https://blog.cloudera.com/blog/2017/12/hadoop-delegation-tokens-explained/

Delegation Tokens in Hadoop are generated and verified following the HMAC mechanism. There are two parts of information in a Delegation Token: a public part and a private part. Delegation Tokens are stored at the server side as a hashmap with the public information as the key and the private information as the value.

The public information is used for token identification in the form of an identifier object. It consists of:

Kind	The kind of token (HDFS_DELEGATION_TOKEN, or kms-dt). The token also contains the kind, matching the identifier's kind.
Owner	The user who owns the token.
Renewer	The user who can renew the token.
Real User	Only relevant if the owner is impersonated. If the token is created by an impersonating user, this will identify the impersonating user. For example, when oozie impersonates user joe, Owner will be joe and Real User will be oozie.
Issue Date	Epoch time when the token was issued.

Max Date	Epoch time when the token can be renewed until.
Sequence Number	UUID to identify the token.
Master Key ID	ID of the master key used to create and verify the token.

The private information is represented by class DelegationTokenInformation in AbstractDelegationTokenSecretManager, it is critical for security and contains the following fields:

renewDate	Epoch time when the token is expected to be renewed. If it's smaller than the current time, it means the token has expired.
password	The password calculated as HMAC of Token Identifier using master key as the HMAC key. It's used to validate the Delegation Token provided by the client to the server.

trackingId	A tracking identifier that can be used to associate usages of a token across multiple client sessions. It is computed as the MD5 of the token identifier.

Table 2: Delegation Token Information (private part of a Delegation Token)

Example: Delegation Tokens' Lifecycle

Now that we understand what a Delegation Token is, let's take a look at how it's used in practice. The graph below shows an example flow of authentications for running a typical application, where the job is submitted through YARN, then is distributed to multiple worker nodes in the cluster to execute.

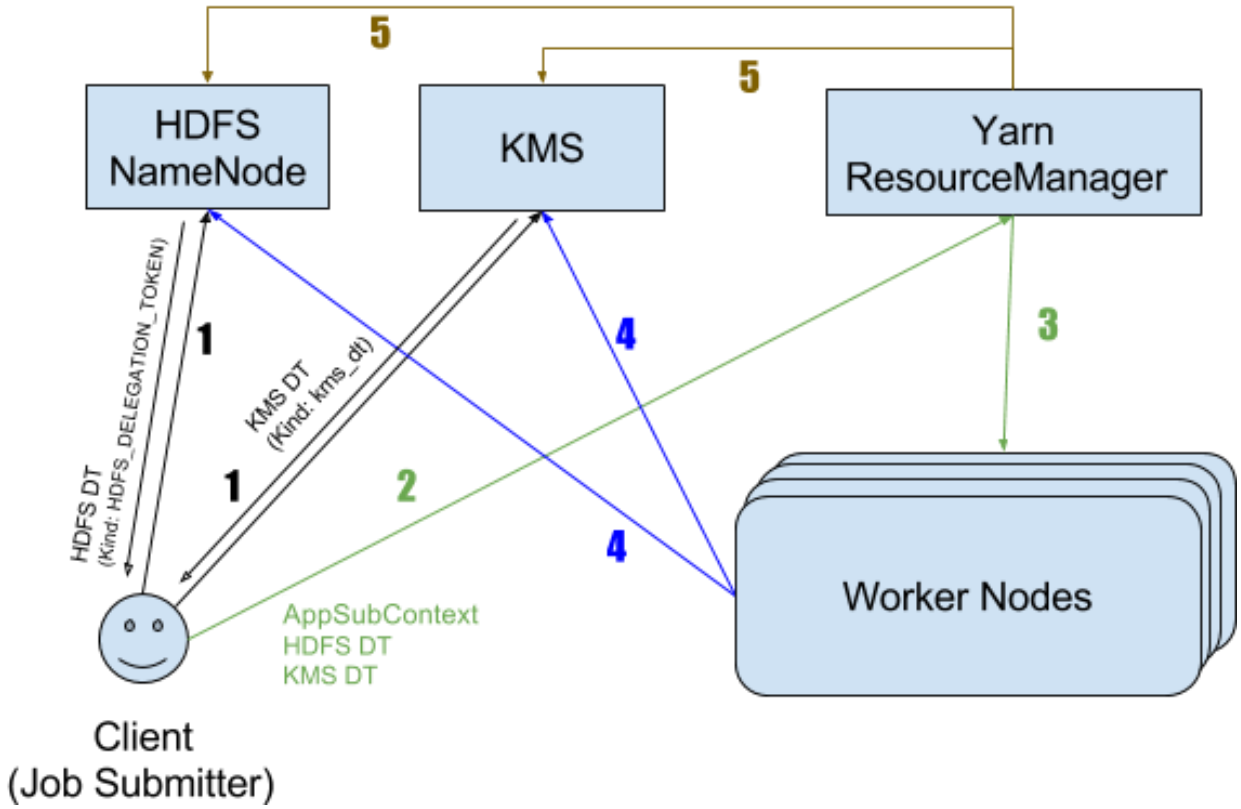


Figure 3: How Delegation Token is Used for Authentication In A Typical Job

For brevity, the steps of Kerberos authentication and the details about the task distribution are omitted. There are 5 steps in general in the graph:

1. The client wants to run a job in the cluster. It gets an HDFS Delegation Token from HDFS NameNode, and a KMS Delegation Token from the KMS.
2. The client submits the job to the YARN Resource Manager (RM), passing the Delegation Tokens it just acquired, as well as the ApplicationSubmissionContext.
3. YARN RM verifies the Delegation Tokens are valid by immediately renewing them. Then it launches the job, which is distributed (together with the Delegation Tokens) to the worker nodes in the cluster.
4. Each worker node authenticates with HDFS using the HDFS Delegation Token as they access HDFS data, and authenticates with KMS using the KMS Delegation Token as they decrypt the HDFS files in encryption zones.
5. After the job is finished, the RM cancels the Delegation Tokens for the job.

Note: A step not drawn in the above diagram is, RM also tracks each Delegation Token's expiration time, and renews the Delegation Token

when it's at 90% of the expiration time. Note that Delegation Tokens are tracked in the RM on an individual basis, not Token-Kind basis. This way, tokens with different renewal intervals can all be renewed correctly. The token renewer classes are implemented using Java ServiceLoader, so RM doesn't have to be aware of the token kinds. For extremely interested readers, the relevant code is in RM's DelegationTokenRenewer class.

What is this InvalidToken error?

Now we know what Delegation Tokens are and how they are used when running a typical job. But don't stop here! Let's look at some typical Delegation Token related error messages from application logs, and figure out what they mean.

Token is expired

Sometimes, the application fails with AuthenticationException, with an InvalidToken exception wrapped inside. The exception message indicates that "token is expired". Guess why this could happen?

https://cwiki.apache.org/confluence/display/Hive/Design

- User Interface (UI) calls the execute interface to the Driver.
- The driver creates a session handle for the query. Then it sends the query to the compiler to generate an execution plan.

