



Cloudera Introduction

Important Notice

© 2010-2017 Cloudera, Inc. All rights reserved.

Cloudera, the Cloudera logo, and any other product or service names or slogans contained in this document are trademarks of Cloudera and its suppliers or licensors, and may not be copied, imitated or used, in whole or in part, without the prior written permission of Cloudera or the applicable trademark holder.

Hadoop and the Hadoop elephant logo are trademarks of the Apache Software Foundation. All other trademarks, registered trademarks, product names and company names or logos mentioned in this document are the property of their respective owners. Reference to any products, services, processes or other information, by trade name, trademark, manufacturer, supplier or otherwise does not constitute or imply endorsement, sponsorship or recommendation thereof by us.

Complying with all applicable copyright laws is the responsibility of the user. Without limiting the rights under copyright, no part of this document may be reproduced, stored in or introduced into a retrieval system, or transmitted in any form or by any means (electronic, mechanical, photocopying, recording, or otherwise), or for any purpose, without the express written permission of Cloudera.

Cloudera may have patents, patent applications, trademarks, copyrights, or other intellectual property rights covering subject matter in this document. Except as expressly provided in any written license agreement from Cloudera, the furnishing of this document does not give you any license to these patents, trademarks copyrights, or other intellectual property. For information about patents covering Cloudera products, see <http://tiny.cloudera.com/patents>.

The information in this document is subject to change without notice. Cloudera shall not be liable for any damages resulting from technical errors or omissions which may be present in this document, or from use of this document.

Cloudera, Inc.

1001 Page Mill Road, Bldg 3

Palo Alto, CA 94304

info@cloudera.com

US: 1-888-789-1488

Intl: 1-650-362-0488

www.cloudera.com

Release Information

Version: Cloudera Enterprise 5.13.x

Date: October 12, 2017

Table of Contents

About Cloudera Introduction.....6

Documentation Overview.....	6
-----------------------------	---

CDH Overview.....8

Apache Hive Overview in CDH.....	8
<i>Use Cases for Hive.....</i>	<i>9</i>
<i>Hive Components.....</i>	<i>9</i>
<i>How Hive Works with Other Components.....</i>	<i>10</i>
Apache Impala (incubating) Overview.....	10
<i>Impala Benefits.....</i>	<i>10</i>
<i>How Impala Works with</i>	<i>11</i>
<i>Primary Impala Features.....</i>	<i>11</i>
Cloudera Search Overview.....	12
<i>How Cloudera Search Works.....</i>	<i>13</i>
<i>Understanding Cloudera Search.....</i>	<i>14</i>
<i>Cloudera Search and Other Cloudera Components.....</i>	<i>14</i>
<i>Cloudera Search Architecture.....</i>	<i>16</i>
<i>Cloudera Search Tasks and Processes.....</i>	<i>18</i>
Apache Kudu Overview.....	20
<i>Kudu-Impala Integration.....</i>	<i>21</i>
<i>Example Use Cases.....</i>	<i>21</i>
<i>Related Information.....</i>	<i>22</i>
Apache Sentry Overview.....	22
Apache Spark Overview.....	22
File Formats and Compression.....	23
<i>Using Apache Parquet Data Files with CDH.....</i>	<i>23</i>
<i>Using Apache Avro Data Files with CDH.....</i>	<i>33</i>
<i>Data Compression.....</i>	<i>37</i>
External Documentation.....	39

Cloudera Manager 5 Overview.....40

Terminology.....	40
Architecture.....	43
State Management.....	44
Configuration Management.....	45
Process Management.....	48

Software Distribution Management.....	48
Host Management.....	49
Resource Management.....	50
User Management.....	51
Security Management.....	51
Cloudera Management Service.....	52
Cloudera Manager Admin Console.....	53
<i>Starting and Logging into the Admin Console.....</i>	<i>55</i>
<i>Cloudera Manager Admin Console Home Page.....</i>	<i>55</i>
<i>Displaying Cloudera Manager Documentation.....</i>	<i>58</i>
<i>Automatic Logout.....</i>	<i>59</i>
Cloudera Manager API.....	59
<i>Backing Up and Restoring the Cloudera Manager Configuration</i>	<i>61</i>
<i>Using the Cloudera Manager API for Cluster Automation.....</i>	<i>62</i>
Extending Cloudera Manager.....	64

Cloudera Navigator Data Management Overview.....65

Data Management Challenges.....	65
Cloudera Navigator Data Management Capabilities.....	65
<i>Analytics.....</i>	<i>66</i>
<i>Auditing.....</i>	<i>66</i>
<i>Metadata.....</i>	<i>66</i>
<i>Policies.....</i>	<i>66</i>
<i>Search.....</i>	<i>66</i>
Getting Started with Cloudera Navigator.....	66
<i>Cloudera Navigator Console.....</i>	<i>67</i>
<i>Cloudera Navigator APIs.....</i>	<i>69</i>
<i>Cloudera Navigator Data Management Documentation Library.....</i>	<i>71</i>
Cloudera Navigator Frequently Asked Questions.....	72
<i>Is Cloudera Navigator a module of Cloudera Manager?.....</i>	<i>72</i>
<i>Does Cloudera Navigator have a user interface?.....</i>	<i>72</i>

Cloudera Navigator Data Encryption Overview.....75

Cloudera Navigator Encryption Architecture.....	77
Cloudera Navigator Encryption Integration with an EDH.....	77
Cloudera Navigator Key Trustee Server Overview.....	78
<i>Key Trustee Server Architecture.....</i>	<i>78</i>
Cloudera Navigator Key HSM Overview.....	79
<i>Key HSM Architecture.....</i>	<i>80</i>
Cloudera Navigator Encrypt Overview.....	80
<i>Process-Based Access Control List.....</i>	<i>81</i>

<i>Encryption Key Storage and Management.....</i>	<i>83</i>
Cloudera Navigator Optimizer.....	84
Frequently Asked Questions About Cloudera Software.....	85
Getting Support.....	86
Cloudera Support.....	86
<i>Information Required for Logging a Support Case.....</i>	<i>86</i>
Community Support.....	86
Get Announcements about New Releases.....	87
Report Issues.....	87

About Cloudera Introduction

Cloudera provides a scalable, flexible, integrated platform that makes it easy to manage rapidly increasing volumes and varieties of data in your enterprise. Cloudera products and solutions enable you to deploy and manage Apache Hadoop and related projects, manipulate and analyze your data, and keep that data secure and protected.

Cloudera provides the following products and tools:

- [CDH](#)—The Cloudera distribution of Apache Hadoop and other related open-source projects, including Apache Impala (incubating) and Cloudera Search. CDH also provides security and integration with numerous hardware and software solutions.
- [Apache Impala \(incubating\)](#)—A massively parallel processing SQL engine for interactive analytics and business intelligence. Its highly optimized architecture makes it ideally suited for traditional BI-style queries with joins, aggregations, and subqueries. It can query Hadoop data files from a variety of sources, including those produced by MapReduce jobs or loaded into Hive tables. The YARN resource management component lets Impala coexist on clusters running batch workloads concurrently with Impala SQL queries. You can manage Impala alongside other Hadoop components through the Cloudera Manager user interface, and secure its data through the Sentry authorization framework.
- [Cloudera Search](#)—Provides near real-time access to data stored in or ingested into Hadoop and HBase. Search provides near real-time indexing, batch indexing, full-text exploration and navigated drill-down, as well as a simple, full-text interface that requires no SQL or programming skills. Fully integrated in the data-processing platform, Search uses the flexible, scalable, and robust storage system included with CDH. This eliminates the need to move large data sets across infrastructures to perform business tasks.
- [Cloudera Manager](#)—A sophisticated application used to deploy, manage, monitor, and diagnose issues with your CDH deployments. Cloudera Manager provides the Admin Console, a web-based user interface that makes administration of your enterprise data simple and straightforward. It also includes the Cloudera Manager API, which you can use to obtain cluster health information and metrics, as well as configure Cloudera Manager.
- [Cloudera Navigator](#)—End-to-end data management and security for the CDH platform. Cloudera Navigator Data Management enables administrators, data managers, and analysts explore vast data collections in Hadoop. Cloudera Navigator Encrypt and simplifies the storage and management of encryption keys. The robust auditing, data management, lineage management, lifecycle management, and encryption key management in Cloudera Navigator allow enterprises to adhere to stringent compliance and regulatory requirements.

This introductory guide provides a general overview of CDH, Cloudera Manager, and Cloudera Navigator. This guide also includes frequently asked questions about Cloudera products and describes how to get support, report issues, and receive information about updates and new releases.

Documentation Overview

The following guides are included in the Cloudera documentation set:

Guide	Description
Overview of Cloudera and the Cloudera Documentation Set	Cloudera provides a scalable, flexible, integrated platform that makes it easy to manage rapidly increasing volumes and varieties of data in your enterprise. Cloudera products and solutions enable you to deploy and manage Apache Hadoop and related projects, manipulate and analyze your data, and keep that data secure and protected.
Cloudera Release Notes	This guide contains release and download information for installers and administrators. It includes release notes as well as information about versions and downloads. The guide also provides a release matrix that shows which major and minor release version of a product is supported with which release version of Cloudera Manager and CDH.

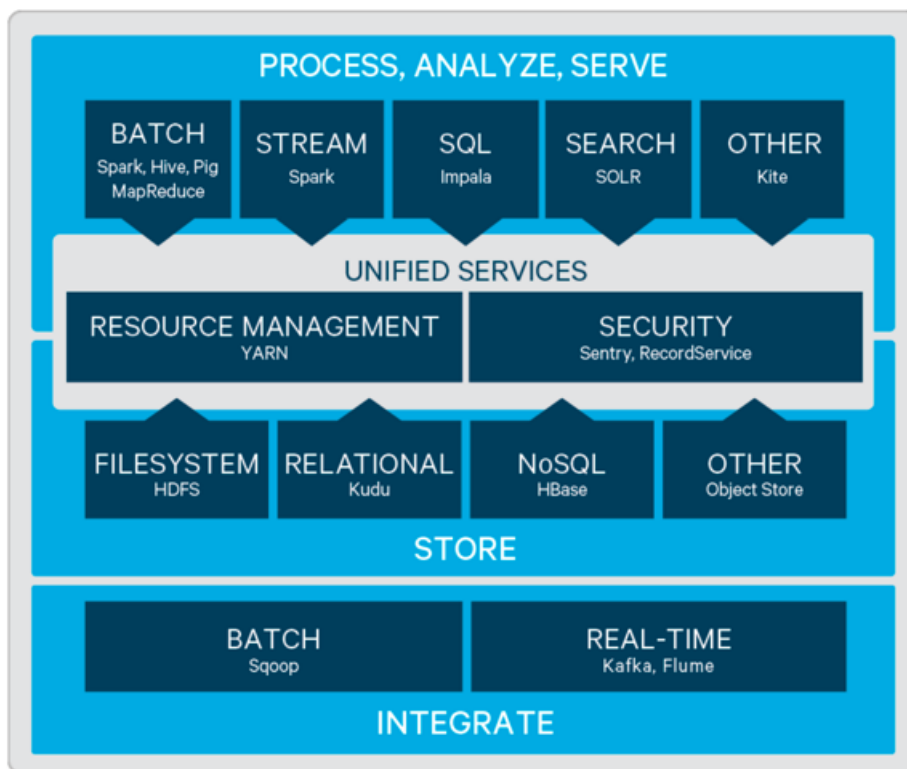
Guide	Description
Cloudera QuickStart	This set of guides describes ways to rapidly begin experimenting with Cloudera software. The first section describes how to download and use QuickStart virtual machines, which provide everything you need to try CDH, Cloudera Manager, Impala, and Cloudera Search. Subsequent sections show you how to create a new installation of Cloudera Manager 5, CDH5, and managed services on a cluster of four hosts and an unmanaged CDH pseudo cluster. Quick start installations are for demonstration and POC applications only and are not recommended for production use.
Cloudera Installation	This guide provides instructions for installing Cloudera software.
Cloudera Upgrade Overview	This topic provides an overview of upgrade procedures for Cloudera Manager and CDH.
Cloudera Administration	This guide describes how to configure and administer a Cloudera deployment. Administrators manage resources, availability, and backup and recovery configurations. In addition, this guide shows how to implement high availability, and discusses integration.
Cloudera Navigator Data Management	This guide shows you how to use Cloudera Navigator Data Management component for comprehensive data governance, compliance, data stewardship, and other data management tasks.
Cloudera Operation	This guide shows how to monitor the health of a Cloudera deployment and diagnose issues. You can obtain metrics and usage information and view processing activities. This guide also describes how to examine logs and reports to troubleshoot issues with cluster configuration and operation as well as monitor compliance.
Cloudera Security	This guide is intended for system administrators who want to secure a cluster using data encryption, user authentication, and authorization techniques. It provides conceptual overviews and how-to information about setting up various Hadoop components for optimal security, including how to setup a gateway to restrict access. This guide assumes that you have basic knowledge of Linux and systems administration practices, in general.
Apache Impala (incubating) - Interactive SQL	This guide describes Impala, its features and benefits, and how it works with CDH. This topic introduces Impala concepts, describes how to plan your Impala deployment, and provides tutorials for first-time users as well as more advanced tutorials that describe scenarios and specialized features. You will also find a language reference, performance tuning, instructions for using the Impala shell, troubleshooting information, and frequently asked questions.
Cloudera Search Guide	This guide explains how to configure and use Cloudera Search. This includes topics such as extracting, transforming, and loading data, establishing high availability, and troubleshooting.
Spark Guide	This guide describes Apache Spark, a general framework for distributed computing that offers high performance for both batch and interactive processing. The guide provides tutorial Spark applications, how to develop and run Spark applications, and how to use Spark with other Hadoop components.
Cloudera Glossary	This guide contains a glossary of terms for Cloudera components.

CDH Overview

CDH is the most complete, tested, and popular distribution of Apache Hadoop and related projects. CDH delivers the core elements of Hadoop – scalable storage and distributed computing – along with a Web-based user interface and vital enterprise capabilities. CDH is Apache-licensed open source and is the only Hadoop solution to offer unified batch processing, interactive SQL and interactive search, and role-based access controls.

CDH provides:

- **Flexibility**—Store any type of data and manipulate it with a variety of different computation frameworks including batch processing, interactive SQL, free text search, machine learning and statistical computation.
- **Integration**—Get up and running quickly on a complete Hadoop platform that works with a broad range of hardware and software solutions.
- **Security**—Process and control sensitive data.
- **Scalability**—Enable a broad range of applications and scale and extend them to suit your requirements.
- **High availability**—Perform mission-critical business tasks with confidence.
- **Compatibility**—Leverage your existing IT infrastructure and investment.



For information about CDH components, which is out of scope for Cloudera documentation, see the links in [External Documentation](#) on page 39.

Apache Hive Overview in CDH

Hive data warehouse software enables reading, writing, and managing large datasets in distributed storage. Using the Hive query language (HiveQL), which is very similar to SQL, queries are converted into a series of jobs that execute on a Hadoop cluster through MapReduce or Apache Spark.

Users can run batch processing workloads with Hive while also analyzing the same data for interactive SQL or machine-learning workloads using tools like Apache Impala (incubating) or Apache Spark—all within a single platform.

As part of CDH, Hive also benefits from:

- Unified resource management provided by YARN
- Simplified deployment and administration provided by Cloudera Manager
- Shared security and governance to meet compliance requirements provided by Apache Sentry and Cloudera Navigator

Use Cases for Hive

Because Hive is a petabyte-scale data warehouse system built on the Hadoop platform, it is a good choice for environments experiencing phenomenal growth in data volume. The underlying MapReduce interface with HDFS is hard to program directly, but Hive provides an SQL interface, making it possible to use existing programming skills to perform data preparation.

Hive on MapReduce or Spark is best-suited for batch data preparation or ETL:

- You must run scheduled batch jobs with very large ETL sorts with joins to prepare data for Hadoop. Most data served to BI users in Impala is prepared by ETL developers using Hive.
- You run data transfer or conversion jobs that take many hours. With Hive, if a problem occurs partway through such a job, it recovers and continues.
- You receive or provide data in diverse formats, where the Hive SerDes and variety of UDFs make it convenient to ingest and convert the data. Typically, the final stage of the ETL process with Hive might be to a high-performance, widely supported format such as Parquet.

Hive Components

Hive consists of the following components:

The Metastore Database

The metastore database is an important aspect of the Hive infrastructure. It is a separate database, relying on a traditional RDBMS such as MySQL or PostgreSQL, that holds metadata about Hive databases, tables, columns, partitions, and Hadoop-specific information such as the underlying data files and HDFS block locations.

The metastore database is shared by other components. For example, the same tables can be inserted into, queried, altered, and so on by both Hive and Impala. Although you might see references to the “Hive metastore”, be aware that the metastore database is used broadly across the Hadoop ecosystem, even in cases where you are not using Hive itself.

The metastore database is relatively compact, with fast-changing data. Backup, replication, and other kinds of management operations affect this database. See [Configuring the Hive Metastore for CDH](#) for details about configuring the Hive metastore.

Cloudera recommends that you deploy the Hive metastore, which stores the metadata for Hive tables and partitions, in “remote mode.” In this mode the metastore service runs in its own JVM process and other services, such as HiveServer2, HCatalog, and Apache Impala (incubating) communicate with the metastore using the Thrift network API.

See [Starting the Hive Metastore in CDH](#) for details about starting the Hive metastore service.

HiveServer2

HiveServer2 is a server interface that enables remote clients to submit queries to Hive and retrieve the results. It replaces HiveServer1, which has been deprecated and will be removed in a future release of CDH. HiveServer2 supports multi-client concurrency, capacity planning controls, Sentry authorization, Kerberos authentication, LDAP, SSL, and provides better support for JDBC and ODBC clients.

HiveServer2 is a container for the Hive execution engine. For each client connection, it creates a new execution context that serves Hive SQL requests from the client. It supports JDBC clients, such as the Beeline CLI, and ODBC clients. Clients connect to HiveServer2 through the Thrift API-based Hive service.

See [Configuring HiveServer2 for CDH](#) for details on configuring HiveServer2 and see [Starting, Stopping, and Using HiveServer2 in CDH](#) for details on starting/stopping the HiveServer2 service and information about using the Beeline CLI to connect to HiveServer2. For details about managing HiveServer2 with its native web user interface (UI), see [Using HiveServer2 Web UI in CDH](#).

How Hive Works with Other Components

Hive integrates with other components, which serve as query execution engines or as data stores:

Hive on Spark

Hive traditionally uses MapReduce behind the scenes to parallelize the work, and perform the low-level steps of processing a SQL statement such as sorting and filtering. Hive can also use Spark as the underlying computation and parallelization engine. See [Running Apache Hive on Spark in CDH](#) for details about configuring Hive to use Spark as its execution engine and see [Tuning Apache Hive on Spark in CDH](#) for details about tuning Hive on Spark.

Hive and HBase

Apache HBase is a NoSQL database that supports real-time read/write access to large datasets in HDFS. See [Using Apache Hive with HBase in CDH](#) for details about configuring Hive to use HBase. For information about running Hive queries on a secure HBase server, see [Using Hive to Run Queries on a Secure HBase Server](#).

Hive on Amazon S3

Use the Amazon S3 filesystem to efficiently manage transient Hive ETL (extract-transform-load) jobs. For step-by-step instructions to configure Hive to use S3 and multiple scripting examples, see [Configuring Transient Hive ETL Jobs to Use the Amazon S3 Filesystem](#). To optimize how Hive writes data to and reads data from S3-backed tables and partitions, see [Tuning Hive Performance on the Amazon S3 Filesystem](#). For information about setting up a shared Amazon Relational Database Service (RDS) as your Hive metastore, see [How To Set Up a Shared Amazon RDS as Your Hive Metastore for CDH](#).

Hive on Microsoft Azure Data Lake Store

In CDH 5.12 and higher, both Hive on MapReduce2 and Hive-on-Spark can access tables on Microsoft Azure Data Lake store (ADLS). In contrast to Amazon S3, ADLS more closely resembles native HDFS behavior, providing consistency, file directory structure, and POSIX-compliant ACLs. See [Configuring ADLS Connectivity](#) for information about configuring and using ADLS with Hive on MapReduce2.

Apache Impala (incubating) Overview

Impala provides fast, interactive SQL queries directly on your Apache Hadoop data stored in HDFS, HBase, or the Amazon Simple Storage Service (S3). In addition to using the same unified storage platform, Impala also uses the same metadata, SQL syntax (Hive SQL), ODBC driver, and user interface (Impala query UI in Hue) as Apache Hive. This provides a familiar and unified platform for real-time or batch-oriented queries.

Impala is an addition to tools available for querying big data. Impala does not replace the batch processing frameworks built on MapReduce such as Hive. Hive and other frameworks built on MapReduce are best suited for long running batch jobs, such as those involving batch processing of Extract, Transform, and Load (ETL) type jobs.



Note: Impala was accepted into the Apache incubator on December 2, 2015. In places where the documentation formerly referred to “Cloudera Impala”, now the official name is “Apache Impala (incubating)”.

Impala Benefits

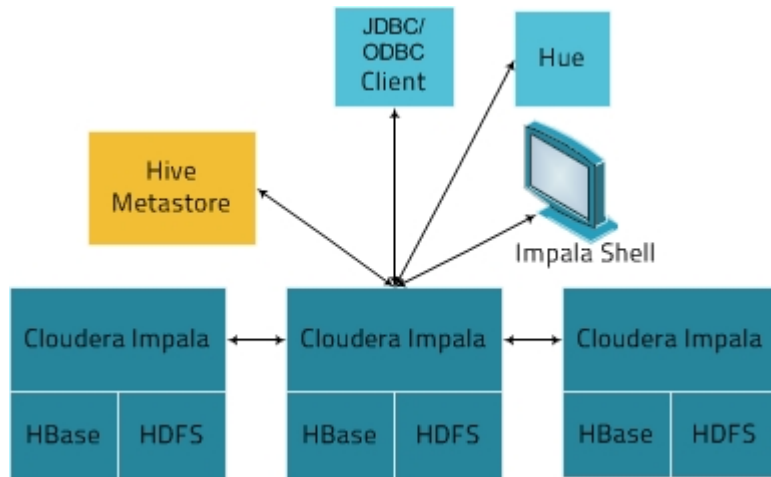
Impala provides:

- Familiar SQL interface that data scientists and analysts already know.
- Ability to query high volumes of data (“big data”) in Apache Hadoop.

- Distributed queries in a cluster environment, for convenient scaling and to make use of cost-effective commodity hardware.
- Ability to share data files between different components with no copy or export/import step; for example, to write with Pig, transform with Hive and query with Impala. Impala can read from and write to Hive tables, enabling simple data interchange using Impala for analytics on Hive-produced data.
- Single system for big data processing and analytics, so customers can avoid costly modeling and ETL just for analytics.

How Impala Works with

The following graphic illustrates how Impala is positioned in the broader Cloudera environment:



The Impala solution is composed of the following components:

- Clients - Entities including Hue, ODBC clients, JDBC clients, 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.
- 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 introduced in Impala 1.2.
- Impala - This process, which runs on DataNodes, coordinates and executes queries. Each instance of Impala can receive, plan, and coordinate queries from Impala clients. Queries are distributed among Impala nodes, and these nodes then act as workers, executing parallel query fragments.
- HBase and HDFS - Storage for data to be queried.

Queries executed using Impala are handled as follows:

1. 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.
2. Impala parses 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.
3. Services such as HDFS and HBase are accessed by local `impalad` instances to provide data.
4. Each `impalad` returns data to the coordinating `impalad`, which sends these results to the client.

Primary Impala Features

Impala provides support for:

- Most common SQL-92 features of Hive Query Language (HiveQL) including [SELECT](#), [joins](#), and [aggregate functions](#).
- HDFS, HBase, and Amazon Simple Storage System (S3) storage, including:
 - [HDFS file formats](#): delimited text files, Parquet, Avro, SequenceFile, and RCFile.

- Compression codecs: Snappy, GZIP, Deflate, BZIP.
- Common data access interfaces including:
 - [JDBC driver](#).
 - [ODBC driver](#).
 - Hue Beeswax and the Impala Query UI.
- [impala-shell command-line interface](#).
- [Kerberos authentication](#).

Cloudera Search Overview

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.

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 CDH. 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 incorporates [Apache Solr](#), which includes Apache Lucene, SolrCloud, Apache Tika, and Solr Cell. Cloudera Search is included with CDH 5 and higher.

Using Cloudera Search with the CDH 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 Cloudera Search features.

Table 1: Cloudera Search Features

Feature	Description
Unified management and monitoring with Cloudera Manager	Cloudera Manager provides unified and centralized management and monitoring for CDH and Cloudera Search. Cloudera Manager simplifies deployment, configuration, and monitoring of your search services. Many existing search solutions lack management and monitoring capabilities and fail to provide deep insight into utilization, system health, trending, and other supportability aspects.
Index storage in HDFS	<p>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.</p> <p>Cloudera Search is optimized for fast read and write of indexes in HDFS while indexes are served and queried through standard Solr mechanisms. Because</p>

Feature	Description
	data and indexes are co-located, data processing does not require transport or separately managed storage.
Batch index creation through MapReduce	To facilitate index creation for large data sets, Cloudera Search has built-in MapReduce jobs for indexing data stored in HDFS or HBase. As a result, the linear scalability of MapReduce is applied to the indexing pipeline, off-loading Solr index serving resources.
Real-time and scalable indexing at data ingest	Cloudera Search provides integration with Flume to support near real-time indexing. As new events pass through a Flume hierarchy and are written to HDFS, those events can be written directly to Cloudera Search indexers. In addition, Flume supports routing events, filtering, and annotation of data passed to CDH. These features work with Cloudera Search for improved index sharding, index separation, and document-level access control.
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 simplifies data preparation for indexing over a variety of file formats. Users can easily implement Morphlines for Flume, Kafka, and HBase, or re-use the same Morphlines for other applications, such as MapReduce or Spark jobs.
HBase search	Cloudera Search integrates with HBase, enabling full-text search of HBase data without affecting HBase performance or duplicating data storage. A listener monitors the replication event stream from HBase RegionServers and captures each write or update-replicated event, enabling extraction and mapping (for example, using Morphlines). The event is then sent directly to Solr for indexing and storage in HDFS, using the same process as for other indexing workloads of Cloudera Search. The indexes can be served immediately, enabling free-text searching of HBase data.

How Cloudera Search Works

In near real-time indexing use cases, such as log or event stream analytics, Cloudera Search indexes events that are streamed through Apache Flume, Apache Kafka, Spark Streaming, or HBase. Fields and events are mapped to standard Solr indexable schemas. Lucene indexes the incoming events and the index is written and stored in standard Lucene index files in HDFS. Regular Flume event routing and storage of data in partitions in HDFS can also be applied. Events can be routed and streamed through multiple Flume agents and written to separate Lucene indexers that can write into separate index shards, for better scale when indexing and quicker responses when searching.

The indexes are loaded from HDFS to Solr cores, exactly like Solr would have read from local disk. The difference in the design of Cloudera Search is the robust, distributed, and scalable storage layer of HDFS, which helps eliminate costly downtime and allows for flexibility across workloads without having to move data. Search queries can then be submitted to Solr through either the standard Solr API, or through a simple search GUI application, included in Cloudera Search, which can be deployed in Hue.

