HDP ZooKeeper ACLs 3

Apache ZooKeeper ACLs

Date of Publish: 2018-07-15

http://docs.hortonworks.com

Contents

Apache ZooKeeper ACLs Best Practices

As more and more components begin to rely on ZooKeeper within a Hadoop cluster, there are various permissions that need to be maintained to ensure the integrity and security of the znodes. These permissions are different from component to component. You must follow the required steps for tightening the ZooKeeper ACLs or permissions when provisioning a secure cluster as a best practices guideline.

Introduction

Some components only use ZooKeeper when they are running in their component specific HA mode. Others have separate secure and unsecure ACLs defined and switch between which to enforce based on the component knowledge of whether the cluster is secured or not.

In general, ACLs are pretty open and assume an unsecure cluster by default. These permissions need to be hardened for secure clusters in order to avoid inappropriate access or modification of this critical platform state.

Unaffected Components

The following components require no action:

- Ambari
  - ZooKeeper Usage: Ambari does not use ZooKeeper; however it does install, configure, and manage it so that services running on the cluster can use it.
  - Default ACLs: None. Ambari does not create or use any znodes.
  - Security Best Practice ACLs/Permissions and Required Steps: None. Ambari does not create or use any znodes.
- Calcite
- DataFu
- Knox
- MapReduce
- Phoenix
  - ZooKeeper Usage: Phoenix does not use ZooKeeper on its own. All usages are covered in the HBase section.
  - Security Best Practice ACLs/Permissions and Required Steps: None. HBase correctly protects all znodes in ZooKeeper automatically.
- Pig
- Spark
- Sqoop
- Stargate/HBase RestServer
  - No ZooKeeper usage outside of normal HBase client usage.
- Tez
- Zeppelin
-

ZooKeeper ACLs Best Practices: Accumulo

You must follow the best practices for tightening the ZooKeeper ACLs or permissions for Accumulo when provisioning a secure cluster.

- ZooKeeper Usage:
  - /accumulo - Parent ZNode for all of Accumulo use in ZooKeeper
  - /accumulo/$UUID - Parent ZNode for a specific Accumulo instance

    - /accumulo/instances - Contains mappings of human-readable Accumulo names to the UUID
    - /accumulo/$UUID/users - Accumulo user database
    - /accumulo/$UUID/problems - Persisted advertisement of reported problems in Accumulo
    - /accumulo/$UUID/root_tables - The “root” Accumulo table (points to the Accumulo metadata table)
    - /accumulo/$UUID/hdfs_reservations - ZNode to coordinate unique directories in HFDS for bulk imports of Accumulo files to a table
    - /accumulo/$UUID/gc - Advertisement and leader election for Accumulo GarbageCollector
    - /accumulo/$UUID/table_locks - RW-locks per Accumulo table
    - /accumulo/$UUID/fate - Parent znode for Accumulo’s FATE (distributed, multi-step transactions)
    - /accumulo/$UUID/tservers - Advertisement and ephemeral znodes(keep-alive) for TabletServers
    - /accumulo/$UUID/tables - The “database” of Accumulo tables (metadata)
    - /accumulo/$UUID/namespaces - The “database” of Accumulo namespaces (metadata)
    - /accumulo/$UUID/next_file - Coordinates unique name generation for files in HDFS
    - /accumulo/$UUID/config - Dynamic configuration for Accumulo
    - /accumulo/$UUID/masters - Advertisement and leader election for the Accumulo Master
    - /accumulo/$UUID/monitor - Advertisement and leader election for the Accumulo Monitor
    - /accumulo/$UUID/bulk_failed_copyq - Tracking files to bulk import which failed
    - /accumulo/$UUID/recovery - Used to coordinate recovery of write-ahead logs
- Default ACLs:
    - All znodes not specified otherwise are world-readable and cdrwa ‘accumulo’. Those below are not world-readable:

      /accumulo/$UUID/users/*
- Security Best Practice ACLs/Permissions and Required Steps:
    - The user does not need to alter any ACLs in ZooKeeper. Accumulo protects all ZNodes automatically.