Table 9.3 Linux Process States Overview

Key Topic

State	Meaning
Running (R)	The process is currently active and using CPU time, or in the queue of runnable processes waiting to get services.
Sleeping (S)	The process is waiting for an event to complete.
Uninterruptable sleep (D)	The process is in a sleep state that cannot be stopped. This usually happens while a process is waiting for I/O.
Stopped (S)	The process has been stopped, which typically has happened to an interactive shell process, using the **Ctrl+Z** key sequence.
Zombie (Z)	The process has been stopped but could not be removed by its parent, which has put it in an unmanageable state.

When requesting the current status of a systemd unit as in Listing 18.6, you can see different kinds of information about it. Table 18.2 shows the different kinds of information that you can get about unit files when using the **systemctl status** command:

Table 18.2 Systemd Status Overview

Status	Description
Loaded	The unit file has been processed and the unit is active.
Active(running)	Running with one or more active processes.
Active(exited)	Successfully completed a one-time configuration.
Active(waiting)	Running and waiting for an event.
Inactive	Not running.
Enabled	Will be started at boot time.
Disabled	Will not be started at boot time.
Static	This unit can not be enabled but may be started by another unit automatically.

When requesting the current status of a systemd unit as in Listing 18.6, you can see different kinds of information about it. Table 18.2 shows the different kinds of information that you can get about unit files when using the **systemctl status** command:

Table 18.2 Systemd Status Overview

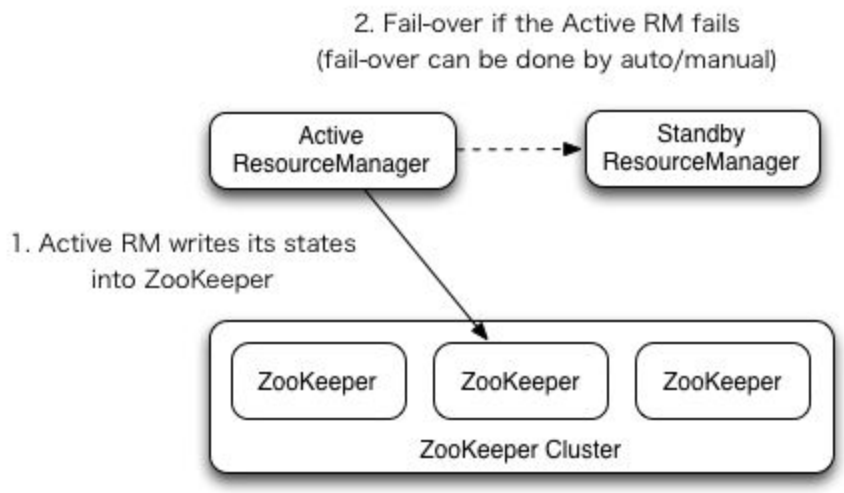
Status	Description
Loaded	The unit file has been processed and the unit is active.
Active(running)	Running with one or more active processes.
Active(exited)	Successfully completed a one-time configuration.
Active(waiting)	Running and waiting for an event.
Inactive	Not running.
Enabled	Will be started at boot time.
Disabled	Will not be started at boot time.
Static	This unit can not be enabled but may be started by another unit automatically.

https://hadoop.apache.org/docs/stable/hadoop-yarn/hadoop-yarn-site/ResourceManagerHA.html

Introduction

This guide provides an overview of High Availability of YARN's ResourceManager, and details how to configure and use this feature. The ResourceManager (RM) is responsible for tracking the resources in a cluster, and scheduling applications (e.g., MapReduce jobs). Prior to Hadoop 2.4, the ResourceManager is the single point of failure in a YARN cluster. The High Availability feature adds redundancy in the form of an Active/Standby ResourceManager pair to remove this otherwise single point of failure.

Architecture



RM Failover

ResourceManager HA is realized through an Active/Standby architecture - at any point of time, one of the RMs is Active, and one or more RMs are in Standby mode waiting to take over should anything happen to the Active. The trigger to transition-to-active comes from either the admin (through CLI) or through the integrated failover-controller when automatic-failover is enabled.

Manual transitions and failover

When automatic failover is not enabled, admins have to manually transition one of the RMs to Active. To failover from one RM to the other, they are expected to first transition the Active-RM to Standby and transition a Standby-RM to Active. All this can be done using the “yarn rmadmin” CLI.

Automatic failover

The RMs have an option to embed the Zookeeper-based ActiveStandbyElector to decide which RM should be the Active. When the Active goes down or becomes unresponsive, another RM is automatically elected to be the Active which then takes over. Note that, there is no need to run a separate ZKFC daemon as is the case for HDFS because ActiveStandbyElector embedded in RMs acts as a failure detector and a leader elector instead of a separate ZKFC deamon.

Client, ApplicationMaster and NodeManager on RM failover

When there are multiple RMs, the configuration (yarn-site.xml) used by clients and nodes is expected to list all the RMs. Clients, ApplicationMasters (AMs) and NodeManagers (NMs) try connecting to the RMs in a round-robin fashion until they hit the Active RM. If the Active goes down, they resume the round-robin polling until they hit the “new” Active. This default retry logic is implemented as org.apache.hadoop.yarn.client.ConfiguredRMFailoverProxyProvider. You can override the logic by implementing org.apache.hadoop.yarn.client.RMFailoverProxyProvider and setting the value of yarn.client.failover-proxy-provider to the class name.

Recovering previous active-RM’s state

With the ResourceManger Restart enabled, the RM being promoted to an active state loads the RM internal state and continues to operate from where the previous active left off as much as possible depending on the RM restart feature. A new attempt is spawned for each managed application previously submitted to the RM. Applications can checkpoint periodically to avoid losing any work. The state-store must be visible from the both of Active/Standby RMs. Currently, there are two RMStateStore implementations for persistence - FileSystemRMStateStore and ZKRMStateStore. The ZKRMStateStore implicitly allows write access to a single RM at any point in time, and hence is the recommended store to use in an HA cluster. When using the ZKRMStateStore, there is no need for a separate fencing mechanism to address a potential split-brain situation where multiple RMs can potentially assume the Active role. When using the ZKRMStateStore, it is advisable to NOT set the "zookeeper.DigestAuthenticationProvider.superDigest" property on the Zookeeper cluster to ensure that the zookeeper admin does not have access to YARN application/user credential information.