Cloudera Search batch-oriented indexing capabilities can address needs for searching across batch uploaded files or large data sets that are less frequently updated and less in need of near-real-time indexing. It can also be conveniently used for re-indexing (a common pain point in stand-alone Solr) or ad-hoc indexing for on-demand data exploration. Often, batch indexing is done on a regular basis (hourly, daily, weekly, and so on) as part of a larger workflow.

For such cases, Cloudera Search includes a highly scalable indexing workflow based on MapReduce or Spark. A MapReduce or Spark workflow is launched for specified files or folders in HDFS, or tables in HBase, and the field extraction and Solr schema mapping occurs during the mapping phase. Reducers use embedded Lucene to write the data as a single index or as index shards, depending on your configuration and preferences. After the indexes are stored in HDFS, they can be queried by using standard Solr mechanisms, as previously described above for the near-real-time indexing use case. You can also configure these batch indexing options to post newly indexed data directly into live, active indexes, served by Solr. This *GoLive* option enables a streamlined data pipeline without interrupting service to process regularly incoming batch updates.

The Lily HBase Indexer Service is a flexible, scalable, fault tolerant, transactional, near real-time oriented system for processing a continuous stream of HBase cell updates into live search indexes. The Lily HBase Indexer uses Solr to index data stored in HBase. As HBase applies inserts, updates, and deletes to HBase table cells, the indexer keeps Solr consistent with the HBase table contents, using standard HBase replication features. The indexer supports flexible custom application-specific rules to extract, transform, and load HBase data into Solr. Solr search results can contain `columnFamily:qualifier` links back to the data stored in HBase. This way applications can use the Search result set to directly access matching raw HBase cells. Indexing and searching do not affect operational stability or write throughput of HBase because the indexing and searching processes are separate and asynchronous to HBase.

Understanding Cloudera Search

Cloudera Search fits into the broader set of solutions available for analyzing information in large data sets. CDH provides both the means and the tools to store the data and run queries. You can explore data through:

- MapReduce or Spark jobs
- Cloudera Impala queries
- Cloudera Search queries

CDH provides storage for and access to large data sets using MapReduce jobs, but creating these jobs requires technical knowledge, and each job can take minutes or more to run. The longer run times associated with MapReduce jobs can interrupt the process of exploring data.

To provide more immediate queries and responses and to eliminate the need to write MapReduce applications, you can use Apache Impala (incubating). Impala returns results in seconds instead of minutes.

Although Impala is a fast, powerful application, it uses SQL-based querying syntax. Using Impala can be challenging for users who are not familiar with SQL. If you do not know SQL, you can use Cloudera Search. Although Impala, Apache Hive, and Apache Pig all require a structure that is applied at query time, Search supports free-text search on any data or fields you have indexed.

How Search Uses Existing Infrastructure

Any data already in a CDH deployment can be indexed and made available for query by Cloudera Search. For data that is not stored in CDH, Cloudera Search provides tools for loading data into the existing infrastructure, and for indexing data as it is moved to HDFS or written to Apache HBase.

By leveraging existing infrastructure, Cloudera Search eliminates the need to create new, redundant structures. In addition, Cloudera Search uses services provided by CDH and Cloudera Manager in a way that does not interfere with other tasks running in the same environment. This way, you can reuse existing infrastructure without the cost and problems associated with running multiple services in the same set of systems.

Cloudera Search and Other Cloudera Components

Cloudera Search interacts with other Cloudera components to solve different problems. The following table lists Cloudera components that contribute to the Search process and describes how they interact with Cloudera Search:

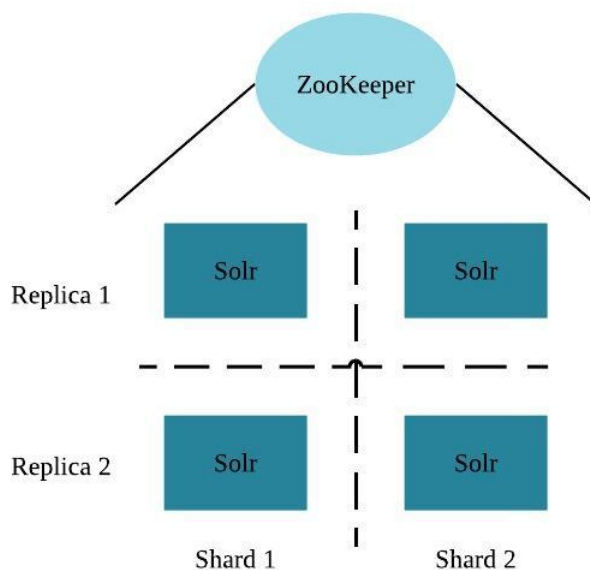
Component	Contribution	Applicable To
HDFS	Stores source documents. Search indexes source documents to make them searchable. Files that support Cloudera Search, such as Lucene index files and write-ahead logs, are also stored in HDFS. Using HDFS provides simpler provisioning on a larger base, redundancy, and fault tolerance. With HDFS, Cloudera Search servers are essentially stateless, so host failures have minimal consequences. HDFS also provides snapshotting, inter-cluster replication, and disaster recovery.	All cases
MapReduce	Search includes a pre-built MapReduce-based job. This job can be used for on-demand or scheduled indexing of any supported data set stored in HDFS. This job uses cluster resources for scalable batch indexing.	Many cases
Flume	Search includes a Flume sink that enables writing events directly to indexers deployed on the cluster, allowing data indexing during ingestion.	Many cases
Hue	Hue includes a GUI-based Search application that uses standard Solr APIs and can interact with data indexed in HDFS. The application provides support for the Solr standard query language and visualization of faceted search functionality.	Many cases
Morphlines	A morphline is a rich configuration file that defines an ETL transformation chain. Morphlines can consume any kind of data from any data source, process the data, and load the results into Cloudera Search. Morphlines run in a small, embeddable Java runtime system, and can be used for near real-time applications such as the flume agent as well as batch processing applications such as a Spark job.	Many cases
ZooKeeper	Coordinates distribution of data and metadata, also known as shards. It provides automatic failover to increase service resiliency.	Many cases
Spark	The CrunchIndexerTool can use Spark to move data from HDFS files into Apache Solr, and run the data through a morphline for extraction and transformation.	Some cases
HBase	Supports indexing of stored data, extracting columns, column families, and key information as fields. Although HBase does not use secondary indexing, Cloudera Search can facilitate full-text searches of content in rows and tables in HBase.	Some cases
Cloudera Manager	Deploys, configures, manages, and monitors Cloudera Search processes and resource utilization across services on the cluster. Cloudera Manager helps simplify Cloudera Search administration, but it is not required.	Some cases
Cloudera Navigator	Cloudera Navigator provides governance for Hadoop systems including support for auditing Search operations.	Some cases
Sentry	Sentry enables role-based, fine-grained authorization for Cloudera Search. Sentry can apply a range of restrictions to various actions, such as accessing data, managing configurations through config objects, or creating collections. Restrictions are consistently applied, regardless of how users attempt to complete actions. For example, restricting access to data in a collection restricts that access whether queries come from the command line, a browser, Hue, or through the admin console.	Some cases

Component	Contribution	Applicable To
Oozie	Automates scheduling and management of indexing jobs. Oozie can check for new data and begin indexing jobs as required.	Some cases
Impala	Further analyzes search results.	Some cases
Hive	Further analyzes search results.	Some cases
Parquet	Provides a columnar storage format, enabling especially rapid result returns for structured workloads such as Impala or Hive. Morphlines provide an efficient pipeline for extracting data from Parquet.	Some cases
Avro	Includes metadata that Cloudera Search can use for indexing.	Some cases
Kafka	Search uses this message broker project to increase throughput and decrease latency for handling real-time data.	Some cases
Sqoop	Ingests data in batch and enables data availability for batch indexing.	Some cases

Cloudera Search Architecture

Cloudera Search runs as a distributed service on a set of servers, and each server is responsible for a portion of the searchable data. The data is split into smaller pieces, copies are made of these pieces, and the pieces are distributed among the servers. This provides two main advantages:

- **Dividing** the content into smaller pieces distributes the task of indexing the content among the servers.
- **Duplicating** the pieces of the whole allows queries to be scaled more effectively and enables the system to provide higher levels of availability.



Each Cloudera Search server can handle requests independently. Clients can send requests to index documents or perform searches to any Search server, and that server routes the request to the correct server.

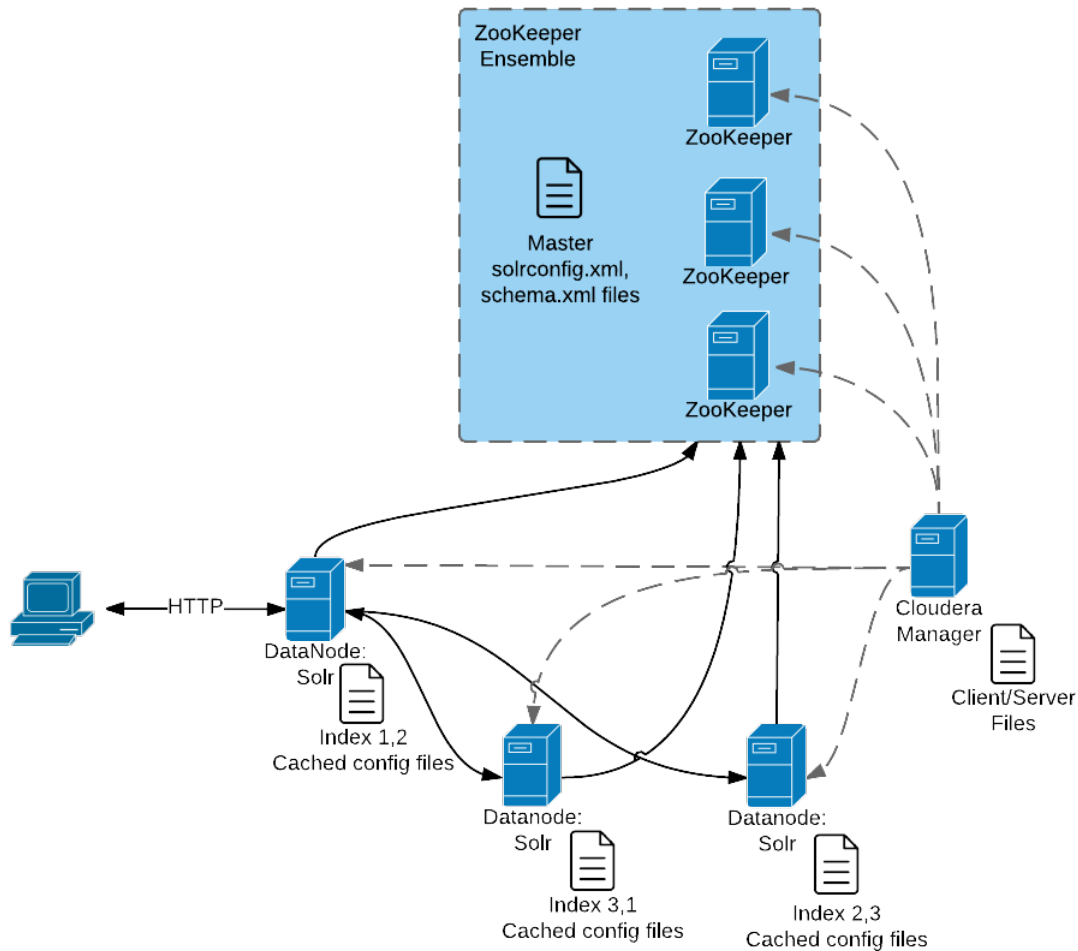
Each Search deployment requires:

- ZooKeeper on at least one host. You can install ZooKeeper, Search, and HDFS on the same host.
- HDFS on at least one, but as many as all hosts. HDFS is commonly installed on all cluster hosts.
- Solr on at least one but as many as all hosts. Solr is commonly installed on all cluster hosts.

More hosts with Solr and HDFS provides the following benefits:

- More Search servers processing requests.
- More Search and HDFS collocation increasing the degree of data locality. More local data provides faster performance and reduces network traffic.

The following graphic illustrates some of the key elements in a typical deployment.



This graphic illustrates:

1. A client submit a query over HTTP.
2. The response is received by the NameNode and then passed to a DataNode.
3. The DataNode distributes the request among other hosts with relevant shards.
4. The results of the query are gathered and returned to the client.

Also notice that the:

- Cloudera Manager provides client and server configuration files to other servers in the deployment.
- ZooKeeper server provides information about the state of the cluster and the other hosts running Solr.

The information a client must send to complete jobs varies:

- For queries, a client must have the hostname of the Solr server and the port to use.
- For actions related to collections, such as adding or deleting collections, the name of the collection is required as well.
- Indexing jobs, such as `MapReduceIndexer` jobs, use a MapReduce driver that starts a MapReduce job. These jobs can also process morphlines and index the results to Solr.

Cloudera Search Configuration Files

Configuration files in a Cloudera Search deployment include:

- Solr config files stored in ZooKeeper. Copies of these files exist on all Solr servers:
 - `solrconfig.xml`: Contains Solr configuration parameters.
 - `schema.xml`: Contains configuration that specifies the fields a document can contain, and how those fields are processed when adding documents to the index, or when querying those fields.
- Files copied from `hadoop-conf` in HDFS configurations to Solr servers:
 - `core-site.xml`
 - `hdfs-site.xml`
 - `ssl-client.xml`
 - `hadoop-env.sh`
 - `topology.map`
 - `topology.py`
- Cloudera Manager manages the following configuration files:
 - `cloudera-monitor.properties`
 - `cloudera-stack-monitor.properties`
- Logging and security configuration files:
 - `log4j.properties`
 - `jaas.conf`
 - `solr.keytab`
 - `sentry-site.xml`

You can use parcels or packages to deploy Search. Some files are always installed to the same location and some files are installed to different locations based on whether the installation is completed using parcels or packages.

Client Files

Client files are always installed to the same location and are required on any host where corresponding services are installed. In a Cloudera Manager environment, Cloudera Manager manages settings. In an unmanaged deployment, all files can be manually edited. Client configuration locations are:

- `/etc/solr/conf` for Solr client settings files
- `/etc/hadoop/conf` for HDFS, MapReduce, and YARN client settings files
- `/etc/zookeeper/conf` for ZooKeeper configuration files

Server Files

Server configuration file locations vary based on how services are installed.

- Cloudera Manager environments store configuration files in `/var/run/`.
- Unmanaged environments store configuration files in `/etc/<service>/conf`. For example:
 - `/etc/solr/conf`
 - `/etc/zookeeper/conf`
 - `/etc/hadoop/conf`

Cloudera Search Tasks and Processes

For content to be searchable, it must exist in CDH and be indexed. Content can either already exist in CDH and be indexed on demand, or it can be updated and indexed continuously. To make content searchable, first ensure that it is ingested or stored in CDH.

Ingestion

You can move content to CDH by using:

- Flume, a flexible, agent-based data ingestion framework.
- A copy utility such as `distcp` for HDFS.
- Sqoop, a structured data ingestion connector.

Indexing

Content must be indexed before it can be searched. Indexing comprises the following steps:

1. Extraction, transformation, and loading (ETL) - Use existing engines or frameworks such as Apache Tika or Cloudera Morphlines.
 - a. Content and metadata extraction
 - b. Schema mapping
2. Index creation using Lucene.
 - a. Index creation
 - b. Index serialization

Indexes are typically stored on a local file system. Lucene supports additional index writers and readers. One HDFS-based interface implemented as part of Apache Blur is integrated with Cloudera Search and has been optimized for CDH-stored indexes. All index data in Cloudera Search is stored in and served from HDFS.

You can index content in three ways:

Batch indexing using MapReduce

To use MapReduce to index documents, run a MapReduce job on content in HDFS to produce a Lucene index. The Lucene index is written to HDFS, and this index is subsequently used by Search services to provide query results.

Batch indexing is most often used when bootstrapping a Search cluster. The Map phase of the MapReduce task parses input into indexable documents, and the Reduce phase indexes the documents produced by the Map. You can also configure a MapReduce-based indexing job to use all assigned resources on the cluster, utilizing multiple reducing steps for intermediate indexing and merging operations, and then writing the reduction to the configured set of shard sets for the service. This makes the batch indexing process as scalable as MapReduce workloads.

Near real-time (NRT) indexing using Flume

Cloudera Search includes a Flume sink that enables you to write events directly to the indexer. This sink provides a flexible, scalable, fault-tolerant, near real-time (NRT) system for processing continuous streams of records to create live-searchable, free-text search indexes. Typically, data ingested using the Flume sink appears in search results in seconds, although you can tune this delay.

Data can flow from multiple sources through multiple flume hosts. These hosts, which can be spread across a network, route this information to one or more Flume indexing sinks. Optionally, you can split the data flow, storing the data in HDFS while writing it to be indexed by Lucene indexers on the cluster. In that scenario, data exists both as data and as indexed data in the same storage infrastructure. The indexing sink extracts relevant data, transforms the material, and loads the results to live Solr search servers. These Solr servers are immediately ready to serve queries to end users or search applications.

This flexible, customizable system scales effectively because parsing is moved from the Solr server to the multiple Flume hosts for ingesting new content.

Search includes parsers for standard data formats including Avro, CSV, Text, HTML, XML, PDF, Word, and Excel. You can extend the system by adding additional custom parsers for other file or data formats in the form of Tika plug-ins. Any type of data can be indexed: a record is a byte array of any format, and custom ETL logic can handle any format variation.

In addition, Cloudera Search includes a simplifying ETL framework called Cloudera Morphlines that can help adapt and pre-process data for indexing. This eliminates the need for specific parser deployments, replacing them with simple commands.

Cloudera Search is designed to handle a variety of use cases:

- Search supports routing to multiple Solr collections to assign a single set of servers to support multiple user groups (multi-tenancy).
- Search supports routing to multiple shards to improve scalability and reliability.
- Index servers can be colocated with live Solr servers serving end-user queries, or they can be deployed on separate commodity hardware, for improved scalability and reliability.
- Indexing load can be spread across a large number of index servers for improved scalability and can be replicated across multiple index servers for high availability.

This flexible, scalable, highly available system provides low latency data acquisition and low latency querying. Instead of replacing existing solutions, Search complements use cases based on batch analysis of HDFS data using MapReduce. In many use cases, data flows from the producer through Flume to both Solr and HDFS. In this system, you can use NRT ingestion and batch analysis tools.

NRT indexing using the API

Other clients can complete NRT indexing. This is done when the client first writes files directly to HDFS and then triggers indexing using the Solr REST API. Specifically, the API does the following:

1. Extract content from the document contained in HDFS, where the document is referenced by a URL.
2. Map the content to fields in the search schema.
3. Create or update a Lucene index.

This is useful if you index as part of a larger workflow. For example, you could trigger indexing from an Oozie workflow.

Querying

After data is available as an index, the query API provided by the Search service allows direct queries to be completed or to be facilitated through a command-line tool or graphical interface. Hue includes a simple GUI-based Search application, or you can create a custom application based on the standard Solr API. Any application that works with Solr is compatible and runs as a search-serving application for Cloudera Search because Solr is the core.

Apache Kudu Overview

Apache Kudu is a columnar storage manager developed for the Hadoop platform. Kudu shares the common technical properties of Hadoop ecosystem applications: It runs on commodity hardware, is horizontally scalable, and supports highly available operation.

Apache Kudu is a top-level project in the Apache Software Foundation.

Kudu's benefits include:

- Fast processing of OLAP workloads.
- Integration with MapReduce, Spark, Flume, and other Hadoop ecosystem components.
- Tight integration with Apache Impala (incubating), making it a good, mutable alternative to using HDFS with Apache Parquet.
- Strong but flexible consistency model, allowing you to choose consistency requirements on a per-request basis, including the option for strict serialized consistency.
- Strong performance for running sequential and random workloads simultaneously.
- Easy administration and management through Cloudera Manager.
- High availability. Tablet Servers and Master use the Raft consensus algorithm, which ensures availability as long as more replicas are available than unavailable. Reads can be serviced by read-only follower tablets, even in the event of a leader tablet failure.
- Structured data model.

By combining all of these properties, Kudu targets support applications that are difficult or impossible to implement on currently available Hadoop storage technologies. Applications for which Kudu is a viable solution include:

- Reporting applications where new data must be immediately available for end users
- Time-series applications that must support queries across large amounts of historic data while simultaneously returning granular queries about an individual entity
- Applications that use predictive models to make real-time decisions, with periodic refreshes of the predictive model based on all historical data

Kudu-Impala Integration

Apache Kudu has tight integration with Apache Impala (incubating), allowing you to use Impala to insert, query, update, and delete data from Kudu tablets using Impala's SQL syntax, as an alternative to using the Kudu APIs to build a custom Kudu application. In addition, you can use JDBC or ODBC to connect existing or new applications written in any language, framework, or business intelligence tool to your Kudu data, using Impala as the broker.

- **CREATE/ALTER/DROP TABLE** - Impala supports creating, altering, and dropping tables using Kudu as the persistence layer. The tables follow the same internal/external approach as other tables in Impala, allowing for flexible data ingestion and querying.
- **INSERT** - Data can be inserted into Kudu tables from Impala using the same mechanisms as any other table with HDFS or HBase persistence.
- **UPDATE/DELETE** - Impala supports the `UPDATE` and `DELETE` SQL commands to modify existing data in a Kudu table row-by-row or as a batch. The syntax of the SQL commands is designed to be as compatible as possible with existing solutions. In addition to simple `DELETE` or `UPDATE` commands, you can specify complex joins in the `FROM` clause of the query, using the same syntax as a regular `SELECT` statement.
- **Flexible Partitioning** - Similar to partitioning of tables in Hive, Kudu allows you to dynamically pre-split tables by hash or range into a predefined number of tablets, in order to distribute writes and queries evenly across your cluster. You can partition by any number of primary key columns, with any number of hashes, a list of split rows, or a combination of these. A partition scheme is required.
- **Parallel Scan** - To achieve the highest possible performance on modern hardware, the Kudu client used by Impala parallelizes scans across multiple tablets.
- **High-efficiency queries** - Where possible, Impala pushes down predicate evaluation to Kudu, so that predicates are evaluated as close as possible to the data. Query performance is comparable to Parquet in many workloads.

Example Use Cases

Streaming Input with Near Real Time Availability

A common business challenge is one where new data arrives rapidly and constantly, and the same data needs to be available in near real time for reads, scans, and updates. Kudu offers the powerful combination of fast inserts and updates with efficient columnar scans to enable real-time analytics use cases on a single storage layer.

Time-Series Application with Widely Varying Access Patterns

A time-series schema is one in which data points are organized and keyed according to the time at which they occurred. This can be useful for investigating the performance of metrics over time or attempting to predict future behavior based on past data. For instance, time-series customer data might be used both to store purchase click-stream history and to predict future purchases, or for use by a customer support representative. While these different types of analysis are occurring, inserts and mutations might also be occurring individually and in bulk, and become available immediately to read workloads. Kudu can handle all of these access patterns simultaneously in a scalable and efficient manner.

Kudu is a good fit for time-series workloads for several reasons. With Kudu's support for hash-based partitioning, combined with its native support for compound row keys, it is simple to set up a table spread across many servers without the risk of "hotspotting" that is commonly observed when range partitioning is used. Kudu's columnar storage engine is also beneficial in this context, because many time-series workloads read only a few columns, as opposed to the whole row.

In the past, you might have needed to use multiple datastores to handle different data access patterns. This practice adds complexity to your application and operations, and duplicates your data, doubling (or worse) the amount of storage required. Kudu can handle all of these access patterns natively and efficiently, without the need to off-load work to other datastores.

Predictive Modeling

Data scientists often develop predictive learning models from large sets of data. The model and the data might need to be updated or modified often as the learning takes place or as the situation being modeled changes. In addition, the scientist might want to change one or more factors in the model to see what happens over time. Updating a large set of data stored in files in HDFS is resource-intensive, as each file needs to be completely rewritten. In Kudu, updates happen in near real time. The scientist can tweak the value, re-run the query, and refresh the graph in seconds or minutes, rather than hours or days. In addition, batch or incremental algorithms can be run across the data at any time, with near-real-time results.

Combining Data In Kudu With Legacy Systems

Companies generate data from multiple sources and store it in a variety of systems and formats. For instance, some of your data might be stored in Kudu, some in a traditional RDBMS, and some in files in HDFS. You can access and query all of these sources and formats using Impala, without the need to change your legacy systems.

Related Information

- [Apache Kudu Concepts and Architecture](#)
- [Overview of Apache Kudu Installation and Upgrade in CDH](#)
- [Kudu Security Overview](#)
- [More Resources for Apache Kudu](#)

Apache Sentry Overview

Apache Sentry is a granular, role-based authorization module for Hadoop. Sentry provides the ability to control and enforce precise levels of privileges on data for authenticated users and applications on a Hadoop cluster. Sentry currently works out of the box with Apache Hive, Hive Metastore/HCatalog, Apache Solr, Impala, and HDFS (limited to Hive table data).

Sentry is designed to be a pluggable authorization engine for Hadoop components. It allows you to define authorization rules to validate a user or application's access requests for Hadoop resources. Sentry is highly modular and can support authorization for a wide variety of data models in Hadoop.

For more information, see [Authorization With Apache Sentry](#).

Apache Spark Overview

[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:

- [Spark SQL](#) - Module for working with structured data. Allows you to seamlessly mix SQL queries with Spark programs.
- [Spark Streaming](#) - API that allows you to build scalable fault-tolerant streaming applications.
- [MLlib](#) - API that implements common [machine learning](#) algorithms.
- [GraphX](#) - API for graphs and graph-parallel computation.

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 supports two cluster managers: YARN and Spark Standalone. When run on YARN, Spark application processes are managed by the YARN ResourceManager and NodeManager roles. When run on Spark Standalone, Spark application processes are managed by Spark Master and Worker roles.

Unsupported Features

The following Spark features are not supported:

- Spark SQL:
 - Thrift JDBC/ODBC server
 - Spark SQL CLI
- Spark Dataset API
- SparkR
- GraphX
- Spark on Scala 2.11
- Mesos cluster manager

Related Information

- [Managing Spark](#)
- [Monitoring Spark Applications](#)
- [Spark Authentication](#)
- [Spark Encryption](#)
- [Cloudera Spark forum](#)
- [Apache Spark documentation](#)

File Formats and Compression

CDH supports all standard Hadoop file formats. For information about the file formats, see the File-Based Data Structures section of the Hadoop I/O chapter in [Hadoop: The Definitive Guide](#).

The file format has a significant impact on performance. Use [Avro](#) if your use case typically scans or retrieves all of the fields in a row in each query. [Parquet](#) is a better choice if your dataset has many columns, and your use case typically involves working with a subset of those columns instead of entire records. For more information, see this [Parquet versus Avro benchmark study](#).

All file formats include support for [compression](#), which affects the size of data on the disk and, consequently, the amount of I/O and CPU resources required to serialize and deserialize data.

