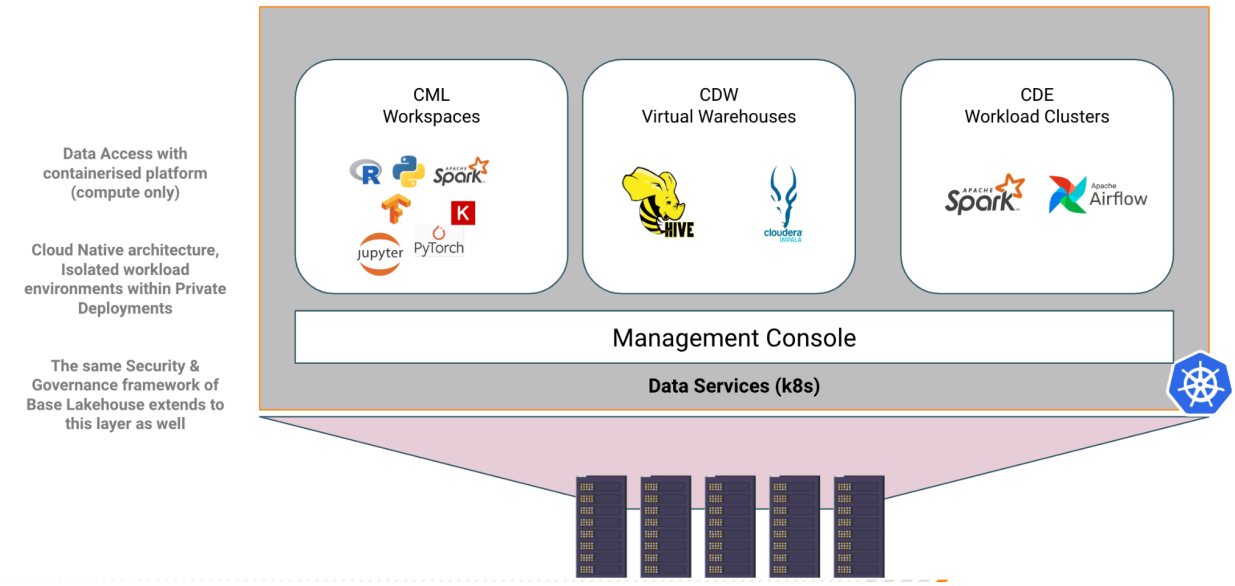
CDP Private Cloud Base



The above diagram depicts how Cloudera Data Platform is logically segregated to support different functionalities within a Data LakeHouse. The platform software runs across a bunch of underlying servers, collectively managing resources with a robust framework which can scale to hundreds of Petabytes.

The platform supports both a distributed storage file system and an object storage which is compatible with Hadoop API & S3 API at the same time uniquely making it the apt one for big data workloads in private cloud environments. The Platform includes Apache Iceberg an open table format for large scale analytics, with multi functional analytics allowing consistency across engines with additional features like partition evolution, time travel etc.

CDP Private Cloud - Data Services



Cloudera Private Cloud Data Services is a unique product allowing customers to deploy cloud native architectures in On-Premises Data centers. PvC Data Services is an extension to the Base platform which serves as a compute layer running on kubernetes cluster. WIth the ability to run different workloads like Machine learning, Data Engineering and Data Warehouse and isolated compute workloads this provides a unique way to create a true cloud like environment within data centers. The same services extend to our public cloud as well providing a easy way to migrate to the public cloud with no code changes as and when needed.

Deployment Considerations

Cloudera Data Platform is a combination of different technology stacks most of which are distributed storage/processing frameworks designed to be fault tolerant to hardware/network failures. The ability to take advantage of these designs largely depends on how they are deployed on host/network level.

We should look at redundancy to avoid any downtimes due to disk/service/host/switch downtimes, which is why Cloudera recommends certain guidelines to efficiently set up a cluster with minimum to no downtimes.

Data Availability

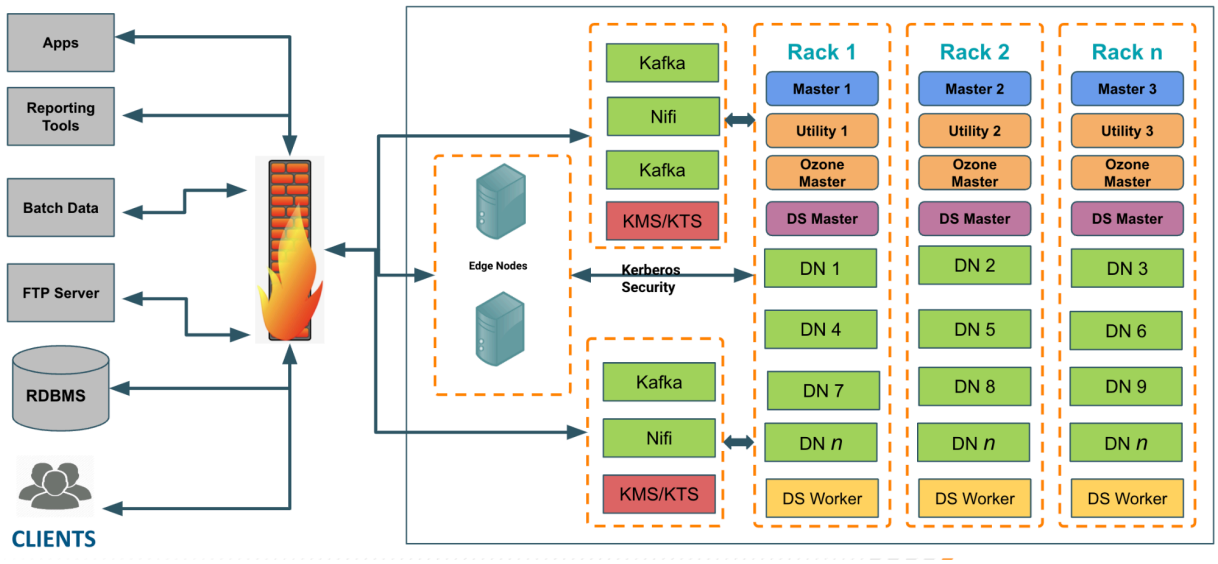
All the storage technologies available in Cloudera Platform are designed to be fault tolerant by default with replication. Replication enables the creation of the same copies of data in different disks/nodes/racks to make sure there is redundancy and any failures of disks, servers or switches will not impact neither data accessibility nor availability.

We can make sure data is distributed to different nodes in different racks to avoid single point of failures by enabling **Rack Awareness** within Cloudera Configuration.

Service Availability

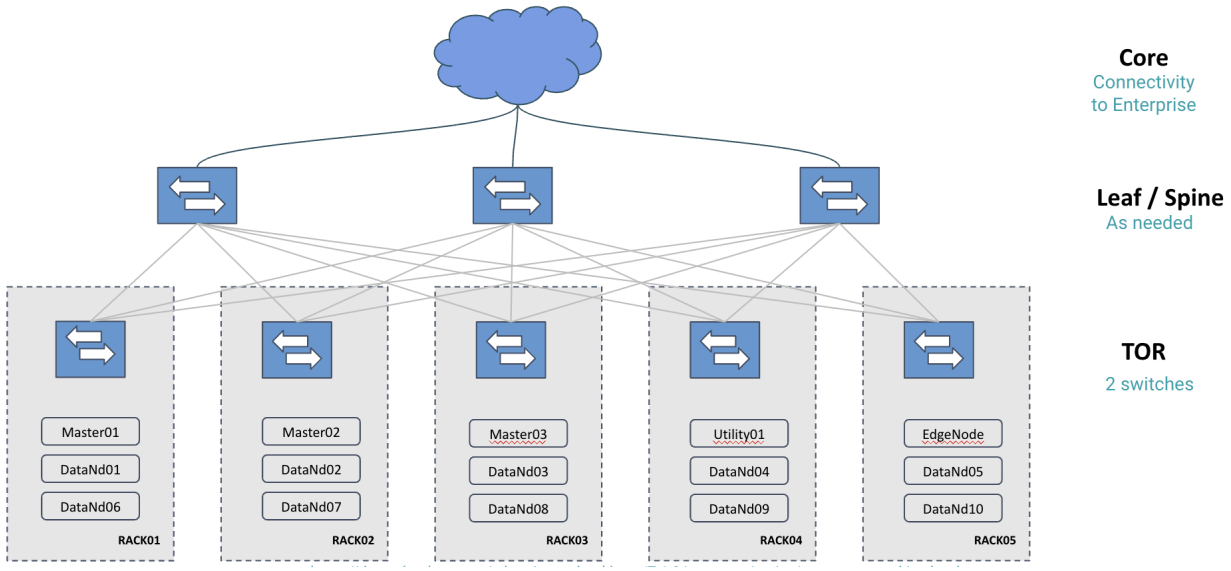
Similar to storage, service availability can also be handled by creating redundancy at service level by enabling HA wherever possible, this comes with additional resource cost. All services within the Cloudera stack allows service redundancy for either a minimum of 2 or more allowing us to eliminate unexpected downtimes.

A simple deployment architecture to visualize the same.



Network Topology

The below topology graphic shows a cluster deployed across several racks (Rack 1, Rack 2, … Rack n). Each host is connected to two top-of-rack (TOR) switches which are in turn connected to a collection of spine switches which are then connected to the enterprise network. This deployment model allows each host to maximize throughput and minimize latency, while encouraging scalability.



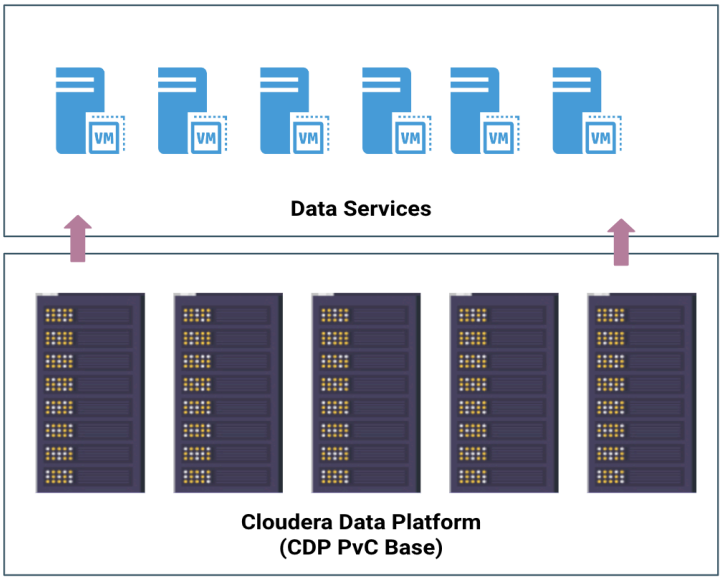
Big Data Platforms can consume all available network bandwidth with inter-node communication as large scale data processing requires data exchange within the nodes . For this reason, Cloudera recommends that the entire data platform be placed in a separate physical network with its **own core switches**.

Infrastructure Choices

Cloudera Data Platform is hardware vendor agnostic and can be deployed on any commodity hardware, given the data and workload scale at SBI we recommend the below approach to make best use of existing infrastructure deployed at the bank. As per the EOI we need to provision at least 4 environments, Production, DR, Dev and UAT.

For **Production Deployment**, based on the EOI functional/technical requirements, Cloudera proposes the deployment of **CDP Private Cloud Base** to host data and build data models etc. preferably on bare metal with certain disk layouts for predictable performance. That being said the platform can run on Virtual Machines with zero loss in functionality, with the right set of configurations the platform can provide the same level of performance on a VM deployment model as well. We also propose the use of a more modern, scalable and cloud native approach to workloads in the form of **CDP Private Cloud Data Services,** a compute only containerized platform which interacts with data from Base platform through a unified security

and governance layer. **These Data Services can be deployed on SBI's Meghdoot platform (on VMs) preferably with RedHat Openshift Container Platform for kubernetes cluster.**



The same deployment model can be considered for DR as well, depending on the availability of Meghdoot/equivalent in the DR location. In addition DR location can also be considered for hosting the needed **SandBoxes** making use of the deployed hardware.

In addition, the Meghdoot platform can also host the lower environments Dev and UAT, considering these environments will not have strict performance/SLA requirements.

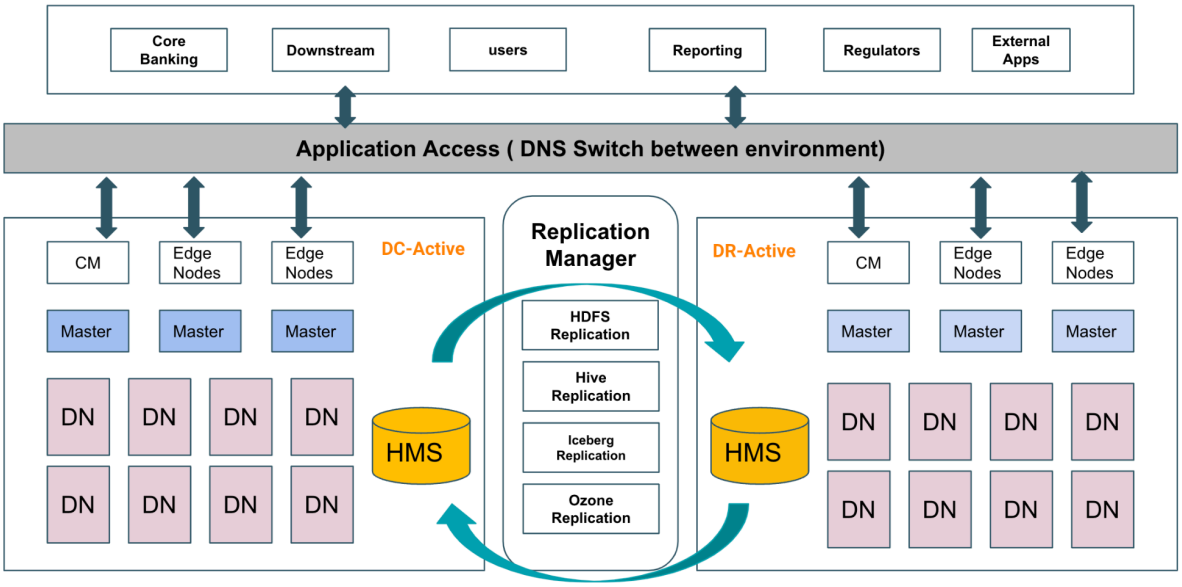
While data services can be deployed on Bare Metal and Virtual Machines, we should definitely consider the pros and cons of this deployment choice.

Metric	Bare Metal	Virtual Machine
Performance	Top performance with local SSD disks	Overall performance is lower than Bare Metal
Flexibility	Setup time is longer	Fastest way to deploy on VMWare Infra if customer

		already has licensing
Cost	Might get costlier considering bare metal procurement	More cost effective if vSphere servers are of high configuration

Disaster Recovery Approach

Cloudera Data Platform provides tools to plan for disaster recovery with reduced RPO/RTO. We understand that SBI would be looking at an equivalent DR site and would need to keep it in sync with the primary site.



The above diagram represents how Cloudera supports keeping the primary and secondary sites in sync using Replication Manager, a framework designed to replicate data and metadata between Cloudera clusters. However, the replication process is asynchronous and is schedule based and not real-time, meaning that the time it takes to replicate to DR depends on how we schedule these syncs and also how long a sync takes for the next one to be triggered. Having dedicated bandwidth between the two sites will help replicate data faster allowing us to reduce the RPO.

The bandwidth can be calculated based on incremental data generated for a certain time period (let's say an hour) and time in which we need it to be available in the target site. In most scenarios the data coming into the data lakehouse is available in sources for reloading as well.

Security Approach

Cloudera's Security Reference architecture reflects the four pillars of security engineering best practice, Perimeter, Data, Access and Visibility. A layered approach is recommended for the highest level of depth in security. Cloudera provides different levers to implement the best security framework for a private deployment.

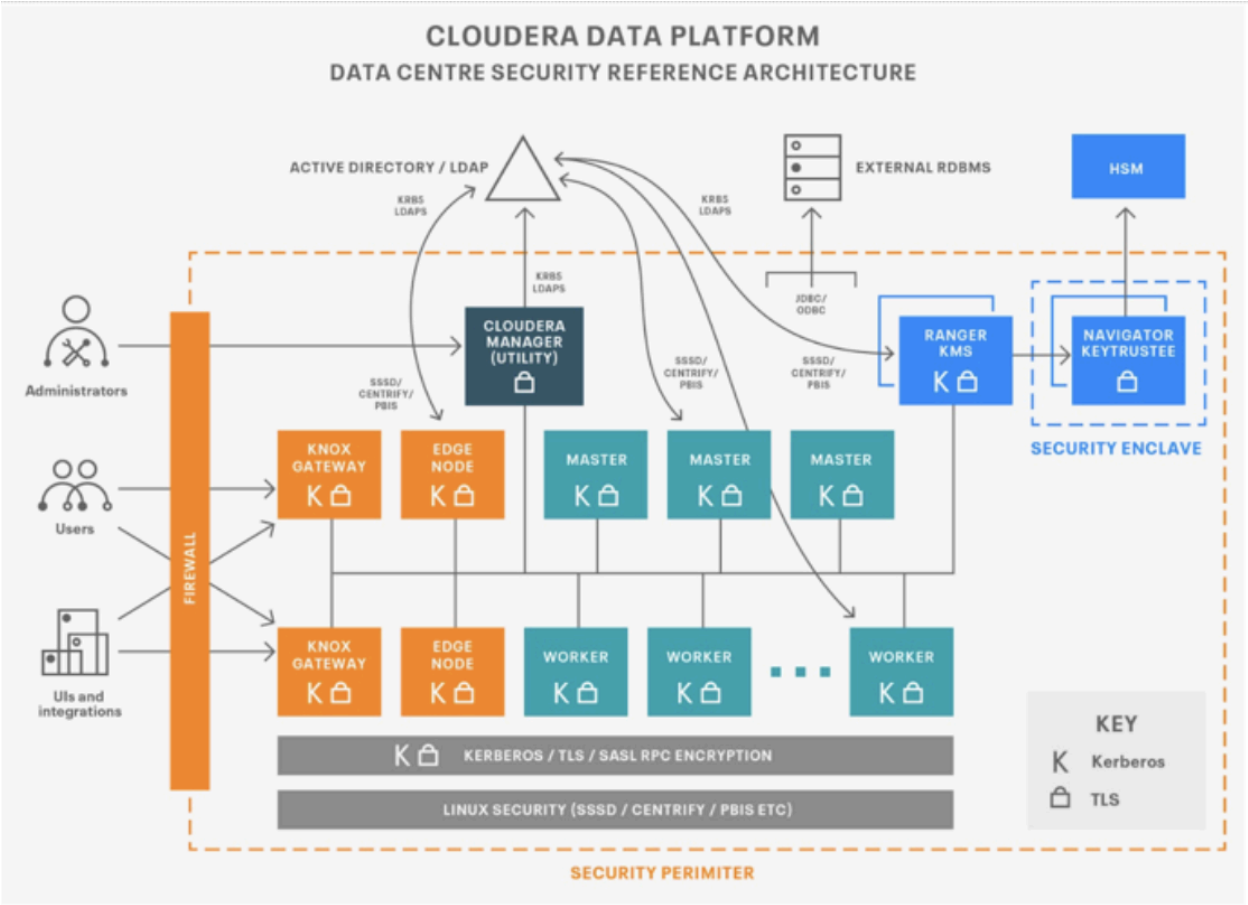
Each layer is defined as follows:

"The most secure cluster is one in which all data, both data-at-rest and data-in-transit, is encrypted and the key management system is fault-tolerant. Auditing mechanisms comply with industry, government, and regulatory standards (PCI, HIPAA, NIST, for example), and extend from the Cluster to the other systems that integrate with it. Cluster administrators are well-trained, security procedures have been certified by an expert, and the cluster can pass technical review"

To comply with the regulatory standards of a most secure implementation, customers will create network topologies that ensure only privileged administrators are able to access core CDP services with applications, analysts and developers limited to appropriate gateway services such as Hue and appropriate management and monitoring web interfaces. The addition of Apache Knox significantly simplifies the provisioning of secure access with users benefiting from robust single sign on. Apache Ranger consolidates security policy management with tag based access controls, robust auditing and integration with existing corporate directories.

Logical Architecture

The cluster architecture can be split across a number of zones as illustrated in the following diagram:



Outside the perimeter are source data and applications, the gateway zones are where administrators and applications will interact with the core cluster zones where the work is performed. These are then supported by the data tier where configuration and key material is maintained. Services in each zone use a combination of kerberos and transport layer security (TLS) to authenticate connections and APIs calls between the respective host roles, this allows authorization policies to be enforced and audit events to be captured. Cloudera Manager will generate credentials either directly against a local KDC or via an intermediary in the corporate directory. Similarly, Cloudera Manager Auto-TLS enables per host certificates to be generated and signed by established certificate authorities. If necessary, authority can be delegated to Cloudera Manager to sign the certificates in order to simplify the implementation.

Authentication

Typically a cluster will be integrated with an existing corporate directory, simplifying credentials management and align with well established HR procedures for managing and maintaining both user and service accounts. Kerberos is used to authenticate all service accounts within the cluster with credentials generated in the corporate directory (IDM/AD) and distributed by Cloudera Manager. To ensure that these procedures are secured it's important that all interactions between CM, the Corporate Directory and the cluster hosts are encrypted using TLS security. Signed Certificates are distributed to each cluster host enabling service roles to mutually authenticate. This includes the Cloudera Agent process which will perform an TLS handshake with the Cloudera Manager server in order that configuration changes such as the generation and distribution of Kerberos credentials are undertaken across an encrypted channel. In addition to the CM agent, all the cluster service roles such as Impala Daemons, HDFS worker roles and management roles typically use TLS.

Kerberos

With Kerberos enabled, all cluster roles are able to authenticate each other providing they have a valid kerberos ticket. The authentication tickets are issued by the KDC, typically a local Active Directory Domain Controller, FreeIPA, or MIT Kerberos server with a trust established with the corporate kerberos infrastructure, upon presentation of valid credentials. Cloudera Manager generates and distributes these credentials to each of the service roles using an elevated privilege that is securely maintained within its database. Typically the administration privilege will enable the creation and deletion of kerberos principles within a specific organization unit (OU) within the corporate directory (see, Delegating Administration by Using OU Objects). Good practice is to first enable TLS security between the Cloudera Manager and agents in order to ensure the Kerberos keytab files are transported over an encrypted connection.

Impersonalisation

Within a CDP system there are two methods of impersonation that are supported. The first is a simple "doAs" system, where certain services are trusted to assert the identity of connecting users. For example, Oozie, Hue and Hive are *trusted* within the CDP environment to act on behalf of the user who connected to them - this is

known as impersonation and is configured by the `hadoop.proxyuser.*` set of parameters.

This allows, for example, a user to connect to Hue or HiveServer2 (and we trust Hue/HiveServer2 to correctly identify that user) and then for Hue/HiveServer2 to act on that user's behalf.

Encryption

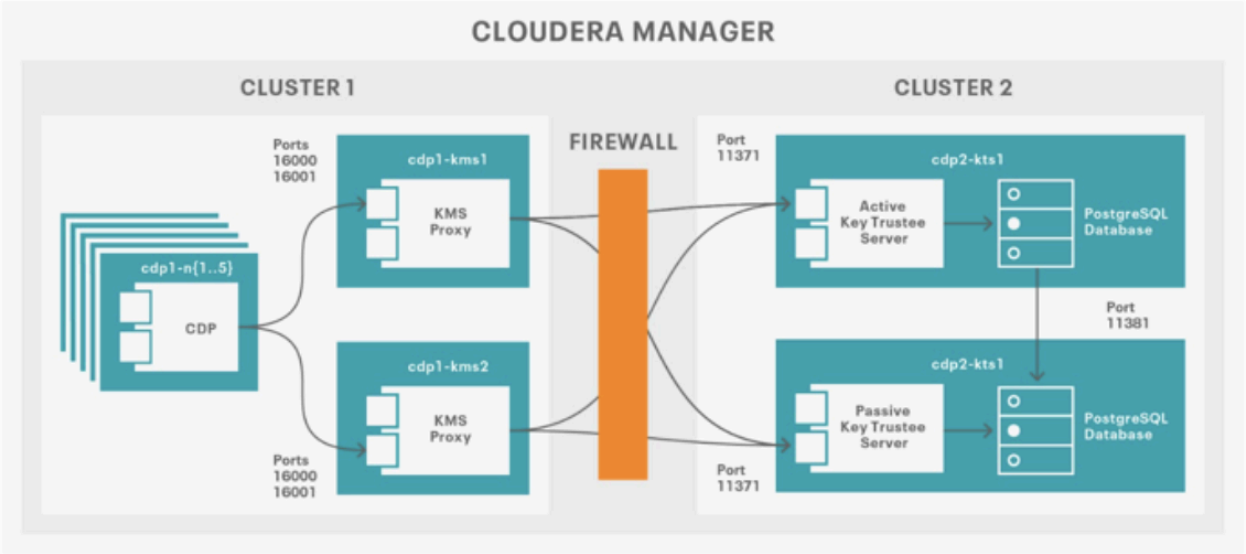
Cloudera Data Platform supports encryption for both Data in Transit and at rest.

Encryption in transit is enabled using TLS security with two modes of deployment: manual or Auto TLS. For manual TLS, customers use their own scripts to generate and deploy their own certificates to the cluster hosts and the locations used are then configured in Cloudera Manager to enable their use by the cluster services.

