cloudera®

# Cloudera Enterprise Reference Architecture for Bare Metal Deployments

## Important Notice

© 2010-2018 Cloudera, Inc. All rights reserved.

Cloudera, the Cloudera logo, and any other product or service names or slogans contained in this document, except as otherwise disclaimed, 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 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.

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.**
395 Page Mill Road
Palo Alto, CA 94306
info@cloudera.com
US: 1-888-789-1488
Intl: 1-650-362-0488
www.cloudera.com

**Release Information**
Version: 5.14
Date: 2018-01-26

# Table of Contents

# Abstract

An organization's requirements for a big-data solution are simple: acquire and combine any amount or type of data in its original fidelity, in one place, for as long as necessary, and deliver insights to all kinds of users, as quickly as possible.

Cloudera, an enterprise data management company, introduced the concept of the enterprise data hub (EDH): a central system to store and work with all data. The EDH has the flexibility to run a variety of enterprise workloads (for example, batch processing, interactive SQL, enterprise search, and advanced analytics) while meeting enterprise requirements such as integrations to existing systems, robust security, governance, data protection, and management. The EDH is the emerging center of enterprise data management. EDH builds on Cloudera Enterprise, which consists of the open source Cloudera Distribution including Apache Hadoop (CDH), a suite of management software and enterprise-class support.

As organizations embrace Hadoop-powered big data deployments, they also want enterprise-grade security, management tools, and technical support--all of which are part of Cloudera Enterprise.

Cloudera Reference Architecture documents illustrate example cluster configurations and certified partner products. The RAs are not replacements for official statements of supportability, rather they're guides to assist with deployment and sizing options. Statements regarding supported configurations in the RA are informational and should be cross-referenced with the latest documentation.

This document is a high-level design and best-practices guide for deploying Cloudera Enterprise on bare metal.

# Infrastructure

## System Architecture Best Practices

This section describes Cloudera's recommendations and best practices applicable to Hadoop cluster system architecture.

### Java

Cloudera Manager and CDH are certified to run on Oracle JDK. At this time OpenJDK is not supported. Cloudera distributes a compatible version of the Oracle JDK through the Cloudera Manager repository. Customers are also free to install a compatible version of the Oracle JDK distributed by Oracle.

Refer to CDH and Cloudera Manager Supported JDK Versions for a list of supported JDK versions.

### Right-size Server Configurations

Cloudera recommends deploying three or four machine types into production:

- **Master Node.** Runs the Hadoop master daemons: NameNode, Standby NameNode, YARN Resource Manager and History Server, the HBase Master daemon, Sentry server, and the Impala StateStore Server and Catalog Server. Master nodes are also the location where Zookeeper and JournalNodes are installed. The daemons can often share single pool of servers. Depending on the cluster size, the roles can instead each be run on a dedicated server. Kudu Master Servers should also be deployed on master nodes.
- **Worker Node.** Runs the HDFS DataNode, YARN NodeManager, HBase RegionServer, Impala impalad, Search worker daemons and Kudu Tablet Servers.
- **Utility Node.** Runs Cloudera Manager and the Cloudera Management Services. It can also host a MySQL (or another supported) database instance, which is used by Cloudera Manager, Hive, Sentry and other Hadoop-related projects.
- **Edge Node.** Contains all client-facing configurations and services, including gateway configurations for HDFS, YARN, Impala, Hive, and HBase. The edge node is also a good place for Hue, Oozie, HiveServer2, and Impala HAProxy. HiveServer2 and Impala HAProxy serve as a gateway to external applications such as Business Intelligence (BI) tools.

For more information refer to Cluster Hosts and Role Assignments.

> **Note:**
> The edge and utility nodes can be combined in smaller clusters.

### General Cluster Architecture

This section contains general guidelines for cluster layout, assuming the usage of HDFS, YARN, Hive, Impala, Hue, Oozie, and ZooKeeper.

In this reference architecture, we consider different kinds of workloads that are run on top of an Enterprise Data Hub. The initial requirements focus on host types that are suitable for a diverse set of workloads. As service offerings change, these requirements may change to specify host types that are unique to specific workloads. You choose host types based on the workload you run on the cluster. You should also do a cost-performance analysis.

***Small Cluster (up to 20 worker nodes)***

- 2 Master Nodes
    - HDFS NameNode (2, HA)
    - HDFS JournalNode (3)
    - HDFS FailoverController (2, collocated with NN services)
    - YARN Resource Manager (2, HA)
    - YARN History Server
    - HBase Master (2, HA)
    - Sentry Service
    - ZooKeeper (2)
    - Kudu Master Server (2)
- 1 Utility/Edge Node
    - Cloudera Manager
    - Cloudera Management Services
    - HDFS JournalNode
    - Hive Metastore Server
    - HiveServer2
    - HBase Thrift or REST Server
    - Impala Catalog Server
    - Impala StateStore Server
    - Hue
    - Oozie
    - Flume Agent
    - Database server
    - Gateway (client) configurations
    - ZooKeeper Node[1]
    - Kudu Master Server (1)
- 20 Worker Nodes
    - HDFS DataNode
    - YARN NodeManager
    - HBase Region Server
    - Impalad
    - Kudu Tablet Server

***Medium Cluster (up to 200 worker nodes)***

- 3 Master Nodes
    - HDFS NameNode (2, HA)
    - HDFS JournalNode (3)
    - HDFS FailoverController (2, collocated with NN services)
    - YARN Resource Manager (2, HA)
    - YARN History Server
    - HBase Master (3, HA)
    - Sentry Service
    - ZooKeeper (3)

[1] ZooKeeper Node and JournalNode should have their own dedicated spindle in this configuration.

    - Kudu Master Servers (3)
- 2 Utility Nodes
    - Cloudera Manager (node 1)
    - Cloudera Management Services (node 2)
    - Hive Metastore Server (node 1)
    - Impala Catalog Server (node 1)
    - Impala StateStore Server (node 1)
    - Oozie (node 1)
    - RDBMS servers (both)
- 1+ Edge Nodes
    - Hue
    - HiveServer2
    - HBase Thrift or REST Server
    - Flume Agent
    - Gateway (Client) configurations
- 200 Worker Nodes
    - HDFS DataNode
    - YARN NodeManager
    - HBase Region Server
    - Impalad
    - Kudu Tablet Server

***Large Cluster (up to 500 worker nodes)***

- 5 Master Nodes
    - HDFS NameNode (2, HA)
    - HDFS JournalNode (5)
    - HDFS FailoverController (collocated with NN services)
    - YARN Resource Manager (2, HA)
    - HBase Master (5, HA)
    - Sentry Service
    - ZooKeeper Node (5)
    - Kudu Master Server (5)
- 2 Utility Nodes
    - Cloudera Manager (node 1)
    - Cloudera Management Services (node 2)
    - Hive Metastore Server (node 1)
    - Impala Catalog Server (node 1)
    - Impala StateStore Server (node 1)
    - Oozie (node 1)
    - Database server (both)
- 1+ Edge Nodes
    - Hue
    - HiveServer2
    - HBase Thrift or REST Server
    - Flume Agent
    - Gateway (Client) configurations
- 500 Worker Nodes
    - HDFS DataNode

    - YARN NodeManager
    - HBase Region Server
    - Impalad
    - Kudu Tablet Server

### *Very Large Cluster (500+ worker nodes)*

Very Large clusters have a layout similar to Large clusters but require detailed environment and workload analysis and custom configuration. Contact Cloudera for additional information.

### *Encryption Infrastructure (for all cluster sizes)*

- 4 Utility nodes
    - Key Management Server (2)
    - Key Trustee Server (2)

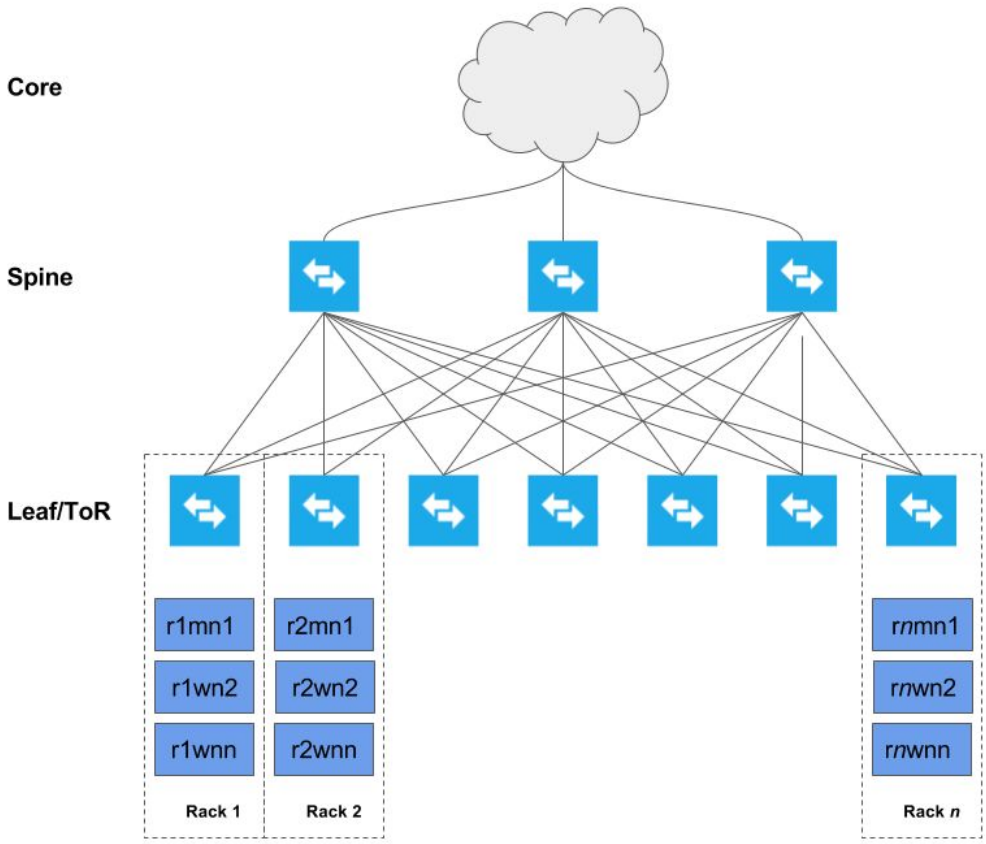
Further detail can be found in the Encryption discussion later in this document.

> **Note:**
> Edge nodes in the medium and large cluster architectures are mostly driven by use cases. Complex ingest pipelines and/or lots of client access will require more edge nodes to handle the load.

> **Note:**
> Without three master servers Kudu does not provide HA. Five masters can be utilized; in this case the loss of 2 Kudu Master Servers will be tolerated.

## Deployment Topology

The graphic below depicts a cluster deployed across several racks (Rack 1, Rack 2, … Rack *n*).




Each host is networked to two top-of-rack (TOR) switches which are in turn connected to a collection of spine switches which are then connected to the enterprise network. This deployment model allows each host maximum throughput, minimize of latency, while encouraging scalability. The specifics of the network topology are described in the subsequent sections.