Using Apache Parquet Data Files with CDH

[Apache Parquet](#) is a [columnar storage](#) format available to any component in the Hadoop ecosystem, regardless of the data processing framework, data model, or programming language. The Parquet file format incorporates several features that support data warehouse-style operations:

- Columnar storage layout - A query can examine and perform calculations on all values for a column while reading only a small fraction of the data from a data file or table.
- Flexible compression options - Data can be compressed with any of several codecs. Different data files can be compressed differently.
- Innovative encoding schemes - Sequences of identical, similar, or related data values can be represented in ways that save disk space and memory. The encoding schemes provide an extra level of space savings beyond overall compression for each data file.
- Large file size - The layout of Parquet data files is optimized for queries that process large volumes of data, with individual files in the multimegabyte or even gigabyte range.

Parquet is automatically installed when you install CDH, and the required libraries are automatically placed in the classpath for all CDH components. Copies of the libraries are in `/usr/lib/parquet` or `/opt/cloudera/parcels/CDH/lib/parquet`.

CDH lets you use the component of your choice with the Parquet file format for each phase of data processing. For example, you can read and write Parquet files using Pig and MapReduce jobs. You can convert, transform, and query Parquet tables through Hive, Impala, and Spark. And you can interchange data files between all of these components.

Compression for Parquet Files

For most CDH components, by default Parquet data files are not compressed. Cloudera recommends enabling compression to reduce disk usage and increase read and write performance.

You do not need to specify configuration to read a compressed Parquet file. However, to write a compressed Parquet file, you must specify the compression type. The supported compression types, the compression default, and how you specify compression depends on the CDH component writing the files.

Using Parquet Files in HBase

Parquet files in HBase is a common use case. See [Using a Custom MapReduce Job](#).

Using Parquet Tables in Hive

To create a table named `PARQUET_TABLE` that uses the Parquet format, use a command like the following, substituting your own table name, column names, and data types:

```
hive> CREATE TABLE parquet_table_name (x INT, y STRING) STORED AS PARQUET;
```



Note:

- Once you create a Parquet table, you can query it or insert into it through other components such as Impala and Spark.
- Set `dfs.block.size` to 256 MB in `hdfs-site.xml`.

If the table will be populated with data files generated outside of Impala and Hive, you can create the table as an external table pointing to the location where the files will be created:

```
hive> create external table parquet_table_name (x INT, y STRING)
  ROW FORMAT SERDE 'parquet.hive.serde.ParquetHiveSerDe'
  STORED AS
    INPUTFORMAT "parquet.hive.DeprecatedParquetInputFormat"
    OUTPUTFORMAT "parquet.hive.DeprecatedParquetOutputFormat"
  LOCATION '/test-warehouse/tinytable';
```

To populate the table with an `INSERT` statement, and to read the table with a `SELECT` statement, see [Using the Parquet File Format with Impala Tables](#).

To set the compression type to use when writing data, configure the `parquet.compression` property:

```
set parquet.compression=GZIP;
INSERT OVERWRITE TABLE tinytable SELECT * FROM texttable;
```

The supported compression types are `UNCOMPRESSED`, `GZIP`, and `SNAPPY`.

Using Parquet Tables in Impala

Impala can create tables that use Parquet data files, insert data into those tables, convert the data into Parquet format, and query Parquet data files produced by Impala or other components. The only syntax required is the `STORED AS`

PARQUET clause on the CREATE TABLE statement. After that, all SELECT, INSERT, and other statements recognize the Parquet format automatically. For example, a session in the `impala-shell` interpreter might look as follows:

```
[localhost:21000] > create table parquet_table (x int, y string) stored as parquet;
[localhost:21000] > insert into parquet_table select x, y from some_other_table;
Inserted 50000000 rows in 33.52s
[localhost:21000] > select y from parquet_table where x between 70 and 100;
```

Once you create a Parquet table this way in Impala, you can query it or insert into it through either Impala or Hive.

The Parquet format is optimized for working with large data files. In Impala 2.0 and higher, the default size of Parquet files written by Impala is 256 MB; in lower releases, 1 GB. Avoid using the `INSERT ... VALUES` syntax, or partitioning the table at too granular a level, if that would produce a large number of small files that cannot use Parquet optimizations for large data chunks.

Inserting data into a partitioned Impala table can be a memory-intensive operation, because each data file requires a memory buffer to hold the data before it is written. Such inserts can also exceed HDFS limits on simultaneous open files, because each node could potentially write to a separate data file for each partition, all at the same time. Make sure table and column statistics are in place for any table used as the source for an `INSERT ... SELECT` operation into a Parquet table. If capacity problems still occur, consider splitting insert operations into one `INSERT` statement per partition.

Impala can query Parquet files that use the `PLAIN`, `PLAIN_DICTIONARY`, `BIT_PACKED`, and `RLE` encodings. Currently, Impala does not support `RLE_DICTIONARY` encoding. When creating files outside of Impala for use by Impala, make sure to use one of the supported encodings. In particular, for MapReduce jobs, `parquet.writer.version` must not be defined (especially as `PARQUET_2_0`) for writing the configurations of Parquet MR jobs. Use the default version (or format). The default format, 1.0, includes some enhancements that are compatible with older versions. Data using the 2.0 format might not be consumable by Impala, due to use of the `RLE_DICTIONARY` encoding.

If you use Sqoop to convert RDBMS data to Parquet, be careful with interpreting any resulting values from `DATE`, `DATETIME`, or `TIMESTAMP` columns. The underlying values are represented as the Parquet `INT64` type, which is represented as `BIGINT` in the Impala table. The Parquet values represent the time in milliseconds, while Impala interprets `BIGINT` as the time in seconds. Therefore, if you have a `BIGINT` column in a Parquet table that was imported this way from Sqoop, divide the values by 1000 when interpreting as the `TIMESTAMP` type.

For complete instructions and examples, see [Using the Parquet File Format with Impala Tables](#).

Using Parquet Files in MapReduce

MapReduce requires Thrift in its `CLASSPATH` and in `libjars` to access Parquet files. It also requires `parquet-format` in `libjars`. Set up the following before running MapReduce jobs that access Parquet data files:

```
if [ -e /opt/cloudera/parcels/CDH ] ; then
    CDH_BASE=/opt/cloudera/parcels/CDH
else
    CDH_BASE=/usr
fi
THRIFTJAR=`ls -l $CDH_BASE/lib/hive/lib/libthrift*.jar | awk '{print $9}' | head -1`
export HADOOP_CLASSPATH=$HADOOP_CLASSPATH:$THRIFTJAR
export LIBJARS=`echo "$CLASSPATH" | awk 'BEGIN { RS = ":" } { print }' | grep
parquet-format | tail -1`
export LIBJARS=$LIBJARS,$THRIFTJAR

hadoop jar my-parquet-mr.jar -libjars $LIBJARS
```

Reading Parquet Files in MapReduce

Using the `Example` helper classes in the Parquet JAR files, a simple map-only MapReduce job that reads Parquet files can use the `ExampleInputFormat` class and the `Group` value class. The following example demonstrates how to read a Parquet file in a MapReduce job; portions of code specific to Parquet are shown in bold.

```
import static java.lang.Thread.sleep;
import java.io.IOException;
```

```

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.conf.Configured;
import org.apache.hadoop.util.Tool;
import org.apache.hadoop.util.ToolRunner;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.NullWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.mapreduce.Mapper.Context;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;

import parquet.Log;
import parquet.example.data.Group;
import parquet.hadoop.example.ExampleInputFormat;

public class TestReadParquet extends Configured
    implements Tool {
    private static final Log LOG =
        Log.getLog(TestReadParquet.class);

    /**
     * Read a Parquet record
     */
    public static class MyMap extends
        Mapper<LongWritable, Group, NullWritable, Text> {

        @Override
        public void map(LongWritable key, Group value, Context context) throws IOException,
            InterruptedException {
            NullWritable outKey = NullWritable.get();
            String outputRecord = "";
            // Get the schema and field values of the record
            String inputRecord = value.toString();
            // Process the value, create an output record
            // ...
            context.write(outKey, new Text(outputRecord));
        }
    }

    public int run(String[] args) throws Exception {

        Job job = new Job(getConf());

        job.setJarByClass(getClass());
        job.setJobName(getClass().getName());
        job.setMapOutputKeyClass(LongWritable.class);
        job.setMapOutputValueClass(Text.class);
        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(Text.class);
        job.setMapperClass(MyMap.class);
        job.setNumReduceTasks(0);

        job.setInputFormatClass(ExampleInputFormat.class);
        job.setOutputFormatClass(TextOutputFormat.class);

        FileInputFormat.setInputPaths(job, new Path(args[0]));
        FileOutputFormat.setOutputPath(job, new Path(args[1]));

        job.waitForCompletion(true);
        return 0;
    }

    public static void main(String[] args) throws Exception {
        try {
            int res = ToolRunner.run(new Configuration(), new TestReadParquet(), args);
            System.exit(res);
        } catch (Exception e) {

```

```

        e.printStackTrace();
        System.exit(255);
    }
}

```

Writing Parquet Files in MapReduce

When writing Parquet files, you must provide a schema. Specify the schema in the `run` method of the job before submitting it; for example:

```

...
import parquet.Log;
import parquet.example.data.Group;
import parquet.hadoop.example.GroupWriteSupport;
import parquet.hadoop.example.ExampleInputFormat;
import parquet.hadoop.example.ExampleOutputFormat;
import parquet.hadoop.metadata.CompressionCodecName;
import parquet.hadoop.ParquetFileReader;
import parquet.hadoop.metadata.ParquetMetadata;
import parquet.schema.MessageType;
import parquet.schema.MessageTypeParser;
import parquet.schema.Type;
...
public int run(String[] args) throws Exception {
...

    String writeSchema = "message example {\n" +
        "required int32 x;\n" +
        "required int32 y;\n" +
        "}";
    ExampleOutputFormat.setSchema(
        job,
        MessageTypeParser.parseMessageType(writeSchema));

    job.submit();
}

```

If input files are in Parquet format, the schema can be extracted using the `getSchema` method:

```

import org.apache.hadoop.fs.FileSystem;
import org.apache.hadoop.fs.FileStatus;
import org.apache.hadoop.fs.LocatedFileStatus;
import org.apache.hadoop.fs.RemoteIterator;
...

public int run(String[]
    args) throws Exception {
...

    String inputFile = args[0];
    Path parquetFilePath = null;
    // Find a file in case a directory was passed

    RemoteIterator<LocatedFileStatus> it = FileSystem.get(getConf()).listFiles(new
    Path(inputFile), true);
    while(it.hasNext()) {
        FileStatus fs = it.next();

        if(fs.isFile()) {
            parquetFilePath = fs.getPath();
            break;
        }
    }
    if(parquetFilePath == null) {
        LOG.error("No file found for " + inputFile);
        return 1;
    }
    ParquetMetadata readFooter =
        ParquetFileReader.readFooter(getConf(), parquetFilePath);
    MessageType schema =

```

```
readFooter.getFileMetaData().getSchema();
GroupWriteSupport.setSchema(schema, getConf());

job.submit();
```

You can then write records in the mapper by composing a `Group` value using the `Example` classes and no key:

```
protected void map(LongWritable key, Text value,
    Mapper<LongWritable, Text, Void, Group>.Context context)
    throws java.io.IOException, InterruptedException {
    int x;
    int y;
    // Extract the desired output values from the input text
    //
    Group group = factory.newGroup()
        .append("x", x)
        .append("y", y);
    context.write(null, group);
}
}
```

To set the compression type before submitting the job, invoke the `setCompression` method:

```
ExampleOutputFormat.setCompression(job, compression_type);
```

The supported compression types are `CompressionCodecName.UNCOMPRESSED`, `CompressionCodecName.GZIP`, and `CompressionCodecName.SNAPPY`.

Using Parquet Files in Pig

In CDH 5.12 and earlier, Pig could write to Parquet files using the `ParquetStorer` and `ParquetLoader` libraries by using a two-step process to write to Hive tables stored in the Parquet format:

- First you write to the Parquet files.
- Then you execute the DDL or DML statements from Hive.

This process is described [below](#).

In CDH 5.13 and later, Pig can directly write to Parquet tables using `HCatalog`, the table management and storage layer for Hadoop. Both partitioned and non-partitioned tables are supported. This process is described [below](#).

Using the `ParquetLoader` and `ParquetStorer` Libraries to Write to Parquet Hive Tables with Pig

The following process writes to Parquet-formatted Hive tables with Pig using the `ParquetLoader` and the `ParquetStorer` libraries. It does not use `HCatalog`. This method of writing to Parquet-formatted Hive tables with Pig is supported in CDH 5.12 and earlier, and continues to be supported in CDH 5.13 and later. See the Apache documentation for [Pig Latin Basics](#), which is the reference for the Pig scripting language.

1. If the external table is created and populated, read the Parquet file into a Pig relation and specify the relation schema. In the following example, the relation is named **A** and the example directories and schema definition are displayed in a bold font. These can be replaced with your directory names and schema definition:

```
grunt> A = LOAD '/user/hive/warehouse/web_logs_par/dd=2015-11-18' USING
    parquet.pig.ParquetLoader AS (version:int, app:chararray,
    bytes:int, city:chararray, client_ip:chararray);
```

2. Write the Parquet file from the Pig relation **A** into a table partition, specifying the partition location. The partition directory is created if it does not already exist. The example directories and schema definition are displayed in a bold font:

```
grunt> store A into '/user/hive/warehouse/web_logs_par/dd=2015-11-19' USING
```

```
parquet.pig.ParquetStorer;
```

To set the compression type, set the `parquet.compression` property before the first `store` instruction in a Pig script. For example, the following statement sets the compression type to `gzip`:

```
grunt> SET parquet.compression gzip;
```

The supported compression types are `uncompressed`, `gzip`, and `snappy`, which is the default.

- Now that the partition data is in the appropriate directory of a Hive table, create the corresponding partition in Hive using Beeline. Example directories are displayed in a bold font:

```
ALTER TABLE web_logs_par ADD PARTITION (dd='2015-11-19') LOCATION
'/user/hive/warehouse/web_logs_par/dd=2015-11-19';
```

Now, the table contains the new partition and can be used from Hive or Impala.

Using HCatalog to Write to Parquet Hive Tables with Pig

The following example shows how to dynamically write to partitioned Hive tables in the Parquet format with Pig using HCatalog. In this example, we have a Hive table that stores foreign currency exchange rates over a period of time, which is partitioned by the year and month. The table was created with the following DDL statement:

```
CREATE TABLE fxbyyearmonth (price double) PARTITIONED BY (year string, month string)
STORED AS PARQUET;
```

We want to move data from the above table to another Hive table that stores foreign currency exchange rates that is partitioned by the year, month, *and* the currency. This table was created with the following DDL statement:

```
CREATE TABLE fxbyyearmonthccy (price double) PARTITIONED BY (year string, month string,
currency
string) STORED AS PARQUET;
```

- Run the following command to bring up the Grunt shell while simultaneously invoking HCatalog:

```
pig -useHCatalog
```

- Load the data into Pig using a relation named `A`:

```
grunt> A = LOAD 'fxbyyearmonth' USING org.apache.hive.hcatalog.pig.HCatLoader();
```

If you run the `describe A` command in the Grunt shell, it returns the following information:

```
A: {price: double,currency: chararray,year: chararray,month: chararray}}
```

This shows that the `HCatLoader` places the partition columns at the end of the relation `A` schema. These column names are displayed in bold font.



Note: If you want to load only one partition, you must apply an additional filter operation. Filter operators push down information to HCatalog so it loads only that part of the table. For example:

```
grunt> B = FILTER A BY year == '2016';
```

3. Dynamically write partitions. HCatStorer can dynamically write partitions when the partition columns have the expected positions in the schema. In our example, the dataset must be ordered as: price, month, currency. To do this in Pig, run:

```
grunt> B = FOREACH A generate price, year, month, currency;
```

4. Store the dataset contained in B into the Hive table:

```
grunt> STORE B INTO 'fxbyyearmonthccy' USING org.apache.hive.hcatalog.pig.HCatStorer();
```

This stores everything from the dataset in B into the Hive table by putting all records in their correct partitions dynamically. All partition columns must be present for this to work properly. After running the STORE command, the table contains the dataset and can be used from Hive or Impala.



Note: You can also statically specify the target partition. In this case, you do not need the columns to be present in the dataset. For example, you can filter for only a certain partition with Pig, and then use only the price column:

```
grunt> C = FILTER B BY (year == '2016') AND (month=='01') AND  
(currency=='EUR');  
grunt> D = FOREACH C GENERATE price;  
grunt> STORE D into 'fxbyyearmonthccy' USING  
org.apache.hive.hcatalog.pig.HCatStorer('year=2016,month=01,currency=EUR');
```

Using Parquet Files in Spark

See [Accessing External Storage from Spark](#) and [Accessing Parquet Files From Spark SQL Applications](#).

Parquet File Interoperability

Impala has always included Parquet support, using high-performance code written in C++ to read and write Parquet files. The Parquet JARs for use with Hive, Pig, and MapReduce are available with CDH 4.5 and higher. Using the Java-based Parquet implementation on a CDH release lower than CDH 4.5 is not supported.

A Parquet table created by Hive can typically be accessed by Impala 1.1.1 and higher with no changes, and vice versa. Before Impala 1.1.1, when Hive support for Parquet was not available, Impala wrote a dummy SerDe class name into each data file. These older Impala data files require a one-time ALTER TABLE statement to update the metadata for the SerDe class name before they can be used with Hive. See [Apache Impala \(incubating\) Incompatible Changes and Limitations](#) for details.

A Parquet file written by Hive, Impala, Pig, or MapReduce can be read by any of the others. Different defaults for file and block sizes, compression and encoding settings, and so on might cause performance differences depending on

which component writes or reads the data files. For example, Impala typically sets the HDFS block size to 256 MB and divides the data files into 256 MB chunks, so that each I/O request reads an entire data file.

In CDH 5.5 and higher, non-Impala components that write Parquet files include extra padding to ensure that the Parquet row groups are aligned with HDFS data blocks. The maximum amount of padding is controlled by the `parquet.writer.max-padding` setting, specified as a number of bytes. By default, up to 8 MB of padding can be added to the end of each row group. This alignment helps prevent remote reads during Impala queries. The setting does not apply to Parquet files written by Impala, because Impala always writes each Parquet file as a single HDFS data block.

Each release may have limitations. The following are current limitations in CDH:

- The `TIMESTAMP` data type in Parquet files is not supported in Hive, Pig, or MapReduce in CDH 4. Attempting to read a Parquet table created with Impala that includes a `TIMESTAMP` column fails.
- Parquet has not been tested with HCatalog. Without HCatalog, Pig cannot correctly read dynamically partitioned tables; this is true for all file formats.
- Impala supports table columns using nested data types or complex data types such as `map`, `struct`, or `array` only in Impala 2.3 (corresponding to CDH 5.5) and higher. Impala 2.2 (corresponding to CDH 5.4) can query only the scalar columns of Parquet files containing such types. Lower releases of Impala cannot query any columns from Parquet data files that include such types.
- Cloudera supports some but not all of the object models from the upstream `Parquet-MR` project. Currently supported object models are:
 - `parquet-avro` (recommended for Cloudera users)
 - `parquet-thrift`
 - `parquet-protobuf`
 - `parquet-pig`
 - The Impala and Hive object models built into those components, not available in external libraries. (CDH does not include the `parquet-hive` module of the `parquet-mr` project, because recent versions of Hive have Parquet support built in.)

Parquet File Structure

To examine the internal structure and data of Parquet files, you can use the `parquet-tools` command that comes with CDH. Make sure this command is in your `$PATH`. (Typically, it is symlinked from `/usr/bin`; sometimes, depending on your installation setup, you might need to locate it under a CDH-specific `bin` directory.) The arguments to this command let you perform operations such as:

- `cat`: Print a file's contents to standard out. In CDH 5.5 and higher, you can use the `-j` option to output JSON.
- `head`: Print the first few records of a file to standard output.
- `schema`: Print the Parquet schema for the file.
- `meta`: Print the file footer metadata, including key-value properties (like Avro schema), compression ratios, encodings, compression used, and row group information.
- `dump`: Print all data and metadata.

Use `parquet-tools -h` to see usage information for all the arguments. Here are some examples showing `parquet-tools` usage:

```
$ # Be careful doing this for a big file! Use parquet-tools head to be safe.
$ parquet-tools cat sample.parq
year = 1992
month = 1
day = 2
dayofweek = 4
dep_time = 748
crs_dep_time = 750
arr_time = 851
crs_arr_time = 846
carrier = US
flight_num = 53
```

```

actual_elapsed_time = 63
crs_elapsed_time = 56
arrdelay = 5
depdelay = -2
origin = CMH
dest = IND
distance = 182
cancelled = 0
diverted = 0

year = 1992
month = 1
day = 3
...

```

```

$ parquet-tools head -n 2 sample.parq
year = 1992
month = 1
day = 2
dayofweek = 4
dep_time = 748
crs_dep_time = 750
arr_time = 851
crs_arr_time = 846
carrier = US
flight_num = 53
actual_elapsed_time = 63
crs_elapsed_time = 56
arrdelay = 5
depdelay = -2
origin = CMH
dest = IND
distance = 182
cancelled = 0
diverted = 0

year = 1992
month = 1
day = 3
...

```

```

$ parquet-tools schema sample.parq
message schema {
  optional int32 year;
  optional int32 month;
  optional int32 day;
  optional int32 dayofweek;
  optional int32 dep_time;
  optional int32 crs_dep_time;
  optional int32 arr_time;
  optional int32 crs_arr_time;
  optional binary carrier;
  optional int32 flight_num;
  ...
}

```

```

$ parquet-tools meta sample.parq
creator:          impala version 2.2.0-cdh5.4.3 (build
517bb0f71cd604a00369254ac6d88394df83e0f6)

file schema:      schema
-----
year:             OPTIONAL INT32  R:0 D:1
month:            OPTIONAL INT32  R:0 D:1
day:              OPTIONAL INT32  R:0 D:1

```



```

dayofweek:          OPTIONAL INT32 R:0 D:1
dep_time:           OPTIONAL INT32 R:0 D:1
crs_dep_time:       OPTIONAL INT32 R:0 D:1
arr_time:           OPTIONAL INT32 R:0 D:1
crs_arr_time:       OPTIONAL INT32 R:0 D:1
carrier:            OPTIONAL BINARY R:0 D:1
flight_num:         OPTIONAL INT32 R:0 D:1
...

row group 1:        RC:20636601 TS:265103674
-----
year:               INT32 SNAPPY DO:4 FPO:35 SZ:10103/49723/4.92 VC:20636601
ENC:PLAIN_DICTIONARY,RLE,PLAIN
month:              INT32 SNAPPY DO:10147 FPO:10210 SZ:11380/35732/3.14 VC:20636601
ENC:PLAIN_DICTIONARY,RLE,PLAIN
day:                INT32 SNAPPY DO:21572 FPO:21714 SZ:3071658/9868452/3.21 VC:20636601
ENC:PLAIN_DICTIONARY,RLE,PLAIN
dayofweek:          INT32 SNAPPY DO:3093276 FPO:3093319 SZ:2274375/5941876/2.61
VC:20636601 ENC:PLAIN_DICTIONARY,RLE,PLAIN
dep_time:           INT32 SNAPPY DO:5367705 FPO:5373967 SZ:28281281/28573175/1.01
VC:20636601 ENC:PLAIN_DICTIONARY,RLE,PLAIN
crs_dep_time:       INT32 SNAPPY DO:33649039 FPO:33654262 SZ:10220839/11574964/1.13
VC:20636601 ENC:PLAIN_DICTIONARY,RLE,PLAIN
arr_time:           INT32 SNAPPY DO:43869935 FPO:43876489 SZ:28562410/28797767/1.01
VC:20636601 ENC:PLAIN_DICTIONARY,RLE,PLAIN
crs_arr_time:       INT32 SNAPPY DO:72432398 FPO:72438151 SZ:10908972/12164626/1.12
VC:20636601 ENC:PLAIN_DICTIONARY,RLE,PLAIN
carrier:            BINARY SNAPPY DO:83341427 FPO:83341558 SZ:114916/128611/1.12
VC:20636601 ENC:PLAIN_DICTIONARY,RLE,PLAIN
flight_num:         INT32 SNAPPY DO:83456393 FPO:83488603 SZ:10216514/11474301/1.12
VC:20636601 ENC:PLAIN_DICTIONARY,RLE,PLAIN
...

```

Examples of Java Programs to Read and Write Parquet Files

You can find full examples of Java code at the Cloudera [Parquet examples](#) GitHub repository.

The [TestReadWriteParquet.java](#) example demonstrates the “identity” transform. It reads any Parquet data file and writes a new file with exactly the same content.

The [TestReadParquet.java](#) example reads a Parquet data file, and produces a new text file in CSV format with the same content.

Using Apache Avro Data Files with CDH

[Apache Avro](#) is a serialization system. Avro supports rich data structures, a compact binary encoding, and a container file for sequences of Avro data (often referred to as **Avro data files**). Avro is language-independent and there are several language bindings for it, including Java, C, C++, Python, and Ruby.

Avro data files have the `.avro` extension. Make sure the files you create have this extension, because some tools use it to determine which files to process as Avro (for example, `AvroInputFormat` and `AvroAsTextInputFormat` for MapReduce and streaming).

Avro does not rely on generated code, so processing data imported from Flume or Sqoop 1 is simpler than using Hadoop Writables in SequenceFiles, where you must ensure that the generated classes are on the processing job classpath. Pig and Hive cannot easily process SequenceFiles with custom Writables, so users often revert to using text, which has disadvantages in compactness and compressibility. Generally, you cannot split compressed text, which makes it difficult to process efficiently using MapReduce.

All components in CDH that produce or consume files support Avro data files.

Compression for Avro Data Files

By default Avro data files are not compressed, but Cloudera recommends enabling compression to reduce disk usage and increase read and write performance. Avro data files support [Deflate](#) and [Snappy](#) compression. Snappy is faster, but Deflate is slightly more compact.

You do not need to specify configuration to read a compressed Avro data file. However, to write an Avro data file, you must specify the type of compression. How you specify compression depends on the component.

Using Avro Data Files in Flume

The [HDFSEventSink](#) used to serialize event data onto HDFS supports plug-in implementations of the [EventSerializer](#) interface. Implementations of this interface have full control over the serialization format and can be used in cases where the default serialization format provided by the sink is insufficient.