Typically, security artefacts such as certificates will be stored on the local filesystem /opt/cloudera/security. The cloudera-deploy Ansible automation (see, Cloudera Labs: Cloudera Deploy) uses this method.

Auto TLS enables Cloudera Manager to act as a certificate authority, standalone or delegated by an existing corporate authority. CM can then generate, sign and then deploy the signed certificates around the cluster along with any associated truststores. Once in place, the certificates are then used to encrypt network traffic between the cluster services. There are three primary communication channels, HDFS Transparent Encryption, Data Transfer and Remote Procedure Calls, and communications with the various user Interfaces and APIs.

Encryption at Rest, In order to store sensitive data securely it is vital to ensure that data is encrypted at rest whilst also being available for processing by appropriately privileged users and services that are granted the ability to decrypt. A secure CDP cluster will feature full transparent HDFS Encryption, often with separate encryption zones for the various storage tenants and use cases. We caution against encrypting the entire HDFS, rather those directory hierarchies that require encryption.



As well as HDFS other key local storage locations such as YARN and Impala scratch directories, log files can be similarly encrypted using block encryption. Both HDFS and the local filesystem can be integrated to a secure Key Trustee Service, typically deployed into a separate cluster that assumes responsibility for key management. This ensures separation of duties from the cluster administrators and the security administrators that are responsible for the encryption keys. Furthermore Key Trustee client side encryption provides defence in depth for vulnerabilities such as Apache Log4j2 (see, CVE-2021-44228).

Component Mapping

Below is a high level component mapping for existing functionality and proposed functionality

Data Warehouse Components

Existing Stack	Proposed Stack	Functionality
IBM Integrated Analytics System(IIAS)	Cloudera Data Platform (Hive/Impala on HDFS)	Production Database
Database Archival Solution (DAS)	Cloudera Data Platform (Hive/Impala on Ozone)	Archival Database
Change Data Capture (CDC)	Cloudera Data Platform (Flink/NiFi + Debezium)	Sourcing from Source Systems
DataStage	Cloudera Data Platform (NiFi + Spark)	Data Processing
Information Governance Catalogue (IGC)	Cloudera Data Platform (Apache Atlas)	Data Dictionary and Data Lineage
Information Analyzer (IA)	3rd Party	Data Profiling
QualityStage (QS)	3rd Party	Data Quality
Cognos	3rd Party	Reporting
Tivoli Workload Scheduler (TWS)	Cloudera Data Platform (Oozie/CRON)	Job Scheduling
SCAPM		Hardware Monitoring
TSM & SKLM		Data Backup
RTC		Versioning
SFG		File Sharing
DWH Portal		Front end interface for users

Data Lake Components

Existing Stack	Proposed Stack	Functionality
Object Storage	Cloudera Data Platform (Ozone)	Object Storage
DB2 Warehouse	Cloudera Data Platform (Hive/Impala on HDFS/Ozone)	Relational database to store structured data
Network File System (NFS)	Cloudera Data Platform (Ozone) ??	Used to store data files in text format
Elastic Storage Server	Cloudera Data Platform (Ozone) ??	External storage disk used for storing structured semi-structured and unstructured data.
CP4D Virtualization	3rd Party	Used to virtualize data from multiple databases. CP4D's data virtualization capability will integrate multiple data sources with varying data formats seamlessly and will provide a single interface to access data from multiple sources in real time.
Watson Knowledge Catalog	Cloudera Data Platform (Atlas) + Data Validation	Used for searching, cataloging, data validation and data lineage.
Data Refinery	Cloudera Machine Learning (CDV)	Used to visualize insight of data; help to prepare data for ML model development
Watson Studio	Cloudera Machine Learning	Development Env for DS/ML engineers
Watson Machine Learning	Cloudera Machine Learning	ML Deployment and Maintenance
Object Storage	Cloudera Data Platform (Ozone)	Object Storage
DB2 Warehouse	Cloudera Data Platform (Hive/Impala)	Relational database to store structured data
Network File System (NFS)	Network File System (NFS)	Used to store data files in text format
Elastic Storage Server	Elastic Storage Server	External storage disk used for storing structured semi-structured

		and unstructured data.
CP4D Virtualization	3rd Party	Used to virtualize data from multiple databases. CP4D's data virtualization capability will integrate multiple data sources with varying data formats seamlessly and will provide single interface to access data from multiple sources in real time.
GitLab Open Source	GitLab Open Source	Used to manage code base
Windows Active Directory	Windows Active Directory	Used to authenticate users.
IBM Cloud Pak for Data	Cloudera Data Platform (HDFS/Ozone)	Data Lake Platform
BM Cloud Pak for Integration	Cloudera Data Platform (NiFi/Flink)	Platform to ingest real-time / near real-time data
IBM SPSS	3rd Party	Data Analysis Tool (IBM Statistical Package of Social Sciences)

Cloudera Value Proposition

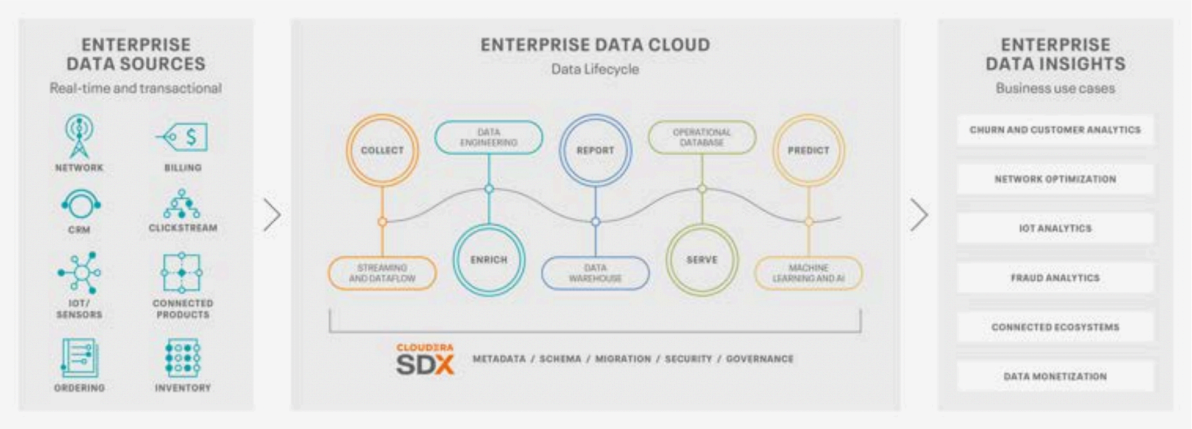
Cloudera Data Platform (CDP) is a Hybrid Data platform built for the enterprise. With CDP businesses manage and secure the end-to-end data lifecycle - collecting, enriching, analyzing, experimenting and predicting with their data - to drive actionable insights and data-driven decision making. The most 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.

An Integrated Data Platform

CDP provides an integrated data platform that creates agility along lines-of-business while facilitating efficiency and security within IT, enabling the entire organization to be more productive. As organizations react quickly to changing business requirements, CDP delivers mission-critical advantages:

- Designed for data engineers, data scientists, BI analysts, developers and enterprise IT
- Simple to use cloud-native service and automatically secure by design
- Best-in-class analytics and integrated data lifecycle
- Self-serve and custom analytics
- Public cloud and on-premises

CDP enables enterprise IT to embrace these seemingly opposing forces because it delivers the capabilities of an enterprise data cloud.



Data Lifecycle Integration: From the 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. With CDP, companies are able to connect disparate workloads to develop critical apps on one data management platform to address current and future workload needs. 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.

Secure and Governed

Cloudera's Shared Data Experience (SDX) ensures consistent data security, governance, and control across the data lifecycle and across all environments while mitigating risk and costs. It provides data access policies 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.

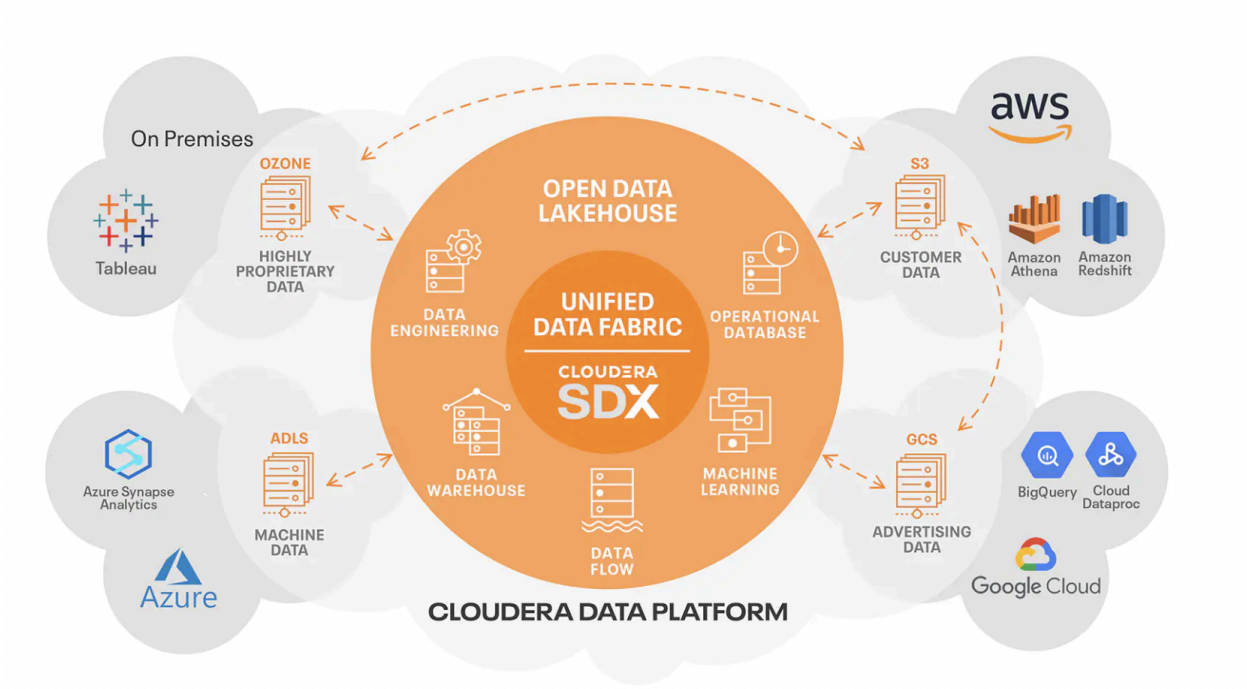
100% Open

Open source software (Cloudera Runtime) is the foundation of CDP. LiceSBId under either Apache or AGPL, this approach accelerates innovation, prevents vendor lock-in and delivers enterprise-grade software. Using open source ensures CDP is interoperable with a broad range of analytic and business intelligence applications. This interoperability helps minimize the expeSBI and effort required to connect customers' IT infrastructures with CDP's data and processing capabilities.

Choice of Deployment

Optimized for hybrid cloud, CDP delivers the same data management and analytic capabilities seamlessly across private and public clouds. Companies can provision a data management and analytic solution in the environment of their choice. CDP separates data management from infrastructure strategy, enabling companies to move data and apps from one environment to another without rewriting applications and retraining personnel. These environments have varying degrees of scalability,

flexibility, and cost-efficiency. CDP ensures that businesses are able to choose the environment that makes the most seSBI for their use cases (data privacy regulations, types of analytics required, etc.), business objectives (speed of deployment, cost efficiency, etc.), and technical considerations (source and location of data, elasticity required, etc.).



CDP Private Cloud

CDP Private Cloud delivers powerful analytic, transactional, and machine learning workloads in a hybrid data platform. With a choice of traditional as well as elastic analytics and scalable object storage, CDP Private Cloud modernizes traditional monolithic cluster deployments in a powerful and efficient platform.

CDP Private Cloud provides the first step for data center customers toward true data and workload mobility, managed from a single pane of glass with consistent data security and governance.

With CDP Private Cloud, organizations benefit from:

- Rapid time to value through simplified provisioning of easy-to-use, self-service analytics in minutes rather than days

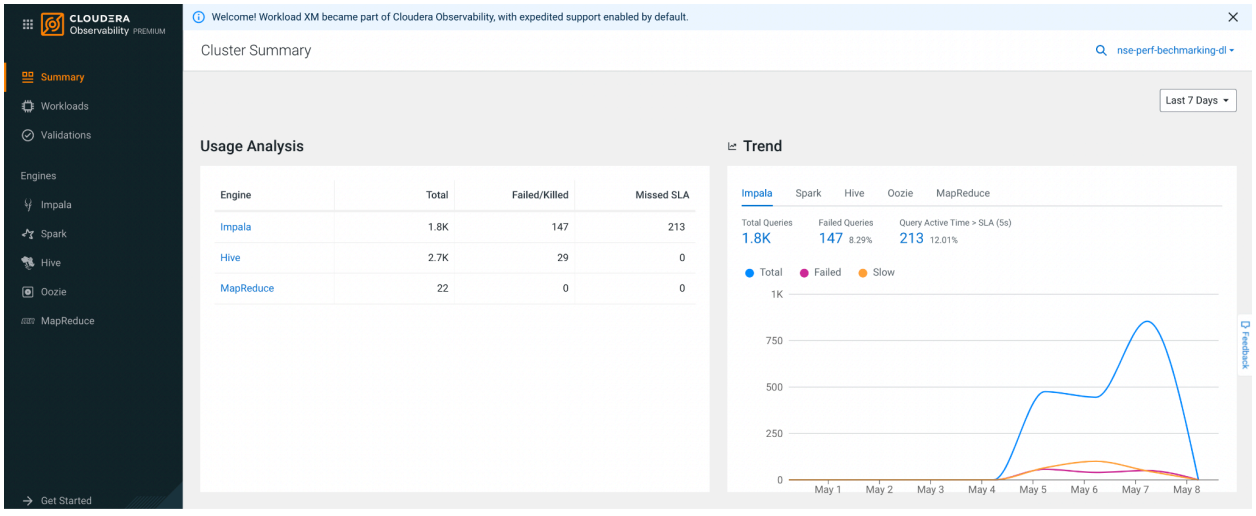
- Improved cost efficiency with optimized resource utilization and the decoupling of compute and storage
- Predictable performance thanks to workload isolation and perfectly managed multi-tenancy

Cloudera Observability

Cloudera Observability is a Cloudera service that helps you interactively understand your environment, data services, workloads, clusters, and resources. Its wide range of metrics and health tests help you identify and troubleshoot existing and potential problems. This service also provides prescriptive guidance and recommendations that help you quickly address those problems and optimize solutions. When a workload completes, diagnostic information about the job or query and the cluster that processed them is collected by Telemetry Publisher, a role in the Cloudera Manager Management Service, and sent to Cloudera Observability.

Cloudera Observability enables you to interactively understand your environment, data services, workloads, clusters, and resources, and optimize your systems through:

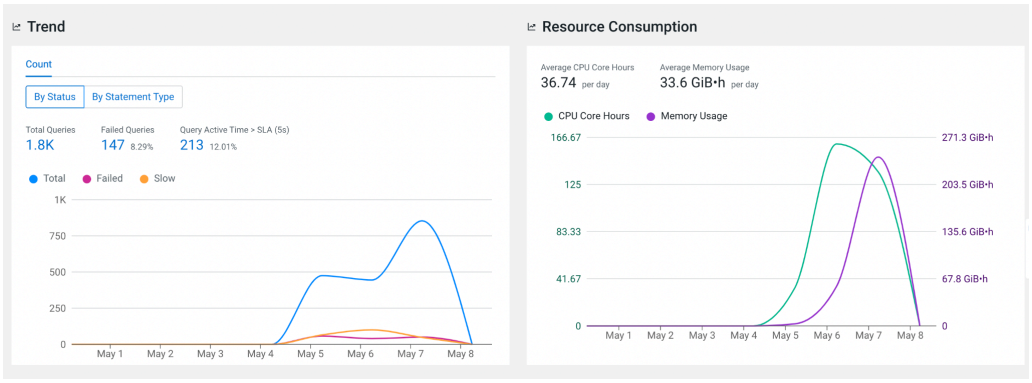
- A wide range of metrics and health tests that help you identify and troubleshoot both existing and potential issues.
- Prescriptive guidance and recommendations that help you quickly address those problems and optimize solutions.
- Performance baselines and historical analysis that help you identify and address performance problems.

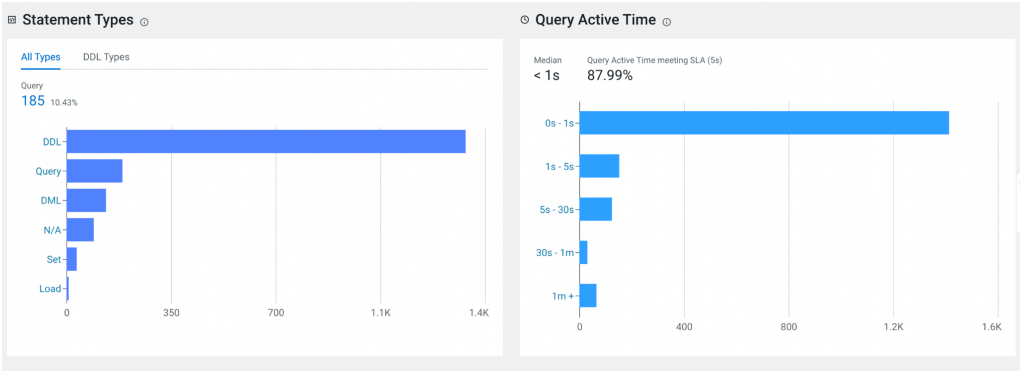


In addition, Cloudera Observability also enables you to:

- Visually display your workload cluster's current and historical costs that help you plan and forecast budgets, future workload environments, and justify current user groups and resources.
- Trigger actions in real-time across jobs and queries that help you take steps to alleviate potential problems.
- Enable the daily delivery of your cluster statistics to your email address that help you to track, compare, and monitor without having to log in to the cluster.
- Break down your workload metrics into more meaningful views for your business requirements that help you analyze specific workload criteria. For example, you can analyze how queries that access a particular database or that use a specific resource pool are performing against your SLAs. Or you can examine how all the queries are performing on your cluster that are sent by a specific user.

Below are some examples of out of the box reporting





Technology Components

Cloudera Data Platform comes provisioned with a range of technologies designed to solve certain business problems. Below is a quick insight into the key technology components with details.

Apache Hadoop HDFS

Hadoop Distributed File System (HDFS) is a Java-based file system for storing large volumes of data. Designed to span large clusters of commodity servers, HDFS provides scalable and reliable data storage.

HDFS forms the data management layer of Apache Hadoop. YARN provides the resource management while HDFS provides the storage.

HDFS is a scalable, fault-tolerant, distributed storage system that works closely with a wide variety of concurrent data access applications. By distributing storage and computation across many servers, the combined storage resource grows linearly with demand.

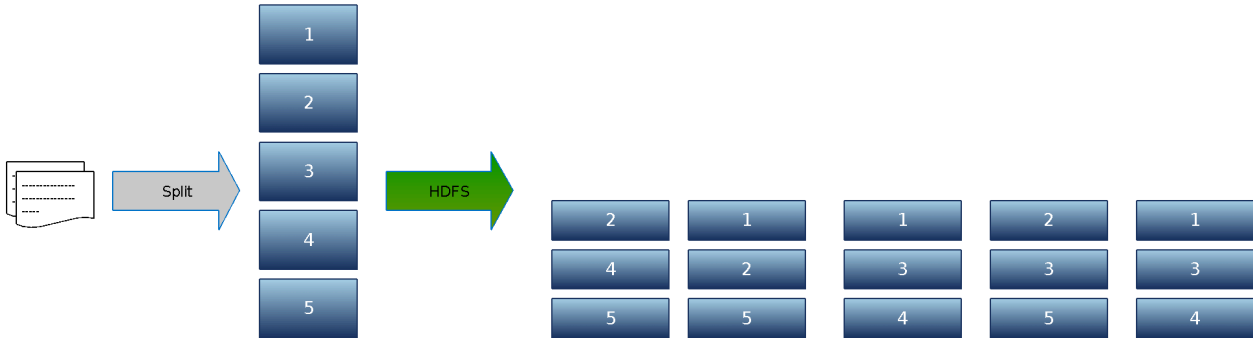
Components of an HDFS cluster

An HDFS cluster contains the following main components: a NameNode and DataNodes.

The NameNode manages the cluster metadata that includes file and directory structures, permissions, modifications, and disk space quotas. The file content is split into multiple data blocks, with each block replicated at multiple DataNodes.

The NameNode actively monitors the number of replicas of a block. In addition, the NameNode maintains the namespace tree and the mapping of blocks to DataNodes, holding the entire namespace image in RAM.

High-Availability (HA) clusters contain a standby for the active NameNode to avoid a single point of failure. These clusters use JournalNodes to synchronize the active and standby NameNodes.



Hadoop slits any files into smaller blocks of usually 128MB each and these blocks are stored across the cluster nodes in a distributed fashion along with replication of individual blocks by default 3 times. No single node of the cluster will have 2 copies of the same block. When rack detection is enabled HDFS intelligently understands when the Nodes are available outside the current rack and moves data in a way that a copy is stored outside the current rack as well for rack failure scenarios.

Hadoop by default is built to sustain double disk failures, even if two nodes fail at the same time, the data loss does not occur and the system continues to serve the end user requests

Benefits of HDFS

HDFS provides the following benefits as a result of which data is stored efficiently and is highly available in the cluster:

1. **Rack awareness:** A node's physical location is considered when allocating storage and scheduling tasks.
2. **Minimal data motion:** Hadoop moves compute processes to the data on HDFS. Processing tasks can occur on the physical node where the data resides. This significantly reduces network I/O and provides very high aggregate bandwidth.
3. **Utilities:** Dynamically diagnose the health of the file system and rebalance the data on different nodes.
4. **Version rollback:** Allows operators to perform a rollback to the previous version of HDFS after an upgrade, in case of human or systematic errors.
5. **Standby NameNode:** Provides redundancy and supports high availability (HA).
6. **Operability:** HDFS requires minimal operator intervention, allowing a single operator to maintain a cluster of thousands of nodes

Apache Ozone

Apache Ozone is a scalable, redundant, and distributed object store optimized for big data workloads. Apart from scaling to billions of objects of varying sizes, applications that use frameworks like Apache Spark, Apache YARN and Apache Hive work natively on Ozone object store without any modifications. HDFS works best when most of the files are large but HDFS suffers from small file limitations. Ozone is a distributed key-value object store that can manage both small and large files alike. Ozone natively supports the S3 API and provides a Hadoop-compatible file system interface. Ozone object store is available in a CDP Private Cloud Base deployment.

Ozone storage elements

Ozone consists of the following important storage elements:

- **Volumes** Volumes are similar to accounts. Volumes can be created or deleted only by administrators. An administrator creates a volume for an organization or a team.
- **Buckets** A volume can contain zero or more buckets. Ozone buckets are similar to Amazon S3 buckets. Depending on their requirements, regular users can create buckets in volumes.
- **Keys** Each key is part of a bucket. Keys are unique within a given bucket and are similar to S3 objects. Ozone stores data as keys inside buckets.

When a client writes a key, Ozone stores the associated data on DataNodes in chunks called blocks. Therefore, each key is associated with one or more blocks. Within a DataNode, multiple unrelated blocks can reside in a storage container.

Key features of Ozone Key features of Ozone are:

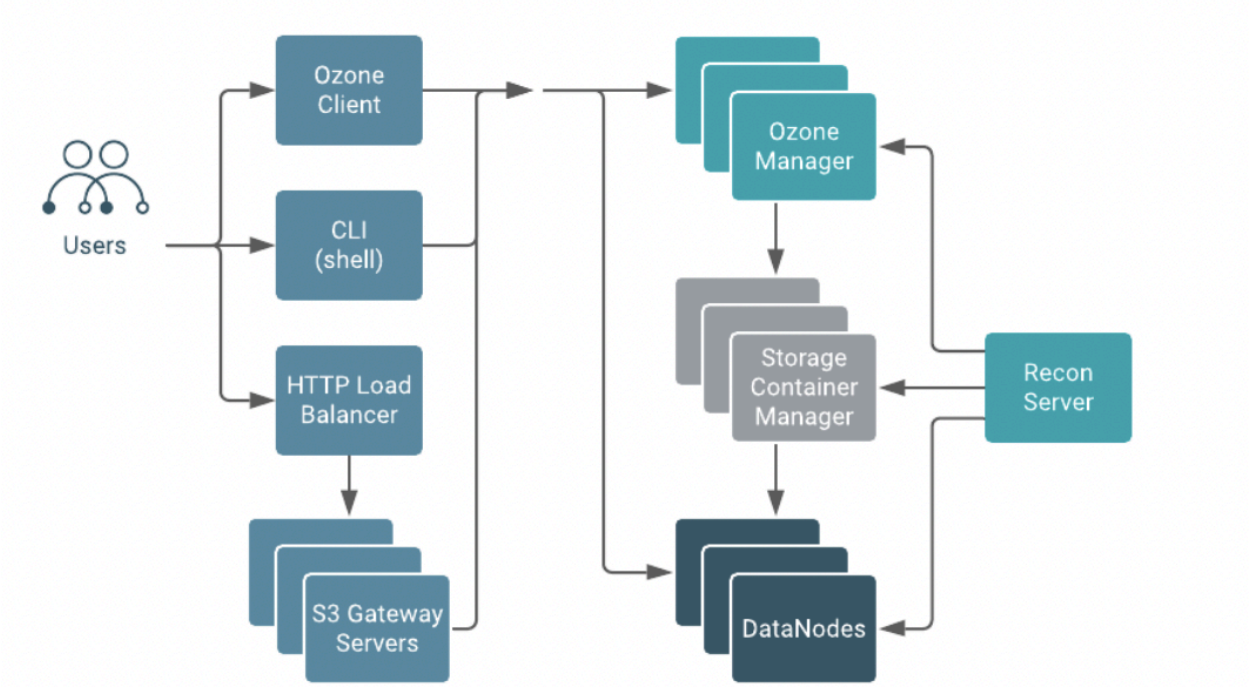
- **Consistency** Strong consistency simplifies application design. Ozone object store is designed to provide strict serializability.
- **Architectural simplicity** A simple architecture is easy to use and easy to debug when things go wrong. The architecture of Ozone object store is simple and at the same time scalable. It is designed to store over 100 billion objects in a single cluster.
- **Layered architecture** The layered file system of Ozone object store helps to achieve the scale required for the modern storage systems . It separates the namespace management from the block and node management layer, which allows users to independently scale on both axes.

