Cloudera Runtime 7.0.0

Configuring Apache Hadoop YARN Security

Date published: 2019-08-21
Date modified:

https://docs.cloudera.com/

Legal Notice

Contents

Managing Access Control Lists

An Access Control List (ACL) is a list of specific permissions or controls that allow individual users or groups to perform specific actions upon specific objects, as well as defining what operations are allowed on a given object.

YARN ACLs do not deny access; rather, they identify a user, list of users, group, or list of groups who can access a particular object.

Like HDFS ACLs, YARN ACLs provide a way to set different permissions for specific named users or named groups. ACLs enhance the traditional permissions model by defining access control for arbitrary combinations of users and groups instead of a single owner/user or a single group.

YARN ACL rules

Learn about the YARN ACL rules to which all YARN ACLs must adhere.

All YARN ACLs must adhere to the following rules:

- Special Values:
  - The wildcard character (*) indicates that everyone has access.

    Note: You cannot use the wildcard (*) character along with a list of users and/or groups in the same ACL. If you use the wildcard character it must be the only item in the ACL.
  - A single space entry indicates that no one has access.
- If there are no spaces in an ACL, then all entries (the listed users and/or groups) are considered authorized users.
- Group names in YARN Resource Manager ACLs are case sensitive. So, if you specify an uppercase group name in the ACL, it will not match the group name resolved from the Active Directory because Active Directory group names are resolved in lowercase.
- If an ACL starts with a single space, then it must consist of groups only.
- All entries after the occurrence of a second single space in an ACL are ignored.
- There are no ACLs that deny access to a user or group. However, if you wish to block access to an operation entirely, enter a value for a non-existent user or group (for example,'NOUSERS NOGROUPS'), or simply enter a single space. By doing so, you ensure that no user or group maps to a particular operation by default.
- If you wish to deny only a certain set of users and/or groups, specify every single user and/or group that requires access. Users and/or groups that are not included are "implicitly" denied access.

YARN ACL syntax

Example of YARN ACL syntax.

Note: In all cases where a single space is required, you will see: <single space>.

- Users only

  ```
  user1,user2,userN
  ```

  Use a comma-separated list of user names. Do not place spaces after the commas separating the users in the list.

- Groups only

  ```
  <single space>HR,marketing,support
  ```

  You must begin group-only ACLs with a single space. Group-only ACLs use the same syntax as users, except each entry is a group name rather than user name.
- Users and Groups

  ```
  fred,alice,haley<single space>datascience,marketing,support
  ```

  A comma-separated list of user names, followed by a single space, followed by a comma-separated list of group names. This sample ACL authorizes access to users “fred”, “alice”, and “haley”, and to those users in the groups “datascience”, “marketing”, and “support”.

Examples

The following ACL entry authorizes access only to the members of “my_group”:

```
<single space>my_group
```

The following ACL entry authorizes access to anyone:

```
*
```

The following ACL authorizes access to the users “john”, “jane”, and the group “HR”:

```
john,jane<single space>HR
```

In this example, six groups (“group_1” through “group_6”) are defined in the system. The following ACL authorizes access to a subset of the defined groups, allowing access to all members of groups 1 through 5 (and implicitly denies access to members of the group “group_6”):

```
<single space>group_1,group_2,group_3,group_4,group_5
```

YARN ACL types

There are three different kinds of YARN ACL types: YARN Admin ACLs, YARN Queue ACLs and YARN Application ACLs.

YARN Admin ACLs

Use the YARN Admin ACLs to allow users to run YARN administrator sub-commands, which are executed via the yarn rmadmin <command>.

Use the YARN Admin ACL to allow users to run YARN administrator sub-commands, which are executed via the yarn rmadmin <command>.

Important: The YARN Admin ACL is triggered and applied only when you run YARN sub-commands via yarn rmadmin <cmd>. If you run other YARN commands via the YARN command line (for example, starting the Resource or Node Manager), it does not trigger the YARN Admin ACL check or provide the same level of security.

The default YARN Admin ACL is set to the wildcard character (*), meaning all users and groups have YARN Administrator access and privileges. So after YARN ACL enforcement is enabled, (via the yarn.acl.enable property) every user has YARN ACL Administrator access. Unless you wish for all users to have YARN Admin ACL access, edit the yarn.admin.acl setting upon initial YARN configuration, and before enabling YARN ACLs.