An abstract implementation of the EventSerializer interface, called [AbstractAvroEventSerializer](#), is provided with Flume. This class can be extended to support custom schemas for Avro serialization over HDFS. The [FlumeEventAvroEventSerializer](#) class provides a simple implementation that maps the events to a representation of a String header map and byte payload in Avro. Use this class by setting the serializer property of the sink as follows:

```
agent-name.sinks.sink-name.serializer = AVRO_EVENT
```

Using Avro Data Files in Hive

The following example demonstrates how to create a Hive table backed by Avro data files:

```
CREATE TABLE doctors
ROW FORMAT
SERDE 'org.apache.hadoop.hive.serde2.avro.AvroSerDe'
STORED AS
INPUTFORMAT 'org.apache.hadoop.hive.ql.io.avro.AvroContainerInputFormat'
OUTPUTFORMAT 'org.apache.hadoop.hive.ql.io.avro.AvroContainerOutputFormat'
TBLPROPERTIES ('avro.schema.literal'='{
  "namespace": "testing.hive.avro.serde",
  "name": "doctors",
  "type": "record",
  "fields": [
    {
      "name": "number",
      "type": "int",
      "doc": "Order of playing the role"
    },
    {
      "name": "first_name",
      "type": "string",
      "doc": "first name of actor playing role"
    },
    {
      "name": "last_name",
      "type": "string",
      "doc": "last name of actor playing role"
    },
    {
      "name": "extra_field",
      "type": "string",
      "doc": "an extra field not in the original file",
      "default": "fishfingers and custard"
    }
  ]
}');

LOAD DATA LOCAL INPATH '/usr/share/doc/hive-0.7.1+42.55/examples/files/doctors.avro'
INTO TABLE doctors;
```

You can also create an Avro backed Hive table by using an Avro schema file:

```
CREATE TABLE my_avro_table(notused INT)
ROW FORMAT SERDE
'org.apache.hadoop.hive.serde2.avro.AvroSerDe'
WITH SERDEPROPERTIES (
  'avro.schema.url'='file:///tmp/schema.avsc')
STORED as INPUTFORMAT
'org.apache.hadoop.hive.ql.io.avro.AvroContainerInputFormat'
```

```
OUTPUTFORMAT
'org.apache.hadoop.hive.ql.io.avro.AvroContainerOutputFormat';
```

`avro.schema.url` is a URL (here a `file://` URL) pointing to an Avro schema file used for reading and writing. It could also be an `hdfs://` URL; for example, `hdfs://hadoop-namenode-uri/examplefile`.

To enable Snappy compression on output files, run the following before writing to the table:

```
SET hive.exec.compress.output=true;
SET avro.output.codec=snappy;
```

Also include the `snappy-java` JAR in `--auxpath`, which is located at `/usr/lib/hive/lib/snappy-java-1.0.4.1.jar` or `/opt/cloudera/parcels/CDH/lib/hive/lib/snappy-java-1.0.4.1.jar`.

[Haivvreo SerDe](#) has been merged into Hive as [AvroSerDe](#) and is no longer supported in its original form. `schema.url` and `schema.literal` have been changed to `avro.schema.url` and `avro.schema.literal` as a result of the merge. If you were using [Haivvreo SerDe](#), you can use the Hive `AvroSerDe` with tables created with the `Haivvreo SerDe`. For example, if you have a table `my_avro_table` that uses the `Haivvreo SerDe`, add the following to make the table use the new `AvroSerDe`:

```
ALTER TABLE my_avro_table SET SERDE 'org.apache.hadoop.hive.serde2.avro.AvroSerDe';

ALTER TABLE my_avro_table SET FILEFORMAT
INPUTFORMAT 'org.apache.hadoop.hive.ql.io.avro.AvroContainerInputFormat'
OUTPUTFORMAT 'org.apache.hadoop.hive.ql.io.avro.AvroContainerOutputFormat';
```

Using Avro Data Files in MapReduce

The Avro MapReduce API is an Avro module for running MapReduce programs that produce or consume Avro data files.

If you are using [Maven](#), add the following dependency to your POM:

```
<dependency>
  <groupId>org.apache.avro</groupId>
  <artifactId>avro-mapred</artifactId>
  <version>1.7.6-cdh5.12.0</version>
  <classifier>hadoop2</classifier>
</dependency>
```

Then write your program, using the [Avro MapReduce javadoc](#) for guidance.

At run time, include the `avro` and `avro-mapred` JARs in the `HADOOP_CLASSPATH` and the `avro`, `avro-mapred` and `paranamer` JARs in `-libjars`.

To enable Snappy compression on output, call `AvroJob.setOutputCodec(job, "snappy")` when configuring the job. You must also include the `snappy-java` JAR in `-libjars`.

Using Avro Data Files in Pig

CDH provides `AvroStorage` for Avro integration in Pig.

To use it, first register the `piggybank` JAR file and supporting libraries:

```
REGISTER piggybank.jar
REGISTER lib/avro-1.7.6.jar
REGISTER lib/json-simple-1.1.jar
REGISTER lib/snappy-java-1.0.4.1.jar
```

Then load Avro data files as follows:

```
a = LOAD 'my_file.avro' USING org.apache.pig.piggybank.storage.avro.AvroStorage();
```

Pig maps the Avro schema to a corresponding Pig schema.

You can store data in Avro data files with:

```
store b into 'output' USING org.apache.pig.piggybank.storage.avro.AvroStorage();
```

With `store`, Pig generates an Avro schema from the Pig schema. You can override the Avro schema by specifying it literally as a parameter to `AvroStorage` or by using the same schema as an existing Avro data file. See the [Pig wiki](#) for details.

To store two relations in one script, specify an index to each `store` function. For example:

```
set1 = load 'input1.txt' using PigStorage() as ( ... );
store set1 into 'set1' using org.apache.pig.piggybank.storage.avro.AvroStorage('index',
'1');

set2 = load 'input2.txt' using PigStorage() as ( ... );
store set2 into 'set2' using org.apache.pig.piggybank.storage.avro.AvroStorage('index',
'2');
```

For more information, search for "index" in the [AvroStorage wiki](#).

To enable Snappy compression on output files, do the following before issuing the `STORE` statement:

```
SET mapred.output.compress true
SET mapred.output.compression.codec org.apache.hadoop.io.compress.SnappyCodec
SET avro.output.codec snappy
```

For more information, see the [Pig wiki](#). The version numbers of the JAR files to register are different on that page, so adjust them as shown above.

Importing Avro Data Files in Sqoop 1

On the command line, use the following option to import Avro data files:

```
--as-avrodatafile
```

Sqoop 1 automatically generates an Avro schema that corresponds to the database table being exported from.

To enable Snappy compression, add the following option:

```
--compression-codec snappy
```



Note: Sqoop 2 does not currently support Avro.

Using Avro Data Files in Spark

See [Accessing External Storage from Spark](#) and [Accessing Avro Data Files From Spark SQL Applications](#).

Using Avro Data Files in Streaming Programs

To read from Avro data files from a streaming program, specify `org.apache.avro.mapred.AvroAsTextInputFormat` as the input format. This format converts each datum in the Avro data file to a string. For a "bytes" schema, this is the raw bytes; in general cases, this is a single-line [JSON](#) representation.

To write to Avro data files from a streaming program, specify `org.apache.avro.mapred.AvroTextOutputFormat` as the output format. This format creates Avro data files with a "bytes" schema, where each datum is a tab-delimited key-value pair.

At run time, specify the `avro`, `avro-mapred`, and `paranamer` JARs in `-libjars` in the streaming command.

To enable Snappy compression on output files, set the property `avro.output.codec` to `snappy`. You must also include the `snappy-java` JAR in `-libjars`.

Data Compression

Data compression and compression formats can have a significant impact on performance. Three important places to consider data compression are in MapReduce and Spark jobs, data stored in HBase, and Impala queries. For the most part, the principles are similar for each.

You must balance the processing capacity required to compress and uncompress the data, the disk IO required to read and write the data, and the network bandwidth required to send the data across the network. The correct balance of these factors depends upon the characteristics of your cluster and your data, as well as your usage patterns.

Compression is not recommended if your data is already compressed (such as images in JPEG format). In fact, the resulting file can sometimes be larger than the original.

For more information about compression algorithms in Hadoop, see the Compression section of the Hadoop I/O chapter in [Hadoop: The Definitive Guide](#).

Compression Types

Hadoop supports the following compression types and codecs:

- `gzip` - `org.apache.hadoop.io.compress.GzipCodec`
- `bzip2` - `org.apache.hadoop.io.compress.BZip2Codec`
- `LZO` - `com.hadoop.compression.lzo.LzopCodec`
- `Snappy` - `org.apache.hadoop.io.compress.SnappyCodec`
- `Deflate` - `org.apache.hadoop.io.compress.DeflateCodec`

Different file types and CDH components support different compression types. For details, see [Using Apache Avro Data Files with CDH](#) on page 33 and [Using Apache Parquet Data Files with CDH](#) on page 23.

For guidelines on choosing compression types and configuring compression, see [Choosing and Configuring Data Compression](#).

Snappy Compression

[Snappy](#) is a compression/decompression library. It optimizes for very high-speed compression and decompression, and moderate compression instead of maximum compression or compatibility with other compression libraries.

Snappy is supported for all CDH components. How you specify compression depends on the component.

Using Snappy with HBase

If you install Hadoop and HBase from RPM or Debian packages, Snappy requires no HBase configuration.

Using Snappy with Hive

To enable Snappy compression for Hive output when creating `SequenceFile` outputs, use the following settings:

```
SET hive.exec.compress.output=true;
SET mapred.output.compression.codec=org.apache.hadoop.io.compress.SnappyCodec;
SET mapred.output.compression.type=BLOCK;
```

Using Snappy with MapReduce

Enabling MapReduce intermediate compression can make jobs run faster without requiring application changes. Only the temporary intermediate files created by Hadoop for the shuffle phase are compressed; the final output may or may not be compressed. Snappy is ideal in this case because it compresses and decompresses very quickly compared to other compression algorithms, such as Gzip. For information about choosing a compression format, see [Choosing and Configuring Data Compression](#).

To enable Snappy for MapReduce intermediate compression for the whole cluster, set the following properties in `mapred-site.xml`:

- MRv1

```
<property>
  <name>mapred.compress.map.output</name>
  <value>true</value>
</property>
<property>
  <name>mapred.map.output.compression.codec</name>
  <value>org.apache.hadoop.io.compress.SnappyCodec</value>
</property>
```

- YARN

```
<property>
  <name>mapreduce.map.output.compress</name>
  <value>true</value>
</property>
<property>
  <name>mapred.map.output.compress.codec</name>
  <value>org.apache.hadoop.io.compress.SnappyCodec</value>
</property>
```

You can also set these properties on a per-job basis.

Use the properties in the following table to compress the final output of a MapReduce job. These are usually set on a per-job basis.

MRv1 Property	YARN Property	Description
mapred.output.compress	mapreduce.output.fileoutputformat.compress	Whether to compress the final job outputs (true or false).
mapred.output.compression.codec	mapreduce.output.fileoutputformat.compress.codec	If the final job outputs are to be compressed, the codec to use. Set to <code>org.apache.hadoop.io.compress.SnappyCodec</code> for Snappy compression.
mapred.output.compression.type	mapreduce.output.fileoutputformat.compress.type	For SequenceFile outputs, the type of compression to use (NONE, RECORD, or BLOCK). Cloudera recommends BLOCK.



Note: The MRv1 property names are also supported (but deprecated) in YARN. You do not need to update them in this release.

Using Snappy with Pig

Set the same properties for Pig as for MapReduce.

Using Snappy with Spark SQL

To enable Snappy compression for Spark SQL when writing tables, specify the `snappy` codec in the `spark.sql.parquet.compression.codec` configuration:

```
sqlContext.setConf("spark.sql.parquet.compression.codec", "snappy")
```

Using Snappy Compression with Sqoop 1 and Sqoop 2 Imports

- **Sqoop 1** - On the command line, use the following option to enable Snappy compression:

```
--compression-codec org.apache.hadoop.io.compress.SnappyCodec
```

Cloudera recommends using the `--as-sequencefile` option with this compression option.

- **Sqoop 2** - When you create a job (`sqoop:000> create job`), choose 7 (SNAPPY) as the compression format.

External Documentation

Cloudera provides documentation for CDH as a whole, whether your CDH cluster is managed by Cloudera Manager or not. In addition, you may find it useful to refer to documentation for the individual components included in CDH. Where possible, these links point to the main documentation for a project, in the Cloudera release archive. This ensures that you are looking at the correct documentation for the version of a project included in CDH. Otherwise, the links may point to the project's main site.

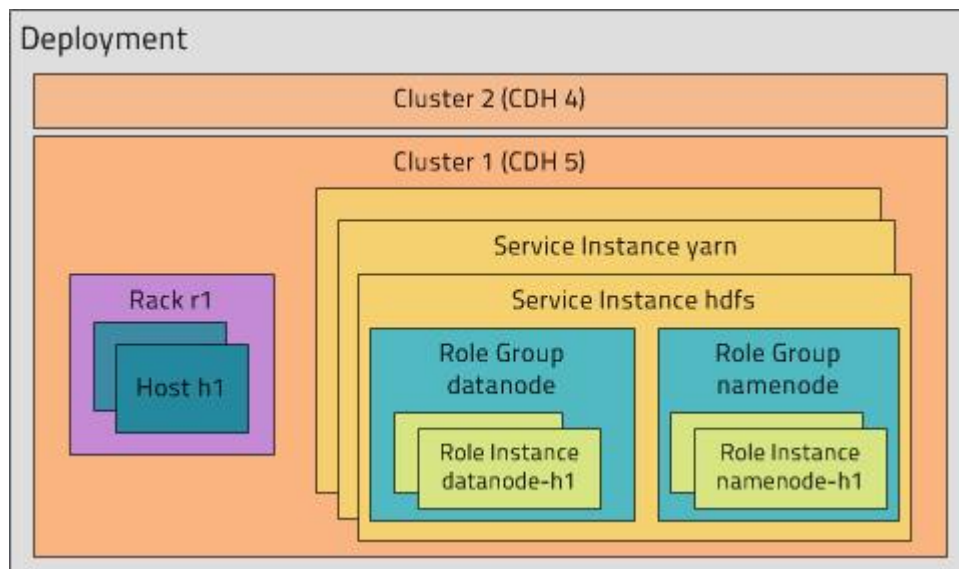
- [Apache Avro](#)
- [Apache Crunch](#)
- [Apache DataFu](#)
- [Apache Flume](#)
- [Apache Hadoop](#)
- [Apache HBase](#)
- [Apache Hive](#)
- [Hue](#)
- [Kite](#)
- [Apache Mahout](#)
- [Apache Oozie](#)
- [Apache Parquet](#)
- [Apache Pig](#)
- [Apache Sentry](#)
- [Apache Solr](#)
- [Apache Spark](#)
- [Apache Sqoop](#)
- [Apache Sqoop2](#)
- [Apache Whirr](#)
- [Apache ZooKeeper](#)

Cloudera Manager 5 Overview

Cloudera Manager is an end-to-end application for managing CDH clusters. Cloudera Manager sets the standard for enterprise deployment by delivering granular visibility into and control over every part of the CDH cluster—empowering operators to improve performance, enhance quality of service, increase compliance and reduce administrative costs. With Cloudera Manager, you can easily deploy and centrally operate the complete CDH stack and other managed services. The application automates the installation process, reducing deployment time from weeks to minutes; gives you a cluster-wide, real-time view of hosts and services running; provides a single, central console to enact configuration changes across your cluster; and incorporates a full range of reporting and diagnostic tools to help you optimize performance and utilization. This primer introduces the basic concepts, structure, and functions of Cloudera Manager.

Terminology

To effectively use Cloudera Manager, you should first understand its terminology. The relationship between the terms is illustrated below and their definitions follow:



Some of the terms, such as cluster and service, will be used without further explanation. Others, such as role group, gateway, host template, and parcel are expanded upon in later sections.

A common point of confusion is the overloading of the terms **service** and **role** for both types and instances; Cloudera Manager and this section sometimes uses the same term for type and instance. For example, the Cloudera Manager Admin Console **Home** > **Status** tab and **Clusters** > **ClusterName** menu lists service instances. This is similar to the practice in programming languages where for example the term "string" may indicate either a type (`java.lang.String`) or an instance of that type ("hi there"). When it's necessary to distinguish between types and instances, the word "type" is appended to indicate a type and the word "instance" is appended to explicitly indicate an instance.

deployment

A configuration of Cloudera Manager and all the clusters it manages.

dynamic resource pool

In Cloudera Manager, a named configuration of resources and a policy for scheduling the resources among YARN applications or Impala queries running in the pool.

cluster

- A set of computers or racks of computers that contains an [HDFS](#) filesystem and runs [MapReduce](#) and other processes on that data. A pseudo-distributed cluster is a [CDH](#) installation run on a single machine and useful for demonstrations and individual study.
- In Cloudera Manager, a logical entity that contains a set of hosts, a single version of CDH installed on the hosts, and the service and role instances running on the hosts. A host can belong to only one cluster. Cloudera Manager can manage multiple CDH clusters, however each cluster can only be associated with a single Cloudera Manager Server or [Cloudera Manager HA pair](#).

host

In Cloudera Manager, a physical or virtual machine that runs role instances. A host can belong to only one cluster.

rack

In Cloudera Manager, a physical entity that contains a set of physical hosts typically served by the same switch.

service

- A Linux command that runs a System V init script in `/etc/init.d/` in as predictable an environment as possible, removing most environment variables and setting the current working directory to `/`.
- A category of managed functionality in Cloudera Manager, which may be distributed or not, running in a cluster. Sometimes referred to as a service type. For example: MapReduce, HDFS, YARN, Spark, and Accumulo. In traditional environments, multiple services run on one host; in distributed systems, a service runs on many hosts.

service instance

In Cloudera Manager, an instance of a service running on a cluster. For example: "HDFS-1" and "yarn". A service instance spans many role instances.

role

In Cloudera Manager, a category of functionality within a service. For example, the HDFS service has the following roles: NameNode, SecondaryNameNode, DataNode, and Balancer. Sometimes referred to as a role type. See also [user role](#).

role instance

In Cloudera Manager, an instance of a role running on a host. It typically maps to a Unix process. For example: "NameNode-h1" and "DataNode-h1".

role group

In Cloudera Manager, a set of configuration properties for a set of role instances.

host template

A set of role groups in Cloudera Manager. When a template is applied to a host, a role instance from each role group is created and assigned to that host.

gateway

In Cloudera Manager, role that designates a host that should receive a client configuration for a service when the host does not have any role instances for that service running on it.

parcel


A binary distribution format that contains compiled code and meta-information such as a package description, version, and dependencies.

static service pool




















In Cloudera Manager, a static partitioning of total cluster resources—CPU, memory, and I/O weight—across a set of services.

Cluster Example

Consider a cluster **Cluster 1** with four hosts as shown in the following listing from Cloudera Manager:

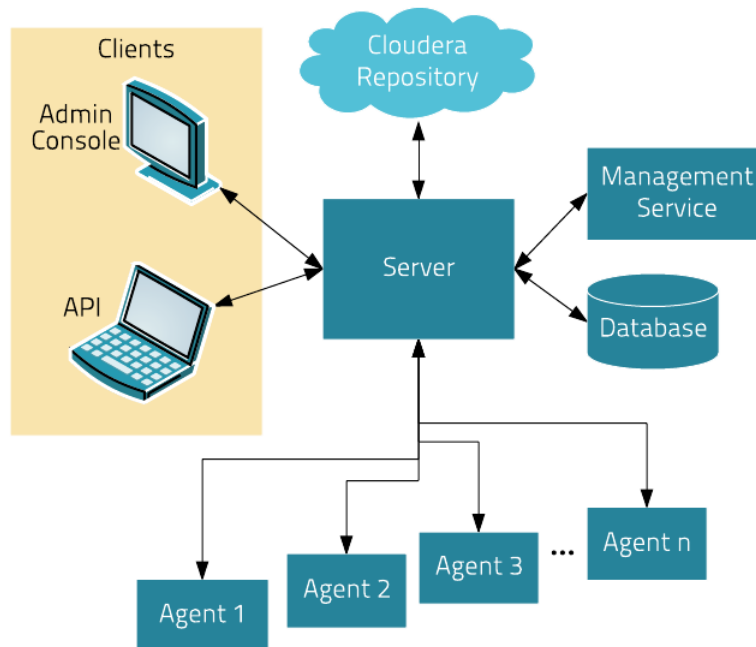
<input type="checkbox"/>	⬆ ⬆ Name	⬆ IP	⬆ Roles	⬆ Load Average	⬆ Disk Usage	⬆ Physical Memory	⬆ Swap Space
<input type="checkbox"/>	 tcdn501-1.ent.cloudera.com	10.20.195.240	➤ 21 Role(s)	0.04 0.15 0.26	<div><div></div>11.3 GiB / 57 GiB</div>	<div><div></div>6.3 GiB / 9.7 GiB</div>	<div><div></div>4.7 MiB / 2 GiB</div>
<input type="checkbox"/>	 tcdn501-2.ent.cloudera.com	10.20.81.81	➤ 7 Role(s)	0.07 0.07 0.05	<div><div></div>8.9 GiB / 57 GiB</div>	<div><div></div>2 GiB / 9.7 GiB</div>	<div><div></div>0 B / 2 GiB</div>
<input type="checkbox"/>	 tcdn501-3.ent.cloudera.com	10.20.190.234	➤ 7 Role(s)	0.08 0.11 0.04	<div><div></div>8.9 GiB / 57 GiB</div>	<div><div></div>2 GiB / 9.7 GiB</div>	<div><div></div>0 B / 2 GiB</div>
<input type="checkbox"/>	 tcdn501-4.ent.cloudera.com	10.20.195.243	➤ 7 Role(s)	0.06 0.23 0.23	<div><div></div>8.9 GiB / 57 GiB</div>	<div><div></div>2 GiB / 9.7 GiB</div>	<div><div></div>0 B / 2 GiB</div>
<div><div>First</div><div>Previous</div><div>1</div><div>Next</div><div>Last</div></div>							

The host **tcdn501-1** is the "master" host for the cluster, so it has many more role instances, 21, compared with the 7 role instances running on the other hosts. In addition to the CDH "master" role instances, **tcdn501-1** also has Cloudera Management Service roles:

Service	Instance	Name
None	None	deploy-client-config
 HBase	Master	hbase-MASTER
 HDFS	NameNode	hdfs-NAMENODE
 HDFS	SecondaryNameNode	hdfs-SECONDARYNAMENODE
 Hive	Hive Metastore Server	hive-HIVEMETASTORE
 Hive	HiveServer2	hive-HIVESERVER2
 Hue	Hue Server	hue-HUE_SERVER
 Impala	Impala Catalog Server	impala-CATALOGSERVER
 Impala	Impala StateStore	impala-STATESTORE
 Cloudera Management Service	Alert Publisher	cloudera-mgmt-ALERTPUBLISHER
 Cloudera Management Service	Event Server	cloudera-mgmt-EVENTSERVER
 Cloudera Management Service	Host Monitor	cloudera-mgmt-HOSTMONITOR
 Cloudera Management Service	Navigator Audit Server	cloudera-mgmt-NAVIGATOR
 Cloudera Management Service	Navigator Metadata Server	cloudera-mgmt-NAVIGATORMETASERVER
 Cloudera Management Service	Reports Manager	cloudera-mgmt-REPORTSMANAGER
 Cloudera Management Service	Service Monitor	cloudera-mgmt-SERVICEMONITOR
 Oozie	Oozie Server	oozie-OOZIE_SERVER
 Spark	Master	spark-SPARK_MASTER
 YARN (MR2 Included)	JobHistory Server	yarn-JOBHISTORY
 YARN (MR2 Included)	ResourceManager	yarn-RESOURCEMANAGER

Architecture

As depicted below, the heart of Cloudera Manager is the Cloudera Manager Server. The Server hosts the Admin Console Web Server 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:
 - **Admin Console** - Web-based UI with which administrators manage clusters and Cloudera Manager.
 - **API** - API with which developers create custom Cloudera Manager applications.

Heartbeating

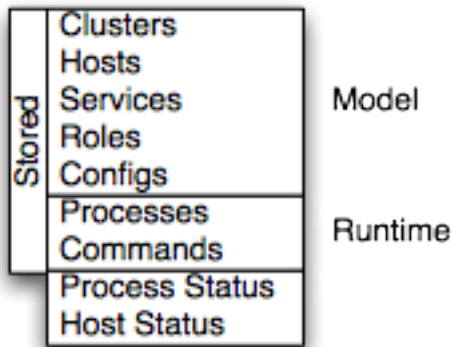
Heartbeats are a primary communication mechanism in Cloudera Manager. By default Agents send heartbeats every 15 seconds to the Cloudera Manager Server. However, to reduce user latency the frequency is increased when state is changing.

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.

State Management

The Cloudera Manager Server maintains the state of the cluster. This state can be divided into two categories: "model" and "runtime", both of which are stored in the Cloudera Manager Server database.

State Maintained by CM Server



Cloudera Manager models CDH and managed services: their roles, configurations, and inter-dependencies. *Model state* captures what is supposed to run where, and with what configurations. For example, model state captures the fact that a cluster contains 17 hosts, each of which is supposed to run a DataNode. You interact with the model through the Cloudera Manager Admin Console configuration screens and API and operations such as "Add Service".

Runtime state is what processes are running where, and what commands (for example, rebalance HDFS or run a Backup/Disaster Recovery schedule or rolling restart or stop) are currently running. The runtime state includes the exact configuration files needed to run a process. When you select Start in the Cloudera Manager Admin Console, the server gathers up all the configuration for the relevant services and roles, validates it, generates the configuration files, and stores them in the database.

When you update a configuration (for example, the Hue Server web port), you have updated the model state. However, if Hue is running while you do this, it is still using the old port. When this kind of mismatch occurs, the role is marked as having an "outdated configuration". To resynchronize, you restart the role (which triggers the configuration re-generation and process restart).

While Cloudera Manager models all of the reasonable configurations, some cases inevitably require special handling. To allow you to work around, for example, a bug or to explore unsupported options, Cloudera Manager supports an "[advanced configuration snippet](#)" mechanism that lets you add properties directly to the configuration files.

Configuration Management








Cloudera Manager defines configuration at several levels:

- The service level may define configurations that apply to the entire service instance, such as an HDFS service's default replication factor (`dfs.replication`).
- The [role group](#) level may define configurations that apply to the member roles, such as the DataNodes' handler count (`dfs.datanode.handler.count`). This can be set differently for different groups of DataNodes. For example, DataNodes running on more capable hardware may have more handlers.
- The role instance level may override configurations that it inherits from its role group. This should be used sparingly, because it easily leads to configuration divergence within the role group. One example usage is to temporarily enable debug logging in a specific role instance to troubleshoot an issue.
- Hosts have configurations related to monitoring, software management, and resource management.
- Cloudera Manager itself has configurations related to its own administrative operations.

Role Groups

You can set configuration at the service instance (for example, HDFS) or role instance (for example, the DataNode on host17). An individual role inherits the configurations set at the service level. Configurations made at the role level override those inherited from the service level. While this approach offers flexibility, configuring a set of role instances in the same way can be tedious.

Cloudera Manager supports role groups, a mechanism for assigning configurations to a group of role instances. The members of those groups then inherit those configurations. For example, in a cluster with heterogeneous hardware, a DataNode role group can be created for each host type and the DataNodes running on those hosts can be assigned to their corresponding role group. That makes it possible to set the configuration for all the DataNodes running on the same hardware by modifying the configuration of one role group. The HDFS service discussed earlier has the following role groups defined for the service's roles:

<input type="checkbox"/>  Role Name	State	Host	Role Group
<input type="checkbox"/>  Balancer	N/A	tcdn501-1.ent.cloudera.com	Balancer Default Group
<input type="checkbox"/>  DataNode	Started	tcdn501-2.ent.cloudera.com	DataNode Default Group
<input type="checkbox"/>  DataNode	Started	tcdn501-3.ent.cloudera.com	DataNode Default Group
<input type="checkbox"/>  DataNode	Started	tcdn501-4.ent.cloudera.com	DataNode Default Group
<input type="checkbox"/>  NameNode (Active)	Started	tcdn501-1.ent.cloudera.com	NameNode Default Group
<input type="checkbox"/>  SecondaryNameNode	Started	tcdn501-1.ent.cloudera.com	SecondaryNameNode Default Group

In addition to making it easy to manage the configuration of subsets of roles, role groups also make it possible to maintain different configurations for experimentation or managing shared clusters for different users or workloads.