Deployment

Configurations

Most of the failover functionality is tunable using various configuration properties. Following is a list of required/important ones. yarn-default.xml carries a full-list of knobs. See yarn-default.xml for more information including default values. See the document for ResourceManger Restart also for instructions on setting up the state-store.

yarn.resourcemanager.zk-address	Address of the ZK-quorum. Used both for the state-store and embedded leader-election.
yarn.resourcemanager.ha.enabled	Enable RM HA.
yarn.resourcemanager.ha.rm-ids	List of logical IDs for the RMs. e.g., "rm1,rm2".
yarn.resourcemanager.hostname.*rm-id*	For each *rm-id*, specify the hostname the RM corresponds to. Alternately, one could set each of the RM's service addresses.
yarn.resourcemanager.address.*rm-id*	For each *rm-id*, specify host:port for clients to submit jobs. If set, overrides the hostname set in yarn.resourcemanager.hostname.*rm-id*.
yarn.resourcemanager.scheduler.address.*rm-id*	For each *rm-id*, specify scheduler host:port for ApplicationMasters to obtain resources. If set, overrides the hostname set in yarn.resourcemanager.hostname.*rm-id*.
yarn.resourcemanager.resource-tracker.address.*rm-id*	For each *rm-id*, specify host:port for NodeManagers to connect. If set, overrides the hostname set in yarn.resourcemanager.hostname.*rm-id*.
yarn.resourcemanager.admin.address.*rm-id*	For each *rm-id*, specify host:port for administrative commands. If set, overrides the hostname set in yarn.resourcemanager.hostname.*rm-id*.
yarn.resourcemanager.webapp.address.*rm-id*	For each *rm-id*, specify host:port of the RM web application corresponds to. You do not need this if you set yarn.http.policy to HTTPS_ONLY. If set, overrides the hostname set in yarn.resourcemanager.hostname.*rm-id*.
yarn.resourcemanager.webapp.https.address.*rm-id*	For each *rm-id*, specify host:port of the RM https web application corresponds to. You do not need this if you set yarn.http.policy to HTTP_ONLY. If set, overrides the hostname set in yarn.resourcemanager.hostname.*rm-id*.

yarn.resourcemanager.ha.id	Identifies the RM in the ensemble. This is optional; however, if set, admins have to ensure that all the RMs have their own IDs in the config.
yarn.resourcemanager.ha.automatic-failover.enabled	Enable automatic failover; By default, it is enabled only when HA is enabled.
yarn.resourcemanager.ha.automatic-failover.embedded	Use embedded leader-elector to pick the Active RM, when automatic failover is enabled. By default, it is enabled only when HA is enabled.
yarn.resourcemanager.cluster-id	Identifies the cluster. Used by the elector to ensure an RM doesn't take over as Active for another cluster.
yarn.client.failover-proxy-provider	The class to be used by Clients, AMs and NMs to failover to the Active RM.
yarn.client.failover-max-attempts	The max number of times FailoverProxyProvider should attempt failover.
yarn.client.failover-sleep-basech-ms	The sleep base (in milliseconds) to be used for calculating the exponential delay between failovers.
yarn.client.failover-sleep-max-ms	The maximum sleep time (in milliseconds) between failovers.
yarn.client.failover-retries	The number of retries per attempt to connect to a ResourceManager.
yarn.client.failover-retries-on-socket-timeouts	The number of retries per attempt to connect to a ResourceManager on socket timeouts.

https://hadoop.apache.org/docs/current/hadoop-yarn/hadoop-yarn-site/YARN.html

→ Difference between buffer and cache
https://stackoverflow.com/questions/6345020/what-is-the-difference-between-buffer-vs-cache-memory-in-linux

A buffer is something that has yet to be "written" to disk.

A cache is something that has been "read" from the disk and stored for later use.

Cache Pages:

A cache is the part of the memory which transparently stores data so that future requests for that data can be served faster. This memory is utilized by the kernel to cache disk data and improve i/o performance.

The Linux kernel is built in such a way that it will use as much RAM as it can to cache information from your local and remote filesystems and disks. As the time passes over various reads and writes are performed on the system, kernel tries to keep data stored in the memory for the various processes which are running on the system or the data that of relevant processes which would be used in the near future. The cache is not reclaimed at the time when process get stop/exit, however when the other processes requires more

memory then the free available memory, kernel will run heuristics to reclaim the memory by storing the cache data and allocating that memory to new process.

When any kind of file/data is requested then the kernel will look for a copy of the part of the file the user is acting on, and, if no such copy exists, it will allocate one new page of cache memory and fill it with the appropriate contents read out from the disk.

The data that is stored within a cache might be values that have been computed earlier or duplicates of original values that are stored elsewhere in the disk. When some data is requested, the cache is first checked to see whether it contains that data. The data can be retrieved more quickly from the cache than from its source origin.

SysV shared memory segments are also accounted as a cache, though they do not represent any data on the disks. One can check the size of the shared memory segments using ipcs -m command and checking the bytes column.

Buffers :

Buffers are the disk block representation of the data that is stored under the page caches. Buffers contains the metadata of the files/data which resides under the page cache. Example: When there is a request of any data which is present in the page cache, first the kernel checks the data in the buffers which contain the metadata which points to the actual files/data contained in the page caches. Once from the metadata the actual block address of the file is known, it is picked up by the kernel for processing.

→ Page Cache:
https://www.thomas-krenn.com/en/wiki/Linux_Page_Cache_Basics

https://www.usna.edu/Users/cs/aviv/classes/ic221/s16/lec/21/lec.html

vm swapinness:
Redhat page :635
Table 28.5 Most Useful sysctl Tunables
Tunable Use
net.ipv4.ip_forward Set to 1 to enable packet forwarding between
network interfaces.
net.ipv4.icmp_echo_ignore_all Set to 1 to disable all ping.
net.ipv4.icmp_echo_ignore_broadcasts Set to 1 to disable broadcast ping.
vm.swappiness Use a value between 0 and 100 to increase the
willingness of your server to swap data.
kernel.hostname Set the hostname of this syste

Swappiness The willingness of the Linux kernel to move memory pages from physical RAM to swap. Swappiness is set as a parameter in the /proc/sys/vm directory, which can be modified to tune the swap behavior.

Swappiness is a property for the Linux kernel that changes the balance between swapping out runtime memory, as opposed to dropping pages from the system page cache. Swappiness can be set to values between 0 and 100, inclusive. A low value means the kernel will try to avoid swapping as much as possible where a higher value instead will make the kernel aggressively try to use swap space

Hive Execution Flow
https://cwiki.apache.org/confluence/display/Hive/Design

- User Interface (UI) calls the execute interface to the Driver.
- The driver creates a session handle for the query. Then it sends the query to the compiler to generate an execution plan.

- UI – The user interface for users to submit queries and other operations to the system. As of 2011 the system had a command line interface and a web based GUI was being developed.
- Driver – The component which receives the queries. This component implements the notion of session handles and provides execute and fetch APIs modeled on JDBC/ODBC interfaces.

The compiler needs the metadata. So it sends a request for *getMetaData*. Thus receives the *sendMetaData* request from Metastore.