A typical YARN Admin ACL looks like the following, where the system's Hadoop administrator and multiple groups are granted access:

```
hadoopadmin<space>yarnadmgroup,hadoopadmgroup
```

YARN Application ACLs

Use Application ACLs to provide a user or group access to an application.

Use Application ACLs to provide a user and/or group–neither of whom is the owner–access to an application. The most common use case for Application ACLs occurs when you have a team of users collaborating on or managing a set of applications, and you need to provide read access to logs and job statistics, or access to allow for the modification of a job (killing the job) and/or application. Application ACLs are set per application and are managed by the application owner.

Users who start an application (the owners) always have access to the application they start, which includes the application logs, job statistics, and ACLs. No other user can remove or change owner access. By default, no other users have access to the application data because the Application ACL defaults to " " (single space), which means no one has access.

MapReduce Application ACL

Create and use the following MapReduce Application ACLs to view YARN logs.

- mapreduce.job.acl-view-job

  Provides read access to the MapReduce history and the YARN logs.
- mapreduce.job.acl-modify-job

  Provides the same access as mapreduce.job.acl-view-job, and also allows the user to modify a running job.

  Note: Job modification is currently limited to killing the job. No other YARN system modifications are supported.

During a search or other activities, you may come across the following two legacy settings from MapReduce; they are not supported by YARN. Do not use them:

- mapreduce.cluster.acls.enabled
- mapreduce.cluster.administrators

Related Information

Viewing application logs

Spark Application ACL

Spark ACLs using a separate property for users and groups.

Both user and group lists use a comma-separated list of entries. The wildcard character "*" allows access to anyone, and the single space " " allows access to no one. Enable Spark ACLs using the property spark.acls.enable, which is set to false by default (not enabled) and must be changed to true to enforce ACLs at the Spark level.

Create and use the following Application ACLs for the Spark application:

- Set spark.acls.enable to true (default is false).
- Set spark.admin.acls and spark.admin.acls.groups for administrative access to all Spark applications.
- Set spark.ui.view.acls and spark.ui.view.acls.groups for view access to the specific Spark application.
- Set spark.modify.acls and spark.modify.acls.groups for administrative access to the specific Spark application.

Viewing application logs

The MapReduce Application ACL mapreduce.job.acl-view-job determines whether or not you can view an application log.

Aaccess is evaluated through the following ACLs:

- YARN Admin and Queue ACLs
- Application ACLs

After an application is in the “finished” state, logs are aggregated, depending on your cluster setup. You can access the aggregated logs via the MapReduce History server web interface. Aggregated logs are stored on shared cluster storage, which in most cases is HDFS. You can also share log aggregation via storage options like S3 or Azure by modifying the yarn.nodemanager.remote-app-log-dir setting in Cloudera Manager to point to either S3 or Azure, which should already be configured.

The shared storage on which the logs are aggregated helps to prevent access to the log files via file level permissions. Permissions on the log files are also set at the file system level, and are enforced by the file system: the file system can block any user from accessing the file, which means that the user cannot open/read the file to check the ACLs that are contained within.

In the cluster storage use case of HDFS, you can only access logs that are aggregated via the:

- Application owner
- Group defined for the MapReduce History server

When an application runs, generates logs, and then places the logs into HDFS, a path/structure is generated (for example: /tmp/logs/john/logs/application_1536220066338_0001). So access for the application owner "john" might be set to 700, which means read, write, execute; no one else can view files underneath this directory. If you don’t have HDFS access, you will be denied access. Command line users identified in mapreduce.job.acl-view-job are also denied access at the file level. In such a use case, the Application ACLs stored inside the aggregated logs will never be evaluated because the Application ACLs do not have file access.

For clusters that do not have log aggregation, logs for running applications are kept on the node where the container runs. You can access these logs via the Resource Manager and Node Manager web interface, which performs the ACL checks.

Killing an application

The Application ACL mapreduce.job.acl-modify-job determines whether or not a user can modify a job, but in the context of YARN, this only allows the user to kill an application.