ZooKeeper ACLs Best Practices: Ambari Solr

You must follow the best practices for tightening the ZooKeeper ACLs or permissions for Ambari Solr when provisioning a secure cluster.

Ambari Solr is used by LogSearch, Ranger and Atlas.

- ZooKeeper Usage:
    - /ambari-solr - Solr node for storing collection, leaders, configuration, etc.
- Default ACLs:
    - /ambari-solr - world:anyone:cdrwa
- Security Best Practice ACLs/Permissions and Required Steps:
    - /ambari-solr - world:anyone:r
    - /ambari-solr - sasl:solr:cdrwa

ZooKeeper ACLs Best Practices: Atlas

You must follow the best practices for tightening the ZooKeeper ACLs or permissions for Atlas when provisioning a secure cluster.

- ZooKeeper Usage:
    - /apache_atlas - Root zookeeper node which is configured for curator, under which nodes for leader election are created.

    - /apache_atlas/active_server_info - Znode used in HA environments for storing active server information.
    - /apache_atlas/setup_in_progress - Transient Znode used to ensure some setup steps are executed only from one instance. This gets deleted after use and should normally not be seen.
- Default ACLs:
    - All znodes have world:anyone:cdrwa by default.
- Security Best Practice ACLs/Permissions and Required Steps:
    - No user intervention is required for creating/using the Znodes. They are all managed internally by Atlas. Atlas exposes two configuration properties that define the auth and ACL - to use while creating these Znodes. Ambari should configure these correctly for a secure cluster. The recommended configuration is atlas.server.ha.zookeeper.auth=sasl:atlas@<domain.com> and atlas.server.ha.zookeeper.acl=sasl:atlas@<domain.com> , where <domain.com> should be replaced with the right value of the atlas service user principal. (Assuming atlas is the service user name). When set this way, the ACLs for all znodes will be atlas.server.ha.zookeeper.acl=sasl:atlas@<domain.com>:cdrwa. (Note we don't allow configuration of the permissions from Ambari).

ZooKeeper ACLs Best Practices: HBase

You must follow the best practices for tightening the ZooKeeper ACLs or permissions for HBase when provisioning a secure cluster.

- ZooKeeper Usage:
    - /hbase-unsecure - Default znode for unsecured clusters
    - /hbase-secure - Default znode used for secured clusters
- Default ACLs:
    - /hbase-unsecure - world:hbase:cdrwa
        - All children ZNodes are also world cdrwa
    - Open for global read, write protected: world:anyone:r, sasl:hbase:cdrwa
        - /hbase-secure
        - /hbase-secure/master
        - /hbase-secure/meta-region-server
        - /hbase-secure/hbaseid
        - /hbase-secure/table
        - /hbase-secure/rs
    - No global read, r/w protected: sasl:hbase:cdrwa:
        - /hbase-secure/acl
        - /hbase-secure/namespace
        - /hbase-secure/backup-masters
        - /hbase-secure/online-snapshot
        - /hbase-secure/draining
        - /hbase-secure/replication
        - /hbase-secure/region-in-transition
        - /hbase-secure/splitWAL
        - /hbase-secure/table-lock
        - /hbase-secure/recovering-regions
        - /hbase-secure/running
        - /hbase-secure/tokenauth
    - Security Best Practice ACLs/Permissions and Required Steps:

- HBase code determines which ACL to enforce based on the configured security mode of the cluster/hbase. Users are not expected to perform any modification of ZooKeeper ACLs on ZNodes and users should not alter any ACLs by hand.

ZooKeeper ACLs Best Practices: HDFS/WebHDFS

You must follow the best practices for tightening the ZooKeeper ACLs or permissions for HDFS/WebHDFS when provisioning a secure cluster.