- **Easy recovery** A key strength of HDFS is that it can effectively recover from catastrophic events like cluster-wide power loss without losing data and without expensive recovery steps. Rack and node losses are relatively minor events. Ozone object store is similarly robust in the face of failures.
- **Open source in Apache** The Apache Open Source community is critical to the success of Ozone object store. All Ozone design and development are being done in the Apache Hadoop community.
- **Interoperability with Hadoop ecosystem** Ozone object store is usable by the existing Apache Hadoop ecosystem and related applications like Apache Hive, Apache Spark and traditional MapReduce jobs

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:



Ozone Manager

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. OM uses Apache Ratis (Raft protocol) to replicate Ozone manager state. While RocksDB (embedded storage engine) persists the metadata or keyspace, the flush of the Ratis transaction to the local disk ensures data durability in Ozone. Therefore, Cloudera recommends a low-latency local storage device like NVMe SSD on each OM node for maximum throughput. A typical Ozone deployment has three OM nodes for redundancy.

DataNodes

Ozone DataNodes, not the same as HDFS DataNodes, store the actual user data in the Ozone cluster. DataNodes store the blocks of data that clients write. A collection of these blocks is a storage container. The client streams data in the form of fixed-size chunk files (4MB) for a block. The chunk files constitute what gets actually written to the disk. The DataNode sends heartbeats to SCM at fixed time intervals. With every heartbeat, the DataNode sends container reports. Every storage container in the DataNode has its own RocksDB instance which stores the metadata for the blocks and

individual chunk files. Every DataNode can be a part of one or more active pipelines. A pipeline of DataNodes is actually a Ratis quorum with an elected leader and follower DataNodes accepting writes. Reads from the client go directly to the DataNode and not over Ratis.

Cloudera recommends an SSD on a DataNode to persist the Ratis logs for the active pipelines and significantly boosts the write throughput.

Storage Container Manager

The Storage Container Manager (SCM) is a master service in Ozone.

A storage container is the replication unit in Ozone. Unlike HDFS, which manages block level replication, Ozone manages the replication of a collection of storage blocks called storage containers. The default size of a container is 5 GB. SCM manages DataNode pipelines and placement of containers on the pipelines. A pipeline is a collection of DataNodes based on the replication factor. For example, given the default replication factor of three, each pipeline contains three DataNodes. SCM is responsible for creating and managing active write pipelines of DataNodes on which the block allocation happens.

The client directly writes blocks to open containers on the DataNode, and the SCM is not directly on the data path. A container is immutable after it is closed. SCM uses RocksDB to persist the pipeline metadata and the container metadata. The size of this metadata is much smaller compared to the keyspace managed by OM.

SCM is a highly available component which makes use of Apache Ratis. Cloudera recommends an SSD on SCM nodes for the Ratis write-ahead log and the RocksDB.

A typical Ozone deployment has three SCM nodes for redundancy. SCM service instances can be collocated with OM instances; therefore, you can use the same master nodes for both SCM and OM.

Recon Server

Recon is a centralized monitoring and management service within an Ozone cluster that provides information about the metadata maintained by different Ozone components such as OM and SCM.

Recon takes a snapshot of the OM and SCM metadata while also receiving heartbeats from the Ozone DataNode. Recon asynchronously builds an offline copy of the full state of the cluster in an incremental manner depending on how busy the cluster is. Recon usually trails the OM by a few transactions in terms of updating its snapshot

of the OM metadata. Cloudera recommends using an SSD for maintaining the snapshot information because an SSD would help Recon in staying updated with the OM.

Ozone File System Interfaces

Ozone is a multi-protocol storage system with support for the following interfaces:

- **ofs:** Hadoop-compatible file system allows any application that expects an HDFS like interface to work against Ozone with no changes. Frameworks like Apache Spark, YARN, and Hive work against Ozone without the need for any change.
- **s3:** Amazon's Simple Storage Service (S3) protocol. You can use S3 clients and S3 SDK-based applications without any modifications to Ozone.
- **o3fs:** A bucket-rooted Hadoop Compatible file system interface.
- **o3:** An object store interface that can be used from the Ozone shell.

S3 Gateway

S3 gateway is a stateless component that provides REST access to Ozone over HTTP and supports the AWS-compatible s3 API. S3 gateway supports multipart uploads and encryption zones.

In addition, S3 gateway transforms the s3 API calls over HTTP to rpc calls to other Ozone components. To scale your S3 access, Cloudera recommends deploying multiple gateways behind a load balancer like haproxy that support Direct Server Return (DSR) so that the load balancer is not on the data path.

Ozone security architecture

Apache Ozone is a scalable, distributed, and high performance object store optimized for big data workloads and can handle billions of objects of varying sizes. Applications that use frameworks like Apache Spark, Apache YARN and Apache Hive work natively on Ozone without any modifications. Therefore, it is essential to have robust security mechanisms to secure your cluster and to protect your data in Ozone.

There are various authentication and authorization mechanisms available in Apache Ozone.

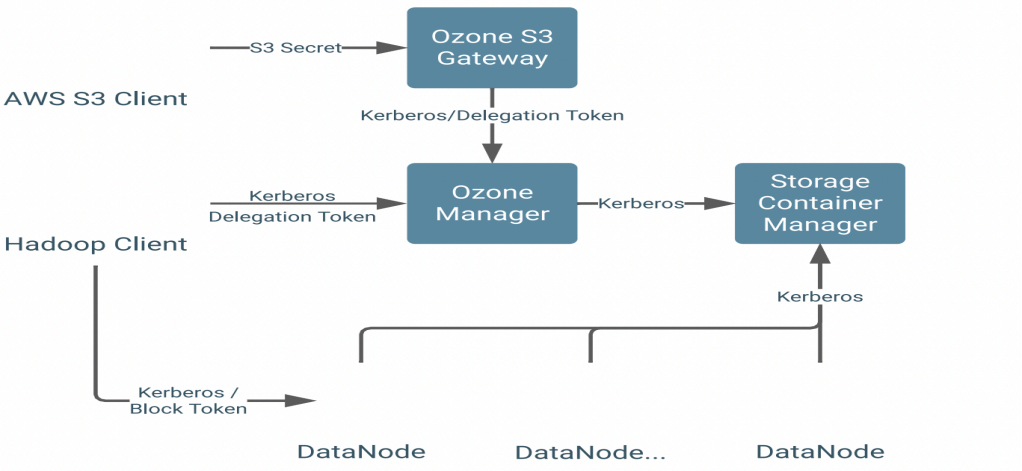
Authentication in Ozone

Authentication is the process of recognizing a user's identity for Ozone components. Apache Ozone supports strong authentication using Kerberos and security tokens.

Kerberos-based authentication Ozone depends on Kerberos to make the clusters secure. To enable security in an Ozone cluster, you must set the following parameters:

Property	Value
ozone.security.enabled	true
hadoop.security.authentication	kerberos

The following diagram illustrates how service components, such as Ozone Manager (OM), Storage Container Manager (SCM), DataNodes, and Ozone Clients are authenticated with each other through Kerberos:



Authorization in Ozone

Authorization is the process of specifying access rights to Ozone resources. Once a user is authenticated, authorization enables you to specify what the user can do within an Ozone cluster. For example, you can allow users to read volumes, buckets, and keys while restricting them from creating volumes.

Ozone supports authorization through the Apache Ranger plugin or through native Access Control Lists (ACLs).

Using Ranger for authorization

Apache Ranger provides a centralized security framework to manage access control through an user interface that ensures consistent policy administration across

Cloudera Data Platform (CDP) components. For Ozone to work with Ranger, you can use Cloudera Manager and enable the Ranger service instance with which you want to use Ozone. If Ranger authorization is enabled, the native ACLs are ignored and ACLs specified through Ranger are considered.

Ozone containers

Containers are the fundamental replication unit of Ozone and are managed by the Storage Container Manager (SCM) service. Containers are big binary units (5 Gb by default) that can contain multiple blocks.

Blocks are local information that are not managed by SCM. Therefore, even if billions of small files are created in the system (which means billions of blocks are created), only the status of the containers will be reported by the DataNodes and the containers will be replicated.

Whenever Ozone Manager requests a new block allocation from the SCM, SCM identifies the suitable container and generates a block id which contains a ContainerId and a LocalId. The client connects to the DataNode which stores the container, and the DataNode manages the separated block based on the LocalId.

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

How Ozone manages delete operations

Ozone is a consistent object store and hence delete operations in Ozone are synchronous. When the delete request is complete, Ozone Manager deletes the key from the active namespace and marks the file for garbage collection.

Ozone also follows the principle of asynchronous deletion. Garbage collection and the mechanism to recover a storage space is managed asynchronously on most file systems to ensure that the delete operations do not compete with the read and write operations.

The files marked as deleted in Ozone Manager are aggregated by containers and the request is sent to SCM to delete the blocks. SCM then forwards the requests to the DataNode to recover the actual space from the disk. A block deletion happens only over closed containers. For example, if you delete objects in the namespace, the deletion reflects only for the corresponding blocks in the closed containers. For blocks present in containers with state other than CLOSED, the deletion happens only after the container reaches a CLOSED state.

Apache Impala

The Apache Impala provides high-performance, low-latency SQL queries on data stored in popular Apache Hadoop file formats. The Impala solution is composed of the following components. Impala is a MPP (Massive Parallel Processing) SQL query engine for processing huge volumes of data that is stored in a Hadoop cluster.

It is an open source software which is written in C++ and Java. It provides high performance and low latency compared to other SQL engines for Hadoop. In other words, Impala is the highest performing SQL engine (giving RDBMS-like experience) which provides the fastest way to access data that is stored in a Hadoop Distributed File System.

Impala

The Impala service coordinates and executes queries received from clients. Queries are distributed among Impala nodes, and these nodes then act as workers, executing parallel query fragments.

Hive Metastore Stores

Information about the data available to Impala. For example, the metastore lets Impala know what databases are available and what the structure of those databases is. As you create, drop, and alter schema objects, load data into tables, and so on through Impala SQL statements, the relevant metadata changes are automatically broadcast to all Impala nodes by the dedicated catalog service.

Clients

Entities including Hue, ODBC clients, JDBC clients, Business Intelligence applications, and the Impala Shell can all interact with Impala. These interfaces are typically used to issue queries or complete administrative tasks such as connecting to Impala.

Storage for data to be queried

Queries executed using Impala are handled as follows:

- User applications send SQL queries to Impala through ODBC or JDBC, which provide standardized querying interfaces. The user application may connect to any impalad in the cluster. This impalad becomes the coordinator for the query.
- Impala parsers the query and analyzes it to determine what tasks need to be performed by impalad instances across the cluster. Execution is planned for optimal efficiency.
- Storage services are accessed by local impalad instances to provide data.
- Each impalad returns data to the coordinating impalad, which sends these results to the client.

Components of Impala

The Impala service is a distributed, massively parallel processing (MPP) database engine. It consists of different daemon processes that run on specific hosts within your Hadoop cluster.

Impala service consists of the following categories of processes, referred to as roles.

Impala Daemon

The core Impala component is the Impala daemon, physically represented by the impalad process. A few of the key functions that an Impala daemon performs are:

- Reads and writes to data files.
- Accepts queries transmitted from the impala-shell command, Hue, JDBC, or ODBC.
- Parallelizes the queries and distributes work across the cluster.
- Transmits intermediate query results back to the central coordinator.

Impala daemons can be deployed in one of the following ways:

- HDFS and Impala are co-located, and each Impala daemon runs on the same host as a DataNode.
- Impala is deployed separately in a compute cluster and reads remotely from HDFS, S3, ADLS, etc.

The Impala daemons are in constant communication with StateStore, to confirm which daemons are healthy and can accept new work.

They also receive broadcast messages from the Catalog Server daemon whenever an Impala daemon in the cluster creates, alters, or drops any type of object, or when an **ISBIRT** or **LOAD DATA** statement is processed through Impala. This background communication minimizes the need for **REFRESH** or **INVALIDATE METADATA** statements that were needed to coordinate metadata across Impala daemons.

You can control which hosts act as query coordinators and which act as query executors, to improve scalability for highly concurrent workloads on large clusters.

Impala StateStore

The Impala StateStore checks on the health of all Impala daemons in a cluster, and continuously relays its findings to each of those daemons. It is physically represented by a daemon process named statestored. You only need such a process on one host in a cluster. If an Impala daemon goes offline due to hardware failure, network error, software issue, or other reason, the StateStore informs all the other Impala daemons so that future queries can avoid making requests to the unreachable Impala daemon.

Because the StateStore's purpose is to help when things go wrong and to broadcast metadata to coordinators, it is not always critical to the normal operation of an Impala cluster. If the StateStore is not running or becomes unreachable, the Impala daemons continue running and distributing work among themselves as usual when working with the data known to Impala. The cluster just becomes less robust if other Impala daemons fail, and metadata becomes less consistent as it changes while the StateStore is offline. When the StateStore comes back online, it re-establishes communication with the Impala daemons and resumes its monitoring and broadcasting functions.

If you issue a DDL statement while the StateStore is down, the queries that access the new object the DDL created will fail.

Impala Catalog Server

The Catalog Server relays the metadata changes from Impala SQL statements to all the Impala daemons in a cluster. It is physically represented by a daemon process named **catalogd**. You only need such a process on one host in a cluster. Because the requests are passed through the StateStore daemon, it makes seSBI to run the statestored and **catalogd** services on the same host.

The catalog service avoids the need to issue **REFRESH** and **INVALIDATE METADATA** statements when the metadata changes are performed by statements issued through Impala. When you create a table, load data, and so on through Hive, you do need to issue **REFRESH** or **INVALIDATE METADATA** on an Impala node before executing a query there.

Apache Hive

Hive is a data warehouse system for summarizing, querying, and analyzing huge, disparate data sets. Cloudera Runtime (CR) services include Hive on Tez and Hive Metastore. Hive on Tez is based on Apache Hive 3.x, a SQL-based data warehouse system. The enhancements in Hive 3.x over previous versions can improve SQL query performance, security, and auditing capabilities. The Hive metastore (HMS) is a separate service, not part of Hive, not even necessary on the same cluster. HMS stores the metadata on the backend for Hive, Impala, Spark, and other components.

Hive on Tez

The Cloudera Data Platform (CDP) service provides an Apache Hive SQL database that Apache Tez executes.

The Hive on Tez service provides a SQL-based data warehouse system based on Apache Hive 3.x. The enhancements in Hive 3.x over previous versions can improve SQL query performance, security, and auditing capabilities. The Hive metastore (HMS) is a

separate service, not part of Hive, not even necessarily on the same cluster. HMS stores the metadata on the backend for Hive, Impala, Spark, and other components.

Apache Tez is the Hive execution engine for the Hive on Tez service, which includes HiveServer (HS2) in Cloudera Manager. MapReduce is not supported. In a Cloudera cluster, if a legacy script or application specifies MapReduce for execution, an exception occurs. Most user-defined functions (UDFs) require no change to run on Tez instead of MapReduce.

With expressions of directed acyclic graphs (DAGs) and data transfer primitives, execution of Hive queries on Tez instead of MapReduce improves query performance. In the Cloudera Data Platform (CDP), Tez is usually used only by Hive, and launches and manages Tez AM automatically when Hive on Tez starts. SQL queries you submit to Hive are executed as follows:

- Hive compiles the query.
- Tez executes the query.
- Resources are allocated for applications across the cluster.
- Hive updates the data in the data source and returns query results.

ACID transaction processing

Hive 3 tables are ACID (Atomicity, Consistency, Isolation, and Durability)-compliant. Hive 3 write and read operations improve the performance of transactional tables. Atomic operations include simple writes and iSBIrts, writes to multiple partitions, and multiple iSBIrts in a single SELECT statement. A read operation is not affected by changes that occur during the operation. You can iSBIrt or delete data, and it remains consistent throughout software and hardware crashes. Creation and maintenance of Hive tables is simplified because there is no longer any need to bucket tables.

Materialized views

Because multiple queries frequently need the same intermediate roll up or joined table, you can avoid costly, repetitious query portion sharing, by precomputing and caching intermediate tables into views.

Query results cache

Hive filters and caches similar or identical queries. Hive does not recompute the data that has not changed. Caching repetitive queries can reduce the load substantially when hundreds or thousands of users of BI tools and web services query Hive.

Scheduled Queries

Using SQL statements, you can schedule Hive queries to run on a recurring basis, monitor query progress, temporarily ignore a query schedule, and limit the number running in parallel. You can use scheduled queries to start compaction and periodically rebuild materialized views, for example.

Security improvements

Apache Ranger secures Hive data by default. To meet demands for concurrency improvements, ACID support, render security, and other features, Hive tightly controls the location of the warehouse on a file system, or object store, and memory resources.

With Apache Ranger and Apache Hive ACID support, your organization will be ready to support and implement GDPR (General Data Protection Regulation).

Apache Spark3

Apache Spark is a distributed, in-memory data processing engine designed for large-scale data processing and analytics.

Apache Spark is a general framework for distributed computing that offers high performance for both batch and interactive processing. It exposes APIs for Java, Python, and Scala and consists of Spark core and several related projects.

You can run Spark applications locally or distributed across a cluster, either by using an interactive shell or by submitting an application. Running Spark applications interactively is commonly performed during the data-exploration phase and for ad hoc analysis.

To run applications distributed across a cluster, Spark requires a cluster manager. Cloudera Data Platform (CDP) supports only the YARN cluster manager. When run on YARN, Spark application processes are managed by the YARN ResourceManager and NodeManager roles

CDP supports Apache Spark, Apache Livy for local and remote access to Spark through the Livy REST API, and Apache Zeppelin for browser-based notebook access to Spark.

CDP supports both Spark2 and Spark3 on the same cluster, they both can co-exist and run in parallel without affecting each other.

Spark3 has considerable feature upgrades and is inherently faster than spark2. Some of the important feature upgrades of Spark3 are as below.

- major advances in Python and SQL capabilities and a focus on ease of use for both exploration and production
- Introduction of Adaptive Query Execution (AQE) framework for performance boost to Spark SQL
- Dynamic partition pruning for DW queries and broadcast joins
- Improvements in Python Pandas API's
- Better Python error handling, simplifying PySpark exceptions

The new **Adaptive Query Execution (AQE)** framework improves performance and simplifies tuning by generating a better execution plan at runtime, even if the initial plan is suboptimal due to absent/inaccurate data statistics and misestimated costs. Because of the storage and compute separation in Spark, data arrival can be unpredictable. For all these reasons, runtime adaptivity becomes more critical for Spark than for traditional systems. This release introduces three major adaptive optimizations:

- **Dynamically coalescing shuffle partitions** simplifies or even avoids tuning the number of shuffle partitions. Users can set a relatively large number of shuffle partitions at the beginning, and AQE can then combine adjacent small partitions into larger ones at runtime.
- **Dynamically switching join strategies** partially avoids executing suboptimal plans due to missing statistics and/or size misestimation. This adaptive optimization can automatically convert sort-merge join to broadcast-hash join at runtime, further simplifying tuning and improving performance.
- **Dynamically optimizing skew joins** is another critical performance enhancement, since skew joins can lead to an extreme imbalance of work and severely downgrade performance. After AQE detects any skew from the shuffle file statistics, it can split the skew partitions into smaller ones and join them with the corresponding partitions from the other side. This optimization can parallelize skew processing and achieve better overall performance.
-

https://blog.cloudera.com/how-does-apache-spark-3-0-increase-the-performance-of-your-sql-workloads/

Dynamic Partition Pruning is applied when the optimizer is unable to identify at compile time the partitions it can skip. This is not uncommon in star schemas, which consist of one or multiple fact tables referencing any number of dimension tables. In such join operations, we can prune the partitions the join reads from a fact table, by identifying those partitions that result from filtering the dimension tables. In a TPC-DS benchmark, 60 out of 102 queries show a significant speedup between 2x and 18x.

ANSI SQL compliance is critical for workload migration from other SQL engines to Spark SQL. To improve compliance, this release switches to Proleptic Gregorian calendar and also enables users to forbid using the reserved keywords of ANSI SQL as identifiers. Additionally, we've introduced runtime overflow checking in numeric operations and compile-time type enforcement when iSBIrting data into a table with a predefined schema. These new validations improve data quality.

Join hints: While we continue to improve the compiler, there's no guarantee that the compiler can always make the optimal decision in every situation — join algorithm selection is based on statistics and heuristics. When the compiler is unable to make the best choice, users can use join hints to influence the optimizer to choose a better plan. This release extends the existing join hints by adding new hints: SHUFFLE_MERGE, SHUFFLE_HASH and SHUFFLE_REPLICATE_NL.

Apache Iceberg

Apache Iceberg is a cloud-native, high-performance open table format for organizing petabyte-scale analytic datasets on a file system or object store. Combined with Cloudera Data Platform (CDP), users can build an open data lakehouse architecture for multi-function analytics and to deploy large scale end-to-end pipelines.

Open data lakehouse on CDP simplifies advanced analytics on all data with a unified platform for structured and unstructured data and integrated data services to enable any analytics use case from ML, BI to stream analytics and real-time analytics. Apache Iceberg is the secret sauce of the open lakehouse.

Apache Iceberg is a table format for huge analytics datasets on storage systems such as HDFS. You can efficiently query large Iceberg tables on object stores. Iceberg supports concurrent reads and writes on all storage media.

You create Iceberg tables and run queries from Hive or Impala in CDP. The Hive metastore stores Iceberg metadata, including the location of the table.

Hive metastore plays a lightweight role in the Catalog operations. Iceberg relieves Hive metastore (HMS) pressure by storing partition information in metadata files on the file system/object store instead of within the HMS. This architecture supports rapid scaling without performance hits.

By default, Hive and Impala use the Iceberg HiveCatalog. Cloudera recommends the default HiveCatalog to create an Iceberg table.

You can use Iceberg when a single table contains tens of petabytes of data, and you can read these tables without compromising performance. From Apache Hive and Apache Impala, you can query Iceberg tables. The following features are included:

- Listing table snapshot and history
- Expiring snapshots from Hive and Impala
- Migrating external tables to iceberg in Hive
- Iceberg table rollback from Hive
- Creating an Iceberg table from Hive with a metadata location
- Expiring and removing old snapshots
- Performance and scalability enhancements

Apache Iceberg integrates Apache Ranger for security. You can use Ranger integration with Hive and Impala to apply fine-grained access control to sensitive data in Iceberg tables. Iceberg is also integrated with Data Visualization for creating dashboards and other graphics of your Iceberg data.

Iceberg comes with a host of features, below are some of key features

Partition Evolution - Evolving a partition means changing it without rewriting data files. To evolve an Iceberg partition from Hive or Impala, you learn to use ALTER to change identity partitions. By setting a partition spec for an identity transformation partition, you alter the table.

Rollback feature - In the event of a problem with your table, you can reset a table to a good state as long as the snapshot of the good table is available. From Hive, roll back the table data to the state at an older table snapshot, or to a timestamp. From Impala, this feature is not supported.

Time Travel - you can run point in time queries for auditing and regulatory workflows on Iceberg tables. Time travel queries can be time-based or based on a snapshot ID.

Cloudera Search

Cloudera Search is Apache Solr fully integrated in the Cloudera platform, taking advantage of the flexible, scalable, and robust storage system and data processing

frameworks included in Cloudera Data Platform (CDP). This eliminates the need to move large data sets across infrastructures to perform business tasks. It further enables a streamlined data pipeline, where search and text matching is part of a larger workflow.

Cloudera Search provides easy, natural language access to data stored in or ingested into Hadoop, HBase, or cloud storage. End users and other web services can use full-text queries and faceted drill-down to explore text, semi-structured, and structured data as well as quickly filter and aggregate it to gain business insight without requiring SQL or programming skills.

Using Cloudera Search with the CDP infrastructure provides:

- Simplified infrastructure
- Better production visibility and control
- Quicker insights across various data types
- Quicker problem resolution
- Simplified interaction and platform access for more users and use cases beyond SQL
- Scalability, flexibility, and reliability of search services on the same platform used to run other types of workloads on the same data
- A unified security model across all processes with access to your data
- Flexibility and scale in ingest and pre-processing options

The following table describes a few main Cloudera Search features.