The kill action is application agnostic and part of the YARN framework. Other application types, like MapReduce or Spark, implement their own kill action independent of the YARN framework. MapReduce provides the kill actions via the mapred command.

For YARN, the following three groups of users are allowed to kill a running application:

- The application owner
- A cluster administrator defined in yarn.admin.acl
- A queue administrator defined in aclAdministerApps for the queue in which the application is running

Note that for the queue administrators, ACL inheritance applies, as described earlier.

Related Information

Example: Killing an application in the "Production" queue

Application ACL evaluation

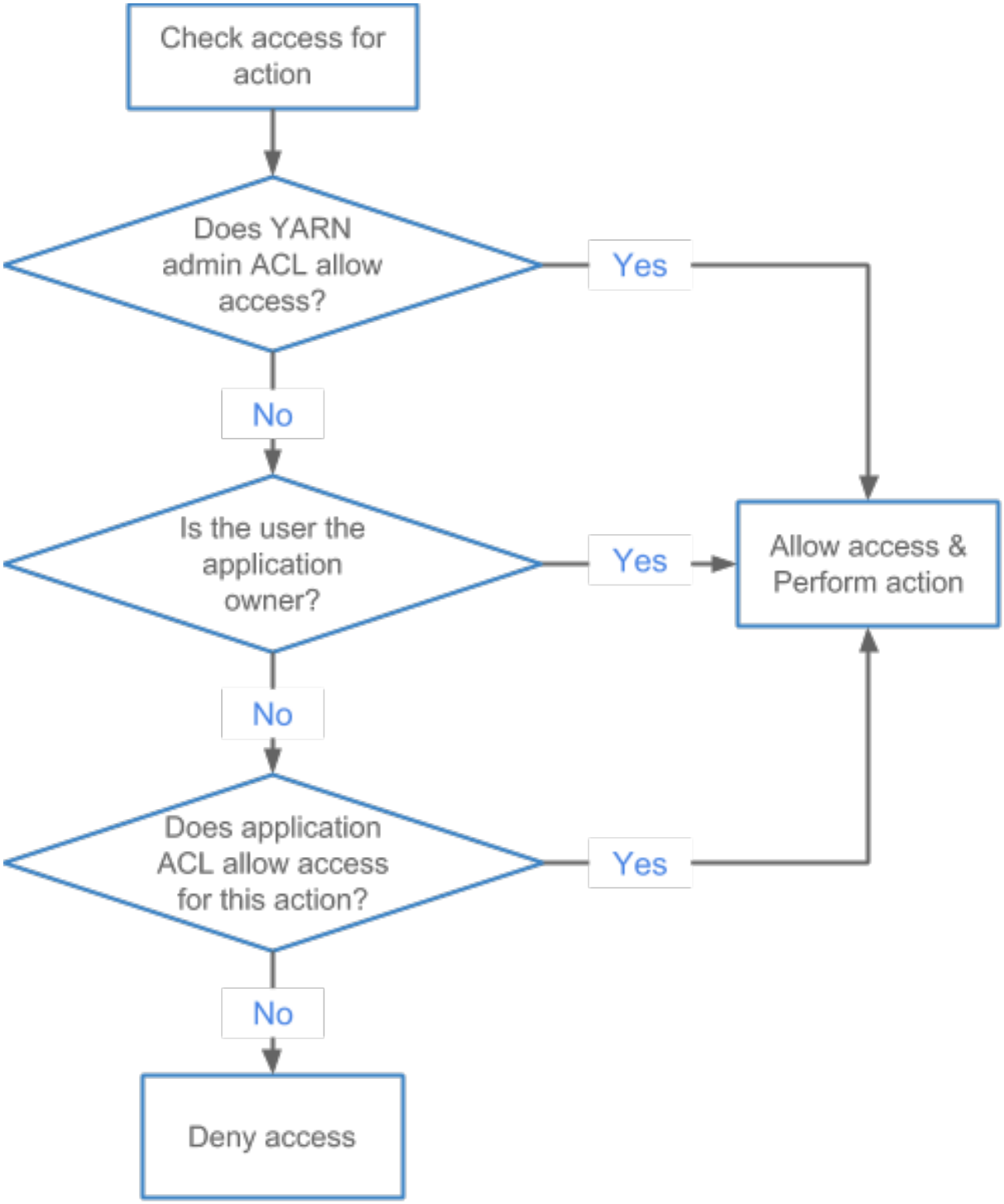
The better you understand how YARN ACLs are evaluated, the more prepared you will be to define and configure the various YARN ACLs available to you.

