Managing Data Operating System

Date of Publish: 2019-12-17

https://docs.hortonworks.com

Contents

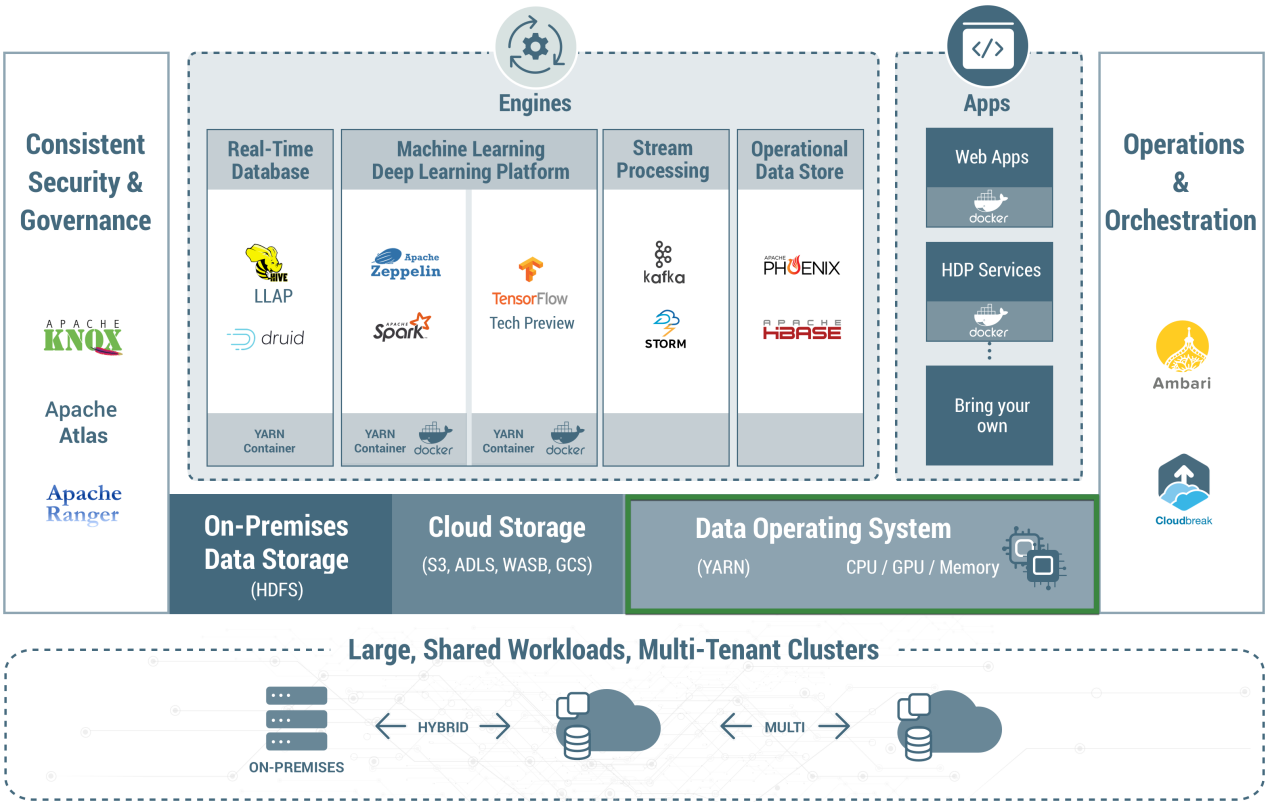
Introduction

Apache YARN is the processing layer for managing distributed applications that run on multiple machines in a network.

Understanding YARN architecture and features

YARN, the Hadoop operating system, enables you to manage resources and schedule jobs in Hadoop.

YARN allows you to use various data processing engines for batch, interactive, and real-time stream processing of data stored in HDFS (Hadoop Distributed File System). You can use different processing frameworks for different use-cases, for example, you can run Hive for SQL applications, Spark for in-memory applications, and Storm for streaming applications, all on the same Hadoop cluster.


Consistent Security & Governance
APACHE KNOX
Apache Atlas
Apache Ranger
Engines
Real-Time Database
LLAP
druid
YARN Container
Machine Learning Deep Learning Platform
Apache Zeppelin
Spark
TensorFlow
Tech Preview
YARN Container docker
YARN Container docker
Stream Processing
kafka
STORM
Operational Data Store
PHOENIX
HBASE
Apps
Web Apps
docker
HDP Services
docker
Bring your own
Operations & Orchestration
Ambari
Cloudbreak
On-Premises Data Storage
(HDFS)
Cloud Storage
(S3, ADLS, WASB, GCS)
Data Operating System
(YARN)
CPU / GPU / Memory
Large, Shared Workloads, Multi-Tenant Clusters
ON-PREMISES
HYBRID
MULTI

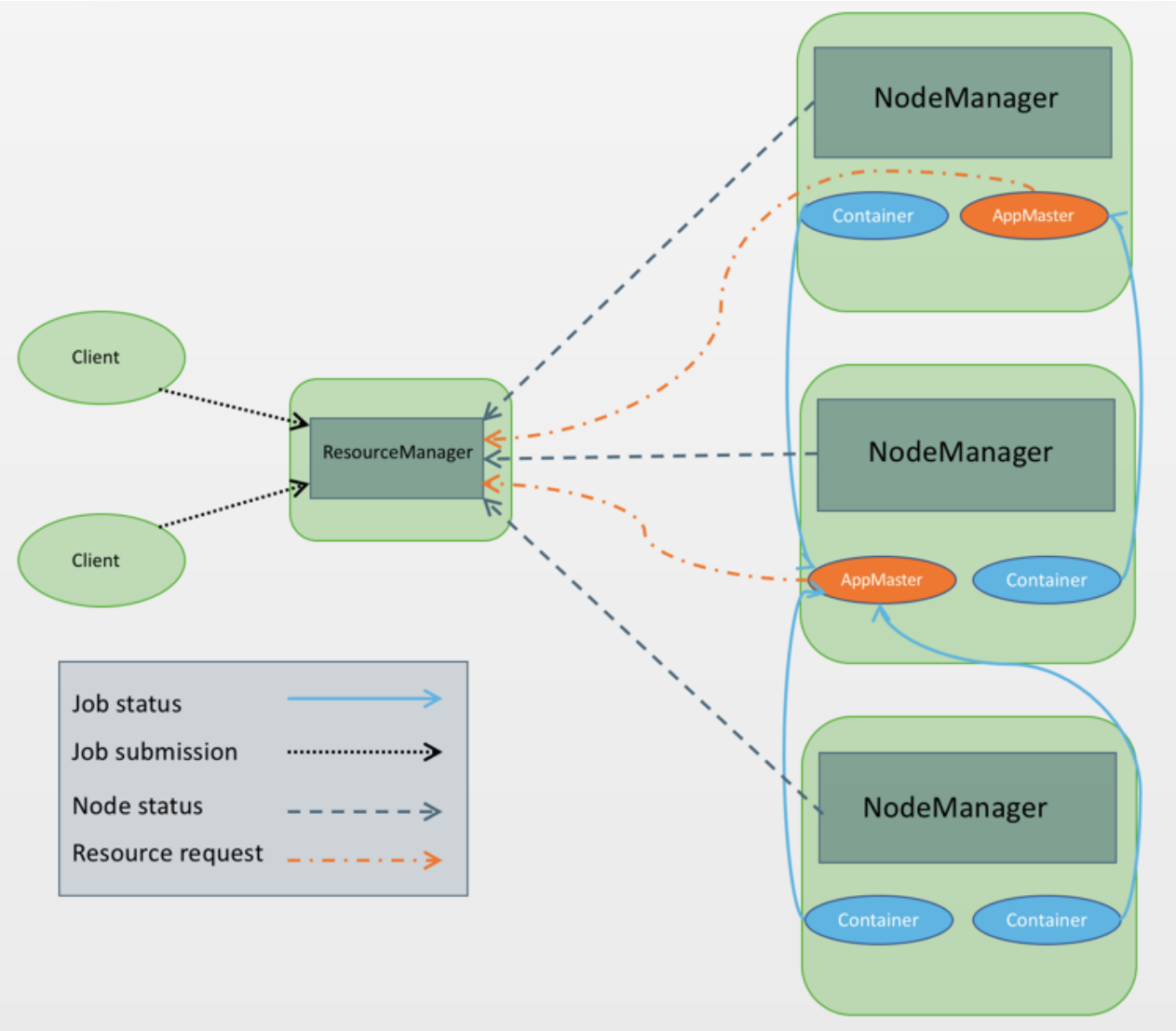
YARN extends the power of Hadoop to new technologies found within the data center so that you can take advantage of cost-effective linear-scale storage and processing. It provides independent software vendors and developers a consistent framework for writing data access applications that run in Hadoop.

Support for Docker containerization makes it easier for you to package and distribute applications, it allows you to focus on running and fine-tuning your applications, therefore reducing "time-to-deployment" and "time-to-insight".

YARN architecture and workflow

YARN has three main components:

- ResourceManager: Allocates cluster resources using a Scheduler and ApplicationManager.
- ApplicationMaster: Manages the life-cycle of a job by directing the NodeManager to create or destroy a container for a job. There is only one ApplicationMaster for a job.
- NodeManager: Manages jobs or workflow in a specific node by creating and destroying containers in a cluster node.


NodeManager
Container
AppMaster
Client
ResourceManager
NodeManager
Client
AppMaster
Container
Job status
Job submission
Node status
Resource request
NodeManager
Container
Container

YARN features

YARN provides the following features:

Multi-tenancy	You can use multiple open-source and proprietary data access engines for batch, interactive, and real-time access to the same dataset. Multi-tenant data processing improves an enterprise's return on its Hadoop investments.
Docker containerization	You can use docker containerization to run multiple versions of the same applications side-by-side.
Cluster utilization	You can dynamically allocate cluster resources to improve resource utilization.
Multiple resource types	You can use multiple resource types such as memory, CPU, and GPU.
Scalability	Significantly improved data center processing power. YARN's ResourceManager focuses exclusively on scheduling and keeps pace as clusters expand to thousands of nodes managing petabytes of data.
Compatibility	MapReduce applications developed for Hadoop 1 runs on YARN without any disruption to existing processes. YARN maintains API compatability with the previous stable release of Hadoop.

Related Information

Apache Hadoop YARN

Application Development

You can use the YARN REST APIs to manage applications. To collect generic and per-framework information about YARN applications, you can use the timeline server.

Using the YARN REST APIs to Manage Applications

You can use the YARN REST APIs to submit, monitor, and kill applications.

Important: In a non-secure cluster, you must append a request with ?user.name=<user>.

Example: Get application data

- Without ?user.name=<user>:

```
curl http://localhost:19888/jobhistory/job/job_1516861688424_0001
Access denied: User null does not have permission to view job
 job_1516861688424_0001
```

- With ?user.name=<user>:

```
curl http://localhost:19888/jobhistory/job/job_1516861688424_0001?user.name=hrt_1
{"job":{"submitTime":1516863297896,"startTime":1516863310110,"finishTime":1516863330610,
"id":"job_1516861688424_0001","name":"Sleepjob","queue":"default","user":"hrt_1",
"state":"SUCCEEDED","mapsTotal":1,"mapsCompleted":1,"reducesTotal":1,"reducesCompleted":1,
"uberized":false,"diagnostics":"","avgMapTime":10387,"avgReduceTime":536,"avgShuffleTime":4727,
"avgMergeTime":27,"failedReduceAttempts":0,"killedReduceAttempts":0,"successfulReduceAttempts":1,
"failedMapAttempts":0,"killedMapAttempts":0,"successfulMapAttempts":1,"acls":
[{"name":"mapreduce.job.acl-
view-job","value":" "},{"name":"mapreduce.job.acl-modify-job","value":" "}]}}
```

Get an Application ID

You can use the New Application API to get an application ID, which can then be used to submit an application. For example:

```
curl -v -X POST 'http://localhost:8088/ws/v1/cluster/apps/new-application'
```

The response returns the application ID, and also includes the maximum resource capabilities available on the cluster. For example:

```
    {
    application-id: application_1409421698529_0012",
    "maximum-resource-capability":{"memory":"8192","vCores":"32"}
    }
```

Set Up an Application .json File

Before you submit an application, you must set up a .json file with the parameters required by the application. This is analogous to creating your own ApplicationMaster. The application .json file contains all of the fields you are required to submit in order to launch the application.

The following is an example of an application .json file:

```
 {
    "application-id":"application_1404203615263_0001",
    "application-name":"test",
    "am-container-spec":
    {
       "local-resources":
       {
          "entry":
          [
             {
                "key":"AppMaster.jar",
                "value":
                {
                   "resource":"hdfs://hdfs-namenode:9000/user/testuser/
DistributedShell/demo-app/AppMaster.jar",
                   "type":"FILE",
                   "visibility":"APPLICATION",
                   "size": "43004",
                   "timestamp": "1405452071209"
                }
             }
          ]
       },
       "commands":
       {
          "command":"{{JAVA_HOME}}/bin/java -Xmx10m
 org.apache.hadoop.yarn.applications.distributedshell.ApplicationMaster --
container_memory 10 --container_vcores 1 --num_containers 1 --priority 0
 1><LOG_DIR>/AppMaster.stdout 2><LOG_DIR>/AppMaster.stderr"
       },
       "environment":
       {
          "entry":
          [
             {
                "key": "DISTRIBUTEDSHELLSCRIPTTIMESTAMP",
                "value": "1405459400754"
```

```
            },
            {
               "key": "CLASSPATH",
               "value": "{{CLASSPATH}}<CPS>./
*<CPS>{{HADOOP_CONF_DIR}}<CPS>{{HADOOP_COMMON_HOME}}/share/hadoop/
common/*<CPS>{{HADOOP_COMMON_HOME}}/share/hadoop/common/lib/
*<CPS>{{HADOOP_HDFS_HOME}}/share/hadoop/hdfs/*<CPS>{{HADOOP_HDFS_HOME}}/
share/hadoop/hdfs/lib/*<CPS>{{HADOOP_YARN_HOME}}/share/hadoop/yarn/
*<CPS>{{HADOOP_YARN_HOME}}/share/hadoop/yarn/lib/*<CPS>./log4j.properties"
            },
            {
               "key": "DISTRIBUTEDSHELLSCRIPTLEN",
               "value": "6"
            },
            {
               "key": "DISTRIBUTEDSHELLSCRIPTLOCATION",
               "value": "hdfs://hdfs-namenode:9000/user/testuser/demo-app/
shellCommands"
            }
         ]
      }
   },
   "unmanaged-AM":"false",
   "max-app-attempts":"2",
   "resource":
   {
      "memory":"1024",
      "vCores":"1"
   },
   "application-type":"YARN",
   "keep-containers-across-application-attempts":"false"
 }
```

Submit an Application

You can use the Submit Application API to submit applications. For example:

```
curl -v -X POST -d @example-submit-app.json -H "Content-type: application/
json"'http://localhost:8088/ws/v1/cluster/apps'
```

After you submit an application the response includes the following field:

```
HTTP/1.1 202 Accepted
```

The response also includes the Location field, which you can use to get the status of the application (app ID). The following is an example of a returned Location code:

```
Location: http://localhost:8088/ws/v1/cluster/apps/
application_1409421698529_0012
```

Monitor an Application

You can use the Application State API to query the application state. To return only the state of a running application, use the following command format:

```
curl 'http://localhost:8088/ws/v1/cluster/apps/
application_1409421698529_0012/state'
```

You can also use the value of the Location field (returned in the application submission response) to check the application status. For example:

```
curl -v 'http://localhost:8088/ws/v1/cluster/apps/
application_1409421698529_0012'
```

You can use the following command format to check the logs:

```
yarn logs -appOwner 'dr.who' -applicationId application_1409421698529_0012 |
 less
```

Kill an Application

You can also use the Application State API to kill an application by using a PUT operation to set the application state to KILLED. For example:

```
curl -v -X PUT -d '{"state": "KILLED"}''http://localhost:8088/ws/v1/cluster/
apps/application_1409421698529_0012'
```

Related Information

Apache YARN REST APIs

Collect Application Data with the Timeline Server

The Timeline Server enables you to collect generic and per-framework information about YARN applications.

The Timeline Server maintains historical state and provides metrics visibility for YARN applications, similar to the functionality the Job History Server provides for MapReduce.

The Timeline Server provides the following information:

- Generic Information about Completed Applications. Generic information includes application-level data such as queue name, user information, information about application attempts, a list of Containers that were run under each application attempt, and information about each Container. Generic data about completed applications can be accessed using the web UI or via REST APIs.
- Per-Framework Information for Running and Completed Applications. Per-framework information is specific to an application or framework. For example, the Hadoop MapReduce framework can include pieces of information such as the number of map tasks, reduce tasks, counters, etc. Application developers can publish this information to the Timeline Server via the TimelineClient (from within a client), the ApplicationMaster, or the application's Containers. This information can then be queried through REST APIs that enable rendering by application or framework-specific UIs.

The Timeline Server is a stand-alone server daemon that is deployed to a cluster node. It may or may not be co-located with the ResourceManager.

Timeline Service 2.0 Overview

YARN Timeline Service 2.0 is the next major version of timeline server after versions 1.0 and 1.5. Timeline Service 2.0 uses a distributed writer architecture and a back-end storage that are more scalable than the earlier versions.

YARN Timeline Service 2.0 addresses the following two major challenges associated with the earlier versions of Timeline Server:

Scalability	Timeline Server 1.0 is limited to a single instance of reader or writer and storage. Therefore, it does not scale well beyond small clusters. YARN Timeline Service 2.0 separates data writes from data reads. It uses distributed collectors, essentially one collector for each YARN application. The readers are separate instances that are dedicated to serving queries through REST APIs.

	YARN Timeline Service 2.0 uses Apache HBase as the primary backing storage, because Apache HBase scales well to a large size while maintaining good response times for reads and writes.
Usability	YARN Timeline Service 2.0 provides improved usability because it supports aggregation of information at the level of flows or logical group of YARN applications. It supports configuration information and metrics.

Related Information

The YARN Timeline Service v.2

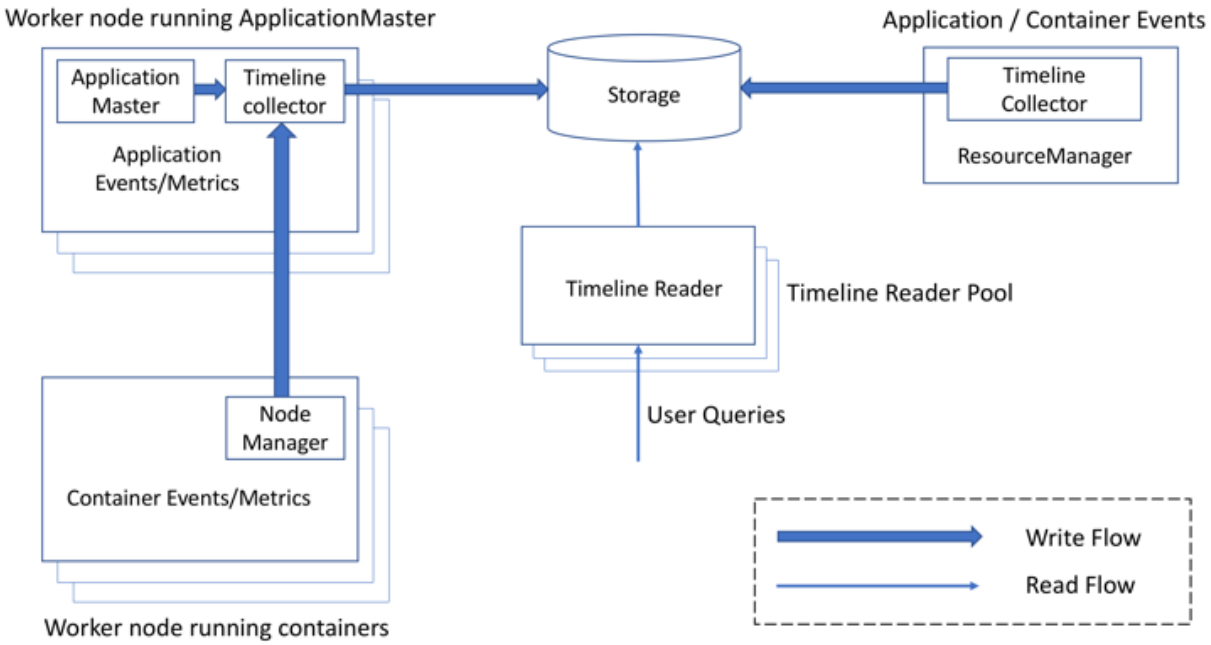
Architecture of Timeline Service 2.0

YARN Timeline Service 2.0 uses a set of collectors or writers to write data to the back-end storage. The collectors are present at the level of individual applications or the ResourceManager.

The application-level collectors are distributed and co-located with dedicated ApplicationManagers. These collectors collect data that belongs to the applications. For a given application, the ApplicationMaster can write data to the co-located timeline collector. In addition, NodeManagers of other nodes that are running the containers for the application can write data to the timeline collector on the node that is running the ApplicationMaster.

The collector associated with the ResourceManager collects generic life cycle events.

In addition to the collectors, there are separate daemons called timeline readers that are dedicated to serving queries through REST APIs.



Timeline Service 2.0 Installation

Timeline Service 2.0 is automatically installed as part of the YARN installation process when deploying an HDP cluster using Ambari.

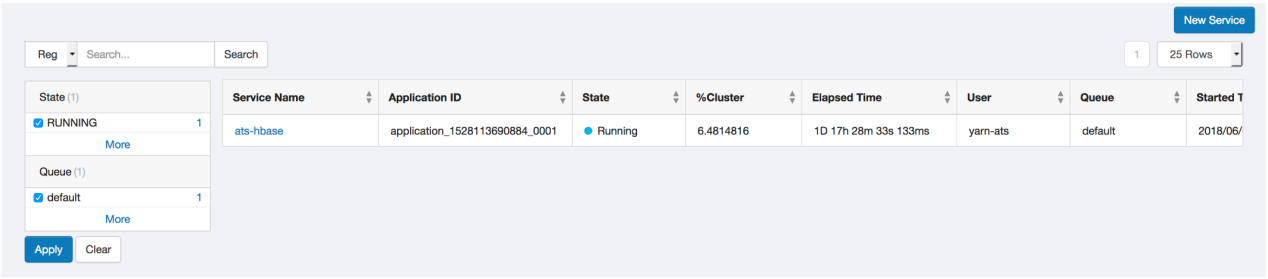
Timeline Service 2.0 uses HBase as the primary backing storage. Therefore, you can use either of the following approaches to configure HBase for Timeline Service 2.0:

- Enable the automatic installation of HBase along with Timeline Service 2.0 as part of the YARN installation process. *This is different from the HBase service that you install using Ambari.*
- Configure YARN to point to an external instance of HBase service installed using Ambari.

Installation Modes for HBase associated with Timeline Service 2.0

The HBase component available with Timeline Service 2.0 is automatically installed either in embedded mode or as a separate system service depending on the cluster capacity.

- Embedded mode: This mode of installation works on smaller clusters with less than 50 GB of total cluster capacity and less than 12 GB capacity of the NodeManager. This HBase cluster runs along with the Timeline Service V2.0 Reader component.
- System service mode: This mode of installation works on clusters where the capacity on each node is at least 10 GB and the total cluster capacity is at least 50 GB. In this mode, HBase runs as a separate service named ats-hbase on YARN. The system service mode is recommended for installation on large clusters for the purpose of scalability.



Configure External HBase for Timeline Service 2.0

Based on your requirements, you can configure YARN to point to an external instance of the HBase service instead of using the HBase instance that is available with Timeline Service 2.0.

Before you begin

You must have installed HBase on your Hortonworks Data Platform (HDP) cluster before configuring the instance for Timeline Service 2.0.

About this task

You can configure the HBase service for Timeline Service 2.0 on your HDP cluster either before or after installing YARN.

Procedure

1. Enable the use_external_hbase property under **Advanced yarn-hbase-env**.

Depending on your requirement, use either of the following options to access the use_external_hbase property using Ambari:

If...	Then...
YARN is already installed on the HDP cluster	In **Ambari Web**, browse to **Services** > **YARN** > **Configs**, then expand **Advanced yarn-hbase-env**.
You are using Ambari to install YARN on the HDP cluster	On the **Customize Services** screen of the **Add Service** wizard, browse to **YARN** > **Configs**, then expand **Advanced yarn-hbase-env**. **Note:** You must perform this step as part of the procedure to install YARN as a service using Ambari. For more information about using Ambari to add a service, see the *Managing and Monitoring Ambari* documentation.

2. Update the following properties under **Advanced yarn-hbase-env** to match the values of the corresponding properties in the external HBase instance:
   - hbase.zookeeper.quorum
   - hbase.zookeeper.property.clientPort
   - zookeeper.znode.parent
3. Save the configured property changes and restart all YARN services.
4. Copy the hbase-site.xml file to the timeline server configuration directory.

```
mv /usr/hdp/3.1.4.0-315/hadoop/conf/embedded-yarn-ats-hbase/hbase-
site.xml /usr/hdp/3.1.4.0-315/hadoop/conf/embedded-yarn-ats-hbase/hbase-
site.xml.bak
cp /etc/hbase/conf/hbase-site.xml /usr/hdp/3.1.4.0-315/hadoop/conf/
embedded-yarn-ats-hbase/
```

Note: <hdp-version> indicates the installed HDP version.