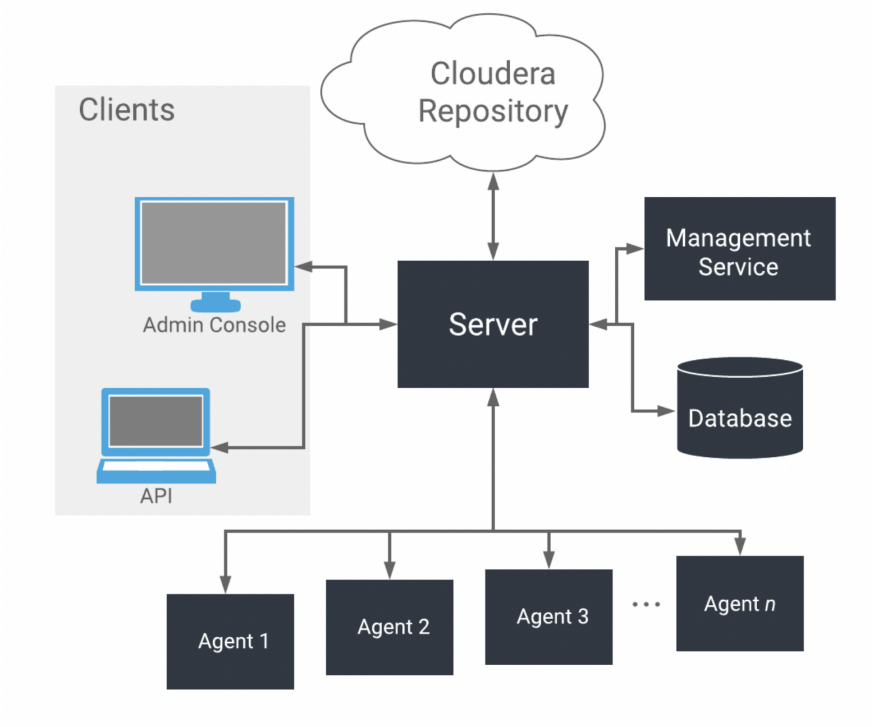
Index storage in HDFS	Cloudera Search is integrated with HDFS for robust, scalable, and self-healing index storage. Indexes created by Solr/Lucene are directly written in HDFS with the data, instead of to local disk, thereby providing fault tolerance and redundancy. Cloudera Search is optimized for fast read and write of indexes in HDFS while indexes are served and queried through standard Solr mechanisms. Because data and indexes are co-located, data processing does not require transport or separately managed storage.
Easy interaction and data exploration through Hue	A Cloudera Search GUI is provided as a Hue plug-in, enabling users to interactively query data, view result files, and do faceted exploration. Hue can also schedule standing

	queries and explore index files. This GUI uses the Cloudera Search API, which is based on the standard Solr API. The drag-and-drop dashboard interface makes it easy for anyone to create a Search dashboard.
Simplified data processing for Search workloads	Cloudera Search can use Apache Tika for parsing and preparation of many of the standard file formats for indexing. Additionally, Cloudera Search supports Avro, Hadoop Sequence, and Snappy file format mappings, as well as Log file formats, JSON, XML, and HTML. Cloudera Search also provides Morphlines, an easy-to-use, pre-built library of common data preprocessing functions. Morphlines simplify data preparation for indexing over a variety of file formats. Users can easily implement Morphlines for Kafka, and HBase, or re-use the same Morphlines for other applications, such as MapReduce or Spark jobs.

Cloudera Manager

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. A single instance of Cloudera Manager can manage multiple clusters, including older versions of Cloudera Runtime and CDH.

As depicted below, the heart of Cloudera Manager is the Cloudera Manager Server. The Server hosts the Cloudera Manager Admin Console, the Cloudera Manager API, and the application logic, and is responsible for installing software, configuring, starting, and stopping services, and managing the cluster on which the services run.



The Cloudera Manager Server works with several other components:

- **Agent** - installed on every host. The agent is responsible for starting and stopping processes, unpacking configurations, triggering installations, and monitoring the host.
- **Management Service** - a service consisting of a set of roles that perform various monitoring, alerting, and reporting functions.
- **Database** - stores configuration and monitoring information. Typically, multiple logical databases run across one or more database servers. For example, the Cloudera Manager Server and the monitoring roles use different logical databases.
- **Cloudera Repository** - repository of software for distribution by Cloudera Manager.
- **Clients** - are the interfaces for interacting with the server:
    - **Cloudera Manager Admin Console** - Web-based user interface that administrators use to manage clusters and Cloudera Manager.
    - **Cloudera Manager API** - API developers use to create custom Cloudera Manager applications.

Heart Beating

Heartbeats are a primary communication mechanism in Cloudera Manager. By default Agents send heartbeats every 15 seconds to the Cloudera Manager Server. However, to reduce user latency the frequency is increased when state is changing.

During the heartbeat exchange, the Agent notifies the Cloudera Manager Server of its activities. In turn the Cloudera Manager Server responds with the actions the Agent should be performing. Both the Agent and the Cloudera Manager Server end up doing some reconciliation. For example, if you start a service, the Agent attempts to start the relevant processes; if a process fails to start, the Cloudera Manager Server marks the start command as having failed.

Cloudera Stream Processing

Cloudera Stream Processing (CSP) provides advanced messaging, stream processing and analytics capabilities powered by Apache Kafka as the core stream processing engine. It also provides stream management capabilities. Two streams management services have been added for Kafka around monitoring and replication. Streams Messaging Manager (SMM) provides a single monitoring dashboard for a Kafka cluster. Streams Replication Manager (SRM) provides enterprises with the ability to implement cross-cluster Kafka topic replication.

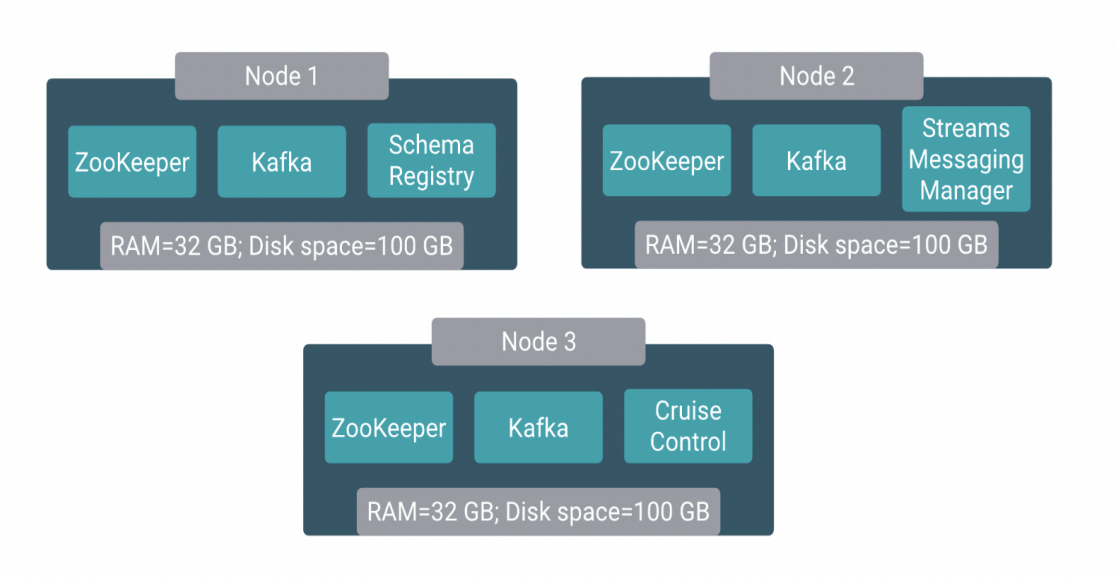
Cloudera's streaming components empower enterprises to handle some of the most complex and sophisticated streaming use cases. You can evaluate the streaming components in CDP Private Cloud Base for up to 60 days. This document walks you through the process of installing the trial software and creating a streams cluster for your proof-of-concept (POC) or sandbox environment. After you evaluate CDP Private Cloud Base, you can contact Cloudera to request a liceSBI for your production environment.

The CDP Private Cloud Base streaming components provide advanced messaging, real-time processing, and analytics on real-time streaming data. The components are:

- Cruise Control
- Apache Kafka
- Schema Registry
- Streams Messaging Manager (SMM)

- Streams Replication Manager

The following diagram shows the layout of the streaming components across a three-node cluster for reference:



Apache Kafka

Apache Kafka is a high performance, highly available, and redundant streaming message platform.Kafka functions much like a publish/subscribe messaging system, but with better throughput, built-in partitioning, replication, and fault tolerance. Kafka is a good solution for large scale message processing applications. It is often used in tandem with Apache Hadoop, and Spark Streaming

You might think of a log as a time-sorted file or data table. Newer entries are appended to the log over time, from left to right. The log entry number is a convenient replacement for a timestamp.

Kafka integrates this unique abstraction with traditional publish/subscribe messaging concepts (such as producers, consumers, and brokers), parallelism, and enterprise features for improved performance and fault tolerance.

The original use case for Kafka was to track user behavior on websites. Site activity (page views, searches, or other actions users might take) is published to central topics, with one topic per activity type.

Kafka can be used to monitor operational data, aggregating statistics from distributed applications to produce centralized data feeds. It also works well for log aggregation, with low latency and convenient support for multiple data sources.

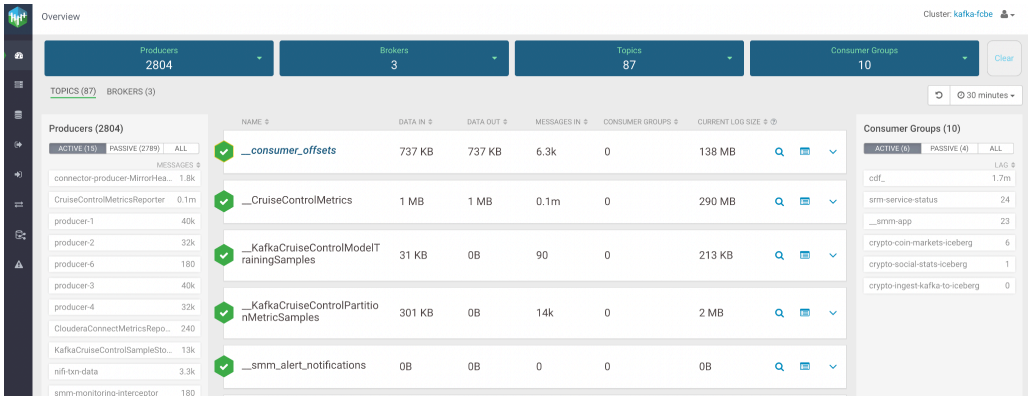
Kafka provides the following:

- Persistent messaging with O(1) disk structures, meaning that the execution time of Kafka's algorithms is independent of the size of the input. Execution time is constant, even with terabytes of stored messages.
- High throughput, supporting hundreds of thousands of messages per second, even with modest hardware.
- Explicit support for partitioning messages over Kafka servers. It distributes consumption over a cluster of consumer machines while maintaining the order of the message stream.
- Support for parallel data load into Hadoop.

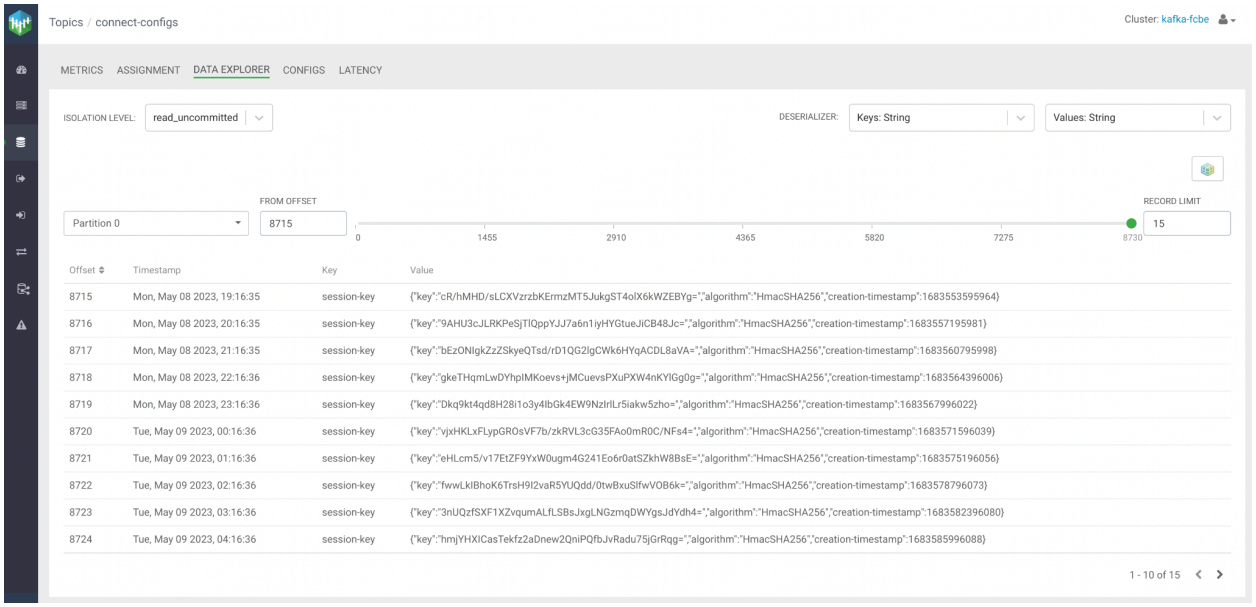
Stream Messaging Manager

Streams Messaging Manager (SMM) is an operations monitoring and management tool that provides end-to-end visibility in an enterprise Apache Kafka® environment.

With SMM, you can gain clear insights about your Kafka clusters. You can understand the end-to-end flow of message streams from producers to topics to consumers. SMM helps you troubleshoot your Kafka environment to identify bottlenecks, throughputs, consumer patterns, traffic flow etc. SMM enables you to analyze the stream dynamics between producers and consumers using various filters. You can optimize your Kafka environment based on the key performance insights gathered from various brokers and topics. With the tight integration of Apache Atlas, you can gain complete data lineage across multiple Kafka hops, producers and consumers with powerful data flow visualization.



Stream Messaging Manager also allows for real time view of the contents of the topics on web UI with Data Explorer



Simplifies troubleshooting Kafka environments

SMM provides intelligence-based filtering that allows a user to select a producer, broker, topic or consumer and see only related entities based on the selection. SMM is

smart enough to show only those producers that are sending data to the selected topics and show only those consumer groups that are consuming from those topics. The filtering works on the selection of any of the four entities. This enables users to quickly hone in on the root cause when troubleshooting and debugging Kafka issues.

Visualizes end-to-end Kafka stream flows

Another powerful feature of SMM is its ability to visualize all the data streams/flows across all your Kafka clusters. You can select any entity and visualize how data flows with respect to the entity selected.

Extends monitoring and management capabilities with REST API

SMM offers REST endpoints for all of its capabilities. This enables developers to integrate SMM with their other enterprise tools such as APM or case or ticketing systems.

Tracks data lineage and governance from edge-to-enterprise

SMM has been fully integrated with Apache Atlas for governance and data lineage, Apache Ranger for security and access control management, Apache Ambari for infrastructure level monitoring and lifecycle actions for the cluster, and Grafana to be able to graph Kafka metrics over time.

Visualize lineage between producers and consumers

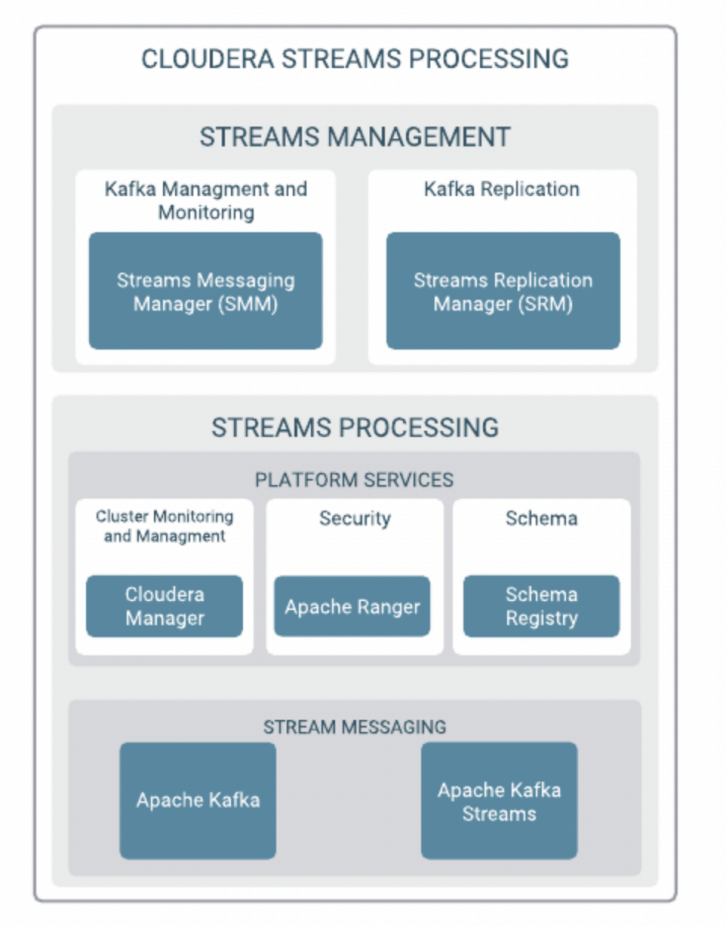
With SMM, you can visualize the lineage between producers and consumers.

Lineage information helps you to understand how the message is moving from a producer to a consumer group and which topics or partitions are part of that flow. This is just a high level information, intended for basic monitoring. For the fully featured lineage monitoring, see Atlas and its Kafka monitoring capabilities in Kafka lineage**.**

Schema Registry

Schema Registry provides a shared repository of schemas that allows applications to flexibly interact with each other.

As displayed in the following diagram, Schema Registry is part of the enterprise services that power streams processing.



Applications built often need a way to share metadata across three dimensions:

- Data format
- Schema
- Semantics or meaning of the data
-

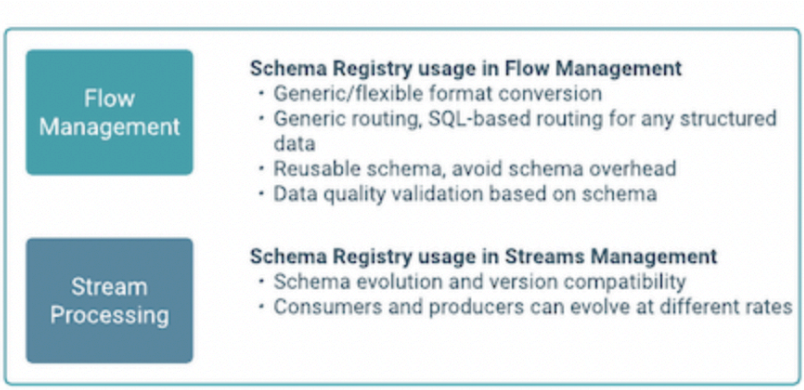
The Schema Registry design principle is to provide a way to tackle the challenges of managing and sharing schemas between components. The schemas are designed to support evolution such that a consumer and producer can understand different versions of those schemas but still read all information shared between both versions and safely ignore the rest.

Hence, the value that Schema Registry provides and the applications that integrate with it are the following:

- Centralized registry
  Provide reusable schema to avoid attaching schema to every piece of data

- Version management
  Define relationship between schema versions so that consumers and producers can evolve at different rates
- Schema validation
  Enable generic format conversion, generic routing and data quality

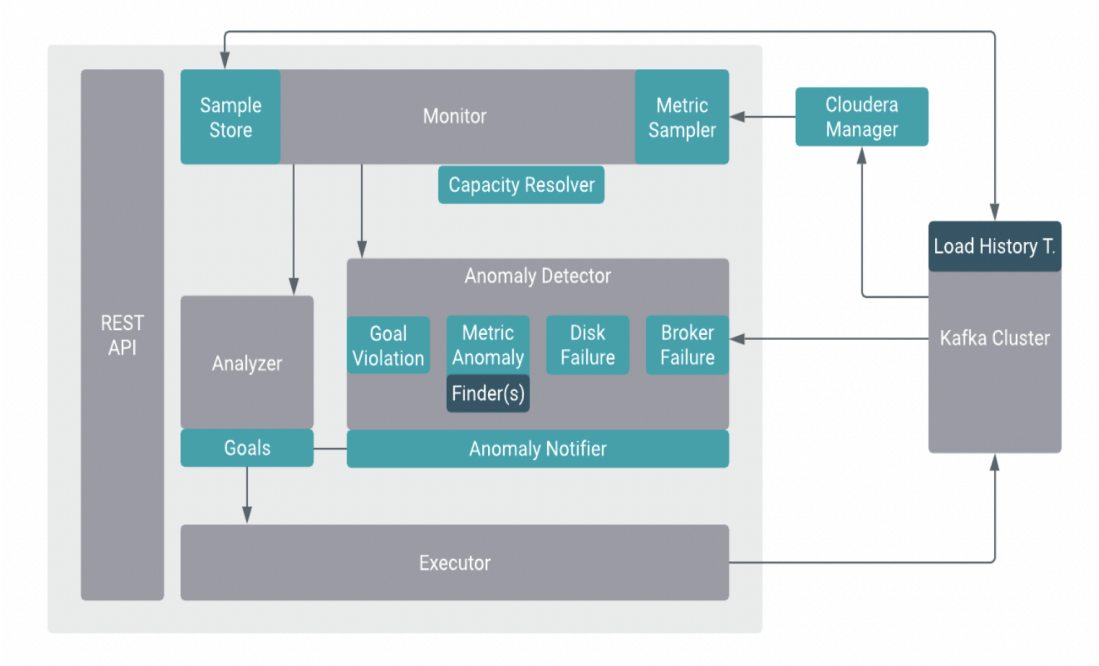
The following image displays Schema Registry usage in Flow and Streams Management:



Cruise Control

You can use Cruise Control as a load balancing component in large Kafka installations to automatically balance the partitions based on specific conditions for your deployment. The elements in the Cruise Control architecture are responsible for different parts of the rebalancing process that uses Kafka metrics and optimization goals.

The following illustration shows the architecture of Cruise Control.



Load Monitor

Generates a cluster workload model based on standard Kafka metrics and resource metrics to utilize disk, CPU, bytes-in rate and bytes-out rate. Feeds the cluster model into Anomaly Detector and Analyzer.

Analyzer

Generates optimization proposals based on optimization goals provided by the user, and cluster workload model from Load Monitor. Hard goals and soft goals can be set. Hard goals must be fulfilled, while soft goals can be left unfulfilled if hard goals are reached. The optimization fails if the hard goal is violated by optimization results.

Anomaly Detector

Responsible for detecting anomalies that can happen during the rebalancing process. The following anomaly detections are supported in Cruise Control:

- Broker failure
- Goal violations
- Disk failure
- Slow broker as Metric Anomaly
- Topic replication factor

The detected anomalies can be resolved by the self-healing feature of Cruise Control. For more information, see the How Cruise Control self-healing works documentation.

Executor

Carries out the optimization proposals and it can be safely interrupted when executing proposals. The executions are always resource-aware processes.

Cloudera Stream Analytics

Cloudera Streaming Analytics (CSA) offers real-time stream processing and streaming analytics powered by Apache Flink. Flink implemented on CDP provides a flexible streaming solution with low latency that can scale to large throughput and state. In addition to Flink, CSA includes SQL Stream Builder (SSB) to offer data analytical experience using SQL queries on your data streams.

Key features of Cloudera Streaming Analytics

Apache Flink : CSA is powered by Apache Flink that offers a framework for real-time stream processing and streaming analytics. CSA offers the features and functionalities of the upstream Apache Flink integrated on CDP Private Cloud Base.

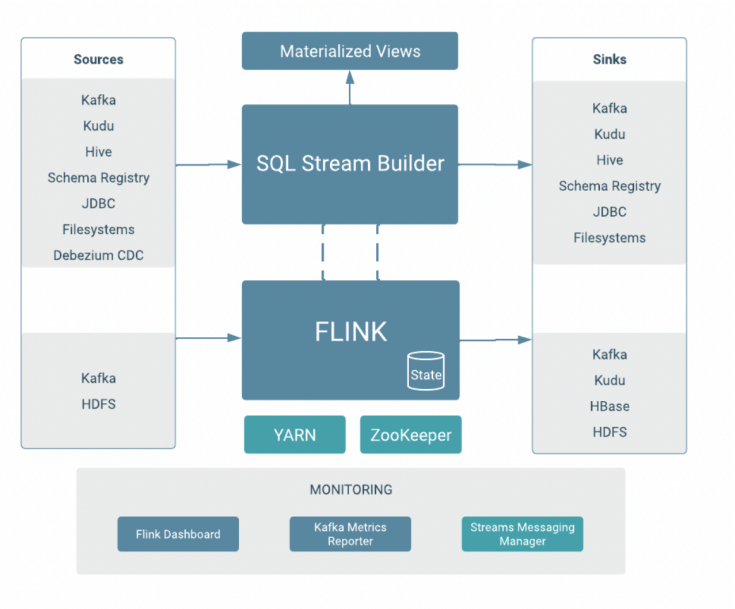
SQL Stream Builder : SQL Stream Builder is a job management interface to compose and run Continuous Streaming SQL on streams using Apache Flink as an engine, as well as to create REST APIs for the results.

Streaming Platform : For streaming analytics, CSA fits into a complete streaming platform augmented by Apache Kafka, Schema Registry, Streams Messaging Manager in the Cloudera Runtime stack.