## Physical Component List

The follow table describes the physical components recommended to deploy an EDH cluster.

| Component | Configuration | Description | Quantity |
|---|---|---|---|
| Physical servers | Two-socket, 8-14 cores per socket > 2 GHz; minimally 128 GB RAM. | Hosts that house the various cluster components. | Based on cluster design. |
| NICs | 10 Gbps Ethernet NICs preferred. | Provide the data network services for the cluster. | At least one per server, although two NICs can be bonded |

| | | | |
|---|---|---|---|
| | | | for additional throughput. |
| Internal HDDs | 500 GB HDD or SSD recommended for operating system and logs; HDD for data disks (size varies with data volume requirements) | These ensure continuity of service on server resets and contain the cluster data. | 6-24 disks per physical server. |
| Ethernet ToR/leaf switches | Minimally 10 Gbps switches with sufficient port density to accommodate the cluster. These require enough ports to create a realistic spine-leaf topology providing ISL bandwidth above a 1:4 oversubscription ratio (preferably 1:1). | Although most enterprises have mature data network practices, consider building a dedicated data network for the Hadoop cluster. | At least two per rack. |
| Ethernet spine switches | Minimally 10 Gbps switches with sufficient port density to accommodate incoming ISL links and ensure required throughput over the spine (for inter-rack traffic). | Same considerations as for ToR switches. | Depends on the number of racks. |

## Network Specification

### *Dedicated Network Hardware*

Hadoop can consume all available network bandwidth. For this reason, Cloudera recommends that Hadoop be placed in a separate physical network with its own core switch.

### *Switch Per Rack*

Hadoop supports the concept of rack locality and takes advantage of the network topology to minimize network congestion. Ideally, nodes in one rack should connect to a single physical switch. Two top- of-rack (TOR) switches can be used for high availability. Each rack switch (i.e. TOR switch) uplinks to a core switch with a significantly bigger backplane. Cloudera recommends 10 GbE (or faster) connections between the servers and TOR switches. TOR uplink bandwidth to the core switch (two switches in a HA configuration) will often be oversubscribed to some extent.

### *Uplink Oversubscription*

How much oversubscription is appropriate is workload dependent. Cloudera's recommendation is that the ratio between the total access port bandwidth and uplink bandwidth be as close to 1:1 as is possible.

This is especially important for heavy ETL workloads, and MapReduce jobs that have a lot of data sent to reducers.

Oversubscription ratios up to 4:1 are generally fine for balanced workloads, but network monitoring is needed to ensure uplink bandwidth is not the bottleneck for Hadoop. The following table provides some examples as a point of reference:

| Access Port Bandwidth (In Use) | Uplink Port Bandwidth (Bonded) | Ratio |
|---|---|---|
| 48 x 1 GbE = 48 Gbit/s | 4 x 10 GbE = 40 Gbit/s | 1.2:1 |
| 24 x 10 GbE = 240 Gbit/s | 2 x 40 Gig CFP = 80 Gbit/s | 3:1 |
| 48 x 10 GbE = 480 Gbit/s | 4 x 40 Gig CFP = 160 Gbit/s | 3:1 |

**Important:**

Do not exceed 4:1 oversubscription ratio. For example, if a TOR has 20 x 10 GbE ports used, the uplink should be at least 50 Gbps. Different switches have dedicated uplink ports of specific bandwidth (often 40 Gbps or 100 Gbps) and therefore careful planning needs to be done in order to choose the right switch types.

### *Redundant Network Switches*

Having redundant core switches in a full mesh configuration will allow the cluster to continue operating in the event of a core switch failure. Redundant TOR switches will prevent the loss of an entire rack of processing and storage capacity in the event of a TOR switch failure. General cluster availability can still be maintained in the event of the loss of a rack, as long as master nodes are distributed across multiple racks.

### *Accessibility*

The accessibility of your Cloudera Enterprise cluster is defined by the network configuration and depends on the security requirements and the workload. Typically, there are edge/client nodes that have direct access to the cluster. Users go through these edge nodes via client applications to interact with the cluster and the data residing there. These edge nodes could be running a web application for real-time serving workloads, BI tools, or simply the Hadoop command-line client used to submit or interact with HDFS.

Cloudera recommends allowing access to the Cloudera Enterprise cluster via edge nodes only. You can configure this in the security groups for the hosts that you provision. The rest of this document describes the various options in detail.

### *Internet Connectivity*

Clusters that do not require heavy data transfer between the Internet or services outside of the immediate network and HDFS still might need access to services like software repositories for updates or other low-volume outside data sources.

If you completely disconnect the cluster from the Internet, you block access for software updates which makes maintenance difficult.

### Cloudera Manager

Cloudera strongly recommends installing CDH using Cloudera Manager (CM). During the CDH installation via CM there is the choice to install using parcels or native packages. A parcel is a binary distribution format. Parcels offer a number of benefits including consistency, flexible installation location, installation without sudo, reduced upgrade downtime, rolling upgrades, and easy downgrades. Cloudera recommends using parcels, though using packages is supported.

## Cluster Sizing Best Practices

Each worker node typically has several physical disks dedicated to raw storage for Hadoop. This number will be used to calculate the total available storage for each cluster. Also, the calculations listed below assume 10% disk space allocated for YARN temporary storage. Cloudera recommends allocating between 10-25% of the raw disk space for temporary storage as a general guideline. This can be changed within Cloudera Manager and should be adjusted after analyzing production workloads. For example, MapReduce jobs that send little data to reducers allow for adjusting this number percentage down considerably.

The following table contains example calculations for a cluster that contains 17 worker nodes. Each server has twelve 3 TB drives available for use by Hadoop. The table below outlines the Hadoop storage available based upon the number of worker nodes:

| | |
|---|---|
| Raw Storage | 612 TB |
| HDFS Storage (Configurable) | 550.8 TB |
| HDFS Unique Storage (default replication factor) | 183.6 TB |
| MapReduce Intermediate Storage (Configurable) | 61.2 TB |

**Note:**

Compressing raw data can effectively increase HDFS storage capacity.

While Cloudera Manager provides tools such as Static Resource Pools, which utilize Linux Cgroups, to allow multiple components to share hardware, in high volume production clusters it can be beneficial to allocate dedicated hosts for roles such as Solr, HBase, and Kafka.

## Cluster Hardware Selection Best Practices

This section will give a high level overview of how different hardware component selections will impact the performance of a Hadoop cluster.

Refer to the Hardware Requirements Guide for detailed workload-specific practices.

### Number of Spindles

Traditionally Hadoop has been thought of as a large I/O platform. While there are many new types of workloads being run on Cloudera clusters that may not be as I/O bound as traditional MapReduce applications, it is still useful to consider the I/O performance when architecting a Cloudera cluster.

Unlike the number of cores in a CPU and the density of RAM, the speed at which data can be read from a spinning hard drive (spindle) has not changed much in the last 10 years[2]. In order to counter the limited performance of hard drive read/write operations, Hadoop reads and writes from many drives in parallel. Every additional spindle added per node increases the overall read/write speed of the cluster.

Additional spindles also come with the likelihood of more network traffic in the cluster. For the majority of cases, network traffic between nodes is generally limited by how fast data can be written to or read from a node. Therefore, the rule normally follows that with more spindles network speed requirements increase.

Generally speaking, the more spindles a node has, the lower the cost per TB. However, a larger the quantity of data stored on one node yields a longer re-replication time if that node goes down. Hadoop clusters are designed to have many nodes. It is generally better to have more average nodes than fewer super nodes. This has a lot to do with both data protection, as well as increased parallelism for distributed computation engines such as MapReduce and Spark.

Lastly, the number of drives per node will impact the number of YARN containers configured for a node. YARN configuration and performance tuning is a complicated topic, but for I/O bound applications, the number of physical drives per host may be a limiting factor in determining the number of container slots configured per node.

Kafka clusters are often run on dedicated servers that do not run HDFS data nodes or processing components such as YARN and Impala. Because Kafka is a message-based system, fast storage and network I/O is critical to performance. Although Kafka does persist messages to disk, it is not generally necessary to store the entire contents of a Kafka topic log on the Kafka cluster indefinitely. Kafka brokers should be configured at a minimum with dedicated spinning hard drives for the log data directories and can benefit from SSDs.

Kafka drives should also be configured as RAID 10 because the loss of a single drive on a Kafka broker will cause the broker to experience an outage.

### Disk Layout

For Master nodes, the following layout is recommended

- 2 x Disks (capacity at least 500GB) in RAID 1 (software or hardware) for OS and logs
- 4 x Disks (>= 1TB each) in RAID 10 for Database data (see Note)
- 2 x Disks (capacity at least 1 TB) in RAID 1 (software or hardware) for NameNode metadata
- 1 x Disk JBOD/RAID 0 for ZooKeeper (>= 1TB) (see Note)
    - ZooKeeper disks must be HDD, not SSD
- 1 x Disk JBOD/RAID 0 for Quorum JournalNode (>= 1TB)

**Note:**

[2] SSDs have dramatically changed the persistent storage performance landscape, but the price per GB of spinning disks is still significantly less than that of SSD storage. As SSDs come down in cost and technologies such as Intel's Optane™ enter the market, workloads may swing back towards being CPU bound. Most Cloudera customers are still deploying clusters that store data on spinning hard disks.

Ideally databases should be run on an external host rather than running on the master node(s).

**Note:**

If customer has experienced fsync delays and other I/O related issues with ZooKeeper, ZooKeeper's `dataDir` and `dataLogDir` can be configured to use separate disks. It's hard to determine ahead of time whether this will be necessary; even a small cluster can result in heavy ZooKeeper activity.

For Worker nodes, the following layout is recommended:

- 2x Disks (capacity at least 500GB) in RAID 1 (software or hardware) for OS and logs
- 15-24 SATA Disks JBOD mode (or as multiple single-drive RAID 0 arrays if using a RAID controller incapable of doing JBOD passthrough) no larger than 4 TB in capacity. If the RAID Controller has cache, use it for write caching (preferably with battery backup); disable read caching. Follow your hardware vendor's best practices where available.
- For higher performance profile, use 10K RPM SATA or faster SAS drives; these often have lower capacity but capacity considerations can be offset by adding more data nodes.

SAS drives are supported but typically do not provide significant enough performance or reliability benefits to justify the additional costs. Hadoop is designed to be fault-tolerant and therefore drive failure can easily be tolerated. In order to achieve a good price-point, SATA drives should typically be used.

RAID controllers should be configured to disable any optimization settings for the RAID 0 arrays.

### Data Density Per Drive

Hard drives today come in many sizes. Popular drive sizes are 1-4 TB although larger drives are becoming more common. When picking a drive size the following points need to be considered.

- **Lower Cost Per TB** – Generally speaking, the larger the drive, the cheaper the cost per TB, which makes for lower TCO.
- **Replication Storms** – Larger drives means drive failures will produce larger re-replication storms, which can take longer and saturate the network while impacting in-flight workloads.
- **Cluster Performance** – In general, drive size has little impact on cluster performance. The exception is when drives have different read/write speeds and a use case that leverages this gain. MapReduce is designed for long sequential reads and writes, so latency timings are generally not as important. HBase can potentially benefit from faster drives, but that is dependent on a variety of factors, such as HBase access patterns and schema design; this also implies acquisition of more nodes. Impala and Cloudera Search workloads can also potentially benefit from faster drives, but for those applications the ideal architecture is to maintain as much data in memory as possible.

Cloudera does not support exceeding 100 TB per data node. You could use 12 x 8 TB spindles or 24 x 4 TB spindles. Cloudera does not support drives larger than 8 TB.[3]

[3] Larger disks offer increased capacity but not increased I/O. Clusters with larger disks can easily result in capacities exceeding 100 TB per-worker, contributing to replication storms mentioned above. Clusters with larger disks that observe the 100 TB limit end up having fewer spindles which reduces HDFS throughput.