For example, if you enable user access in Administrator ACLs, then you must be aware that user may have access to/see sensitive data, and should plan accordingly. So if you are the administrator for an entire cluster, you also have access to the logs for running applications, which means you can view sensitive information in those logs associated with running the application.

Best Practice: A best practice for securing an environment is to set the YARN Admin ACL to include a limited set of users and or groups.

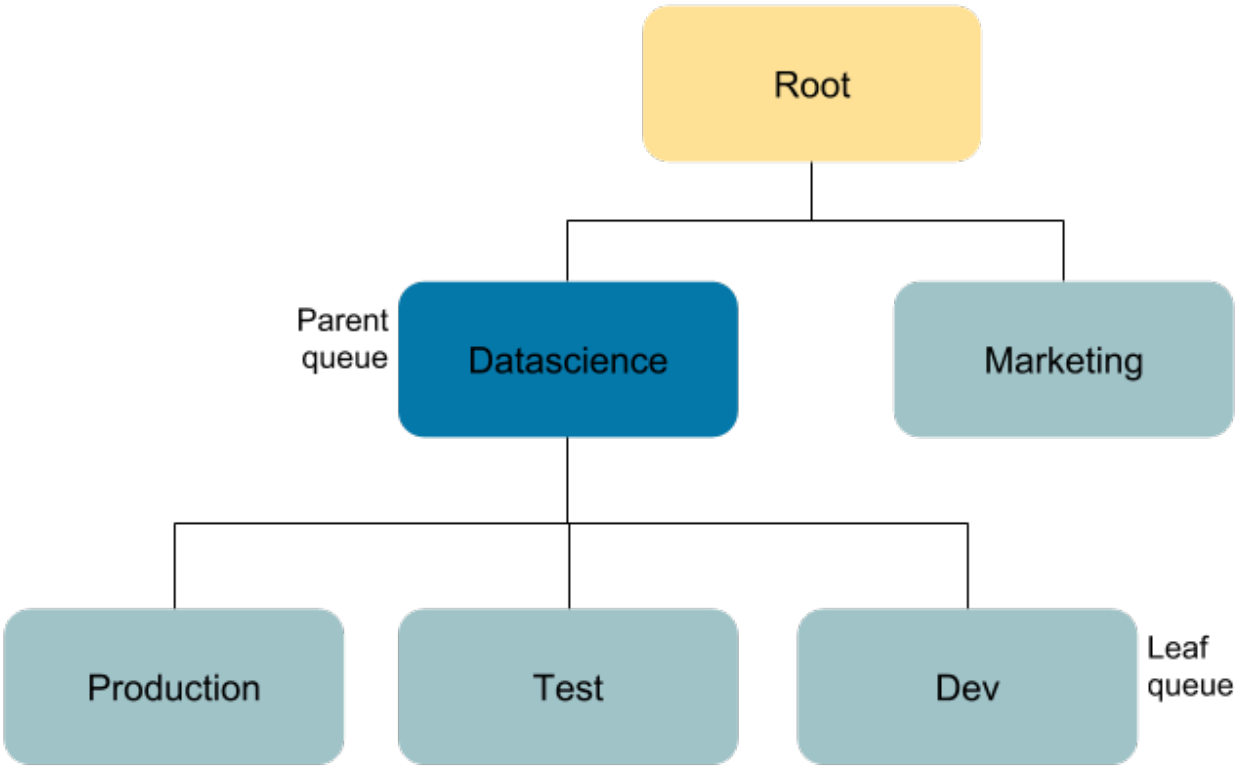
The following diagram shows the evaluation flow for Application ACLs:

Figure 1: Application ACL Evaluation Flow



The following diagram shows a sample queue structure, starting with leaf queues on the bottom, up to root queue at the top:

Figure 2: Queue Structure



Example: Killing an application in the "Production" queue

Provide the privilege to a user to kill an application in a specific queue.