Supported Connectors : CSA offers a set of connectors for Flink and SSB from which you can choose from based on your requirements. Kafka, HBase, HDFS, Kudu and Hive connectors are available for Flink. Kafka, HDFS/S3, JDBC and a set of the Debezium CDC connectors are available for SSB. Other than the connectors, SSB also supports Schema Registry, Hive and Kudu as catalogs.

Monitoring Solutions : In CDP Private Cloud Base, Streams Messaging components provide tools that support the operational needs of CSA. For example, when Kafka is used as a connector, you can use Kafka Metrics Reporter and Streams Messaging

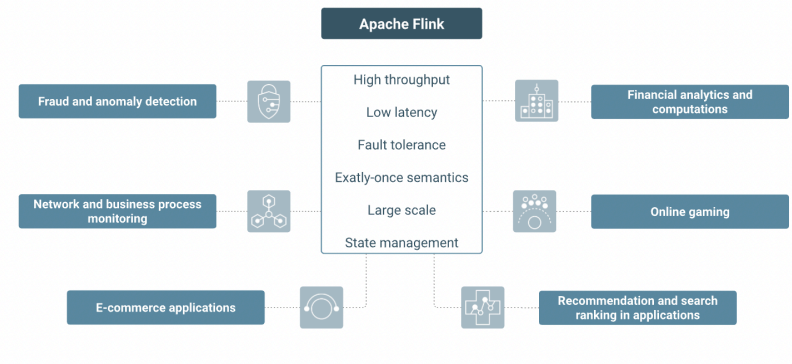
Manager (SMM) for Kafka management and alerting of Kafka actions. Beside SMM, you can use the Flink Dashboard to monitor your Flink and SSB jobs.



Apache Flink

Flink is a distributed processing engine and a scalable data analytics framework. You can use Flink to process data streams at a large scale and to deliver real-time analytical insights about your processed data with your streaming application.

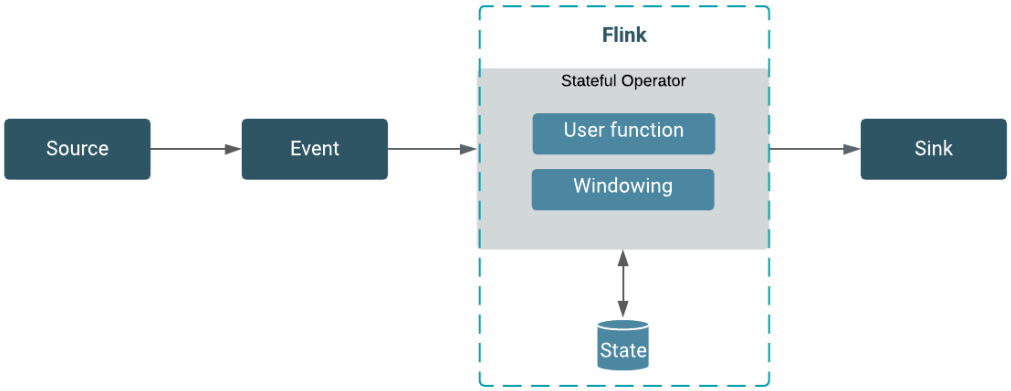
Flink is designed to run in all common cluster environments, perform computations at in-memory speed and at any scale. Furthermore, Flink provides communication, fault tolerance, and data distribution for distributed computations over data streams. A large variety of enterprises choose Flink as a stream processing platform due to its ability to handle scale, stateful stream processing, and event time.



Core features of Flink

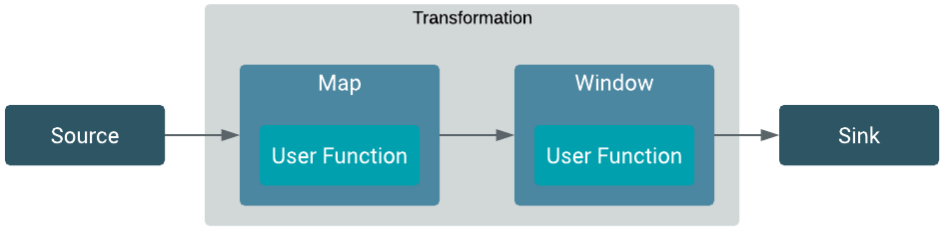
Architecture

The two main components for the task execution process are the Job Manager and Task Manager. The Job Manager on a master node starts a worker node. On a worker node the Task Managers are responsible for running tasks and the Task Manager can also run more than one task at the same time. The resource management for the tasks are completed by the Job manager in Flink. In a Flink cluster, Flink jobs are executed as YARN applications. HDFS is used to store recovery and log data, while ZooKeeper is used for high availability coordination for jobs.



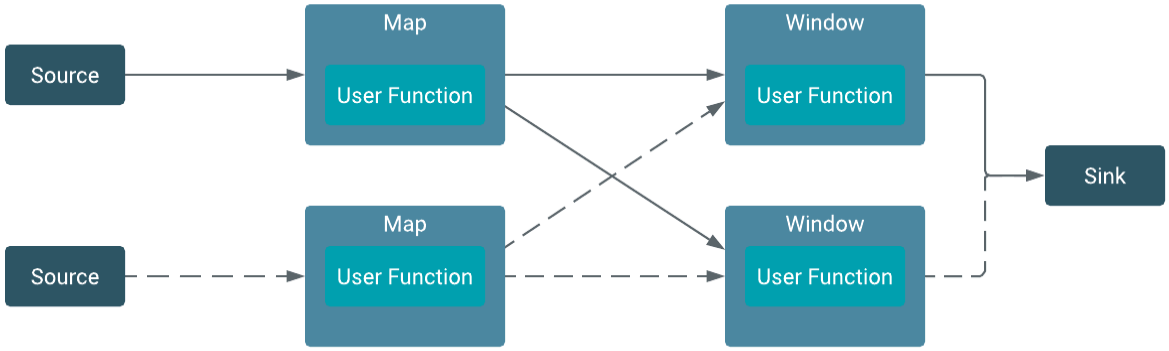
DataStream API

The DataStream API is used as the core API to develop Flink streaming applications using Java or Scala programming languages. The DataStream API provides the core building blocks of the Flink streaming application: the datastream and the transformation on it. In a Flink program, the incoming data streams from a source are transformed by a defined operation which results in one or more output streams to the sink.



Operators

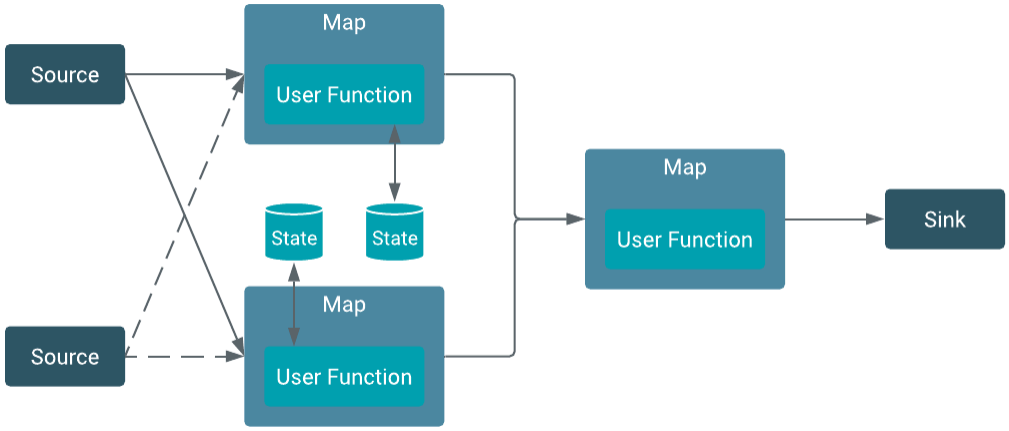
Operators transform one or more DataStreams into a new DataStream. Programs can combine multiple transformations into sophisticated data flow topologies. Other than the standard transformations like map, filter, aggregation, you can also create windows and join windows within the Flink operators. On a dataflow one or more operations can be defined which can be processed in parallel and independently to each other. With windowing functions, different computations can be applied to different streams in the defined time window to further maintain the processing of events. The following image illustrates the parallel structure of dataflows.



State and state backend

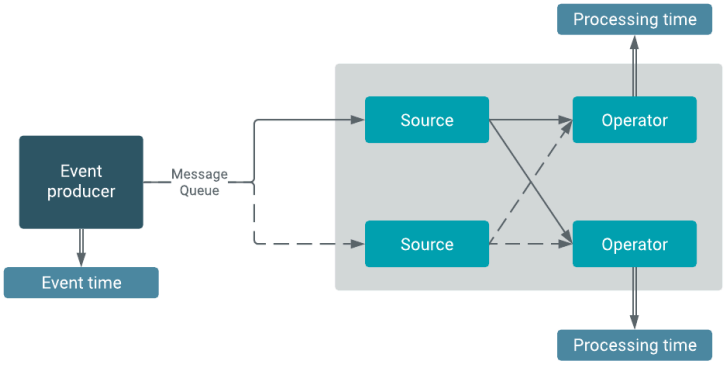
Stateful applications process dataflows with operations that store and access information across multiple events. You can use Flink to store the state of your application locally in state backends that guarantee lower latency when accessing your processed data. You can also create checkpoints and savepoints to have a fault-tolerant

backup of your streaming application on a durable storage.

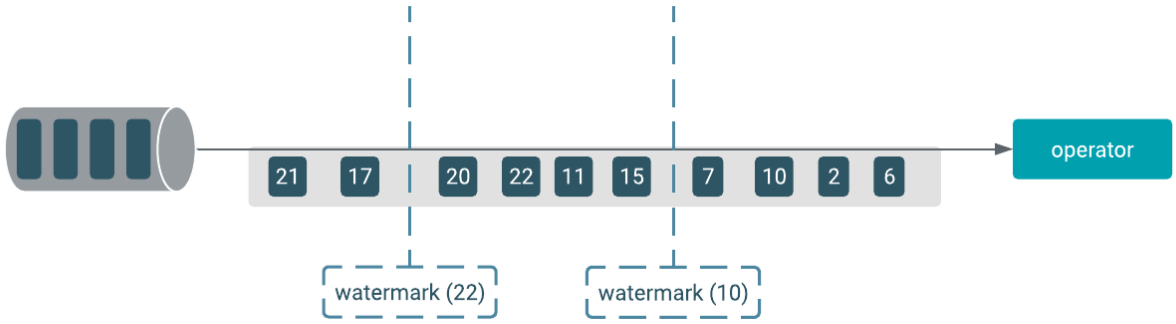


Event time and watermark

In time-sensitive cases where the application uses alerting or triggering functions, it is important to distinguish between event time and processing time. To make the designing of applications easier, you can create your Flink application either based on the time when the event is created or when it is processed by the operator.

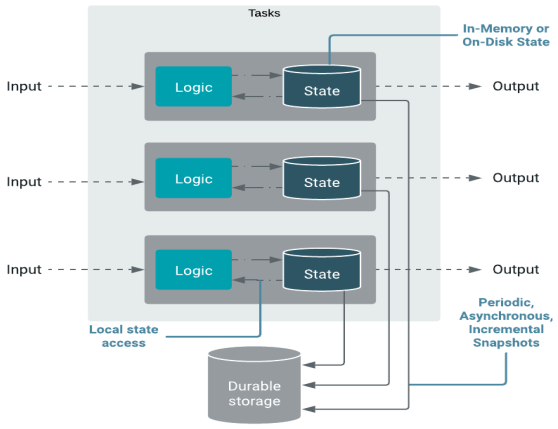


With only the event time, it is not clear when the events are processed in the application. To track the time for an event time based application, watermark can be used.



Checkpoints and savepoints

Checkpoints and savepoints can be created to make the Flink application fault tolerant throughout the whole pipeline. Flink contains a fault tolerance mechanism that creates snapshots of the data stream continuously. The snapshot includes not only the dataflow, but the state attached to it. In case of failure, the latest snapshot is chosen and the system recovers from that checkpoint. This guarantees that the result of the computation can always be consistently restored. While checkpoints are created and managed by Flink, savepoints are controlled by the user. A savepoint can be described as a backup from the executed process.

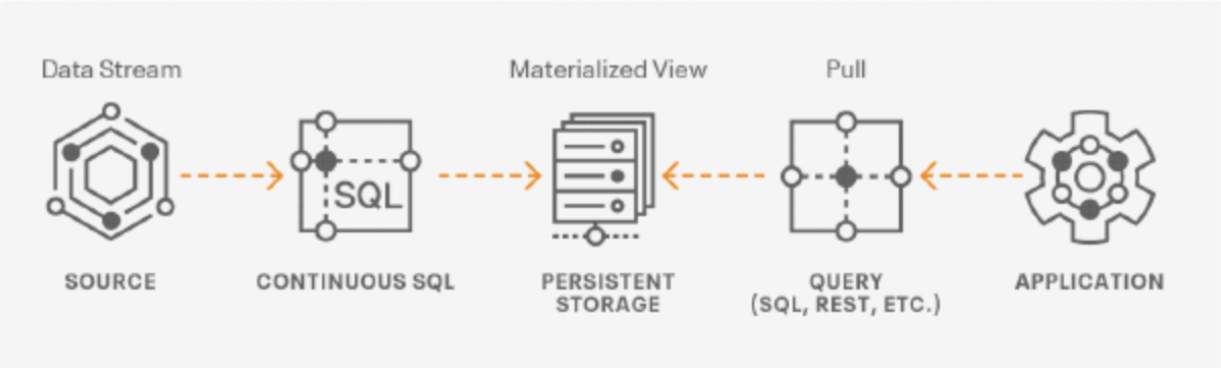


SQL Stream Builder for Continuous Stream

Cloudera Streaming Analytics offers an easy to use and interactive SQL Stream Builder as a service to create queries on streams of data through SQL.

The SQL Stream Builder (SSB) is a comprehensive interactive user interface for creating stateful stream processing jobs using SQL. By using SQL, you can simply and easily declare expressions that filter, aggregate, route, and otherwise mutate streams of data. SSB is a job management interface that you can use to compose and run SQL on streams, as well as to create durable data APIs for the result

SSB runs Structured Query Language (SQL) statements continuously, this is called Continuous SQL or Streaming SQL. Continuous SQL can run against both bounded and unbounded streams of data. The results are sent to a sink of some type, and can be connected to other applications through a Materialized View interface. Compared to traditional SQL, in Continuous SQL the data has a start, but no end. This means that queries continuously process results. When you define your job in SQL, the SQL statement is interpreted and validated against a schema. After the statement is executed, the results that match the criteria are continuously returned.



SSB runs in an interactive fashion where you can quickly see the results of your query and iterate on your SQL syntax. The executed SQL queries run as jobs on the Flink cluster, operating on boundless streams of data until canceled. This allows you to author, launch, and monitor stream processing jobs within SSB as every SQL query is a Flink job. You can use Flink and submit Flink jobs without using Java, as SSB automatically builds and runs the Flink job in the background. A sample of SSB SQL with inbound and outbound joins.

As a result of Flink integration, you are able to use the basic functionalities offered by Flink. You can choose exactly once processing, process your data stream using event time, save your jobs with savepoints, and use Flink SQL to create tables and use connectors based on your requirements. As a result of the various connectors, you are able to enrich your streaming data with data from slowly changing connectors.

Key features of SSB

SQL Stream Builder (SSB) within Cloudera supports out-of-box integration with Flink. Using Flink SQL, you can create tables directly from the Streaming SQL Console window or built-in templates. For integration with Business Intelligence tools you can create Materialized Views.

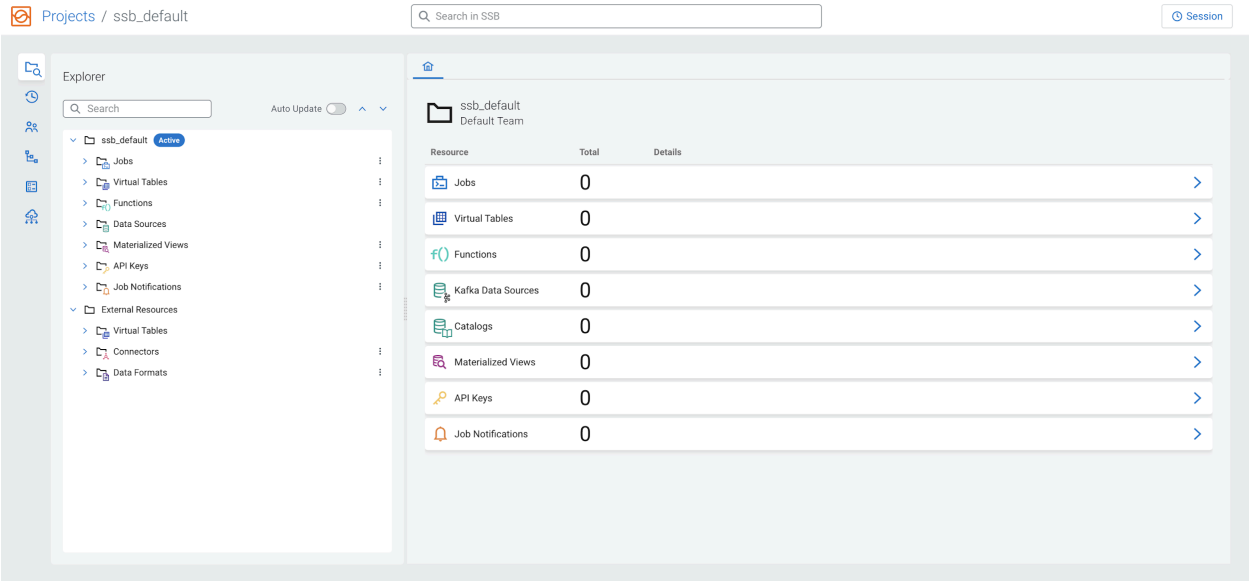
Flink SQL - SQL Stream Builder allows you to use Data Definition Language (DDL), Data Manipulation Language (DML) and Query Language directly from the Streaming SQL Console.

Change Data Capture - SQL Stream Builder supports PostgreSQL, Oracle, MySQL, Db2 and SQL Server as Debezium connectors using Flink SQL. With Change Data Capture (CDC), you can capture changes in your databases and update your applications with the newly added data.

Session Cluster - When you start SSB, a session cluster is deployed to share and maintain resources. The submitted SQL jobs are executed as Flink jobs in the same session cluster that share a Job Manager. The properties for the session cluster can be viewed using the Streaming SQL Console, and can be modified with SET statements in the SQL window.

Built-in Templates - The Build-in Templates in SSB allows you to quickly and simply create tables for your SQL queries. You only need to provide the connection and job specific information to the template to use it in SSB.

Streaming SQL Console - SSB comes with an interactive user interface that allows you to easily create, and manage your SQL jobs in one place. It allows you to create and iterate on SQL statements with robust tooling and capabilities. Query parsing is logged to the console, and results are sampled back to the interface to help with iterating on the SQL statement as required.



Materialized Views - SSB has the capability to materialize results from a Streaming SQL query to a persistent view of the data that can be read through REST and over the PG wire protocol. Applications can use this mechanism to query streams of data in a way of high performance without deploying additional database systems. Materialized Views are built into the SQL Stream Builder service, and require no configuration or maintenance. The Materialized Views act like a special kind of sink, and can even be used in place of a sink. They require no indexing, storage allocation, or specific management.

Input Transform - In case you are not aware of the incoming data structure or raw data is being collected from for example sensors, you can use the Input Transform to clean up and organize the incoming data before querying. Input transforms also allow access to Kafka header metadata directly in the query itself. Input transforms are written in Javascript and compiled to Java bytecode deployed with the Flink jar.

User-defined Functions - You can create customized and complex SQL queries by using User-defined Functions to enrich your data, apply computations or a business logic on it. User defined functions are written in Javascript or Java language

Cloudera Flow Management

Cloudera Flow Management (CFM) powered by Apache NiFi and NiFi Registry is a key part of the Cloudera DataFlow (CDF) platform. SBI can use CFM to ingest and manage data without writing a single line of code. It helps to capture any type of data from any source and move it into the enterprise securely, while SBI transforms and dynamically routes this data in between.

SBI can build complex and scalable data flows, and can manage, edit, and monitor these flows in real time with CFM. SBI can also adopt a DevOps style flow development lifecycle to deliver flow applications faster by using templates, version control, collaboration, and various deployment options.

Cloudera Flow Management integrates seamlessly with CDF components such as Kafka, MiNiFi, Flink, and Spark Streaming, or platform components such as HBase, Hive, Impala, and Kudu.

CFM has two main components:

- **Apache NiFi** – the core data ingestion engine with a no-code graphical user interface. It supports processors for connectivity, transformation, and content routing.
- **Apache NiFi Registry** – the companion to NiFi that enables DevOps style development of flows. It supports versioning flows, promoting flows, and deploying flow applications across environments.

High Level Overview of Key CFM Features

The key features categories include flow management, ease of use, security, extensible architecture, and flexible scaling model.

1. **Flow Management**

Guaranteed Delivery - A core philosophy of NiFi has been that even at very high scale, guaranteed delivery is a must. This is achieved through effective use of a purpose-built persistent write-ahead log and content repository. Together they are designed in such a way as to allow for very high transaction rates, effective load-spreading, copy-on-write, and play to the strengths of traditional disk read/writes.

Data Buffering w/ Back Pressure and Pressure Release - NiFi supports buffering of all queued data as well as the ability to provide back pressure as those queues reach specified limits or to age off data as it reaches a specified age (its value has perished).

Prioritized Queuing - NiFi allows the setting of one or more prioritization schemes for how data is retrieved from a queue. The default is oldest first, but there are times when data should be pulled newest first, largest first, or some other custom scheme.

Flow Specific QoS (latency v throughput, loss tolerance, etc.) - There are points of a dataflow where the data is absolutely critical and it is loss intolerant. There are also times when it must be processed and delivered within seconds to be of any value. NiFi enables the fine-grained flow specific configuration of these concerns.

2. **Ease of Use**

Visual Command and Control - Dataflows can become quite complex. Being able to visualize those flows and express them visually can help greatly to reduce that complexity and to identify areas that need to be simplified. NiFi enables not only the visual establishment of dataflows but it does so in real-time. Rather than being 'design and deploy' it is much more like molding clay. If you make a change to the dataflow that change immediately takes effect. Changes are fine-grained and isolated to the affected components. You don't need to stop an entire flow or set of flows just to make some specific modification.

Flow Templates - Dataflows tend to be highly pattern oriented and while there are often many different ways to solve a problem, it helps greatly to be able to share those best practices. Templates allow subject matter experts to build and publish their flow designs and for others to benefit and collaborate on them.

Data Provenance - NiFi automatically records, indexes, and makes available provenance data as objects flow through the system even across fan-in, fan-out, transformations, and more. This information becomes extremely critical in supporting compliance, troubleshooting, optimization, and other scenarios.

Recovery / Recording a rolling buffer of fine-grained history - NiFi's content repository is designed to act as a rolling buffer of history. Data is removed only as it ages off the

content repository or as space is needed. This combined with the data provenance capability makes for an incredibly useful basis to enable click-to-content, download of content, and replay, all at a specific point in an object's lifecycle which can even span generations.

3. **Security**

System to System - A dataflow is only as good as it is secure. NiFi at every point in a dataflow offers secure exchange through the use of protocols with encryption such as 2-way SSL. In addition NiFi enables the flow to encrypt and decrypt content and use shared-keys or other mechanisms on either side of the sender/recipient equation.

User to System - NiFi enables 2-Way SSL authentication and provides pluggable authorization so that it can properly control a user's access and at particular levels (read-only, dataflow manager, admin). If a user enters a sensitive property like a password into the flow, it is immediately encrypted server side and never again exposed on the client side even in its encrypted form.

Multi-tenant Authorization - The authority level of a given dataflow applies to each component, allowing the admin user to have a fine grained level of access control. This means each NiFi cluster is capable of handling the requirements of one or more organizations. Compared to isolated topologies, multi-tenant authorization enables a self-service model for data flow management, allowing each team or organization to manage flows with a full awareness of the rest of the flow, to which they do not have access.

4. **Extensible Architecture**

Extension - NiFi is at its core built for extension and as such it is a platform on which data flow processes can execute and interact in a predictable and repeatable manner. Points of extension include: processors, Controller Services, Reporting Tasks, Prioritizers, and Customer User Interfaces.

Classloader Isolation - For any component-based system, dependency problems can quickly occur. NiFi addresses this by providing a custom class loader model, ensuring that each extension bundle is exposed to a very limited set of dependencies. As a result, extensions can be built with little concern for whether they might conflict with another extension. The concept of these extension bundles is called 'NiFi Archives' and is discussed in greater detail in the Developer's Guide.

Site-to-Site Communication Protocol - The preferred communication protocol between NiFi instances is the NiFi Site-to-Site (S2S) Protocol. S2S makes it easy to transfer data from one NiFi instance to another easily, efficiently, and securely. NiFi client libraries can be easily built and bundled into other applications or devices to communicate back to NiFi via S2S. Both the socket based protocol and HTTP(S) protocol are supported in S2S as the underlying transport protocol, making it possible to embed a proxy server into the S2S communication.

5. **Flexible Scaling Model**

Scale-out (Clustering) - NiFi is designed to scale-out through the use of clustering many nodes together as described above. If a single node is provisioned and configured to handle hundreds of MB per second, then a modest cluster could be configured to handle GB per second. This then brings about interesting challenges of load balancing and fail-over between NiFi and the systems from which it gets data. Use of asynchronous queuing based protocols like messaging services, Kafka, etc., can help. Use of NiFi's 'site-to-site' feature is also very effective as it is a protocol that allows NiFi and a client (including another NiFi cluster) to talk to each other, share information about loading, and to exchange data on specific authorized ports.