### Number of Cores and Multithreading

Other than cost there is no negative for buying more and better CPUs, however the ROI on additional CPU power must be evaluated carefully. Here are some points to consider:

- **Cluster Bottleneck** – In general, CPU resources (and lack thereof) do not bottleneck MapReduce and HBase. The bottleneck will almost always be drive and/or network performance. There are certainly exceptions to this, such as inefficient Hive queries. Other compute frameworks like Impala, Spark, and Cloudera Search may be CPU-bound depending on the workload.
- **Additional Cores/Threads** – Within a given MapReduce job, a single task will typically use one thread at a time. As outlined earlier, the number of slots allocated per node may be a function of the number of drives in the node. As long as there is not a huge disparity in the number of cores (threads) and the number of drives, it does not make sense to pay for additional cores. In addition, a MapReduce task is going to be I/O bound for typical jobs, thus a given thread used by the task will have a large amount of idle time while waiting for I/O response.
- **Clock Speed** – Because Cloudera clusters often begin with a small number of use cases and associated workloads and grow over time, it makes sense to purchase the fastest CPUs available. Actual CPU usage is use case and workload dependent; for instance, computationally intensive Spark jobs would benefit more from faster CPUs than I/O bound MapReduce applications.

**Important:**

Allocate two vCPUs for the operating system and other non-Hadoop use (although this amount may need to be higher if additional non-Hadoop applications are running on the cluster nodes, such as third-party active monitoring/alerting tools). The more services you are running, the more vCPUs will be required; you will need to use more capable hosts to accommodate these needs..

For worker nodes, a mid-range 12-14 core CPU running at 2.4-2.5 GHz would typically provide a good cost/performance tradeoff. For master nodes, a mid-range 8 core CPU with a slightly faster clock speed (e.g. 2.6 GHz) would suffice. Where available, Simultaneous Multi-Threading implementations should be enabled (for example Intel's HyperThreading). BIOS settings for CPU and memory should be set to Maximum Performance mode or equivalent.

Refer to the Hardware Requirements Guide for detailed workload-specific practices.

### RAM

More memory is always good and it is recommended to purchase as much as the budget allows. Applications such as Impala and Cloudera Search are often configured to use large amounts of heap, and a mixed workload cluster supporting both services should have sufficient RAM to allow all required services.

Refer to the Hardware Requirements Guide for detailed workload-specific practices.

**Important:**

Allocate at least 4 GB memory for the operating system and other non-Hadoop use (although this amount may need to be higher if additional non-Hadoop applications are running on the cluster nodes, such as third-party active monitoring/alerting tools). The more services you are running, the more memory will be required; you will need to use more capable hosts to accommodate these needs.

It is critical to performance that the total memory allocated to all Hadoop-related processes (including processes such as HBase) is less than the total memory on the node, taking into account the operating system and non-Hadoop processes. Oversubscribing the memory on a system can lead to the Linux kernel's out-of-memory process killer being invoked and important processes being terminated. It can also be harmful to performance to unnecessarily over-allocate memory to a Hadoop process as this can lead to long Java garbage collection pauses.

For optimum performance, one should aim to populate all of the memory channels available on the given CPUs. This may mean a greater number of smaller DIMMS, but this means that both memory and CPU are operating at their best performance. Confer with your hardware vendor for defining optimal memory configuration layout.

Whilst 128 GB RAM can be accommodated, this typically constrains the amount of memory allocated to services such as YARN and Impala, therefore reducing the query capacity of the cluster. A value of 256 GB would typically be recommended with higher values also possible.

### Power Supplies

Hadoop software is designed around the expectation that nodes will fail. Redundant hot-swap power supplies are not necessary for worker nodes, but should be used for master, utility, and edge nodes.

## Operating System Best Practices

Cloudera currently supports running the EDH platform on several Linux distributions. To receive support from Cloudera, a supported version of the operating system must be in use. The Requirements and Supported Versions guide lists the supported operating systems for each version of Cloudera Manager and CDH.

### Hostname Naming Convention

Cloudera recommends using a hostname convention that allows for easy recognition of roles and/or physical connectivity. This is especially important for easily configuring rack awareness within Cloudera Manager. Using a project name identifier, followed by the rack ID, the machine class, and a machine ID is an easy way to encode useful information about the cluster. For example:

```
acme-test-r01m01
```

This hostname would represent the ACME customer's test project, rack #1, master node #1.

### Hostname Resolution

Cloudera recommends using DNS for hostname resolution. The usage of `/etc/hosts` becomes cumbersome quickly, and routinely is the source of hard to diagnose problems. `/etc/hosts` should only contain an entry for 127.0.0.1, and `localhost` should be the only name that resolves to it. The machine name must not resolve to the 127.0.0.1 address. All hosts in the cluster must have forward and reverse

lookups be the inverse of each other for Hadoop to function properly. An easy test to perform on the hosts to ensure proper DNS resolution is to execute:

```
dig <hostname>
dig –x <ip_address_returned_from_hostname_lookup)
```

For example:

```
dig themis.apache.org
themis.apache.org.    1758  IN    A     140.211.11.105

dig -x 140.211.11.105
105.11.211.140.in-addr.arpa. 3513 IN       PTR   themis.apache.org.
```

This is the behavior we should see for every host in the cluster.

### Functional Accounts

Cloudera Manager and CDH make use of dedicated functional accounts for the associated daemon processes. By default these accounts are created as local accounts on every machine in the cluster that needs them if they do not already exist (locally or from a directory service, such as LDAP). The Requirements and Supported Versions guide includes a table showing the UNIX user and group associated with each service.

As of Cloudera Manager 5.3, it is also possible to install the cluster in single user mode where all services share a single service account. This feature is provided for customers who have policy requirements that prevent the use of multiple service accounts. Cloudera does not recommend the use of this feature unless the customer has this requirement, as CDH uses separate accounts to achieve proper security isolation and therefore removing this feature will reduce the overall security of the installation. Additional information about single user mode can be found in the Cloudera Installation and Upgrade manual: Configuring Single User Mode.

### Time

All machines in the cluster need to have the same time and date settings, including time zones. Use of the Network Time Protocol (NTP) is highly recommended. Many cluster services are sensitive to time (e.g. HBase, Kudu, ZooKeeper) and troubleshooting will be greatly eased if time is consistent across all hosts. Doing the following will enable the NTP daemon:

(RHEL/CentOS 6)

```
service ntpd start
chkconfig ntpd on
```

(RHEL/CentOS 7)

```
systemctl start ntpd.service
systemctl enable ntpd.service
```

**Note:**

Chrony may be preferred on newer operating systems.

### Name Service Caching

It is recommended that name service caching be enabled, particularly for clusters that use non-local Hadoop functional accounts, such as the `hdfs` and `yarn` users. This becomes critical in the case where the latter is combined with using Kerberos. Many difficult-to-diagnose problems can arise when name service lookups time out or fail during heavy cluster utilization. Doing the following will enable the Name Service Cache Daemon (nscd):

(RHEL/CentOS 6)

```
service nscd start
chkconfig nscd on
```

(RHEL/CentOS 7)

```
systemctl start nscd.service
systemctl enable nscd.service
```

If you are running Red Hat SSSD you will need to modify the nscd configuration to not cache passwd, group, or netgroup information.

### SELinux

Cloudera Enterprise (without Cloudera Navigator) is supported on platforms with Security-Enhanced Linux (SELinux) enabled, however we recommend SELinux be disabled on all machines in the Hadoop cluster until you get your cluster up and running.

The Linux command `getenforce` returns the status of SELinux.

SELinux can be disabled on RHEL/CentOS by editing `/etc/sysconfig/selinux` (RHEL/CentOS 6) Or `/etc/selinux/config` (RHEL/CentOS 7) and setting `SELINUX=disabled`. This change must be done as root (or with proper sudo access), and requires a reboot.

### IPv6

Hadoop does not support IPv6. IPv6 configurations should be removed, and IPv6-related services should be stopped.

### iptables

Cloudera recommends disabling host based firewalls on the cluster, at least until you get your cluster up and running. Many problems that are difficult to diagnose result from incorrect/conflicting iptables entries that interfere with normal cluster communication. Doing the following will disable iptables for both IPv4 and IPv6:

(RHEL/CentOS 6)

```
service iptables stop
service ip6tables stop
chkconfig iptables off
chkconfig ip6tables off
```

(RHEL/CentOS 7)

```
systemctl stop firewalld.service
systemctl disable firewalld.service
```

For those who must restrict access using host-based firewalls, refer to the list of ports used by Cloudera Manager, Cloudera Navigator, and CDH 5.

### Startup Services

As with any production server, unused services should be removed and/or disabled. Some example services that are on by default that are not needed by CDH are:

- bluetooth
- cups
- iptables
- ip6tables
- postfix[4]

This list is by no means exhaustive. To view the list of services that are configured to start during system startup, execute the following command:

(RHEL/CentOS 6)

```
chkconfig –list | grep on
```

(RHEL/CentOS 7)

```
systemctl list-unit-files --type service | grep active
```

### Process Memory

The memory on each node is allocated out to the various Hadoop processes. This predictability reduces the chance of Hadoop processes inadvertently running out of memory and so paging to disk, which in turn leads to severe degradation in performance. See the section on Kernel and OS Tuning for further information.

A minimum of 4 GB of memory should be reserved on all nodes for operating system and other non-Hadoop use. This amount may need to be higher if additional non-Hadoop applications are running on the cluster nodes, such as third party active monitoring/alerting tools.

[4] While not needed by CDH, postfix (or other MTA) may be required by other services to deliver generated notices/alerts from the system.

Memory requirements and allocation for Hadoop components are discussed in further detail in other sections of this document.

## Kernel and OS Tuning

The Cloudera EDH platform depends on a properly tuned underlying host operating system for optimal performance. Cloudera strongly suggests setting the vm.swappiness and transparent hugepage compaction kernel parameters. The Cloudera Administration manual has additional background information and suggested settings: Optimizing Performance in CDH.

### *Entropy*

Cryptographic operations require entropy to ensure randomness. The Cloudera Security guide explains how to check available entropy and how to ensure sufficient entropy is available: Entropy Requirements.

### *Networking Parameters*

The following parameters are to be added to `/etc/sysctl.conf` to optimize various network behaviors.

Disable TCP timestamps to improve CPU utilization (this is an optional parameter and will depend on your NIC vendor):

```
net.ipv4.tcp_timestamps=0
```

Enable TCP sacks to improve throughput:

```
net.ipv4.tcp_sack=1
```

Increase the maximum length of processor input queues:

```
net.core.netdev_max_backlog=250000
```

Increase the TCP max and default buffer sizes using `setsockopt()`:

```
net.core.rmem_max=4194304
net.core.wmem_max=4194304
net.core.rmem_default=4194304
net.core_wmem_default=4194304
net.core.optmem_max=4194304
```

Increase memory thresholds to prevent packet dropping:

```
net.ipv4.tcp_rmem="4096 87380 4194304"
net.ipv4.tcp_wmem="4096 65536 4194304"
```

Enable low latency mode for TCP:

```
net.ipv4.tcp_low_latency=1
```

Set the socket buffer to be divided evenly between TCP window size and application buffer:

```
net.ipv4.tcp_adv_win_scale=1
```

## Filesystems

In Linux there are several choices for formatting and organizing drives. That being said, only a few choices are optimal for Hadoop. First, in RHEL/CentOS, the Logical Volume Manager should never be used. It is not optimal and can lead to combining multiple drives into one logical disk, which is in complete contrast to how Hadoop manages fault tolerance across HDFS. Second, Cloudera recommends using an extent-based file system. This includes `ext3`, `ext4`, and `xfs`. Most new Hadoop clusters use the `ext4` filesystem by default. Red Hat Enterprise Linux 7 uses `xfs` as its default filesystem.

**Important:**

If utilizing Kudu be sure to ensure that filesystem hole punching is a capability of the filesystem. Hole Punching is the use of the `fallocate()` system call with the `FALLOC_FL_PUNCH_HOLE` option set. Newer versions of ext4 and xfs support Hole Punching. ext3 does not support Hole Punching and unpatched RHEL prior to 6.4 does not support this facility. Older versions of ext4 and xfs that do not support Hole Punching cause Kudu to fail to start as Kudu provides a pre-start test for this facility. Without Hole Punching support the block manager is unsafe to use as claimed blocks will never be released and ever more disk space will be consumed.