- ZooKeeper Usage:
  - hadoop-ha - hdfs zkfc automatic NameNode failover
- Default ACLs:
  - hadoop-ha - world: anyone:cdrwa
- Security Best Practice ACLs/Permissions and Required Steps:
  - hadoop-ha - sasl: nn:cdrwa
- Existing SmartSense rule recommends ACL of sasl:nn:rwcda for secured clusters. To set this:

  1. Set ha.zookeeper.acl to sasl:nn:rwcda:

  - Using Ambari:

    Add ha.zookeeper.acl with value sasl:nn:rwcda in Configs>Advanced>Custom core-site.
  - Manually:

    Add this to core-site. xml as root user:

```
<property>
    <name>ha.zookeeper.acl</name>
    <value>sasl:nn:rwcda</value>
</property>
```

  2. Add this HADOOP_ZKFC_OPTS export:

  - Using Ambari:

    In Configs > Advanced > Advanced hadoop-env > hadoop-env template, add the following:

```
export HADOOP_ZKFC_OPTS="Dzookeeper.sasl.client=true
                                    -
Dzookeeper.sasl.client.username=zookeeper
                                    -
Djava.security.auth.login.config=/etc/hadoop/conf/hdfs_jaas.conf
                                    -
Dzookeeper.sasl.clientconfig=Client ${HADOOP_ZKFC_OPTS}"
```

  - Manually:

    Add this to hadoop-env.sh as root user:

```
export HADOOP_ZKFC_OPTS="Dzookeeper.sasl.client=true
                                    -
Dzookeeper.sasl.client.username=zookeeper
                                    -
Djava.security.auth.login.config=/etc/hadoop/conf/hdfs_jaas.conf
                                    -
Dzookeeper.sasl.clientconfig=Client ${HADOOP_ZKFC_OPTS}"
```

3. On two Namenodes, create /etc/hadoop/conf/hdfs_jaas.conf as root user with the following contents:

```
Client {
          com.sun.security.auth.module.Krb5LoginModule required
          useKeyTab=true
          storeKey=true
          useTicketCache=false
          keyTab="/etc/security/keytabs/nn.service.keytab"
          principal="nn/<HOST>@EXAMPLE.COM";
};
```

   nn/<HOST>@EXAMPLE.COM must be changed to the actual hostname and realm, e.g. nn/c6401.ambari.apache.org@EXAMPLE.COM . To get actual principal, on two Namenodes, run the command as hdfs user: klist -k /etc/security/keytabs/nn.service.keytab.
4. Stop the two ZKFCs.
5. On one of Namenodes, run the command as hdfs user: hdfs zkfc -formatZK -force.
6. Start the two ZKFCs.

One of two Namenodes may be stopped in the process, or standby Namenode may be transitioned to active one. Start the stopped namenode if any.

ZooKeeper ACLs Best Practices: Hive/HCatalog

You must follow the best practices for tightening the ZooKeeper ACLs or permissions for Hive/HCatalog when provisioning a secure cluster.

- ZooKeeper Usage:
  - /hiveserver2 - The parent znode used by HiveServer2 when supporting dynamic service discovery. Each server instance creates an ephemeral znode under this namespace. Exposed via the hive config: hive.server2.zookeeper.namespace
  - /hivedelegation/METASTORE - HA ONLY - The root path for token store data, used by Metastore servers to store delegation tokens. Exposed via hive config: hive.cluster.delegation.token.store.zookeeper.znode
  - /hivedelegation/HIVESERVER2 - HA ONLY - The root path for token store data, used by HiveServer2 servers to store delegation tokens. Exposed via hive config: hive.cluster.delegation.token.store.zookeeper.znode
  - /hive_zookeeper_namespace - Used by ZooKeeper-based implementation of Hive's LockMgr (ZooKeeperHiveLockManager) if used. This usage is writable-to by any user as it tries to co-ordinate locking among multiple users. Controlled by hive config : hive.zookeeper.namespace. In addition, which LockMgr we use is also controlled by hive config : hive.lock.manager. (Note also, that if ACID is used, we do not use a ZooKeeper-based lock manager)
  - /llap-<sasl|unsecure>/user-<user_name> is used by LLAP to store cluster node locations. Should be writable by hive, readable by anyone. LLAP takes care of enforcing the ACLs for the secure path.
  - /zkdtsm_<cluster_id>/ZKDTSMRoot/* is used by LLAP token/secret manager, in secure cluster only. Should only be accessible by hive. LLAP sets and validates the ACLs.
- Default ACLs:
  - /hiveserver2 - world:anyone:r
  - /hiveserver2 - sasl:hive:cdrwa
  - /hivedelegation - world:anyone:r
  - /hivedelegation - sasl:hive:cdrwa
  - /hive_zookeeper_namespace - completely-open
  - /llap-sasl/user-<user_name> - sasl:hive:cdrwa, world:anyone:r
  - /llap-unsecure/user-<user_name> - world:anyone:cdrwa
  - /zkdtsm_<cluster_id>/ZKDTSMRoot/* - sasl:hive:cdrwa