Scale-up & down - NiFi is also designed to scale-up and down in a very flexible manner. In terms of increasing throughput from the standpoint of the NiFi framework, it is possible to increase the number of concurrent tasks on the processor under the Scheduling tab when configuring. This allows more processes to execute simultaneously, providing greater throughput. On the other side of the spectrum, you can perfectly scale NiFi down to be suitable to run on edge devices where a small footprint is desired due to limited hardware resources.

Apache NiFi

Apache NiFi is a powerful, flexible, extensible, and reliable system to distribute and process data. It is data source agnostic and supports sources of different formats, schemas, protocols, speeds, and sizes. SBI can design and automate complex data flows on a configurable platform using 300+ processors. SBI can build data flow by simply dragging and dropping a series of processors using the intuitive graphical user interface of Apache NiFi. When the data flow is ready, those can be monitored and controlled in real time.

Put simply, NiFi was built to automate the flow of data between systems. While the term 'data flow' is used in a variety of contexts, we use it here to mean the

automated and managed flow of information between systems. This problem space has been around ever since enterprises had more than one system, where some of the systems created data and some of the systems consumed data. The problems and solution patterns that emerged have been discussed and articulated extensively. A comprehensive and readily consumed form is found in the Enterprise Integration Patterns.

Some of the high-level challenges of dataflow include:

Systems fail - Networks fail, disks fail, software crashes, people make mistakes.

Data access exceeds capacity to consume - Sometimes a given data source can outpace some part of the processing or delivery chain - it only takes one weak-link to have an issue.

Boundary conditions are mere suggestions - will invariably get data that is too big, too small, too fast, too slow, corrupt, wrong, or in the wrong format.

What is noise one day becomes signal the next - Priorities of an organization change - rapidly. Enabling new flows and changing existing ones must be fast.

Systems evolve at different rates - The protocols and formats used by a given system can change anytime and often irrespective of the systems around them. Dataflow exists to connect what is essentially a massively distributed system of components that are loosely or not-at-all designed to work together.

Compliance and security - Laws, regulations, and policies change. Business to business agreements change. System to system and system to user interactions must be secure, trusted, accountable.

Continuous improvement occurs in production - It is often not possible to come even close to replicating production environments in the lab.

Over the years data flow has been one of those necessary evils in an architecture. Now though there are a number of active and rapidly evolving movements making dataflow a lot more interesting and a lot more vital to the success of a given enterprise. These include things like; Service Oriented Architecture, the rise of the API and API2, Internet of Things, and Big Data. In addition, the level of rigor necessary for compliance, privacy, and security is constantly on the rise. Even still with all of these new concepts coming about, the patterns and needs of dataflow are still largely the same. The primary differences then are the scope of complexity, the rate of change necessary to adapt, and that at scale the edge case becomes a common occurrence. NiFi is built to help tackle these modern dataflow challenges.

Core Concepts of NiFi

NiFi's fundamental design concepts closely relate to the main ideas of Flow Based Programming (FBP). Here are some of the main NiFi concepts and how they map to FBP:

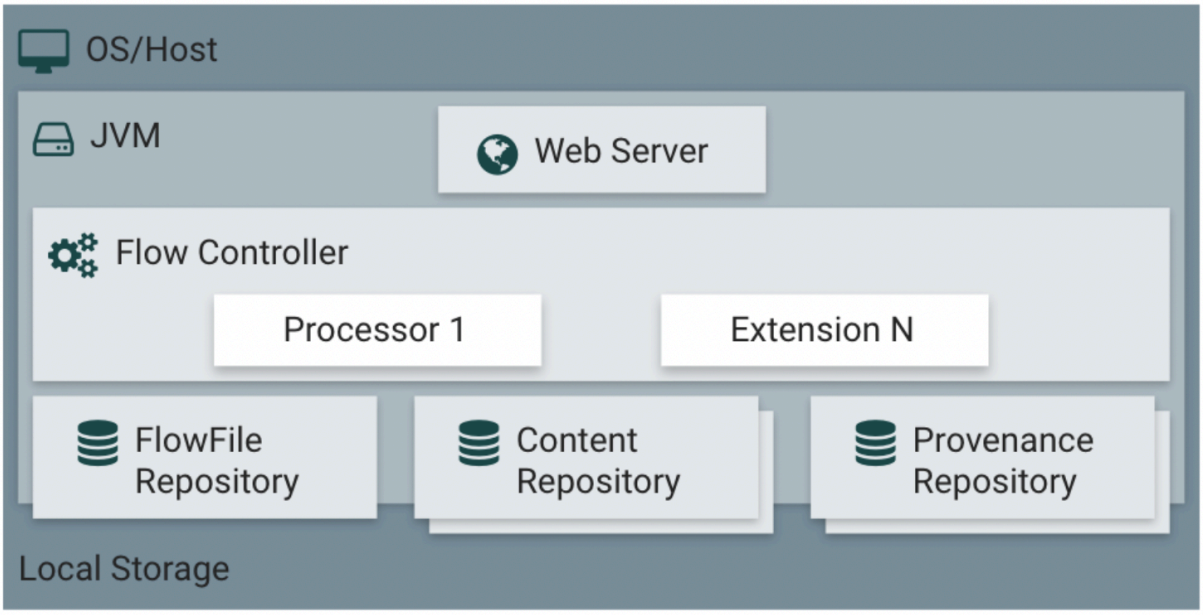
FlowFile	Information Packet	A FlowFile represents each object moving through the system and for each one, NiFi keeps track of a map of key/value pair attribute strings and its associated content of zero or more bytes.
FlowFile Processor	Black Box	Processors actually perform the work. In Enterprise Integration Patterns terms a processor is doing some combination of data routing, transformation, or mediation between systems. Processors have access to attributes of a given FlowFile and its content stream. Processors can operate on zero or more FlowFiles in a given unit of work and either commit that work or rollback.
Connection	Bounded Buffer	Connections provide the actual linkage between processors. These act as queues and allow various processes to interact at differing rates. These queues can be prioritized dynamically and can have upper bounds on load, which enable back pressure.

Flow Controller	Scheduler	The Flow Controller maintains the knowledge of how processes connect and manages the threads and allocations thereof which all processes use. The Flow Controller acts as the broker facilitating the exchange of FlowFiles between processors.
Process Group	subnet	A Process Group is a specific set of processes and their connections, which can receive data via input ports and send data out via output ports. In this manner, process groups allow creation of entirely new components simply by composition of other components.

This design model, also similar to SEDA, provides many beneficial coSBIquences that help NiFi to be a very effective platform for building powerful and scalable dataflows. A few of these benefits include:

- Lends well to visual creation and management of directed graphs of processors
- 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 units
- The resource constrained connections make critical functions such as back-pressure and pressure release very natural and intuitive
- Error handling becomes as natural as the happy-path rather than a coarse grained catch-all
- The points at which data enters and exits the system as well as how it flows through are well understood and easily tracked

Apache NiFi Architecture



NiFi executes within a JVM on a host operating system. The primary components of NiFi on the JVM are as follows:

Web Server

The purpose of the web server is to host NiFi's HTTP-based command and control API.

Flow Controller

The flow controller is the brains of the operation. It provides threads for extensions to run on, and manages the schedule of when extensions receive resources to execute.

Extensions

There are various types of NiFi extensions which are described in other documents. The key point here is that extensions operate and execute within the JVM.

FlowFile Repository

The FlowFile Repository is where NiFi keeps track of the state of what it knows about a given FlowFile that is presently active in the flow. The implementation of the repository is pluggable. The default approach is a persistent Write-Ahead Log located on a specified disk partition.

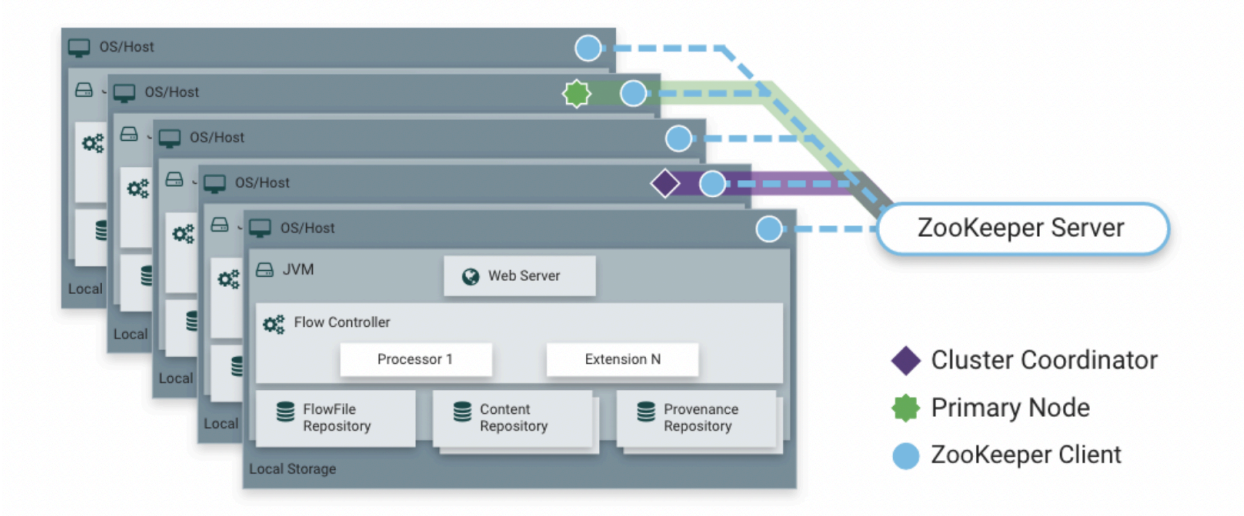
Content Repository

The Content Repository is where the actual content bytes of a given FlowFile live. The implementation of the repository is pluggable. The default approach is a fairly simple mechanism, which stores blocks of data in the file system. More than one file system storage location can be specified so as to get different physical partitions engaged to reduce contention on any single volume.

Provenance Repository

The Provenance Repository is where all provenance event data is stored. The repository construct is pluggable with the default implementation being to use one or more physical disk volumes. Within each location event data is indexed and searchable.

NiFi is also able to operate within a cluster.



In NiFi Clusters a Zero-Leader Clustering paradigm is employed. Each node in a NiFi cluster performs the same tasks on the data, but each operates on a different set of data. Apache ZooKeeper elects a single node as the Cluster Coordinator, and failover is handled automatically by ZooKeeper. All cluster nodes report heartbeat and status information to the Cluster Coordinator. The Cluster Coordinator is responsible for disconnecting and connecting nodes. Additionally, every cluster has one Primary Node, also elected by ZooKeeper. As a DataFlow manager, SBI can interact with the NiFi cluster through the user interface (UI) of any node. Any change made is replicated to all nodes in the cluster, allowing for multiple entry points.

Apache NiFi Registry

Registry is a complimentary application that provides a central location for storing and managing shared and versioned data flow resources. It helps with the full

flow life cycle development. It integrates with NiFi to allow storing, retrieving, and upgrading versioned flows. When designing a data flow in NiFi, we may need to define and test different approaches and work incrementally towards the final goal. NiFi Registry helps to keep track of several flow versions and navigate between them easily

NiFi registry helps with CI/CD automation for moving flows between environments and versioning them as they move to production clusters.

Cloudera Machine Learning

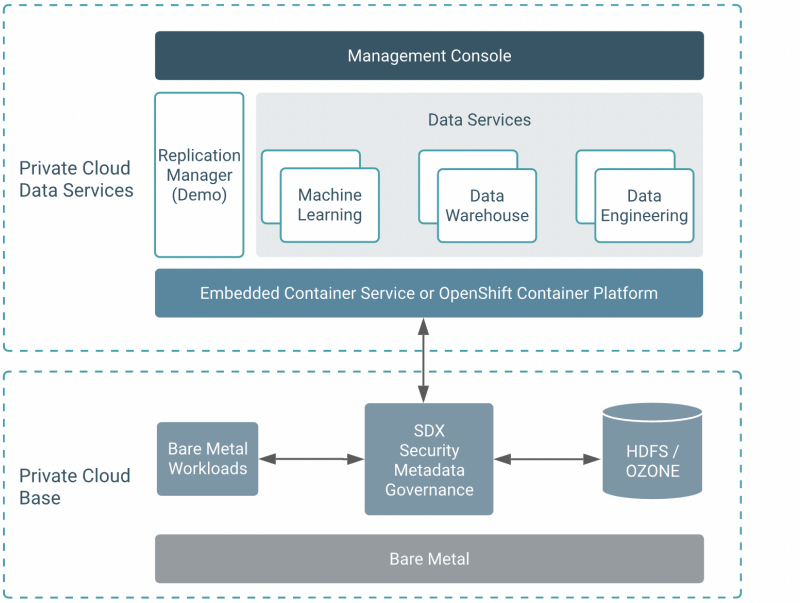
Cloudera Machine Learning is an offering under CDP Private Cloud Data Services, which is a CDP product that brings many of the benefits of the public cloud to the data center.

CDP Private Cloud provides disaggregation of compute and storage allowing independent scaling of compute and storage clusters. The Data Services provide containerized compute analytic applications that scale dynamically and can be upgraded independently. Through the use of containers deployed on Kubernetes, CDP Private Cloud Data Services brings both agility and predictable performance to analytic applications. CDP Private Cloud Data Services gets unified security, governance, and metadata management through Cloudera Shared Data Experience (SDX), which is available on a CDP Private Cloud Base cluster.

CDP Private Cloud Data Services users can rapidly provision and deploy services such as Cloudera Data Warehousing, Cloudera Machine Learning, and Cloudera Data Engineering through the Management Console, and easily scale them up or down as required.

A CDP Private Cloud Data Services deployment requires a Private Cloud Base cluster and container-based clusters to run the Data Services. SBI can either use a dedicated RedHat OpenShift cluster or deploy an Embedded Container Service (ECS) for the containers.

The Private Cloud deployment process involves configuring Management Console, registering an environment by providing details of the Data Lake configured on the Base cluster, and then creating the workloads.



Cloudera Machine Learning

Machine learning has become one of the most critical capabilities for modern businesses to grow and stay competitive today. From automating internal processes to optimizing the design, creation, and marketing processes behind virtually every product consumed, ML models have permeated almost every aspect of our work and personal lives.

Cloudera Machine Learning (CML) is Cloudera's new cloud-native machine learning service, built for CDP. The CML service provisions clusters, also known as ML workspaces, that run natively on Kubernetes.

Each ML workspace enables teams of data scientists to develop, test, train, and ultimately deploy machine learning models for building predictive applications all on the data under management within the enterprise data cloud. ML workspaces are ephemeral, allowing SBI to create and delete them on-demand. ML workspaces support fully-containerized execution of Python, R, Scala, and Spark workloads through flexible and extensible engines. CML works as a natural extension to the CDP Private Cloud base with dedicated compute nodes to run the executions, while sharing the same data from the CDP base environment with the same security, governance and Lineage tracking capabilities.

Cloudera Machine Learning uses Docker containers to deliver application components and run isolated user workloads. On a per project basis, users can run R,

Python, and Scala workloads with different versions of libraries and system packages. CPU and memory are also isolated, ensuring reliable, scalable execution in a multi-tenant setting. CML engines are responsible for running R, Python, and Scala code written by users. You can think of an engine as a virtual machine, customized to have all the necessary dependencies while keeping each project's environment entirely isolated.

To enable multiple users and concurrent access, CML transparently subdivides and schedules containers across multiple hosts. This scheduling is done using Kubernetes, a container orchestration system used internally by CML. Neither Docker nor Kubernetes are directly exposed to end users, with users interacting with Cloudera Machine Learning through a web application.

Base Engine Image

The base engine image is a Docker image that contains all the building blocks needed to launch a Cloudera Machine Learning session and run a workload. It consists of kernels for Python, R, and Scala along with additional libraries that can be used to run common data analytics operations. When a session is launched to run a project, an engine is kicked off from a container of this image. The base image itself is built and shipped along with Cloudera Machine Learning.

Cloudera Machine Learning offers legacy engines and Machine Learning Runtimes. Both legacy engines and ML Runtimes are Docker images and contain OS, interpreters, and libraries to run user code in sessions, jobs, experiments, models, and applications. However, there are significant differences between these choices. See ML Runtimes versus Legacy Engines for a summary of these differences.

New versions of the base engine image are released periodically. However, existing projects are not automatically upgraded to use new engine images. Older images are retained to ensure to be able to test code compatibility with the new engine before upgrading to it manually.

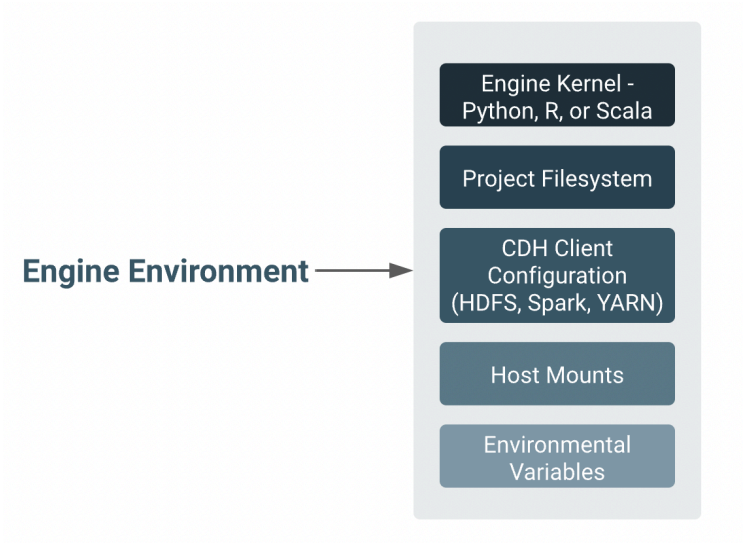
Engine

The term engine refers to a virtual machine-style environment that is created when you run a project (via session or job) in Cloudera Machine Learning. You can use an engine to run R, Python, and Scala workloads on data stored in the underlying CDH cluster.

Cloudera Machine Learning allows you to run code using either a session or a job. A session is a way to interactively launch an engine and run code while a job lets

you batch process those actions and schedule them to run recursively. Each session and job launches its own engine that lives as long as the workload is running (or until it times out).

A running engine includes the following components:



CML supports end-to-end ML lifecycle right from data ingestion, exploration, sampling, experiments, Model training, ML registry followed by model deployment and monitoring. Some feature details are listed below.

Experiments

Cloudera Machine Learning allows data scientists to run batch experiments that track different versions of code, input parameters, and output (both metrics and files). Data Scientists often perform a series of runs with different parameters and samples of data and a small change can have significant impact on the result, it is very important to be able to repeat a particular outcome and that is where experiments will help.

Cloudera Machine Learning uses experiments to facilitate ad-hoc batch execution and model training. Experiments are batch executed workloads where the code, input parameters, and output artifacts are versioned. This feature also provides a lightweight ability to track output data, including files, metrics, and metadata for comparison.

Collaboration

CML allows for different collaboration strategies to choose per SBI's requirements.

Projects

Work closely with trusted colleagues on a particular project, you can add them to the project as collaborators. This is recommended for collaboration over projects created under your personal account. Anyone who belongs to your organization can be added as a project collaborator.

Different access levels cal be set at project level such as Viewer, Operator, Contributor, Admin

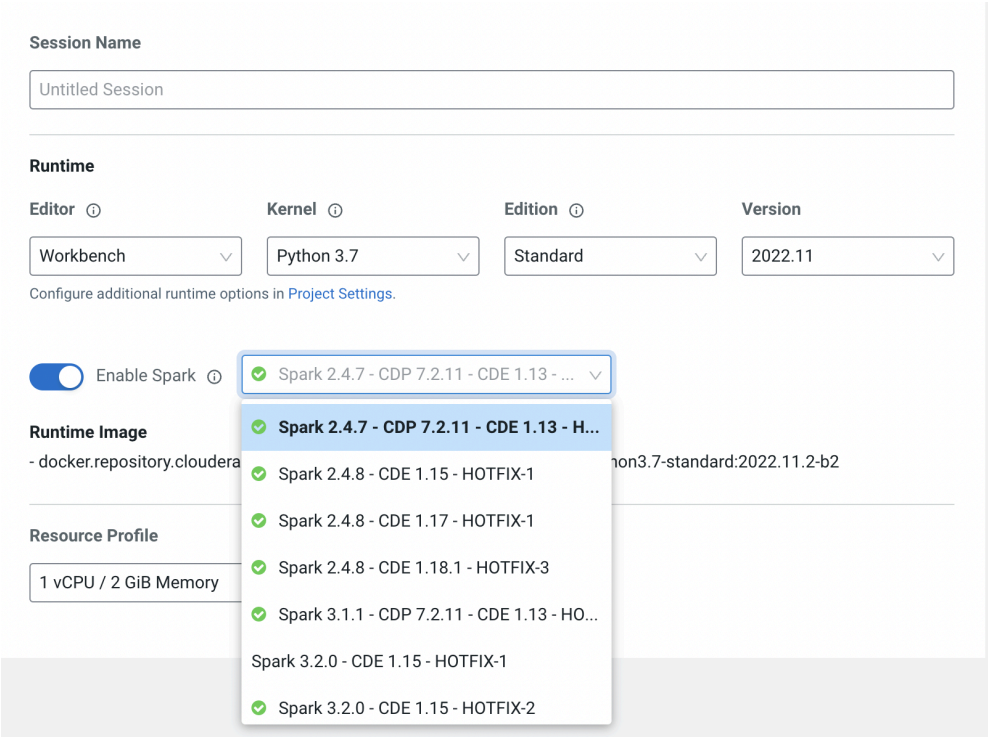
Teams

Users who work together on more than one project and want to facilitate collaboration can create a Team. Teams allow streamlined administration of projects. Team projects are owned by the team, rather than an individual user. Only users that are already part of the team can be added as collaborators to projects created within the team context. Team administrators can add or remove members at any time, assigning each member different access permissions**.**

CML also allows for other options like forking projects and collaboration through Git etc.

Editors and Kernels

CML allows for a choice of editors like workbench, Zeppelin , jupyter and others as well. It also allows for various kernels like Python, R with choice of versions. CML supports Spark2 and Spark3 as well.



Applied ML Prototypes (AMPs)

Applied ML Prototypes (AMPs) provide reference example machine learning projects in Cloudera Machine Learning. More than simplified quickstarts or tutorials, AMPs are fully-developed expert solutions created by Cloudera's research arm, Fast Forward Labs.

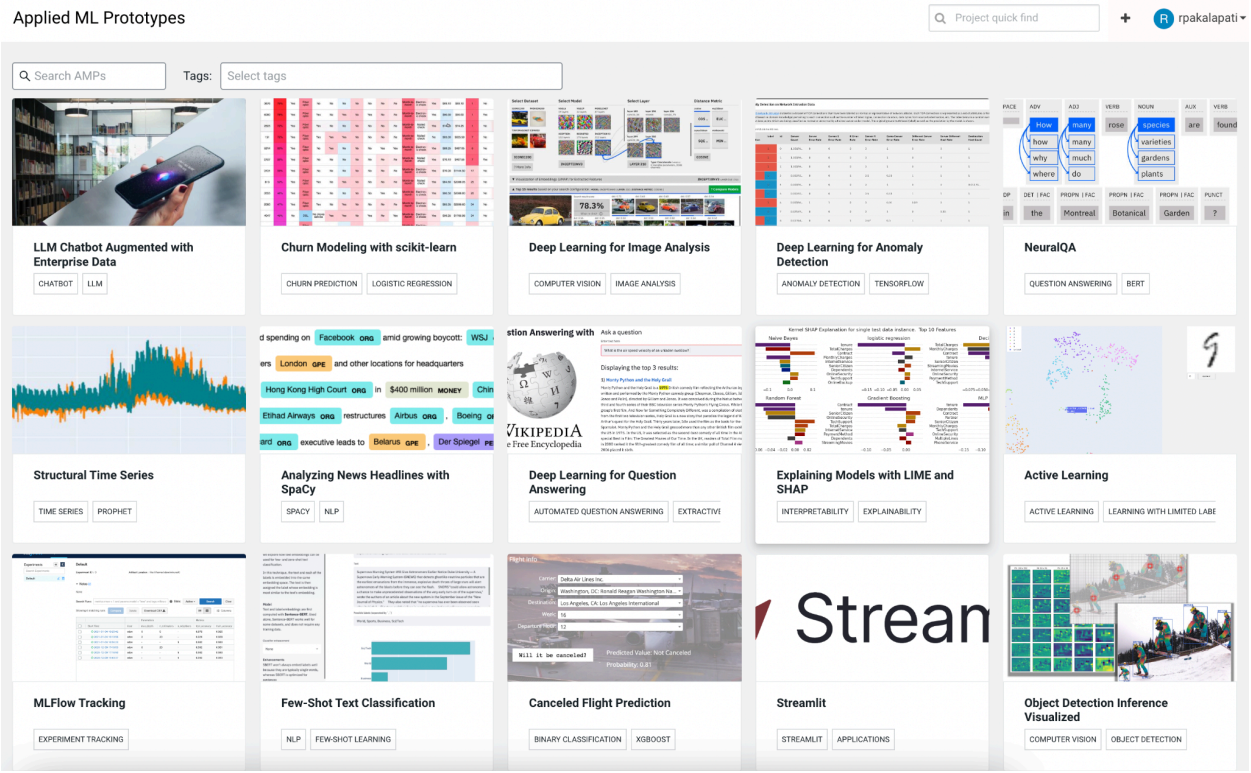
These solutions to common problems in the machine learning field demonstrate how to fully use the power of Cloudera Machine Learning. AMPs show you how to create Cloudera Machine Learning projects to solve your own use cases.