### *Filesystem Creation Options*

When creating ext4 filesystems for use with Hadoop **data volumes**, we recommend reducing the superuser block reservation from 5% to 1% for root (using the `-m1` option) as well as setting the following options:

- use one inode per 1 MB (largefile)
- minimize the number of super block backups (sparse_super)
- enable journaling (has_journal)
- use b-tree indexes for directory trees (dir_index)
- use extent-based allocations (extent)

Creating such an ext4 filesystem might look like this:

```
mkfs -t ext4 -m 1 -O -T largefile \
  sparse_super,dir_index,extent,has_journal /dev/sdb1
```

Creating an xfs filesystem might look like this:

```
mkfs -t xfs /dev/sdb1
```

**Note:**

When creating xfs filesystems, no special options are required.

### *Disk Mount Options*

HDFS by design is a fault tolerant file system. As such, all drives used by DataNode machines for data need to be mounted without the use of RAID. Furthermore, drives should be mounted in `/etc/fstab` using the `noatime` option (which also implies `nodiratime`). In case of SSD or flash also turn on TRIM by specifying the `discard` option when mounting.

In /etc/fstab ensure that the appropriate filesystems have the noatime mount option specified:

```
/dev/sda1    /            ext4    noatime                0 0
```

In order to enable TRIM, edit /etc/fstab and set mount option discard as well.

```
/dev/sdb1    /data        ext4    noatime,discard        0 0
```

### *Disk Mount Naming Convention*

For ease of administration, it is recommended to mount all of the disks on the DataNode machines with a naming pattern, for example:

```
/data1
/data2
/data3
/data4
/data5
/data6
```

# Cluster Configuration

This section contains information about how the cluster is configured, including Cloudera recommendations specific to the customer hardware being used, and some best practices and general recommendations for each Cloudera EDH service. This section is not an exhaustive description of every configuration, but rather a focus on important configurations and those that have been changed from the default setting.

## Teragen and Terasort Performance Baseline

The `teragen` and `terasort` benchmarking tools are part of the standard Apache Hadoop distribution and are included with the Cloudera distribution. In the course of a cluster installation or certification, Cloudera recommends running several `teragen` and `terasort` jobs to obtain a performance baseline for the cluster. The intention is not to demonstrate the maximum performance possible for the hardware or to compare with externally published results, as tuning the cluster for this may be at odds with actual customer operational workloads. Rather the intention is to run a real workload through YARN to functionally test the cluster as well as obtain baseline numbers that can be used for future comparison, such as in evaluating the performance overhead of enabling encryption features or in evaluating whether operational workloads are performance limited by the I/O hardware. Running benchmarks provides an indication of cluster performance and may also identify and help diagnose hardware or software configuration problems by isolating hardware components, such as disks and network, and subjecting them to a higher than normal load.

The `teragen` job will generate an arbitrary amount of data, formatted as 100-byte records of random data, and store the result in HDFS. Each record has a random key and value. The `terasort` job will sort the data generated by `teragen` and write the output to HDFS.

During the first iteration of the `teragen` job, the goal is to obtain a performance baseline on the disk I/O subsystem. The HDFS replication factor should be overridden from the default value 3 and set to 1 so that the data generated by `teragen` is not replicated to additional data nodes. Replicating the data over the network would obscure the raw disk performance with potential network bandwidth constraints.

Once the first `teragen` job has been run, a second iteration should be run with the HDFS replication factor set to the default value. This will apply a high load to the network, and deltas between the first run and second run can provide an indication of network bottlenecks in the cluster.

While the `teragen` application can generate any amount of data, 1 TB is standard. For larger clusters, it may be useful to also run 10 TB or even 100 TB, as the time to write 1 TB may be negligible compared to the startup overhead of the YARN job. Another `teragen` job should be run to generate a dataset that is 3 times the RAM size of the entire cluster. This will ensure we are not seeing page cache effects and are truly exercising the disk I/O subsystem.

The number of mappers for the `teragen` and `terasort` jobs should be set to the maximum number of disks in the cluster. This will likely be less than the total number of YARN `vcores` available, so it is advisable to temporarily lower the vcores available per YARN worker node to the number of disk spindles to ensure an even distribution of the workload. An additional `vcore` will be needed for the YARN ApplicationMaster.

The `terasort` job should also be run with the HDFS replication factor set to 1 as well as with the default replication factor. The `terasort` job hardcodes a DFS replication factor of 1, but it can be overridden or set explicitly by specifying the *mapreduce.terasort.output.replication* parameter as shown below.

### Teragen and Terasort Command Examples

***Teragen Command to Generate 1 TB of Data With HDFS Replication Set to 1***

```
EXAMPLES_PATH=/opt/cloudera/parcels/CDH/lib/hadoop-mapreduce

yarn jar ${EXAMPLES_PATH}/hadoop-mapreduce-examples.jar \
  teragen -Ddfs.replication=1 -Dmapreduce.job.maps=360 \
  10000000000 TS_input1
```

The above command generates 1 TB of data with an HDFS replication factor of 1, using 360 mappers. This command would be appropriate for a cluster with 360 disks.

***Teragen Command to Generate 1 TB of Data With HDFS Default Replication***

```
EXAMPLES_PATH=/opt/cloudera/parcels/CDH/lib/hadoop-mapreduce

yarn jar ${EXAMPLES_PATH}/hadoop-mapreduce-examples.jar \
  teragen -Dmapreduce.job.maps=360 \
  10000000000 TS_input2
```

The above command generates 1 TB of data with the default HDFS replication factor (usually 3), using 360 mappers. This command would be appropriate for a cluster with 360 disks.

***Terasort Command to Sort Data With HDFS Replication Set to 1***

```
EXAMPLES_PATH=/opt/cloudera/parcels/CDH/lib/hadoop-mapreduce

yarn jar ${EXAMPLES_PATH}/hadoop-mapreduce-examples.jar \
  terasort -Dmapreduce.terasort.output.replication=1 \
  -Dmapreduce.job.maps=360 \
  TS_input1 TS_output1
```

The above command sorts the data generated by `terasort` using 360 mappers and writes the sorted output to HDFS with a replication factor of 1. This would be appropriate for a cluster with 360 disks.

***Terasort Command to Sort Data With HDFS Replication Set to 3***

```
EXAMPLES_PATH=/opt/cloudera/parcels/CDH/lib/hadoop-mapreduce

yarn jar ${EXAMPLES_PATH}/hadoop-mapreduce-examples.jar \
  terasort -Dmapreduce.job.maps=360 \
  -Dmapreduce.terasort.output.replication=3 \
  TS_input2 TS_output2
```

The above command sorts the data generated by `terasort` using 360 mappers and writes the sorted output to HDFS with a replication factor of 3 (a typical default). This would be appropriate for a cluster with 360 disks.

***Teragen and Terasort Results***

| Command | HDFS Replication | Number of Mappers | Run Time |
|---|---|---|---|
| Teragen for 1 TB data set | 1 | | |
| Teragen for 3x cluster RAM data set | 1 | | |
| Terasort for 1 TB data set | 1 | | |
| Terasort for 3x cluster RAM data set | 1 | | |
| Teragen for 1 TB data set | 3 | | |
| Teragen for 3x cluster RAM data set | 3 | | |
| Terasort for 1 TB data set | 3 | | |
| Terasort for 3x cluster RAM data set | 3 | | |

# Cluster Configuration Best Practices

## ZooKeeper

ZooKeeper is extremely sensitive to disk latency. While it only uses a modest amount of resources, having ZooKeeper swap out or wait for a disk operation can result in that ZooKeeper node being considered 'dead' by its quorum peers. For this reason, Cloudera recommends against deploying ZooKeeper on worker nodes where loads are unpredictable and are prone to spikes. It is acceptable to deploy Zookeeper on master nodes where load is more uniform and predictable (or generally, on any node where it can be have unimpeded access to disk).

## HDFS

### *Java Heap Sizes*

NameNode memory should be increased over time as HDFS has more files and blocks stored. Cloudera Manager can monitor and alert on memory usage. A rough estimate is that the NameNode needs 1 GB of memory for every 1 million files. Setting the heap size too large when it is not needed leads to inefficient Java garbage collection, which can lead to erratic behavior that is hard to diagnose. NameNode and Standby NameNode heap sizes must always be the same, and thus must be adjusted together.

### *NameNode Metadata Locations*

When a quorum-based high availability HDFS configuration is used, JournalNodes handle the storage of metadata writes. The NameNode daemons require a local location to store metadata as well. Cloudera recommends that only a single directory be used if the underlying disks are configured as RAID, or two directories on different disks if the disks are mounted as JBOD.

### *Block Size*

HDFS stores files in blocks that are distributed over the cluster. A block is typically stored contiguously on disk to provide high read throughput. The choice of block size will influence how long these high throughput reads will run for, and over how many nodes a file is distributed. When reading the many blocks of a single file, a too low block size will spend more overall time in slow disk seek, and a too high block size will have reduced parallelism. Data processing that is I/O heavy will benefit from larger block sizes, and data processing that is CPU heavy will benefit from smaller block sizes.

The default provided by Cloudera Manager is 128MB. The block size can also be specified by an HDFS client on a per-file basis.

### *Replication*

Bottlenecks can occur on a small number of nodes when only small subsets of files on HDFS are being heavily accessed. Increasing the replication factor of the files so that their blocks are replicated over more nodes can alleviate this. This is done at the expense of storage capacity on the cluster. This can be set on individual files, or recursively on directories with the -R parameter, by using the Hadoop shell command `hadoop fs -setrep`. By default, the replication factor is three (3).

### *Rack Awareness*

Hadoop can optimize performance and redundancy when rack awareness is configured for clusters that span across multiple racks, and Cloudera recommends doing so. Rack assignments for nodes can be configured within Cloudera Manager.

When setting up a multi-rack environment, place each master node on a different rack. In the event of a rack failure, the cluster can continue to operate using the remaining master(s).

### *DataNode Failed Volumes Tolerated*

By default, Cloudera Manager sets the HDFS datanode failed volume threshold to half of the data drives in a datanode. In other words, if each datanode has eight drives dedicated to data storage, this threshold would be set to four, meaning that HDFS will mark the datanode dead on the fifth drive failure. This number may need to be adjusted up or down depending on internal policies regarding hard drive replacements, or because of evaluating what behavior is actually seen on the cluster under normal operating conditions. Setting the value too high will end up having a negative impact on the Hadoop cluster. Specifically for YARN, the number of total containers available on the node with many drive failures will still be the same as nodes without drive failures, meaning data locality is less likely on the former, leading to more network traffic and slower performance.

**Important:**

Multiple drive failures in a short amount of time might be indicative of a larger problem with the machine, such as a failed disk controller.

### *DataNode Xciever Count*

Xcievers are handlers in the datanode process that take care of sending and receiving block data. Some services, such as HBase, tend to use a lot of Xcievers. Cloudera Manager typically configures a sufficient xciever count, but it is possible that some workloads may require a higher setting.

### *Balancing*

HDFS will try to spread data evenly across the cluster in order to optimize read access, MapReduce performance, and node utilization. Over time it is possible that the cluster can become out of balance due to a variety of reasons. Hadoop can help mitigate this by rebalancing data across the cluster using the balancer tool. Running the balancer is a manual process that can be executed from within Cloudera Manager as well as from the command line. By default, Cloudera Manager configures the balancer to rebalance a datanode when its utilization is 10% more or less from the average utilization across the cluster. Individual datanode utilization can be viewed from within Cloudera Manager.