- Compiler – The component that parses the query, does semantic analysis on the different query blocks and query expressions and eventually generates an execution plan with the help of the table and partition metadata looked up from the metastore.

https://hadoop.apache.org/docs/stable/hadoop-yarn/hadoop-yarn-site/ResourceManagerHA.html

http://blog.cloudera.com/blog/2014/05/how-apache-hadoop-yarn-ha-works/

Just like MapReduce tasks are given several attempts to succeed (in the face

of hardware

or network failures), applications in YARN are retried in the event of failure. The max-

imum number of attempts to run a MapReduce application master is controlled by the

mapreduce.am.max-attempts property. The default value is 2, so if a MapReduce ap-

plication master fails twice it will not be tried again and the job will fail.

YARN imposes a limit for the maximum number of attempts for any YARN application

master running on the cluster, and individual applications may not exceed this limit.

The limit is set by yarn.resourcemanager.am.max-attempts and defaults to 2, so if

you want to increase the number of MapReduce application master attempts, you will

have to increase the YARN setting on the cluster, too.

The way recovery works is as follows. An application master sends periodic heartbeats

to the resource manager, and in the event of application master failure, the resource

manager will detect the failure and start a new instance of the master running in a new

container (managed by a node manager). In the case of the MapReduce application

master, it will use the job history to recover the state of the tasks that were already run

by the (failed) application so they don’t have to be rerun. Recovery is enabled by default,

but can be disabled by setting yarn.app.mapreduce.am.job.recovery.enable to

false .

When the new resource manager starts, it reads the application information from the

state store, then restarts the application masters for all the applications running on the

cluster. This does not count as a failed application attempt (so it does not count against

yarn.resourcemanager.am.max-attempts), since the application did not fail due to an

error in the application code, but was forcibly killed by the system. In practice, the

application master restart is not an issue for MapReduce applications since they recover

the work done by completed tasks (as we saw in “Application Master Failure” on page

194).

The transition of a resource manager from standby to active is handled by a failover

controller. The default failover controller is an automatic one, which uses ZooKeeper

leader election to ensure that there is only a single active resource manager at one time.

Unlike in HDFS HA (see “HDFS High Availability” on page 48), the failover controller

does not have to be a standalone process, and is embedded in the resource manager by

default for ease of configuration. It is also possible to configure manual failover, but this

is not recommended.

https://stackoverflow.com/questions/33452579/what-happens-when-the-resource-manager-rm-goes-down-in-yarn

When the new resource manager starts, it reads the application information from the

state store, then restarts the application masters for all the applications running on the

cluster. This does not count as a failed application attempt (so it does not count against

yarn.resourcemanager.am.max-attempts), since the application did not fail due to an

error in the application code, but was forcibly killed by the system. In practice, the

application master restart is not an issue for MapReduce applications since they recover

the work done by completed tasks (as we saw in “Application Master Failure” on page

194)

Resource allocation scenarios:

http://blog.cloudera.com/blog/2014/04/apache-hadoop-yarn-avoiding-6-time-consuming-gotchas/

https://community.hortonworks.com/questions/96750/yarn-application-stuck-in-accepted-state.html

https://community.cloudera.com/t5/Batch-Processing-and-Workflow/YARN-apps-stuck-won-t-allocate-resources/td-p/23224

https://www.cloudera.com/documentation/enterprise/release-notes/topics/hardware_requirements_guide.html

https://community.hortonworks.com/questions/15449/which-oozie-metrics-should-be-monitored.html

https://www.cloudera.com/documentation/enterprise/5-11-x/topics/admin_hive_tuning.html

Additional tuning:

For workloads with many coordinators that run with complex workflows (a **max concurrency reached!**warning appears in the log and the Oozie `admin -queuedump` command shows a large queue):

- Increase the value of the `oozie.service.CallableQueueService.callable.concurrency` property to 50.
- Increase the value of the `oozie.service.CallableQueueService.threads` property to 200.

Versions of each service on CM 5.13.1:

https://www.cloudera.com/documentation/enterprise/release-notes/topics/cdh_vd_cdh_package_tarball_513.html

Configuration

Each server in the ensemble of ZooKeeper servers has a numeric identifier that is unique

within the ensemble and must fall between 1 and 255. The server number is specified

in plain text in a file named myid in the directory specified by the dataDir property.

Setting each server number is only half of the job. We also need to give every server all

the identities and network locations of the others in the ensemble. The ZooKeeper

configuration file must include a line for each server, of the form:

http://archive.cloudera.com/cdh5/cdh/5/zookeeper/zookeeperAdmin.html#sc_zkCommands

https://stackoverflow.com/questions/41874718/solr-cloud-what-is-the-recommended-number-of-zookeepers-per-solr-instances

How Many ZooKeepers?

> *"For a ZooKeeper service to be active, there must be a majority of non-failing machines that can communicate with each other.* ***To create a deployment that can tolerate the failure of F machines, you should count on deploying 2xF+1***

> ***machines**. Thus, a deployment that consists of three machines can handle one failure, and a deployment of five machines can handle two failures. Note that a deployment of six machines can only handle two failures since three machines is not a majority.*
>
> *For this reason, ZooKeeper deployments are usually made up of an odd number of machines."*
>
> — *ZooKeeper Administrator's Guide*http://zookeeper.apache.org/doc/r3.4.10/zookeeperAdmin.html

For example, if you only have two ZooKeeper nodes and one goes down, 50% of available servers is not a majority, so ZooKeeper will no longer serve requests. However, if you have three ZooKeeper nodes and one goes down, you have 66% of available servers available, and ZooKeeper will continue normally while you repair the one down node. If you have 5 nodes, you could continue operating with two down nodes if necessary.

The parameters are as follows:

`tickTime`

> Part of what ZooKeeper does is to determine which servers are up and running at any given time, and the minimum session time out is defined as two "ticks". The `tickTime` parameter specifies, in miliseconds, how long each tick should be.

`dataDir`

This is the directory in which ZooKeeper will store data about the cluster. This directory should start out empty.

`clientPort`

This is the port on which Solr will access ZooKeeper.

Once this file is in place, you're ready to start the ZooKeeper instance

Setting up a ZooKeeper Ensemble

With an external ZooKeeper ensemble, you need to set things up just a little more carefully as compared to the Getting Started example.

The difference is that rather than simply starting up the servers, you need to configure them to know about and talk to each other first. So your original `zoo.cfg` file might look like this:

```
dataDir=/var/lib/zookeeperdata/1
clientPort=2181
initLimit=5
syncLimit=2
server.1=localhost:2888:3888
server.2=localhost:2889:3889
server.3=localhost:2890:3890
```

Here you see three new parameters:

initLimit

Amount of time, in ticks, to allow followers to connect and sync to a leader. In this case, you have 5 ticks, each of which is 2000 milliseconds long, so the server will wait as long as 10 seconds to connect and sync with the leader.

syncLimit

Amount of time, in ticks, to allow followers to sync with ZooKeeper. If followers fall too far behind a leader, they will be dropped.

server.X

These are the IDs and locations of all servers in the ensemble, the ports on which they communicate with each other. The server ID must additionally stored in the `<dataDir>/myid` file and be located in the `dataDir` of each ZooKeeper instance. The ID identifies each server, so in the case of this first instance, you would create the file `/var/lib/zookeeperdata/1/myid` with the content "1".