AMPs are available to install and run from the Cloudera Machine Learning user interface. As new AMPs are developed, they will become available to you for your study and use.

It's simple to get started with AMPs.

- Log in to your Cloudera Machine Learning workspace, and in the left panel click AMPs.
- Click on an AMP tile to read its description.

- Click Configure Project and provide any configuration values required by the AMP. The Description field explains how to determine these configuration values. After you click Launch Project, the installation process may take several minutes.
- When the installation is complete, click Overview to read the documentation for the AMP and explore the code and project structure.

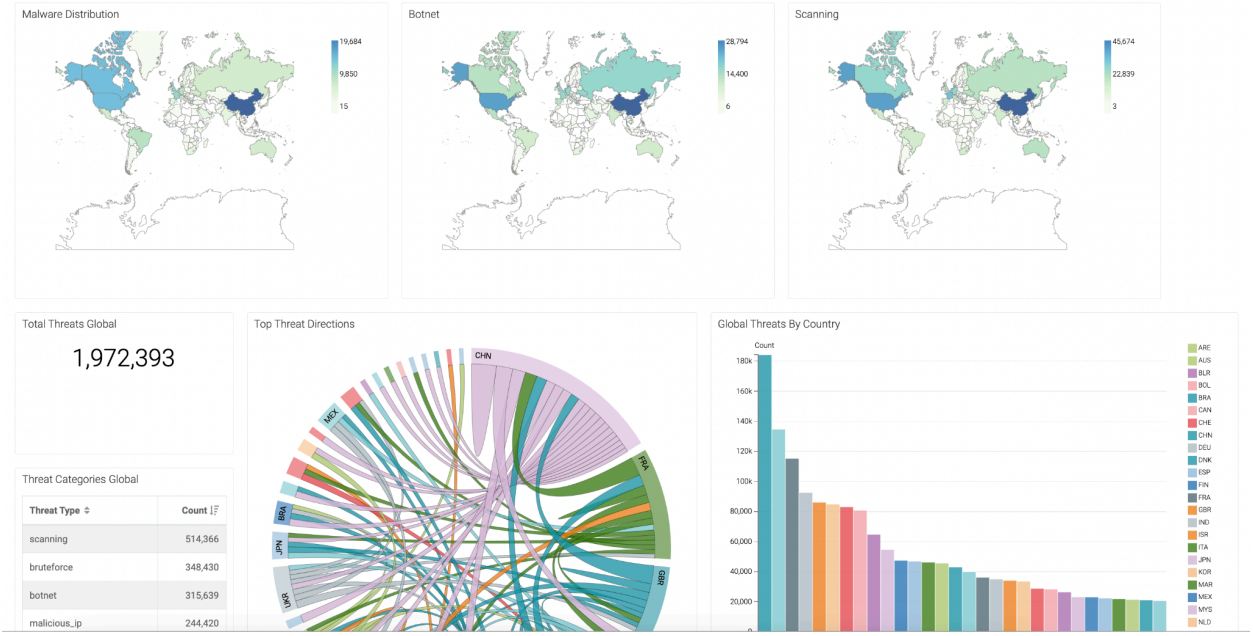


Cloudera Data Visualization

Cloudera Data Visualization enables you to explore data and communicate insights across the whole data lifecycle by using visual objects. The fast and easy self-service data visualization streamlines collaboration in data analytics through the common language of visuals.

Using this rich visualization layer enables you to accelerate advanced data analysis. The web-based, no-code, drag-and-drop user interface is highly intuitive and enables you to build customized visualizations on top of your datasets, build dashboards and applications, and publish them anywhere across the data lifecycle. This solution allows for customization and collaboration, and it provides you with a dynamic and data-driven insight into your business.

Cloudera Data Visualization is integrated with Cloudera Machine Learning (CML) in all form factors. You can use the same visualization tool for structured, unstructured/text, and ML analytics, which means deeper insights and more advanced dashboard applications. You can create native data visualizations to provide easy predictive insights for business users and accelerate production ML workflows from raw data to business impact.



Model Registry

Model Registry stores and manages machine learning models and associated metadata, such as the model's version, dependencies, and performance. The registry enables MLOps and facilitates the development, deployment, and maintenance of machine learning models in a production environment.

Model Registry includes functionality for the following tasks:

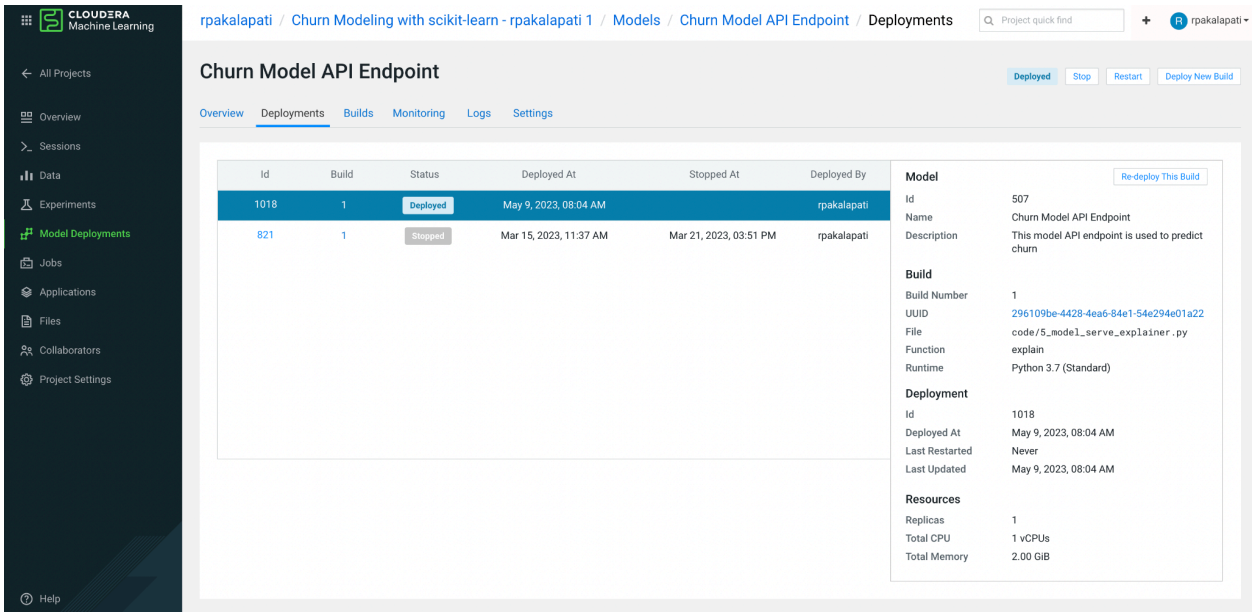
- Storing and organizing different versions of a machine learning model and its associated metadata.
- Tracking the lineage of a model, including who created it, when it was created, and any changes made to it over time.
- Managing dependencies between models and other assets, such as data sets and code.

- Providing APIs for accessing and deploying models, as well as for querying and searching the registry.
- Integrating with CI/CD pipelines and other tools used in the MLOps workflow.

Model registries help organizations improve the quality and reliability of their machine learning models by providing a centralized location for storing and managing models, as well as enabling traceability and reproducibility of model development. They also make deploying and managing models in a production environment easier by providing a single source for model versions and dependencies.

The Model Registry integrates MLFlow and maintains compatibility with the open source ecosystem.

Models in the registry are secure by default, each one has an access key which is needed for requests to be made.



Model Deployment & Monitoring

Cloudera Machine learning allows end users to easily deploy the models from model registry by specifying necessary resources for the deployment. Cloudera Machine Learning allows you to specify a number of replicas that will be deployed to serve requests. For each active model, you can monitor its replicas by going to the model's Monitoring page. On this page you can track the number of requests being served by each replica, success and failure rates, and their associated stderr and stdout logs. Depending on future resource requirements, you can increase or decrease the number of replicas by re-deploying the model.

Security and Governance for ML

CML allows for several security features like LDAP authentication, SSL, SSH keys, Authorization etc. Governance is also an out of the box feature for tracking model lineage, so we know where the model is coming from and where it is relevant.

The Machine Learning (ML) projects, model builds, model deployments, and associated metadata are tracked in Apache Atlas, which is available in the environment's SDX cluster. You can also specify additional metadata to be tracked for a given model build. For example, you can specify metadata that links training data to a project through a special file called the linking file (lineage.yaml).

Platform Security and Governance

Cloudera Data Platform comes with a host of services and components for SBI to implement a robust security and governance framework. Cloudera Runtime security and governance is managed by Apache Ranger, Apache Knox, and Apache Atlas.

Security Overview

The CDP platform is secure by design. We treat security as the top-tier design requirement to manage risk, reduce attack surface and vulnerabilities, and develop design concepts and patterns in an industry-preferred way.

We have designed CDP to meet technical audit requirements out-of-the-box. In fact, many CDP services require security components prior to their installation. The CDP platform gives you the ability to manage your cluster securely, allowing you to audit and understand how and what has happened on your cluster, and manage data policy and governance.

To ensure the security of your data, Cloudera follows the Pillars of Security: authentication, authorization, encryption, and identity management.

Authentication

Authentication is a basic security requirement in a digital environment.Users, processes, and services must verify their identity using credentials known by the authenticator in order to access a system

The components of CDP that provide authentication solutions to your cluster are Apache Knox and Kerberos. Apache Knox provides perimeter security so that the enterprise can extend access to new users while also maintaining compliance with security policies. Kerberos is a network authentication protocol designed to provide strong authentication for client/server applications by using secret-key cryptography.

Authorization

Authorization controls what users can and cannot access within a system. These permissions include viewing, editing, using, or controlling information. In a physical environment, this can also include accessing servers, rooms, and devices.

Apache Ranger is a component of CDP that provides authorization solutions to your cluster. Ranger defines the policies that determine what users can and cannot access.

- **Row-level filtering** helps simplify Hive queries. By moving the access restriction logic down into the Hive layer, Hive applies the access restrictions every time data access is attempted. This helps simplify authoring of the Hive query, and

provides seamless behind-the-scenes enforcement of row-level segmentation without having to add this logic to the predicate of the query.

- **Data Masking** can be done using dynamic resource-based or tag-based to protect sensitive data in Hive in near real-time. You can set policies that mask or anonymize sensitive data columns (such as PII, PCI, and PHI) dynamically from Hive query output. For example, you can mask sensitive data within a column to show only the first or last four characters.
- **Time-bound policies** can also be created using Ranger validity periods that enable you to configure a policy to be effective for a specified time range. You can add a validity period to both resource-based and tag-based policies.

Encryption

Encryption increases the level of security in a digital environment.

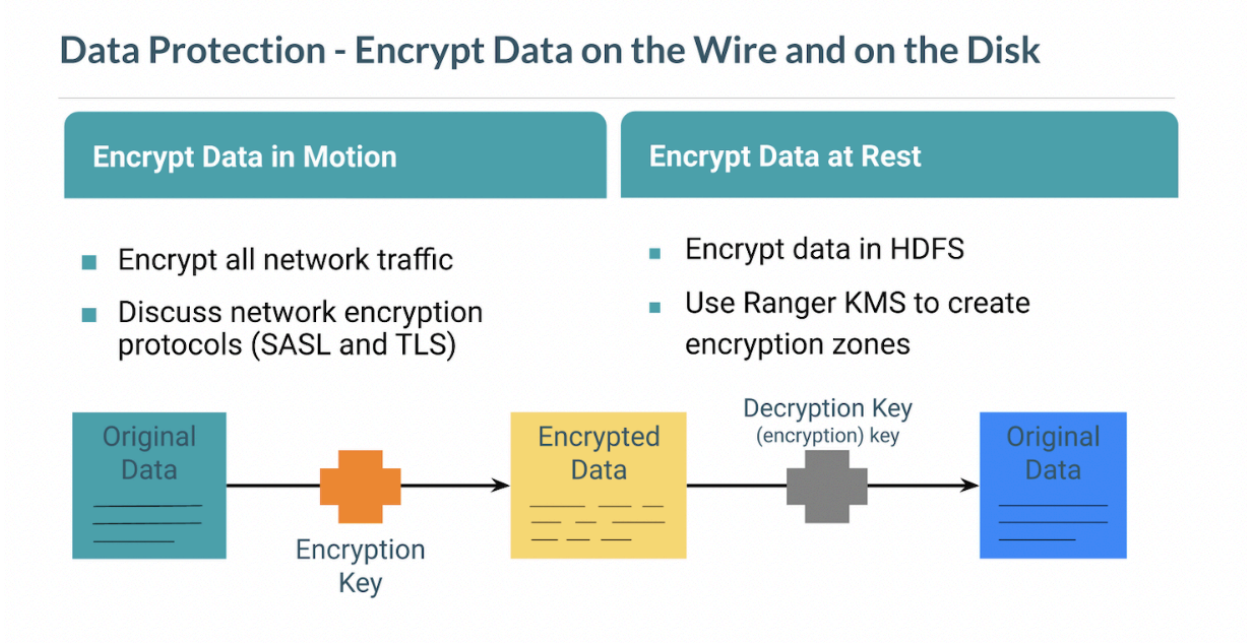
Encryption protects sensitive data through encoding, so the data can only be accessed or decoded with a digital key. Encryption applies to both data at-rest and data in-transit.

Data at-rest, or data that is stored physically on a storage device such as a hard drive or cloud storage and not actively moving, is encrypted and digital keys are distributed to authorized users to decrypt it when needed.

Data en route between its source and its destination is known as data in-transit. End-to-end encryption utilizes protocols like Transparent Layer Security (TLS) to protect data in-transit. Each data transmitting session has a new digital key, which relies on proper authentication and can reduce the risk of unauthorized access.

CDP provides four different components for encryption solutions: Ranger KMS, Key Trustee Server, Key HSM, and Navigator Encrypt.

Ranger extends the Hadoop KMS functionality by allowing you to store keys in a secure database. The Key Trustee Server is a key manager that stores and manages cryptographic keys and other security artifacts. Key HSM allows the Key Trustee Server to seamlessly integrate with a hardware security module (HSM). Navigator Encrypt transparently encrypts and secures data at rest without requiring changes to your applications.



Apache Ranger manages access control through a user interface that ensures consistent policy administration in CDP clusters. Ranger is the one single unified UI to control authorization to all components within the CDP cluster like HDFS, Ozone, HMS Hive, Kafka Topics, NiFI etc.

Security administrators can define security policies at the database, table, column, and file levels, and can administer permissions for groups or individual users. Rules based on dynamic conditions such as time or geolocation can also be added to an existing policy rule. Ranger security zones enable you to organize service resources into multiple security zones.

Ranger also provides a centralized framework for collecting access audit history and reporting data, including filtering on various parameters.

Apache Knox Gateway (“Knox”) is a system to extend the reach of Apache™ Hadoop® services to users outside of a Hadoop cluster without reducing Hadoop Security. Knox also simplifies Hadoop security for users who access the cluster data and execute jobs. The Knox Gateway is designed as a reverse proxy.

Establishing user identity with strong authentication is the basis for secure access in Hadoop. Users need to reliably identify themselves and then have that identity propagated throughout the Hadoop cluster to access cluster resources.

Governance Overview

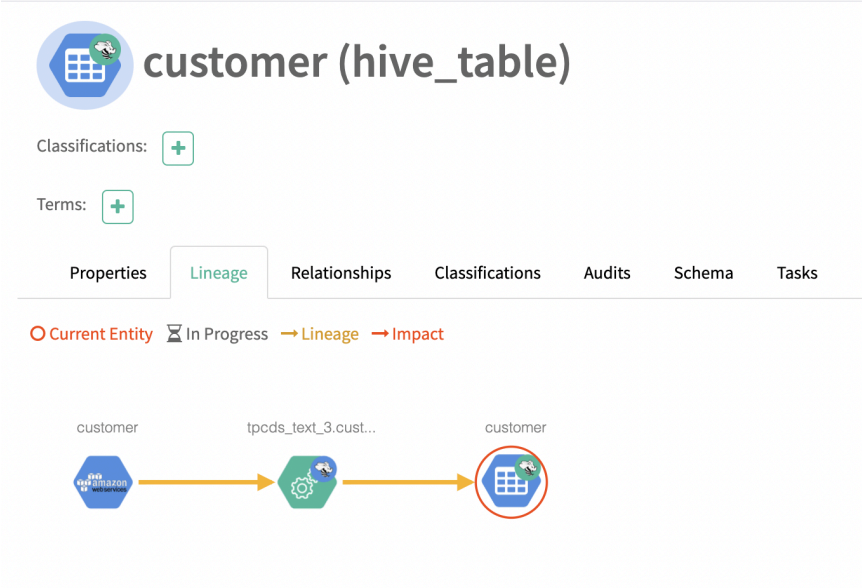
CDP platform comes with robust data governance components for effective collection, storage and usage of metadata in data context. All of this is done using Apache Atlas.

Apache Atlas

Apache Atlas is a metadata management and governance system designed to help you find, organize, and manage data assets. Atlas creates “entities” or metadata representations of objects and operations in your data lake. You can add business metadata to these entities so you can use business vocabulary to make it easier to search for specific assets.

Apache Atlas uses metadata to create lineage relationships

Atlas reads the content of the metadata it collects to build relationships among data assets. When Atlas receives query information, it notes the input and output of the query and generates a lineage map that traces how data is used and transformed over time. This visualization of data transformations allows governance teams to quickly identify the source of data and to understand the impact of data and schema changes.

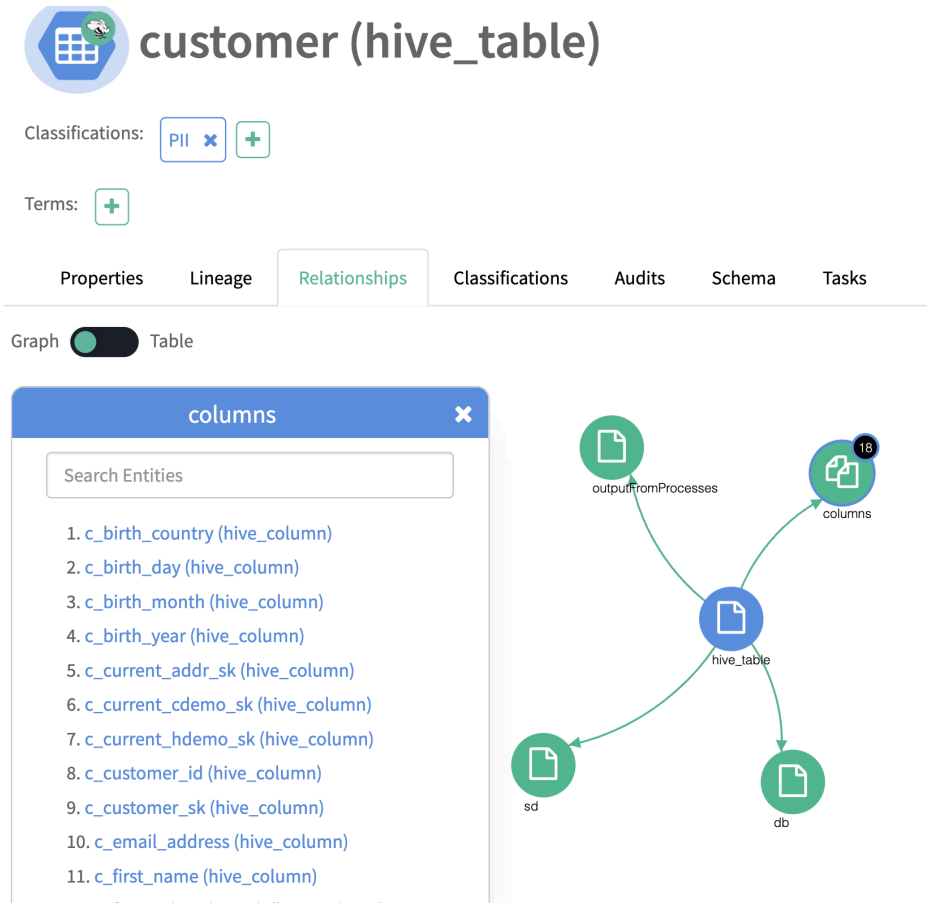


Adding to entity metadata makes searching easier

Atlas manages classifications and labels that you create and use to enhance the metadata for data assets. You can create and organize classifications and labels to use for anything from identifying data cleansing steps to recording user comments and insights on specific data assets.

Classification Propagation

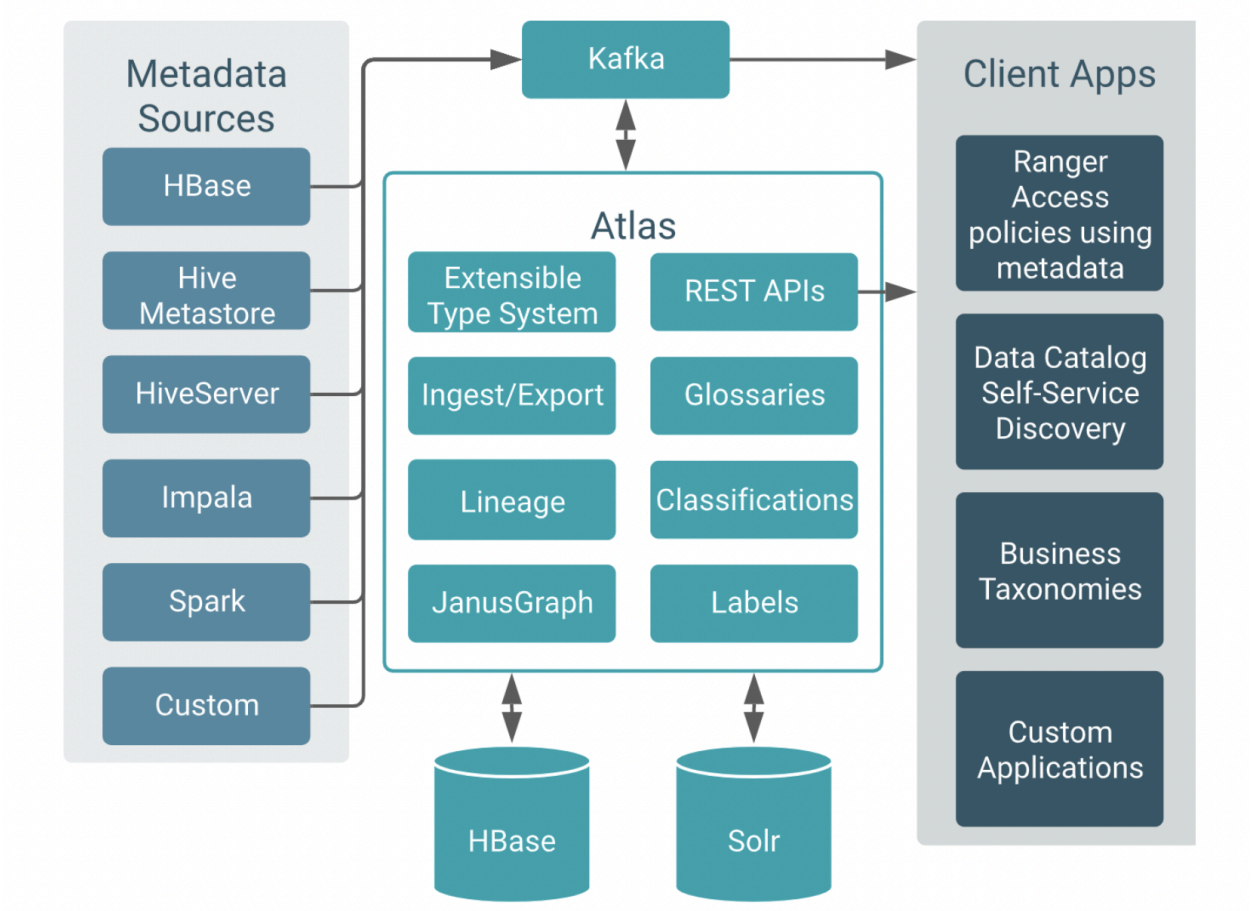
Atlas allows for propagation of tags in the lineage for classified data assets, One such data classification is PII tagging, we can tag a particular data asset as PII and any further data assets based on this are automatically classified as PII with propagation. When you use classifications, the Atlas Dashboard makes it easy to search, group, report, and further annotate the entities you label. Classifications themselves can be organized into hierarchies to make them easier to manage.



Atlas also provides an infrastructure to create and maintain business ontologies to label your data assets. Atlas' "glossaries" include "terms" so you can build agreed-upon lists for department- or organization-wide vocabulary to identify and manage data. Adding a term gives you a single-click report of entities identified by that term.

Apache Atlas architecture

Atlas runs as an independent service in a Hadoop environment. Many Hadoop data processing and storage services include Atlas add-ons that publish metadata for the services' activities to a Kafka message topic. Atlas reads the messages and stores them in JanusGraph to model the relationships among entities. The datastore behind JanusGraph is HBase. Atlas stores a search index in Solr to take advantage of Solr's search functionality.



Pre-defined hooks exist for Hive, Impala, Kafka, NiFi, Spark, and Sqoop.

Atlas also provides "bridges" that import metadata for all of the existing data assets in a given source. For example, if you start Atlas after you've already created

databases and tables in Hive, you can import metadata for the existing data assets using the Hive bridge. Bridges use the Atlas API to import the metadata rather than publishing messages to Kafka.