Note that ACLs are considered recursively applied to nodes inside these roots - i.e., /hivedelegation/METASTORE, /hivedelegation/HIVESERVER2, or /hiveserver2/<first_server> .

- Security Best Practice ACLs/Permissions and Required Steps:
  - /hiveserver2 - world:anyone:r
  - /hiveserver2 - sasl:hive:cdrwa
  - /hivedelegation - world:anyone:r
  - /hivedelegation - sasl:hive:cdrwa
  - /hive_zookeeper_namespace - completely-open
  - /llap-sasl/user-<user_name> - sasl:hive:cdrwa, world:anyone:r
  - /llap-unsecure/user-<user_name> - world:anyone:cdrwa
  - /zkdtsm_<cluster_id>/ZKDTSMRoot/* - sasl:hive:cdrwa

  Note that ACLs are considered recursively applied to nodes inside these roots - i.e., /hivedelegation/METASTORE, /hivedelegation/HIVESERVER2, or /hiveserver2/<first_server> .

ZooKeeper ACLs Best Practices: Kafka

You must follow the best practices for tightening the ZooKeeper ACLs or permissions for Kafka when provisioning a secure cluster.

- ZooKeeper Usage:
  - /controller - Kafka Znode for controller leader election
  - /brokers - Kafka Znode for broker metadata
  - /kafka-acl - Kafka Znode for SimpleAclAuthorizer ACL storage
  - /admin - Kafka admin tool metadata
  - /isr_change_notification - Track changes to In Sync Replication
  - /controller_epoch - Track movement of controller
  - /consumers - Kafka Consumer list
  - /config - Entity configuration
- Default ACLs:
  - N/A -->
- Security Best Practice ACLs/Permissions and Required Steps:
  - /controller - world:anyone:r
  - /controller - sasl:kafka:cdrwa
  - /brokers - world:anyone:cdrwa
  - /kafka-acl - sasl:kafka:cdrwa
  - /admin - world:anyone:cdrwa
  - /isr_change_notification - world:anyone:r
  - /isr_change_notification - sasl:kafka:cdrwa
  - /controller_epoch - world:anyone:cdrwa
  - /consumers - world:anyone:cdrwa
  - /config - world:anyone:cdrwa

  When security is enabled zookeeper.set.acl=true should be in kafkaConfig. Which is not happening now. Users can add this using **Advanced Property** zookeeper.set.acl and add a new zkroot to zookeepr.connect = "host.name:2181:/kafka" to create new nodes as it won't update the ACLs on existing node. Alternatively, they can use kafka.service.keytab to log into zookeeper and set ACLs recursively.

ZooKeeper ACLs Best Practices: Oozie

You must follow the best practices for tightening the ZooKeeper ACLs or permissions for Oozie when provisioning a secure cluster.

- ZooKeeper Usage:
  - Used to coordinate multiple Oozie servers.
- Default ACLs:

  In a secure cluster, Oozie restricts the access to Oozie Znodes to the oozie principals only using Kerberos backed ACLs.
  - /oozie - node that stores oozie server information in HA mode

  Default ACLs:
  - /oozie - world:anyone:cdrwa
- Security Best Practice ACLs/Permissions and Required Steps:
  - Set oozie.zookeeper.secure to secure

ZooKeeper ACLs Best Practices: Ranger

You must follow the best practices for tightening the ZooKeeper ACLs or permissions for Ranger when provisioning a secure cluster.