Now, whereas with Solr you need to create entirely new directories to run multiple instances, all you need for a new ZooKeeper instance, even if it's on the same machine for testing purposes, is a new configuration file. To complete the example you'll create two more configuration files.

tickTime

the length of a single tick, which is the basic time unit used by ZooKeeper, as measured in milliseconds. It is used to regulate heartbeats, and timeouts. For example, the minimum session timeout will be two ticks.

initLimit

(No Java system property)

Amount of time, in ticks (see tickTime), to allow followers to connect and sync to a leader. Increased this value as needed, if the amount of data managed by ZooKeeper is large.

syncLimit

(No Java system property)

Amount of time, in ticks (see tickTime), to allow followers to sync with ZooKeeper. If followers fall too far behind a leader, they will be dropped.

https://www.linkedin.com/pulse/real-comparison-nosql-databases-hbase-cassandra-mongodb-sahu/

Hbase:

Possible Region States

- OFFLINE: the region is offline and not opening
- OPENING: the region is in the process of being opened
- OPEN: the region is open and the RegionServer has notified the master
- FAILED_OPEN: the RegionServer failed to open the region
- CLOSING: the region is in the process of being closed
- CLOSED: the RegionServer has closed the region and notified the master
- FAILED_CLOSE: the RegionServer failed to close the region
- SPLITTING: the RegionServer notified the master that the region is splitting
- SPLIT: the RegionServer notified the master that the region has finished splitting
- SPLITTING_NEW: this region is being created by a split which is in progress
- MERGING: the RegionServer notified the master that this region is being merged with another region
- MERGED: the RegionServer notified the master that this region has been merged
- MERGING_NEW: this region is being created by a merge of two regions
-

Possible Region States

- `OFFLINE`: the region is offline and not opening
- `OPENING`: the region is in the process of being opened
- `OPEN`: the region is open and the RegionServer has notified the master
- `FAILED_OPEN`: the RegionServer failed to open the region
- `CLOSING`: the region is in the process of being closed
- `CLOSED`: the RegionServer has closed the region and notified the master
- `FAILED_CLOSE`: the RegionServer failed to close the region
- `SPLITTING`: the RegionServer notified the master that the region is splitting
- `SPLIT`: the RegionServer notified the master that the region has finished splitting
- `SPLITTING_NEW`: this region is being created by a split which is in progress
- `MERGING`: the RegionServer notified the master that this region is being merged with another region
- `MERGED`: the RegionServer notified the master that this region has been merged
- `MERGING_NEW`: this region is being created by a merge of two regions
-

http://hbase.apache.org/book.html

HBase is a type of "NoSQL" database. "NoSQL" is a general term meaning that the database isn't an RDBMS which supports SQL as its primary access language, but there are many types of NoSQL databases: BerkeleyDB is an example of a local NoSQL database, whereas HBase is very much a distributed database. Technically speaking, HBase is really more a "Data Store" than "Data Base" because it lacks many of the features you find in an RDBMS, such as typed columns, secondary indexes, triggers, and advanced query languages, etc.
However, HBase has many features which supports both linear and modular scaling. HBase clusters expand by adding RegionServers that are hosted on commodity class servers. If a cluster expands from 10 to 20 RegionServers, for example, it doubles both in terms of storage and as

well as processing capacity. An RDBMS can scale well, but only up to a point - specifically, the size of a single database server - and for the best performance requires specialized hardware and storage devices. HBase features of note are:

- Strongly consistent reads/writes: HBase is not an "eventually consistent" DataStore. This makes it very suitable for tasks such as high-speed counter aggregation.
- Automatic sharding: HBase tables are distributed on the cluster via regions, and regions are automatically split and re-distributed as your data grows.
- Automatic RegionServer failover
- Hadoop/HDFS Integration: HBase supports HDFS out of the box as its distributed file system.
- MapReduce: HBase supports massively parallelized processing via MapReduce for using HBase as both source and sink.
- Java Client API: HBase supports an easy to use Java API for programmatic access.
- Thrift/REST API: HBase also supports Thrift and REST for non-Java front-ends.
- Block Cache and Bloom Filters: HBase supports a Block Cache and Bloom Filters for high volume query optimization.
- Operational Management: HBase provides build-in web-pages for operational insight as well as JMX metrics.

Yarn Aggregation:

PROBLEMS WITH EXISTING LOG MANAGEMENT:

- **Truncation**: Users complain about truncated logs more often than that. Few users need to access the complete logs. No limit really satisfies all the users – 100KB works for some, but not so much for others.

- **Run-away tasks**: TaskTrackers/DataNodes can still go out of disk space due to excessive logging as truncation only happens *after* tasks finish.
- **Insufficient retention**: Users complain about the log-retention time. No admin configured limit satisfies all the users – the default of one day works for many users but many gripe about the fact that they cannot do post analysis.
- **Accesss**: Serving logs over HTTP by the TTs is completely dependent on the job finish time and the retention time – not perfectly reliable.
- **Collection status**: The same non-reliability with a log-collector tool – one needs to build more solutions to detect when the log-collection starts and finishes and when to switch over to such collected logs from the usual files managed by TTs.
- **mapred.userlog.limit.kb** increases memory usage of each task (specifically with lots of logging), doesn't work with more generic application containers supported by YARN.
- **Historical analysis**: All of these logs are served over HTTP **only** as long as they exist on the local node – if users want to do post analysis, they will have to employ some log-collection and build more tooling around that.
- **Load-balancing & SPOF**: All the logs are written to a single log-directory – no load balancing across disks & if that one disk is down, all jobs lose all logs.

Admitting that existing solutions were only really stop-gap to a more fundamental problem, we took the opportunity to do the right thing by enabling in-platform log aggregation.

LOG-AGGREGATION IN YARN: DETAILS

So, instead of truncating user-logs, and leaving them on individual nodes for certain time as done by the TaskTracker, the NodeManager addresses the

logs' management issue by providing the option to move these logs securely onto a file-system (FS), for e.g. HDFS, after the application completes.

- Logs for all the containers belonging to a single Application and that ran on a given NM are aggregated and written out to a single (possibly compressed) log file at a configured location in the FS.
- In the current implementation, once an application finishes, one will have
    - an application level log-dir and
    - a per-node log-file that consists of logs for **all** the containers of the application that ran on this node.
- Users have access to these logs via YARN command line tools, the web-UI or directly from the FS – no longer just restricted to a web only interface.
- These logs potentially can be stored for much longer times than was possible in 1.x, given they are stored a distributed file-system.
- We don't need to truncate logs to very small lengths – as long as the log sizes are reasonable, we can afford to store the entire logs.
- In addition to that, while the containers are running, the logs are now written to multiple directories on each node for effective load balancing and improved fault tolerance.
- *AggregatedLogDeletionService* is a service that periodically deletes aggregated logs. Today it is run inside the MapReduce JobHistoryServer only.