By default the maximum bandwidth a datanode will use for rebalancing is set to 10 MB/second (80 Mbit/second). This can be increased but network bandwidth used by rebalancing could potentially impact production cluster application performance. Changing the balancer bandwidth setting within Cloudera Manager requires a restart of the HDFS service, however this setting can also be made instantly across all nodes without a configuration change by running the command:

```
hdfs dfsadmin -setBalancerBandwidth <bytes_per_second>
```

This command must be run as an HDFS superuser. This is a convenient way to change the setting without restarting the cluster, but since it is a dynamic change, it does not persist if the cluster is restarted.

**Important:**

Cloudera generally does not recommend running the balancer on an HBase cluster as it affects data locality for the RegionServers, which can reduce performance. Unfortunately, when HBase and YARN services are collocated and heavy usage is expected on both, there is not a good way to ensure the cluster is optimally balanced.

## YARN

The YARN service manages MapReduce and Spark tasks. Applications run in YARN containers, which use Linux Cgroups for resource management and process isolation. The Cloudera Installation and Upgrade manual has a section on YARN tuning guidance.

## Impala

The Impala service is a distributed, MPP database engine for interactive performance SQL queries over large data sets. Impala performs best when it can operate on data in memory; as a result, Impala is often configured with a very large heap size.

Impala daemon processes must be collocated with HDFS data nodes to use HDFS local reads, which also improve performance.

Impala does not provide any built-in load balancing, so a production Impala deployment should sit behind a load balancer for performance and high availability. The Cloudera Impala product documentation contains detailed information on configuring Impala with a load balancer: Using Impala through a Proxy for High Availability.

The Cloudera Impala product documentation also contains a section on Impala performance tuning that should be reviewed prior to a production deployment: Tuning Impala for Performance.

### Spark

Cloudera supports Spark on YARN-managed deployments for a more flexible and consistent resource management approach. When running under Spark, the number of executors (YARN containers) can be specified when submitting the job. As of CDH 5.5, dynamic allocation is enabled by default but can be disabled either by setting `spark.dynamicAllocation.enabled=false`. If `--num-executors` is specified in the job, dynamic allocation is disabled implicitly. Additional information on Spark configuration and management is available in the Cloudera Administration manual: Managing Spark.

Spark standalone mode is not supported.

### HBase

#### *Automatic Major Compaction*

By default major compactions happen every 7 days. The next major compaction happens 7 days after the last one has finished. This means that the actual time that major compaction happens can impact production processes, which is not ideal if it is desired to run compactions at a specific known off-peak hour (e.g., 3 AM). Cloudera strongly recommends disabling automatic major compaction by setting the interval to zero (hbase.hregion.major.compaction = 0). Major compactions should then be run via cron calling the HBase admin tool.

### Search

Cloudera Search, based on Apache Solr, provides a distributed search engine service. Search engines are often expected to provide fast, interactive performance so it is important to allocate sufficient RAM to the search service.

If other resource intensive applications, such as Impala, are deployed on the same cluster, it's important to use the resource management facilities in Cloudera Manager. In some cases, it may also be preferable to avoid collocating the search service with other services.

### Oozie

Writing Oozie XML configuration files can be tedious and error-prone. Cloudera recommends using the Oozie editor provided by Hue for workflow creation, scheduling, and execution.

### Kafka

Kafka's default configuration with Cloudera Manager is suited to being able to start development quickly but several settings should be changed from the defaults before deploying a Cloudera Kafka cluster in production.

The default ZooKeeper Kafka root is `/`, however Cloudera recommends changing this to `/kafka`. This is the location within ZooKeeper where the znodes for the Kafka cluster are stored. As long as a single

Kafka cluster is using the ZooKeeper service, using `/kafka` is advised. If multiple Kafka clusters (eg. dev, test, qa) are sharing a ZooKeeper service, each Kafka instance should have a unique ZooKeeper root (eg. `/kafka-dev`, `/kafka-test`, `/kafka-qa`).

Cloudera Manager enables Kafka topic auto-creation by default. This means that any data written to a topic will cause the creation of that topic if it does not already exist. While this may be convenient in prototyping and development, it should not be used in production as it leads to cases where arbitrary topics can be created and data may be written to the wrong topic in the case of an improper application configuration without leading to an error.

Cloudera Manager sets the default minimum number of in sync replicas (ISR) to 1. This should generally be increased to a minimum of 2 in a production cluster to prevent data loss.

The Kafka Maximum Process File Descriptors setting may need to be increased in some cases in production deployments. This value can be monitored in Cloudera Manager and increased if usage requires a larger value than the default 64k ulimit.

The default data retention time for Kafka is often acceptable for production but should be reviewed for use case suitability.

## Flume

For Flume agents, use memory channel or file channel. Flume's memory channel offers increased performance at the cost of no data durability guarantees. File channels offer a higher level of durability guarantee because the data is persisted on disk in the form of files.

## Kudu

### *Limitations*

Current versions of Kudu come with a number of usage limitations:

- Kudu does not currently include any built-in features for backup and restore. Users are encouraged to use tools such as Spark or Impala to export or import tables as necessary.
- Kudu does not currently support rack awareness or rolling restarts.
- Kudu does not currently support multi-row transactions. Operations that affect multiple rows will not roll back if the operation fails part way through. This should be mitigated by exploiting the primary key uniqueness constraint to make operations idempotent

### *Impala Compatibility*

Lower versions of CDH (< 5.10) and Cloudera Manager used an experimental fork of Impala which is referred to as IMPALA_KUDU. If you have previously installed the IMPALA_KUDU service, make sure you remove it from your cluster before you proceed. Install Kudu 1.2.x (or later) using either Cloudera Manager or the command-line.

### *Partitioning Guidelines*

Kudu supports partitioning tables by RANGE and HASH partitions. Note that RANGE and HASH partitions can be combined to create more effective partitioning strategies. It is also possible to utilize non-covering RANGE partitions.

For large tables, such as fact tables, aim for as many tablets as you have cores in the cluster.

For small tables, such as dimension tables, aim for a large enough number of tablets that each tablet is at least 1 GB in size.

Further details can be found in the Apache Kudu Guide.

**Note:**

In general, be mindful the number of tablets limits the parallelism of reads, in the current implementation. Increasing the number of tablets significantly beyond the number of cores is likely to have diminishing returns.

# Security Integration

## Security Integration Best Practices

This section describes Cloudera's best practices and recommendations pertaining to security in EDH. It contains security fundamentals, integration techniques with external systems, and relevant links to Cloudera product documentation.

### Security Pillars

Enabling security across Cloudera EDH requires multiple facets of security. These facets are organized into pillars: Authentication, Authorization, Auditing, and Encryption. Each of these pillars individually help secure EDH, but only when used together are they the most effective in providing a comprehensive security stance to meet company standards and any compliance requirements.

#### *Authentication*

The first pillar of security is authentication. Authentication is the act of proving identity between two parties. Identity can be something as simple as a username, or something more complex such as a human being. Authenticating as an identity requires that both parties agree on a common source of identity information. For example, if a U.S. person possesses a driver's license as a form of identification, a police officer in another country may not recognize this as a valid source of identity.

Even though authentication is the act of proving identity, the level of proof required varies greatly. For example, showing a picture ID to someone, who glances at it to make sure the picture resembles you, is an example of weak authentication. The picture ID could easily have been forged, or the picture could be of someone else that looks like you. On the other hand, strong authentication is much more difficult to forge. For example, proving your identity by authenticating with fingerprints or facial recognition is far more accurate.

The use of strong authentication in EDH is a cornerstone of security in the platform.

#### *Authorization*

The next pillar of security is authorization. Authorization determines what an entity is allowed to do, and it happens after said entity has passed authentication. For example, if a person is traveling internationally, they must first provide a valid passport - authentication - then if all security checks pass, the passport control officer authorizes the person to enter the country.

This works the same way in EDH, with the simplest example being HDFS file access. In order to access a file in HDFS I must first authenticate to establish my identity, then the NameNode checks my identity and the permissions of the file to determine if authorization is granted.

Authorization comes in many different flavors, as described later.

#### *Auditing*

Auditing is a pillar that is critical for understanding what is happening in EDH. Without auditing, all of the other security pillars have limited effect because of a lack of visibility. Auditing keeps track of who is doing what on the cluster, which includes both positive events - actions that are successful and allowed, and negative events - actions that are unsuccessful and not allowed. Both positive and negative events are important to understand the complete picture of what is happening.

For EDH, Cloudera Navigator is the cornerstone of providing auditing capabilities. It is a necessary component for data governance.

### *Encryption*

The last pillar is encryption, which generally speaking relates to data protection. There are two styles of encryption: wire encryption and at-rest encryption. Wire encryption protects data while it is in transit over network channels and at-rest encryption protects data when it is persisted to disk. Both of these are necessary and should be considered as complementary security features.

## Kerberos

The Cloudera EDH platform relies on Kerberos as the standard strong authentication mechanism. Kerberos can be enabled for EDH using one of two supported KDC types: MIT Kerberos and Microsoft Active Directory.

### *MIT Kerberos*

The MIT Kerberos libraries are found as standard packages available for most Linux operating system variants. Installing and configuring a MIT Kerberos KDC is straightforward and well documented in the official MIT Kerberos installation guide.

In a Linux environment where identity integration is not with Active Directory, using MIT Kerberos is the best option for enabling Kerberos for Cloudera EDH.

### *Active Directory*

Microsoft Active Directory is the most widely used identity management system for businesses. It incorporates many different technologies with it, including certificate services, LDAP, and Kerberos. In a Linux environment where the operating system is integrated with Active Directory, Cloudera recommends to enable Kerberos with Active Directory and not to use MIT Kerberos. This is the most common Kerberos architecture to use for Cloudera EDH.

### *Other Kerberos Implementations*

There are other software suites that implement the Kerberos protocol, and which may be suitable choices. An example is RedHat IdM / CentOS FreeIPA. It is important to note that Cloudera does not test or provide support assistance with Kerberos implementations other than MIT and Active Directory.

### *Cloudera Manager Kerberos Wizard*

Cloudera Manager provides the ability to enable Kerberos for EDH using a step-by-step wizard. This wizard can be used for both MIT Kerberos and Active Directory configurations. Using automation eases the burden on administrators to create the many Kerberos principals and keytabs necessary for the cluster, as well as prevents errors that may occur as a result of incorrect account creation procedures.

Background information, requirements, and instructions for using the Cloudera Manager Kerberos wizard can be found in Enabling Kerberos Authentication Using the Wizard.