- ZooKeeper Usage:
  - Ranger does not use ZooKeeper directly. Only if **Audit to Solr** is enabled and Solr is configured in SolrCloud mode, Solr nodes will need to access zookeeper node /ranger_audits.

    /ranger_audits
- Default ACLs:
  - /ranger_audits - world:anyone:cdrwa
- Security Best Practice ACLs/Permissions and Required Steps:
  - Only Solr needs access to this Znode:

    /ranger_audits - sasl:solr:cdrwa
  - After enabling SolrCloud, edit the Ranger collection path permission on Znode:
    1. SSH to the cluster where SolrCloud is present.
    2. Go to /usr/hdp/<version>/zookeeper/bin.
    3. Run ./zkCli.sh -server <FQDN SolrCloud host>:2181
    4. After it connects, run: ls /
    5. Verify there is a folder for the Ranger Solr collection.
    6. Execute getAcl /ranger_audits and if the permission is for world,anyone: cdrwa, restrict the permission to “sasl:solr:cdrwa” using this command: setAcl /ranger_audits sasl:solr:cdrwa.
    7. Repeat the above step for all clusters where SolrCloud is installed.

```
[zk: as-ha-27-3.openstacklocal:2181(CONNECTED) 0] ls /
[zookeeper, rmstore, ranger_audits]
[zk: as-ha-27-3.openstacklocal:2181(CONNECTED) 1] getAcl /ranger_audits
'world,'anyone
: cdrwa
```

```
[zk: as-ha-27-3.openstacklocal:2181(CONNECTED) 2] setAcl /ranger_audits
 sasl:solr:cdrwa
cZxid = 0x200000037
ctime = Wed Jun 29 10:40:24 UTC 2016
mZxid = 0x200000037
mtime = Wed Jun 29 10:40:24 UTC 2016
pZxid = 0x200000056
cversion = 7
dataVersion = 0
aclVersion = 1
ephemeralOwner = 0x0
dataLength = 0
numChildren = 7
[zk: as-ha-27-3.openstacklocal:2181(CONNECTED) 3] getAcl /ranger_audits

'sasl,'solr
: cdrwa
[zk: as-ha-27-3.openstacklocal:2181(CONNECTED) 4]
```

ZooKeeper ACLs Best Practices: Ranger KMS/Hadoop KMS

You must follow the best practices for tightening the ZooKeeper ACLs or permissions for Ranger KMS/Hadoop KMS when provisioning a secure cluster.

- ZooKeeper Usage:
  - If multiple instances of KMS are configured, both Ranger KMS and Hadoop KMS use zookeeper znode / hadoop-kms to store HTTP cookie signature secret. See “Http Authentication Signature” section here.

    /hadoop-kms - <HTTP cookie signature secret>
- Default ACLs:
  - /hadoop-kms - world:anyone:cdrwa
- Security Best Practice ACLs/Permissions and Required Steps:
  - /hadoop-kms - sasl:kms:cdrwa
  - Ranger KMS uses the user kms. Only KMS needs access to this znode. This path (hadoop.kms.authentication.signer.secret.provider.zookeeper.path) can be configured in Ambari for Ranger KMS. Set the ACL using these steps:

    1. SSH to the cluster where Ranger KMS is present.
    2. Go to /usr/hdp/<version>/zookeeper/bin
    3. Run ./zkCli.sh -server <FQDN of Ranger KMS host>:2181”
    4. After it connects, run: ls /
    5. Verify there is a folder as specified in hadoop.kms.authentication.signer.secret.provider.zookeeper.path property of Ranger KMS configuration.
    6. Execute getAcl /hadoop-kms and if the permission is for world,anyone: cdrwa, restrict the permission to sasl:kms:cdrwa using this command: setAcl /hadoop-kms sasl:kms:cdrwa.
    7. Repeat the above step for all the clusters where Ranger KMS is installed.

```
[zk: dk-test-0706-3.openstacklocal:2181(CONNECTED) 0] getAcl /hadoop-kms
'world,'anyone
: cdrwa
[zk: dk-test-0706-3.openstacklocal:2181(CONNECTED) 4] setAcl /hadoop-kms
 sasl:kms:cdrwa
cZxid = 0x20000001e
ctime = Tue Jun 07 12:22:58 UTC 2016
mZxid = 0x20000001e
mtime = Tue Jun 07 12:22:58 UTC 2016
```

```
pZxid = 0x20000001f
cversion = 1
dataVersion = 0
aclVersion = 1
ephemeralOwner = 0x0
dataLength = 0
numChildren = 1
[zk: dk-test-0706-3.openstacklocal:2181(CONNECTED) 5] getAcl /hadoop-
kms
'sasl,'kms
: cdrwa
[zk: dk-test-0706-3.openstacklocal:2181(CONNECTED) 6]
```