For this Application ACL evaluation flow example, assume the following for application_1536220066338_0001 running in the queue "Production":

- Application owner: John
- "Datascience" queue administrator: Jane
- YARN cluster administrator: Bob

In this use case, John attempts to kill the application, which is allowed because he is the application owner.

Working as the queue administrator, Jane attempts to kill a job in the queue "Production", which she can do as the queue administrator of the parent queue.

Bob is the YARN cluster administrator and he is also listed as a user in the Admin ACL. He attempts to kill the job for which he is not the owner, but because he is the YARN cluster administrator, he can kill the job.

Related Information

Killing an application

Example: Moving the application and viewing the log in the "Test" queue

Provide the privileges to a user to move application between queues and to view a log in a specific queue.

For this Application ACL evaluation flow example, assume the following for application_1536220066338_0002 running in the queue "Test":

- Application owner: John
- "Marketing" and "Dev" queue administrator: Jane
- Jane has log view rights via the mapreduce.job.acl-view-job ACL
- YARN cluster administrator: Bob

In this use case, John attempts to view the logs for his job, which is allowed because he is the application owner.

Jane attempts to access application_1536220066338_0002 in the queue "Test" to move the application to the "Marketing" queue. She is denied access to the "Test" queue via the queue ACLs–so she cannot submit to or administer the queue "Test". She is also unable to kill a job running in queue "Test". She then attempts to access the logs for application_1536220066338_0002 and is allowed access via the mapreduce.job.acl-view-job ACL.

Bob attempts to access application_1536220066338_0002 in the queue "Test" to move the application to the "Marketing" queue. As the YARN cluster administrator, he has access to all queues and can move the application.

Note: Permissions on the log files are also set at the filesystem level and are enforced by the filesystem: the filesystem can block you from accessing the file, which means that you can not open/read the file to check the ACLs that are contained in the file.

Enable ACLs

Access control lists restrict who can submit work to dynamic resource pools and administer them. You can decide if you want to enable ACL use.

About this task

You can specify whether ACLs are checked using Cloudera Manager.

Procedure

1. In Cloudera Manager, click Clusters.
2. Select the *CLUSTER NAME*.
3. Click Dynamic Resource Pool Configuration.
   If the cluster has a YARN service, the YARN / Resource Pool tab displays. If the cluster has an Impala service enabled, the Impala Admission Control / Resource Pool tab displays.
4. Click the YARN tab.
5. Click Access Control Settings.
6. In the Enable ResourceManagel ACLs property, select the YARN service.
7. Click Save Changes.
8. Click the *Stale Service Restart* icon that is next to the service to invoke the cluster restart wizard.
9. Click Restart Stale Services.
10. Select Re-deploy client configuration.
11. Click Restart Now.

Configure ACLs

Once you have enabled ACLs, you can configure which users and groups can submit and kill applications in any resource pool.

Procedure

1. In Cloudera Manager, click Clusters.
2. Select the *CLUSTER NAME*.
3. Click Dynamic Resource Pool Configuration.
   If the cluster has a YARN service, the YARN / Resource Pool tab displays. If the cluster has an Impala service enabled, the Impala Admission Control / Resource Pool tab displays.
4. Click the YARN tab.
5. Click Access Control Settings.

6. In the Admin ACL property, specify which users and groups can submit and kill application.

   Note: Group names in YARN Resource Manager ACLs are case sensitive. Consequently, if you specify an uppercase group name in the ACL, it will not match the group name resolved from the Active Directory because the Active Directory group name is resolved in lowercase.
7. Click Save Changes.
8. Click the *Stale Service Restart* icon that is next to the service to invoke the cluster restart wizard.
9. Click Restart Stale Services.
10. Select Re-deploy client configuration.
11. Click Restart Now.

Using YARN with a secure cluster

When you are using YARN on a secure cluster you need to do some further configurations to ensure YARN is working properly.

Configure YARN for long-running applications

Long-running applications need additional configuration since the default settings only allow the hdfs user's delegation tokens a maximum lifetime of 7 days which is not always sufficient.

About this task

Long-running applications such as Spark Streaming jobs will need additional configuration since the default settings only allow the hdfs user's delegation tokens a maximum lifetime of 7 days which is not always sufficient.