5. Log on to a cluster node and create the required HBase tables.

```
export HBASE_CLASSPATH_PREFIX={hdp-dir}/hadoop-yarn-client/
timelineservice/*; <hdp-dir>/current/hbase-client/bin/hbase
 org.apache.hadoop.yarn.server.timelineservice.storage.TimelineSchemaCreator
 -Dhbase.client.retries.number=35 -create -s
```

Note: <hdp-dir> indicates the directory where HDP is installed.

6. From the HBase shell, grant the yarn user with the required permissions to access the HBase tables created in Step 4.

```
grant 'yarn', 'yarn-ats' 'RWXCA'
```

Results

The Timeline Service V2.0 Reader component accesses and reads data from the new HBase instance.

Note: If you want to update YARN configuration to point to an embedded HBase instance, you must revert the properties use_external_hbase, hbase.zookeeper.quorum, hbase.zookeeper.property.clientPort, and zookeeper.znode.parent to their earlier values.

Enable System Service Mode

The HBase component associated with Timeline Service 2.0 is installed in the system service mode only when the HDP 3.0+ cluster meets the capacity requirements. The deployment of ats-hbase varies depending on whether you are deploying a new HDP 3.0+ cluster or whether you are upgrading the cluster from HDP 2.6.5 to HDP 3.0+.

On a new HDP 3.0+ cluster that meets the capacity requirements for HBase to run in the system service mode, ats-hbase is automatically submitted to the default queue without any ACLs configured. Therefore, you must configure a separate queue for ats-hbase, assign capacity and the highest priority to the queue, set ACLs, and disable preemption on the queue.

On an HDP cluster that is upgraded from 2.6.5 to 3.0+ and meets the capacity requirements for HBase to run in the system service mode, a separate zero-capacity queue is automatically created for the ats-hbase service. The queue is automatically configured with the required ACLs and priorities, and preemption is disabled. You must assign capacity to the particular queue and enable the system service mode so that ats-hbase is submitted to that queue.

Enable System Service Mode On a Newly Installed Cluster

On a new HDP 3.0+ cluster that meets the capacity requirements for HBase to run in the system service mode, you must configure a separate queue for running ats-hbase.

About this task

Perform the following *recommended* steps to run ats-hbase on your cluster after installing HDP 3.0+.

Procedure

1. Configure a separate queue at the root level, say yarn system, for the ats-hbase service instead of using the default queue.
2. Allocate required resources to the yarn-system queue for launching the ats-hbase service.

   The minimum amount of memory required for launching ats-hbase is 12 GB.

   To allocate resources, update the value of the yarn.scheduler.capacity.root.<queue_path>.capacity property in capacity-scheduler.xml. The following is an example:

   ```
   yarn.scheduler.capacity.root.yarn-system.capacity=10
   ```

3. Set ACLs on the yarn-system queue such that the yarn-ats user can manage the ats-hbase service.

   To set ACLs on the queue, update values of the yarn.scheduler.capacity.root.<queue-path>.acl_submit_applications and yarn.scheduler.capacity.root.<queue-path>.acl_administer_queue in capacity-scheduler.xml.

   ```
   yarn.scheduler.capacity.root.yarn-system.acl_submit_applications=yarn-ats,yarn
   yarn.scheduler.capacity.root.yarn-system.acl_administer_queue=yarn-ats,yarn
   ```

4. Disable preemption on the yarn-system queue.

   Set yarn.scheduler.capacity.root.<queue-path>.disable_preemption and yarn.scheduler.capacity.root.<queue-path>.intra-queue-preemption.disable_preemption in capacity-scheduler.xml to true.

   ```
   yarn.scheduler.capacity.root.yarn-system.disable_preemption=true
   yarn.scheduler.capacity.root.yarn-system.intra-queue-preemption.disable_preemption=true
   ```

5. Assign the highest priority to the yarn-system queue to prevent inadvertent deletion or preemption.

   Set the value of yarn.scheduler.capacity.root.<queue-path>.priority in capacity-scheduler.xml.

   ```
   yarn.scheduler.capacity.root.yarn-system.priority=<maximum-integer-value>
   ```

6. Restart YARN for all the configuration changes to take effect.
7. Move ats-hbase from the default queue to the yarn-system queue.

yarn application -changeQueue yarn-system -appId <app-id>

Here, <app-id> is the ID of the ats-hbase service.

Enable System Service Mode On an Upgraded Cluster

On a cluster that is upgraded from HDP 2.6.5 to HDP 3.0+ and meets the capacity requirements for HBase to run in the system service mode, you must assign capacity to the yarn-system queue and enable the system service mode.

Procedure

1. Allocate required resources to the yarn-system queue for launching the ats-hbase service.

   The minimum amount of memory required for launching ats-hbase is 12 GB.

   To allocate resources, update the value of the yarn.scheduler.capacity.root.<queue_path>.capacity property in capacity-scheduler.xml. The following is an example:

   ```
   yarn.scheduler.capacity.root.yarn-system.capacity=10
   ```

2. Enable the is_hbase_system_service_launch property in yarn-hbase.env.
3. Restart YARN.

Life cycle management of ats-hbase

Depending on requirements, the yarn-ats user can perform the following life cycle management operations on the ats-hbase service: start, stop, or destroy the service; and scale up or scale down HBase component instances for the service.

The yarn-ats user can perform the life cycle management operations by using either REST APIs or the command line interface.

Stopping the ats-hbase service

REST API:

Consider the following examples of using the PUT method for stopping the service:

Secure cluster:

Ensure that you run kinit with /etc/security/keytabs/yarn-ats.hbase-client.headless.keytab.

```
curl -k --negotiate -u: -H "Content-Type: application/json" -X PUT http://
<ResourceManagerHost>:<ResourceManagerPort>/app/v1/services/ats-hbase -d '{
  "state": "STOPPED"
}'
```

Non-secure cluster:

```
curl -k -u: -H "Content-Type: application/json" -X PUT http://
<ResourceManagerHost>:<ResourceManagerPort>/app/v1/services/ats-hbase?
user.name=yarn-ats -d '{
  "state": "STOPPED"
}'
```

Command

Use the yarn app -stop command to stop the service.

```
yarn app -stop ats-hbase
```

Starting the ats-hbase service

REST API:

Consider the following examples of using the PUT method for starting the stopped service:

Secure cluster:

Ensure that you run kinit with /etc/security/keytabs/yarn-ats.hbase-client.headless.keytab.

```
curl -k --negotiate -u: -H "Content-Type: application/json" -X PUT http://
<ResourceManagerHost>:<ResourceManagerPort>/app/v1/services/ats-hbase -d '{
  "state": "STARTED"
}'
```

Non-secure cluster:

```
curl -k -u: -H "Content-Type: application/json" -X PUT http://
<ResourceManagerHost>:<ResourceManagerPort>/app/v1/services/ats-hbase?
user.name=yarn-ats -d '{
  "state": "STARTED"
}'
```

Command

Use the yarn app -start command to start the stopped service.

```
yarn app -start ats-hbase
```

Scaling up or scaling down HBase component instances for the ats-hbase service

You can scale up or scale down the instances of components such as the HBase master or region servers for ats-hbase.

Note: The number of the HBase master component instances cannot be increased by more than two at a time.

REST API:

In the following examples, the number of region server instances is scaled to a value of 10:

Secure cluster:

Ensure that you run kinit with /etc/security/keytabs/yarn-ats.hbase-client.headless.keytab.

```
curl -k --negotiate -u: -H "Content-Type: application/json" -X PUT http://
<ResourceManagerHost>:<ResourceManagerPort>/app/v1/services/ats-hbase/
components/<component> -d '{
  "number_of_containers": "10"
}'
```

Non-secure cluster:

```
curl -k -u: -H "Content-Type: application/json" -X PUT http://
<ResourceManagerHost>:<ResourceManagerPort>/app/v1/services/ats-hbase/
components/<component>?user.name=yarn-ats -d '{
  "number_of_containers": "10"
}'
```

Command

Use the yarn app -flex command and specify the number of component instances to scale.

```
yarn app -flex ats-hbase regionserver 10
```

Destroying the ats-hbase service

REST API:

Consider the following examples of using the PUT method for destroying the service:

Secure cluster:

Ensure that you run kinit with /etc/security/keytabs/yarn-ats.hbase-client.headless.keytab.

```
curl -k --negotiate -u: -H "Content-Type: application/json" -X DELETE
 http://
<ResourceManagerHost>:<ResourceManagerPort>/app/v1/services/ats-hbase
```

Non-secure cluster:

```
curl -k -u: -H "Content-Type: application/json" -X DELETE http://
<ResourceManagerHost>:<ResourceManagerPort>/app/v1/services/ats-hbase?
user.name=yarn-ats
```

Command

Use the yarn app -destroy command to destroy the service.

```
yarn app -destroy ats-hbase
```

Note:

- The ats-hbase service can restart without any user intervention when a ResourceManager restarts or during a ResourceManager HA failover.
- When dependent services such as ZooKeeper or HDFS restart, or when the NameNode moves to a different host or to safe mode, the ats-hbase service continues to retain the earlier configuration, and therefore, becomes inaccessible. In such situations, Hortonworks recommends that you manually restart ats-hbase.

Related Tasks

Remove ats-hbase before switching between clusters

Remove ats-hbase before switching between clusters

Before you migrate between secure and non-secure clusters or HA and non-HA clusters, you must remove ats-hbase from the source cluster. Otherwise, Timeline Service 2.0 might stop working.

About this task

You must destroy the ats-hbase service, remove its configuration from HDFS, and remove its specification file from HDFS.

Note: The following procedure explains with examples the process of removing the ats-hbase service for the yarn-ats user. If required, you must change the user based on your deployment. Similarly, you must use principal and keytab names different from those given in the following examples based on the deployment of your secure cluster.

Procedure

1. Use the yarn app -destroy command to destroy the ats-hbase service.
   - On a non-secure cluster:

     ```
     su - yarn-ats
     yarn app -destroy ats-hbase
     ```

- On a secure cluster:

```
su - yarn-ats
kinit -kt /etc/security/keytabs/yarn-ats.hbase-client.headless.keytab
 yarn-ats@EXAMPLE.COM
yarn app -destroy ats-hbase
```

2. Remove the ats-hbase configuration from HDFS.

- On a non-secure cluster:

```
su - yarn-ats
hadoop fs -rm -R ./{stack_version}/*  #{stack_version} is 3.0.0.0-1557
```

- On a secure cluster:

```
su - yarn-ats
kinit -kt /etc/security/keytabs/yarn-ats.hbase-client.headless.keytab
 yarn-ats@EXAMPLE.COM
hadoop fs -rm -R ./{stack_version}/*  #{stack_version} is 3.0.0.0-1557
```

3. Delete the specification file for ats-hbase from HDFS.

Note: You must be logged in as the hdfs user to perform this step.

- On a non-secure cluster:

```
su - hdfs
hadoop fs -rm -R /services/sync/yarn-ats/hbase.yarnfile
```

- On a secure cluster:

```
su - hdfs
kinit -kt /etc/security/keytabs/hdfs.headless.keytab hdfs@EXAMPLE.COM
hadoop fs -rm -R /services/sync/yarn-ats/hbase.yarnfile
```

Related reference

Life cycle management of ats-hbase

Publish Application-Specific Data

The TimelineV2Client API helps you publish application-specific data to Timeline Service 2.0. For your applications, you must define a TimelineEntity object and publish the events to Timeline Service 2.0.

About this task

The YARN applications that write to Timeline Service 2.0 must be running on the cluster.

Procedure

1. Create and start the timeline client.

   You must specify the Application ID to write to Timeline Service 2.0.

   Following is an example for defining a timeline client:

```
// Create and start the Timeline client v.2
TimelineV2Client timelineClient =
    TimelineV2Client.createTimelineClient(appId);
timelineClient.init(conf);
timelineClient.start();

try {
```

```
  TimelineEntity myEntity = new TimelineEntity();
  myEntity.setType("MY_APPLICATION");
  myEntity.setId("MyApp1");
  // Compose other entity info

  // Blocking write
  timelineClient.putEntities(myEntity);

  TimelineEntity myEntity2 = new TimelineEntity();
  // Compose other info

  // Non-blocking write
  timelineClient.putEntitiesAsync(myEntity2);

} catch (IOException | YarnException e) {
  // Handle the exception
} finally {
  // Stop the Timeline client
  timelineClient.stop();
}
```

2. Set the timeline collector information.

   You can either use amRMClient and register the timeline client or set the information explicitly in the timeline client.

```
amRMClient.registerTimelineV2Client(timelineClient);
```

```
timelineClient.setTimelineCollectorInfo(response.getCollectorInfo());
```

Application Information for Timeline Service 2.0

You can publish different types of information about your application; such entities, events, and metrics, to Timeline Service 2.0.

Timeline Entity Objects

You can provide application information through the following fields in timeline entity objects:

- events: A set of timeline events, ordered by the timestamp of the events in descending order. Each event contains an ID and a map to store related information and is associated with one timestamp.
- configs: A map from a configuration name to a configuration value and representing all the configurations associated with the entity. Users can post the entire configuration or a part of it in this field. The field is supported for application and generic entities.
- metrics: A set of metrics related to a particular entity. There are two types of metrics: single value metric and time series metric. Each metric item contains the metric name, value, and the type of aggregation operation to be performed in the metric. The field is supported for flow run, application and generic entities.
- info: A map from an info key name to an (info value object that holds related information for the entity. The field is supported for application and generic entities.
- isrelatedtoEntities and relatestoEntities: Each entity contains relatestoEntities and isrelatedtoEntities fields to represent relationships with other entities. Both the fields are represented by a map from a relationship name string to a timeline entity.

Timeline Metrics

When posting timeline metrics, you can select how each metric is aggregated. The term *aggregate* means applying a TimelineMetricOperation for a set of entities. Timeline Service 2.0 enables aggregating metrics from different timeline entities within one YARN application. The following are the types of operations supported in TimelineMetricOperation:

- MAX: Retrieves the maximum value among all TimelineMetric objects.

- SUM: Retrieves the sum of all TimelineMetric objects.

The default value of NOP means that no real-time aggregation operation is performed.

Flow Context

Application frameworks must set the flow context whenever possible in order to take advantage of the flow support that Timeline Service 2.0 provides. The flow context consists of the following:

- Flow Name: A string that identifies the high-level flow; for example, distributed grep or any identifiable name that can uniquely represent the application. The default value is the name of the application or the ID of the application if the name is not set.
- Flow Run ID: A monotonically increasing sequence of numbers that distinguish different runs of the same flow. The default value is the application time in UNIX time (milliseconds).
- Flow Version: This optional value is a string identifier that denotes a version of the flow. Flow version can be used to identify changes in the flows, such as code changes or script changes. The default version number is 1.

You can provide the flow context through YARN application tags, as shown in the following example:

```
ApplicationSubmissionContext appContext =
 app.getApplicationSubmissionContext();

// set the flow context as YARN application tags
Set<String> tags = new HashSet<>();
tags.add(TimelineUtils.generateFlowNameTag("distributed grep"));
tags.add(Timelineutils.generateFlowVersionTag
("3df8b0d6100530080d2e0decf9e528e57c42a90a"));
tags.add(TimelineUtils.generateFlowRunIdTag(System.currentTimeMillis()));

appContext.setApplicationTags(tags);
```

Note: The ResourceManager converts YARN application tags to lower case values before storing them. Therefore, you must convert flow names and versions to lower case before using them in REST API queries.

REST APIs for Querying Timeline Service 2.0

You must use REST APIs for querying Timeline Service 2.0 and retrieving information about applications. The API is implemented at the path /ws/v2/timeline/ on the web service for the timeline service.

The following table lists the different types of queries supported on the REST API for Timeline Service 2.0:

If you want to query...	Use this HTTP Request Syntax...
The root path of the API	`GET /ws/v2/timeline/`

Active flows	`GET /ws/v2/timeline/clusters/ {cluster name}/flows/`
	OR
	`GET /ws/v2/timeline/flows/`

Active flow runs	`GET /ws/v2/timeline/clusters/ {cluster name}/users/{user name}/ flows/{flow name}/runs/`

If you want to query...	Use this HTTP Request Syntax...
	OR `GET /ws/v2/timeline/users/{user name}/flows/{flow name}/runs/`

A specific flow run	`GET /ws/v2/timeline/clusters/{cluster name}/users/{user name}/flows/{flow name}/runs/{run id}`
	OR
	`GET /ws/v2/timeline/users/{user name}/flows/{flow name}/runs/{run id}`

YARN applications that belong to a flow	`GET /ws/v2/timeline/clusters/{cluster name}/users/{user name}/flows/{flow name}/apps`
	OR
	`GET /ws/v2/timeline/users/{user name}/flows/{flow name}/apps`

YARN applications that belong to a flow run	`GET /ws/v2/timeline/clusters/{cluster name}/users/{user name}/flows/{flow name}/runs/{run id}/apps`
	OR
	`GET /ws/v2/timeline/users/{user name}/flows/{flow name}/runs/{run id}/apps/`

A specific YARN application	`GET /ws/v2/timeline/clusters/{cluster name}/apps/{app id}`
	OR
	`GET /ws/v2/timeline/apps/{app id}`

Generic timeline entities within the scope of an application	`GET /ws/v2/timeline/clusters/{cluster name}/apps/{app id}/entities/{entity type}`

If you want to query...	Use this HTTP Request Syntax...
	OR `GET /ws/v2/timeline/apps/{app id}/ entities/{entity type}`

Generic timeline entities per user	`GET /ws/v2/timeline/clusters/ {cluster name}/users/{userid}/ entities/{entitytype}`
	OR
	`GET /ws/v2/timeline/users/{userid}/ entities/{entitytype}`

A specific timeline entity	`GET /ws/v2/timeline/clusters/ {cluster name}/apps/{app id}/ entities/{entity type}/{entity id}`
	OR
	`GET /ws/v2/timeline/apps/{app id}/ entities/{entity type}/{entity id}`

A generic timeline entity per user	`GET /ws/v2/timeline/clusters/ {cluster name}/users/{userid}/ entities/{entitytype}/{entityid}`
	OR
	`GET /ws/v2/timeline/users/{userid}/ entities/{entitytype}/{entityid}`

A set of available entity types for an Application ID	`GET /ws/v2/timeline/apps/{appid}/ entity-types`
	OR
	`GET /ws/v2/timeline/clusters/ {clusterid}/apps/{appid}/entity-types`

Note: For more information about these API query types, see https://hadoop.apache.org/docs/r3.1.0/hadoop-yarn/hadoop-yarn-site/TimelineServiceV2.html#Timeline_Service_v.2_REST_API.

Timeline Server 1.5 Overview

Timeline Server 1.5 includes updates that improve performance and scalability.

In HDP 2.2.9, Timeline Server 1.5 includes changes and fixes that improve scalability. Version 1.5 incorporates changes and improvements to the whole stack; it introduces new client APIs, a new storage architecture (the EntityGroupFSTimelineStore), and supports a “plugin” mechanism to work with YARN applications. The most significant change, on the YARN side, is the EntityGroupFSTimelineStore. This storage system has a reader part and

a writer part. To work with this new version 1.5 architecture, applications can use the newly introduced client APIs to write to version 1.5, and provide a plugin that assists version 1.5 on the reader part. Tez currently provides this support and can work with version 1.5 natively.

- RollingLevelDbTimelineStore improves the performance for ATS (Application Timeline Server) operations that need to purge data regarding obsolete entries. Instead of deleting one row at a time, the store is organized as a list of databases so that an entire database can be dropped when all the entries contained within it are obsolete.
- EntityGroupFSTimelineStore enables a completely different reader-writer architecture for the ATS. It has two parts, a writer part and a reader part. The writer part involves configuring Tez with the ATSV15Plugin. The reader part involves specifying the configurations below. The writer in this instance writes summary information to the ATS and the bulk of the data to the filesystem location specified by the configuration path. Before this store can be used the directory has to be created with the correct permissions.
- Changed properties:
  - Property: yarn.timeline-service.store-class
    - Old value: org.apache.hadoop.yarn.server.timeline.LeveldbTimelineStore
    - New value: org.apache.hadoop.yarn.server.timeline.EntityGroupFSTimelineStore
  - Property: yarn.timeline-service.leveldb-timeline-store.path
    - Old value: ${yarn.log.dir}/timeline
    - New value: <Data disk>/ats/leveldb/
- New properties:
  - tez.history.logging.service.class
  - yarn.timeline-service.entity-group-fs-store.active-dir
  - yarn.timeline-service.entity-group-fs-store.cleaner-interval-seconds
  - yarn.timeline-service.entity-group-fs-store.done-dir
  - yarn.timeline-service.entity-group-fs-store.group-id-plugin-classes
  - yarn.timeline-service.entity-group-fs-store.retain-seconds
  - yarn.timeline-service.entity-group-fs-store.scan-interval-seconds
  - yarn.timeline-service.entity-group-fs-store.summary-store
  - yarn.timeline-service.version

Upgrade Timeline Server 1.0 to 1.5

You must configure various properties associates with the Timeline Server to upgrade from version 1.0 to version 1.5.

About this task

Important: Upgrading from ATS v1 to v1.5 may cause data stored in ATS v1.0 storage to be inaccessible. The upgrade process is not rolling: applications may lose previous data stored in ATS v1.0 during upgrade.

Before you begin

Before launching ATS v1.5, ensure your system is compliant with the following:

- yarn.timeline-service.entity-group-fs-store.active-dir and yarn.timeline-service.entity-group-fs-store.done-dir must exist on the cluster on HDFS. Active-dir should have permission 01777, owned by YARN, group admin-group. Done-dir should have permission 0700, owned by yarn, group admin-group.
- You must manually create yarn.timeline-service.entity-group-fs-store.active-dir in HDFS before you start RM and ATS.
- The Tez cache plugin class must be in the timeline server's classpath.
- yarn.timeline-service.leveldb-timeline-store.path must be a path on the local filesystem and must be created with permissions 755. The owner must be the same user that timeline server is running as.

Procedure

1. To upgrade to ATS 1.5, configure the following:

```
 <property>
 <name>yarn.timeline-service.version</name>
 <value>1.5</value>
<description>Timeline service version we’re currently using.</description>
  </property>

  <property>
 <name>yarn.timeline-service.store-class</name>
 <value>org.apache.hadoop.yarn.server.timeline.EntityGroupFSTimelineStore</
value>
<description>Main storage class for YARN timeline server. </description>
  </property>

  <property>
 <name>yarn.timeline-service.entity-group-fs-store.active-dir</name>
 <value>/ats/active/</value>
<description>DFS path to store active application’s timeline data</
description>
  </property>

  <property>
 <name>yarn.timeline-service.entity-group-fs-store.done-dir</name>
 <value>/ats/done/</value>
<description>DFS path to store done application’s timeline data</
description>
  </property>
```

2. The following configuration exists in ATS v1.0. Ensure it is pointing to the correct directory:

```
  <property>
 <name>yarn.timeline-service.leveldb-timeline-store.path</name>
 <value><Data disk>/ats/leveldb/</value>
<description>Local FS path to store the leveldbs that hold the application
 summary data</description>
  </property>
```

3. If you have Tez enabled, the tez-client must be installed on the ATS server. You must also perform this additional step:

```
  <property>
    <name>yarn.timeline-service.entity-group-fs-store.group-id-plugin-
classes</name>

 <value>org.apache.tez.dag.history.logging.ats.TimelineCachePluginImpl</
value>
    <description>Plugins that can translate a timeline entity read request
 into a list of timeline cache ids, separated by commas. </description>
  </property>
```

4. Configure the yarn.timeline-service.entity-group-fs-store.summary-store property:

```
  <property>
    <name>yarn.timeline-service.entity-group-fs-store.summary-store</name>

 <value>org.apache.hadoop.yarn.server.timeline.RollingLevelDBTimelineStore
</value>
    <description>DFS path to store active application’s timeline
 dataSummary storage for ATS v1.5.</description>
  </property>
```

Configure the Timeline Server

You must specify the host name of the Timeline Server web application to enable the Timeline Server. In addition, you can configure certain optional parameters.

Procedure

- Required Properties – Only one property must be specified in the etc/hadoop/conf/yarn-site.xml file in order to enable the Timeline Server: the host name of the Timeline Server web application (yarn.timeline-service.hostname).

```
<property>
 <description>The hostname of the timeline server web application.</
description>
 <name>yarn.timeline-service.hostname</name>
 <value>0.0.0.0</value>
</property>
```

Important: If the Timeline Server is running on the same host as the NameNode and is under heavy load, it may be shut down by the system OOM (Out of Memory) killer. Therefore you should consider deploying the Timeline Server on a separate host if you anticipate read-heavy loads.

- Advanced Properties – In addition to the host name, administrators can also configure the ports of the RPC and the web interfaces, as well as the number of RPC handler threads.
  - yarn.timeline-service.address

    The default address for the Timeline Server to start the RPC server.

    Example:

```
<property>
 <description>This is default address for the timeline server to start
 the RPC server.</description>
 <name>yarn.timeline-service.address</name>
 <value>${yarn.timeline-service.hostname}:10200</value>
</property>
```

  - yarn.timeline-service.webapp.address

    The HTTP address of the Timeline Server web application.