ZooKeeper ACLs Best Practices: Storm

You must follow the best practices for tightening the ZooKeeper ACLs or permissions for Storm when provisioning a secure cluster.

- ZooKeeper Usage:
  - /storm - All data for storm metadata, Storm's root znode
- Default ACLs:
  - /storm - world:anyone:cr
  - /storm - sasl:storm-PRD1:cdrwa

    Where -PRD1 comes from StormClient Principal and Ambari creates the principal with storm-<cluster_name> .
- Security Best Practice ACLs/Permissions and Required Steps:
  - /storm - world:anyone:cr
  - /storm - sasl:storm-PRD1:cdrwa

    Where -PRD1 comes from StormClient Principal and Ambari creates the principal with storm-<cluster_name> .

  -->

ZooKeeper ACLs Best Practices: WebHCat

You must follow the best practices for tightening the ZooKeeper ACLs or permissions for WebHCat when provisioning a secure cluster.

- ZooKeeper Usage:
  - /templeton-hadoop - WebHCat stores status of jobs that users can query in zookeeper (if ZooKeeperStorage is configured to find out the status of jobs - it can also use HDFS for this storage). WebHCat typically will create three znodes inside this root : “jobs”, “overhead” and “created”. This root node is exposed via config : templeton.storage.root. In addition, whether or not ZooKeeperStorage is used is configured by another config parameter : templeton.storage.class. Both these parameters are part of webhcat-site.xml. These nodes are altered from launcher map task as well, which runs as the end user.
- Default ACLs:
  - /templeton-hadoop - world:anyone:cdrwa
- Security Best Practice ACLs/Permissions and Required Steps:
  - /templeton-hadoop - world:anyone:cdrwa

ZooKeeper ACLs Best Practices: YARN

You must follow the best practices for tightening the ZooKeeper ACLs or permissions for YARN when provisioning a secure cluster.

- ZooKeeper Usage:
  - /yarn-leader-election - used for RM leader election
  - /rmstore - used for storing RM application state
- Default ACLs:
  - /yarn-leader-election - world:anyone:cdrwa
  - /rmstore - world:anyone:cdrwa
- Security Best Practice ACLs/Permissions and Required Steps:
  - /yarn-leader-election - world:anyone:r
  - /yarn-leader-election - sasl:rm:rwcda
  - /rmstore - world:anyone:r
  - /rmstore - sasl:rm:rwcda

ZooKeeper ACLs Best Practices: YARN Registry

You must follow the best practices for tightening the ZooKeeper ACLs or permissions for YARN Registry when provisioning a secure cluster.

The YARN registry is a location into which statically and dynamically deployed applications can register service endpoints; client applications can look up these entries to determine the URLs and IPC ports with which to communicate with a service.

It is implemented as a zookeeper tree: services register themselves as system services, under the registry path /system, or user services, which are registered under /users/USERNAME where USERNAME is the name of the user registering the service.

As the purpose of the mechanism is to allow arbitrary clients to look up a service, the entries are always world readable. No secrets should be added to service entries.

In insecure mode, all registry paths are world readable and writeable: nothing may be trusted.