Host Templates

In typical environments, sets of hosts have the same hardware and the same set of services running on them. A host template defines a set of role groups (at most one of each type) in a cluster and provides two main benefits:

- Adding new hosts to clusters easily - multiple hosts can have roles from different services created, configured, and started in a single operation.
- Altering the configuration of roles from different services on a set of hosts easily - which is useful for quickly switching the configuration of an entire cluster to accommodate different workloads or users.

Server and Client Configuration

Administrators are sometimes surprised that modifying `/etc/hadoop/conf` and then restarting HDFS has no effect. That is because service instances started by Cloudera Manager do not read configurations from the default locations. To use HDFS as an example, when not managed by Cloudera Manager, there would usually be one HDFS configuration per host, located at `/etc/hadoop/conf/hdfs-site.xml`. Server-side daemons and clients running on the same host would all use that same configuration.

Cloudera Manager distinguishes between server and client configuration. In the case of HDFS, the file `/etc/hadoop/conf/hdfs-site.xml` contains only configuration relevant to an HDFS client. That is, by default, if you run a program that needs to communicate with Hadoop, it will get the addresses of the NameNode and JobTracker, and other important configurations, from that directory. A similar approach is taken for `/etc/hbase/conf` and `/etc/hive/conf`.

In contrast, the HDFS role instances (for example, NameNode and DataNode) obtain their configurations from a private per-process directory, under `/var/run/cloudera-scm-agent/process/unique-process-name`. Giving each process its own private execution and configuration environment allows Cloudera Manager to control each process independently. For example, here are the contents of an example `879-hdfs-NAMENODE` process directory:

```
$ tree -a /var/run/cloudera-scm-agent/process/879-hdfs-NAMENODE /
/var/run/cloudera-scm-agent/process/879-hdfs-NAMENODE/
  cloudera_manager_agent_fencer.py
  cloudera_manager_agent_fencer_secret_key.txt
  cloudera-monitor.properties
  core-site.xml
  dfs_hosts_allow.txt
  dfs_hosts_exclude.txt
  event-filter-rules.json
  hadoop-metrics2.properties
```

```

hdfs.keytab
hdfs-site.xml
log4j.properties
logs
  stderr.log
  stdout.log
topology.map
topology.py

```

Distinguishing between server and client configuration provides several advantages:

- Sensitive information in the server-side configuration, such as the password for the Hive Metastore RDBMS, is not exposed to the clients.
- A service that depends on another service may deploy with customized configuration. For example, to get good HDFS read performance, Impala needs a specialized version of the HDFS client configuration, which may be harmful to a generic client. This is achieved by separating the HDFS configuration for the Impala daemons (stored in the per-process directory mentioned above) from that of the generic client (`/etc/hadoop/conf`).
- Client configuration files are much smaller and more readable. This also avoids confusing non-administrator Hadoop users with irrelevant server-side properties.

Deploying Client Configurations and Gateways

A client configuration is a zip file that contains the relevant configuration files with the settings for a service. Each zip file contains the set of configuration files needed by the service. For example, the MapReduce client configuration zip file contains copies of `core-site.xml`, `hadoop-env.sh`, `hdfs-site.xml`, `log4j.properties`, and `mapred-site.xml`. Cloudera Manager supports a **Download Client Configuration** action to enable distributing the client configuration file to users outside the cluster.

Cloudera Manager can deploy client configurations within the cluster; each applicable service has a **Deploy Client Configuration** action. This action does not necessarily deploy the client configuration to the entire cluster; it only deploys the client configuration to all the hosts that this service has been assigned to. For example, suppose a cluster has 10 hosts, and a MapReduce service is running on hosts 1-9. When you use Cloudera Manager to deploy the MapReduce client configuration, host 10 will not get a client configuration, because the MapReduce service has no role assigned to it. This design is intentional to avoid deploying conflicting client configurations from multiple services.

To deploy a client configuration to a host that does not have a role assigned to it you use a gateway. A **gateway** is a marker to convey that a service should be accessible from a particular host. Unlike all other roles it has no associated process. In the preceding example, to deploy the MapReduce client configuration to host 10, you assign a MapReduce gateway role to that host.

Gateways can also be used to customize client configurations for some hosts. Gateways can be placed in role groups and those groups can be configured differently. However, unlike role instances, there is no way to override configurations for gateway instances.

In the cluster we discussed earlier, the three hosts (**tcdn501-[2-5]**) that do not have Hive role instances have Hive gateways:

<input type="checkbox"/>	↑ Role Name	↑ State	↑ Host	↑ Role Group
<input type="checkbox"/>	Gateway	N/A	tcdn501-2.ent.cloudera.com	Gateway Default Group
<input type="checkbox"/>	Gateway	N/A	tcdn501-3.ent.cloudera.com	Gateway Default Group
<input type="checkbox"/>	Gateway	N/A	tcdn501-4.ent.cloudera.com	Gateway Default Group
<input type="checkbox"/>	Gateway	N/A	tcdn501-1.ent.cloudera.com	Gateway Default Group
<input type="checkbox"/>	Hive Metastore Server	Started	tcdn501-1.ent.cloudera.com	Hive Metastore Server Default Group
<input type="checkbox"/>	HiveServer2	Started	tcdn501-1.ent.cloudera.com	HiveServer2 Default Group

Process Management

In a non-Cloudera Manager managed cluster, you most likely start a role instance process using an init script, for example, `service hadoop-hdfs-datanode start`. Cloudera Manager does not use `init` scripts for the daemons it manages; in a Cloudera Manager managed cluster, starting and stopping services using `init` scripts will not work.

In a Cloudera Manager managed cluster you can only start or stop role instance processes using Cloudera Manager. Cloudera Manager uses an open source process management tool called `supervisord`, that starts processes, takes care of redirecting log files, notifying of process failure, setting the effective user ID of the calling process to the right user, and so on. Cloudera Manager supports automatically restarting a crashed process. It will also flag a role instance with a bad health flag if its process crashes repeatedly right after start up.

Stopping the Cloudera Manager Server and the Cloudera Manager Agents will not bring down your services; any running role instances keep running.

The Agent is started by `init.d` at start-up. It, in turn, contacts the Cloudera Manager Server and determines which processes should be running. The Agent is monitored as part of Cloudera Manager's host monitoring: if the Agent stops heartbeating, the host is marked as having bad health.

One of the Agent's main responsibilities is to start and stop processes. When the Agent detects a new process from the Server heartbeat, the Agent creates a directory for it in `/var/run/cloudera-scm-agent` and unpacks the configuration. It then contacts `supervisord`, which starts the process.

These actions reinforce an important point: a Cloudera Manager process never travels alone. In other words, a process is more than just the arguments to `exec()` — it also includes configuration files, directories that need to be created, and other information.

Software Distribution Management

A major function of Cloudera Manager is to install CDH and managed service software. Cloudera Manager installs software for new deployments and to upgrade existing deployments. Cloudera Manager supports two software distribution formats: packages and parcels.

A **package** is a binary distribution format that contains compiled code and meta-information such as a package description, version, and dependencies. Package management systems evaluate this meta-information to allow package searches, perform upgrades to a newer version, and ensure that all dependencies of a package are fulfilled. Cloudera Manager uses the native system package manager for each supported OS.

A **parcel** is a binary distribution format containing the program files, along with additional metadata used by Cloudera Manager. The important differences between parcels and packages are:

- Parcels are self-contained and installed in a versioned directory, which means that multiple versions of a given parcel can be installed side-by-side. You can then designate one of these installed versions as the active one. With packages, only one package can be installed at a time so there is no distinction between what is installed and what is active.
- You can install parcels at any location in the filesystem. They are installed by default in `/opt/cloudera/parcels`. In contrast, packages are installed in `/usr/lib`.
- When you install from the Parcels page, Cloudera Manager automatically downloads, distributes, and activates the correct parcel for the operating system running on each host in the cluster. All CDH hosts that make up a logical cluster need to run on the same major OS release to be covered by Cloudera Support. Cloudera Manager needs to run on the same OS release as one of the CDH clusters it manages, to be covered by Cloudera Support. The risk of issues caused by running different minor OS releases is considered lower than the risk of running different major OS releases. Cloudera recommends running the same minor release cross-cluster, because it simplifies issue tracking and supportability. You can, however, use RHEL/Centos 7.2 as the operating system for gateway hosts. See [Operating System Support for Gateway Hosts \(CDH 5.11 and higher only\)](#).

Because of their unique properties, parcels offer the following advantages over packages:

- **Distribution of CDH as a single object** - Instead of having a separate package for each part of CDH, parcels have just a single object to install. This makes it easier to distribute software to a cluster that is not connected to the Internet.
- **Internal consistency** - All CDH components are matched, eliminating the possibility of installing parts from different versions of CDH.
- **Installation outside of `/usr`** - In some environments, Hadoop administrators do not have privileges to install system packages. These administrators needed to use CDH tarballs, which do not provide the infrastructure that packages do. With parcels, administrators can install to `/opt`, or anywhere else, without completing the additional manual steps of regular tarballs.



Note: With parcels, the path to the CDH libraries is `/opt/cloudera/parcels/CDH/lib` instead of the usual `/usr/lib`. Do not link `/usr/lib/` elements to parcel-deployed paths, because the links may cause scripts that distinguish between the two paths to not work.

- **Installation of CDH without `sudo`** - Parcel installation is handled by the Cloudera Manager Agent running as root or another user, so you can install CDH without `sudo`.
- **Decoupled distribution from activation** - With side-by-side install capabilities, you can stage a new version of CDH across the cluster before switching to it. This allows the most time-consuming part of an upgrade to be done ahead of time without affecting cluster operations, thereby reducing downtime.
- **Rolling upgrades** - Packages require you to shut down the old process, upgrade the package, and then start the new process. Any errors in the process can be difficult to recover from, and upgrading requires extensive integration with the package management system to function seamlessly. With parcels, when a new version is staged side-by-side, you can switch to a new minor version by simply changing which version of CDH is used when restarting each process. You can then perform upgrades with [rolling restarts](#), in which service roles are restarted in the correct order to switch to the new version with minimal service interruption. Your cluster can continue to run on the existing installed components while you stage a new version across your cluster, without impacting your current operations. Major version upgrades (for example, CDH 4 to CDH 5) require full service restarts because of substantial changes between the versions. Finally, you can upgrade individual parcels or multiple parcels at the same time.
- **Upgrade management** - Cloudera Manager manages all the steps in a CDH version upgrade. With packages, Cloudera Manager only helps with initial installation.
- **Additional components** - Parcels are not limited to CDH. Cloudera Impala, Cloudera Search, LZO, Apache Kafka, and [add-on service](#) parcels are also available.
- **Compatibility with other distribution tools** - Cloudera Manager works with other tools you use for download and distribution. For example, you can use Puppet. Or, you can download the parcel to Cloudera Manager Server manually if your cluster has no Internet connectivity and then have Cloudera Manager distribute the parcel to the cluster.

Host Management

Cloudera Manager provides several features to manage the hosts in your Hadoop clusters. The first time you run Cloudera Manager Admin Console you can search for hosts to add to the cluster and once the hosts are selected you can map the assignment of CDH roles to hosts. Cloudera Manager automatically deploys all software required to participate as a managed host in a cluster: JDK, Cloudera Manager Agent, CDH, Impala, Solr, and so on to the hosts.

Once the services are deployed and running, the Hosts area within the Admin Console shows the overall status of the managed hosts in your cluster. The information provided includes the version of CDH running on the host, the cluster to which the host belongs, and the number of roles running on the host. Cloudera Manager provides operations to manage the lifecycle of the participating hosts and to add and delete hosts. The Cloudera Management Service Host Monitor role performs health tests and collects host metrics to allow you to monitor the health and performance of the hosts.

Resource Management

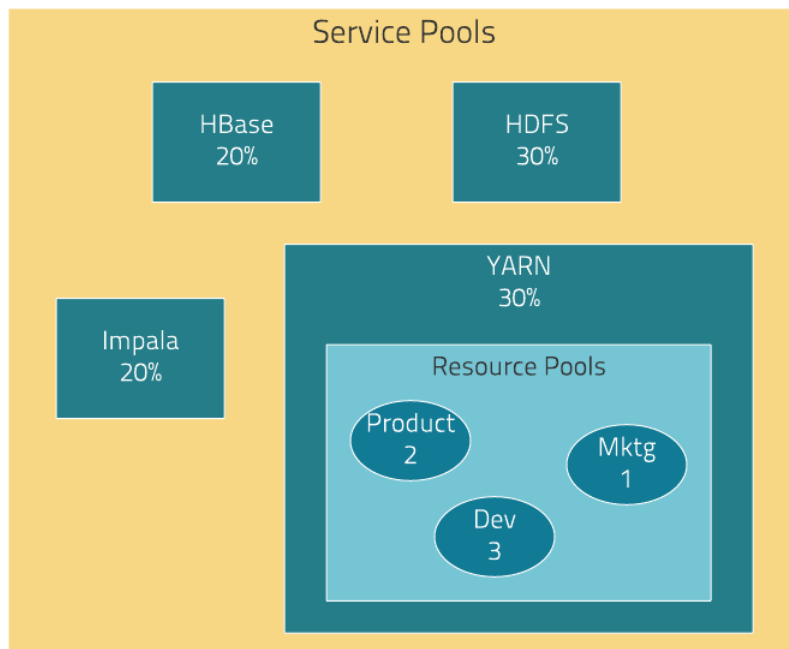
Resource management helps ensure predictable behavior by defining the impact of different services on cluster resources. Use resource management to:

- Guarantee completion in a reasonable time frame for critical workloads.
- Support reasonable cluster scheduling between groups of users based on fair allocation of resources per group.
- Prevent users from depriving other users access to the cluster.

With Cloudera Manager 5, statically allocating resources using cgroups is configurable through a single *static service pool wizard*. You allocate services as a percentage of total resources, and the wizard configures the cgroups.

Static service pools isolate the services in your cluster from one another, so that load on one service has a bounded impact on other services. Services are allocated a static percentage of total resources—CPU, memory, and I/O weight—which are not shared with other services. When you configure static service pools, Cloudera Manager computes recommended memory, CPU, and I/O configurations for the worker roles of the services that correspond to the percentage assigned to each service. Static service pools are implemented per role group within a cluster, using [Linux control groups \(cgroups\)](#) and cooperative memory limits (for example, Java maximum heap sizes). Static service pools can be used to control access to resources by HBase, HDFS, Impala, MapReduce, Solr, Spark, YARN, and [add-on](#) services. Static service pools are not enabled by default.

For example, the following figure illustrates static pools for HBase, HDFS, Impala, and YARN services that are respectively assigned 20%, 30%, 20%, and 30% of cluster resources.



You can dynamically apportion resources that are statically allocated to YARN and Impala by using *dynamic resource pools*.

Depending on the version of CDH you are using, dynamic resource pools in Cloudera Manager support the following scenarios:

- **YARN (CDH 5)** - YARN manages the virtual cores, memory, running applications, maximum resources for undeclared children (for parent pools), and scheduling policy for each pool. In the preceding diagram, three dynamic resource pools—Dev, Product, and Mktg with weights 3, 2, and 1 respectively—are defined for YARN. If an application starts and is assigned to the Product pool, and other applications are using the Dev and Mktg pools, the Product resource

pool receives $30\% \times 2/6$ (or 10%) of the total cluster resources. If no applications are using the Dev and Mktg pools, the YARN Product pool is allocated 30% of the cluster resources.

- **Impala (CDH 5 and CDH 4)** - Impala manages memory for pools running queries and limits the number of running and queued queries in each pool.

User Management

Access to Cloudera Manager features is controlled by user accounts. A user account identifies how a user is authenticated and determines what privileges are granted to the user.

Cloudera Manager provides several mechanisms for authenticating users. You can configure Cloudera Manager to authenticate users against the Cloudera Manager database or against an [external authentication service](#). The external authentication service can be an LDAP server (Active Directory or an OpenLDAP compatible directory), or you can specify another external service. Cloudera Manager also supports using the Security Assertion Markup Language (SAML) to enable single sign-on.

For information about the privileges associated with each of the Cloudera Manager user roles, see [Cloudera Manager User Roles](#).

Security Management

Cloudera Manager strives to consolidate security configurations across several projects.

Authentication

Authentication is a process that requires users and services to prove their identity when trying to access a system resource. Organizations typically manage user identity and authentication through various time-tested technologies, including Lightweight Directory Access Protocol (LDAP) for identity, directory, and other services, such as group management, and Kerberos for authentication.

Cloudera clusters support integration with both of these technologies. For example, organizations with existing LDAP directory services such as Active Directory (included in Microsoft Windows Server as part of its suite of Active Directory Services) can leverage the organization's existing user accounts and group listings instead of creating new accounts throughout the cluster. Using an external system such as Active Directory or OpenLDAP is required to support the user role authorization mechanism implemented in Cloudera Navigator.

For authentication, Cloudera supports integration with MIT Kerberos and with Active Directory, which includes Kerberos implementation for authentication. Kerberos provides **strong authentication**, *strong* meaning that cryptographic mechanisms—rather than passwords alone—are used in the exchange between requesting process and service during the authentication process.

These systems are not mutually exclusive. For example, Microsoft Active Directory is an LDAP directory service that also provides Kerberos authentication services, and Kerberos credentials can be stored and managed in an LDAP directory service. Cloudera Manager Server, CDH nodes, and Cloudera Enterprise components, such as Cloudera Navigator, Apache Hive, Hue, and Impala, can all make use of Kerberos authentication.

Authorization

Authorization is concerned with who or what has access or control over a given resource or service. Since Hadoop merges together the capabilities of multiple varied, and previously separate IT systems as an enterprise data hub that stores and works on all data within an organization, it requires multiple authorization controls with varying granularities. In such cases, Hadoop management tools simplify setup and maintenance by:

- Tying all users to groups, which can be specified in existing LDAP or AD directories.
- Providing role-based access control for similar interaction methods, like batch and interactive SQL queries. For example, Apache Sentry permissions apply to Hive (HiveServer2) and Impala.

CDH currently provides the following forms of access control:

- Traditional POSIX-style permissions for directories and files, where each directory and file is assigned a single owner and group. Each assignment has a basic set of permissions available; file permissions are simply read, write, and execute, and directories have an additional permission to determine access to child directories.
- [Extended Access Control Lists](#) (ACLs) for HDFS that provide fine-grained control of permissions for HDFS files by allowing you to set different permissions for specific named users or named groups.
- Apache HBase uses ACLs to authorize various operations (`READ`, `WRITE`, `CREATE`, `ADMIN`) by column, column family, and column family qualifier. HBase ACLs are granted and revoked to both users and groups.
- Role-based access control with [Apache Sentry](#).

Encryption

Data at rest and data in transit encryption function at different technology layers of the cluster:

Cloudera Management Service

The Cloudera Management Service implements various management features as a set of roles:




- Activity Monitor - collects information about activities run by the MapReduce service. This role is not added by default.
- Host Monitor - collects health and metric information about hosts
- Service Monitor - collects health and metric information about services and activity information from the YARN and Impala services
- Event Server - aggregates relevant Hadoop events and makes them available for alerting and searching
- Alert Publisher - generates and delivers alerts for certain types of events
- Reports Manager - generates reports that provide an historical view into disk utilization by user, user group, and directory, processing activities by user and YARN pool, and HBase tables and namespaces. This role is not added in Cloudera Express.

In addition, for certain editions of the Cloudera Enterprise license, the Cloudera Management Service provides the [Navigator Audit Server](#) and [Navigator Metadata Server](#) roles for [Cloudera Navigator](#).

Health Tests

Cloudera Manager monitors the health of the services, roles, and hosts that are running in your clusters using **health tests**. The Cloudera Management Service also provides health tests for its roles. Role-based health tests are enabled by default. For example, a simple health test is whether there's enough disk space in every NameNode data directory. A more complicated health test may evaluate when the last checkpoint for HDFS was compared to a threshold or whether a DataNode is connected to a NameNode. Some of these health tests also aggregate other health tests: in a distributed system like HDFS, it's normal to have a few DataNodes down (assuming you've got dozens of hosts), so we allow for setting thresholds on what percentage of hosts should color the entire service down.

Health tests can return one of three values: **Good**, **Concerning**, and **Bad**. A test returns **Concerning** health if the test falls below a warning threshold. A test returns **Bad** if the test falls below a critical threshold. The overall health of a service or role instance is a roll-up of its health tests. If any health test is **Concerning** (but none are **Bad**) the role's or service's health is **Concerning**; if any health test is **Bad**, the service's or role's health is **Bad**.

In the Cloudera Manager Admin Console, health tests results are indicated with colors: **Good** , **Concerning** , and **Bad** .

One common question is whether monitoring can be separated from configuration. One of the goals for monitoring is to enable it without needing to do additional configuration and installing additional tools (for example, Nagios). By having a deep model of the configuration, Cloudera Manager is able to know which directories to monitor, which ports to use, and what credentials to use for those ports. This tight coupling means that, when you install Cloudera Manager all the monitoring is enabled.

Metric Collection and Display

To perform monitoring, the Service Monitor and Host Monitor collect metrics. A **metric** is a numeric value, associated with a name (for example, "CPU seconds"), an entity it applies to ("host17"), and a timestamp. Most metric collection is performed by the Agent. The Agent communicates with a supervised process, requests the metrics, and forwards them to the Service Monitor. In most cases, this is done once per minute.


A few special metrics are collected by the Service Monitor. For example, the Service Monitor hosts an HDFS canary, which tries to write, read, and delete a file from HDFS at regular intervals, and measure whether it succeeded, and how long it took. Once metrics are received, they're aggregated and stored.

Using the Charts page in the Cloudera Manager Admin Console, you can query and explore the metrics being collected. Charts display **time series**, which are streams of metric data points for a specific entity. Each metric data point contains a timestamp and the value of that metric at that timestamp.

Some metrics (for example, `total_cpu_seconds`) are counters, and the appropriate way to query them is to take their rate over time, which is why a lot of metrics queries contain the `dt0` function. For example, `dt0(total_cpu_seconds)`. (The `dt0` syntax is intended to remind you of derivatives. The 0 indicates that the rate of a monotonically increasing counter should never have negative rates.)

Events, Alerts, and Triggers

An **event** is a record that something of interest has occurred – a service's health has changed state, a log message (of the appropriate severity) has been logged, and so on. Many events are enabled and configured by default.

An **alert** is an event that is considered especially noteworthy and is triggered by a selected event. Alerts are shown with an  badge when they appear in a list of [events](#). You can configure the Alert Publisher to send alert notifications by email or by SNMP trap to a trap receiver.

A **trigger** is a statement that specifies an action to be taken when one or more specified conditions are met for a service, role, role configuration group, or host. The conditions are expressed as a [tsquery statement](#), and the action to be taken is to change the health for the service, role, role configuration group, or host to either Concerning (yellow) or Bad (red).

Cloudera Manager Admin Console

Cloudera Manager Admin Console is the web-based UI that you use to configure, manage, and monitor CDH.

If no services are configured when you log into the Cloudera Manager Admin Console, the Cloudera Manager installation wizard displays. If services have been configured, the Cloudera Manager top navigation bar:





and [Home](#) page display. The Cloudera Manager Admin Console top navigation bar provides the following tabs and menus:

- **Clusters** > *cluster_name*
 - **Services** - Display individual services, and the Cloudera Management Service. In these pages you can:
 - View the status and other details of a service instance or the role instances associated with the service
 - Make configuration changes to a service instance, a role, or a specific role instance
 - Add and delete a service or role
 - Stop, start, or restart a service or role.
 - View the commands that have been run for a service or a role
 - View an audit event history
 - Deploy and download client configurations
 - Decommission and recommission role instances

- Enter or exit maintenance mode
 - Perform actions unique to a specific type of service. For example:
 - Enable HDFS high availability or NameNode federation
 - Run the HDFS Balancer
 - Create HBase, Hive, and Sqoop directories
 - **Cloudera Manager Management Service** - Manage and monitor the Cloudera Manager Management Service. This includes the following roles: Activity Monitor, Alert Publisher, Event Server, Host Monitor, Navigator Audit Server, Navigator Metadata Server, Reports Manager, and Service Monitor.
 - **Cloudera Navigator** - Opens the Cloudera Navigator user interface.
 - **Hosts** - Displays the hosts in the cluster.
 - **Reports** - Create reports about the HDFS, MapReduce, YARN, and Impala usage and browse HDFS files, and manage quotas for HDFS directories.
 - **Utilization Report** - Opens the **Cluster Utilization Report**. displays aggregated utilization information for YARN and Impala jobs.
 - **MapReduce_service_name Jobs** - Query information about MapReduce jobs running on your cluster.
 - **YARN_service_name Applications** - Query information about YARN applications running on your cluster.
 - **Impala_service_name Queries** - Query information about Impala queries running on your cluster.
 - **Dynamic Resource Pools** - Manage dynamic allocation of cluster resources to YARN and Impala services by specifying the relative weights of named pools.
 - **Static Service Pools** - Manage static allocation of cluster resources to HBase, HDFS, Impala, MapReduce, and YARN services.
- **Hosts** - Display the hosts managed by Cloudera Manager.
 - **All Hosts** - Displays a list of manage hosts in the cluster.
 - **Roles** - Displays the roles deployed on each host.
 - **Host Templates** - Create and manage **Host Templates**, which define sets of role groups that can be used to easily expand a cluster.
 - **Disks Overview** - Displays the status of all disks in the cluster.
 - **Parcels** - Displays parcels available in the cluster and allows you to download, distribute, and activate new parcels.

In this page you can:

- View the status and a variety of detail metrics about individual hosts
 - Make configuration changes for host monitoring
 - View all the processes running on a host
 - Run the Host Inspector
 - Add and delete hosts
 - Create and manage host templates
 - Manage parcels
 - Decommission and recommission hosts
 - Make rack assignments
 - Run the host upgrade wizard
- **Diagnostics** - Review logs, events, and alerts to diagnose problems. The subpages are:
 - **Events** - Search for and displaying events and alerts that have occurred.
 - **Logs** - Search logs by service, role, host, and search phrase as well as log level (severity).
 - **Server Log** - Display the Cloudera Manager Server log.
 - **Audits** - Query and filter audit events across clusters, including logins, across clusters.
 - **Charts** - Query for metrics of interest, display them as charts, and display personalized chart dashboards.
 - **Backup** - Manage replication schedules and snapshot policies.