USAGE

WEB UI

On the web interfaces, the fact that logs are aggregated is completely hidden from the user.

- While a MapReduce application is running, users can see the logs from the ApplicationMaster UI which redirects to the NodeManager UI
- Once an application finishes, the completed information is owned by the MapReduce JobHistoryServer which again serves user-logs transparently.
- For non-MapReduce applications, we are working on a generic ApplicationHistoryServer that does the same thing.

COMMAND LINE ACCESS

In addition to the web-UI, we also have a command line utility to interact with logs.

[shell]

$ $HADOOP_YARN_HOME/bin/yarn logs

Retrieve logs for completed YARN applications.

usage: yarn logs -applicationId <application ID> [OPTIONS]

general options are:

-appOwner <Application Owner> AppOwner (assumed to be current user if

not specified)

-containerId <Container ID> ContainerId (must be specified if node

address is specified)

-nodeAddress <Node Address> NodeAddress in the format nodename:port

(must be specified if container id is

specified)

[/shell]

So, to print all the logs for a given application, one can simply say

[bash]

yarn logs -applicationId <application ID>

[/bash]

On the other hand, if one wants to get the logs of only one container, the following works

[bash]

yarn logs -applicationId <application ID> -containerId <Container ID> -nodeAddress <Node Address>

[/bash]

The obvious advantage with the command line utility is that now you can use the regular shell utils like grep, sort etc to filter out any specific information that one is looking for in the logs!

ADMINISTRATION

GENERAL LOG RELATED CONFIGURATION PROPERTIES

- **yarn.nodemanager.log-dirs**: Determines where the container-logs are stored on the node when the containers are running. Default is ${yarn.log.dir}/userlogs.
    - An application's localized log directory will be found in ${yarn.nodemanager.log-dirs}/application_${appid}.
    - Individual containers' log directories will be below this, in directories named container_{$containerId}.
    - For MapReduce applications, each container directory will contain the files stderr, stdin, and syslog generated by that container.
    - Other frameworks can choose to write more or less files, YARN doesn't dictate the file-names and number of files.
- **yarn.log-aggregation-enable**: Whether to enable log aggregation or not. If disabled, NMs will keep the logs locally (like in 1.x) and not aggregate them.

PROPERTIES RESPECTED WHEN LOG-AGGREGATION IS ENABLED

- **yarn.nodemanager.remote-app-log-dir**: This is on the default file-system, usually HDFS and indictes where the NMs should aggregate logs to. This *should not* be local file-system, otherwise serving daemons

like history-server will not able to serve the aggregated logs. Default is /tmp/logs.

- **yarn.nodemanager.remote-app-log-dir-suffix**: The remote log dir will be created at {yarn.nodemanager.remote-app-log-dir}/${user}/{thisParam}. Default value is “logs””.
- **yarn.log-aggregation.retain-seconds**: How long to wait before deleting aggregated-logs, -1 or a negative number disables the deletion of aggregated-logs. One needs to be careful and not set this to a too small a value so as to not burden the distributed file-system.
- **yarn.log-aggregation.retain-check-interval-seconds**: Determines how long to wait between aggregated-log retention-checks. If it is set to 0 or a negative value, then the value is computed as one-tenth of the aggregated-log retention-time. As with the previous configuration property, one needs to be careful and not set this to low values. Defaults to -1.
- **yarn.log.server.url**: Once an application is done, NMs redirect web UI users to this URL where aggregated-logs are served. Today it points to the MapReduce specific JobHistory.

PROPERTIES RESPECTED WHEN LOG-AGGREGATION IS DISABLED

- **yarn.nodemanager.log.retain-seconds**: Time in seconds to retain user logs on the individual nodes if log aggregation is disabled. Default is 10800.
- **yarn.nodemanager.log.deletion-threads-count**: Determines the number of threads used by the NodeManagers to clean-up logs once the log-retention time is hit for local log files when aggregation is disabled.

OTHER SETUP INSTRUCTIONS

- The remote root log directory is expected to have the permissions 1777 with *${NMUser}* and directory and group-owned by *${NMGroup}* – group to which NMUser belongs.
- Each application level dir will be created with permission 770, but user-owned by the application-submitter and group owned by *${NMGroup}*. This is so that the application-submitter can access aggregated-logs for his own sake and *${NMUser}* can access or modify the files for log management.
- *${NMGroup}* should be a limited access group so that there are no access leaks.

CONCLUSION

In this post, I've described the motivations for implementing log-aggregation and how it looks for the end user as well as the administrators.

Log-aggregation proved to be a very useful feature so far. There are interesting design decisions that we made and some unsolved challenges with the existing implementation – topics for a future post.

Tags:

COMMENTS

 ledion says:

Your comment is awaiting moderation.

November 22, 2013 at 1:28 am

Log aggregation in Yarn is definitely useful, but it's extremely puzzling to me why TFile was chosen as the format to aggregate the container logs into. More info on this post http://blogs.splunk.com/2013/11/18/hadoop-2-0-rant/

Reply

 Otis Gospodnetic says:

Your comment is awaiting moderation.

November 25, 2013 at 12:58 pm

How would one go about pushing their Hadoop logs into something like Logsene (note: Elasticsearch API) – http://sematext.com/logsene – whether used as SaaS or on premises?

Or is the goal to keep logs local/in the same Hadoop cluster? If logs can just be sent to syslog, then all the logging tools and services that know how to "talk syslog" could take those logs…. and these services, like Logsene above for example, make such logs easily searchable, filterable, graphable, dasboardable, alertable, etc. etc.

As a matter of fact, the above example of "yarn logs -applicationId " feel a lot like a filter by application ID, which is precisely what existing logging tools and services do, among other things.

Reply

 adidas says:

April 28, 2014 at 3:19 am

Wow, that’s what I was searching for, what a information!

present here at this webpage, thanks admin of this

website.

Reply

 Janardhan says:

July 5, 2014 at 8:32 am

Very crisp article to understand, debug and set log parameters in YARN.

Reply

 Son Hai Ha says:

August 29, 2014 at 2:11 am

Hi,

Thanks for great post!

Is it the “yarn logs -applicationId” get the content from /app-logs//logs/ on hdfs and display it out?

I saw that the content there are binary files, not text anymore. Is it because you guys compress the text? Is there anyway that we can read directly the content of the logs there? Thanks

Reply

 Ranga Vasudevan says:

November 5, 2014 at 2:59 am

Very useful. Thanks. Wish there would be a single consolidated documentation available for production deployments that includes gems like this.

Reply

 Jules S. Damji says:

November 5, 2014 at 2:48 pm

Thanks. We'll take that note and pass it on to documentation group.