    Example:

```
<property>
 <description>The http address of the timeline server web application.</
description>
 <name>yarn.timeline-service.webapp.address</name>
 <value>${yarn.timeline-service.hostname}:8188</value>
</property>
```

  - yarn.timeline-service.webapp.https.address

    The HTTPS address of the Timeline Server web application.

    Example:

```
<property>
 <description>The https adddress of the timeline server web
 application.</description>
 <name>yarn.timeline-service.webapp.https.address</name>
 <value>${yarn.timeline-service.hostname}:8190</value>
</property>
```

  - yarn.timeline-service.handler-thread-count

The handler thread count to serve the client RPC requests.

Example:

```
<property>
 <description>Handler thread count to serve the client RPC requests.</
description>
 <name>yarn.timeline-service.handler-thread-count</name>
 <value>10</value>
</property>
```

Enable Generic Data Collection

You must enable the Generic History Service (GHS) for collecting historical data of YARN applications.

Procedure

- yarn.resourcemanager.system-metrics-publisher.enabled – This property indicates to the ResourceManager, as well as to clients, whether or not the Generic History Service (GHS) is enabled. If the GHS is enabled, the ResourceManager begins recording historical data that the GHS can consume, and clients can redirect to the GHS when applications finish running.

  Example:

```
<property>
 <description>Enable or disable the GHS</description>
 <name>yarn.resourcemanager.system-metrics-publisher.enabled</name>
 <value>true</value>
</property>
```

Configure Per-framework Data Collection

You must enable the Timeline Server to configure per-framework data collection.

Procedure

- yarn.timeline-service.enabled – Indicates to clients whether or not the Timeline Server is enabled. If it is enabled, the TimelineClient library used by end-users will post entities and events to the Timeline Server.

  Example:

```
<property>
 <description>Enable or disable the Timeline Server.</description>
 <name>yarn.timeline-service.enabled</name>
 <value>true</value>
</property>
```

Configure the Timeline Server Store

You must configure the Timeline Server store to which the YARN ResourceManager publishes generic application-level data.

Procedure

- yarn.timeline-service.store-class – The class name for the Timeline store.

  Example:

```
<property>
 <description>Store class name for timeline store</description>
 <name>yarn.timeline-service.store-class</name>
 <value>org.apache.hadoop.yarn.server.timeline.LeveldbTimelineStore</
value>
```

```
</property>
```

- yarn.timeline-service.leveldb-timeline-store.path – The store file path and name for the Timeline Server LevelDB store (if the LevelDB store is used).

  Example:

```
<property>
 <description>Store file name for leveldb timeline store</description>
 <name>yarn.timeline-service.leveldb-timeline-store.path</name>
 <value>${yarn.log.dir}/timeline</value>
</property>
```

- yarn.timeline-service.ttl-enable – Enable age-off of timeline store data.

  Example:

```
<property>
 <description>Enable age off of timeline store data.</description>
 <name>yarn.timeline-service.ttl-enable</name>
 <value>true</value>
</property>
```

- yarn.timeline-service.ttl-ms – The Time-to-live for timeline store data (in milliseconds).

  Example:

```
<property>
 <description>Time to live for timeline store data in milliseconds.</
description>
 <name>yarn.timeline-service.ttl-ms</name>
 <value>604800000</value>
</property>
```

Configure Timeline Server Security

You can configure Kerberos authentication for the Timeline Server. In addition, you can configure ACLs and SSL for the Timeline Server.

Procedure

- Configure Kerberos Authentication

  To configure Kerberos Authentication for the Timeline Server, add the following properties to the yarn-site.xml file.

```
<property>
 <name>yarn.timeline-service.http-authentication.type</name>
 <value>kerberos</value>
</property>

<property>
 <name>yarn.timeline-service.http-authentication.kerberos.principal</name>
 <value>HTTP/localhost@EXAMPLE.COM</value>
</property>

<property>
 <name>yarn.timeline-service.http-authentication.kerberos.keytab</name>
 <value>/etc/krb5.keytab</value>
</property>
```

- Configure Timeline Server Authorization (ACLs)

Timeline Server ACLs are configured in the same way as other YARN ACLs. To configure Timeline Server authorization with ACLs, add the following properties to the yarn-site.xml file.

```
<property>
 <name>yarn.acl.enable</name>
 <value>true</value>
</property>

<property>
 <name>yarn.admin.acl</name>
 <value> </value>
</property>
```

- Configure Timeline Server SSL

  Timeline Server SSL is configured in the same way as other Hadoop components. To configure Timeline Server SSL, add the following properties to the core-site.xml file.

```
<property>
 <name>hadoop.ssl.require.client.cert</name>
 <value>false</value>
</property>

<property>
 <name>hadoop.ssl.hostname.verifier</name>
 <value>DEFAULT</value>
</property>

<property>
 <name>hadoop.ssl.keystores.factory.class</name>
 <value>org.apache.hadoop.security.ssl.FileBasedKeyStoresFactory</value>
</property>

<property>
 <name>hadoop.ssl.server.conf</name>
 <value>ssl-server.xml</value>
</property>

<property>
 <name>hadoop.ssl.client.conf</name>
 <value>ssl-client.xml</value>
</property>
```

Run the Timeline Server

You can run the yarn timelineserver command and start the Timeline Server or start the server as a daemon.

Procedure

- To start the Timeline Server, run the following command:

```
yarn timelineserver
```

- To start the Timeline Server as a daemon, run the following command:

```
sbin/yarn-daemon.sh start timelineserver
```

Access Generic Data from the Command Line

You can use various commands to access generic application history data from the command line.

Procedure

- The following commands can be used to access generic data from the command line and to obtain corresponding information about running applications.

```
yarn application -status <Application ID>
yarn applicationattempt -list <Application ID>
yarn applicationattempt -status <Application Attempt ID>
yarn container -list <Application Attempt ID>
yarn container -status <Container ID>
```

Publish Per-framework Data in Applications

Developers can define the information they would like to record for their applications by composing TimelineEntity and TimelineEvent objects, and then putting the entities and events to the Timeline server using TimelineClient.

Procedure

- Create and start the timeline client.

  Consider the following example of defining the timeline client.

```
// Create and start the Timeline client
TimelineClient client = TimelineClient.createTimelineClient();
client.init(conf);
client.start();

TimelineEntity entity = null;
// Compose the entity
try {
TimelinePutResponse response = client.putEntities(entity);
} catch (IOException e) {
// Handle the exception
} catch (YarnException e) {
// Handle the exception
}

// Stop the Timeline client
client.stop();
```

Application Management

You can use YARN Services API to manage long-running YARN applications. You can use the YARN CLI to view the logs for running applications, In addition, you can use YARN distributed cache to deploy multiple versions of Mapreduce.

Manage Long-running YARN Applications

You can use the YARN Services API to manage long-running YARN applications.

About this task

Previously in HDP 2.x, deploying a new service on YARN involved writing a native YARN application or leveraging an application framework such as Apache Slider. From HDP 3.0 onwards, Apache Slider has been removed from the stack. You can still write a native YARN application, but it is recommended that you use the new YARN Services API, which greatly simplifies the deployment and management of YARN services.

Procedure

1. Dockerize your application.
2. Use the YARN Services API to run a POST operation on your application, specifying a long or unlimited lifetime in the POST attributes.
3. Use the YARN Services API to manage your application.
   - Increase or decrease the number of application instances.
   - Perform other application life cycle tasks.

Related reference

Using the YARN Services API

Related Information

Get Started with Docker

Dockerize an Application

Using the YARN Services API

The YARN Services API helps in creating and managing the life cycle of YARN services.

<p>Previously, deploying a new service on YARN was not a simple experience. The APIs of existing frameworks were either too low level (native YARN), required writing new code (for frameworks with programmatic APIs), or required writing a complex specification (for declarative frameworks). Apache Slider was developed to run services such as HBase, Storm, Accumulo, and Solr on YARN by exposing higher-level APIs that supported running these services on YARN without modification.

The new YARN Services API greatly simplifies the deployment and management of YARN services.

Related Concepts

How YARN mounts an external volume using CSI

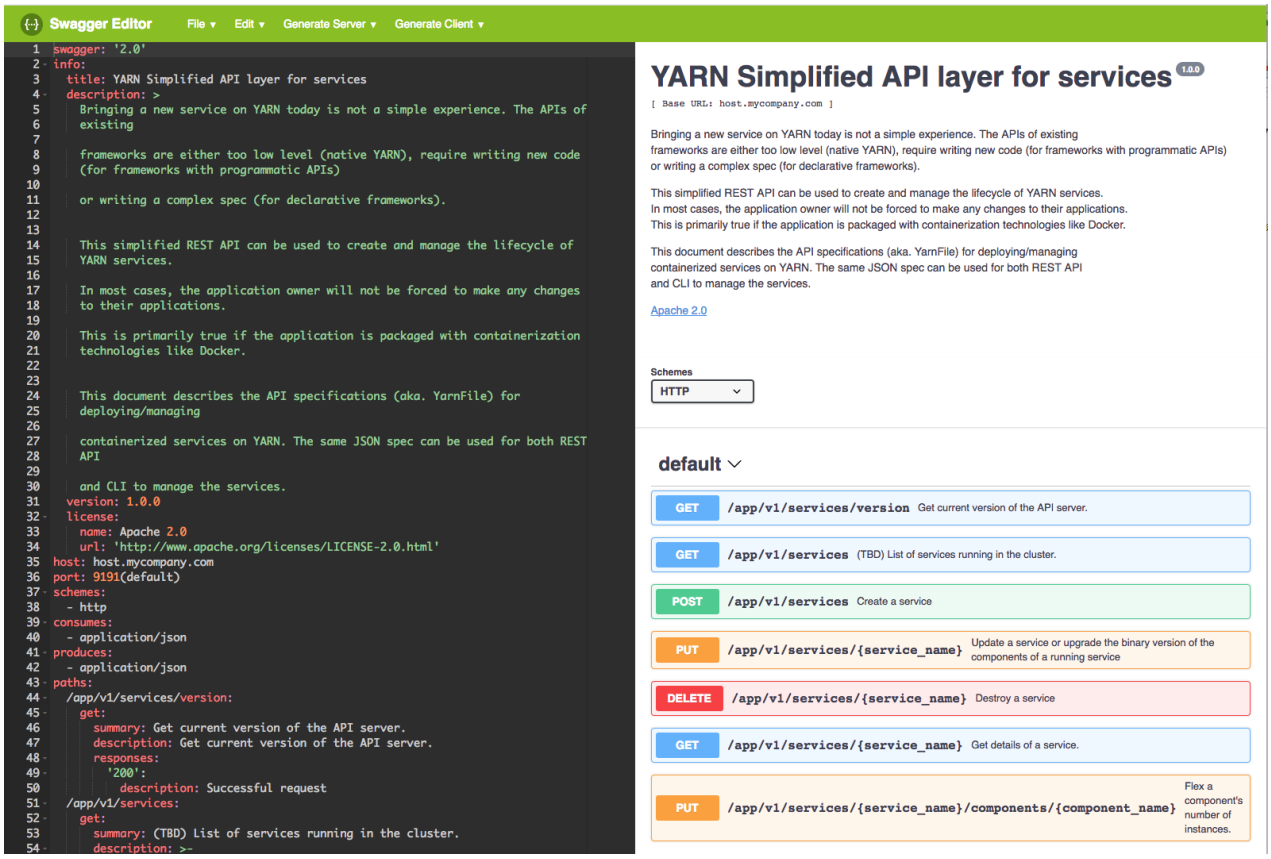
Related Tasks

Manage Long-running YARN Applications

YARN Services API Swagger Specification

Use the Swagger Editor to view the YARN Services API.

You can use the Swagger Editor to view the YARN Services API Swagger Specification.


Swagger Editor
File
Edit
Generate Server
Generate Client
swagger: '2.0'
info:
title: YARN Simplified API layer for services
description: >
Bringing a new service on YARN today is not a simple experience. The APIs of existing
frameworks are either too low level (native YARN), require writing new code
(for frameworks with programmatic APIs)
or writing a complex spec (for declarative frameworks).
This simplified REST API can be used to create and manage the lifecycle of
YARN services.
In most cases, the application owner will not be forced to make any changes
to their applications.
This is primarily true if the application is packaged with containerization
technologies like Docker.
This document describes the API specifications (aka. YarnFile) for
deploying/managing
containerized services on YARN. The same JSON spec can be used for both REST
API
and CLI to manage the services.
version: 1.0.0
license:
name: Apache 2.0
url: 'http://www.apache.org/licenses/LICENSE-2.0.html'
host: host.mycompany.com
port: 9191(default)
schemes:
- http
consumes:
- application/json
produces:
- application/json
paths:
/app/v1/services/version:
get:
summary: Get current version of the API server.
description: Get current version of the API server.
responses:
'200':
description: Successful request
/app/v1/services:
get:
summary: (TBD) List of services running in the cluster.
description: >-
YARN Simplified API layer for services
1.0.0
[Base URL: host.mycompany.com]
Bringing a new service on YARN today is not a simple experience. The APIs of existing
frameworks are either too low level (native YARN), require writing new code (for frameworks with programmatic APIs)
or writing a complex spec (for declarative frameworks).
This simplified REST API can be used to create and manage the lifecycle of YARN services.
In most cases, the application owner will not be forced to make any changes to their applications.
This is primarily true if the application is packaged with containerization technologies like Docker.
This document describes the API specifications (aka. YarnFile) for deploying/managing
containerized services on YARN. The same JSON spec can be used for both REST API
and CLI to manage the services.
Apache 2.0
Schemes
HTTP
default
GET /app/v1/services/version Get current version of the API server.
GET /app/v1/services (TBD) List of services running in the cluster.
POST /app/v1/services Create a service
PUT /app/v1/services/{service_name} Update a service or upgrade the binary version of the components of a running service
DELETE /app/v1/services/{service_name} Destroy a service
GET /app/v1/services/{service_name} Get details of a service.
PUT /app/v1/services/{service_name}/components/{component_name} Flex a component's number of instances.

Related Information

Swagger Editor

YARN Services API Swagger Specification

Deploying and Managing Services and Microservices on YARN

Using the YARN Services API, you can run simple and complex template-based apps on containers such as docker containers.

Without having the need to write new code or modify your apps, you can create and manage the life cycle of these YARN services.

```
{
  "name": "sleeper-service",
  "version": "1.0.0",
  "components" : [
    {
      "name": "sleeper",
      "number_of_containers": 2,
      "launch_command": "sleep 900000",
      "resource": {
      "cpus": 1,
      "memory": "256"
     }
    }
   ]
  }
```

Each service file contains, at a minimum, a name, version, and list of components. Each component of a service has a name, a number of containers to be launched (also referred to as component instances), a launch command, and an amount of resources to be requested for each container.

Components optionally also include an artifact specification. The artifact can be the default type (with no artifact specified, like the sleeper-service example above) or can have other artifact types such as DOCKER, TARBALL, or SERVICE.

An example YARN service file that specifies Docker containers running the centos/httpd-24-centos7:latest Docker image appears as follows:

```
{
  "name": "simple-httpd-service",
  "version": "1.0.0",
  "lifetime": "3600",
  "components": [
    {
      "name": "httpd",
      "number_of_containers": 2,
      "launch_command": "/usr/bin/run-httpd",
      "artifact": {
        "id": "centos/httpd-24-centos7:latest",
        "type": "DOCKER"
      },
      "resource": {
        "cpus": 1,
        "memory": "1024"
      },
      "configuration": {
        "files": [
          {
```

```
            "type": "TEMPLATE",
            "dest_file": "/var/www/html/index.html",
            "properties": {
              "content": "<html><header><title>Title</title></
header><body>Hello from ${COMPONENT_INSTANCE_NAME}!</body></html>"
            }
          }
        ]
      }
    }
  ]
}
```

In addition to illustrating the DOCKER artifact type, this service file introduces two additional features, a lifetime that indicates the number of seconds after which a service will be stopped and a configuration section. The example shows one type of configuration file that can be created for the service (other supported file types include STATIC, ARCHIVE, HADOOP_XML, PROPERTIES, JSON, and YAML). The configuration specification also may include an env section for specifying environment variables for the container and a properties section that is typically used to pass configuration properties to the YARN service Application Master (AM).

Launch the Service YARN file

Launching a service saves the service file to HDFS and starts the service.

Run the following command to launch the sleeper service example. This sleeper example is provided with YARN, so it can be referred to by the name *;sleeper;* or the path to the JSON file can be specified:

```
yarn app -launch sleeper-service <sleeper OR /path/to/sleeper.json>
```

This command saves and starts the sleeper service with two instances of the sleeper component. The service could also be launched by making calls to the REST API instead of using the command line. The service can be stopped and destroyed as follows. The stop command stops the service from running, but leaves the service JSON file stored in HDFS so that the service could be started again using a start command. The destroy command stops the service if it is running and removes the service information entirely.

```
yarn app -stop sleeper-service
```

```
yarn app -destroy sleeper-service
```

Save the YARN file as Service App

You can save a service YARN file initially without starting the service and later refer to this service YARN file while launching other services.

Run the following command to save the simple-httpd-service YARN file:

```
yarn app -save simple-httpd-service /path/to/simple-httpd-service.json
```

Saving or launching the service from a YARN file stores a modified version of the YARN file in HDFS. This service specification can then be referenced by other services, allowing users to assemble more complex services.

Using the Saved YARN file to Assemble a Complex Service

This is an example of a service file that references the saved simple-httpd-service service specification.

```
{
  "name": "httpd-proxy-service",
  "version": "1.0.0",
  "lifetime": "3600",
  "components": [
    {
      "name": "simple-httpd-service",
      "artifact": {
```

```
        "id": "simple-httpd-service",
        "type": "SERVICE"
      }
    },
    {
      "name": "httpd-proxy",
      "number_of_containers": 1,
      "artifact": {
        "id": "centos/httpd-24-centos7:latest",
        "type": "DOCKER"
      },
      "launch_command": "/usr/bin/run-httpd",
      "resource": {
        "cpus": 1,
        "memory": "1024"
      },
      "configuration": {
        "files": [
          {
            "type": "TEMPLATE",
            "dest_file": "/etc/httpd/conf.d/httpd-proxy.conf",
            "src_file": "httpd-proxy.conf"
          }
        ]
      }
    }
  ],
  "quicklinks": {
    "Apache HTTP Server": "http://httpd-proxy-0.${SERVICE_NAME}.${USER}.
${DOMAIN}:8080"
  }
}
```

This YARN service file specifies one component of artifact type SERVICE and one component of artifact DOCKER. The SERVICE component will be read from the YARN service file we saved as the service named simple-httpd-service. So after the httpd-proxy-service service is launched, it will contain two httpd containers (which were specified by simple-httpd-service) and one httpd-proxy container. In this case, the httpd-proxy container will balance the load between the two httpd containers. Before launching this service, the httpd-proxy.conf template file that is specified as a src_file for the httpd-proxy component must be uploaded to HDFS. This httpd-proxy-service example is similar to the httpd example provided with YARN, so the same file may be used.

```
hdfs dfs -copyFromLocal /usr/hdp/current/hadoop-yarn-client/yarn-service-
examples/httpd/httpd-proxy.conf
```

```
yarn app -launch httpd-proxy-service /path/to/httpd-proxy-service.json
```

Managing the YARN service life cycle through the REST API

You can perform various operations to manage the life cycle of a YARN service through the REST API.

Create a service

Use the following endpoint to create a service:

```
POST /app/v1/services
```

The execution of this command confirms the success of the service creation request. You cannot be sure if the service will reach a running state. Resource availability and other factors will determine if the service will be deployed in the cluster. You have to call the GET API to get the details of the service and determine its state.

Update a service or upgrade the binary version of a service

You can update the runtime properties of a service. You can update the lifetime, and start or stop a service. You can also upgrade the service containers to a newer version of their artifacts.

Use the following endpoint to update the service:

```
PUT /app/v1/services/{service_name}
```

Destroy a service

Use the following endpoint to destroy a service and release all its resources.

```
DELETE /app/v1/services/{service_name}
```

Get the details of a service

Use the following endpoint to view the details (including containers) of a running service.

```
GET /app/v1/services/{service_name}
```

Set the number of instances of a component

Use the following endpoint to set a component's desired number of instances:

```
PUT /app/v1/services/{service_name}/components/{component_name}
```

YARN Services API Examples

You can use the YARN Services API for situations such as creating a single-component service and performing various operations on the service.

- Create a simple single-component service with most attribute values as defaults

  POST URL – http://localhost:8088/app/v1/services

  POST Request JSON

```
{
  "name": "hello-world",
  "version": "1",
  "components": [
    {
      "name": "hello",
      "number_of_containers": 1,
      "artifact": {
        "id": "nginx:latest",
        "type": "DOCKER"
      },
      "launch_command": "./start_nginx.sh",
      "resource": {
        "cpus": 1,
        "memory": "256"
      }
    }
  ]
}
```

  GET Response JSON

GET URL – http://localhost:8088/app/v1/services/hello-world

Note that a lifetime value of -1 means unlimited lifetime.


```
{
    "name": "hello-world",
    "version": "1",
    "id": "application_1503963985568_0002",
    "lifetime": -1,
    "components": [
        {
            "name": "hello",
            "dependencies": [],
            "resource": {
                "cpus": 1,
                "memory": "256"
            },
            "configuration": {
                "properties": {},
                "env": {},
                "files": []
            },
            "quicklinks": [],
            "containers": [
                {
                    "id": "container_e03_1503963985568_0002_01_000001",
                    "ip": "10.22.8.143",
                    "hostname": "myhost.local",
                    "state": "READY",
                    "launch_time": 1504051512412,
                    "bare_host": "10.22.8.143",
                    "component_name": "hello-0"
                },
                {
                    "id": "container_e03_1503963985568_0002_01_000002",
                    "ip": "10.22.8.143",
                    "hostname": "myhost.local",
                    "state": "READY",
                    "launch_time": 1504051536450,
                    "bare_host": "10.22.8.143",
                    "component_name": "hello-1"
                }
            ],
            "launch_command": "./start_nginx.sh",
            "number_of_containers": 1,
            "run_privileged_container": false
        }
    ],
    "configuration": {
        "properties": {},
        "env": {},
        "files": []
    },
    "quicklinks": {}
}
```


- Update the lifetime of a service

  PUT URL – http://localhost:8088/app/v1/services/hello-world

  PUT Request JSON

  Note that irrespective of what the current lifetime value is, this update request will set the lifetime of the service to 3600 seconds (1 hour) from the time the request is submitted. Therefore, if a service has remaining lifetime

of 5 minutes (for example) and would like to extend it to an hour, OR if an application has remaining lifetime of 5 hours (for example) and would like to reduce it down to one hour, then for both scenarios you would need to submit the following request.

```
{
  "lifetime": 3600
}
```

- Stop a service

  PUT URL – http://localhost:8088/app/v1/services/hello-world

  PUT Request JSON

```
{
    "state": "STOPPED"
}
```

- Start a service

  PUT URL – http://localhost:8088/app/v1/services/hello-world

  PUT Request JSON

```
{
    "state": "STARTED"
}
```

- Increase or decrease the number of containers (instances) of a component of a service

  PUT URL – http://localhost:8088/app/v1/services/hello-world/components/hello

  PUT Request JSON

```
{
    "number_of_containers": 3
}
```

- Destroy a service

  DELETE URL – http://localhost:8088/app/v1/services/hello-world

- Create a complicated service – HBase

  POST URL - http://localhost:8088/app/v1/services/hbase-app-1

```
{
  "name": "hbase-app-1",
  "lifetime": "3600",
  "version": "2.0.0.3.0.0.0",
  "artifact": {
    "id": "hbase:2.0.0.3.0.0.0",
    "type": "DOCKER"
  },
  "configuration": {
    "env": {
      "HBASE_LOG_DIR": "<LOG_DIR>"
    },
    "files": [
      {
        "type": "TEMPLATE",
        "dest_file": "/etc/hadoop/conf/core-site.xml",
        "src_file": "core-site.xml"
      },
      {
```

```
        "type": "TEMPLATE",
        "dest_file": "/etc/hadoop/conf/hdfs-site.xml",
        "src_file": "hdfs-site.xml"
      },
      {
        "type": "XML",
        "dest_file": "/etc/hbase/conf/hbase-site.xml",
        "properties": {
          "hbase.cluster.distributed": "true",
          "hbase.zookeeper.quorum": "${CLUSTER_ZK_QUORUM}",
          "hbase.rootdir": "${SERVICE_HDFS_DIR}/hbase",
          "zookeeper.znode.parent": "${SERVICE_ZK_PATH}",
          "hbase.master.hostname": "hbasemaster-0.${SERVICE_NAME}.${USER}.
${DOMAIN}",
          "hbase.master.info.port": "16010"
        }
      }
    ]
  },
  "components": [
    {
      "name": "hbasemaster",
      "number_of_containers": 1,
      "launch_command": "sleep 15; /usr/hdp/current/hbase-master/bin/hbase
 master start",
      "resource": {
        "cpus": 1,
        "memory": "2048"
      },
      "configuration": {
        "env": {
          "HBASE_MASTER_OPTS": "-Xmx2048m -Xms1024m"
        }
      }
    },
    {
      "name": "regionserver",
      "number_of_containers": 1,
      "launch_command": "sleep 15; /usr/hdp/current/hbase-regionserver/
bin/hbase regionserver start",
      "dependencies": [
        "hbasemaster"
      ],
      "resource": {
        "cpus": 1,
        "memory": "2048"
      },
      "configuration": {
        "files": [
          {
            "type": "XML",
            "dest_file": "/etc/hbase/conf/hbase-site.xml",
            "properties": {
              "hbase.regionserver.hostname": "${COMPONENT_INSTANCE_NAME}.
${SERVICE_NAME}.${USER}.${DOMAIN}"
            }
          }
        ],
        "env": {
          "HBASE_REGIONSERVER_OPTS": "-
XX:CMSInitiatingOccupancyFraction=70 -Xmx2048m -Xms1024m"
        }
      }
    },
```

```
    {
      "name": "hbaseclient",
      "number_of_containers": 1,
      "launch_command": "sleep infinity",
      "resource": {
        "cpus": 1,
        "memory": "1024"
      }
    }
  ],
  "quicklinks": {
    "HBase Master Status UI": "http://hbasemaster-0.${SERVICE_NAME}.
${USER}.${DOMAIN}:16010/master-status"
  }
}
```

Use the YARN CLI to View Logs for Running Applications

You can use the YARN CLI (Command Line Interface) to view log files for running applications.