You can work around this by configuring the ResourceManager as a proxy user for the corresponding HDFS NameNode so that the ResourceManager can request new tokens when the existing ones are past their maximum lifetime. YARN will then be able to continue performing localization and log-aggregation on behalf of the hdfs user.

Procedure

1. In Cloudera Manager, select the YARN service.
2. Click the Configuration tab.
3. Search for yarn-site.xml.
4. Find the ResourceManager Advanced Configuration Snippet (Safety Valve) for yarn-site.xml.
5. Click the plus icon and add the followoing:
   - Name: yarn.resourcemanager.proxy-user-privileges.enabled
   - Value: true
6. Click Save Changes.
7. Go back to the home page by clicking the Cloudera Manager logo.
8. Select the HDFS service.
9. Click the Configuration tab.
10. Search for core-site.xml.
11. Find the Cluster-wide Advances Configuration Snippet (Safety Valve) for core-site.xml.
12. Click the plus icon and add the following:
    - Name: hadoop.proxyuser.yarn.hosts
    - Value: *

13. Click the plus icon again and add the following:
    - Name: hadoop.proxyuser.yarn.groups
    - Value: *
14. Click Save Changes.
15. Click the *Stale Service Restart* icon that is next to the service to invoke the cluster restart wizard.
16. Click Restart Stale Services.
17. Select Re-deploy client configuration.
18. Click Restart Now.

Configure TLS/SSL for core Hadoop services

TLS/SSL for the core Hadoop services (HDFS and YARN) must be enabled as a group.

TLS/SSL for the core Hadoop services (HDFS and YARN) must be enabled as a group. Enabling TLS/SSL on HDFS is required before it can be enabled on YARN.

Note: If you enable TLS/SSL for HDFS, you must also enable it for YARN.

The steps in the following topics include enabling Kerberos authentication for HTTP Web-Consoles. Enabling TLS/SSL for the core Hadoop services on a cluster without enabling authentication displays a warning.

Before You Begin

- Before enabling TLS/SSL, keystores containing certificates bound to the appropriate domain names will need to be accessible on all hosts on which at least one HDFS or YARN daemon role is running.
- Since HDFS and YARN daemons act as TLS/SSL clients as well as TLS/SSL servers, they must have access to truststores. In many cases, the most practical approach is to deploy truststores to all hosts in the cluster, as it may not be desirable to determine in advance the set of hosts on which clients will run.
- Keystores for HDFS and YARN must be owned by the hadoop group, and have permissions 0440 (that is, readable by owner and group). Truststores must have permissions 0444 (that is, readable by all)
- Cloudera Manager supports TLS/SSL configuration for HDFS and YARN at the service level. For each of these services, you must specify absolute paths to the keystore and truststore files. These settings apply to all hosts on which daemon roles of the service in question run. Therefore, the paths you choose must be valid on all hosts.

  An implication of this is that the keystore file names for a given service must be the same on all hosts. If, for example, you have obtained separate certificates for HDFS daemons on hosts node1.example.com and node2.example.com, you might have chosen to store these certificates in files called hdfs-node1.keystore and hdfs-node2.keystore (respectively). When deploying these keystores, you must give them both the same name on the target host — for example, hdfs.keystore.
- Multiple daemons running on a host can share a certificate. For example, in case there is a DataNode and an Oozie server running on the same host, they can use the same certificate.

Configure TLS/SSL for HDFS

Enabling TLS/SSL on HDFS is required before it can be enabled on YARN.

About this task

Enabling TLS/SSL on HDFS is required before it can be enabled on YARN.

Cloudera recommends you enable web UI authentication for the HDFS service. Web UI authentication uses SPNEGO. After enabling this, you cannot access the Hadoop web consoles without a valid Kerberos ticket and proper client-side configuration.

Procedure

1. In Cloudera Manager, select the HDFS service.
2. Click the Configuration tab.
3. Search for TLS/SSL.
4. Find and edit the following properties according to you cluster configuration:

Property	Description
Hadoop TLS/SSL Server Keystore File Location	Path to the keystore file containing the server certificate and private key.
Hadoop TLS/SSL Server Keystore File Password	Password for the server keystore file.
Hadoop TLS/SSL Server Keystore Key Password	Password that protects the private key contained in the server keystore.