## LDAP

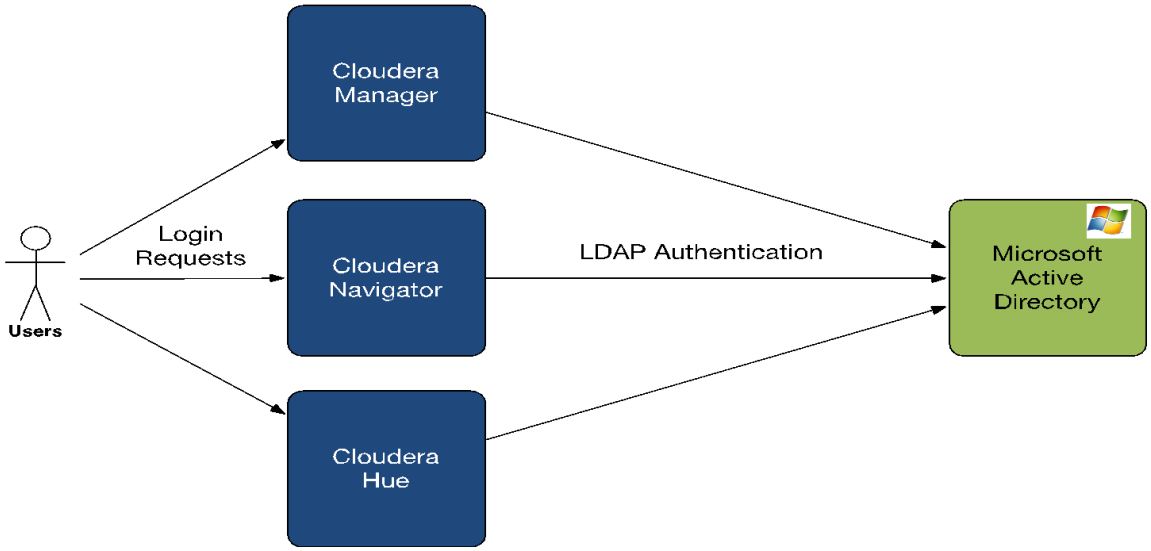
Linux systems out of the box utilize local users and groups for identities. For larger enterprises, a centralized identity database is required not only for ease of administration, but also for security and

traceability. The common widely accepted identity database and management protocol is LDAP. LDAP is an integral part of enterprise systems such as Microsoft Active Directory.

For enterprise deployments of EDH, Cloudera recommends integrating with LDAP.

### *LDAP Authentication*

While Kerberos is the standard for strong authentication in EDH, there are certain instances where it makes sense for users to authenticate using a username and password instead of providing a Kerberos ticket. Where this is most prevalent is for authentication when accessing ODBC/JDBC services like Hive and Impala, as well as authenticating users accessing web consoles such as Cloudera Manager, Cloudera Navigator, and Hue. The system diagram showing web console authentication using LDAP is as follows:




When enabling LDAP authentication, Cloudera strongly recommends using LDAP over TLS to ensure password information is not sent over the wire in the clear.

To enable LDAP authentication for Hive, consult HiveServer2 Security Configuration.

To enable LDAP authentication for Impala, consult Impala Authentication.

### *LDAP Authorization*

EDH authorization across many of the components relies on group memberships that a user belongs to. These group memberships provide direct authorization, such as the case with HDFS file permissions, or the group memberships can be used for more advanced authorization, such as constructing role-based access controls (RBAC) used with Sentry.

With LDAP authorization, group memberships are ascertained by performing a lookup against the LDAP directory, as opposed to obtaining this information from locally defined groups on the operating system. There is a subtle distinction between the two, however, when considering Hadoop group mapping choices. This is described in the next section.

### *Hadoop Group Mapping Choices*

Hadoop has the ability to discern which groups a user belongs to using several different choices. The main two options to consider are using the local operating system to find the groups for a user, or to directly use LDAP integration to find the groups for a user.

By default, Cloudera Manager configures Hadoop with `ShellBasedUnixGroupsMapping`. This configures Hadoop to make an operating system call, "`id -Gn`", to find the list of groups for a user. The other common choice is to use `LdapGroupsMapping`. This configures Hadoop to make LDAP calls to find group information from the LDAP directory and is often used to work around complex forrest/user naming issues.

Cloudera recommends that `ShellBasedUnixGroupsMapping` be used as the preferred option, even when LDAP integration is desired. A best practice for LDAP integration is to integrate the underlying operating system with LDAP such that all applications, including EDH components, utilize the same identity database. Configuring applications individually can lead to duplicate configurations, or worse, differing configurations that may violate internal security policies. Additionally, operating system integration provides for more flexible and stable caching for LDAP lookups.

Refer to Configuring LDAP Group Mappings for more information.

## Operating System Integration

In order to take advantage of `ShellBasedUnixGroupsMapping` with LDAP, as described in the last section, the Linux operating system itself must be integrated with LDAP. There are many choices available for this type of integration. Cloudera recommends using an integration software package that comes with some kind of support since Linux integration with LDAP is a critical component in the stack. Popular (and recommended) choices include Centrify Server Suite, Dell Authentication Services (formerly Quest QAS/VAS) and Red Hat SSSD.

These integration suites not only provide LDAP integration, but also Kerberos integration.

## Role-Based Access Controls (RBAC)

While the canonical example of authorization is file permissions in HDFS, other frameworks require a more extensive authorization model. Role-based access controls (RBAC) provide a mechanism to control authorization by the type of work a user or application is doing. The type of work being done is organized into a role, and this role is then mapped to one or more groups.

For example, if a set of business analysts across several application teams all access the same types of data, it makes sense to create a business analyst role and map it to the one or more groups that represent the application teams. This not only eases the burden on administration by grouping authorization to job function, but it also more clearly describes the purpose of authorization.

In Cloudera EDH, Sentry provides RBAC functionality. Sentry has three different privilege models for authorization: SQL and Search.

### *SQL*

The Sentry SQL privilege model governs access to Hive and Impala. It provides the ability to define authorization controls at the Server, Database, Table and View level. The complete SQL privilege model for Sentry is described at The Sentry Service.

### *Search*

The Sentry Search privilege model governs access to Solr. It provides the ability to define authorization controls at the Collection or Document level. The complete Search privilege model for Sentry is described at Configuring Sentry Authorization for Cloudera Search.

### *Kafka*

The Sentry Kafka privilege model governs access to Kafka. It provides the ability to define authorization controls at Cluster, Topic, and Consumer Group levels. The complete Kafka privilege model for Sentry is described at Configuring Flume Security with Kafka.

### *HDFS Integration*

When using Sentry for SQL authorization, policies are defined on the assumption that users will access data using SQL via Hive and Impala interfaces. This may not always be the case. If users require access to the same data, but using other access paths such as Spark and MapReduce, it is necessary to also have Sentry policies replicated to the underlying HDFS directories and files.

Cloudera recommends that Sentry HDFS ACL synchronization be turned on for all paths where Hive/Impala data is stored. This ensures that a single set of Sentry policies governs access to the data, regardless of what the processing engine or access path is. For information about Sentry HDFS ACL synchronization, consult Synchronizing HDFS ACLs and Sentry Permissions.

## Wire Encryption

Protecting data as it travels over the network is a critical component to security in Hadoop. There are two primary methods for providing wire encryption in Cloudera EDH: Transport Layer Security (TLS) and SASL Quality of Protection (QoP).

### *Transport Layer Security (TLS)*

TLS, the evolution of SSL, is the canonical choice for wire encryption. EDH uses TLS to protect communications in many of the components in the stack. TLS utilizes certificates to verify the identity of each party in the communication process. A trusted certificate authority issues these certificates such that each party trusts the other, if and only if a trusted certificate authority issued the certificate presented to them.

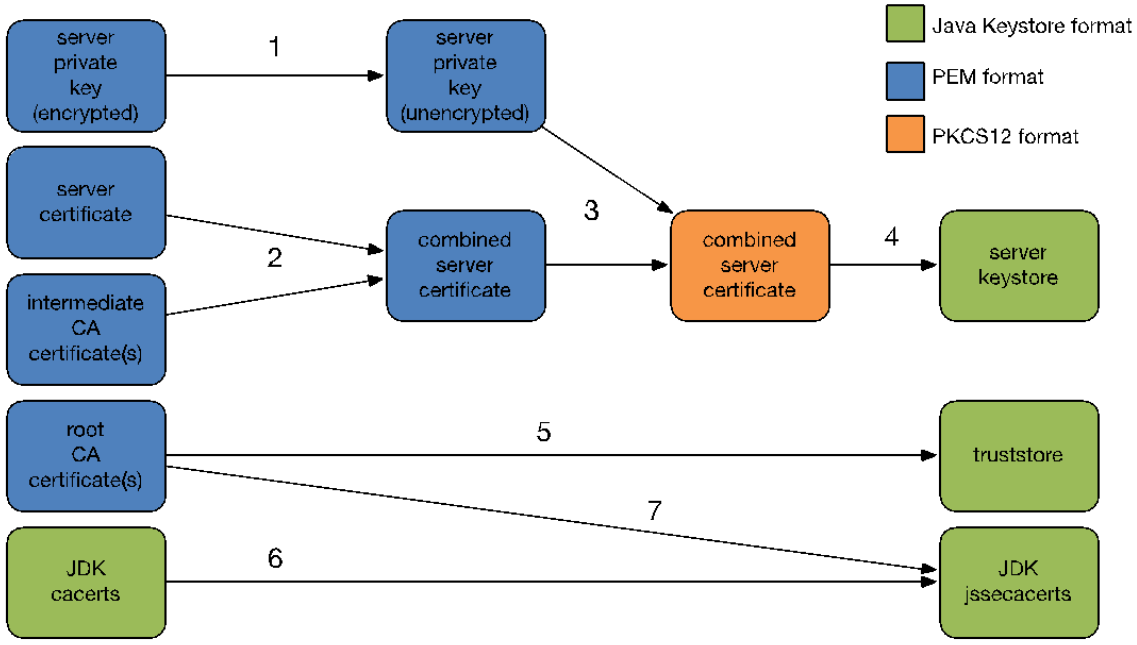
For an EDH deployment utilizing TLS across all components in the stack, it is necessary for every server in the cluster to have a unique certificate. The certificates have the following requirements/recommendations:

- Must be able to be used for both client and server authentication
- Must be able to be used for digital signatures and encryption
- Does not have to be a public certificate issued by a commercial certificate authority - certificates issued by an internal certificate authority is the recommended approach
- Must have the certificate common name (CN) set to the server's fully qualified domain name (FQDN), such as server1.example.com

In addition to the properties for the server certificates themselves, the following are also requirements:

- Server certificate and private key must have a copy in PEM format
- Server certificate and private key must have a copy in a Java KeyStore (JKS) format
- Root certificate authority certificate must have a copy in PEM format
- Root certificate authority certificate must have a copy in a Java KeyStore (JKS) format
- Intermediate certificate authority certificates must be part of the server certificate formats

Preparing certificates in the PEM and JKS formats require Java (`keytool`) and OpenSSL libraries. The following diagram describes this process:




1 - openssl rsa -in `hostname -f`.key.temp -out `hostname -f`.key
2 - cat server.pem int-CA.pem > `hostname -f`.pem
3 - openssl pkcs12 -export -in `hostname -f`.pem -inkey `hostname -f`.key -out `hostname -f`.pfx
4 - keytool -importkeystore -srcstoretype PKCS12 -srckeystore `hostname -f`.pfx -destkeystore `hostname -f`.jks
5 - keytool -importcert -file root-CA.pem -alias root-CA -keystore truststore.jks
6 - cp $JAVA_HOME/jre/lib/security/cacerts $JAVA_HOME/jre/lib/security/jssecacerts
7 - keytool -importcert -file root-CA.pem -alias root-CA -keystore $JAVA_HOME/jre/lib/security/jssecacerts

```
1 - openssl rsa -in $(hostname -f).key.temp -out $(hostname -f).key
2 - cat server.pem int-CA.pem > $(hostname -f).pem
3 - openssl pkcs12 -export -in $(hostname -f).pem -inkey \
      $(hostname -f).key -out $(hostname -f).pfx
4 - keytool -importkeystore -srcstoretype PKCS12 -srckeystore \
      $(hostname -f).pfx -destkeystore $(hostname -f).jks
5 - keytool -importcert -file root-CA.pem -alias root-CA \
      -keystore truststore.jks
6 - cp $JAVA_HOME/jre/lib/security/cacerts \
```

```
        $JAVA_HOME/jre/lib/security/jssecacerts
7 - keytool -importcert -file root-CA.pem -alias root-CA \
        -keystore $JAVA_HOME/jre/lib/security/jssecacerts
```

It is recommended to place all of the resulting certificate files into a common place that exists on every node. For example, `/opt/cloudera/security`.