In a secure cluster, the registry is designed to work as follows:

1. Kerberos + SASL provides the identification and authentication.
2. /system services can only be registered by designated system applications (YARN, HDFS, etc)/
3. User-specific services can only be registered by the user deploying the application.
4. If a service is registered under a user's path, it may be trusted, and any published public information (such as HTTPS certifications) assumed to have been issued by the user.
5. All user registry entries should also be registered as world writeable with the list of system accounts defined in hadoop.registry.system.accounts; this is a list of ZK SASL-authenticated accounts to be given full access. This is needed to support system administration of the entries, especially automated deletion of old entries after application failures.
6. The default list of system accounts are yarn, mapred, hdfs, and hadoop; these are automatically associated with the Kerberos realm of the process interacting with the registry, to create the appropriate sasl:account@REALM ZK entries.
7. If applications are running from different realms, the configuration option hadoop.registry.kerberos.realm must be set to the desired realm, or hadoop.registry.system.accounts configured with the full realms of the accounts.

8. There is support for ZooKeeper id:digest authentication; this is to allow a user's short-lived YARN applications to register service endpoints without needing the Kerberos TGT. This needs active use by the launching application (which must explicitly create a user service node with an id:digest permission, or by setting hadoop.registry.user.accounts, to the list of credentials to be permitted.
9. System services must not use id:digest authentication —nor should they need to; any long-lived service already needs to have a kerberos keytab.
10. The per-user path for their user services, /users/USERNAME , is created by the YARN resource manager when users launch services, if the RM is launched with the option hadoop.registry.rm.enabled set to true.
11. When hadoop.registry.rm.enabled is true, the RM will automatically purge application and container service records when the applications and containers terminate.
12. Communication with ZK is over SASL, using the java.security.auth.login.config system property to configure the binding. The specific JAAS context to use can be set in hadoop.registry.jaas.context if the default value, Client, is not appropriate.

ZK Paths and Permissions:

All paths are world-readable; permissions are set up when the RM creates the root entry and user paths and hadoop.registry.secure=true.

Path	**Role**	**Permissions**
/registry	Base registry path	yarn, hdfs, mapred, hadoop : cdrwa
/registry/system	System services	yarn, hdfs, mapred, hadoop : cdrwa
/registry/users	Users	yarn, hdfs, mapred, hadoop : cdrwa
/registry/users/USER	The registry tree for the user USER.	USER: rwa yarn, hdfs, mapred, hadoop : cdrwa

Configuration options for secure registry access

Name	**Recommended Value**
hadoop.registry.secure	true
hadoop.registry.rm.enabled	true
hadoop.registry.system.accounts	sasl:yarn@, sasl:mapred@, sasl:hdfs@, sasl:hadoop@ Grants system accounts write access to the root registry paths. A tighter version would be sasl:yarn@ which will only give the RM the right to manipulate these, or explicitly declare a realm, such as sasl:yarn@EXAMPLE
hadoop.registry.kerberos.realm	(empty) The Kerberos realm to use when converting the system accounts to full realms. If left empty, uses the realm of the user
hadoop.registry.user.accounts	(empty)
hadoop.registry.client.auth	kerberos How to authenticate with ZK. Alternative (insecure) options: anonymous, digest.
hadoop.registry.jaas.context	Client The JAAS context to use for registry clients to authenticate with ZooKeeper.

ZooKeeper ACLs Best Practices: ZooKeeper

You must follow the best practices for tightening the ZooKeeper ACLs or permissions for ZooKeeper when provisioning a secure cluster.

- ZooKeeper Usage:
  - /zookeeper - node stores metadata of ZooKeeper itself.
  - /zookeeper/quota stores quota information. In the Apache ZooKeeper 3.5 release line.
  - /zookeeper/config stores dynamic reconfiguration information, but this is not applicable to HDP, which bases its ZooKeeper release off of the Apache ZooKeeper 3.4 release line.
- Default ACLs:
  - /zookeeper - world:anyone:cdrwa
- Security Best Practice ACLs/Permissions and Required Steps:

  The following steps must be manually performed by users who are using the ZooKeeper quota feature. Components in HDP do not use this feature by default -- most users do not need to execute the following commands.

  - /zookeeper - sasl:zookeeper:cdrwa
  - setAcl sasl:zookeeper:rwcda