- **Administration** - Administer Cloudera Manager. The subpages are:
 - **Settings** - Configure Cloudera Manager.
 - **Alerts** - Display when alerts will be generated, configure alert recipients, and send test alert email.
 - **Users** - Manage Cloudera Manager users and user sessions.
 - **Security** - Generate Kerberos credentials and inspect hosts.
 - **License** - Manage Cloudera licenses.
 - **Language** - Set the language used for the content of activity events, health events, and alert email messages.
 - **AWS Credentials** - Configure S3 connectivity to Cloudera Manager.
-  **Parcel Icon** - link to the **Hosts > Parcels** page.
- **Running Commands Indicator**  displays the number of commands currently running for all services or roles.
- **Search** - Supports searching for services, roles, hosts, configuration properties, and commands. You can enter a partial string and a drop-down list with up to sixteen entities that match will display.
- **Support** - Displays various support actions. The subcommands are:
 - **Send Diagnostic Data** - Sends data to Cloudera Support to support troubleshooting.
 - **Support Portal (Cloudera Enterprise)** - Displays the Cloudera Support portal.
 - **Mailing List (Cloudera Express)** - Displays the Cloudera Manager Users list.
 - **Scheduled Diagnostics: Weekly** - Configure the frequency of automatically collecting diagnostic data and sending to Cloudera support.
 - The following links open the latest documentation on the Cloudera web site:
 - **Help**
 - **Installation Guide**
 - **API Documentation**
 - **Release Notes**
 - **About** - Version number and build details of Cloudera Manager and the current date and time stamp of the Cloudera Manager server.
- **Logged-in User Menu** - The currently logged-in user. The subcommands are:
 - **Change Password** - Change the password of the currently logged in user.
 - **Logout**

Starting and Logging into the Admin Console

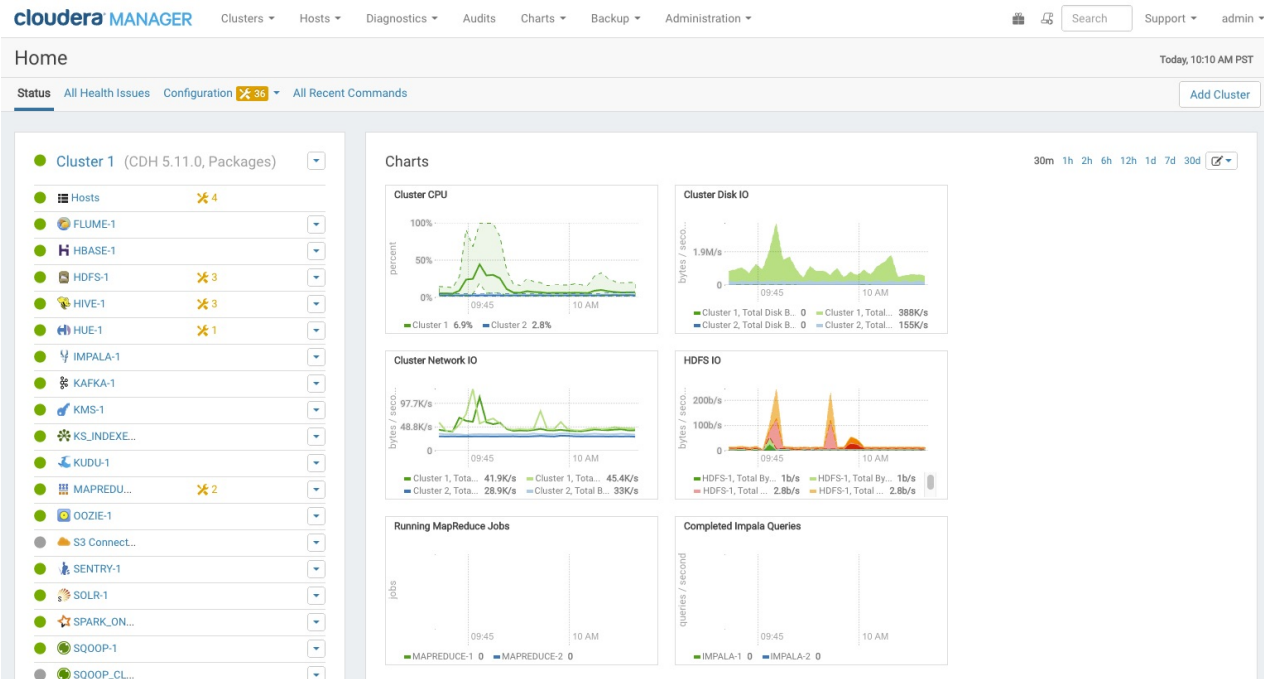
1. In a web browser, enter `http://Server host:7180`, where *Server host* is the FQDN or IP address of the host where the Cloudera Manager Server is running.
The login screen for Cloudera Manager Admin Console displays.
2. Log into Cloudera Manager Admin Console using the [credentials](#) assigned by your administrator. User accounts are assigned [roles](#) that constrain the features available to you.



Note: You can configure the Cloudera Manager Admin Console to automatically log out a user after a configurable period of time. See [Automatic Logout](#) on page 59.

Cloudera Manager Admin Console Home Page

When you start the [Cloudera Manager Admin Console](#) on page 53, the **Home > Status** tab displays.



You can also go to the **Home > Status** tab by clicking the Cloudera Manager logo in the top navigation bar.

Status

The Status tab contains:






- **Clusters** - The clusters being managed by Cloudera Manager. Each cluster is displayed either in summary form or in full form depending on the configuration of the **Administration > Settings > Other > Maximum Cluster Count Shown In Full** property. When the number of clusters exceeds the value of the property, only cluster summary information displays.
 - **Summary Form** - A list of links to cluster status pages. Click **Customize** to jump to the **Administration > Settings > Other > Maximum Cluster Count Shown In Full** property.
 - **Full Form** - A separate section for each cluster containing a link to the cluster status page and a table containing links to the Hosts page and the status pages of the services running in the cluster.

Each service row in the table has a menu of actions that you select by clicking



and can contain one or more of the following indicators:

Indicator	Meaning	Description
	Health issue	<p>Indicates that the service has at least one health issue. The indicator shows the number of health issues at the highest severity level. If there are Bad health test results, the indicator is red. If there are no Bad health test results, but Concerning test results exist, then the indicator is yellow. No indicator is shown if there are no Bad or Concerning health test results.</p> <div>Important: If there is one Bad health test result and two Concerning health results, there will be three health issues, but the number will be one.</div> <p>Click the indicator to display the Health Issues pop-up dialog box.</p> <p>By default only Bad health test results are shown in the dialog box. To display Concerning health test results, click the Also show <i>n</i> concerning issue(s)</p>

Indicator	Meaning	Description
		link. Click the link to display the Status page containing with details about the health test result.
	Configuration issue	<p>Indicates that the service has at least one configuration issue. The indicator shows the number of configuration issues at the highest severity level. If there are configuration errors, the indicator is red. If there are no errors but configuration warnings exist, then the indicator is yellow. No indicator is shown if there are no configuration notifications.</p> <div style="border: 1px solid #f1c40f; padding: 10px; margin: 10px 0;"> <p> Important: If there is one configuration error and two configuration warnings, there will be three configuration issues, but the number will be one.</p> </div> <p>Click the indicator to display the Configuration Issues pop-up dialog box.</p> <p>By default only notifications at the Error severity level are listed, grouped by service name are shown in the dialog box. To display Warning notifications, click the Also show <i>n</i> warning(s) link. Click the message associated with an error or warning to be taken to the configuration property for which the notification has been issued where you can address the issue. See Managing Services.</p>
 Restart Needed  Refresh Needed	Configuration modified	<p>Indicates that at least one of a service's roles is running with a configuration that does not match the current configuration settings in Cloudera Manager.</p> <p>Click the indicator to display the Stale Configurations page. To bring the cluster up-to-date, click the Refresh or Restart button on the Stale Configurations page or follow the instructions in Refreshing a Cluster, Restarting a Cluster, or Restarting Services and Instances after Configuration Changes.</p>
	Client configuration redeployment required	<p>Indicates that the client configuration for a service should be redeployed.</p> <p>Click the indicator to display the Stale Configurations page. To bring the cluster up-to-date, click the Deploy Client Configuration button on the Stale Configurations page or follow the instructions in Manually Redeploying Client Configuration Files.</p>

- **Cloudera Management Service** - A table containing a link to the Cloudera Manager Service. The Cloudera Manager Service has a menu of actions that you select by clicking



- **Charts** - A set of charts ([dashboard](#)) that summarize resource utilization (IO, CPU usage) and processing metrics.

Click a line, stack area, scatter, or bar chart to expand it into a full-page view with a legend for the individual charted entities as well more fine-grained axes divisions.

By default the time scale of a dashboard is 30 minutes. To change the time scale, click a duration link

[30m](#) [1h](#) [2h](#) [6h](#) [12h](#) [1d](#) [7d](#) [30d](#) at the top-right of the dashboard.

To set the dashboard type, click  and select one of the following:

- **Custom** - displays a custom dashboard.
- **Default** - displays a default dashboard.
- **Reset** - resets the custom dashboard to the predefined set of charts, discarding any customizations.

All Health Issues

Displays all health issues by cluster. The number badge has the same semantics as the per service health issues reported on the Status tab.

- By default only Bad health test results are shown in the dialog box. To display Concerning health test results, click the **Also show *n* concerning issue(s)** link.
- To group the health test results by entity or health test, click the buttons on the **Organize by Entity/Organize by Health Test** switch.
- Click the link to display the Status page containing with details about the health test result.

All Configuration Issues

Displays all configuration issues by cluster. The number badge has the same semantics as the per service configuration issues reported on the Status tab. By default only notifications at the Error severity level are listed, grouped by service name are shown in the dialog box. To display Warning notifications, click the **Also show *n* warning(s)** link. Click the message associated with an error or warning to be taken to the configuration property for which the notification has been issued where you can address the issue.

All Recent Commands

Displays all commands run recently across the clusters. A badge



indicates how many recent commands are still running. Click the command link to display details about the command and child commands. See also [Viewing Running and Recent Commands](#).

Starting and Logging into the Cloudera Manager Admin Console

1. In a web browser, enter `http://Server host:7180`, where *Server host* is the FQDN or IP address of the host where the Cloudera Manager Server is running.

The login screen for Cloudera Manager Admin Console displays.

2. Log into Cloudera Manager Admin Console using the [credentials](#) assigned by your administrator. User accounts are assigned [roles](#) that constrain the features available to you.



Note: You can configure the Cloudera Manager Admin Console to automatically log out a user after a configurable period of time. See [Automatic Logout](#) on page 59.

Displaying the Cloudera Manager Server Version and Server Time

To display the version, build number, and time for the Cloudera Manager Server:

1. Open the Cloudera Manager Admin Console.
2. Select **Support > About**.

Displaying Cloudera Manager Documentation

To display Cloudera Manager documentation:

1. Open the Cloudera Manager Admin Console.
2. Select **Support > Help, Installation Guide, API Documentation, or Release Notes**. By default, the Help and Installation Guide files from the Cloudera web site are opened. This is because local help files are not updated after installation. You can configure Cloudera Manager to open either the latest Help and Installation Guide from the Cloudera web site (this option requires Internet access from the browser) or locally-installed Help and Installation Guide by configuring the **Administration > Settings > Support > Open latest Help files from the Cloudera website** property.

Automatic Logout

For security purposes, Cloudera Manager automatically logs out a user session after 30 minutes. You can change this session logout period.

To configure the timeout period:

1. Click **Administration > Settings**.
2. Click **Category > Security**.
3. Edit the **Session Timeout** property.
4. Click **Save Changes** to commit the changes.

When the timeout is one minute from triggering, the user sees the following message:

Automatic Logout for Your Protection

Due to inactivity, your current work session is about to expire. For your security, Cloudera Manager sessions automatically end after 30 minutes of inactivity.

Your current session will expire in **1 minute**.
Press any key or click anywhere to continue.

If the user does not click the mouse or press a key, the user is logged out of the session and the following message appears:

Automatic Log Out Due to Inactivity

You are now logged out of your account.

We hadn't heard from you for about 30 minute(s), so for your security Cloudera Manager automatically logged you out of your account. Log back in below to continue.

☐ Remember me

Cloudera Manager API

The Cloudera Manager API provides configuration and service lifecycle management, service health information and metrics, and allows you to configure Cloudera Manager itself. The API is served on the same host and port as the [Cloudera Manager Admin Console](#) on page 53, and does not require an extra process or extra configuration. The API supports HTTP Basic Authentication, accepting the same users and credentials as the Cloudera Manager Admin Console.

Resources

- [Quick Start](#)
- [Cloudera Manager API tutorial](#)

- [Cloudera Manager API documentation](#)
- [Python client](#)
- [Using the Cloudera Manager API for Cluster Automation](#) on page 62

Obtaining Configuration Files

1. Obtain the list of a service's roles:

```
http://cm_server_host:7180/api/v18/clusters/clusterName/services/serviceName/roles
```

2. Obtain the list of configuration files a process is using:

```
http://cm_server_host:7180/api/v18/clusters/clusterName/services/serviceName/roles/roleName/process
```

3. Obtain the content of any particular file:

```
http://cm_server_host:7180/api/v18/clusters/clusterName/services/serviceName/roles/roleName/process/  
configFiles/configFileName
```

For example:

```
http://cm_server_host:7180/api/v18/clusters/Cluster%201/services/OOZIE-1/roles/  
OOZIE-1-OOZIE_SERVER-e121641328fcb107999f2b5fd856880d/process/configFiles/oozie-site.xml
```

Retrieving Service and Host Properties

To update a service property using the Cloudera Manager APIs, you'll need to know the name of the property, not just the display name. If you know the property's display name but not the property name itself, retrieve the documentation by requesting any configuration object with the query string `view=FULL` appended to the URL. For example:

```
http://cm_server_host:7180/api/v18/clusters/Cluster%201/services/service_name/config?view=FULL
```

Search the results for the display name of the desired property. For example, a search for the display name **HDFS Service Environment Advanced Configuration Snippet (Safety Valve)** shows that the corresponding property name is `hdfs_service_env_safety_valve`:

```
{  
  "name" : "hdfs_service_env_safety_valve",  
  "require" : false,  
  "displayName" : "HDFS Service Environment Advanced Configuration Snippet (Safety  
Valve)",  
  "description" : "For advanced use only, key/value pairs (one on each line) to be  
inserted into a roles  
environment. Applies to configurations of all roles in this service except client  
configuration.",  
  "relatedName" : "",  
  "validationState" : "OK"  
}
```

Similar to finding service properties, you can also find host properties. First, get the host IDs for a cluster with the URL:

```
http://cm_server_host:7180/api/v18/hosts
```

This should return host objects of the form:

```
{  
  "hostId" : "2c2e951c-aaf2-4780-a69f-0382181f1821",  
  "ipAddress" : "10.30.195.116",  
  "hostname" : "cm_server_host",  
  "rackId" : "/default",  
  "hostUrl" :  
"http://cm_server_host:7180/cmfd/hostRedirect/2c2e951c-adf2-4780-a69f-0382181f1821",  
}
```

```

    "maintenanceMode" : false,
    "maintenanceOwners" : [ ],
    "commissionState" : "COMMISSIONED",
    "numCores" : 4,
    "totalPhysMemBytes" : 10371174400
  }

```

Then obtain the host properties by including one of the returned host IDs in the URL:

```
http://cm_server_host:7180/api/v18/hosts/2c2e951c-adf2-4780-a69f-0382181f1821?view=FULL
```

Backing Up and Restoring the Cloudera Manager Configuration

You can use the Cloudera Manager REST API to export and import all of its configuration data. The API exports a JSON document that contains configuration data for the Cloudera Manager instance. You can use this JSON document to back up and restore a Cloudera Manager deployment.

Minimum Required Role: [Cluster Administrator](#) (also provided by **Full Administrator**)

Exporting the Cloudera Manager Configuration

1. Log in to the Cloudera Manager server host as the `root` user.
2. Run the following command:

```
# curl -u admin_uname:admin_pass "http://cm_server_host:7180/api/v18/cm/deployment" >
path_to_file/cm-deployment.json
```

Where:

- `admin_uname` is a username with either the Full Administrator or Cluster Administrator role.
- `admin_pass` is the password for the `admin_uname` username.
- `cm_server_host` is the hostname of the Cloudera Manager server.
- `path_to_file` is the path to the file where you want to save the configuration.

Redacting Sensitive Information from the Exported Configuration

The exported configuration may contain passwords and other sensitive information. You can configure redaction of the sensitive items by specifying a JVM parameter for Cloudera Manager. When you set this parameter, API calls to Cloudera Manager for configuration data do not include the sensitive information.



Important: If you configure this redaction, you cannot use an exported configuration to restore the configuration of your cluster due to the redacted information.

To configure redaction for the API:

1. Log in the Cloudera Manager server host.
2. Edit the `/etc/default/cloudera-scm-server` file by adding the following property (separate each property with a space) to the line that begins with `export CMF_JAVA_OPTS`:

```
-Dcom.cloudera.api.redaction=true
```

For example:

```
export CMF_JAVA_OPTS="-Xmx2G -Dcom.cloudera.api.redaction=true"
```

3. Restart Cloudera Manager:

```
$ sudo service cloudera-scm-server restart
```

Restoring the Cloudera Manager Configuration



Important: This feature is available only with a Cloudera Enterprise license. It is not available in Cloudera Express. For information on Cloudera Enterprise licenses, see [Managing Licenses](#).

Using a previously saved JSON document that contains the Cloudera Manager configuration data, you can restore that configuration to a running cluster.

1. Using the Cloudera Manager Administration Console, stop all running services in your cluster:

- a. On the **Home > Status** tab, click



to the right of the cluster name and select **Stop**.

- b. Click **Stop** in the confirmation screen. The **Command Details** window shows the progress of stopping services.

When **All services successfully stopped** appears, the task is complete and you can close the **Command Details** window.



Warning: If you do not stop the cluster before making this API call, the API call will stop *all* cluster services before running the job. Any running jobs and data are lost.

2. Log in to the Cloudera Manager server host as the `root` user.

3. Run the following command:

```
# curl --upload-file path_to_file/cm-deployment.json  
-u admin_username:admin_pass  
http://cm_server_host:7180/api/v18/cm/deployment?deleteCurrentDeployment=true
```

Where:

- `admin_username` is a username with either the Full Administrator or Cluster Administrator role.
- `admin_pass` is the password for the `admin_username` username.
- `cm_server_host` is the hostname of the Cloudera Manager server.
- `path_to_file` is the path to the file containing the JSON configuration file.

Using the Cloudera Manager API for Cluster Automation

One of the complexities of Apache Hadoop is the need to deploy clusters of servers, potentially on a regular basis. If you maintain hundreds of test and development clusters in different configurations, this process can be complex and cumbersome if not automated.

Cluster Automation Use Cases

Cluster automation is useful in various situations. For example, you might work on many versions of CDH, which works on a wide variety of OS distributions (RHEL 5 and RHEL 6, Ubuntu Precise and Lucid, Debian Wheezy, and SLES 11). You might have complex configuration combinations—highly available HDFS or simple HDFS, Kerberized or non-secure, YARN or MRv1, and so on. With these requirements, you need an easy way to create a new cluster that has the required setup. This cluster can also be used for integration, testing, customer support, demonstrations, and other purposes.

You can install and configure Hadoop according to precise specifications using the Cloudera Manager [REST API](#). Using the API, you can add hosts, install CDH, and define the cluster and its services. You can also tune heap sizes, set up

HDFS HA, turn on Kerberos security and generate keytabs, and customize service directories and ports. Every configuration available in Cloudera Manager is exposed in the API.

The API also provides access to management functions:

- Obtaining logs and monitoring the system
- Starting and stopping services
- Polling cluster events
- Creating a disaster recovery replication schedule

For example, you can use the API to retrieve logs from HDFS, HBase, or any other service, without knowing the log locations. You can also stop any service with no additional steps.

Use scenarios for the Cloudera Manager API for cluster automation might include:

- OEM and hardware partners that deliver Hadoop-in-a-box appliances using the API to set up CDH and Cloudera Manager on bare metal in the factory.
- Automated deployment of new clusters, using a combination of Puppet and the Cloudera Manager API. Puppet does the OS-level provisioning and installs the software. The Cloudera Manager API sets up the Hadoop services and configures the cluster.
- Integrating the API with reporting and alerting infrastructure. An external script can poll the API for health and metrics information, as well as the stream of events and alerts, to feed into a custom dashboard.

Java API Example

This example covers the Java API client.

To use the Java client, add this dependency to your project's `pom.xml`:

```
<project>
  <repositories>
    <repository>
      <id>cdh.repo</id>
      <url>https://repository.cloudera.com/groups/cloudera-repos</url>
      <name>Cloudera Repository</name>
    </repository>
    ...
  </repositories>
  <dependencies>
    <dependency>
      <groupId>com.cloudera.api</groupId>
      <artifactId>cloudera-manager-api</artifactId>
      <version>4.6.2</version>      <!-- Set to the version of Cloudera Manager you use -->
    </dependency>
    ...
  </dependencies>
  ...
</project>
```

The Java client works like a proxy. It hides from the caller any details about REST, HTTP, and JSON. The entry point is a handle to the root of the API:

```
RootResourceV18 apiRoot = new ClouderaManagerClientBuilder().withHost("cm.cloudera.com")
    .withUsernamePassword("admin", "admin").build().getRootV18();
```

From the root, you can traverse down to all other resources. (It's called "v18" because that is the current Cloudera Manager API version, but the same builder will also return a root from an earlier version of the API.) The tree view shows some key resources and supported operations:

- RootResourceV18
 - ClustersResourceV18 - host membership, start cluster
 - ServicesResourceV18 - configuration, get metrics, HA, service commands

- RolesResource - add roles, get metrics, logs
- RoleConfigGroupsResource - configuration
- ParcelsResource - parcel management
- HostsResource - host management, get metrics
- UsersResource - user management

For more information, see the [Javadoc](#).

The following example lists and starts a cluster:

```
// List of clusters
ApiClientList clusters = apiRoot.getClustersResource().readClusters(DataView.SUMMARY);
for (ApiClient cluster : clusters) {
    LOG.info("{}: {}", cluster.getName(), cluster.getVersion());
}

// Start the first cluster
ApiClient cmd = apiRoot.getClustersResource().startCommand(clusters.get(0).getName());
while (cmd.isActive()) {
    Thread.sleep(100);
    cmd = apiRoot.getCommandsResource().readCommand(cmd.getId());
}
LOG.info("Cluster start {}", cmd.getSuccess() ? "succeeded" : "failed " +
cmd.getResultMessage());
```

Python Example

You can see an example of automation with Python at the following link: [Python example](#). The example contains information on the requirements and steps to automate a cluster deployment.

Extending Cloudera Manager

In addition to the set of software packages and services managed by Cloudera Manager, you can also define and add new types of services using [custom service descriptors](#). When you deploy a custom service descriptor, the implementation is delivered in a Cloudera Manager [parcel](#) or other software package. For information on the extension mechanisms provided by Cloudera Manager for creating custom service descriptors and parcels, see [Cloudera Manager Extensions](#).

Cloudera Navigator Data Management Overview

Cloudera Navigator Data Management is a complete solution for data governance, auditing, and related data management tasks that is fully integrated with the Hadoop platform. Cloudera Navigator lets compliance groups, data stewards, administrators, and others work effectively with data at scale. This brief introduction includes the following topics:

- [Data Management Challenges](#) on page 65
- [Cloudera Navigator Data Management Capabilities](#) on page 65
- [Getting Started with Cloudera Navigator](#) on page 66
- [Cloudera Navigator Frequently Asked Questions](#) on page 72

Data Management Challenges

As Hadoop clusters have become ubiquitous in organizations large and small, the ability to store and analyze all types of data at any scale brings with it some management challenges for reasons such as these:

- Data volumes are extremely large and continue to grow, with increased velocity while comprising various data types.
- Data ingested by the cluster at one point in time is transformed by many different processes, users, or applications. The result is that the provenance of any data entity can be difficult at best to trace.
- Multi-user and multi-tenant access to the data is the norm. Each user group may need different levels of access to the data, at varying degrees of granularity.

These characteristics make it difficult to answer questions such as these:

- Where did the data originate? Has it been altered, and if so, by whom or by what process?
- Are downstream processes using that data, and if so, how?
- Have unauthorized people been trying to get to the data? Can we verify all accesses and attempted access to our data?
- Are we prepared for an audit? For example, can we prove that the values shown in the organization's G/L have not been mishandled? Will the bank examiners approve our data governance operations? In general, can the data's validity be trusted, and better yet, proven?
- Are data retention mandates being met? Are we complying with all industry and government regulations for preservation and deletion of data? Can we automate the life cycle of our data?
- Where is the most important data? Is there a way to get an at-a-glance view about overall cluster activity, over various time periods, by data type?

Enabling organizations to quickly answer questions such as these and many more is one of the chief benefits of Cloudera Navigator.

Cloudera Navigator Data Management Capabilities

As an organization's clusters consume more and more data, its business users want self-service access to that data. For example, business users might want to find data relevant for a current market analysis by searching using the nomenclature common to their field of expertise rather than needing to know all the file types that might contain such data.

At the same time, the organization's security team wants to know about all attempts to access any and all data. They want to be able to quickly identify confidential data and to track any data entity to its source (provenance). The compliance group wants to be audit-ready at all times. And everyone, organization wide, wants to completely trust the integrity of the data they are using.

These are the kinds of data management challenges that Cloudera Navigator data management was designed to meet head on. Data stewards, administrators, business analysts, data scientists, and developers can obtain better value from the vast amounts of data stored in Hadoop clusters through the growing set of features provided by Cloudera Navigator data management.

The following capabilities are all available through the [Cloudera Navigator console](#).

Analytics

The Cloudera Navigator analytics system leverages the metadata system, policies, and auditing features to provide a starting place for most users, from data stewards to administrators. Get at-a-glance overviews of all cluster data across various dimensions and using a variety of interactive tools to easily filter and drill down into specific data objects. For example, Data Stewardship Analytics has a [Dashboard](#) and Data Explorer that provide comprehensive yet intuitive tools for interacting with all data in the system. The HDFS Analytics page displays histograms of technical metadata for HDFS files (file size, block size, and so on). [Use the mouse and brush over any histogram to display the lower level details](#), along with selectors and other filters for digging further below the surface of the data objects.

Auditing

In addition to meeting the needs of data stewards, Cloudera Navigator also meets governance needs by providing secure real-time auditing. The Navigator Audit Server creates a complete and immutable record of cluster activity (in its own database) that is easy to retrieve when needed. Compliance groups can configure, collect, and view audit events that show who accessed data, when, and how.

Metadata

The analytics, lineage, and search capabilities of Cloudera Navigator rely on metadata, both the technical metadata inherent in the data object itself (date, file type, and so on) or the metadata you define (managed metadata) to characterize data objects so they can be easily found by data stewards, data scientists, and other business users.

For example, Navigator Metadata Server captures information about table, file, and database activity, including file and table creation and modification trends, all of which is displayed in visually clean renderings on the Data Stewardship dashboard.

Policies

Cloudera Navigator [policies](#) let you automate actions based on data access or on a schedule, to add metadata, create alerts, move data, or purge data.

Search

By default, the Cloudera Navigator console opens to the Search menu. Use this sophisticated (yet simple) filtering scheme to find all types of entities that meet your criteria, then drill down to explore a specific entity's lineage. High level diagrams can be filtered further using the interactive tools provided, letting you follow the data upstream or downstream and in myriad other ways.

Getting Started with Cloudera Navigator

[Cloudera Navigator features and functions](#) are available through the Cloudera Navigator console, a web-based user interface provided by the host configured with the Navigator Metadata Server role (daemon). The Cloudera Navigator console invokes the REST APIs exposed by the web service provided by the Navigator Metadata Server role instance. The APIs can be called directly by in-house, third-party, and other developers who want to further automate tasks or implement functionality not currently provided by the Cloudera Navigator console. This topic discusses both of these interfaces to the Cloudera Navigator metadata component.