Reply

 Andrea says:

January 9, 2015 at 10:45 am

Hello,

thanks for the post it is very useful. However, I still cannot get my stdout (println) because the stdout folder is empty. Any idea?

Many Thanks.

Reply

 Praveen S says:

March 1, 2015 at 1:41 am

Hi,

Maybe your stdout logs are printed in a different container. There usually are 3 of them by default. Just change the URL to point to a different container, as such..

http://localhost:8042/node/containerlogs/container_1420734149115_0014_01_000002/andrea

http://localhost:8042/node/containerlogs/container_1420734149115_0014_01_000003/andrea

Reply

 Andrea says:

January 9, 2015 at 11:13 am

Hello,

Thanks for this very useful post. However, I cannot see my stdout (println) output in the container folder http://localhost:8042/node/containerlogs/container_1420734149115_0014_01_000001/andrea. Any idea?

Many Thanks.

Reply

 Sumit says:

June 11, 2015 at 7:56 am

Would these properties apply at a global level only? Can I use them for specific applications only by passing them to AM at runtime?

Reply

 Vinod Kumar Vavilapalli says:

June 12, 2015 at 10:07 am

All of them are cluster / global configurations, configured on NodeManagers.

Reply

 Tarun says:

July 2, 2015 at 5:04 pm

Nice blog,

I have one question, if i set log-aggregation true and then how can i stop local log creation. Are they different?

Thanks

Tarun

Reply

 Nikhil says:

August 14, 2015 at 5:06 am

Thanks for the really useful post. Is the ‘generic ApplicationHistoryServer’ out yet. If yes, how do I access it.

I am running Spark on YARN and would like to access Spark executor logs through Web UI.

Thanks again.

Reply

 Minh says:

October 29, 2015 at 8:09 am

Hi,

Do you know why i’ve got this messages ?

Owner of application_Id is pantunes.

User pantunes and hdfs01 have same group apcku

http://localhost:19888/jobhistory/logs/localhost:45454/container_1445849282248_0112_01_000001/container_1445849282248_0112_01_000001/pantunes

User [hdfs01] is not authorized to view the logs for container_1445849282248_0112_01_000001 in log file [localhost_45454_1446124373814]

No logs available for container container_1445849282248_0112_01_000001

With yarn logs -appOwner patunes -applicationId application_1445849282248_0112, the resultat is ok.

thanks for answer

Reply

 Minh says:

October 29, 2015 at 8:09 am

Hi,

Do you know why i’ve got this messages ?

Owner of application_Id is pantunes.

User pantunes and hdfs01 have same group apcku

http://localhost:19888/jobhistory/logs/localhost:45454/container_1445849282248_0112_01_000001/container_1445849282248_0112_01_000001/pantunes

User [hdfs01] is not authorized to view the logs for container_1445849282248_0112_01_000001 in log file [localhost_45454_1446124373814]

No logs available for container container_1445849282248_0112_01_000001

With yarn logs -appOwner patunes -applicationId application_1445849282248_0112, the resultat is ok.

thanks for answer

Reply

 Venkatesh says:

April 22, 2016 at 7:15 am

Dear Hortonworks, Am using hdp2.4 sandbox, once my job finished (eg. sqoop import job) not able to view the history logs through resource manager UI. If i click history am getting error as

This site can't be reached

sandbox.hortonworks.com's server DNS address could not be found.

Looking yours support!

Mail me with the solution

Reply

 Venkatesh says:

April 22, 2016 at 7:23 am

Dear Hortonworks,

Am using hdp2.4 sandbox, once my job finished (eg. sqoop import job) not able to view the history logs through resource manager UI. If i click history am getting error as below. All the quick links in MR2 getting the same even if history server is started and running fine.

http://sandbox.hortonworks.com:19888/ –> Job history server UI

This site can’t be reached

sandbox.hortonworks.com’s server DNS address could not be found.

Looking yours support!

Mail me with the solution

Reply

 JY says:

June 23, 2016 at 5:10 pm

Since each application level dir is created with permission 770, this prevents ‘others’ to access the logs under it. This would prevent log collectors, like Logstash, from reading the logs. Is this intended behavior? Could you grant at least the read permission? Thanks.

Reply

 Francis Kim says:

June 25, 2016 at 2:41 am

Nice write up Vinod!

Reply

 Raj Mannem says:

March 29, 2017 at 1:03 pm

“In addition to that, while the containers are running, the logs are now written to multiple directories on each node for effective load balancing and improved fault tolerance.”

Does this mean , yarn.nodemanager.log-dirs setting supports(shares) multiple mount directories for load balancing ?

Reply

 karthik says:

June 12, 2017 at 10:39 pm

“yarn.log-aggregation.retain-seconds: How long to wait before deleting aggregated-logs, -1 or a negative number disables the deletion of aggregated-logs. One needs to be careful and not set this to a too small a value so as to not burden the distributed file-system.”

-1 : will not delete any logs – which means lots of small files (logs) causing NM trouble

small value : means, retention is small – so less small files – this is good for NM – but why is it written the opposite?

Reply

 Download TutuApp says:

August 3, 2017 at 11:15 pm

Awesome news for iPhone users here. Now we can download paid apps for free using tutuapp. Download tutuapp for iPhone now.

Reply

 Sreenivas says:

January 10, 2018 at 12:55 am

How can i limit container logs size to 2GB with in the application.

user@ip-address:/data/hadoop/yarn/log/application_1511288551669_9427# du -sch *

HDFS Dir for logs :

/tmp/logs/c20755a/logs/application_5510

https://www.cloudera.com/documentation/enterprise/5-13-x/topics/cdh_ig_yarn_cluster_deploy.html

You must also specify a log aggregation time interval using the `yarn.nodemanager.log-aggregation.roll-monitoring-interval-seconds` property. The logs for running applications are aggregated at the specified interval. The minimum monitoring interval value is 3600 seconds (one hour).

You can also set the monitoring interval value to -1 to disable log aggregation for running applications, and wait until the application finishes running to enable log aggregation.

```
<property>
 <name>yarn.log-aggregation-enable</name>
 <value>true</value>
 </property>

 <property>

<name>yarn.nodemanager.log-aggregation.roll-monitoring-interval-seconds</name>
 <value>3600</value>
 </property>
```

PROPERTIES RESPECTED WHEN LOG-AGGREGATION IS ENABLED

- **yarn.nodemanager.remote-app-log-dir**: This is on the default file-system, usually HDFS and indictes where the NMs should aggregate logs to. This *should not* be local file-system, otherwise serving daemons

like history-server will not able to serve the aggregated logs. Default is /tmp/logs.