In order to configure services globally, the server certificate files on each machine need to have a common name. To achieve this, use symbolic links:

```
ln -s /opt/cloudera/security/x509/$(hostname -f).key \
  /opt/cloudera/security/x509/server.key
ln -s /opt/cloudera/security/x509/$(hostname -f).pem \
  /opt/cloudera/security/x509/server.pem
ln -s /opt/cloudera/security/jks/$(hostname -f).jks \
  /opt/cloudera/security/jks/server.jks
```

Additionally, since the certificates will be used by multiple components, appropriate permissions need to be set to both allow service users access, but not allow world access. This can be done using Unix extended ACLs, a feature that is enabled by default by most modern Linux operating systems. An example script that sets permissions for all Cloudera EDH users is as follow:

```
#!/bin/bash

ROOTDIR=/opt/cloudera/security
HOSTNAME=$(hostname -f)
EDH_USERS=(cloudera-scm flume hbase hdfs hive httpfs hue
impala kms llama mapred oozie solr spark sqoop sqoop2 yarn zookeeper)

for EDH_USER in "${EDH_USERS[@]}"
do
    setfacl -m "u:$EDH_USER:r--" $ROOTDIR/jks/$HOSTNAME.jks
    setfacl -m "u:$EDH_USER:r--" $ROOTDIR/x509/$HOSTNAME.key
    setfacl -m "u:$EDH_USER:r--" $ROOTDIR/x509/$HOSTNAME.pem
done
```

For a list of the latest EDH users, consult the Cloudera documentation: Hadoop Users in Cloudera Manager and CDH.

For steps to enable TLS in EDH, consult the documentation: Encryption Mechanisms Overview. For Hive, refer to Configuring Encrypted Communication Between HiveServer2 and Client Drivers.

### *SASL Quality of Protection (QoP)*

Another mechanism for wire encryption is the usage of Quality of Protection, or QoP. This is a module available in the Java SASL suite of security components. It is easy to implement (from a developer point of view) and is an extension to Kerberos authentication.

Some of the components in EDH have the ability to utilize QoP for wire encryption. There are three possible modes for QoP: authentication (`auth`), integrity (`auth-int`), and confidentiality (`auth-conf`). The authentication mode only protects the network channel during authentication. The second mode verifies the integrity of all data sent over the network channel in addition to protecting the authentication channel. The last mode uses encryption in addition to protecting authentication and verifying integrity.

The components that use SASL QoP are HDFS and HBase.

For HDFS, refer to How to Configure Encrypted Transport for HDFS Data for more information.

Use the hbase.rpc.protection property to set the HBase RPC security mechanism . It should be inserted as both a Service-Wide and Gateway advanced configuration snippet for HBase. Valid values, in order of increasing protection are: `authentication`, `integrity`, and `privacy`. For more information see the documentation on configuring HBase transport encryption.

Example, ensure transport encryption:

```
<property>
      <name>hbase.rpc.protection</name>
      <value>privacy</value>
</property>
```

## Encryption-at-Rest and Key Management

Protecting data while it is stored using encryption is fundamental in protecting against certain types of attacks. EDH has the ability to provide for protection on both data persisted in HDFS, as well as protection that is persisted outside of HDFS in more general areas such as staging directories, database storage, and log files.

This section describes the at-rest encryption options available, and how to securely store and manage the keys used to protect data at rest.

### *HDFS Encryption*

HDFS encryption provides encryption at rest at the application layer. This allows for the encryption of logical paths in HDFS in what is defined as encryption zones. The encryption zones specify a directory path in HDFS, and a name of an encryption key to use. This 1:1 mapping between encryption zone and encryption key allows for a separation of security zones, which is important for multi-tenant environments.

For example, a sensitive source A may have a security requirement for encryption at rest using a non-shared encryption key. A sensitive source B may also have the same security requirement. In this

example, ingesting source A and source B into different encryption zones allows them to have separate and distinct encryption keys.

HDFS encryption requires two separate components: a key management service (KMS), and a key store. The key management service serves as a proxy between clients, HDFS, and the key store. This creates a security boundary between HDFS and the unencrypted keys used to encrypt/decrypt data.

CDH bundles a default KMS that is backed by a file-based keystore, which is effectively a Java KeyStore object. This KMS implementation combines the KMS and key store functions into a single service. This implementation should not be used in a production environment with sensitive data. It is useful to deploy in development environments to understand architecture and cluster administration tasks. A separate KMS implementation that uses Navigator Key Trustee as the key store is Cloudera's recommended implementation. This separates the KMS and key store roles and allows them to be separated on different servers, which in turn provides better key protection. Navigator Key Trustee is discussed later.

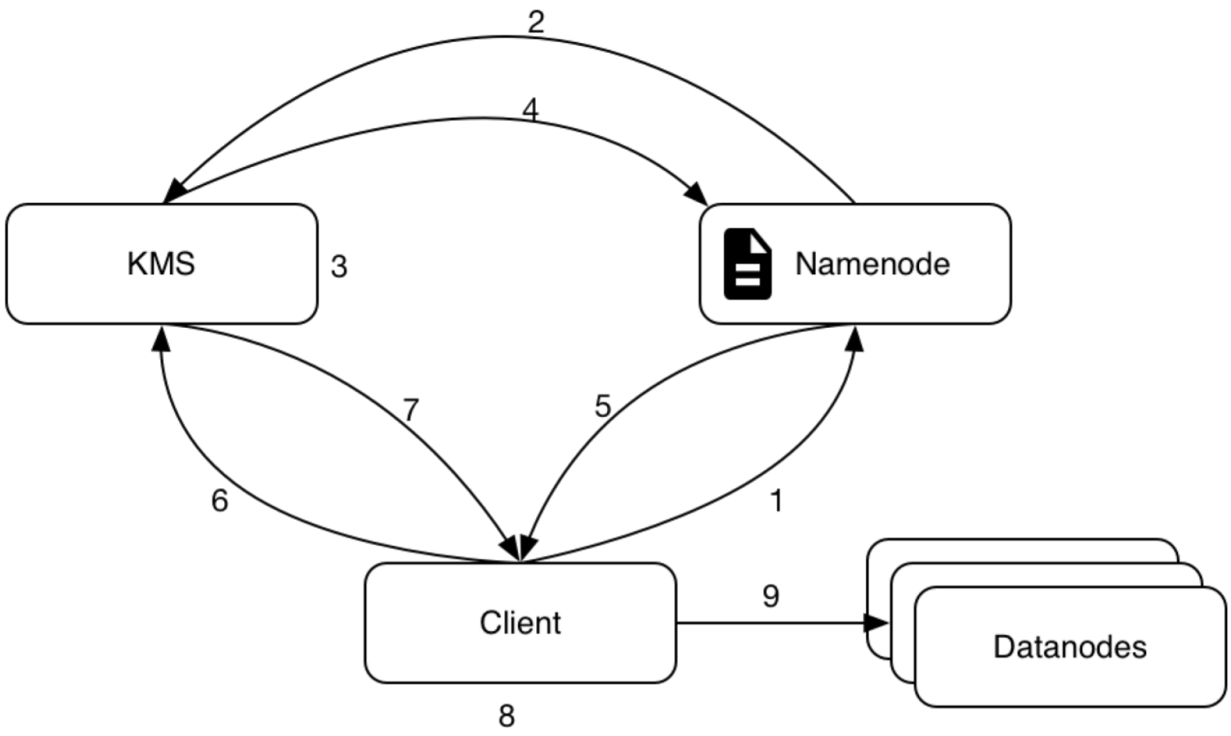
HDFS encryption involves two different keys. The first key, as described early, is directly related to the encryption zone. It is often called the encryption zone key, or EZK for short. The second key is for the encryption and decryption of an actual file in HDFS. Every file in an encryption zone has a unique encryption key. This key is called a data encryption key, or DEK for short. Finally, another object is the encrypted DEK, or EDEK. This object results from encrypting the DEK using the EZK. The EDEK is persisted as part of the NameNode metadata. EZKs are persisted in the key provider.

Because EDEKs are part of the NameNode metadata, it is important to understand the memory footprint of this extra metadata. The additional attribute that is added for an encrypted file in HDFS is about 50-200 bytes on average. So if an encryption zone in HDFS contains 1 million files, this increases the memory footprint of the NameNode metadata by about 50-200 MB.

To summarize:

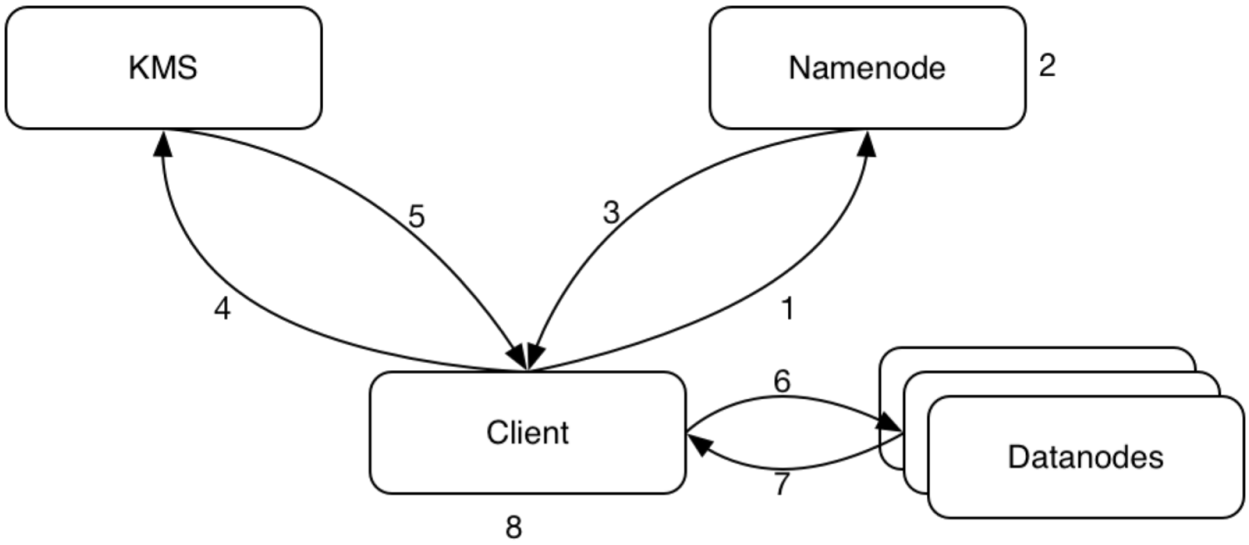
- All files use a unique encryption key, called a DEK
- All DEKs in the same encryption zone are encrypted using the encryption zone key, or EZK
- The resulting encrypted DEK is called a EDEK
- EDEKs are persisted as part of the NameNode metadata at about 50-200 bytes each
- EZKs are persisted in the key provider
- DEKs are not persisted anywhere

The workflow for HDFS encryption on writing looks like the following (After the encryption key has been setup and an encryption zone is created):




1. Start to write a file
2. Get EDEK from KMS
3. Generate a new EDEK using EZK
4. Persist EDEK in NN with file metadata
5. EDEK sent to Client
6. Client asks KMS to decrypt EDEK
7. KMS sends DEK to Client
8. Client encrypts the file contents using DEK
9. Encrypted file saved on Data nodes

The workflow for HDFS encryption on reading looks like the following:




1. Client asks Namenode to read a file
2. Namenode checks if client has access to the requested file
3. Namenode sends the EDEK to the client along with block info
4. Client asks KMS to decrypt EDEK using EZK
5. KMS sends DEK to the Client
6. Client asks Datanodes to send data
7. Datanodes sends encrypted data to the Client
8. Client decrypts the data using DEK

Based on the workflows, several important security facts are noteworthy:

- HDFS (NameNodes, DataNodes, etc.) only have access to the EDEKs, which cannot be used to decrypt data
- HDFS clients do not have access to EZKs, but do have access to DEKs
- The KMS has access to both EZKs and DEKs
- Neither HDFS nor clients have access to the key store

HDFS encryption administration is exposed through several command line functions. These functions and additional information are described in HDFS Transparent Encryption.