If you are not using the default trustore, do the following:

5. Configure TLS/SSL client trustore properties.

Important: The HDFS properties below define a cluster-wide default truststore that can be overridden by YARN.

Property	Description
Cluster-Wide Default TLS/SSL Client Truststore Location	Path to the client truststore file. This truststore contains certificates of trusted servers, or of Certificate Authorities trusted to identify servers.
Cluster-Wide Default TLS/SSL Client Truststore Password	Password for the client truststore file.

If you want to enable web UI authentication for the HDFS service, do the following:

6. Search for web consoles.
7. Find the Enable Authentication for HTTP Web-Consoles property.
8. Check the property to enable web UI authentication.

Note: This is effective only if security is enabled for the HDFS service.

9. Click Save Changes.
10. Configure TLS/SSL for YARN.

Related Information

Configure TLS/SSL for YARN

Configure TLS/SSL for YARN

If you enabled TLS/SSL for HDFS, you must also enable it for YARN.

About this task

If you enable TLS/SSL for HDFS, you must also enable it for YARN.

Cloudera recommends to enable Web UI authentication for YARN.

Procedure

1. In Cloudera Manager, select the YARN service.
2. Click the Configuration tab.
3. Search for TLS/SSL.

4. Find and edit the following properties according to your cluster configuration:

Property	Description
Hadoop TLS/SSL Server Keystore File Location	Path to the keystore file containing the server certificate and private key.
Hadoop TLS/SSL Server Keystore File Password	Password for the server keystore file.
Hadoop TLS/SSL Server Keystore Key Password	Password that protects the private key contained in the server keystore.

If you want to override the cluster-wide defaults set by the HDFS properties, do the following:

5. Configure the following TLS/SSL client truststore properties for YARN.

Property	Description
TLS/SSL Client Truststore File Location	Path to the client truststore file. This truststore contains certificates of trusted servers, or of Certificate Authorities trusted to identify servers.
TLS/SSL Client Truststore File Password	Password for the client truststore file.

If you want to enable Web UI authentication for YARN, do the following:

6. Search for web consoles.
7. Find the Enable Authentication for HTTP Web-Consoles property.
8. Check the property to enable web UI authentication.

Note: This is effective only if security is enabled for the HDFS service.

9. Click Save Changes.
10. Go back to the home page, by clicking the Cloudera Manager logo.
11. Select the HDFS service.
12. Click the Configuration tab.
13. Search for Hadoop SSL Enabled.
14. Find and select the Hadoop SSL Enabled property.
    The SSL communication for HDFS and YARN is enabled.
15. Click Save Changes.
16. Click the *Stale Service Restart* icon that is next to the service to invoke the cluster restart wizard.
17. Click Restart Stale Services.
18. Select Re-deploy client configuration.
19. Click Restart Now.

Related Information

Configure TLS/SSL for HDFS

Linux Container Executor

A setuid binary called container-executor is part of the hadoop-yarn package and is installed in /var/lib/hadoop-yarn/bin/container-executor.

This container-executor program, which is used on YARN only and supported on GNU/Linux only, runs the containers as the user who submitted the application. It requires all user accounts to be created on the cluster hosts where the containers are launched. It uses a setuid executable that is included in the Hadoop distribution. The NodeManager uses this executable to launch and kill containers. The setuid executable switches to the user who has submitted the application and launches or kills the containers. For maximum security, this executor sets up restricted permissions and user/group ownership of local files and directories used by the containers such as the shared objects,

jars, intermediate files, and log files. As a result, only the application owner and NodeManager can access any of the local files/directories including those localized as part of the distributed cache.

Parcel Deployments

In a parcel deployment the container-executor file is located inside the parcel at /opt/cloudera/parcels/CDH/lib/hadoop-yarn/bin/container-executor. For the /var/lib mount point, setuid should not be a problem. However, the parcel could easily be located on a different mount point. If you are using a parcel, ensure that the mount point for the parcel directory is without the -nosuid and -noexec options.

Note: Regardless of the used mount point, -nosuid and -noexec options should be omitted when mounting.