You can access container log files using the YARN ResourceManager web UI, but more options are available when you use the yarn logs CLI command.

View all Log Files for a Running Application

Use the following command format to view all logs for a running application:

```
yarn logs -applicationId <Application ID>
```

View a Specific Log Type for a Running Application

Use the following command format to view all logs of a particular type for a running application:

```
yarn logs -applicationId <Application ID> -log_files <log_file_type>
```

For example, to view only the stderr error logs:

```
yarn logs -applicationId <Application ID> -log_files stderr
```

The -logFiles option also supports Java regular expressions, so the following format would return all types of log files:

```
yarn logs -applicationId <Application ID> -log_files .*
```

View ApplicationMaster Log Files

Use the following command format to view all ApplicationMaster container log files for a running application:

```
yarn logs -applicationId <Application ID> -am ALL
```

Use the following command format to view only the first ApplicationMaster container log files:

```
yarn logs -applicationId <Application ID> -am 1
```

List Container IDs

Use the following command format to list all container IDs for a running application:

```
yarn logs -applicationId <Application ID> -show_application_log_info
```

View Log Files for One Container

Once you have the container IDs, you can use the following command format to list the log files for a particular container:

```
yarn logs -applicationId <Application ID> -containerId <Container ID>
```

Show Container Log File Information

Use the following command format to list all of the container log file names (types) for a running application:

```
yarn logs -applicationId <Application ID> -show_container_log_info
```

You can then use the -logFiles option to view a particular log type.

View a Portion of the Log Files for One Container

For large container log files, you can use the following command format to list only a portion of the log files for a particular container:

```
yarn logs -applicationId <Application ID> -containerId <Container ID> -size
 <bytes>
```

To view the first 1000 bytes:

```
yarn logs -applicationId <Application ID> -containerId <Container ID> -size
 1000
```

To view the last 1000 bytes:

```
yarn logs -applicationId <Application ID> -containerId <Container ID> -size
 -1000
```

Download Logs for a Running Application

Use the following command format to download logs to a local folder:

```
yarn logs -applicationId <Application ID> -out <path_to_local_folder>
```

The container log files are organized in parent folders labeled with the applicable node ID.

Display Help for YARN Logs

To display Help for yarn logs, run the following command:

```
yarn logs -help
```

Run Multiple MapReduce Versions Using the YARN Distributed Cache

Use the YARN distributed cache to deploy multiple versions of MapReduce.

About this task

Beginning in HDP 2.2, multiple versions of the MapReduce framework can be deployed using the YARN Distributed Cache. By setting the appropriate configuration properties, you can run jobs using a different version of the MapReduce framework than the one currently installed on the cluster.

Distributed cache ensures that the MapReduce job framework version is consistent throughout the entire job life cycle. This enables you to maintain consistent results from MapReduce jobs during a rolling upgrade of the cluster. Without using Distributed Cache, a MapReduce job might start with one framework version, but finish with the new (upgrade) version, which could lead to unpredictable results.

YARN Distributed Cache enables you to efficiently distribute large read-only files (text files, archives, .jar files, etc) for use by YARN applications. Applications use URLs (hdfs:///) to specify the files to be cached, and the Distributed Cache framework copies the necessary files to the applicable nodes before any tasks for the job are executed. Its efficiency stems from the fact that the files are copied only once per job, and archives are extracted after they are copied to the applicable nodes. Note that Distributed Cache assumes that the files to be cached (and specified via hdfs:// URLs) are already present on the HDFS file system and are accessible by every node in the cluster.

Procedure

- Configuring MapReduce for the YARN Distributed Cache
  - Copy the tarball that contains the version of MapReduce you would like to use into an HDFS directory that applications can access.

    ```
    HADOOP_HOME/bin/hdfs dfs -put mapreduce.tar.gz /mapred/framework/
    ```

  - In the mapred-site.xml file, set the value of the mapreduce.application.framework.path property URL to point to the archive file you just uploaded. The URL allows you to create an alias for the archive if a URL fragment identifier is specified. In the following example, ${hdp.version} should be replaced with the applicable HDP version, and mr-framework is specified as the alias:

    ```
    <property>
     <name>mapreduce.application.framework.path</name>
     <value>hdfs:/hdp/apps/${hdp.version}/mapreduce/mapreduce.tar.gz#mr-
    framework</value>
    </property>
    ```

  - In the mapred-site.xml file, the default value of the mapreduce.application.classpath uses the ${hdp.version} environment variable to reference the currently installed version of HDP:

    ```
    <property>
     <name>mapreduce.application.classpath</name>
     <value>$PWD/mr-framework/hadoop/share/hadoop/mapreduce/*:$PWD/mr-
     framework/hadoop/share/hadoop/mapreduce/lib/*:$PWD/mr-
     framework/hadoop/share/hadoop/common/*:$PWD/mr-
     framework/hadoop/share/hadoop/common/lib/*:$PWD/mr-
     framework/hadoop/share/hadoop/yarn/*:$PWD/mr-
     framework/hadoop/share/hadoop/yarn/lib/*:$PWD/mr-
     framework/hadoop/share/hadoop/hdfs/*:$PWD/mr-
     framework/hadoop/share/hadoop/hdfs/lib/*:/usr/hdp/${hdp.version}/
    hadoop/lib/hadoop-lzo-
     0.6.0.${hdp.version}.jar</value>
    </property>
    ```

    Change the value of the mapreduce.application.classpath property to reference the applicable version of the MapReduce framework .jar files. In this case we need to replace ${hdp.version} with the applicable HDP version, which in our example is 2.2.0.0-2041. Note that in the following example the mr-framework alias is used in the path references.

    ```
    <property>
     <name>mapreduce.application.classpath</name>
    ```

```
<value>$PWD/mr-framework/hadoop/share/hadoop/mapreduce/*:$PWD/mr-
framework/hadoop/share/hadoop/mapreduce/lib/*:$PWD/mr-
framework/hadoop/share/hadoop/common/*:$PWD/mr-
framework/hadoop/share/hadoop/common/lib/*:$PWD/mr-
framework/hadoop/share/hadoop/yarn/*:$PWD/mr-
framework/hadoop/share/hadoop/yarn/lib/*:$PWD/mr-
framework/hadoop/share/hadoop/hdfs/*:$PWD/mr-
framework/hadoop/share/hadoop/hdfs/lib/*:/usr/hdp/2.2.0.0-2041/hadoop/
lib/hadoop-lzo-
0.6.0.2.2.0.0-2041.jar</value>
</property>
```

With this configuration in place, MapReduce jobs will run on the version 2.2.0.0-2041 framework referenced in the mapred-site.xml file.

You can upload multiple versions of the MapReduce framework to HDFS and create a separate mapred-site.xml file to reference each version of the framework. Users can then run jobs against a specific version by referencing the applicable mapred-site.xml file. The following example would run a MapReduce job on version 2.1 of the MapReduce framework:

```
hadoop jar share/hadoop/mapreduce/hadoop-mapreduce-examples-*.jar pi -
conf etc/hdp-2.1.0.0/mapred-site.xml 10 10
```

You can use the ApplicationMaster log file to confirm that the job ran on the localized version of MapReduce on the Distributed Cache. For example:

```
2014-06-10 08:19:30,199 INFO [main] org.mortbay.log: Extract jar: file:/
<nm-local-
dirs>/filecache/10/hadoop-2.3.0.tar.gz/hadoop-2.3.0/share/hadoop/yarn/
hadoop-yarn-common-
2.3.0.jar!/webapps/mapreduce to /tmp/
Jetty_0_0_0_0_42544_mapreduce____.pryk9q/webapp
```

- Limitations

  Support for deploying the MapReduce framework via the YARN Distributed Cache currently does not address the job client code used to submit and query jobs. It also does not address the ShuffleHandler code that runs as an auxiliary service within each NodeManager. Therefore, the following limitations apply to MapReduce versions that can be successfully deployed via the Distributed Cache:

  - The MapReduce version must be compatible with the job client code used to submit and query jobs. If it is incompatible, the job client must be upgraded separately on any node on which jobs are submitted using the new MapReduce version.
  - The MapReduce version must be compatible with the configuration files used by the job client submitting the jobs. If it is incompatible with that configuration (that is, a new property must be set, or an existing property value must be changed), the configuration must be updated before submitting jobs.
  - The MapReduce version must be compatible with the ShuffleHandler version running on the cluster nodes. If it is incompatible, the new ShuffleHandler code must be deployed to all nodes in the cluster, and the NodeManagers must be restarted to pick up the new ShuffleHandler code.
- Troubleshooting Tips

  - You can use the ApplicationMaster log file to check the version of MapReduce being used by a running job. For example:

```
2014-11-20 08:19:30,199 INFO [main] org.mortbay.log: Extract jar: file:/
<nm-local-
dirs>/filecache/{...}/hadoop-2.6.0.tar.gz/hadoop-2.6.0/share/hadoop/
yarn/hadoop-yarn-
common-2.6.0.jar!/webapps/mapreduce to /tmp/
Jetty_0_0_0_0_42544_mapreduce____.pryk9q/webapp
```

- If shuffle encryption is enabled, MapReduce jobs may fail with an exception similar to the following:

```
2014-10-10 02:17:16,600 WARN [fetcher#1] org.apache.hadoop.mapreduce.task.reduce.Fetcher: Failed to
 connect to junping-du-centos6.x-3.cs1cloud.internal:13562 with 1 map outputs
javax.net.ssl.SSLHandshakeException: sun.security.validator.ValidatorException: PKIX path building
 failed: sun.security.provider.certpath.SunCertPathBuilderException: unable to find valid certification
 path to requested target
 at com.sun.net.ssl.internal.ssl.Alerts.getSSLException(Alerts.java:174)
 at com.sun.net.ssl.internal.ssl.SSLSocketImpl.fatal(SSLSocketImpl.java:1731)
 at com.sun.net.ssl.internal.ssl.Handshaker.fatalSE(Handshaker.java:241)
 at com.sun.net.ssl.internal.ssl.Handshaker.fatalSE(Handshaker.java:235)
 at com.sun.net.ssl.internal.ssl.ClientHandshaker.serverCertificate(ClientHandshaker.java:1206)
 at com.sun.net.ssl.internal.ssl.ClientHandshaker.processMessage(ClientHandshaker.java:136)
 at com.sun.net.ssl.internal.ssl.Handshaker.processLoop(Handshaker.java:593)
 at com.sun.net.ssl.internal.ssl.Handshaker.process_record(Handshaker.java:529)
 at com.sun.net.ssl.internal.ssl.SSLSocketImpl.readRecord(SSLSocketImpl.java:925)
 at com.sun.net.ssl.internal.ssl.SSLSocketImpl.performInitialHandshake(SSLSocketImpl.java:1170)
 at com.sun.net.ssl.internal.ssl.SSLSocketImpl.startHandshake(SSLSocketImpl.java:1197)
 at com.sun.net.ssl.internal.ssl.SSLSocketImpl.startHandshake(SSLSocketImpl.java:1181)
 at sun.net.www.protocol.https.HttpsClient.afterConnect(HttpsClient.java:434)
 at
 sun.net.www.protocol.https.AbstractDelegateHttpsURLConnection.setNewClient(AbstractDelegateHttpsURLConnection.java:81
 at
 sun.net.www.protocol.https.AbstractDelegateHttpsURLConnection.setNewClient(AbstractDelegateHttpsURLConnection.java:61
 at sun.net.www.protocol.http.HttpURLConnection.writeRequests(HttpURLConnection.java:584)
 at sun.net.www.protocol.http.HttpURLConnection.getInputStream(HttpURLConnection.java:1193)
 at java.net.HttpURLConnection.getResponseCode(HttpURLConnection.java:379)
 at sun.net.www.protocol.https.HttpsURLConnectionImpl.getResponseCode(HttpsURLConnectionImpl.java:318)
 at org.apache.hadoop.mapreduce.task.reduce.Fetcher.verifyConnection(Fetcher.java:427)
 ....
```

To fix this problem, create a sub-directory under $HADOOP_CONF ($HADOOP_HOME/etc/hadoop by default), and copy the ssl-client.xml file to that directory. Add this new directory path (/etc/hadoop/conf/secure) to the MapReduce classpath specified in mapreduce.application.classpath in the mapred-site.xml file.

Enable Cross-Origin Support on YARN

You must enable Cross-Origin Resource Sharing (CORS) on YARN such that the corresponding services accept cross-origin requests from only selected domains. Enabling CORS also helps the YARN UI fetch data endpoints from the browser.

About this task

You must first enable CORS at the cluster level and then on individual components such as YARN.

Procedure

1. In **Ambari Web**, browse to **Services** > **HDFS** > **Configs**, then expand **Advanced core-site**.
2. Add the value org.apache.hadoop.security.HttpCrossOriginFilterInitializer to the existing set of values configured for the hadoop.http.filter.initializers property.
3. Configure values for the following properties depending on your requirements:

Property	Description
hadoop.http.cross-origin.allowed-headers	Comma-separated list of headers allowed for cross-origin support. The default values are X-Requested-With, Content-Type, Accept, Origin, WWW-Authenticate, Accept-Encoding, and Transfer-Encoding.
hadoop.http.cross-origin.allowed-origins	Comma-separated list of origins allowed for cross-origin support. The default value is *. Mention only specific origins so that the services do not accept all the cross-origin requests.

Property	Description
hadoop.http.cross-origin.allowed-methods	Comma-separated list of methods allowed for cross-origin support. The default values are GET, PUT, POST, OPTIONS, HEAD, and DELETE.
hadoop.http.cross-origin.max-age	Number of seconds until when a preflight request can be cached. The default value is 1800 seconds.

4. Click **Save**.
5. Add a description of the modified HDFS configuration and click **Save**.
6. In **Ambari Web**, browse to **Services** > **YARN** > **Configs**, then expand **Advanced yarn-site**.
7. Configure values for the following properties depending on your requirements:

Property	Description
yarn.nodemanager.webapp.cross-origin.enabled	Enable cross-origin support for the NodeManager. The default value is true.
yarn.resourcemanager.webapp.cross-origin.enabled	Enable cross-origin support for the ResourceManager. The default value is true.
yarn.timeline-service.http-cross-origin.enabled	Enable cross-origin support for the timeline service. The default value is true.

Note: The value of the hadoop.http.cross-origin.allowed-origins property in yarn-site.xml overrides the value of the same property in core-site.xml. You can configure the value of this property to allow access to specific domains; for example, regex:.*[.]hwx[.]site(:\d*)?.

8. Click **Save**.
9. Add a description of the modified YARN configuration and click **Save**.
10. Restart the HDFS and YARN services.

Run Docker Containers on YARN

You can configure YARN to run Docker containers.

About this task

Docker containerization makes it easier to package and distribute applications, thereby allowing you to focus on running and fine-tuning applications, as well as significantly reducing "time to deployment" and “time to insight." Docker containerization also provides isolation, and enables you to run multiple versions of the same applications side-by-side. You can have a stable production version of an application, while also evaluating test versions.

Background: the YARN ContainerExecutor

Since its inception, YARN has supported the notion of the ContainerExecutorabstraction. The ContainerExecutor is responsible for:

1. Localizing (downloading and setting up) the resources required for running the container on any given node.
2. Setting up the environment for the container to run (such as creating the directories for the container).
3. Managing the life cycle of the YARN container (launching, monitoring, and cleaning up the container).

In the past, Apache Hadoop shipped with three ContainerExecutors – DefaultContainerExecutor, LinuxContainerExecutor, and WindowsSecureContainerExecutor. Each of these was created to address a specific need. DefaultContainerExecutor is meant for non-secure clusters where all YARN containers are launched as the same user as the NodeManager (providing no security). LinuxContainerExecutor is meant for secure clusters where tasks are launched and run as the user who submitted them. WindowsSecureContainerExecutorprovides similar functionality but on Windows.

The Experimental DockerContainerExecutor

As Docker grew in popularity, DockerContainerExecutor was added to the list of ContainerExecutors. DockerContainerExecutor was the first attempt to add support for Docker in YARN. It would allow users to run tasks as Docker containers. It added support in YARN for Docker commands to allow the NodeManager to launch, monitor and clean up Docker containers as it would for any other YARN container.

There were a couple of limitations of the DockerContainerExecutor – some related to implementation and some architectural. The limits related to implementation were things such a not allowing users to specify the image they wished to run (it required all users to use the same image).

However, the bigger architectural issue is that in YARN, you can use one ContainerExecutor per NodeManager. All tasks will use the ContainerExecutor specified in the node's configuration. As a result, once the cluster was configured to use DockerContainerExecutor, users would be unable to launch regular MapReduce, Tez, or Spark jobs. Additionally, implementing a new ContainerExecutor means that all of the benefits of the existing LinuxContainerExecutor (such as cgroups and traffic shaping) now need to be reimplemented in the new ContainerExecutor. As a result of these challenges, DockerContainerExecutor has been deprecated in favor of a newer abstraction – container runtimes – and DockerContainerExecutor will be removed in a future Apache Hadoop release.

Introducing Container Runtimes

To address these deficiencies, YARN added support for container runtimes in LinuxContainerExecutor. Container runtimes split up the ContainerExecutor into two distinct pieces – the underlying framework required to carry out the functionalities, and a runtime piece that can change depending on the type of container you wish to launch. With these changes, we solve the architectural problem of being able to run regular YARN process containers alongside Docker containers. The life cycle of the Docker container is managed by YARN just as any other container. The change also allows YARN to add support for other containerization technologies in the future.

Currently, two runtimes exist; the process tree based runtime (DefaultLinuxContainerRuntime) and the new Docker runtime (DockerLinuxContainerRuntime). The process-tree based runtime launches containers the same way YARN has always done, whereas, the Docker runtime launches Docker containers. Interfaces exist that can be extended to add new container runtimes. Support for container runtimes, and specifically the DockerLinuxContainerRuntime, is being add through YARN-3611.

Related Concepts

Prerequisites for Running Containerized Spark Jobs

Related Information

Launching Applications Using Docker Containers

Install Docker

YARN-3611

Prerequisites for installing Docker

Docker's container management capabilities are highly dependent on the Linux OS kernel version and capabilities. As a result, Docker only supports systems with modern kernels. It is recommended to check with your operating system vendor for supported system configurations for use with Docker.

See the Support Matrix for the operating system requirements.

Related Information

Docker Storage Drivers

Support Matrix

Recommendations for running Docker containers on YARN

YARN expects that Docker is already installed on all NodeManager hosts where Docker containers will run. Consider these recommendations before installing and configuring Docker for use with YARN.

Docker Version

1.12.5 is the minimum recommended version. Docker is rapidly evolving and shipping multiple releases per year. Not all versions of Docker have been tested. Docker versioning changed in 2017, and is now known as Docker CE. Running a recent version of Docker CE is recommended. Note that recent versions of Docker CE have switched to using the overlay2 storage driver which may not work for all workloads.

RHEL/CentOS provides a version of Docker that can be installed via yum.

Storage Driver

Selecting a storage driver is dependent on OS kernel, workload, and Docker version. It is highly recommended that administrators read the documentation, consult with their operating system vendor, and test the desired workload before making a determination.

Testing has shown that device mapper using LVM is generally stable. Under high write load to the container's root filesystem, device mapper has exhibited panics. SSDs for the Docker graph storage are recommended in this case, but care still needs to be taken. Overlay and overlay2 perform significantly better than device mapper and are recommended if the OS kernel and workload support it.

CGroup Support

YARN provides isolation through the use of cgroups. Docker also has cgroup management built in. If isolation through cgroups if desired, the only recommended solution is to use YARN's cgroup management at this time. YARN will create the cgroup hierarchy and set the the --cgroup-parent flag when launching the container.

For more information about setting YARN cgroups, see *Enabling cgroups*.

The cgroupdriver must be set to cgroupfs. You must ensure that Docker is running using the --exec-opt native.cgroupdriver=cgroupfs docker daemon option.

Note:

The Docker version included with RHEL/CentOS 7.2+ sets the cgroupdriver to systemd. You must change this, typically in the docker.service systemd unit file.

```
vi /usr/lib/systemd/system/docker.service
```

Find and fix the cgroupdriver:

```
--exec-opt native.cgroupdriver=cgroupfs \
```

Also, this version of Docker may include oci-hooks that expect to use the systemd cgroupdriver. Search for oci on your system and remove these files. For example:

```
rm -f /usr/libexec/oci/hooks.d/oci-systemd-hook
rm -f /usr/libexec/oci/hooks.d/oci-register-machine
```

Networking

YARN has support for running Docker containers on a user specified network, however, it does not manage the Docker networks. Administrators are expected to create the networks prior to running the containers. Node labels can be used to isolate particular networks. It is vital to read and understand the Docker networking documentation. Swarm based options are not recommended, however, overlay networks can be used if setup using an external store, such as etcd.

YARN will ask Docker for the networking details, such as IP address and hostname.
As a result, all networking types are supported. Set the environment variable YARN_CONTAINER_RUNTIME_DOCKER_CONTAINER_NETWORK to specify the network to use.

Host networking is only recommended for testing. If the network where the NodeManagers are running has a sufficient number of IP addresses. The bridge networking with --fixed-cidr option works well. Each NodeManager is allocated a small portion of the larger IP space, and then allocates those IP addresses to containers.

To use an administrator defined network, add the network to docker.allowed.networks in container-executor.cfg and yarn.nodemanager.runtime.linux.docker.allowed-container-networks in yarn-site.xml.

Image Management

Images can be preloaded on all NodeManager hosts or they can be implicitly pulled at runtime if they are available in a public Docker registry, such as Docker hub. If the image does not exist on the NodeManager and cannot be pulled, the container will fail.

Docker Bind Mounted Volumes

Note:

Care should be taken when enabling this feature. Enabling access to directories such as, but not limited to, /, /etc, /run, or /home is not advisable and can result in containers negatively impacting the host or leaking sensitive information.

Files and directories from the host are commonly needed within the Docker containers, which Docker provides through volumes. Examples include localized resources, Apache Hadoop binaries, and sockets. In order to make use of this feature, the following must be configured.

The administrator must define the volume whitelist in container-executor.cfg by setting docker.allowed.ro-mounts and docker.allowed.rw-mounts to the list of parent directories that are allowed to be mounted.

The application submitter requests the required volumes at application submission time using the YARN_CONTAINER_RUNTIME_DOCKER_MOUNTS environment variable.

The administrator supplied whitelist is defined as a comma separated list of directories that are allowed to be mounted into containers. The source directory supplied by the user must either match or be a child of the specified directory.

The user supplied mount list is defined as a comma separated list in the form source:destination:mode. The source is the file or directory on the host. The destination is the path within the container where the source will be bind mounted. The mode defines the mode the user expects for the mount, which can be ro (read-only) or rw (read-write).

Related Information

Docker Storage Drivers
Specifying Custom cgroups

Install Docker

Identify the version of Docker provided by your operating system vendor and install it.

About this task

It is recommended to install the version of Docker that is provided by your Operating System vendor. The Docker package has been known by several names; docker-engine, docker, and docker-ce.

Procedure

- Install the package by entering the following command for the respective operating system:

Operating System	Command for Installation
RHEL/CentOS/Oracle Linux 6	yum install docker
SLES 12	zypper install docker
Ubuntu 14 Ubuntu 16	apt-get install docker
Debian 7	apt-get install docker

Configure Docker

Edit the docker daemon.json file and add the required options.

About this task

After Docker is installed, the following is the minimum recommended configuration. Note that configuring the storage driver is not included. This should be added after reviewing the storage driver options.

Procedure

- Edit /etc/docker/daemon.json and add the following options:

```
{
        "live-restore" : true,
        "debug" : true
}
```

- As previously noted, ensure that Docker is using the cgroupfs cgroupdriver option if enabled YARN cgroups support.

  vi /usr/lib/systemd/system/docker.service

  Find and fix the cgroupdriver:

  --exec-opt native.cgroupdriver=cgroupfs \

Configure YARN for running Docker containers

Running Docker containers on YARN works very similar to running existing containers. Containers have access to files that are localized for the container as well as logging.

About this task

To facilitate the use of YARN features, a few rules need to be followed. For the example applications, these steps have already been taken care of.

1. The processes in the containers must run as the user submitting the application (or the local-user in insecure mode).
2. The mount whitelist must include the yarn.local.dirs so that the files needed for the application are available in the container.