### *Navigator Encrypt*

Navigator Encrypt provides at-rest encryption for data that lives outside of HDFS. This includes staging areas before data is ingested into HDFS, log directories, temporary file storage, database storage directories, and component storage areas such as Flume and Kafka.

Navigator Encrypt is a complementary at-rest encryption solution to HDFS encryption. Cloudera does not support using Navigator Encrypt to encrypt HDFS data directories.

### *Navigator Key Trustee*

Navigator Key Trustee is the most important part of a Cloudera encryption implementation. Encrypting data, regardless of the mechanism, is simply applying well-known mathematical algorithms to transform

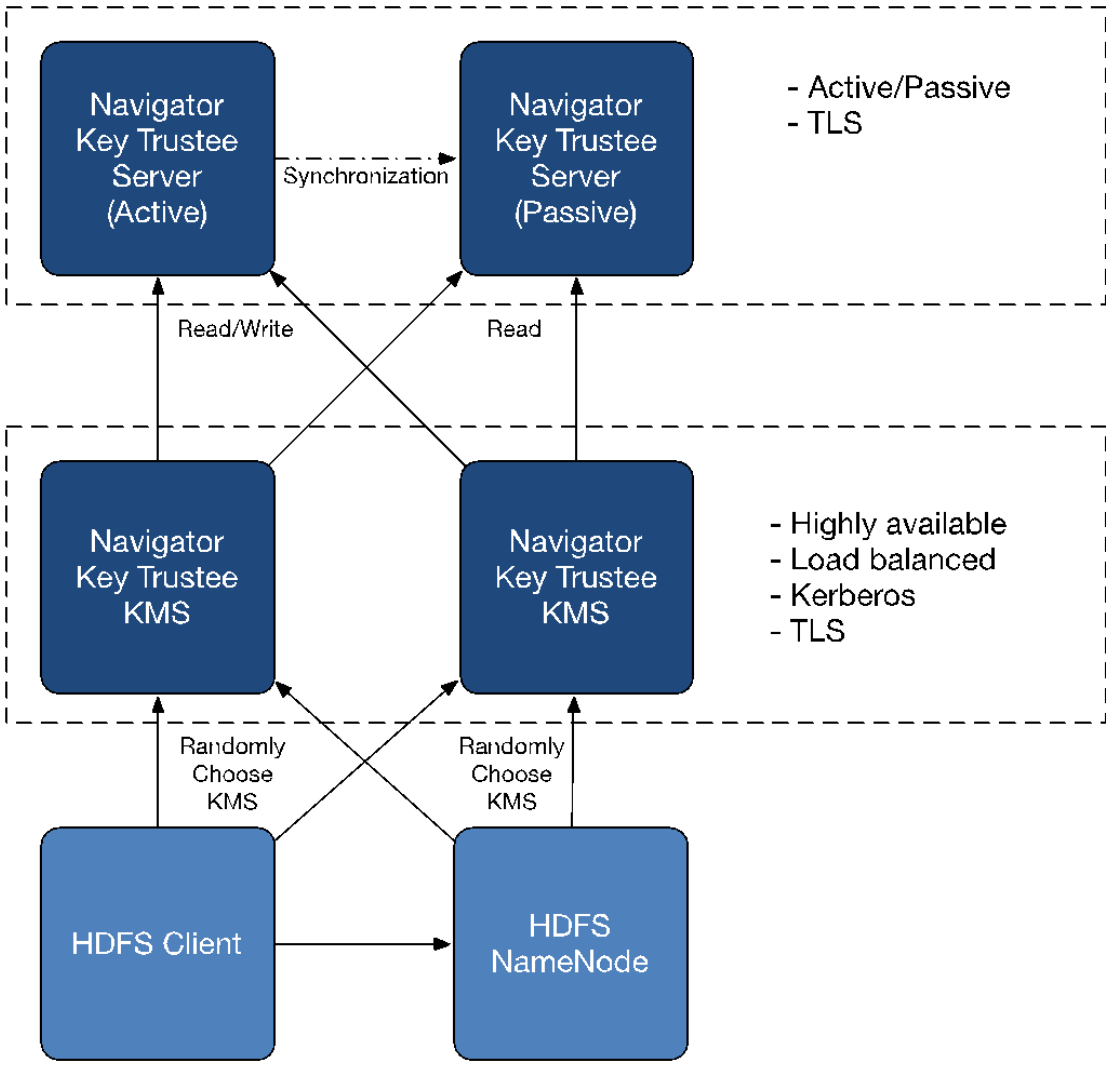
data. The difficulty is ensuring that the encryption keys used must be kept in a secure place to ensure that unauthorized users do not decrypt encrypted data. Navigator Key Trustee provides secure storage for sensitive key material, as well as other objects such as SSL certificates and passwords.

Navigator Key Trustee's architecture includes a Postgres database that stores all key material. Also, as of version 3.7, Navigator Key Trustee supports a master/slave architecture with synchronous database replication. This ensures that Navigator Key Trustee is not a single point of failure.

Integration with hardware security modules (HSMs) is supported. Currently Navigator Key Trustee supports integration with HSMs from Thales and SafeNet.

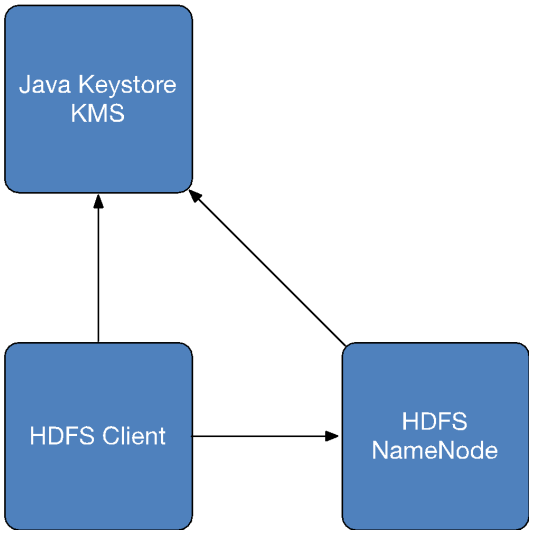
***System Architecture***

Based on the components described in this section, the following system architecture represents Cloudera's recommendation for a production deployment:




With this architecture, Cloudera recommends that each of the Key Trustee Servers and Key Trustee KMS be located on their own machine, separate from any other EDH components. This provides some additional security guarantees and opportunities to further separate administration (separation of duties).

For a development environment, the architecture is much simpler:




With this implementation, the KMS is using a file-based keystore, thus HA is not possible. Also, with this architecture it is assumed that since it is a development environment, it is feasible to collocate the KMS with other cluster services, such as on a utility node that contains Cloudera Manager. Again, this configuration should not be used in production, as it does not provide the same security guarantees as the architecture with Navigator Key Trustee.

***Intermediate/Spill File Encryption***

### MapReduce (MR2)

When MapReduce jobs execute, temporary files are created and written to local disk outside of HDFS. If the original source data in HDFS is sensitive and encrypted, the temporary files that are derived from the sensitive data must also be protected.

In MapReduce version 2 (on YARN), this is possible to configure intermediate file encryption. This is not available on MapReduce version 1. The main caveat with intermediate file encryption is that it is job configuration property, thus it is set in a client configuration. Clients can choose to toggle intermediate file encryption on or off. It is recommended to set intermediate file encryption to on by default, when HDFS encryption is used. This ensures that jobs will use encryption by default, unless explicitly overridden by a client configuration. This reduces the chance that sensitive data could be written to disk unencrypted.

To enable MapReduce intermediate file encryption, the following advanced configuration snippet is necessary in mapred-site.xml for the Gateway role:

```
<property>
      <name>mapreduce.job.encrypted-intermediate-data</name>
      <value>true</value>
</property>
```

By default, intermediate file encryption uses 128-bit keys. To change this (e.g. to 256), also add the following advanced configuration snippet:

```
<property>
     <name>mapreduce.job.encrypted-intermediate-data-key-size-bits</name>
     <value>256</value>
</property>
```

### Impala

Like MapReduce, Impala also writes temporary files to disk when the data does not fit wholly into memory. Starting with Impala 2.0, Impala daemons can encrypt spill files before writing to disk. To enable this feature, check the box for **Disk Spill Encryption** under the Impala Daemon Security configuration in Cloudera Manager.

## Cloudera Navigator

Cloudera Navigator sits at the heart of data governance capabilities in EDH. It provides fundamental features including auditing, data lineage, and metadata tagging.

Cloudera recommends that an EDH be setup with Navigator as soon as the cluster is installed. This ensures that all auditing and data lineage is captured at the earliest point, prior to users and data being onboarded to the cluster.

### *External Integration*

Cloudera Navigator provides a wealth of information and capabilities that external systems can leverage. The first is audit data. Navigator includes a mechanism to send all audit records to external systems by using a separate log4j appender. This log4j appender can be attached to using any of the default appenders, such as a SysLogAppender or a RollingFileAppender, or a custom appender that is developed to integrate with an external system in a specialized way.

Information about using log4j to capture Navigator audit logs, consult the documentation: Configuring Audit and Log Properties.

Another way to integrate external systems with Navigator is by way of the REST APIs. The REST APIs are useful for both reading data collected and stored in Navigator, to include metadata tags and lineage, but also for writing custom metadata back into Navigator.

Full documentation about the Navigator REST API can be found in the Help menu of Cloudera Navigator.

# Common Questions

## Multiple Data Centers

Cloudera EDH deployments are restricted to single data centers. Single clusters spanning data centers are not supported.

## Operating Systems

A list of supported operating systems for CDH and Cloudera Manager can be found here.

## Storage Options and Configuration

The HDFS data directories should use local storage, which provides all the benefits of keeping compute resources close to the storage and not reading remotely over the network.

The root device size for Cloudera Enterprise clusters should be at least 500 GB to allow parcels and logs to be stored.

To guard against data center failures, you can set up a scheduled distcp operation to persist data to AWS S3 (see the examples in the distcp documentation) or leverage Cloudera Manager's Backup and Data Recovery (BDR) features to backup data to another running cluster.

## Relational Databases

Cloudera Enterprise deployments require relational databases for the following components: Cloudera Manager, Cloudera Navigator, Hive metastore, Hue, Sentry, Oozie, and others. The database credentials are required during Cloudera Enterprise installation.

Refer to Cloudera Manager and Managed Service Datastores for more information.

**Important:**

The embedded PostgreSQL database is not supported for use in production systems.

# References

## Cloudera Enterprise

Product Documentation
- Hardware Requirements Guide
- Requirements and Supported Versions

Cloudera Services & Support

# Acknowledgements

Many individuals from Cloudera contributed to this document, spread across engineering, marketing, professional services, support, etc.

Alan Jackoway, Alex Breshears, Alex Moundalexis, Ben Spivey, David Beech, David Powell, Dwai Lahiri, Hung Pham, Jake Miller, Jeongho Park, Jeremy Beard, Jim Heyssel, Kaufman Ng, Michael Ridley, Morris Hicks, Mubashir Kazia, Rachel Asher, Rick Halihan, Rob Siwicki, Roger Ding, Ronan Stokes, Ryan Fishel, Scott Pace, Sean Busbey, Sean Kane, Todd Grayson, Travis Campbell, Tristan Stevens, Vartika Singh, and Zoltan Kiss.