- **yarn.nodemanager.remote-app-log-dir-suffix**: The remote log dir will be created at {yarn.nodemanager.remote-app-log-dir}/${user}/{thisParam}. Default value is “logs””.
- **yarn.log-aggregation.retain-seconds**: How long to wait before deleting aggregated-logs, -1 or a negative number disables the deletion of aggregated-logs. One needs to be careful and not set this to a too small a value so as to not burden the distributed file-system.
- **yarn.log-aggregation.retain-check-interval-seconds**: Determines how long to wait between aggregated-log retention-checks. If it is set to 0 or a negative value, then the value is computed as one-tenth of the aggregated-log retention-time. As with the previous configuration property, one needs to be careful and not set this to low values. Defaults to -1.
- **yarn.log.server.url**: Once an application is done, NMs redirect web UI users to this URL where aggregated-logs are served. Today it points to the MapReduce specific JobHistory.

https://docs.hortonworks.com/HDPDocuments/Ambari-2.6.0.0/bk_ambari-security/content/kerberos_overview.html

https://docs.hortonworks.com/HDPDocuments/Ambari-2.6.0.0/bk_ambari-security/content/kerberos_overview.html

Strongly authenticating and establishing a user’s identity is the basis for secure access in Hadoop. Users need to be able to reliably “identify” themselves and then have that identity propagated throughout the Hadoop cluster. Once this is done, those users can access resources (such as files or directories) or interact with the cluster (like running

MapReduce jobs). Besides users, Hadoop cluster resources themselves (such as Hosts and Services) need to authenticate with each other to avoid potential malicious systems or daemon's "posing as" trusted components of the cluster to gain access to data.

Hadoop uses Kerberos as the basis for strong authentication and identity propagation for both user and services. Kerberos is a third party authentication mechanism, in which users and services rely on a third party - the Kerberos server - to authenticate each to the other. The Kerberos server itself is known as the **Key Distribution Center**, or **KDC**. At a high level, it has three parts:

- A database of the users and services (known as **principals**) that it knows about and their respective Kerberos passwords
- An **Authentication Server** (**AS**) which performs the initial authentication and issues a **Ticket Granting Ticket** (**TGT**)
- A **Ticket Granting Server** (**TGS**) that issues subsequent service tickets based on the initial **TGT**

A **user principal** requests authentication from the AS. The AS returns a TGT that is encrypted using the user principal's Kerberos password, which is known only to the user principal and the AS. The user principal decrypts the TGT locally using its Kerberos password, and from that point forward, until the ticket expires, the user principal can use the TGT to get service tickets from the TGS. Service tickets are what allow a principal to access various services.

Because cluster resources (hosts or services) cannot provide a password each time to decrypt the TGT, they use a special file, called a **keytab**, which contains the resource principal's authentication credentials. The set of hosts, users, and services over which the Kerberos server has control is called a **realm**.

Terminology

Term	Description
Key Distribution Center, or KDC	The trusted source for authentication in a Kerberos-enabled environment.
Kerberos KDC Server	The machine, or server, that serves as the Key Distribution Center (KDC).
Kerberos Client	Any machine in the cluster that authenticates against the KDC.
Principal	The unique name of a user or service that authenticates against the KDC.
Keytab	A file that includes one or more principals and their keys.
Realm	The Kerberos network that includes a KDC and a number of Clients.
KDC Admin Account	An administrative account used by Ambari to create principals and generate keytabs in the KDC.

Resetting the Root Password

A common scenario for a Linux administrator is that the root password has gone

missing. If that happens, you need to reset it. The only way to do that is by booting

into minimal mode, which allows you to log in without entering a password. To do

so, follow these steps:

1. On system boot, press e when the GRUB 2 boot menu is shown.

2. Enter rd.break as boot argument to the line that loads the kernel and press

Ctrl+X to boot with this option.

3. You'll now be dropped at the end of the boot stage where initramfs is loaded,

just before a mount of the root file system on the directory /.

4. Type mount -o remount,rw /sysroot to get read/write access to the system

image.

5. At this point, make the contents of the /sysimage directory your new root

directory by typing chroot /sysroot .

6. Now you can enter passwd and set the new password for the user root.

7. Because at this very early boot stage SELinux has not been activated yet, the

context type on /etc/shadow will be messed up. If you reboot at this point,

no one will be able to log in. So you must make sure that the context type is

set correctly. To do this, at this point you should load the SELinux policy by

using load_policy -i .444

Red Hat RHCSA/RHCE 7 Cert Guide

8. Now you can manually set the correct context type to /etc/shadow. To do this,

type chcon -t shadow_t /etc/shadow .

9. Reboot. You can now log in with the changed password for user root.

Understanding the RHEL 7 Boot Procedure

To fix boot issues, it is essential to have a good understanding of the boot proce-

dure. If issues occur during boot, you need to be able to judge in which phase of the

boot procedure the issue occurs so that you can select the appropriate tool to fix the

issue.

The following steps summarize how the boot procedure happens on Linux.

1. Performing POST: The machine is powered on. From the system firmware, which can be the modern Universal Extended Firmware Interface (UEFI) or the classical Basic Input Output System (BIOS), the Power-On Self-Test (POST) is executed, and the hardware that is required to start the system is initialized.

2. Selecting the bootable device: Either from the UEFI boot firmware or from the Master Boot Record, a bootable device is located.

3. Loading the boot loader: From the bootable device, a boot loader is located. On Red Hat, this is usually GRUB 2.

4. Loading the kernel: The boot loader may present a boot menu to the user,

or can be configured to automatically start a default operating system. To load

Linux, the kernel is loaded together with the initramfs. The initramfs contains

kernel modules for all hardware that is required to boot, as well as the initial

scripts required to proceed to the next stage of booting. On RHEL 7, the init-

ramfs contains a complete operational system (which may be used for trouble-

shooting purposes).

5. Starting /sbin/init: Once the kernel is loaded into memory, the first of all

processes is loaded, but still from the initramfs. This is the /sbin/init process,

which on Red Hat is linked to systemd. The udev daemon is loaded as well to

take care of further hardware initialization. All this is still happening from the initramfs image.

6. Processing initrd.target: The systemd process executes all units from the initrd.target, which prepares a minimal operating environment, where the root file system on disk is mounted on the /sysroot directory. At this point, enough is loaded to pass to the system installation that was written to the hard drive.

7. Switching to the root file system: The system switches to the root file system that is on disk and at this point can load the systemd process from disk as well.Chapter 19: Troubleshooting the Boot Procedure 433

8. Running the default target: Systemd looks for the default target to execute and runs all of its units. In this process, a login screen is presented, and the

user can authenticate. Notice that the login prompt can be prompted before

all systemd unit files have been loaded successfully. So, seeing a login prompt

does not necessarily mean that your server is fully operational yet.

In each of the phases listed, issues may occur because of misconfiguration or other

problems. Table 19.2 summarizes where a specific phase is configured and what you

can do to troubleshoot if things go wrong.

Troubleshooting has always been a part of the RHCSA exam. If you encoun-

ter an issue, make sure that you can identify in which phase of the boot procedure

it occurs and what you can do to fix it.

```
# free -m
```

```
# top
# lsof -p $AMBARI_PID
# netstat -tnlpa | grep  $AMBARI_PID
```