The following configuration runs LinuxContainerExecutor in an insecure mode and is only used for testing or where use cases are highly controlled. Kerberos configurations are recommended for production. The local-user is assumed to be nobody, this means that all containers will run as the nobody user.

Before you begin

Make sure YARN cgroups are enabled before configruing YARN for running Docker containers.

To leverage YARN cgroup support, the nodemanager must be configured to use LinuxContainerExecutor. The Docker YARN integration also requires this container executor.

Procedure

1. Set the following properties in the yarn-site.xml file.

```
<property>
    <description>The UNIX user that containers will run as when
    Linux-container-executor is used in nonsecure mode</description>
    <name>yarn.nodemanager.linux-container-executor.nonsecure-mode.local-
user</name>
    <value>nobody</value>
</property>

<property>
    <description>Comma separated list of runtimes that are allowed when
 using
    LinuxContainerExecutor.</description>
    <name>yarn.nodemanager.runtime.linux.allowed-runtimes</name>
    <value>default,docker</value>
    </property>

<property>
    <description>This configuration setting determines the capabilities
    assigned to docker containers when they are launched. While these may
 not
    be case-sensitive from a docker perspective, it is best to keep these
    uppercase. To run without any capabilities, set this value to
    "none" or "NONE"</description>
    <name>yarn.nodemanager.runtime.linux.docker.capabilities</name>
    <value>CHOWN,DAC_OVERRIDE,FSETID,FOWNER,MKNOD,NET_RAW,SETGID,SETUID,
SETFCAP,SETPCAP,NET_BIND_SERVICE,SYS_CHROOT,KILL,AUDIT_WRITE</value>
</property>

<property>
    <description>This configuration setting determines if
    privileged docker containers are allowed on this cluster.
    The submitting user must be part of the privileged container acl and
    must be part of the docker group or have sudo access to the docker
 command
    to be able to use a privileged container. Use with extreme care.</
description>
    <name>yarn.nodemanager.runtime.linux.docker.privileged-
containers.allowed</name>
    <value>false</value>
</property>

<property>
    <description>This configuration setting determines the submitting
    users who are allowed to run privileged docker containers on this
 cluster.
    The submitting user must also be part of the docker group or have sudo
 access
    to the docker command. No users are allowed by default. Use with
 extreme care.
    </description>
```

```
    <name>yarn.nodemanager.runtime.linux.docker.privileged-
containers.acl</name>
    <value> </value>
</property>

<property>
    <description>The set of networks allowed when launching containers</
description>
    <name>yarn.nodemanager.runtime.linux.docker.allowed-container-
networks</name>
    <value>host,bridge</value>
</property>

<property>
    <description>The network used when launching containers when no
 network is specified
    in the request. This network must be one of the (configurable) set of
 allowed
    container networks. The default is host, which may not be appropriate
 for multiple
    containers on a single node, use bridge in that case. See docker
 networking for more.
    </description>
    <name>yarn.nodemanager.runtime.linux.docker.default-container-
network</name>
    <value>host</value>
</property>
```

2. Set the following properties in a container-executor.cfg file.

```
yarn.nodemanager.local-dirs=<yarn.nodemanager.local-dirs from yarn-
site.xml>
yarn.nodemanager.log-dirs=<yarn.nodemanager.log-dirs from yarn-site.xml>
yarn.nodemanager.linux-container-executor.group=hadoop
banned.users=hdfs,yarn,mapred,bin
min.user.id=50

[docker]
module.enabled=true
docker.binary=/usr/bin/docker
docker.allowed.capabilities=CHOWN,DAC_OVERRIDE,FSETID,FOWNER,MKNOD,NET_RAW,
SETGID,SETUID,SETFCAP,SETPCAP,NET_BIND_SERVICE,SYS_CHROOT,KILL,AUDIT_WRITE,
DAC_READ_SEARCH,SYS_PTRACE,SYS_ADMIN
docker.allowed.devices=
docker.allowed.networks=bridge,host,none
docker.allowed.ro-mounts=/sys/fs/cgroup,<yarn.nodemanager.local-dirs from
 yarn-site.xml>
docker.allowed.rw-mounts=<yarn.nodemanager.local-dirs from yarn-site.xml>,
<yarn.nodemanager.log-dirs from yarn-site.xml>
docker.privileged-containers.enabled=false
docker.trusted.registries=local,centos,hortonworks
docker.allowed.volume-drivers=
```

The details of the properties are as follows.

Configuration	Description
yarn.nodemanager.linux-container-executor.group	The Unix group of the NodeManager. It should match the yarn.nodemanager.linux-container-executor.group in the yarn-site.xml file.

Configuration	Description
banned.users	A comma-separated list of usernames who should not be allowed to launch applications. The default setting is: yarn, mapred, hdfs, and bin.
min.user.id	The minimum UID that is allowed to launch applications. The default is no minimum
module.enabled	Must be "true" or "false" to enable or disable launching Docker containers respectively. Default value is 0.
docker.binary	The binary used to launch Docker containers. /usr/bin/docker by default.
docker.allowed.capabilities	The minimum UID that is allowed to launch applications. The default is no minimum.
docker.allowed.devices	Comma separated devices that containers are allowed to mount. By default no devices are allowed to be added.
docker.allowed.networks	Comma separated networks that containers are allowed to use. If no network is specified when launching the container, the default Docker network will be used.
docker.allowed.ro-mounts	Comma separated directories that containers are allowed to mount in read-only mode. By default, no directories are allowed to mounted.
docker.allowed.rw-mounts	Comma separated directories that containers are allowed to mount in read-write mode. By default, no directories are allowed to mounted.
docker.privileged-containers.enabled	Set to "true" or "false" to enable or disable launching privileged containers. Default value is "false". The submitting user must be defined in the privileged container acl setting and must be part of the docker group or have sudo access to the docker command to be able to use a privileged container. Use with extreme care.
docker.trusted.registries	Comma separated list of trusted docker registries for running trusted privileged docker containers. By default, no registries are defined. If the image used for the application does not appear in this list, all capabilities, mounts, and privileges will be stripped from the container.
docker.allowed.volume-drivers	Comma separated volume drivers that containers are allowed to use.

Run Docker on YARN using the YARN services API

You can deploy a load balanced web server pair on a single node HDP cluster using the *YARN Services API*.

About this task

Note that the networking setup here is not appropriate for a multi-node cluster. See the *Networking Recommendations* for more.

Procedure

1. Create the following Yarnfile and save it to /tmp/httpd.json:

```
{
  "name":"httpd-service",
  "version":"1.0.0",
  "lifetime":"3600",
  "configuration":{
     "properties":{
        "docker.network":"bridge"
     }
  },
  "components":[
     {
```

```
         "name":"httpd",
         "number_of_containers":2,
         "artifact":{
            "id":"centos/httpd-24-centos7:latest",
            "type":"DOCKER"
         },
         "launch_command":"/usr/bin/run-httpd",
         "resource":{
            "cpus":1,
            "memory":"1024"
         },
         "readiness_check":{
            "type":"HTTP",
            "properties":{
               "url":"http://${THIS_HOST}:8080"
            }
         },
         "configuration":{
            "files":[
               {
                  "type":"TEMPLATE",
                  "dest_file":"/var/www/html/index.html",
                  "properties":{
                     "content":"<html><header><title>Title</title></
header><body>Hello from ${COMPONENT_INSTANCE_NAME}!</body></html>"
                  }
               }
            ]
         }
      },
      {
         "name":"httpd-proxy",
         "number_of_containers":1,
         "dependencies":[
            "httpd"
         ],
         "artifact":{
            "id":"centos/httpd-24-centos7:latest",
            "type":"DOCKER"
         },
         "launch_command":"/usr/bin/run-httpd",
         "resource":{
            "cpus":1,
            "memory":"1024"
         },
         "configuration":{
            "files":[
               {
                  "type":"TEMPLATE",
                  "dest_file":"/etc/httpd/conf.d/httpd-proxy.conf",
                  "src_file":"httpd-proxy-no-dns.conf"
               }
            ]
         }
      }
   ]
}
```

2. Copy the HTTPD proxy configuration to HDFS. This configuration will be used to configure the HTTPD proxy container to load balance traffic between two backend HTTPD servers.

hdfs dfs -copyFromLocal /usr/hdp/current/hadoop-yarn-client/yarn-service-examples/httpd-no-dns/httpd-proxy-no-dns.conf

3. Submit the HTTPD application using the YARN CLI:

   yarn app -launch httpdservice /tmp/httpd.json

4. Once the application has launched, get the IP address for the HTTPD proxy container to validate that the backend servers have successfully started. The components need to reach a STABLE state and the containers a READY state in the status output before proceeding with next step.

   yarn app -status httpdservice

   Note: use jq or python -m json.tool to pretty print the output.

   The output will appear as follows.

```
{
   "name":"httpdservice",
   "id":"application_1525954234278_0006",
   "lifetime":3549,
   "components":[
      {
         "name":"httpd",
         "dependencies":[

         ],
         "artifact":{
            "id":"centos/httpd-24-centos7:latest",
            "type":"DOCKER"
         },
         "resource":{
            "cpus":1,
            "memory":"1024",
            "additional":{

            }
         },
         "state":"STABLE",
         "configuration":{
            "properties":{
               "docker.network":"bridge"
            },
            "env":{

            },
            "files":[
               {
                  "type":"TEMPLATE",
                  "properties":{
                     "content":"<html><header><title>Title</title></
header><body>Hello from ${COMPONENT_INSTANCE_NAME}!</body></html>"
                  },
                  "dest_file":"/var/www/html/index.html"
               }
            ]
         },
         "quicklinks":[

         ],
         "containers":[
            {
```

```
               "id":"container_1525954234278_0006_01_000002",
               "ip":"172.17.0.2",
               "hostname":"httpd-0.httpdservice.user.domain",
               "state":"READY",
               "launch_time":1525980550770,
               "bare_host":"node.domain",
               "component_instance_name":"httpd-0"
            },
            {
               "id":"container_1525954234278_0006_01_000003",
               "ip":"172.17.0.3",
               "hostname":"httpd-1.httpdservice.user.domain",
               "state":"READY",
               "launch_time":1525980551772,
               "bare_host":"node.domain",
               "component_instance_name":"httpd-1"
            }
         ],
         "readiness_check":{
            "type":"HTTP",
            "properties":{
               "url":"http://${THIS_HOST}:8080"
            }
         },
         "launch_command":"/usr/bin/run-httpd",
         "number_of_containers":2,
         "run_privileged_container":false
      },
      {
         "name":"httpd-proxy",
         "dependencies":[
            "httpd"
         ],
         "artifact":{
            "id":"centos/httpd-24-centos7:latest",
            "type":"DOCKER"
         },
         "resource":{
            "cpus":1,
            "memory":"1024",
            "additional":{

            }
         },
         "state":"STABLE",
         "configuration":{
            "properties":{
               "docker.network":"bridge"
            },
            "env":{

            },
            "files":[
               {
                  "type":"TEMPLATE",
                  "properties":{

                  },
                  "dest_file":"/etc/httpd/conf.d/httpd-proxy.conf",
                  "src_file":"httpd-proxy-no-dns.conf"
               }
            ]
         },
         "quicklinks":[
```

```
         ],
         "containers":[
            {
               "id":"container_1525954234278_0006_01_000005",
               "ip":"172.17.0.4",
               "hostname":"httpd-proxy-0.httpdservice.user.domain",
               "state":"READY",
               "launch_time":1525980579818,
               "bare_host":"node.domain",
               "component_instance_name":"httpd-proxy-0"
            }
         ],
         "launch_command":"/usr/bin/run-httpd",
         "number_of_containers":1,
         "run_privileged_container":false
      }
   ],
   "configuration":{
      "properties":{
         "docker.network":"bridge"
      },
      "env":{

      },
      "files":[

      ]
   },
   "state":"STARTED",
   "quicklinks":{

   },
   "version":"1.0.0",
   "kerberos_principal":{

   }
}
```


Find the details regarding the httpd-proxy-0 instance and record the IP address associated with the container. In the example above, the IP is 172.17.0.4.

5. Using curl, or a similar HTTP client, perform a GET request to the HTTPD proxy on port 8080, using the IP address obtained in the previous step.

   curl http://172.17.0.4:8080/

   The requests should be routed to both backend HTTPD servers, as seen in the following output:

   [user@node ~]$ curl http://172.17.0.4:8080/ <html><header><title>Title</title></header><body>Hello from httpd-0!</body></html>

   [user@node ~]$ curl http://172.17.0.4:8080/ <html><header><title>Title</title></header><body>Hello from httpd-1!</body></html>

   [user@node ~]$ curl http://172.17.0.4:8080/ <html><header><title>Title</title></header><body>Hello from httpd-0!</body></html>

   [user@node ~]$ curl http://172.17.0.4:8080/ <html><header><title>Title</title></header><body>Hello from httpd-1!</body></html>

Accessing the YARN services examples

The YARN Services examples are available in the hdp-assemblies GitHub repository. See this repository for instructions on how to build and store the example YARN Services.

Related Information

The hdp-assemblies repository

Quick start for running YARN services API on Docker containers

You can manage dockerized YARN services on your Hadoop cluster using Docker swarm and an overlay network.

Related Information

Dockerized YARN Services - Quickstart

Configure Docker Swarm and an Overlay Network

You must first install Docker on each cluster node and then configure Docker Swarm and an overlay network.

About this task

Maintain a host list file that specifies every host in the cluster and a worker list file that specifies all the nodes except the one selected to be the Docker Swarm master. You can use this information to run commands in parallel across the cluster.

Procedure

1. Use the pssh command to install Docker on all the hosts in the cluster.

   The following example shows the required commands:

```
pssh -i -h hostlist -l user1 -x "-i ~/user1.pem" "sudo yum install -y yum-
utils device-mapper-persistent-data lvm2"
pssh -i -h hostlist -l user1 -x "-i ~/user1.pem" "sudo yum-config-manager
 --add-repo https://download.docker.com/linux/centos/docker-ce.repo"
pssh -i -h hostlist -l user1 -x "-i ~/user1.pem" "sudo yum install -y
 docker-ce"
pssh -i -h hostlist -l user1 -x "-i ~/user1.pem" "sudo systemctl start
 docker"
pssh -i -h hostlist -l user1 -x "-i ~/user1.pem" "sudo systemctl enable
 docker"
pssh -i -h hostlist -l user1 -x "-i ~/user1.pem" "sudo docker run --rm
 hello-world"
```

2. Configure Docker Swarm and create an overlay network.

   The following example shows the required commands:

```
ssh -i ~/user1.pem user1@<masternode> "sudo docker swarm init"
pssh -i -h workerlist -l user1 -x "-i ~/user1.pem" "sudo <output from last
 command: docker swarm join ...>"
ssh -i ~/user1.pem user1@<masternode> "sudo docker network create -d
 overlay --attachable yarnnetwork"
```

3. Optional: If Kerberos is not enabled, create a default user for containers.

   The following example shows how you can create a default user:

```
pssh -i -h hostlist -l user1 -x "-i ~/user1.pem" "sudo useradd dockeruser"
```

What to do next

Configure Docker settings for YARN using Ambari.

Configure Docker settings for YARN

After installing Docker Swarm and configuring an overlay network on your cluster, you must enable Docker runtime on YARN and set the containers to use the overlay network.

Procedure

1. In the Ambari web, select **Services > YARN > Advanced** to view the YARN advanced properties.
2. In the **YARN Features** section, toggle **Docker Runtime** to **Enabled**.

   Enabling the specified option changes the following setting in **Advanced yarn-site:**

   ```
   yarn.nodemanager.container-
   executor.class=org.apache.hadoop.yarn.server.nodemanager.LinuxContainerExecutor
   ```

3. Expand **Advanced yarn-site** and change the following properties to enable the Docker containers use the overlay network by default:

   ```
   yarn.nodemanager.runtime.linux.docker.default-container-
   network=yarnnetwork
   yarn.nodemanager.runtime.linux.docker.allowed-container-
   networks=host,none,bridge,yarnnetwork
   ```

4. Expand **Custom yarn-site** and add the following property if Kerberos is not enabled:

   ```
   yarn.nodemanager.linux-container-executor.nonsecure-mode.local-
   user=dockeruser
   ```

5. Expand **Advanced Container Executor** and specify the Docker trusted registries in the corresponding field.

   Note: You must specify the Docker images in the form:<registry>/<imageName>:<tag>. You can specify images from other registries as comma-separated values.

6. Click **Save** to save the configurations that you added and restart YARN.

Run the YARN service on a cluster

You can use the YARN services API to define a service and run it on the Docker environment that you have configured. The procedure varies depending on whether your cluster is Kerberized or non-Kerberized.

Run the YARN service on a Kerberized cluster

You must create a Kerberos principal and a keytab, and upload the latter to HDFS. In addition, you must specify the principal in your service definition.

Procedure

1. Create a Kerberos principal of the format <username>/<hostname>@<realm>.
2. Create a keytab for the principal and upload it to HDFS.

   Note: You must ensure that the user for which you are creating the principal has write permissions to the HDFS home directory.

   The following example shows the creation of a keytab for the principal user1/host1.example.com@EXAMPLE.COM, and the command to upload the keytab to HDFS:

   ```
   kadmin.local
   >addprinc user1/host1.example.com@EXAMPLE.COM
   ...
   >xst -k user1_host1.keytab user1/host1.example.com@EXAMPLE.COM
   ...
   >exit
   ```

```
hadoop fs -put user1_host1.keytab hdfs:/user/user1/
hadoop fs -chown user1 hdfs:/user/user1/
```

3. Create a YARN service definition JSON using the REST API.

Note: Ensure that the service definition is unique in the cluster.

The following example shows a YARN service definition added to a JSON file named yarnservice.json:

```
{
  "name": "redis-service",
  "version": "1.0.0",
  "description": "redis example",
  "components" :
    [
      {
        "name": "redis",
        "number_of_containers": 1,
        "artifact": {
          "id": "library/redis",
          "type": "DOCKER"
        },
        "launch_command": "",
        "resource": {
          "cpus": 1,
          "memory": "256"
        },
        "configuration": {
          "env": {
            "YARN_CONTAINER_RUNTIME_DOCKER_RUN_OVERRIDE_DISABLE": "true"
          }
        }
      }
    ],
    "kerberos_principal": {
      "principal_name": "user1/host1.example.com@EXAMPLE.COM",
      "keytab": "hdfs:/user/user1/user1_host1.keytab"
    }
}
```

4. Submit the service definition as specified.

YARN responds with the Application ID.

The following example shows the curl command to submit the service definition:

```
curl --negotiate -u : -X POST -H "Content-Type: application/json" http://
<resource manager>:8088/app/v1/services -d @yarnservice.json
```

5. Optional: Track the status of the service through the YARN UI or by using the REST APIs.

The following example shows the curl command to read the status of the service:

```
curl --negotiate -u : http://<resource manager>:8088/app/v1/services/
redis-service | python -m json.tool
```

Run the YARN service on a non-Kerberized cluster

You can create a YARN service definition JSON using the services API and run it on your cluster.

Procedure

1. Create a YARN service definition JSON using the REST API.

Note: Ensure that the service definition is unique in the cluster.

The following example shows a YARN service definition added to a JSON file named yarnservice.json:

```
{
  "name": "redis-service",
  "version": "1.0.0",
  "description": "redis example",
  "components" :
    [
      {
        "name": "redis",
        "number_of_containers": 1,
        "artifact": {
          "id": "library/redis",
          "type": "DOCKER"
        },
        "launch_command": "",
        "resource": {
          "cpus": 1,
          "memory": "256"
        },
        "configuration": {
          "env": {
            "YARN_CONTAINER_RUNTIME_DOCKER_RUN_OVERRIDE_DISABLE": "true"
          }
        }
      }
    ]
}
```

2. Submit the service definition as specified.

Note: You can submit the service definition only if you have write permissions to the HDFS home directory.

YARN responds with the Application ID.

The following example shows the curl command to submit the service definition:

```
curl --negotiate -u : -X POST -H "Content-Type: application/json" http://
<resource manager>:8088/app/v1/services -d @yarnservice.json
```

3. Optional: Track the status of the service through the YARN UI or by using the REST APIs.

The following example shows the curl command to read the status of the service:

```
curl --negotiate -u : http://<resource manager>:8088/app/v1/services/
redis-service | python -m json.tool
```

Add a local Docker registry

You can create a local Docker registry on your Hadoop cluster so that users can download Docker images locally.

About this task

This procedure *does not* include the steps to configure security options for the registry, and therefore, is *not recommended in a production environment*.

Procedure

1. On the master node, create an instance of the Docker registry container.

   The following example shows the command to create the Docker registry container and bind the registry to port 5000 on the master node:

   ```
   docker run -d -p 5000:5000 --restart=always --name registry -v /mnt/
   registry:/var/lib/registry registry:2
   ```

2. Configure each of the cluster hosts to skip HTTPS checks.

   The following example shows commands to skip HTTPS checks on CentOS 7 hosts:

   ```
   pssh -i -h hostlist -l user1 -x "-i ~/user1.pem" "sudo echo '{\"insecure-
   registries\": [\"<registryHost>:5000\"]}' | sudo tee --append /etc/docker/
   daemon.json"
   pssh -i -h hostlist -l user1 -x "-i ~/user1.pem" "sudo systemctl restart
    docker"
   ```

3. Add the local Docker registry to the list of registries allowed by YARN.

   a) In the Ambari web, select **Services** > **YARN** > **Advanced** to view the YARN advanced properties.

   b) Expand **Advanced Container Executor** and add <registryHost>:5000 to the list of specified container executor values.

Test the local Docker registry

After configuring the local Docker registry, you can push Docker images to the registry so that when a service definition uses an image with that registry prefix, YARN can use the image from the local registry instead of the default public location.

Procedure

1. Build, tag, and push a Docker image to the registry by using docker commands.

   The following example shows the use of the respective commands on an image named myImage:

   ```
   docker build -t myImage:1 .
   docker tag myImage:1 <registryHost>:5000/myImage:1
   docker push <registryHost>:5000/myImage:1
   ```

2. View the pushed image with the help of REST commands.

   The following example shows the use of curl commands to view the image named myImage:

   ```
   curl <registryHost>:5000/v2/_catalog
   curl <registryHost>:5000/v2/_catalog/myImage/tags/list
   ```

3. Download the image to all the hosts in the cluster.

   Note: This step is required to only to demonstrate connectivity. Docker and YARN automatically pull images from the specified registry path.

   The following example shows how you can download the image named myImage to all the hosts in your cluster:

   ```
   pssh -i -h hostlist -l user1 -x "-i ~/user1.pem" "sudo docker pull
    <registryHost>:5000/myImage:1"
   ```

Attaching storage to YARN containers by using CSI

You can attach external storage volumes to Docker containers on YARN by using Container Storage Interface (CSI). The external storage system provides a CSI driver that helps expose the volume to container orchestration frameworks.

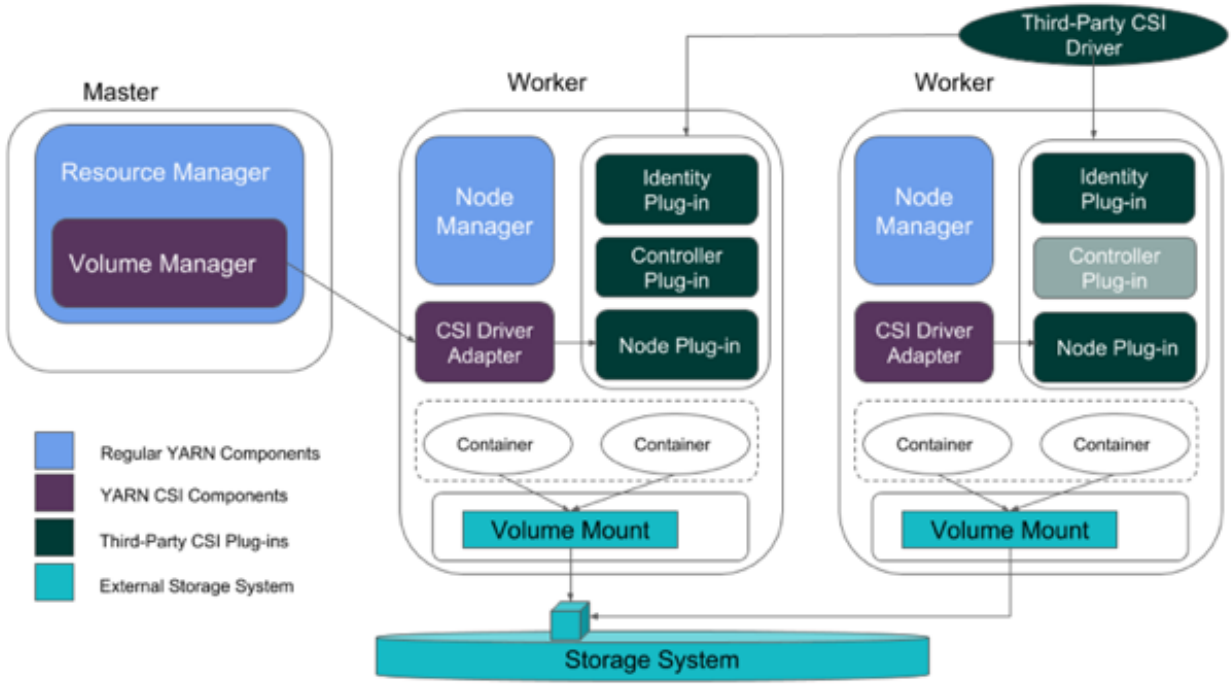
YARN can mount the external volume so that the Docker containers on YARN can read or write the volume data as if it were on the local file system. Using Docker containers on YARN, you can access data on already provisioned Ozone volumes.