Note: Navigator Metadata Server is one of the roles (daemons) that comprises Cloudera Navigator. The other role is Navigator Audit Server. See [Cloudera Navigator Data Management](#) for details.

The following steps assume that the Cloudera Navigator data management component is running, and that you know the host name and port of the node in the cluster configured with the Navigator Metadata Server role. See [Cloudera Installation](#) and [Cloudera Administration](#) guides for more information about setting up Cloudera Navigator.

Cloudera Navigator Console

The Cloudera Navigator console is the web-based user interface that provides data stewards, compliance groups, auditing teams, data engineers, and other business users access to [Cloudera Navigator features and functions](#).

The Cloudera Navigator console provides a unified view of auditing, lineage, and other data management capabilities across all clusters managed by a given Cloudera Manager instance. That is, if the Cloudera Manager instance manages five clusters, the Cloudera Navigator console can obtain auditing, lineage, metadata management, and so on for the same five clusters. Cloudera Navigator uses its technical metadata gathering feature to keep track of which cluster originates the data.

Accessing the Cloudera Navigator Console

The Cloudera Navigator console can be accessed from the Cloudera Manager Admin Console or directly on the Navigator Metadata Server instance. Only Cloudera Manager users with the role of Navigator Administrator (or Full Administrator) can access Cloudera Navigator console from the Cloudera Manager Admin Console.

From the Cloudera Manager Admin Console:

1. Open your browser.
2. Navigate to the Cloudera Manager Admin Console.
3. Log in as either Navigator Administrator or Full Administrator.
4. From the menu, select **Clusters** > **Cluster-n**.
5. Select **Cloudera Navigator** from the menu.
6. Log in to the Cloudera Navigator console.

To access the Cloudera Navigator console directly:

1. Open your browser.
2. Navigate to the host within the cluster running the Cloudera Navigator Metadata Server role.

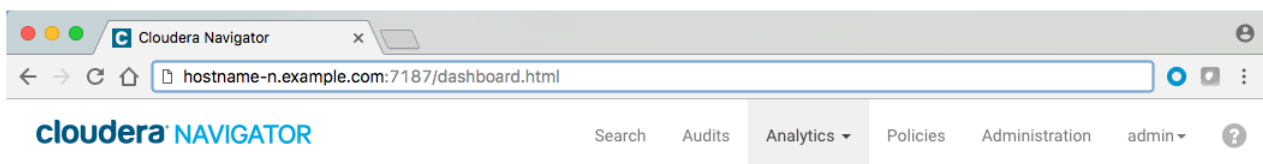
`http://fqdn-1.example.com:7187/login.html`

In this example, node 1 of the cluster is running the Navigator Metadata Server role, hosted on the default port 7187. The login page displays.

3. Log in to the Cloudera Navigator console using the [credentials](#) assigned by your administrator.


Menu and Icon Overview

The Cloudera Navigator console opens to an empty Search results page and displays the following menu.



The `user_name` (the account that is logged in to Cloudera Navigator) and question mark icon (see the table) always display. However, other available menu choices depend on the Cloudera Navigator user role of the logged in account. In this example, the **admin** account is logged in. That account has Navigator Administrator privileges which means that all menu items display and the user has full access to all system features. For a user account limited to Policy Editor privileges, only the Policies menu item displays.

This table provides a reference to all menu choices.

Icon or menu	Description
Search	Displays sets of filters that can be applied to find specific entities by source type, type, owner, cluster name, and tags.
Audits	Displays summary list of current audit events for the selected filter. Define filters for audit events, create reports based on selected time frames and filters, export selected audit details as CSV or JSON files.
Analytics	Menu for HDFS Analytics and Data Stewardship Analytics selections. This is your entry point for several dashboards that display the overall state of cluster data and usage trends: <ul style="list-style-type: none"> Data Stewardship Analytics displays Dashboard and Data Explorer tabs. HDFS Analytics displays Metadata and Audit tabs.
Policies	Displays a list of existing policies and enables you to create new policies.
Administration	Displays tabs for Managed Metadata, Role Management, and Purge schedule settings .
<i>user_name</i>	Displays account (admin , for example) currently logged in to Cloudera Navigator. Menu for Command Actions, My Roles, Enable Debug Mode, and Logout selections.
	Help icon and menu. Provides access to Help, API Documentation, and About selections. Select Help to display Cloudera Navigator documentation . Select About to display version information .

This is just an overview of the menu labels in the Cloudera Navigator console and pointers to some of the sub-menus. See the [Cloudera Navigator Data Management](#) guide for details.

Displaying Documentation

Online documentation is accessible from Cloudera Navigator console as follows:

1. Click [Help](#).
2. Select **Help** from the menu.

The [Cloudera Data Management](#) guide displays.

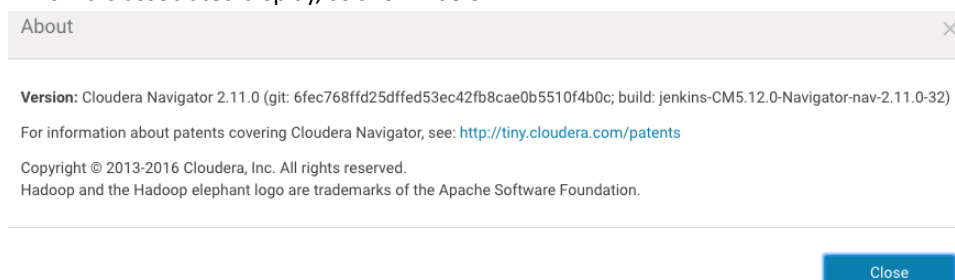


Note: You can also view the [Cloudera Data Management](#) guide outside the scope of a running instance of Cloudera Navigator. See also the complete [list of other Cloudera Navigator data management component documentation](#).

Displaying Version Information

To display the version and build number for the Cloudera Navigator data management component:

1. Click [Help](#).
2. Select **About** from the menu. The versions for both Cloudera Navigator and the Cloudera Manager instance with which it is associated display, as shown below:



3. Click **Close** to dismiss the message and redisplay the Cloudera Navigator console.

Cloudera Navigator APIs

Cloudera Navigator exposes REST APIs on the host running the Navigator Metadata Server role. In-house, third-party, and other developers can use the APIs to provide functionality not currently available through the Cloudera Navigator console or to customize their users' experience.

The APIs are used by the Cloudera Navigator console at runtime as users interact with the system. The Cloudera Navigator console has a debug mode that captures API calls and responses to a file that can then be downloaded and analyzed. See [Using Debug Mode](#) on page 70 for details.

Documentation and a brief tutorial for the Cloudera Navigator APIs are available on the same node of the cluster that runs the Navigator Metadata Server role. These are also accessible from the Cloudera Navigator console. Example pathnames for direct access (outside the Cloudera Navigator console) are shown in the table.

Resource	Default location
APIs	<code>http://fqdn-n.example.com:port/api/vn/operation</code>
API Documentation	<code>http://fqdn-n.example.com:port/api-console/index.html</code>
API Usage Tutorial	<code>http://fqdn-n.example.com:port/api-console/tutorial.html</code>

Operations on the API are invoked using the following general pattern:

```
http://fqdn-n.example.com:port/api/vn/operation
```

The *n* in *vn* for the [APIs](#) represents the version number, for example, v11 for the Cloudera Navigator version 2.11.x. The API version number is located at the bottom of the Cloudera Navigator API documentation page.

The API uses HTTP Basic Authentication and accepts the same users and credentials that have access to the Cloudera Navigator console.

The resources listed in the table are aimed at technical and general audiences for Cloudera Navigator Data Management. The GitHub repository is aimed at developers.

Resource	Default location
GitHub cloudera/navigator-sdk	Cloudera Navigator SDK is a client library that provides functionality to help users extract metadata from Navigator and to enrich the metadata in Navigator with custom metadata models, entities, and relationships.
Cloudera Community Data Management forum	Cloudera Community > Data Management > Data Discovery, Optimization forum for Cloudera Navigator users and developers.
Cloudera Engineering Blog	In-depth technical content provided by Cloudera engineers about components and specific subject areas.

Accessing API Documentation

The API documentation is available from the Cloudera Navigator console from a menu selection that navigates to the same URL listed in the table above.

To access the API documentation and tutorial:

1. Click [Help](#).
2. Select **API Documentation**. The interactive API documentation page displays in a new window.
3. Click the Tutorial link near the top of the page to view the Cloudera Navigator API Usage Tutorial to learn about the APIs.

Locating the Version Information for the Navigator API

1. Click [Help](#).
2. Select **API Documentation** to display the API documentation.
3. Scroll to the bottom of the page. The version number is the last item on this page:

[BASE URL: <http://node-5.example.com:7187/api/api-docs>, API VERSION: v10]

The API is structured into resource categories. Click a category to display the resource endpoints.

Using Debug Mode

Cloudera Navigator console debug mode captures API responses and requests during interaction with the server. Use debug mode to facilitate development efforts or when asked by Cloudera Support to troubleshoot issues.

Tip: Exploring debug files can be a good way to learn about the APIs.

The general process for obtaining debug files to send to Cloudera Support is as follows:


1. Navigate to the Cloudera Navigator console menu item or page for which you want to obtain debug information.
2. Enable debug mode.
3. Reload the page and perform the specific action. For example, if you need to send a debug file to Cloudera Support for an error message that displays when you click on an entity in one of the charts, perform that action to raise the error message.
4. Download the debug file or files. Each time the counter changes, the content of the debug file content changes, so you should download each time the counter updates.
5. Disable debug mode.

To follow these steps you must be [logged in to the Cloudera Navigator console](#).

Enable Debug Mode

The Enable Debug Mode menu selection is effectively a toggle. In other words, when you Enable Debug Mode, the menu selection changes to Disable Debug Mode. To enable debug mode:

1. Click the **user_name** from the menu to open the drop-down menu.
2. Select **Enable Debug Mode**. The Cloudera Navigator consoles displays a notification message in the bottom right of the page:



Debug mode enabled. Captured 0 calls. [Disable](#) [Download debug file](#)

Notice the links to Disable and Download debug file, which you can use at any time to do either. Do not disable debug mode until you have downloaded the debug file you need.

3. Reload the browser page.

The counter *n* displays the number of API calls captured in the debug file for the current session interaction. Here is an example from a downloaded debug file showing one of the responses after reloading the page:

```
{
  "type": "GET",
  "url": "/api/v10/dashboard/clusters",
  "status": 200,
  "responseText": "{\\n  \\\"clusterInfos\\\" : [ {\\n    \\\"clusterId\\\" : \\\"Cluster 1\\\"\\n    ,\\n    \\\"clusterDisplayText\\\" : \\\"Cluster 1\\\"\\n  } ]\\n}",
  "page": "http://fqdn-1.example.com:7187/dashboard.html",
  "timestamp": 1497719524407
},
```

Download Debug File or Files

As you perform actions with the Cloudera Navigator console, the counter refreshes and keeps tracks of current calls only (stateless nature of REST). In other words, download the debug file for any given action or selection that you want to examine.

To download the debug file:

- Click the **Download debug file** link in the debug-enabled notification message.
- Save the file to your local workstation.

Debug files use a naming convention that includes the prefix "api-data" followed by the name of the host and the UTC timestamp of the download time and the JSON file extension. The pattern is as follows:

```
api-data-fqdn-yyyy-mm-ddThh-mm-ss.json
```

For example:

```
api-data-fqdn-2.example.com-2017-06-15T09-00-32.json
```

Disable Debug Mode

When you are finished capturing and downloading debug files, you can disable debug mode.

- Click the **Disable debug mode** link in the notification message box.

The message box is dismissed and debug mode is disabled.

Or, use the Cloudera Navigator console menu:

- Select **user_name** > **Disable Debug Mode**.

Cloudera Navigator Data Management Documentation Library

The [Cloudera Documentation](#) portal includes the following information for Cloudera Navigator data management component:

FAQ	Cloudera Navigator Frequently Asked Questions answers common questions about Cloudera Navigator data management component and how it interacts with other Cloudera products and cluster components.
Introduction	Cloudera Navigator Data Management Overview provides preliminary information about common Cloudera Navigator data management for data stewards, governance and compliance teams, data engineers, and administrators. Includes an introduction and overview of the Cloudera Navigator console (the UI) and the Cloudera Navigator API.
User Guide	Cloudera Navigator Data Management guide shows data stewards, compliance officers, and other business users how to use Cloudera Navigator for data governance, compliance, data stewardship, and other tasks. Topics include Auditing , Metadata , Lineage Diagrams . The guide includes a Services and Security Management chapter that shows systems administrators how to configure, customize, and tune the back-end services, and how to integrate Cloudera Navigator and the Cloud .
Installation	Installing the Cloudera Navigator Data Management Component
Upgrade	Upgrading the Cloudera Navigator Data Management Component
Security	Configuring Authentication for Cloudera Navigator
	Configuring TLS/SSL for Navigator Metadata Server
Release Notes	Cloudera Navigator Data Management Release Notes includes new features, changed features, known issues, and resolved issues for each release. Current users of Cloudera Navigator data management component should always read the release notes before upgrading.

Cloudera Navigator Frequently Asked Questions

Cloudera Navigator offers components for security, data management, and data optimization, as follows:

- [Cloudera Navigator Data Encryption](#) is a data-at-rest encryption and key management suite that includes [Cloudera Navigator Encrypt](#), [Cloudera Navigator Key Trustee Server](#), among other components.
- [Cloudera Navigator Data Management](#) is a comprehensive auditing, data governance, compliance, data stewardship, and data lineage discovery component that is fully integrated with Hadoop.
- [Cloudera Navigator Optimizer](#) is an analysis tool that profiles and analyzes query text in SQL workloads. Use Cloudera Navigator Optimizer to better understand workloads, identify and offload queries best-suited for Hadoop, and optimize workloads already running on Hive or Impala.

The FAQs below discuss the Cloudera Navigator Data Management component only.

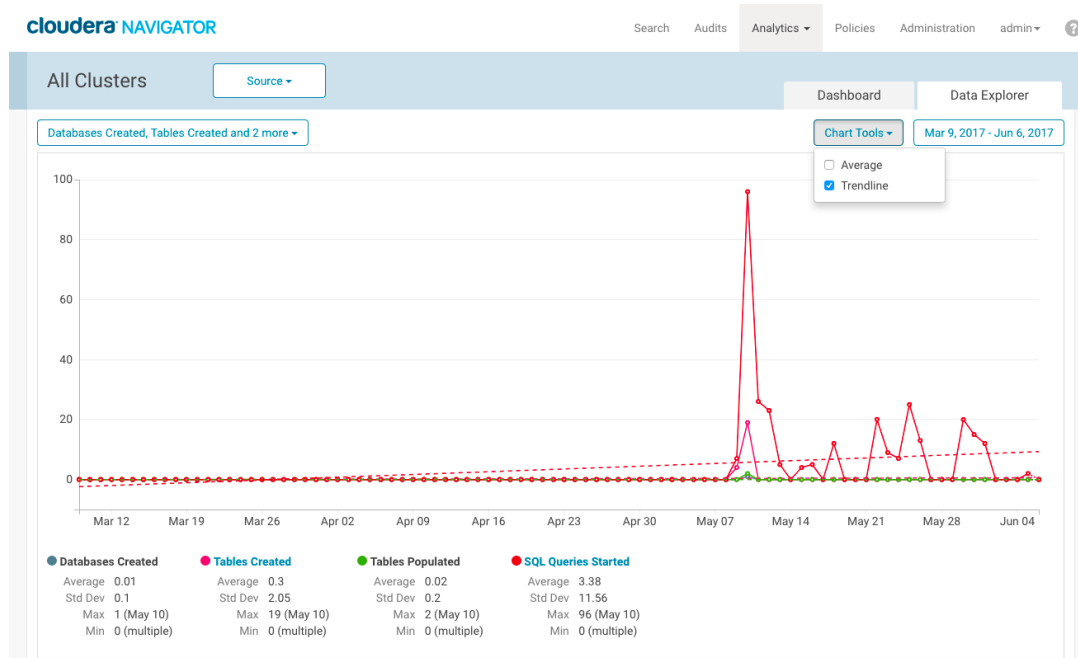
Is Cloudera Navigator a module of Cloudera Manager?

Not exactly. Cloudera Navigator is installed separately, after Cloudera Manager is installed, and it interacts behind the scenes with Cloudera Manager to deliver some of its core functionality. Cloudera Manager is used by cluster administrators to manage the cluster and all its services. Cloudera Navigator is used by administrators but also by security and governance teams, data stewards, and others to audit, trace data lineage from source raw data through final form, and perform other comprehensive data governance and stewardship tasks.

Does Cloudera Navigator have a user interface?

Yes. The Cloudera Navigator console is a browser-based user interface hosted on the node running an instance of the Navigator Metadata Server role. It provides a unified view of clusters associated with a given Cloudera Manager instance and lets you customize metadata, trace lineage, view audit events, and so on.

For example, the screenshot below shows a Data Explorer view of databases and tables created, tables populated, and SQL queries started within the specified timeframe (May 9, 2017 to June 6, 2017), with a trendline (the dashed line) displayed in the context of the results:



Clicking anywhere within the chart lets you drill down into the entities and begin identifying sources of spikes and trends and open lineage diagrams.

- See the [Cloudera Navigator Data Management Overview](#) to learn more about the types of questions that Cloudera Navigator is designed to answer.

- See [Getting Started with Cloudera Navigator](#) for an overview of the Cloudera Manager console.

How can I access the Cloudera Navigator console?

The Cloudera Navigator console can be accessed from the Cloudera Manager Admin Console or directly on the Navigator Metadata Server instance. Using the Cloudera Manager Admin Console as a starting point requires the Cloudera Manager roles of either Navigator Administrator or Full Administrator.

From the Cloudera Manager Admin Console:

1. Select **Clusters** > **Cluster-n**.
2. Select **Cloudera Navigator** from the menu.

To access the Cloudera Navigator console directly:

1. Open the browser to the host name of the node in the cluster that runs the Cloudera Navigator Metadata Server. For example, if node 2 in the cluster is running the Navigator Metadata Server role, the URL to the Cloudera Navigator console (assuming the default port of 7187) might look as follows:

```
http://fqdn-2.example.com:7187/login.html
```

2. Log in to the Cloudera Navigator console using the [credentials](#) assigned by your administrator.

What are the requirements for Cloudera Navigator?

Cloudera Navigator data management is a standard component of the Cloudera Enterprise edition. It is not included with Cloudera Express and requires an enterprise license. The Cloudera Enterprise license is an annual subscription that provides a complete portfolio of support and services. See [Cloudera Product FAQ](#) for details.

Can Cloudera Navigator be purchased separately?

No. Cloudera Navigator auditing and metadata subsystems interact with various subsystems of Cloudera Manager. For companies without Kerberos or other external means of authentication, Cloudera Navigator uses Cloudera Manager for user authentication. In short, Cloudera Manager is a pre-requisite for Cloudera Navigator, and must be installed before you can install Cloudera Navigator.

What versions of Cloudera Manager, CDH, and Impala does Cloudera Navigator support?

See [Product Compatibility Matrix for Cloudera Navigator](#).

Is Cloudera Navigator an open source project?

No. Cloudera Navigator is closed source software that is fully integrated with the Hadoop platform. Cloudera Navigator complements Cloudera Manager and part of the Cloudera suite of management capabilities for Hadoop clusters.

Do Cloudera Navigator logs differ from Cloudera Manager logs?

Yes. For example, Cloudera Navigator tracks and aggregates accesses to the data stored in CDH services and is used for audit reports and analysis. Cloudera Manager monitors and logs all the activity performed by CDH services that helps administrators maintain the health of the cluster. Together these logs provide better visibility into both data access and system activity for an enterprise cluster.

How can I learn more about Cloudera Navigator?

See [Cloudera Navigator Data Management Overview](#) for a more detailed introduction to Cloudera Navigator. Other Cloudera Navigator information resources are listed in the table. Installation, upgrade, administration, and security guides are all aimed at administrators.

User Guide	Cloudera Navigator Data Management guide shows data stewards, compliance officers, and business users how to use Cloudera Navigator for data governance, compliance, data stewardship, and other tasks. Topics include Auditing , Metadata , Lineage Diagrams , Cloudera Navigator and the Cloud , Services and Security Management , and more.
------------	---

Installation	Installing the Cloudera Navigator Data Management Component
Upgrade	Upgrading the Cloudera Navigator Data Management Component
Security	Configuring Authentication for Cloudera Navigator
	Configuring TLS/SSL for Navigator Audit Server
	Configuring TLS/SSL for Navigator Metadata Server

Cloudera Navigator Data Encryption Overview



Warning: Encryption transforms coherent data into random, unrecognizable information for unauthorized users. It is *absolutely critical* that you follow the documented procedures for encrypting and decrypting data, and that you regularly back up the encryption keys and configuration files. Failure to do so can result in irretrievable data loss. See [Backing Up and Restoring Key Trustee Server and Clients](#) for more information.

Do not attempt to perform any operations that you do not understand. If you have any questions about a procedure, contact Cloudera Support before proceeding.

Cloudera Navigator includes a turnkey encryption and key management solution for data at rest, whether data is stored in HDFS or on the local Linux filesystem. Cloudera Navigator encryption comprises the following components:

- [Cloudera Navigator Key Trustee Server](#)

Key Trustee Server is an enterprise-grade virtual safe-deposit box that stores and manages cryptographic keys. With Key Trustee Server, encryption keys are separated from the encrypted data, ensuring that sensitive data is protected in the event that unauthorized users gain access to the storage media.

- [Cloudera Navigator Key HSM](#)

Key HSM is a service that allows Key Trustee Server to integrate with a hardware security module (HSM). Key HSM enables Key Trustee Server to use an HSM as the root of trust for cryptographic keys, taking advantage of Key Trustee Server's policy-based key and security asset management capabilities while satisfying existing internal security requirements regarding treatment of cryptographic materials.

- [Cloudera Navigator Encrypt](#)

Navigator Encrypt is a client-side service that transparently encrypts data at rest without requiring changes to your applications and with minimal performance lag in the encryption or decryption process. Advanced key management with Key Trustee Server and process-based access controls in Navigator Encrypt enable organizations to meet compliance regulations and ensure unauthorized parties or malicious actors never gain access to encrypted data.

- Key Trustee KMS

For [HDFS Transparent Encryption](#), Cloudera provides Key Trustee KMS, a customized [key management server \(KMS\)](#) that uses Key Trustee Server for robust and scalable encryption key storage and management instead of the file-based Java KeyStore used by the default Hadoop KMS.

- Cloudera Navigator HSM KMS

Also for [HDFS Transparent Encryption](#), Navigator HSM KMS provides a customized [key management server \(KMS\)](#) that uses third-party HSMs to provide the highest level of key isolation, storing key material on the HSM. When using the Navigator HSM KMS, encryption zone key material originates on the HSM and never leaves the HSM. While Navigator HSM KMS allows for the highest level of key isolation, it also requires some overhead for network calls to the HSM for key generation, encryption and decryption operations.

- Cloudera Navigator HSM KMS Services and HA

Navigator HSM KMSs running on a single node fulfill the functional needs of users, but do not provide the non-functional qualities of service necessary for production deployment (primarily key data high availability and key data durability). You can achieve high availability (HA) of key material through the HA mechanisms of the backing HSM. However, metadata cannot be stored on the HSM directly, so the HSM KMS provides for high availability of key metadata via a built-in replication mechanism between the metadata stores of each KMS role instance. This release supports a two-node topology for high availability. When deployed using this topology,

there is a durability guarantee enforced for key creation and roll such that a key create or roll operation will fail if it cannot be successfully replicated between the two nodes.

Cloudera Navigator encryption provides:

- High-performance transparent data encryption for files, databases, and applications running on Linux
- Separation of cryptographic keys from encrypted data
- Centralized management of cryptographic keys
- Integration with hardware security modules (HSMs) from Thales and SafeNet
- Support for Intel AES-NI cryptographic accelerator for enhanced performance in the encryption and decryption process
- Process-Based Access Controls

Cloudera Navigator encryption can be deployed to protect different assets, including (but not limited to):

- Databases
- Log files
- Temporary files
- Spill files
- HDFS data

For planning and deployment purposes, this can be simplified to two types of data that Cloudera Navigator encryption can secure:

1. HDFS data
2. Local filesystem data

The following table outlines some common use cases and identifies the services required.

Table 2: Encrypting Data at Rest

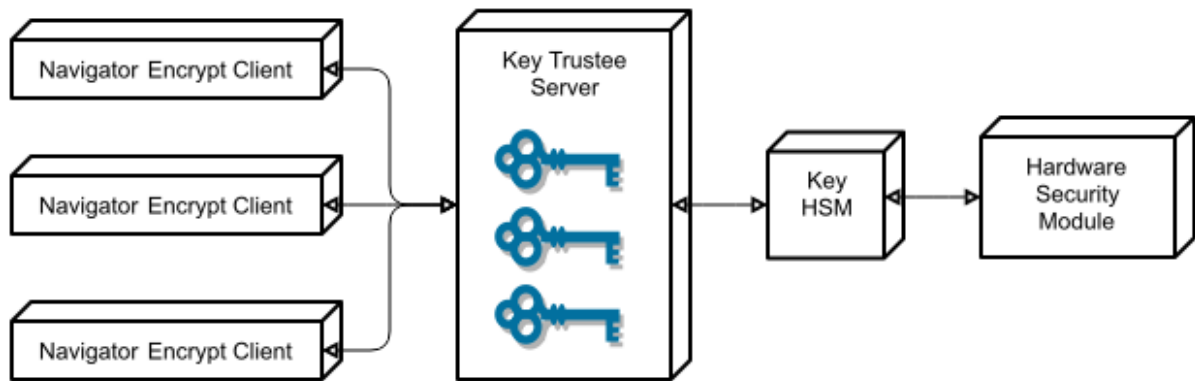
Data Type	Data Location	Key Management	Additional Services Required
HDFS	HDFS	Key Trustee Server	Key Trustee KMS
Metadata databases, including: <ul style="list-style-type: none"> • Hive Metastore • Cloudera Manager • Cloudera Navigator Data Management • Sentry 	Local filesystem	Key Trustee Server	Navigator Encrypt
Temp/spill files for CDH components with native encryption: <ul style="list-style-type: none"> • Impala • YARN • MapReduce • Flume • HBase • Accumulo 	Local filesystem	N/A (temporary keys are stored in memory only)	None (enable native temp/spill encryption for each component)
Temp/spill files for CDH components without native encryption:	Local filesystem	Key Trustee Server	Navigator Encrypt

Data Type	Data Location	Key Management	Additional Services Required
<ul style="list-style-type: none"> • Spark • Kafka • Sqoop2 • HiveServer2 			
Log files	Local filesystem	Key Trustee Server	Navigator Encrypt Log Redaction

For instructions on using Navigator Encrypt to secure local filesystem data, see [Cloudera Navigator Encrypt](#).