The container-executor program must have a very specific set of permissions and ownership to function correctly. In particular, it must:

1. Be owned by root.
2. Be owned by a group that contains only the user running the YARN daemons.
3. Be setuid.
4. Be group readable and executable. This corresponds to the ownership root:yarn and the permissions 6050.

```
---Sr-s--- 1 root yarn 91886 2012-04-01 19:54 container-executor
```

Important: Configuration changes to the Linux container executor could result in local NodeManager directories (such as usercache) being left with incorrect permissions. To avoid this, when making changes using either Cloudera Manager or the command line, first manually remove the existing NodeManager local directories from all configured local directories (yarn.nodemanager.local-dirs), and let the NodeManager recreate the directory structure.

Troubleshooting

A list of numeric error codes communicated by the container-executor to the NodeManager that appear in the /var/log/hadoop-yarn NodeManager log.

Table 1: Numeric error codes that are applicable to the container-executor in YARN, but are used by the LinuxContainerExecutor only.

Numeric Code	Name	Description
1	INVALID_ARGUMENT_NUMBER	• Incorrect number of arguments provided for the given container-executor command • Failure to initialize the container localizer
2	INVALID_USER_NAME	The user passed to the container-executor does not exist.
3	INVALID_COMMAND_PROVIDED	The container-executor does not recognize the command it was asked to run.
5	INVALID_NM_ROOT	The passed NodeManager root does not match the configured NodeManager root (yarn.nodemanager.local-dirs), or does not exist.
6	SETUID_OPER_FAILED	Either could not read the local groups database, or could not set UID or GID
7	UNABLE_TO_EXECUTE_CONTAINER_SCRIPT	The container-executor could not run the container launcher script.
8	UNABLE_TO_SIGNAL_CONTAINER	The container-executor could not signal the container it was passed.

Numeric Code	Name	Description
9	INVALID_CONTAINER_PID	The PID passed to the container-executor was negative or 0.
18	OUT_OF_MEMORY	The container-executor couldn't allocate enough memory while reading the container-executor.cfg file, or while getting the paths for the container launcher script or credentials files.
20	INITIALIZE_USER_FAILED	Couldn't get, stat, or secure the per-user NodeManager directory.
21	UNABLE_TO_BUILD_PATH	The container-executor couldn't concatenate two paths, most likely because it ran out of memory.
22	INVALID_CONTAINER_EXEC_PERMISSIONS	The container-executor binary does not have the correct permissions set.
24	INVALID_CONFIG_FILE	The container-executor.cfg file is missing, malformed, or has incorrect permissions.
25	SETSID_OPER_FAILED	Could not set the session ID of the forked container.
26	WRITE_PIDFILE_FAILED	Failed to write the value of the PID of the launched container to the PID file of the container.
255	Unknown Error	This error has several possible causes. Some common causes are: • User accounts on your cluster have a user ID less than the value specified for the min.user.id property in the cont ainer-executor.cfg file. The default value is 1000 which is appropriate on Ubuntu systems, but may not be valid for your operating system. For information about setting min. user.id in the container-executor.cfg file. • This error is often caused by previous errors; look earlier in the log file for possible causes.

Table 2: Exit status codes apply to all containers in YARN. These exit status codes are part of the YARN framework and are in addition to application specific exit codes that can be set.

Numeric Code	Name	Description
0	SUCCESS	Container has finished succesfully.
-1000	INVALID	Initial value of the container exit code. A container that does not have a COMPLETED state will always return this status.
-100	ABORTED	Containers killed by the framework, either due to being released by the application or being 'lost' due to node failures, for example.
-101	DISKS_FAILED	Container exited due to local disks issues in the NodeManager node. This occurs when the number of good nodemanager-local-directories or nodemanager-log-directories drops below the health threshold.
-102	PREEMPTED	Containers preempted by the framework. This does not count towards a container failure in most applications.
-103	KILLED_EXCEEDED_VMEM	Container terminated because of exceeding allocated virtual memory limit.
-104	KILLED_EXCEEDED_PMEM	Container terminated because of exceeding allocated physical memory limit.

Numeric Code	Name	Description
-105	KILLED_BY_APPMASTER	Container was terminated on request of the application master.
-106	KILLED_BY_RESOURCEMANAGER	Container was terminated by the resource manager.
-107	KILLED_AFTER_APP_COMPLETION	Container was terminated after the application finished.