Note: Currently, YARN supports only CSI v1. Therefore, YARN is compatible only with drivers that implement the CSI v1 protocol.

YARN components for CSI support

To support Container Storage Interface on YARN, you must deploy the CSI driver associated with the external storage volume on each NodeManager. YARN includes the following additional components for CSI support: volume manager and CSI driver adapter.

The following diagram outlines the interactions between the YARN components and the external storage device with the help of the CSI



driver:

CSI Driver

Provided by the external storage system to interact with YARN, the CSI driver implements all the APIs required to manage the lifecycle of the storage volume. The CSI driver comprises of three gRPC services: identity, controller, and node. The CSI driver is deployed on NodeManagers such that every NodeManager runs instances of identity and node services while one of the cluster nodes runs an instance of the controller service.

Volume Manager

Reads the volume resource specification and handles the required volume operation with the help of the controller service on the CSI driver. This component manages the lifecycle of the storage volume in YARN.

CSI Driver Adapter

Runs on all NodeManagers along with the CSI driver. The adapter proxies all invocations between YARN and the CSI driver. In addition, the adapter registers with the volume manager so that either of them can work with the controller service for handling the volume operations.

Configure CSI on YARN

You must configure the different CSI components on the YARN cluster before you can attach external volumes to YARN containers.

About this task

The steps to configure CSI components on YARN are as follows:

Configure CSI volume processor in ResourceManager

You must add the CSI volume processor as an Application Master Service Processor.

Procedure

Configure the value of the yarn.resourcemanager.application-master-service.processors property in yarn-site.xml to add the CSI volume processor.

Property Name	Value	Description
yarn.resourcemanager.application-master-service.processors	org.apache.hadoop.yarn.server.resourcemanager.volume.csi.processor.VolumeAMSP	Add the CSI volume processor as an Application Master Service Processor.

Deploy CSI driver on each NodeManager

You must manually deploy the CSI driver associated with the external storage on YARN. The CSI driver listens on a UNIX domain socket. Therefore, you must ensure that each NodeManager has the driver configured and running, and that the yarn user has read-write access to the UNIX domain socket file.

Procedure

Configure the values of specific properties in yarn-site.xml to deploy the CSI driver on the NodeManager.

The following table lists the specific properties:

Property Name	Example Value	Description
yarn.nodemanager.csi-driver.names	ch.ctrox.csi.s3-driver	One or more names of the CSI drivers on the NodeManager. The value of this property must be the same as the name returned by the driver's GetPluginInfo API.
yarn.nodemanager.csi-driver.ch.ctrox.csi.s3-driver.endpoint	unix:///tmp/ch.ctrox.csi.s3-driver.sock	The UNIX domain socket path on which the driver listens. This value depends on how the driver is configured.
yarn.nodemanager.csi-driver-adaptor.ch.ctrox.csi.s3-driver.address	0.0.0.0:8901	The service address of the CSI driver adapter for the driver ch.ctrox.csi.s3-driver. After the property is configured, YARN starts a CSI driver adapter service on this address and proxies requests to the CSI driver.

Configure YARN CSI driver adapters

After configuring the CSI volume processor and deploying the CSI driver, you must configure the CSI driver adapters. These adapters run as auxiliary services on NodeManagers. YARN provides a pluggable built-in adapter implementation to integrate with the Ozone CSI driver. The interface helps in plugging in customized CSI driver adaptors, if required.

Procedure

Configure values of specific properties related to auxiliary services in yarn-site.xml to configure the CSI driver adapters.

The following table lists the specific properties:

Property Name	Example Value	Description
yarn.nodemanager.aux-services	csi	One or more keys for auxiliary services.
yarn.nodemanager.aux-services.csi.class	org.apache.hadoop.yarn.csi.adaptor.CsiAdaptorServices	The implementation of the auxiliary service.
yarn.nodemanager.aux-services.csi.classpath	share/hadoop/yarn/csi/hadoop-yarn-csi-3.3.0-SNAPSHOT.jar:share/hadoop/yarn/csi/lib/*	The class path for the auxiliary service csi. The NodeManager launches this service with this separated classpath.
yarn.nodemanager.aux-services.csi.system-classes	org.apache.commons.logging.,org.apache.log4j.	The system classes for the auxiliary service csi. For more information, see Auxiliary Service Classpath Isolation.

Advanced: configure customized CSI driver adapters

If required, you can provide a customized implementation for the CSI driver adapter plug-in, and plug it to the CSI auxiliary services.

Procedure

Configure the value of the specified CSI driver adapter property in yarn-site.xml to a customized value depending on your requirements.

For example, you can plug a new implementation for ch.ctrox.csi.s3-driver, as mentioned in the following table:

Property Name	Example Value	Description
yarn.nodemanager.csi-driver-adaptor.ch.ctrox.csi.s3-driver.class	x.y.z.CustomizedAdaptorService	The implementation class for the CSI driver adapter that maps to the driver csi-driver-adaptor.ch.ctrox.

Cluster Management

You can manage resources for the applications running on your cluster by allocating resources through scheduling, limiting CPU usage by configuring cgroups, partitioning the cluster into subclusters using node labels, and launching applications on Docker containers.

Using Scheduling to Allocate Resources

You can allocate CPU, GPU, and memory among users and groups in a Hadoop cluster. You can use scheduling to allocate the best possible nodes for application containers.

The *CapacityScheduler* is responsible for scheduling. The *ResourceCalculator* is part of the YARN *CapacityScheduler*. The *CapacityScheduler* is used to run Hadoop applications as a shared, multi-tenant cluster in an operator-friendly manner while maximizing the throughput and the utilization of the cluster.

If you have only one type of resource, typically a CPU virtual core (vcore), use the DefaultResourceCalculator. If you have multiple resource types, use the DominantResourceCalculator.

Related Tasks

Enable Cgroups

YARN Resource Allocation

You can manage your cluster capacity using the Capacity Scheduler in YARN. You can use the Capacity Scheduler's DefaultResourceCalculator or the DominantResourceCalculator to allocate available resources.

The fundamental unit of scheduling in YARN is the queue. The capacity of each queue specifies the percentage of cluster resources available for applications submitted to the queue. You can set up queues in a hierarchy that reflects the database structure, resource requirements, and access restrictions required by the organizations, groups, and individuals who use the cluster resources.

You can use the default resource calculator when you want the resource calculator to consider only the available memory for resource calculation. When you use the default resource calculator (DefaultResourceCalculator), resources are allocated based on the available memory.

If you have multiple resource types, use the dominant resource calculator (DominantResourceCalculator) to enable CPU, GPU, and memory scheduling. The dominant resource calculator is based on the Dominant Resource Fairness (DRF) model of resource allocation. DRF is designed to fairly distribute CPU, GPU, and memory resources among different types of processes in a mixed-workload cluster.

For example, if User A runs CPU-heavy tasks and User B runs memory-heavy tasks, the DRF allocates more CPU and less memory to the tasks run by User A, and allocates less CPU and more memory to the tasks run by User B. In a single resource case, in which all jobs are requesting the same resources, the DRF reduces to max-min fairness for that resource.

Related Information

Dominant Resource Fairness: Fair Allocation of Mulitple Resources

Use CPU Scheduling

Cgroups with CPU scheduling helps you effectively manage mixed workloads.

MapReduce jobs only

If you primarily run MapReduce jobs on your cluster, enabling CPU scheduling does not change performance much. The dominant resource for MapReduce is memory, so the DRF scheduler continues to balance MapReduce jobs in a manner similar to the default resource calculator. In the case of a single resource, the DRF reduces to max-min fairness for that resource.

Mixed workloads

An example of a mixed workload is a cluster that runs both MapReduce and Storm on YARN. MapReduce is not CPU-constrained, but Storm on YARN is; its containers require more CPU than memory. As you add Storm jobs along with MapReduce jobs, the DRF scheduler tries to balance memory and CPU resources, but you might see some performance degradation in as a result. As you add more CPU-intensive Storm jobs, individual jobs start to take longer to run as the cluster CPU resources are consumed.

To solve this problem, you can use cgroups along with CPU scheduling. Using cgroups provides isolation for CPU-intensive processes such as Storm on YARN, thereby enabling you to predictably plan and constrain the CPU-intensive Storm containers.

You can also use node labels in conjunction with CPU scheduling and cgroups to restrict Storm on YARN jobs to a subset of cluster nodes.

Note: You should use CPU scheduling only in a Linux environment, because there is no isolation mechanism (cgroups equivalent) for Windows.

Configure CPU Scheduling and Isolation

You can configure CPU scheduling on your Ambari or non-Ambari cluster to allocate the best possible nodes having the required CPU resources for application containers.

Enable CPU scheduling and isolation on an Ambari cluster

To enable CPU scheduling on an Ambari cluster, select YARN > CONFIGS on the Ambari dashboard, then click CPU Scheduling and Isolation under CPU. Click Save, then restart all cluster components that require a restart.

Enable CPU scheduling on a non-Ambari cluster

1. On the ResourceManager and NodeManager hosts, enable CPU scheduling in capacity-scheduler.xml by replacing the DefaultResourceCalculator portion of the <value> string with DominantResourceCalculator:

   Property: yarn.scheduler.capacity.resource-calculator

   Value: org.apache.hadoop.yarn.util.resource.DominantResourceCalculator

   Example:

   ```
   <property>
    <name>yarn.scheduler.capacity.resource-calculator</name>
    <!--
    <value>org.apache.hadoop.yarn.util.resource.DefaultResourceCalculator</
   value> -->
    <value>org.apache.hadoop.yarn.util.resource.DominantResourceCalculator</
   value>
   </property>
   ```

2. Set vcores in yarn-site.xml

   On the ResourceManager and NodeManager hosts, set the number of vcores to match the number of physical CPU cores on the NodeManager host by providing the number of physical cores as the <value>.

   You should set the number of vcores to match the number of physical CPU cores on the NodeManager hosts. Set the following property in the /etc/hadoop/conf/yarn-site.xml file on the ResourceManager and NodeManager hosts:

   Property: yarn.nodemanager.resource.cpu-vcores

   Value: <number_of_physical_cores>

   Example:

   ```
   <property>
    <name>yarn.nodemanager.resource.cpu-vcores</name>
   <value>16</value>
   </property>
   ```

What to do next

Enable cgroups along with CPU scheduling. Cgroups is used as the isolation mechanism for CPU processes. With cgroups strict enforcement activated, each CPU process receives only the resources it requests. Without cgroups activated, the DRF scheduler attempts to balance the load, but unpredictable behavior may occur. For more information, see *Enabling cgroups*.

Related Concepts

Using Scheduling to Allocate Resources

Limit CPU Usage with Cgroups

Configure GPU Scheduling and Isolation

On an Ambari cluster, you can configure GPU scheduling and isolation. On a non-Ambari cluster, you must configure certain properties in the capacity-scheduler.xml, resource-types.xml, and yarn-site.xml files. Currently only Nvidia GPUs are supported in YARN.

Before you begin

- YARN NodeManager must be installed with the Nvidia drivers.

Enable GPU scheduling and isolation on an Ambari cluster

1. Select YARN > CONFIGS on the Ambari dashboard.
2. Click GPU Scheduling and Isolation under GPU.
3. In the Absolute path of nvidia-smi on NodeManagers field, enter the absolute path to the nvidia-smi GPU discovery executable. For example, /usr/local/bin/nvidia-smi
4. Click Save, and then restart all the cluster components that require a restart.

If the NodeManager fails to start, and you see the following error:

```
INFO  gpu.GpuDiscoverer (GpuDiscoverer.java:initialize(240)) - Trying to
 discover GPU information ...
WARN  gpu.GpuDiscoverer (GpuDiscoverer.java:initialize(247)) - Failed to
 discover GPU information from system,
exception message:ExitCodeException exitCode=12:  continue...
```

Export the LD_LIBRARY_PATH in the yarn -env.sh using the following command: export LD_LIBRARY_PATH=/usr/local/nvidia/lib:/usr/local/nvidia/lib64:$LD_LIBRARY_PATH

Enable GPU scheduling and isolation on a non-Ambari cluster

DominantResourceCalculator must be configured first before you enable GPU scheduling/isolation. Configure the following property in the/etc/hadoop/conf/capacity-scheduler.xml file

Property: yarn.scheduler.capacity.resource-calculator

Value: org.apache.hadoop.yarn.util.resource.DominantResourceCalculator

1. Enable GPU scheduling in the /etc/hadoop/conf/resource-types.xml file on the ResourceManager and NodeManager hosts:

   Property: yarn.resource-types

   Value: yarn.io/gpu

   Example:

   ```
   <configuration>
     <property>
        <name>yarn.resource-types</name>
        <value>yarn.io/gpu</value>
     </property>
   </configuration>
   ```

2. Enable GPU isolation in the the /etc/hadoop/conf/yarn-site.xml file on the NodeManager host:

   Property: yarn.nodemanager.resource-plugins

   Value: yarn.io/gpu

   Example:

   ```
   <configuration>
     <property>
        <name>yarn.nodemanager.resource-plugins</name>
        <value>yarn.io/gpu</value>
     </property>
   </configuration>
   ```

3. Set the following advanced properties in the /etc/hadoop/conf/yarn-site.xml file on the NodeManager host:

- To allow GPU devices:

  Property: yarn.nodemanager.resource-plugins.gpu.allowed-gpu-devices

  Value: auto

  Note: The auto setting enables YARN to automatically detect and manage GPU devices. For other options, see YARN-7223.

- To allow YARN NodeManager to to locate discovery executable:

  Property: yarn.nodemanager.resource-plugins.gpu.path-to-discovery-executables

  Value: <absolute_path_to_nvidia-smi_binary>

  Note: Supports only nvidia-smi.

  Example: /usr/local/bin/nvidia-smi

4. Set the following property in the /etc/hadoop/conf/yarn-site.xml file on the NodeManager host to automatically mount cgroup sub-devices:

   - Property: yarn.nodemanager.linux-container-executor.cgroups.mount

     Value: true

5. Set the following configuration in the /etc/hadoop/conf/container-executor.cfg to run GPU applications under non-Docker environment:

   - In the GPU section, set:

     Property: module.enabled=true

   - In the cgroups section, set:

     Property: root=/sys/fs/cgroup

     Note: This should be same as yarn.nodemanager.linux-container-executor.cgroups.mount-path in the yarn-site.xml file

     Property: yarn-hierarchy=yarn

     Note: This should be same as yarn.nodemanager.linux-container-executor.cgroups.hierarchy in the yarn-site.xml file

Related Concepts

Using Scheduling to Allocate Resources

Options to Run Distributed Shell and GPU

You can run the distributed shell by specifying resources other than memory and vCores.

Procedure

- Use the following command to run the distributed shell and GPU without a Docker container:

```
yarn jar <path/to/hadoop-yarn-applications-distributedshell.jar> \
  -jar <path/to/hadoop-yarn-applications-distributedshell.jar> \
  -shell_command /usr/local/nvidia/bin/nvidia-smi \
  -container_resources memory-mb=3072,vcores=1,yarn.io/gpu=2 \
  -num_containers 2
```

You receive output similar to the following:

```
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 375.66                 Driver Version: 375.66                    |
|-------------------------------+----------------------+----------------------+
```

```
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|===============================+======================+======================|
|   0  Tesla P100-PCIE...  Off  | 0000:04:00.0     Off |                    0 |
| N/A   30C    P0    24W / 250W |      0MiB / 12193MiB |      0%      Default |
+-------------------------------+----------------------+----------------------+
|   1  Tesla P100-PCIE...  Off  | 0000:82:00.0     Off |                    0 |
| N/A   34C    P0    25W / 250W |      0MiB / 12193MiB |      0%      Default |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                       GPU Memory |
|  GPU       PID  Type  Process name                               Usage      |
|=============================================================================|
|  No running processes found                                                 |
+-----------------------------------------------------------------------------+
```

- Use the following command to run the distributed shell and GPU with a Docker container:

```
yarn jar <path/to/hadoop-yarn-applications-distributedshell.jar> \
       -jar <path/to/hadoop-yarn-applications-distributedshell.jar> \
       -shell_env YARN_CONTAINER_RUNTIME_TYPE=docker \
       -shell_env YARN_CONTAINER_RUNTIME_DOCKER_IMAGE=<docker-image-name>
 \
       -shell_command nvidia-smi \
       -container_resources memory-mb=3072,vcores=1,yarn.io/gpu=2 \
       -num_containers 2
```

GPU support for Docker

You can use GPUs in big data applications such as machine learning, data analytics, and genome sequencing. Docker containerization makes it easier for you to package and distribute applications. You can enable GPU support when using YARN on Docker containers.

The NVIDIA Docker plug-in enables you to deploy GPU-accelerated applications with NVIDIA Docker support. When you use Docker containers, you must install NVIDIA Docker plug-in 1.0 to detect and set up GPU containers automatically.

Related Tasks

Enable GPU Support for Docker on an Ambari Cluster

Enable GPU Support for Docker on a non-Ambari Cluster

Related Information

nvidia-docker

Enable GPU Support for Docker on an Ambari Cluster

On an Ambari cluster, you can enable GPU support for Docker using the Ambari web user interface.

Procedure

1. On the Ambari dashboard, select **YARN** > **CONFIGS**.
2. Under **YARN Features**, click **Docker Runtime**.
3. Click Save.

What to do next

After saving the configuration changes, a Restart indicator appears next to components that require a restart. You must restart components affected by a configuration change so that the component uses the updated configuration values.

Related Concepts

GPU support for Docker

Related Information

nvidia-docker

Enable GPU Support for Docker on a non-Ambari Cluster

If you decide not to perform default installation and configuration of NVIDIA Docker plug-in, you must manually configure a non-Ambari cluster to enable GPU support for Docker.

Procedure

1. Go to the /etc/hadoop/conf/yarn-site.xml configuration file in your NodeManager.
2. Configure the Nvidia Docker plugin for GPU:

   Property: yarn.nodemanager.resource-plugins.gpu.docker-plugin

   Default Value: nvidia-docker-v1
3. Configure the Docker end point for nvidia-docker-plugin:

   Property: yarn.nodemanager.resource-plugins.gpu.docker-plugin.nvidia-docker-v1.endpoint

   Default Value: http://localhost:3476/v1.0/docker/cli

Results

Sample configuration in your yarn-site.xml file:

Example

```
<property>
   <name>yarn.nodemanager.resource-plugins.gpu.docker-plugin</name>
   <value>nvidia-docker-v1</value>
</property>

<property>
   <name>yarn.nodemanager.resource-plugins.gpu.docker-plugin.nvidia-docker-
v1.endpoint</name>
   <value>http://localhost:3476/v1.0/docker/cli</value>
</property>
```

Related Concepts

GPU support for Docker

Related Information

nvidia-docker

Limit CPU Usage with Cgroups

You can use cgroups to limit CPU usage in a Hadoop Cluster.

You can use cgroups to isolate CPU-heavy processes in a Hadoop cluster. If you are using CPU Scheduling, you should also use cgroups to constrain and manage CPU processes. If you are not using CPU Scheduling, do not enable cgroups.

When you enable CPU Scheduling, queues are still used to allocate cluster resources, but both CPU and memory are taken into consideration using a scheduler that utilizes Dominant Resource Fairness (DRF). In the DRF model, resource allocation takes into account the dominant resource required by a process. CPU-heavy processes (such as Storm-on-YARN) receive more CPU and less memory. Memory-heavy processes (such as MapReduce) receive more memory and less CPU. The DRF scheduler is designed to fairly distribute memory and CPU resources among different types of processes in a mixed- workload cluster.

Cgroups compliments CPU Scheduling by providing CPU resource isolation. It enables you to set limits on the amount of CPU resources granted to individual YARN containers, and also lets you set a limit on the total amount of CPU resources used by YARN processes.

Cgroups represents one aspect of YARN resource management capabilities that includes CPU Scheduling, node labels, archival storage, and memory as storage. If CPU Scheduling is used, cgroups should be used along with it to constrain and manage CPU processes.

Related Tasks

Configure CPU Scheduling and Isolation

Enable Cgroups

On an Ambari cluster, you can enable CPU Scheduling to enable cgroups. On a non-Ambari cluster, you must configure certain properties in yarn-site.xml on the ResourceManager and NodeManager hosts to enable cgroups.

About this task

cgroups is a Linux kernel feature. cgroups is supported on the following Linux operating systems:

- CentOS 6.9, 7.3
- RHEL 6.9, 7.3
- SUSE 12
- Ubuntu 16

At this time there is no cgroups equivalent for Windows. cgroups are not enabled by default on HDP. cgroups require that the HDP cluster be Kerberos enabled.

Important:

The yarn.nodemanager.linux-container-executor.cgroups.mount property must be set to false. Setting this value to true is not currently supported.

Procedure

Enable cgroups

The following commands must be run on every reboot of the NodeManager hosts to set up the cgroup hierarchy. Note that operating systems use different mount points for the cgroup interface. Replace /sys/fs/cgroup with your operating system equivalent.

```
mkdir -p /sys/fs/cgroup/cpu/yarn
chown -R yarn /sys/fs/cgroup/cpu/yarn
mkdir -p /sys/fs/cgroup/memory/yarn
chown -R yarn /sys/fs/cgroup/memory/yarn
mkdir -p /sys/fs/cgroup/blkio/yarn
chown -R yarn /sys/fs/cgroup/blkio/yarn
mkdir -p /sys/fs/cgroup/net_cls/yarn
chown -R yarn /sys/fs/cgroup/net_cls/yarn
mkdir -p /sys/fs/cgroup/devices/yarn
chown -R yarn /sys/fs/cgroup/devices/yarn
```

- To enable cgroups on an Ambari cluster, select YARN > Configs on the Ambari dashboard, then click CPU Isolation under CPU. Click Save, then restart all cluster components that require a restart. cgroups should be enabled along with CPU Scheduling.
- On a non-Ambari cluster, set the following properties in the /etc/hadoop/conf/yarn-site.xml file on the ResourceManager and NodeManager hosts.

  Property: yarn.nodemanager.container-executor.class

  Value: org.apache.hadoop.yarn.server.nodemanager.LinuxContainerExecutor

  Example:

  ```
  <property>
   <name>yarn.nodemanager.container-executor.class</name>
  ```

```
 <value>org.apache.hadoop.yarn.server.nodemanager.LinuxContainerExecutor</
value>
</property>
```

Property: yarn.nodemanager.linux-container-executor.group

Value: hadoop

Example:

```
<property>
 <name>yarn.nodemanager.linux-container-executor.group</name>
 <value>hadoop</value>
</property>
```

Property: yarn.nodemanager.linux-container-executor.resources-handler.class

Value: org.apache.hadoop.yarn.server.nodemanager.util.cgroupsLCEResourcesHandler

Example:

```
<property>
 <name>
yarn.nodemanager.linux-container-executor.resources-handler.class</name>
 <value>
org.apache.hadoop.yarn.server.nodemanager.util.cgroupsLCEResourcesHandler
</value>
</property>
```

Property: yarn.nodemanager.linux-container-executor.cgroups.hierarchy

Value: /yarn

Example:

```
<property>
 <name>yarn.nodemanager.linux-container-executor.cgroups.hierarchy</name>
 <value>/yarn</value>
</property>
```

Property: yarn.nodemanager.linux-container-executor.cgroups.mount

Value: false

Example:

```
<property>
 <name>yarn.nodemanager.linux-container-executor.cgroups.mount</name>
 <value>false</value>
</property>
```

Property: yarn.nodemanager.linux-container-executor.cgroups.mount-path

Value: /sys/fs/cgroup

Example:

```
<property>
 <name>yarn.nodemanager.linux-container-executor.cgroups.mount-path</name>
 <value>/sys/fs/cgroup</value>
</property>
```

Set the Percentage of CPU used by YARN

Set the percentage of CPU that can be allocated for YARN containers. In most cases, the default value of 100% should be used. If you have another process that needs to run on a node that also requires CPU resources, you can lower the percentage of CPU allocated to YARN to free up resources for the other process.

Property: yarn.nodemanager.resource.percentage-physical-cpu-limit

Value: 100

Example:

```
<property>
 <name>yarn.nodemanager.resource.percentage-physical-cpu-limit</name>
 <value>100</value>
</property>
```

Set Flexible or Strict CPU limits

CPU jobs are constrained with CPU scheduling and cgroups enabled, but by default these are flexible limits. If spare CPU cycles are available, containers are allowed to exceed the CPU limits set for them. With flexible limits, the amount of CPU resources available for containers to use can vary based on cluster usage -- the amount of CPU available in the cluster at any given time.

You can use cgroups to set strict limits on CPU usage. When strict limits are enabled, each process receives only the amount of CPU resources it requests. With strict limits, a CPU process will receive the same amount of cluster resources every time it runs.

Strict limits are not enabled (set to false) by default.

Property: yarn.nodemanager.linux-container-executor.cgroups.strict-resource-usage

Value: false

Example:

```
 <property>
 <name>yarn.nodemanager.linux-container-executor.cgroups.strict-resource-
usage</name>
 <value>false</value>
 </property>
```

Note:

Irrespective of whether this property is true or false, at no point will total container CPU usage exceed the limit set in yarn.nodemanager.resource.percentage-physical-cpu-limit.