Cloudera Navigator Encryption Architecture

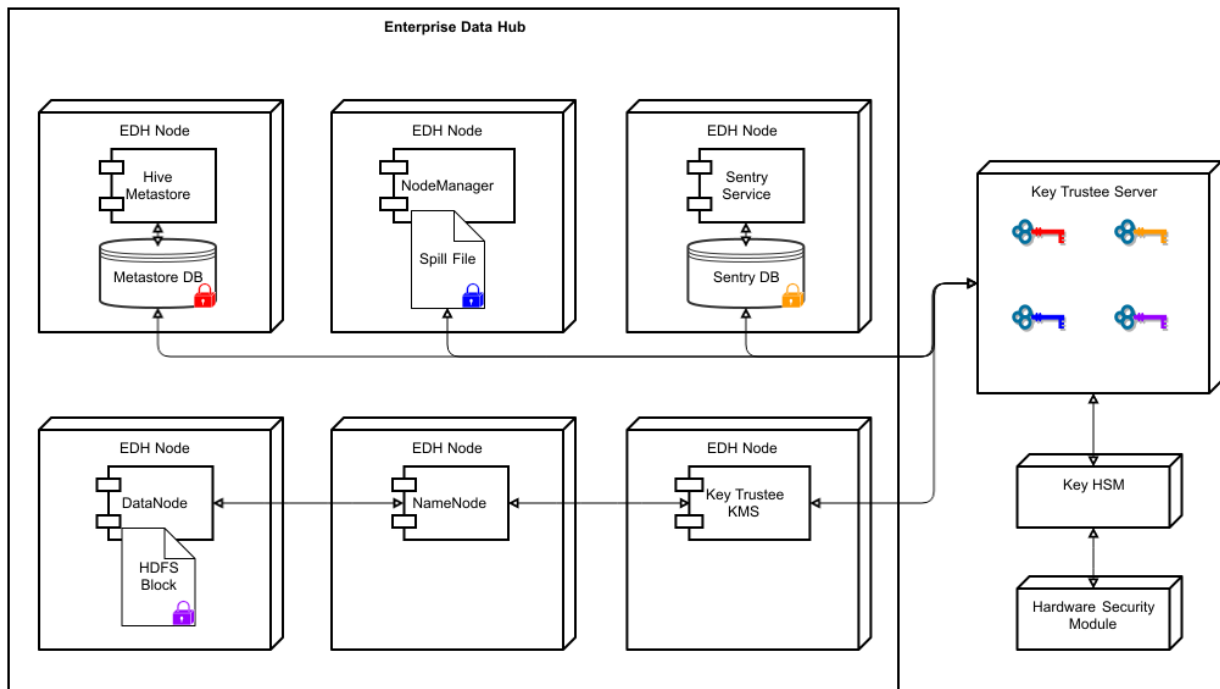
The following diagram illustrates how the Cloudera Navigator encryption components interact with each other:



Key Trustee clients include Navigator Encrypt and Key Trustee KMS. Encryption keys are created by the client and stored in Key Trustee Server.

Cloudera Navigator Encryption Integration with an EDH

The following diagram illustrates how the Cloudera Navigator encryption components integrate with an Enterprise Data Hub (EDH):



For more details on the individual components of Cloudera Navigator encryption, continue reading:

Cloudera Navigator Key Trustee Server Overview

Cloudera Navigator Key Trustee Server is an enterprise-grade virtual safe-deposit box that stores and manages cryptographic keys and other security artifacts. With Navigator Key Trustee Server, encryption keys are separated from the encrypted data, ensuring that sensitive data is still protected if unauthorized users gain access to the storage media.

Key Trustee Server protects these keys and other critical security objects from unauthorized access while enabling compliance with strict data security regulations. For added security, Key Trustee Server can integrate with a [hardware security module](#) (HSM). See [Cloudera Navigator Key HSM Overview](#) on page 79 for more information.

In conjunction with the Key Trustee KMS, Navigator Key Trustee Server can serve as a backing key store for [HDFS Transparent Encryption](#), providing enhanced security and scalability over the file-based Java KeyStore used by the default Hadoop Key Management Server.

Cloudera Navigator Encrypt also uses Key Trustee Server for key storage and management.

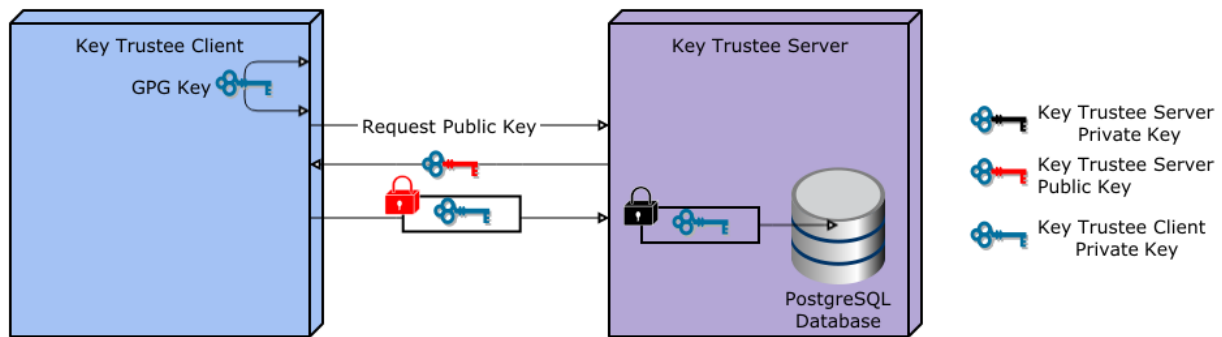
For instructions on installing Navigator Key Trustee Server, see [Installing Cloudera Navigator Key Trustee Server](#). For instructions on configuring Navigator Key Trustee Server, see [Initializing Standalone Key Trustee Server](#) or [Cloudera Navigator Key Trustee Server High Availability](#).

Key Trustee Server Architecture

Key Trustee Server is a secure object store. Clients register with Key Trustee Server, and are then able to store and retrieve objects with Key Trustee Server. The most common use case for Key Trustee Server is storing encryption keys to simplify key management and enable compliance with various data security regulations, but Key Trustee Server is agnostic about the actual objects being stored.

All interactions with Key Trustee Server occur over a TLS-encrypted HTTPS connection.

Key Trustee Server does not generate encryption keys for clients. Clients generate encryption keys, encrypt them with their private key, and send them over a TLS-encrypted connection to the Key Trustee Server. When a client needs to decrypt data, it retrieves the appropriate encryption key from Key Trustee Server and caches it locally to improve performance. This process is demonstrated in the following diagram:



The most common Key Trustee Server clients are Navigator Encrypt and Key Trustee KMS.

When a Key Trustee client registers with Key Trustee Server, it generates a unique fingerprint. All client interactions with the Key Trustee Server are authenticated with this fingerprint. You must ensure that the file containing this fingerprint is secured with appropriate Linux file permissions. The file containing the fingerprint is `/etc/navencrypt/keytrustee/ztrustee.conf` for Navigator Encrypt clients, and `/var/lib/kms-keytrustee/keytrustee/.keytrustee/keytrustee.conf` for Key Trustee KMS.

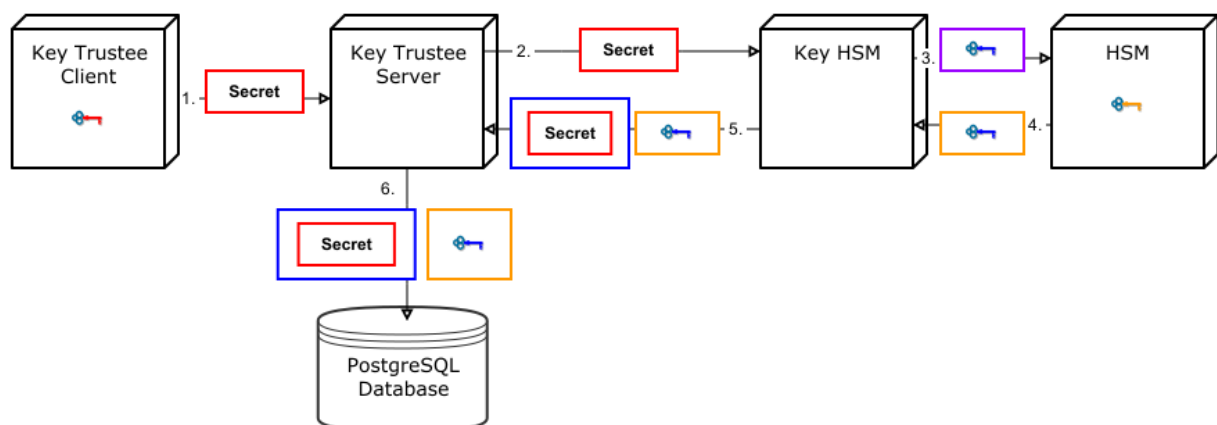
Many clients can use the same Key Trustee Server to manage security objects. For example, you can have several Navigator Encrypt clients using a Key Trustee Server, and also use the same Key Trustee Server as the backing store for Key Trustee KMS (used in HDFS encryption).

Cloudera Navigator Key HSM Overview

Cloudera Navigator Key HSM allows Cloudera Navigator Key Trustee Server to seamlessly integrate with a hardware security module (HSM). Key HSM enables Key Trustee Server to use an HSM as a root of trust for cryptographic keys, taking advantage of Key Trustee Server's policy-based key and security asset management capabilities while satisfying existing, internal security requirements for treatment of cryptographic materials.

Key HSM adds an additional layer of encryption to Key Trustee Server deposits, and acts as a root of trust. If a key is revoked on the HSM, any Key Trustee Server deposits encrypted with that key are rendered irretrievable.

The following diagram demonstrates the flow of storing a deposit in Key Trustee Server when Key HSM is used:



1. A Key Trustee client (for example, Navigator Encrypt or Key Trustee KMS) sends an encrypted secret to Key Trustee Server.
2. Key Trustee Server forwards the encrypted secret to Key HSM.
3. Key HSM generates a symmetric encryption key and sends it to the HSM over an encrypted channel.

Cloudera Navigator Data Encryption Overview

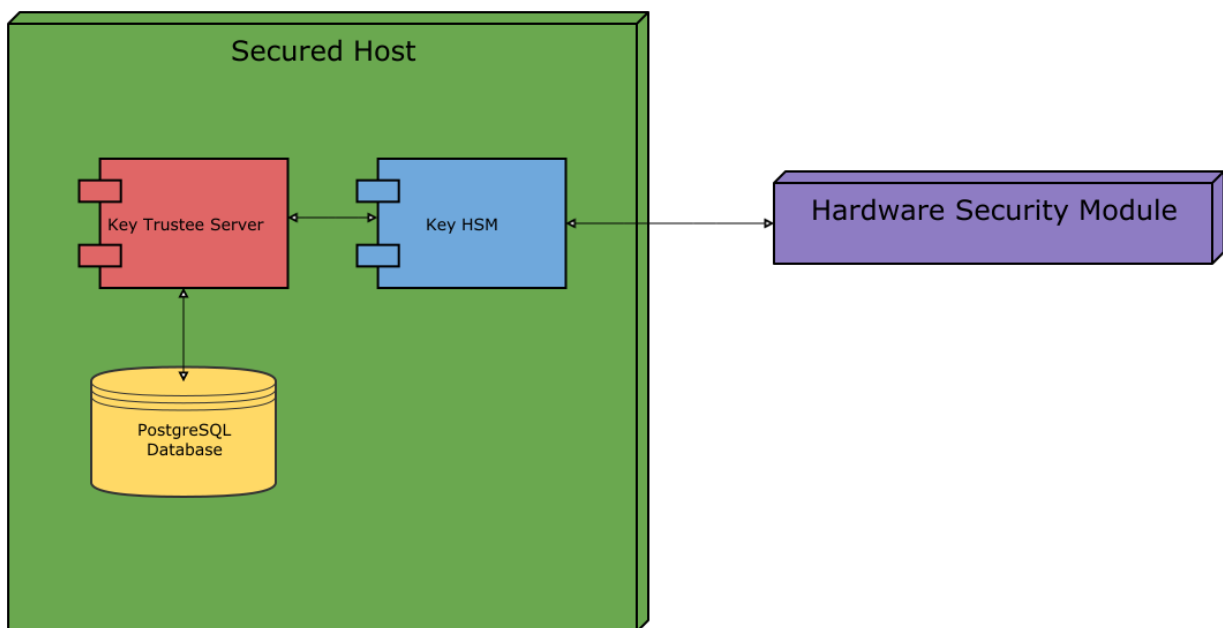
4. The HSM generates a new key pair and encrypts the symmetric key and returns the encrypted symmetric key to Key HSM.
5. Key HSM encrypts the original client-encrypted secret with the symmetric key, and returns the twice-encrypted secret, along with the encrypted symmetric key, to Key Trustee Server. Key HSM discards its copy of the symmetric key.
6. Key Trustee Server stores the twice-encrypted secret along with the encrypted symmetric key in its PostgreSQL database.

The only way to retrieve the original encrypted secret is for Key HSM to request the HSM to decrypt the encrypted symmetric key, which is required to decrypt the twice-encrypted secret. If the key has been revoked on the HSM, it is not possible to retrieve the original secret.

Key HSM Architecture

For increased security, Key HSM should always be installed on the same host running the Key Trustee Server. This reduces the attack surface of the system by ensuring that communication between Key Trustee Server and Key HSM stays on the same host, and never has to traverse a network segment.

The following diagram displays the recommended architecture for Key HSM:



For instructions on installing Navigator Key HSM, see [Installing Cloudera Navigator Key HSM](#). For instructions on configuring Navigator Key HSM, see [Initializing Navigator Key HSM](#).

Cloudera Navigator Encrypt Overview

Cloudera Navigator Encrypt transparently encrypts and secures data at rest without requiring changes to your applications and ensures minimal performance lag in the encryption or decryption process. Advanced key management with [Cloudera Navigator Key Trustee Server](#) and process-based access controls in Navigator Encrypt enable organizations to meet compliance regulations and prevent unauthorized parties or malicious actors from gaining access to encrypted data.

For instructions on installing Navigator Encrypt, see [Installing Cloudera Navigator Encrypt](#). For instructions on configuring Navigator Encrypt, see [Registering Cloudera Navigator Encrypt with Key Trustee Server](#).

Navigator Encrypt features include:

- Automatic key management: Encryption keys are stored in Key Trustee Server to separate the keys from the encrypted data. If the encrypted data is compromised, it is useless without the encryption key.
- Transparent encryption and decryption: Protected data is encrypted and decrypted seamlessly, with minimal performance impact and no modification to the software accessing the data.
- Process-based access controls: Processes are authorized individually to access encrypted data. If the process is modified in any way, access is denied, preventing malicious users from using customized application binaries to bypass the access control.
- Performance: Navigator Encrypt supports the Intel AES-NI cryptographic accelerator for enhanced performance in the encryption and decryption process.
- Compliance: Navigator Encrypt enables you to comply with requirements for HIPAA-HITECH, PCI-DSS, FISMA, EU Data Protection Directive, and other data security regulations.
- Multi-distribution support: Navigator Encrypt supports Debian, Ubuntu, RHEL, CentOS, and SLES.
- Simple installation: Navigator Encrypt is distributed as RPM and DEB packages, as well as SLES KMPs.
- Multiple mountpoints: You can separate data into different mountpoints, each with its own encryption key.

Navigator Encrypt can be used with many kinds of data, including (but not limited to):

- Databases
- Temporary files (YARN containers, spill files, and so on)
- Log files
- Data directories
- Configuration files

Navigator Encrypt uses `dmccrypt` for its underlying cryptographic operations. Navigator Encrypt uses several different encryption keys:

- Master Key: The master key can be a single passphrase, dual passphrase, or RSA key file. The master key is stored in Key Trustee Server and cached locally. This key is used when registering with a Key Trustee Server and when performing administrative functions on Navigator Encrypt clients.
- Mount Encryption Key (MEK): This key is generated by Navigator Encrypt using `openssl rand` by default, but it can alternatively use `/dev/urandom`. This key is generated when preparing a new mount point. Each mount point has its own MEK. This key is uploaded to Key Trustee Server.
- `dmccrypt` Device Encryption Key (DEK): This key is not managed by Navigator Encrypt or Key Trustee Server. It is managed locally by `dmccrypt` and stored in the header of the device.

Process-Based Access Control List

The access control list (ACL) controls access to specified data. The ACL uses a process fingerprint, which is the SHA256 hash of the process binary, for authentication. You can create rules to allow a process to access specific files or directories. The ACL file is encrypted with the client master key and stored locally for quick access and updates.

Here is an example rule:

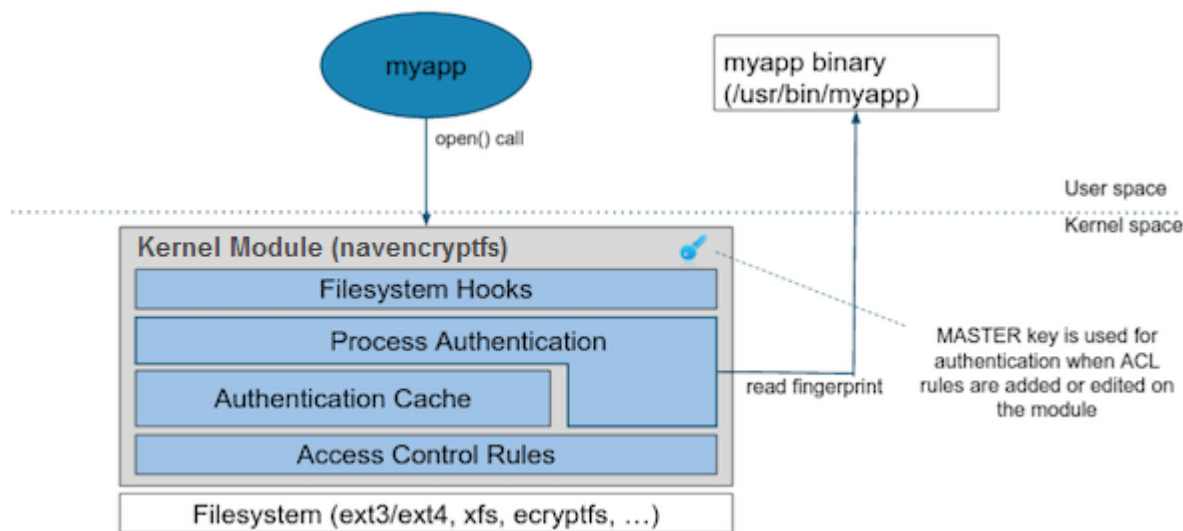
```
"ALLOW @mydata * /usr/bin/myapp"
```

This rule allows the `/usr/bin/myapp` process to access any encrypted path (*) that was encrypted under the category `@mydata`.

Navigator Encrypt uses a kernel module that intercepts any input/output (I/O) sent to an encrypted and managed path. The Linux module filename is `navencryptfs.ko` and it resides in the kernel stack, injecting filesystem hooks. It also authenticates and authorizes processes and caches authentication results for increased performance.

Because the kernel module intercepts and does not modify I/O, it supports any filesystem (`ext3`, `ext4`, `xfs`, and so on).

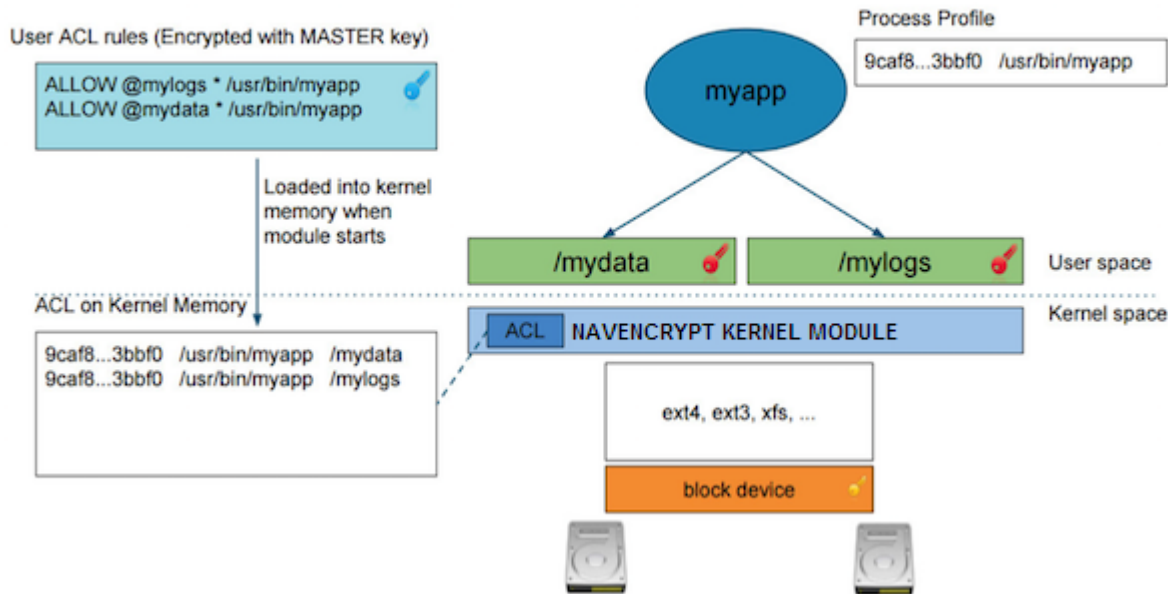
The following diagram shows `/usr/bin/myapp` sending an `open()` call that is intercepted by `navencrypt-kernel-module` as an `open` hook:



The kernel module calculates the process fingerprint. If the authentication cache already has the fingerprint, the process is allowed to access the data. If the fingerprint is not in the cache, the fingerprint is checked against the ACL. If the ACL grants access, the fingerprint is added to the authentication cache, and the process is permitted to access the data.

When you add an ACL rule, you are prompted for the master key. If the rule is accepted, the ACL rules file is updated as well as the `navencrypt-kernel-module` ACL cache.

The next diagram illustrates different aspects of Navigator Encrypt:



The user adds a rule to allow `/usr/bin/myapp` to access the encrypted data in the category `@mylogs`, and adds another rule to allow `/usr/bin/myapp` to access encrypted data in the category `@mydata`. These two rules are loaded into the `navencrypt-kernel-module` cache after restarting the kernel module.

The `/mydata` directory is encrypted under the `@mydata` category and `/mylogs` is encrypted under the `@mylogs` category using `dmccrypt` (block device encryption).

When `myapp` tries to issue I/O to an encrypted directory, the kernel module calculates the fingerprint of the process (`/usr/bin/myapp`) and compares it with the list of authorized fingerprints in the cache.

Encryption Key Storage and Management

The master key and mount encryption keys are securely deposited in Key Trustee Server. One MEK per mount point is stored locally for offline recovery and rapid access. The locally-stored MEKs are encrypted with the master key.

The connection between Navigator Encrypt and Key Trustee Server is secured with TLS/SSL certificates.

The following diagram demonstrates the communication process between Navigator Encrypt and Key Trustee Server:



The master key is encrypted with a local GPG key. Before being stored in the Key Trustee Server database, it is encrypted again with the Key Trustee Server GPG key. When the master key is needed to perform a Navigator Encrypt operation, Key Trustee Server decrypts the stored key with its server GPG key and sends it back to the client (in this case, Navigator Encrypt), which decrypts the deposit with the local GPG key.

All communication occurs over TLS-encrypted connections.

Cloudera Navigator Optimizer

Cloudera Navigator Optimizer profiles and analyzes the query text in SQL workloads. This analysis helps you gain in-depth understanding of your workloads, identify and offload queries best-suited for Hadoop, and optimize your workloads already running on Hive or Impala.

Use Navigator Optimizer to

- **Identify workloads to move to Hadoop:** Offloading non-operational enterprise data warehouse (EDW) workloads to Hadoop can help you increase the efficiency of your EDW system.
- **Optimize workloads already running on Hadoop:** Use Navigator Optimizer to see what is happening with your workloads running on Hadoop and get recommendations for optimizing them.
- **Slice and dice workloads by custom attributes:** Add custom attribute information to your query workload file and Navigator Optimizer can pivot its analysis based on those custom attributes. For example, you can identify data access patterns for specific users, applications, or reports.
- **Get risk alerts based on Hive and Impala best practices:** Before you offload a workload to Hive or Impala, Navigator Optimizer can assess the risk based on compatibility and complexity of these queries and suggest fixes for them.

For more information about Navigator Optimizer, see www.cloudera.com/products/cloudera-navigator-optimizer.html

Frequently Asked Questions About Cloudera Software

The following topics contain frequently asked questions about components and subsystems of the Cloudera Enterprise product:

- [Cloudera Manager 5 Frequently Asked Questions](#)
- [Cloudera Navigator Frequently Asked Questions](#)
- [Impala Frequently Asked Questions](#)
- [Cloudera Search Frequently Asked Questions](#)

Getting Support

This section describes how to get support.

Cloudera Support

Cloudera can help you install, configure, optimize, tune, and run CDH for large scale data processing and analysis. Cloudera supports CDH whether you run it on servers in your own datacenter or on hosted infrastructure services, such as Amazon Web Services, Microsoft Azure, or Google Compute Engine.

If you are a Cloudera customer, you can:

- Register for an account to create a support ticket at the [support site](#).
- Visit the [Cloudera Knowledge Base](#).

If you are not a Cloudera customer, learn how [Cloudera](#) can help you.

Information Required for Logging a Support Case

Before you log a support case, ensure you have either part or all of the following information to help Support investigate your case:

- If possible, provide a diagnostic data bundle following the instructions in [Collecting and Sending Diagnostic Data to Cloudera](#).
- For security issues, see [How to Log a Security Support Case](#).
- Provide details about the issue such as what was observed and what the impact was.
- Provide any error messages that were seen, using screen capture if necessary & attach to the case.
- If you were running a command or performing a series of steps, provide the commands and the results, captured to a file if possible.
- Specify whether the issue took place in a new install or a previously-working cluster.
- Mention any configuration changes made in the follow-up to the issue being seen.
- Specify the type of release environment the issue is taking place in, such as sandbox, development, or production.
- The severity of the impact and whether it is causing outage.

Community Support

There are several vehicles for community support. You can:

- Register for the [Cloudera forums](#).
- If you have any questions or comments about CDH, you can visit the [Using the Platform forum](#).
- If you have any questions or comments about Cloudera Manager, you can
 - Visit the [Cloudera Manager forum](#) forum.
 - Cloudera Express users can access the Cloudera Manager support mailing list from within the Cloudera Manager Admin Console by selecting **Support** > **Mailing List**.
 - Cloudera Enterprise customers can access the [Cloudera Support Portal](#) from within the Cloudera Manager Admin Console, by selecting **Support** > **Cloudera Support Portal**. From there you can register for a support account, create a support ticket, and access the Cloudera Knowledge Base.
- If you have any questions or comments about Cloudera Navigator, you can visit the [Cloudera Navigator forum](#).
- Find more documentation for specific components by referring to [External Documentation](#) on page 39.

Get Announcements about New Releases

To get information about releases and updates for all products, visit the [Release Announcements](#) forum.

Report Issues

Your input is appreciated, but before filing a request:

- Search the [Cloudera issue tracker](#), where Cloudera tracks software and documentation bugs and enhancement requests for CDH.
- Search the [CDH Manual Installation](#), [Using the Platform](#), and [Cloudera Manager](#) forums.