If you need a hook or bridge to automate collecting metadata from another source, use the Atlas Java API to create a custom Atlas addon.

Backup and Disaster Recovery

Cloudera Data Platform is a complex system, comprising many open-source projects that work in conjunction to build a unique data processing platform. It is the foundation for many large-scale installations across all possible industry verticals, storing and processing from hundreds of terabytes, to multiple petabytes worth of data. At any scale there is the need to keep the data safe, and the customary approach is to invest into some kind of backup technology. The difficulties with this approach are manifold:

- Data at scale becomes inherently difficult to move. The inertia of data requires new approaches that either do not move data at all, or keeps movement to a minimum.
- If it cannot be avoided to duplicate data, it is advantageous to keep the backup as an active copy to be able to run workloads on top of it. This allows splitting loads between clusters.
- Traditional backup systems are tailored to single, specific target systems, usually with standardized access protocols, such as JDBC or network shares. Hadoop, at times, does not conform to that.

A Hadoop cluster can include more than a dozen separate systems, with many storing their state in some kind of persistence layer, usually a database or the OSs (operating system) file system(s). Some of these subsystems have rudimentary support to store their data also on out-of-bounds services, such as NFS mounts. This does not address backup nor recovery properly, though could be employed as part of the overall backup architecture.

An additional complication is that even if a database is used by a particular system, that database is usually pluggable, such that many different variants can be chosen from. Often, there is also an option to employ an embedded database, such as Derby or SQLite, enabling some kind of quick start mode. It should be noted that this is never a good choice in a production system as these embedded databases also are usually single user systems, not allowing concurrent users – which is violating one of the core ideas of Hadoop. Further, they are not part of the overall monitoring and are difficult to back up while being in use.

Finally, there are systems used in combination with Hadoop, but considered not part of the core stack, such as Apache Storm, or the actual user applications running within the cluster. All of these make onboarding clients a difficult process, unless a company-wide policy is in place defining where data needs to be stored (as in the Information Architecture) by every user and application, service, or any other writing process.

Policies and Objectives

The various systems within the Hadoop ecosystems, and as bundled into a distribution such as Cloudera's CDP Private Cloud, all deal in some way with the data stored and processed. Yet, not all persist data the same way, or with the same scope and retention policies. It is vital to dissect the entire stack to identify what needs to be addressed during a backup, and what can possibly be guaranteed in terms of SLAs regarding said scope, retention, and also mean-time-to-recovery (MTTR). Usually these are expressed as Recovery Point Objective (RPO) or Recovery Time Objective (RTO), usually defined by the Business Continuity (BC) team.

- The RTO defines the time required for a service to be unavailable without incurring a significant impact on the business operations.
- The RPO defines the maximum time for which data is lost. It does not directly define the amount of data that can be lost, but more so what impact the loss of data means to the business.

Setting certain limits for RTO and RPO drives predominantly the choice of backup strategies, discussed below in detail. For example, requiring no RPO of more than one hour would preclude a backup once per day. On the other hand, asking for a low RTO often precludes constructing a new cluster from scratch, and rather have a warm or even hot standby cluster available. This means that the RPOs and RTOs often define the entire cluster architecture, since without the proper assumptions there is literally no way of automating a viable backup solution.

Common Archetypes for Multi-Cluster Active/Active

There are a number of architectures for replication in an EDH. These architectures all have different pros and cons and some may be a better fit depending on the use case. There are a number of considerations for these architectures and the list below includes some of the highest-level design considerations.

- **Size of the data:** What data and how much of it needs to be available for my business continuity
- **Compression rate:** While compression can mean getting much more data over a fixed pipe between clusters, it comes with trade offs.
- **Latency requirements:** Consider what the SLA is for the given dataset: seconds, minutes, or hours.

- **Underlying storage system:** The replication requirements and solutions vary significantly from HDFS (large batch), Apache Hive Tables (includes metadata), and Apache HBase (near real-time and event-based).

Below is a look at the high-level archetypes that span the majority of all replication requirements for an EDH. However, there may be some use cases that require modifications or customizations to these archetypes. Cloudera Enterprise allows organizations to push the boundaries of what is possible with data management and processing, and its flexibility and production-ready capabilities make it adaptable as new requirements emerge.

Replication Manager

Replication Manager is a service in Cloudera Manager where we can create replication policies in this service to replicate data across data centers for various use cases which include disaster recovery scenarios, running hybrid workloads, migrating data to/from cloud, or a generic backup/restore scenario. SBI can also create HDFS or HBase snapshot policies to take snapshots of HDFS directories and HBase tables respectively.

Cloudera Manager provides the following key functionalities in the Cloudera Manager Admin Console that can be leveraged by Replication Manager:

- Select datasets that are critical for SBI business operations.
- Monitor and track progress of snapshots and replication jobs through a central console and easily identify issues or files that failed to be transferred.
- Issue Alert when a snapshot or replication job fails or is aborted so that the problem can be diagnosed quickly.

SBI can also use Cloudera Manager to schedule, save, and restore snapshots of HDFS directories and HBase tables.

To have a complete backup and disaster recovery solution for the DW, we can make use of below replication policies

- HDFS/Ozone replication
- Hive external & ACID table replication

Batch Table/File Replication

Batch movement of data is the easiest form of replication when working with HDFS and/or Hive Tables. With Cloudera Replication Manager, , included with Cloudera Data Platform Private Cloud, SBI can easily define the following for a dataset (folder), database, or selection of tables:

- **What triggers replication:** Should replication happen on scheduled intervals, on-demand, or off Cloudera Navigator policies as part of the processing workflow?
- **What is the replication frequency:** This is important to prioritization and SLA management.
- **How much bandwidth to allow for:** This is also important for the prioritization of different datasets and for overall management and control over the throughput of the pipe between the clusters.
- **Do you allow for deletion when replicating:** Turning delete replication off or using the trash delete replication protects against human error.
- **Can you select from and to cluster:** To avoid confusion, replication is defined as one way.
- **What and how to be notified:** There are a number of notification options included with Cloudera Replication. This allows SBI to keep track of what data is where so, in the case of failure, and have a complete picture of what data is replicated.

Cloudera Replication is also built upon Distcp and, therefore, the following functionality is included with the Cloudera CDP Private Cloud architecture:

- Parallel copying of data
- Byte-by-byte data copies so compressed data doesn't need to be uncompressed to be transferred
- All permissions replicate with the data
- During the copying process, the file is stored at a TMP location so as not to disrupt the workloads working in that table or folder with partly completed data files.

HDFS Replication

HDFS must be configured for high availability (HA). Three components that enable high availability of data on HDFS are Namenode, Journalnode, and Datanode services. Multiple cluster leader nodes run both Namenode and JournalNode services

to maintain the filesystem metadata. Datanode services run on each worker node, maintaining the individual data blocks and providing data availability through block replication.

HDFS utilizes the DistCp tool to replicate data between clusters. CDP utilizes Replication Manager within Cloudera Manager to manage the replication policies for a given directory structure in HDFS. Replication Manager utilizes a version of the DistCp tool specific for Cloudera, to integrate with CDP cluster management. Replication policies are triggered on a periodic basis, providing an asynchronous replication path for data on HDFS. HDFS supports replicating encrypted data safely across clusters. HDFS Transparent Data Encryption must be enabled for encrypting data at rest.

Some use cases where you can use HDFS replication policies include:

- copying data from legacy on-premises systems to Amazon S3 or Microsoft ADLS Gen2 (ABFS) cloud buckets or from cloud buckets to on-premise systems.
- replicating required data to another cluster to run load-intensive workflows on it which optimizes the primary cluster performance.
- deploying a complete backup-restore solution for your enterprise.

Ozone Replication

Replication manager supports Ozone replication policies to replicate data in Ozone buckets between CDP Private Cloud Base clusters

Ozone replication policies support data replication between:

- FSO buckets in source and target clusters using ofs protocol. Supports incremental replication using file checksums.
- legacy buckets in source and target clusters using ofs protocol. Supports incremental replication using file checksums.
- OBS buckets in source and target clusters that support S3A filesystem using the S3A scheme or replication protocol.

SBI can use these policies to replicate or migrate the required Ozone data to another cluster to run load-intensive workloads, back up data, or for backup-restore use cases.

Snapshots

Snapshots are point-in-time backups of HDFS directories or HBase tables without making data copies. You can create HBase and HDFS snapshots using Cloudera Manager or by using the command-line.

- HBase snapshots allow you to create point-in-time backups of tables without making data copies, and with minimal impact on RegioSBIrvers.

- HDFS snapshots allow you to create point-in-time backups of directories or the entire filesystem without actually cloning the data. They can improve data replication performance and prevent errors caused by changes to a source directory. These snapshots appear on the filesystem as read-only directories that can be accessed just like other ordinary directories.

HDFS snapshots allow taking a quick metadata only stock of all files contained in a snapshot-enabled directory. It uses a copy-on-write (CoW) approach where all later modifications to files will cause for all files contained in an active snapshot to be archived behind the scenes. Access to the snapshot is by a special directory notation that exposes the snapshot by name as a subdirectory of the original directory. All files internally either point to the unchanged live files, or archived files for any modified or deleted file. In Hadoop the client only allows to append to existing files, with a single writer process at any point in time. This makes snapshotting more predictable, as files cannot be mutated freely, and changes are only in the form of appends to existing files. When a snapshot is performed, all blocks that are complete are part of the snapshot. In other words, for a file that is being written to still (for example, a web server log file), you will not be able to snapshot the data in the currently uncommitted last block. This amounts to whatever the HDFS block size is set to, usually defaulting to 128MB.

HBase works very similarly, taking a snapshot of a table as a metadata operation, tagging all files belonging to the table that the command is operated on. Should files be deleted, for example due to a compaction, the same mechanism of archiving and reference counting of the underlying files takes place. A new snapshot will not occupy any additional storage, but will accrue more storage space as changes happen and files need to be retained additionally. HBase snapshots do not use HDFS snapshots, but their own internal feature implementation. This means that HBase snapshots cannot be accessed directly through HDFS, but need a special tool that is provided by HBase. Also, HBase snapshots need to trigger a flush operation to persist any pending mutations, as they do not include the write-ahead logs (WALs). This might cause a negative impact on the cluster performance, and needs to be considered carefully.

In more general terms, snapshots allow to freeze some moments in time for the data they operate on. They provide a versioning mechanism, and with the appropriate access tool a file consistent copy can be made. Snapshots are local to the storage system, and do not help with disaster recovery, which means a proper backup architecture would include a copy of the snapshot to another system or cluster. Finally, not all tools have support for built-in snapshots, so this only applies where available.

Backups

Backups are essentially a point-in-time based copy of data, which is then copied to another set of servers for access in case of data loss. As mentioned, this could be erroneous deletion of data, or a server failure. For the former, it would be beneficial for a short-circuit, partial or full restore mechanism, depending on the location of the copy. For true disaster recovery though, the copy should be versioned, retained for a configurable amount of copies, and stored in one or multiple locations to be able to always restore the data, even if the RTO is not great.

Due to the complexity of Hadoop, there is no single solution that can provide a consistent backup across all the subsystems in the Hadoop stack. Implementing backups will span many systems, and use many techniques, including replication and snapshots. Based on the necessary RTO and RPO, the setup can vary a lot, so planning ahead of time is crucial to guaranteeing SLAs later on.

Rack-Awareness and High Availability

Hadoop – and HDFS specifically – has another feature, named rack-awareness, provided to set up cluster topologies that allow for another often asked requirement for production grade systems: high availability. The data that is written to HDFS is automatically replicated three times (by default, but this is settable per file), so that there is always a node that holds the necessary data for a HDFS file, even if you lose 1-2 servers. But with common data center architectures, servers are usually grouped and connected to the same power distribution unit (PDU), connected to a single power circuit.

Further, the servers might be connected to a single top-of-the-rack (ToR) switch, adding to the single point of failure scenario. For the latter, network connections are usually wired in such a way that servers are connected to two separate uplinks. For the former, you could connect redundant power supply units to multiple power circuits. But that might raise the cost per server and power circuits substantially, something to be avoided especially at scale – and therefore good practice even with smaller clusters. HDFS was designed to be fault tolerant, and fail-over safe, leading to the rack-awareness feature, allowing the system to place data block replicas in such a way that they are located on different, independent power circuits and network paths. This

keeps each server simple, while increasing the probability for no loss of access to data in case of partial failures.

It is certainly vital to design a cluster topology with rack-awareness in mind to increase the availability of the data it is providing. It does not, though, replace an end-to-end replication or backup strategy, because it only deals with a subset of possible failure scenarios. More specifically, rack-awareness allows for some failure scenarios to not pose any impact on cluster operations. It does not cover loss of data through user errors, such as erroneous deletion of files in HDFS, dropping managed tables in Hive, or deleting tables in HBase. It also does not handle more widespread outages, such as datacenter failures. Even network partitions/splits will pose an operational hazard, as you may end up with an operational part of the cluster (majority vote succeeds) but not being able to access all the data.

Cloudera Components for High Availability

Cloudera Data Platform Private Cloud, provides High Availability (HA) to most of the components distributed through the product. It is important for SBI to consider what components are critical for the business continuity and configure HA for those components. As generally understood implementation of HA comes with additional hardware/operational requirements

Below is list of components that Cloudera recommends for SBI to be deployed in HA mode

HDFS Namenode

In a standard configuration, the NameNode is a single point of failure (SPOF) in an HDFS cluster. Each cluster has a single NameNode, and if that host or process becomes unavailable, the cluster as a whole is unavailable until the NameNode is either restarted or brought up on a new host. The Secondary NameNode does not provide failover capability.

The standard configuration reduces the total availability of an HDFS cluster in two major ways:

- In the case of an unplanned event such as a host crash, the cluster is unavailable until an operator restarts the NameNode.

- Planned maintenance events such as software or hardware upgrades on the NameNode machine result in periods of cluster downtime.

HDFS NameNode HA avoids this by facilitating either a fast failover to a standby NameNode during machine crash, or a graceful administrator-initiated failover during planned maintenance.

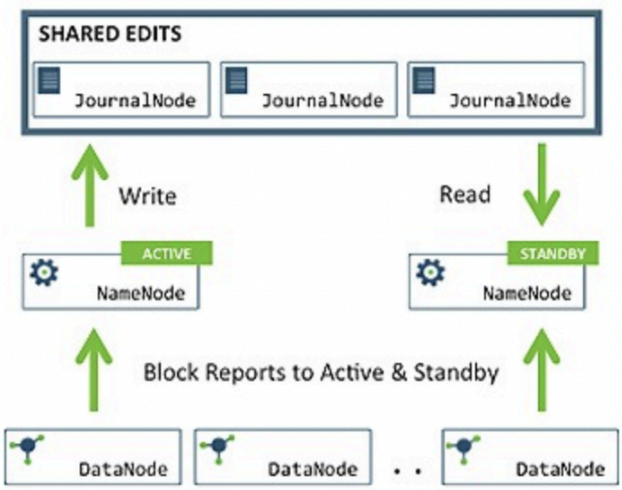
NameNode architecture

In a typical HA cluster, two different machines are configured as NameNodes. In a working cluster, one of the NameNode machine is in the Active state, and the other is in the Standby state.

The Active NameNode is responsible for all client operations in the cluster, while the Standby NameNode acts as a backup. The Standby machine maintains enough state to provide a fast failover (if required).

In order for the Standby node to keep its state synchronized with the Active node, both nodes communicate with a group of separate daemons called JournalNodes (JNs). When the Active node performs any namespace modification, the Active node durably logs a modification record to a majority of these JNs. The Standby node reads the edits from the JNs and continuously watches the JNs for changes to the edit log. Once the Standby Node observes the edits, it applies these edits to its own namespace. When using QJM, JournalNodes act as the shared editlog storage. In a failover event, the Standby ensures that it has read all of the edits from the JounalNodes before promoting itself to the Active state. (This mechanism ensures that the namespace state is fully synchronized before a failover completes.)

To provide a fast failover, it is also necessary that the Standby node has up-to-date information on the location of blocks in your cluster. To get accurate information about the block locations, DataNodes are configured with the location of all the NameNodes, and send block location information and heartbeats to all the NameNode machines.



It is vital for the correct operation of an HA cluster that only one of the NameNodes should be Active at a time. Failure to do so would cause the namespace state to quickly diverge between the NameNode machines, thus causing potential data loss. (This situation is referred to as a split-brain scenario.)

To prevent the split-brain scenario, the JournalNodes allow only one NameNode to be a writer at a time. During failover, the NameNode, that is chosen to become active, takes over the role of writing to the JournalNodes. This process prevents the other NameNode from continuing in the Active state and thus lets the new Active node proceed with the failover safely.

YARN Resource Manager

The ResourceManager high availability (HA) feature adds redundancy in the form of an active-standby ResourceManager pair.

The YARN ResourceManager is responsible for tracking the resources in a cluster and scheduling applications (for example, MapReduce jobs). The ResourceManager high availability (HA) feature adds redundancy in the form of an active-standby ResourceManager pair to remove this single point of failure. Furthermore, upon failover from the active ResourceManager to the standby, the applications can resume from the last state saved to the state store; for example, map tasks in a MapReduce job are not

run again if a failover to a new active ResourceManager occurs after the completion of the map phase. This allows events such the following to be handled without any significant performance effect on running applications:

- Unplanned events such as machine crashes
- Planned maintenance events such as software or hardware upgrades on the machine running the ResourceManager

ResourceManager HA requires ZooKeeper and HDFS services to be running.

ResourceManager HA is implemented by means of an active-standby pair of ResourceManagers. On start-up, each ResourceManager is in the standby state; the process is started, but the state is not loaded. When one of the ResourceManagers is transitioning to the active state, the ResourceManager loads the internal state from the designated state store and starts all the internal services. The stimulus to transition to active comes from either the administrator (through the CLI) or through the integrated failover controller when automatic failover is enabled.

ResourceManager Restart

Restarting the ResourceManager allows for the recovery of in-flight applications if recovery is enabled. To achieve this, the ResourceManager stores its internal state, primarily application-related data and tokens, to the RMStateStore; the cluster resources are re-constructed when the NodeManagers connect. The available alternatives for the state store are MemoryRMStateStore (a memory-based implementation) and ZKRMStateStore (ZooKeeper-based implementation).

Fencing

When running two ResourceManagers, a split-brain situation can arise where both ResourceManagers assume they are active. To avoid this, only a single ResourceManager should be able to perform active operations and the other ResourceManager should be "fenced". The ZooKeeper-based state store (ZKRMStateStore) allows only a single ResourceManager to make changes to the stored state, implicitly fencing the other ResourceManager. This is accomplished by the ResourceManager claiming exclusive create-delete permissions on the root znode. The ACLs on the root znode are automatically created based on the ACLs configured for the store; in case of secure clusters, Cloudera recommends that you set ACLs for the root host such that both ResourceManagers share read-write-admin access, but have exclusive create-delete access. The fencing is implicit and does not require explicit configuration (as fencing in HDFS does). You can plug in a custom "Fencer" if you choose to – for example, to use a different implementation of the state store.

Configuration and FailoverProxy

In an HA setting, SBI should configure two ResourceManagers to use different ports (for example, ports on different hosts). To facilitate this, YARN uses the notion of an ResourceManager Identifier (rm-id). Each ResourceManager has a unique rm-id, and all the RPC configurations (<rpc-address>; for example yarn.resourcemanager.address) for that ResourceManager can be configured via <rpc-address>.<rm-id>. Clients, ApplicationMasters, and NodeManagers use these RPC addresses to talk to the active ResourceManager automatically, even after a failover. To achieve this, they cycle through the list of ResourceManagers in the configuration. This is done automatically and does not require any configuration (as it does in HDFS).

Automatic Failover

By default, ResourceManager HA uses ZKFC (ZooKeeper-based failover controller) for automatic failover in case the active ResourceManager is unreachable or goes down. Internally, the StandbyElector is used to elect the active ResourceManager. The failover controller runs as part of the ResourceManager.

SBI can plug in a custom failover controller if you prefer.

Manual Transitions and Failover

SBI can use the command-line tool ***yarn rmadmin*** to transition a particular ResourceManager to active or standby state, to fail over from one ResourceManager to the other, to get the HA state of an ResourceManager, and to monitor an ResourceManager's health.

Hive MetaStore

You can enable Hive metastore high availability (HA) so that your cluster is resilient to failures if a metastore becomes unavailable. When HA mode is enabled, one of the metastores is designated as the master and the others are slaves. If a master metastore fails, one of the slave metastores takes over.

Prerequisites

- Cloudera recommends that each instance of the metastore runs on a separate cluster host, to maximize high availability.
- Hive metastore HA requires a database that is also highly available, such as MySQL or Oracle DB with replication in active-active mode. Refer to the documentation for your database of choice to configure it correctly.

Multiple HMS instances run in active/active mode. No load balancing occurs. An HMS client always reaches the first instance unless it is down. In this case, the client scans the hive.metastore.uris property that lists the HMS instances for a replacement HMS. The second HMS is the designated replacement if hive.metastore.uri.selection is set to SEQUENTIAL (recommended and the default); otherwise, the replacement is selected randomly from the list if hive.metastore.uri.selection is set to RANDOM.

HiveServer2

To enable high availability for multiple HiveServer2 hosts, configure a load balancer to manage them. To increase stability and security, configure the load balancer on a proxy server.

For clusters with multiple users and availability requirements, you can configure a proxy server to relay requests to and from each HiveServer2 host. Applications connect to a single well-known host and port, and connection requests to the proxy succeed even when hosts running HiveServer2 become unavailable

Impala

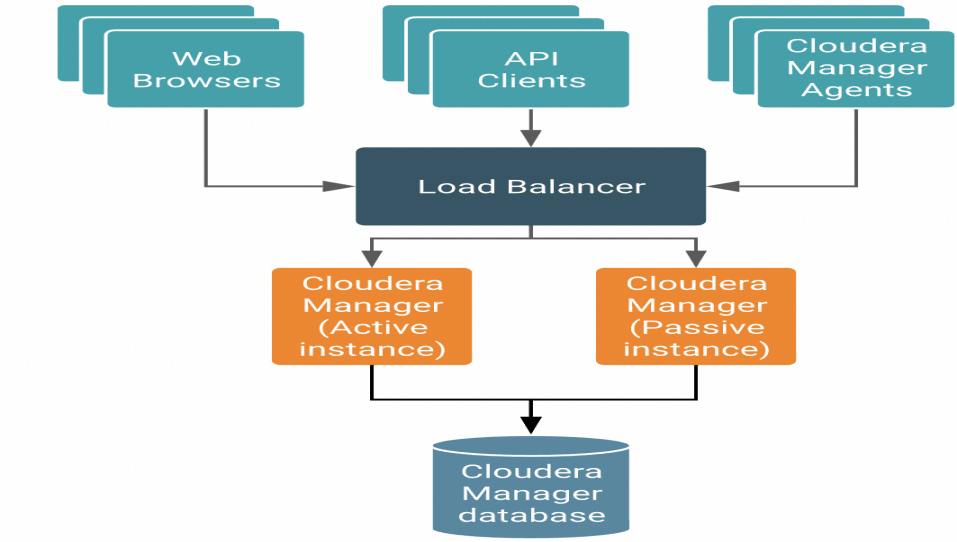
Each Impala daemon has the ability to act as a coordinator, but when used in external systems like BI tools etc, the host name provided will always act as the coordinator and any failure to that impala daemon will lead to service disruption. To handle such failure scenarios it is expected to set up a load-balancing proxy server to relay requests to and from Impala.

When using a load balancer for Impala, applications connect to a single well-known host and port, rather than keeping track of the hosts where a specific Impala daemon is running. The load balancer also lets the Impala nodes share resources to balance out the work loads.

Cloudera Manager

Cloudera Manager is the single common entry into cluster maintenance and management. It is the most used application with anything and everything SBI wants to perform across cluster components is majorly done via Cloudera Manager. Cloudera Manager provides visibility into Cluster resources, utilization and health.

SBI can configure Cloudera Manager for high availability. This configuration provides failover capability in case an instance of Cloudera Manager fails. We can configure Cloudera Manager into an highly available Active-Passive configuration. The Active-Passive configuration supports only 2 hosts running Cloudera Manager along with an external load balancer. Load balancer configuration defines the active vs passive hosts and active host serves all requests during normal operation. The load balancer monitors both active and passive hosts and automatically diverts requests to the passive host if the active host is unresponsive for a given (configured in the load balancer) period of time. If the active host becomes available again, the load balancer automatically redirects the requests back to the active host. Cloudera Manager also internally uses a priority parameter to decide which host will take the active role to process commands within the system and which host will take over the passive role when both hosts boot up and become functional



Platform Component's Benchmarking

Technology Component	Details
Apache Ozone	https://blog.cloudera.com/apache-ozone-a-high-performance-object-store-for-cdp-private-cloud/
Apache Ozone	https://blog.cloudera.com/benchmarking-ozone-clouderas-next-generation-storage-for-cdp/
Apache Impala	https://blog.cloudera.com/apache-impala-leads-traditional-analytic-database/
Apache Nifi	https://blog.cloudera.com/benchmarking-nifi-performance-and-scalability/
Apache Kudu	https://blog.cloudera.com/benchmarking-time-series-workloads-on-apache-kudu-using-tsbs/
Apache Kafka	https://engineering.linkedin.com/kafka/benchmarking-apache-kafka-2-million-writes-second-three-cheap-machines