Important: CPU resource isolation leverages advanced features in the Linux kernel. At this time, setting yarn.nodemanager.linux-container-executor.cgroups.strict-resource-usage to true is *not* recommended due to known kernel panics. In addition, with some kernels, setting yarn.nodemanager.resource.percentage-physical-cpu-limit to a value less than 100 can result in kernel panics. If you require either of these features, you must perform scale testing to determine if the in-use kernel and workloads are stable. As a starting point, Linux kernel version 4.8.1 works with these features. However, testing the features with the desired workloads is very important.

Related Concepts

Using Scheduling to Allocate Resources

Recommendations for running Docker containers on YARN

Using Cgroups

You can use strict cgroups CPU limits to constrain CPU processes in mixed workload clusters.

One example of a mixed workload is a cluster that runs both MapReduce and Storm-on-YARN. MapReduce is not CPU-constrained (MapReduce containers do not ask for much CPU). Storm-on-YARN is CPU-constrained: its

containers ask for more CPU than memory. As you start adding Storm jobs along with MapReduce jobs, the DRF scheduler attempts to balance memory and CPU resources, but as more CPU-intensive Storm jobs are added, they may begin to take up the majority of the cluster CPU resources.

You can use cgroups along with CPU scheduling to help manage mixed workloads. cgroups provide isolation for CPU-heavy processes such as Storm-on-YARN, thereby enabling you to predictably plan and constrain the CPU-intensive Storm containers.

When you enable strict cgroup CPU limits, each resource gets only what it asks for, even if there is extra CPU available. This is useful for scenarios involving charge-backs or strict SLA enforcement, where you always need to know exactly what percentage or CPU is being used.

Also, enabling strict CPU limits would make job performance predictable, whereas without setting strict limits a CPU-intensive job would run faster when the cluster was not under heavy use, but slower when more jobs were running in the cluster. Strict CPU limits would therefore also be useful for benchmarking.

You can also use node labels in conjunction with cgroups and CPU scheduling to restrict Storm-on-YARN jobs to a subset of cluster nodes.

If you are using cgroups and want more information on CPU performance, you can review the statistics available in the /cgroup/cpu/yarn/cpu.stat file.

Managing Device Plug-ins (Technical Preview)

Hortonworks Data Platform (HDP) 3.x releases before 3.1.4 supported the YARN GPU/FPGA device plug-in by writing custom code. Implementing such a device plug-in required knowledge of YARN integration and internal configuration related to NodeManager.

Starting with HDP 3.1.4, the plug-in framework simplifies the development process with minimal YARN configuration parameters and provides a more flexible way to integrate with YARN.

Note: This feature is a technical preview and considered under development. Do not use this feature in your production systems. If you have questions regarding this feature, contact Support by logging a case on our Cloudera Support Portal at https://my.cloudera.com/support.html.

Use the device plug-in

You can use the sample GPU/FPGA device plug-in available in the framework or develop your own plug-in.

As an example, the new framework includes a sample implementation of Nvidia GPU plug-in supporting the detection of Nvidia GPUs with a customized scheduler, and isolating containers run with both YARN cgroups and Nvidia Docker runtime v2. The examples discussed here are based on the Nvidia sample plug-in.

Related Concepts

Package the plug-in

Prerequisites for using the device plug-in

Before you use the sample GPU/FPGA device plug-in in your framework or develop your own plug-in, you must ensure that you install the required software and enable a few configuration parameters.

- Enable LinuxContainerExecutor on YARN to handle resource isolation and Docker.

  For information on enabling LinuxContainerExecutor, see Configure YARN for running Docker containers.
- Install the required GPU drivers and Docker runtime on the nodes. For information about the required drivers, refer to the device documentation.
- To use the YARN capacity scheduler, add the DominantResourceCalculator configuration in the capacity-scheduler.xml file.

```
<property>
  <name>yarn.scheduler.capacity.resource-calculator</name>
<value>org.apache.hadoop.yarn.util.resource.DominantResourceCalculator</
value>
```

```
</property>
```

Enable the device plug-in framework

After you install the required software and enable the required configuration parameters listed as prerequisites for using the device plug-in, configure the yarn.nodemanager.pluggable-device-framework.enabled parameter in the yarn-site.xml file.

Procedure

- To enable the device plug-in, set the following property in the /etc/hadoop/conf/yarn-site.xml file on the ResourceManager host:

```
<property>
   <name>yarn.nodemanager.pluggable-device-framework.enabled</name>
        <value>true</value>
 </property>
```

- To enable the isolation native module, set the following property in the container-executor.cfg file.

```
<property>
   <name>module.enabled </name>
        <value>true</value>
 </property>
```

Configure the device plug-in

You must first define a resource name for the device plug-in to identify the resource name. After you define a name, add the resource name in the yarn-site.xml file.

Procedure

1. Define the resource name in the resource-types.xml file for the pluggable device framework to identify the resource name the plug-in is handling.

   The following example defines the resource name as nvidia.com/gpu:

```
<property>
  <name>yarn.resource-types</name>
  <value>nvidia.com/gpu</value>
</property>
```

2. Define the resource name handled by the plug-in in the yarn-site.xml file. The property value must be a full class name of the plug-in.

   For example:

```
<property>
  <name>yarn.nodemanager.pluggable-device-framework.device-classes</name>
 <value>org.apache.hadoop.yarn.server.nodemanager.containermanager.resourceplugin.com
value>
</property>
```

   From HDP 3.1.4 onwards, a new plug-in for NEC's vector engine is available and you can enable the following configuration property to use it. Use comma separated multiple values in below configurations to use different hardware resources at the same time.

   Add the following in the resource-types.xml file

```
<property>
  <name>yarn.resource-types</name>
   <value>nec.com/ve</value>
</property>
```

Add the following in the yarn-site.xml file

```
<property>
  <name>yarn.nodemanager.pluggable-device-framework.device-classes</name>

 <value>org.apache.hadoop.yarn.server.nodemanager.containermanager.resourceplugin.com
value>
</property>
```

Restart YARN and run a job

You must restart YARN services for all the configuration changes to take effect.

Procedure

1. Restart the YARN cluster service.
2. (Optional) After you restart the YARN, you can see the new device resource count while accessing the YARN UI2 Overview and the NodeManager pages or by running the following command:

   yarn node -list -showDetails
3. Run the job requesting several nvidia.com/gpu resources:

   yarn jar <path/to/hadoop-yarn-applications-distributedshell.jar> \

Node Manager API to query resource allocation

When you run a job with resources like nvidia.com/gpu, you can query a Node Manager node's resource allocation using the RESTful API. Note that the resource name should be in the URL encoded format.

In this example, nvidia.com%2Fgpu is the name of the resource.

```
node:port/ws/v1/node/resources/nvidia.com%2Fgpu
```

Use the following command to get the JSON format resource allocation:

```
curl localhost:8042/ws/v1/node/resources/nvidia.com%2Fgpu | jq .
```

Related reference

Develop a device plug-in

Develop a device plug-in

When you start the Node Manager, the new device plug-in is automatically loaded into the framework. You must implement only the DevicePlugin interface and optionally the DevicePluginScheduler interface.

As an example, the new framework includes a sample implementation of Nvidia GPU plug-in supporting Nvidia GPUs detection with a custom scheduler, and isolating containers run with both YARN cgroups and Nvidia Docker runtime v2. You can use the example as a reference for developing your device plug-in.

Related Concepts

Node Manager API to query resource allocation

Prerequisites for developing a device plug-in

You can develop your device plug-in using Maven or SBT. You must add the required property in your build file.

- If you are using Maven to build the project, add the following property in the pom.xml file:

```
<dependencies>
  <dependency>
    <groupId>org.apache.hadoop</groupId>
      <artifactId>hadoop-yarn-server-nodemanager</artifactId>
      <version>3.3.0</version>
```

```
      <scope>provided</scope>
  </dependency>
</dependencies>
```

- If you are using SBT to build your project, add the following line in the build.sbt file:

```
libraryDependencies += "org.apache.hadoop" % "hadoop-yarn-server-
nodemanager" % "3.3.0"
```

After adding the property, you can implement the plug-in interfaces based on classes provided in org.apache.hadoop.yarn.server.nodemanager.api.deviceplugin.

Define the device plug-in interface

You must define the plug-in class for implementing the DevicePlugin interface.

DevicePlugin Interface

```
/**
 * A must interface for vendor plugin to implement.
 * */
public interface DevicePlugin {
  /**
   * Called first when device plugin framework wants to register.
   * @return DeviceRegisterRequest {@link DeviceRegisterRequest}
   * @throws Exception
   * */
  DeviceRegisterRequest getRegisterRequestInfo()
      throws Exception;

  /**
   * Called when update node resource.
   * @return a set of {@link Device}, {@link java.util.TreeSet} recommended
   * @throws Exception
   * */
  Set<Device> getDevices() throws Exception;

  /**
   * Asking how these devices should be prepared/used
   * before/when container launch. A plugin can do some tasks in its own or
   * define it in DeviceRuntimeSpec to let the framework do it.
   * For instance, define {@code VolumeSpec} to let the
   * framework to create volume before running container.
   *
   * @param allocatedDevices A set of allocated {@link Device}.
   * @param yarnRuntime Indicate which runtime YARN will use
   *        Could be {@code RUNTIME_DEFAULT} or {@code RUNTIME_DOCKER}
   *        in {@link DeviceRuntimeSpec} constants. The default means YARN's
   *        non-docker container runtime is used. The docker means YARN's
   *        docker container runtime is used.
   * @return a {@link DeviceRuntimeSpec} description about environment,
   * {@link         VolumeSpec}, {@link MountVolumeSpec}. etc
   * @throws Exception
   * */
  DeviceRuntimeSpec onDevicesAllocated(Set<Device>; allocatedDevices,
      YarnRuntimeType yarnRuntime) throws Exception;

  /**
   * Called after device released.
   * @param releasedDevices A set of released devices
   * @throws Exception
   * */
  void onDevicesReleased(Set<Device> releasedDevices)
```

```
        throws Exception;
}
```

Property	Description
getRegisterRequestInfo	This method is used for the plug-in to get a new resource type name and then the ResourceManager. The DeviceRegisterRequest returned by the method consists of a plug-in version and a resource type name. For example, nvidia.com/gpu.
getDevices	This method is used to get the latest vendor device list in this Node Manager node. The resource count pre-defined in node-resources.xml will be overridden. It is recommended that the vendor plug-in manages the allowed devices reported to YARN in its own configuration. YARN can only have a blacklist configuration specified using the devices.denied-numbers parameter in the container-executor.cfg file. In this method, you may invoke a shell command or invoke RESTful/RPC to remote service to get the list of devices whenever required.  **Note:** The Device object can describe a fake device. If the major device number, minor device number and device path are blank, the framework does not do isolation for it. This provides feasibility for you to define a fake device without real hardware.
onDevicesAllocated	This method is invoked to provide information to the framework on how to use these devices. The Node Manager invokes this interface to let the plug-in start preparation tasks like create volume before container launch and provides information on how to expose the devices to container when launching it. This is described in the DeviceRuntimeSpec interface. For example, DeviceRuntimeSpec can describe the container launch requirements like environment variables, device and volume mounts, Docker runtime type, and so on.
onDeviceReleased	This method is used for the plug-in to do clean up work like device reset before the container terminates.

Implement the DevicePluginScheduler interface

You can implement the DevicePluginScheduler interface if you want to use the plug-in with a more efficient scheduler. The allocateDevices method is invoked by YARN each time when checking the plug-in's recommended devices for one container. This interface is optional because YARN provides a very basic scheduler.

A sample NvidiaGPUPluginForRuntimeV2 plug-in for a customized scheduler is available in the framework for your reference. This sample scheduler is for Nvidia GPU topology and can get considerable performance boost for the container.

Package the plug-in

After you update the interfaces, package it as a jar and save it in the Hadoop classpath. The recommended directory of the plug-in is $HADOOP_COMMOND_HOME/share/hadoop/yarn.

Related reference

Use the device plug-in

Partition a Cluster Using Node Labels

You can use Node labels to partition a cluster into sub-clusters so that jobs run on nodes with specific characteristics.

You can use Node labels to run YARN applications on cluster nodes that have a specified node label. Node labels can be set as exclusive or shareable:

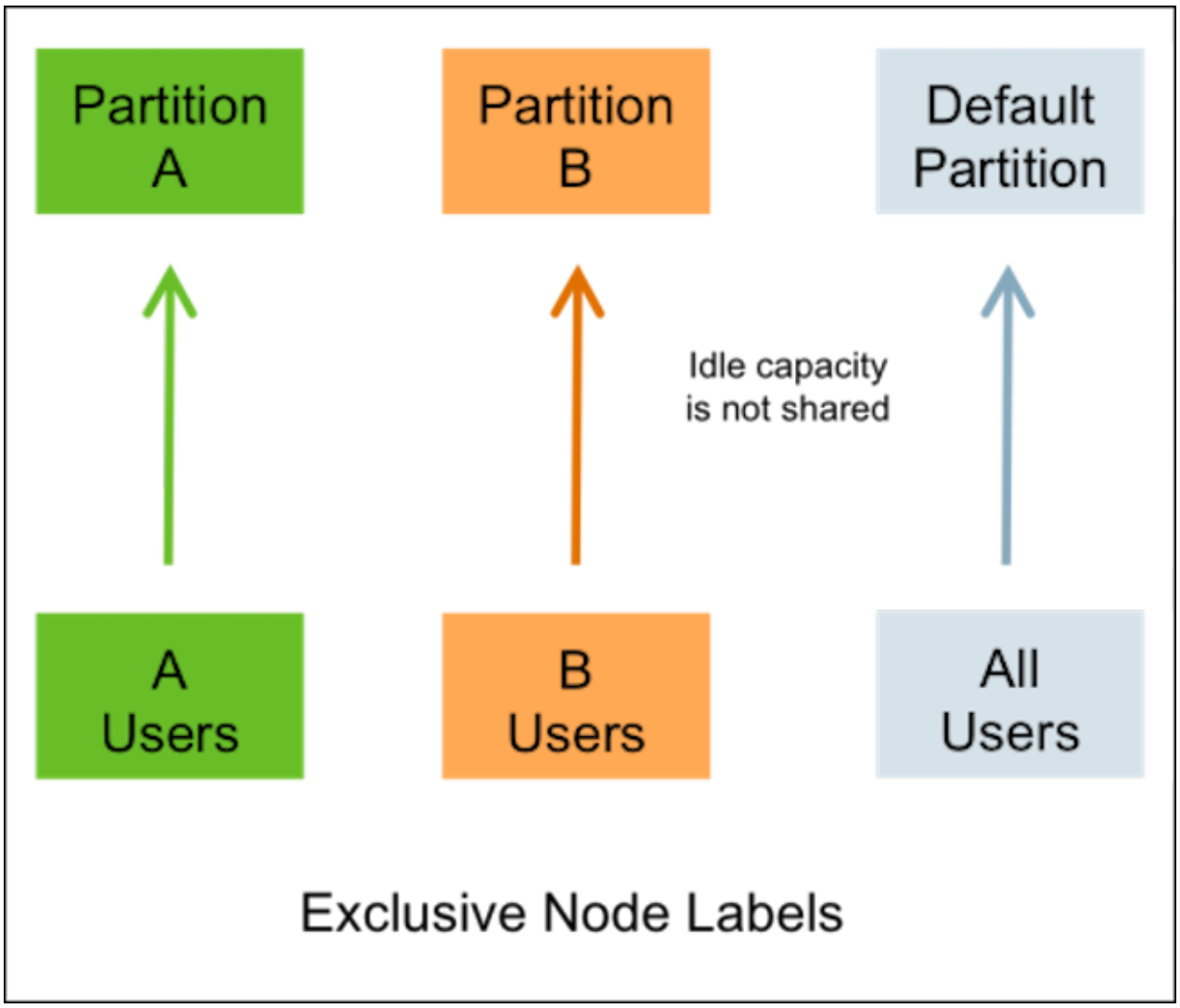
- exclusive -- Access is restricted to applications running in queues associated with the node label.
- sharable -- If idle capacity is available on the labeled node, resources are shared with all applications in the cluster.

The fundamental unit of scheduling in YARN is the queue. The capacity of each queue specifies the percentage of cluster resources that are available for applications submitted to the queue. Queues can be set up in a hierarchy that reflects the resource requirements and access restrictions required by the various organizations, groups, and users that utilize cluster resources.

Node labels enable you partition a cluster into sub-clusters so that jobs can be run on nodes with specific characteristics. For example, you can use node labels to run memory-intensive jobs only on nodes with a larger amount of RAM. Node labels can be assigned to cluster nodes, and specified as exclusive or shareable. You can then associate node labels with capacity scheduler queues. Each node can have only one node label.

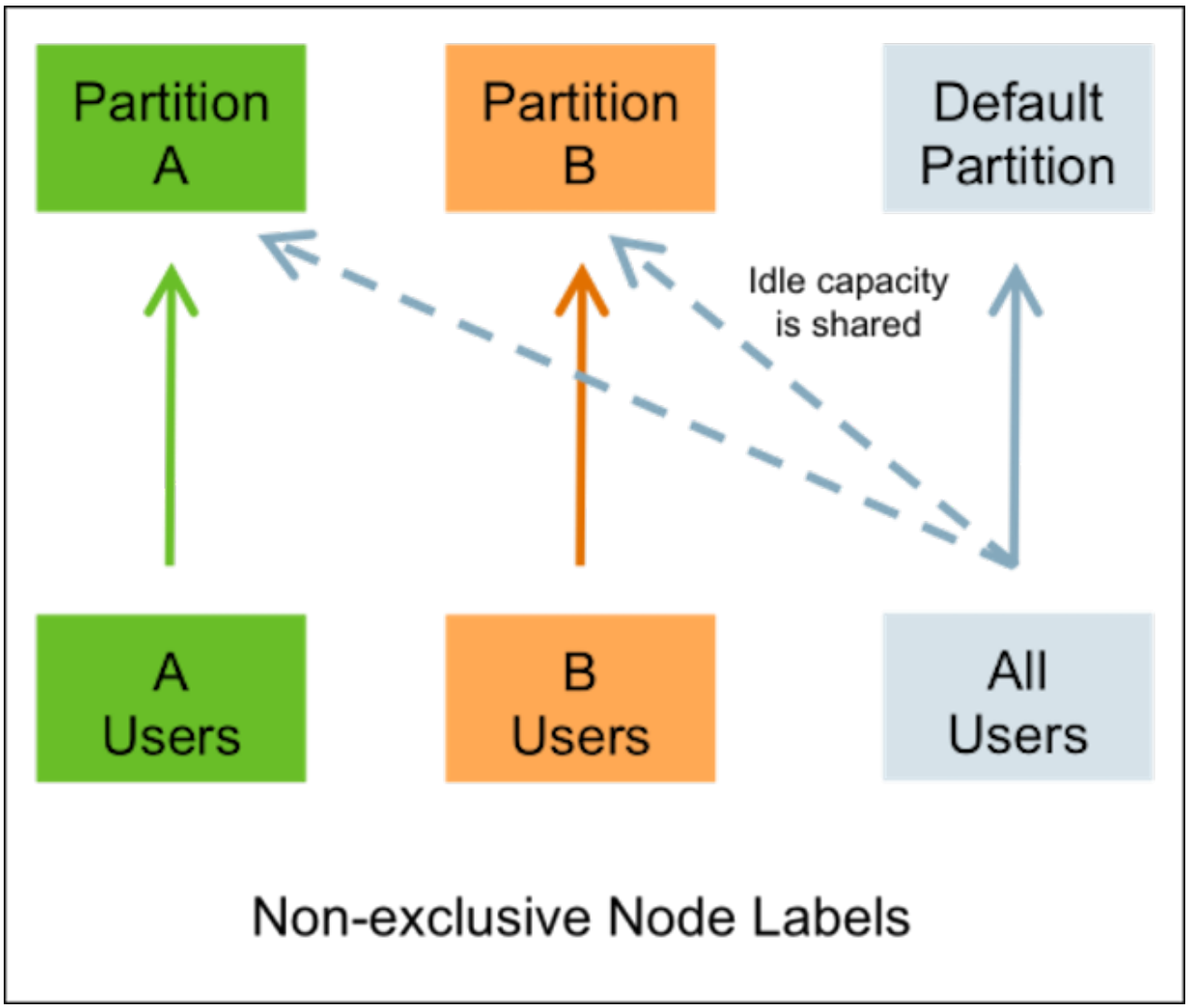
Exclusive Node Labels

When a queue is associated with one or more exclusive node labels, all applications submitted by the queue have exclusive access to nodes with those labels.



Shareable Node Labels

When a queue is associated with one or more shareable (non-exclusive) node labels, all applications submitted by the queue get first priority on nodes with those labels. If idle capacity is available on the labeled nodes, resources are shared with other non-labeled applications in the cluster. Non-labeled applications will be preempted if labeled applications request new resources on the labeled nodes.



Queues without Node Labels

If no node label is assigned to a queue, the applications submitted by the queue can run on any node without a node label, and on nodes with shareable node labels if idle resources are available.

Preemption

Labeled applications that request labeled resources preempt non-labeled applications on labeled nodes. If a labeled resource is not explicitly requested, the normal rules of preemption apply. Non-labeled applications cannot preempt labeled applications running on labeled nodes.

Configure Node Labels

You can either use Ambari to configure node labels on a cluster, or make configuration changes on the YARN ResourceManager host if you are using a non-Ambari cluster.

Enable Node Labels on an Ambari Cluster

1. Select YARN > Configs on the Ambari dashboard, then enable Node Labels under YARN Features.
2. Click Save, then restart all cluster components that require a restart.

Enable Node Labels on a non-Ambari Cluster

To enable Node Labels on a non-Ambari cluster, make the following configuration changes on the YARN ResourceManager host.

1. Create a Label Directory in HDFS

   Use the following commands to create a "node-labels" directory in which to store the Node Labels in HDFS.

   ```
   sudo su hdfs
   hadoop fs -mkdir -p /yarn/node-labels
   hadoop fs -chown -R yarn:yarn /yarn
   hadoop fs -chmod -R 700 /yarn
   ```

   -chmod -R 700 specifies that only the yarn user can access the "node-labels" directory.

   You can then use the following command to confirm that the directory was created in HDFS.

   ```
   hadoop fs -ls /yarn
   ```

   The new node label directory should appear in the list returned by the following command. The owner should be yarn, and the permission should be drwx.

   ```
   Found 1 items
   drwx------ - yarn yarn 0 2014-11-24 13:09 /yarn/node-labels
   ```

   Use the following commands to create a /user/yarn directory that is required by the distributed shell.

   ```
   hadoop fs -mkdir -p /user/yarn
   hadoop fs -chown -R yarn:yarn /user/yarn
   hadoop fs -chmod -R 700 /user/yarn
   ```

   The preceding commands assume that the yarn user will be submitting jobs with the distributed shell. To run the distributed shell with a different user, create the user, then use /user/<user_name> in the file paths of the commands above to create a new user directory.
2. Configure YARN for Node Labels

   Add the following properties to the /etc/hadoop/conf/yarn-site.xml file on the ResourceManager host.

   Set the following property to enable Node Labels:

   ```
   <property>
         <name>yarn.node-labels.enabled</name>
         <value>true</value>
   </property>
   ```

   Set the following property to reference the HDFS node label directory:

   ```
   <property>
         <name>yarn.node-labels.fs-store.root-dir</name>
         <value>hdfs://<host>:<port>/
   <absolute_path_to_node_label_directory></value>
    </property>
   ```

   For example:

   ```
   <property>
        <name>yarn.node-labels.fs-store.root-dir</name>
        <value>hdfs://node-1.example.com:8020/yarn/node-labels/</value>
   </property>
   ```
3. Start or Restart the YARN ResourceManager

In order for the configuration changes in the yarn-site.xml file to take effect, you must stop and restart the YARN ResourceManager if it is running, or start the ResourceManager if it is not running.

Use the following command to stop the ResourceManager:

```
su -l yarn -c "/usr/hdp/current/hadoop-yarn-resourcemanager/sbin/yarn-
daemon.sh stop resourcemanager"
```

Use the following command to start the ResourceManager:

```
su -l yarn -c "/usr/hdp/current/hadoop-yarn-resourcemanager/sbin/yarn-
daemon.sh start resourcemanager"
```

Add Node Labels

Use the following command format to add Node Labels. You should run these commands as the yarn user. Node labels must be added before they can be assigned to nodes and associated with queues.

```
sudo su yarn
yarn rmadmin -addToClusterNodeLabels "<label1>(exclusive=<true|
false>),<label2>(exclusive=<true|false>)"
```

Note:

If exclusive is not specified, the default value is true.

For example, the following commands add the node label "x" as exclusive, and "y" as shareable (non-exclusive).

```
sudo su yarn
yarn rmadmin -addToClusterNodeLabels "x(exclusive=true),y(exclusive=false)"
```

You can use the yarn cluster --list-node-labels command to confirm that Node Labels have been added:

```
[root@node-1 /]# yarn cluster --list-node-labels
15/07/11 13:55:43 INFO impl.TimelineClientImpl: Timeline service address:
 http://node-1.example.com:8188/ws/v1/timeline/
15/07/11 13:55:43 INFO client.RMProxy: Connecting to ResourceManager at
 node-1.example.com/240.0.0.10:8032
Node Labels: <x:exclusivity=true>,<y:exclusivity=false>
```

You can use the following command format to remove Node Labels:

```
yarn rmadmin -removeFromClusterNodeLabels "<label1>,<label2>"
```

Note:

You cannot remove a node label if it is associated with a queue.

Assign Node Labels to Cluster Nodes

Use the following command format to add or replace node label assignments on cluster nodes:

```
yarn rmadmin -replaceLabelsOnNode "<node1>:<port>=<label1>
 <node2>:<port>=<label2>"
```

For example, the following commands assign node label "x" to "node-1.example.com", and node label "y" to "node-2.example.com".

```
sudo su yarn
```

```
yarn rmadmin -replaceLabelsOnNode "node-1.example.com=x
 node-2.example.com=y"
```

Note:

You can only assign one node label to each node. Also, if you do not specify a port, the node label change will be applied to all NodeManagers on the host.

To remove node label assignments from a node, use -replaceLabelsOnNode, but do not specify any labels. For example, you would use the following commands to remove the "x" label from node-1.example.com:

```
sudo su yarn
yarn rmadmin -replaceLabelsOnNode "node-1.example.com"
```

Associate Node Labels with Queues

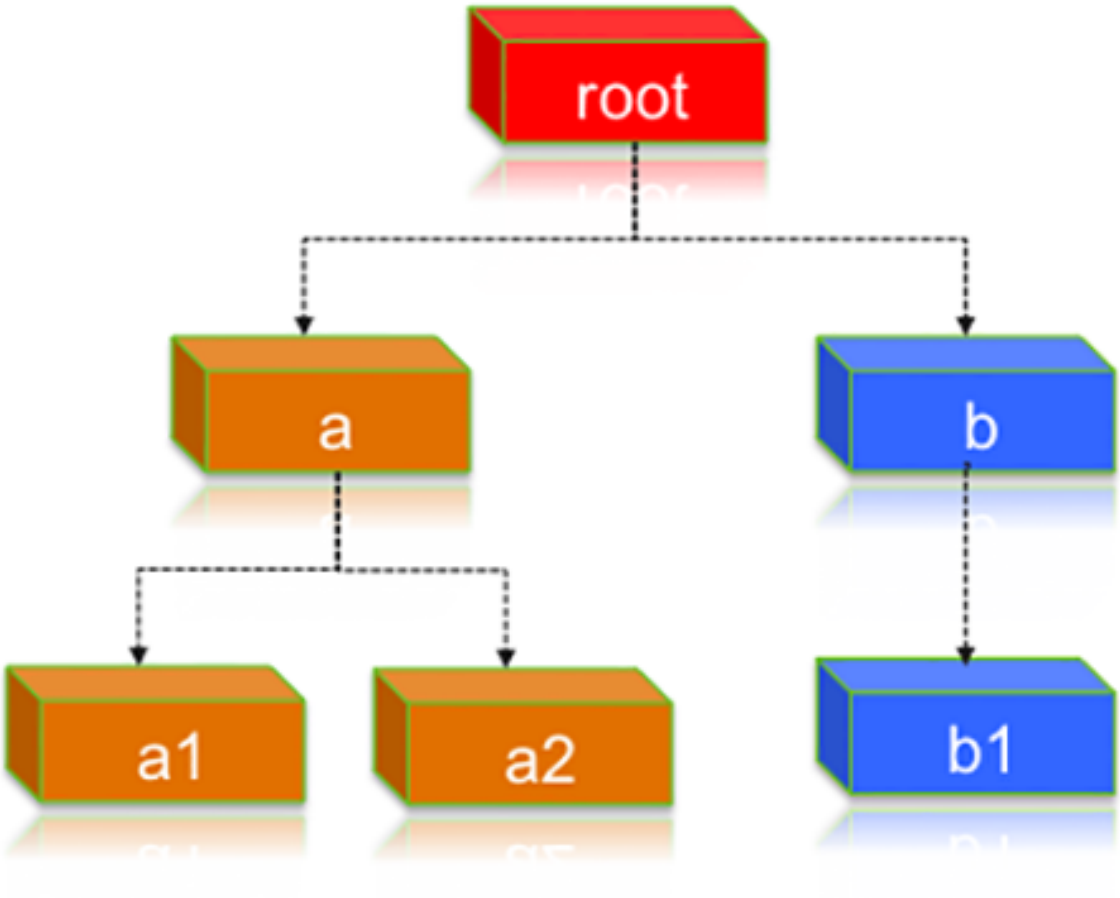
Now that we have created Node Labels, we can associate them with queues in the /etc/hadoop/conf/capacity-scheduler.xml file.

You must specify capacity on each node label of each queue, and also ensure that the sum of capacities of each node-label of direct children of a parent queue at every level is equal to 100%. Node labels that a queue can access (accessible Node Labels of a queue) must be the same as, or a subset of, the accessible Node Labels of its parent queue.

Example:

Assume that a cluster has a total of 8 nodes. The first 3 nodes (n1-n3) have node label=x, the next 3 nodes (n4-n6) have node label=y, and the final 2 nodes (n7, n8) do not have any Node Labels. Each node can run 10 containers.

The queue hierarchy is as follows:



Assume that queue “a” can access Node Labels “x” and “y”, and queue “b” can only access node label “y”. By definition, nodes without labels can be accessed by all queues.

Consider the following example label configuration for the queues:

capacity(a) = 40, capacity(a, label=x) = 100, capacity(a, label=y) = 50; capacity(b) = 60, capacity(b, label=y) = 50

This means that:

- Queue “a” can access 40% of the resources on nodes without any labels, 100% of the resources on nodes with label=x, and 50% of the resources on nodes with label=y.
- Queue “b” can access 60% of the resources on nodes without any labels, and 50% of the resources on nodes with label=y.

You can also see that for this configuration:

capacity(a) + capacity(b) = 100 capacity(a, label=x) + capacity(b, label=x) (b cannot access label=x, it is 0) = 100 capacity(a, label=y) + capacity(b, label=y) = 100

For child queues under the same parent queue, the sum of the capacity for each label should equal 100%.

Similarly, we can set the capacities of the child queues a1, a2, and b1:

a1 and a2: capacity(a.a1) = 40, capacity(a.a1, label=x) =30, capacity(a.a1, label=y) =50 capacity(a.a2) = 60, capacity(a.a2, label=x) =70, capacity(a.a2, label=y) =50 b1: capacity(b.b1) = 100 capacity(b.b1, label=y) = 100

You can see that for the a1 and a2 configuration:

capacity(a.a1) + capacity(a.a2) = 100 capacity(a.a1, label=x) + capacity(a.a2, label=x) = 100 capacity(a.a1, label=y) + capacity(a.a2, label=y) = 100

How many resources can queue a1 access?

Resources on nodes without any labels: Resource = 20 (total containers that can be allocated on nodes without label, in this case n7, n8) * 40% (a.capacity) * 40% (a.a1.capacity) = 3.2 (containers)

Resources on nodes with label=x

Resource = 30 (total containers that can be allocated on nodes with label=x, in this case n1-n3) * 100% (a.label-x.capacity) * 30% = 9 (containers)

To implement this example configuration, you would add the following properties in the /etc/hadoop/conf/capacity-scheduler.xml file.

```
 <property>
 <name>yarn.scheduler.capacity.root.queues</name>
 <value>a,b</value>
 </property>

 <property>
 <name>yarn.scheduler.capacity.root.accessible-node-labels.x.capacity</name>
 <value>100</value>
 </property>

 <property>
 <name>yarn.scheduler.capacity.root.accessible-node-labels.y.capacity</name>
 <value>100</value>
 </property>

 <!-- configuration of queue-a -->
 <property>
 <name>yarn.scheduler.capacity.root.a.accessible-node-labels</name>
 <value>x,y</value>
 </property>

 <property>
 <name>yarn.scheduler.capacity.root.a.capacity</name>
 <value>40</value>
 </property>

 <property>
 <name>yarn.scheduler.capacity.root.a.accessible-node-labels.x.capacity</
name>
 <value>100</value>
 </property>

 <property>
 <name>yarn.scheduler.capacity.root.a.accessible-node-labels.y.capacity</
name>
 <value>50</value>
 </property>

 <property>
 <name>yarn.scheduler.capacity.root.a.queues</name>
 <value>a1,a2</value>
 </property>

 <!-- configuration of queue-b -->
 <property>
 <name>yarn.scheduler.capacity.root.b.accessible-node-labels</name>
 <value>y</value>
 </property>

 <property>
 <name>yarn.scheduler.capacity.root.b.capacity</name>
 <value>60</value>
 </property>
```

```
 <property>
 <name>yarn.scheduler.capacity.root.b.accessible-node-labels.y.capacity</
name>
 <value>50</value>
 </property>

 <property>
 <name>yarn.scheduler.capacity.root.b.queues</name>
 <value>b1</value>
 </property>

 <!-- configuration of queue-a.a1 -->
 <property>
 <name>yarn.scheduler.capacity.root.a.a1.accessible-node-labels</name>
 <value>x,y</value>
 </property>

 <property>
 <name>yarn.scheduler.capacity.root.a.a1.capacity</name>
 <value>40</value>
 </property>

 <property>
 <name>yarn.scheduler.capacity.root.a.a1.accessible-node-labels.x.capacity</
name>
 <value>30</value>
 </property>

 <property>
 <name>yarn.scheduler.capacity.root.a.a1.accessible-node-labels.y.capacity</
name>
 <value>50</value>
 </property>

 <!-- configuration of queue-a.a2 -->
 <property>
 <name>yarn.scheduler.capacity.root.a.a2.accessible-node-labels</name>
 <value>x,y</value>
 </property>

 <property>
 <name>yarn.scheduler.capacity.root.a.a2.capacity</name>
 <value>60</value>
 </property>

 <property>
 <name>yarn.scheduler.capacity.root.a.a2.accessible-node-labels.x.capacity</
name>
 <value>70</value>
 </property>

 <property>
 <name>yarn.scheduler.capacity.root.a.a2.accessible-node-labels.y.capacity</
name>
 <value>50</value>
 </property>

 <!-- configuration of queue-b.b1 -->
 <property>
 <name>yarn.scheduler.capacity.root.b.b1.accessible-node-labels</name>
 <value>y</value>
 </property>
```

```
 <property>
 <name>yarn.scheduler.capacity.root.b.b1.capacity</name>
 <value>100</value>
 </property>

 <property>
 <name>yarn.scheduler.capacity.root.b.b1.accessible-node-labels.y.capacity</
name>
 <value>100</value>
 </property>
```

Refresh Queues

After adding or updating queue node label properties in the capacity-scheduler.xml file, you must run the following commands to refresh the queues:

```
sudo su yarn
yarn rmadmin -refreshQueues
```

Confirm Node Label Assignments

You can use the following commands to view information about node labels.

- List all running nodes in the cluster: yarn node -list

  Example:

  ```
  [root@node-1 /]# yarn node -list
  14/11/21 12:14:06 INFO impl.TimelineClientImpl: Timeline service address:
   http://node-1.example.com:8188/ws/v1/timeline/
  14/11/21 12:14:07 INFO client.RMProxy: Connecting to ResourceManager at
   node-1.example.com/240.0.0.10:8032
  Total Nodes:3
   Node-Id Node-State Node-Http-Address Number-of-Running-Containers
  node-3.example.com:45454 RUNNING node-3.example.com:50060 0
  node-1.example.com:45454 RUNNING node-1.example.com:50060 0
  node-2.example.com:45454 RUNNING node-2.example.com:50060 0
  ```

- List all node labels in the cluster: yarn cluster --list-node-labels

  Example:

  ```
  [root@node-1 /]# yarn cluster --list-node-labels
  15/07/11 13:55:43 INFO impl.TimelineClientImpl: Timeline service address:
   http://node-1.example.com:8188/ws/v1/timeline/
  15/07/11 13:55:43 INFO client.RMProxy: Connecting to ResourceManager at
   node-1.example.com/240.0.0.10:8032
  Node Labels: <x:exclusivity=true>,<y:exclusivity=false>
  ```

- List the status of a node (includes node labels): yarn node -status <Node_ID>

  Example:

  ```
  [root@node-1 /]# yarn node -status node-1.example.com:45454
  14/11/21 06:32:35 INFO impl.TimelineClientImpl: Timeline service address:
   http://node-1.example.com:8188/ws/v1/timeline/
  14/11/21 06:32:35 INFO client.RMProxy: Connecting to ResourceManager at
   node-1.example.com/240.0.0.10:8032
  Node Report :
   Node-Id : node-1.example.com:45454
   Rack : /default-rack
   Node-State : RUNNING
   Node-Http-Address : node-1.example.com:50060
  ```

```
Last-Health-Update : Fri 21/Nov/14 06:32:09:473PST
Health-Report :
Containers : 0
Memory-Used : 0MB
Memory-Capacity : 1408MB
CPU-Used : 0 vcores
CPU-Capacity : 8 vcores
Node-Labels : x
```

Node labels are also displayed in the ResourceManager UI on the Nodes and Scheduler pages.

Specify a Child Queue with No Node Label

If no node label is specified for a child queue, it inherits the node label setting of its parent queue. To specify a child queue with no node label, use a blank space for the value of the node label.

For example:

```
<property>
<name>yarn.scheduler.capacity.root.b.b1.accessible-node-labels</name>
<value> </value>
</property>
```

Set a Default Queue Node Label Expression

You can set a default node label on a queue. The default node label will be used if no label is specified when the job is submitted.

For example, to set "x"as the default node label for queue "b1", you would add the following property in the capacity-scheduler.xml file.

```
<property>
 <name>yarn.scheduler.capacity.root.b.b1.default-node-label-expression</
name>
 <value>x</value>
</property>
```

Use Node Labels

You can use various methods to specify node labels when submitting jobs.

Procedure

- Set Node Labels when Submitting Jobs

  You can use the following methods to specify node labels when submitting jobs:

  - ApplicationSubmissionContext.setNodeLabelExpression(<node_label_expression>) -- sets the node label expression for all containers of the application.
  - ResourceRequest.setNodeLabelExpression(<node_label_expression>) -- sets the node label expression for individual resource requests. This will override the node label expression set inApplicationSubmissionContext.setNodeLabelExpression(<node_label_expression>).
  - Specify setAMContainerResourceRequest.setNodeLabelExpression in ApplicationSubmissionContext to indicate the expected node label for the ApplicationMaster container.

  You can use one of these methods to specify a node label expression, and -queue to specify a queue, when you submit YARN jobs using the distributed shell client. If the queue has a label that satisfies the label expression, it will run the job on the labeled node(s). If the label expression does not reference a label associated with the specified queue, the job will not run and an error will be returned. If no node label is specified, the job will run only on nodes without a node label, and on nodes with shareable node labels if idle resources are available.

Note:

You can only specify one node label in the .setNodeLabelExpression methods.

For example, the following commands run a simple YARN distributed shell "sleep for a long time" job. In this example we are asking for more containers than the cluster can run so we can see which node the job runs on. We are specifying that the job should run on queue "a1", which our user has permission to run jobs on. We are also using the -node_label_expression parameter to specify that the job will run on all nodes with label "x".

```
sudo su yarn
hadoop jar /usr/hdp/current/hadoop-yarn-client/hadoop-yarn-applications-
distributedshell.jar
 -shell_command "sleep 100" -jar /usr/hdp/current/hadoop-yarn-client/
hadoop-yarn-applications-distributedshell.jar
 -num_containers 30 -queue a1 -node_label_expression x
```

If we run this job on the example cluster we configured previously, containers are allocated on node-1, as this node has been assigned node label "x", and queue "a1" also has node label "x":

The following commands run the same job that we specified for node label "x", but this time we will specify queue "b1" rather than queue "a1".

```
sudo su yarn
hadoop jar /usr/hdp/current/hadoop-yarn-client/hadoop-yarn-applications-
distributedshell.jar
 -shell_command "sleep 100000" -jar /usr/hdp/current/hadoop-yarn-client/
hadoop-yarn-applications-distributedshell.jar
 -num_containers 30 -queue b1 -node_label_expression x
```

When we attempt to run this job on our example cluster, the job will fail with the following error message because label "x" is not associated with queue "b1".

```
14/11/24 13:42:21 INFO distributedshell.Client: Submitting application to
 ASM
14/11/24 13:42:21 FATAL distributedshell.Client: Error running Client
org.apache.hadoop.yarn.exceptions.InvalidResourceRequestException: Invalid
 resource request, queue=b1 doesn't
have permission to access all labels in resource request. labelExpression
 of resource request=x. Queue labels=y
```

- MapReduce Jobs and Node Labels

  Currently you cannot specify a node label when submitting a MapReduce job. However, if you submit a MapReduce job to a queue that has a default node label expression, the default node label will be applied to the MapReduce job.

  Using default node label expressions tends to constrain larger portions of the cluster, which at some point starts to become counter-productive for jobs -- such as MapReduce jobs -- that benefit from the advantages offered by distributed parallel processing.

Allocating Resources with the Capacity Scheduler

The Capacity Scheduler enables multiple users and groups to share allocated cluster resources in a predictable and timely manner.

Capacity Scheduler Overview

You can use the Capacity Scheduler to allocate shared cluster resources among users and groups.

The fundamental unit of scheduling in YARN is the queue. Each queue in the Capacity Scheduler has the following properties:

- A short queue name.
- A full queue path name.
- A list of associated child-queues and applications.
- The guaranteed capacity of the queue.
- The maximum capacity of the queue.
- A list of active users and their corresponding resource allocation limits.
- The state of the queue.
- Access control lists (ACLs) governing access to the queue.

Enable the Capacity Scheduler

You must configure the yarn.resourcemanager.scheduler.class property in yarn-site.xml to enable the Capacity Scheduler.

Procedure

- To enable the Capacity Scheduler, set the following property in the /etc/hadoop/conf/yarn-site.xml file on the ResourceManager host:

```
<property>
 <name>yarn.resourcemanager.scheduler.class</name>
 <value>
 org.apache.hadoop.yarn.server.resourcemanager.scheduler.capacity.CapacityScheduler
 </value>
</property>
```

- The settings for the Capacity Scheduler are contained in the /etc/hadoop/conf/capacity-scheduler.xml file on the ResourceManager host. The Capacity Scheduler reads this file when starting, and also when an administrator modifies the capacity- scheduler.xml file and then reloads the settings by running the following command:

```
yarn rmadmin -refreshQueues
```

  This command can only be run by cluster administrators. Administrator privileges are configured with the yarn.admin.acl property on the ResourceManager.

Set up Queues

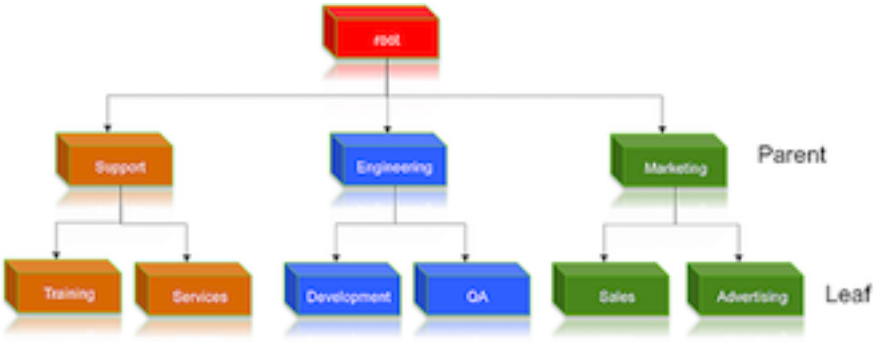
Capacity Scheduler queues can be set up in a hierarchy that reflects the database structure, resource requirements, and access restrictions required by the various organizations, groups, and users that utilize cluster resources.

About this task

The fundamental unit of scheduling in YARN is a queue. The capacity of each queue specifies the percentage of cluster resources that are available for applications submitted to the queue.

Procedure

- For example, suppose that a company has three organizations: Engineering, Support, and Marketing. The Engineering organization has two sub-teams: Development and QA. The Support organization has two sub-teams: Training and Services. And finally, the Marketing organization is divided into Sales and Advertising. The following image shows the queue hierarchy for this example:



Each child queue is tied to its parent queue with the yarn.scheduler.capacity.<queue-path>.queues configuration property in the capacity-scheduler.xml file. The top-level "support", "engineering", and "marketing" queues would be tied to the "root" queue as follows:

Property: yarn.scheduler.capacity.root.queues

Value: support,engineering,marketing

Example:

```
<property>
 <name>yarn.scheduler.capacity.root.queues</name>
 <value>support,engineering,marketing</value>
 <description>The top-level queues below root.</description>
</property>
```

Similarly, the children of the "support" queue would be defined as follows:

Property: yarn.scheduler.capacity.support.queues

Value: training,services

Example:

```
<property>
 <name>yarn.scheduler.capacity.support.queues</name>
 <value>training,services</value>
 <description>child queues under support</description>
</property>
```

The children of the "engineering" queue would be defined as follows:

Property: yarn.scheduler.capacity.engineering.queues

Value: development,qa

Example:

```
<property>
 <name>yarn.scheduler.capacity.engineering.queues</name>
 <value>development,qa</value>
 <description>child queues under engineering</description>
</property>
```

And the children of the "marketing" queue would be defined as follows:

Property: yarn.scheduler.capacity.marketing.queues

Value: sales,advertising

Example:

```
<property>
 <name>yarn.scheduler.capacity.marketing.queues</name>
 <value>sales,advertising</value>
 <description>child queues under marketing</description>
</property>
```

Hierarchical Queue Characteristics

You must consider the various characteristics of the Capacity Scheduler hierarchical queues before setting them up.

- There are two types of queues: parent queues and leaf queues.
- Parent queues enable the management of resources across organizations and sub- organizations. They can contain more parent queues or leaf queues. They do not themselves accept any application submissions directly.
- Leaf queues are the queues that live under a parent queue and accept applications. Leaf queues do not have any child queues, and therefore do not have any configuration property that ends with ".queues".
- There is a top-level parent root queue that does not belong to any organization, but instead represents the cluster itself.
- Using parent and leaf queues, administrators can specify capacity allocations for various organizations and sub-organizations.

Scheduling Among Queues

Hierarchical queues ensure that guaranteed resources are first shared among the sub-queues of an organization before any remaining free resources are shared with queues belonging to other organizations. This enables each organization to have control over the utilization of its guaranteed resources.

- At each level in the hierarchy, every parent queue keeps the list of its child queues in a sorted manner based on demand. The sorting of the queues is determined by the currently used fraction of each queue’s capacity (or the full-path queue names if the reserved capacity of any two queues is equal).
- The root queue understands how the cluster capacity needs to be distributed among the first level of parent queues and invokes scheduling on each of its child queues.
- Every parent queue applies its capacity constraints to all of its child queues.
- Leaf queues hold the list of active applications (potentially from multiple users) and schedules resources in a FIFO (first-in, first-out) manner, while at the same time adhering to capacity limits specified for individual users.

Control Access to Queues with ACLs

Use Access-control lists (ACLs) to control user and administrator to Capacity Scheduler queues.

Application submission can really only happen at the leaf queue level, but an ACL restriction set on a parent queue will be applied to all of its descendant queues.

Note: To enable ACLs, you must set the value of the yarn.acl.enable property in yarn-site.xml to true. The default value of this property is false.

In the Capacity Scheduler, ACLs are configured by granting queue access to a list of users and groups with the acl_submit_applications property. The format of the list is "user1,user2 group1,group2" -- a comma-separated list of users, followed by a space, followed by a comma-separated list of groups.

Note: The default value of acl_submit_applications for a root queue is yarn, which means that only the default yarn user can submit applications to that queue. Therefore, to provide specific users and groups with access to the queue, you must explicitly set the value of acl_submit_applications to those users and groups.

The value of acl_submit_applications can also be set to "*" (asterisk) to allow access to all users and groups, or can be set to "" (space character) to block access to all users and groups.

As mentioned previously, ACL settings on a parent queue are applied to all of its descendant queues. Therefore, if the parent queue uses the "*" (asterisk) value (or is not specified) to allow access to all users and groups, its child queues cannot restrict access. Similarly, before you can restrict access to a child queue, you must first set the parent queue to "" (space character) to block access to all users and groups.

- For example, the following properties would set the root acl_submit_applications value to "" (space character) to block access to all users and groups, and also restrict access to its child "support" queue to the users "sherlock" and "pacioli" and the members of the "cfo-group" group:

  Each child queue is tied to its parent queue with the yarn.scheduler.capacity.<queue-path>.queues configuration property in the capacity-scheduler.xml file. The top-level "support", "engineering", and "marketing" queues would be tied to the "root" queue as follows:

```
<property>
 <name>yarn.scheduler.capacity.root.acl_submit_applications</name>
 <value> </value>
</property>

<property>
 <name>yarn.scheduler.capacity.root.support.acl_submit_applications</name>
 <value>sherlock,pacioli cfo-group</value>
</property>
```

  A separate ACL can be used to control the administration of queues at various levels. Queue administrators can submit applications to the queue, kill applications in the queue, and obtain information about any application in the queue (whereas normal users are restricted from viewing all of the details of other users' applications).

  Administrator ACLs are configured with the acl_administer_queue property. ACLs for this property are inherited from the parent queue if not specified. For example, the following properties would set the root acl_administer_queue value to "" (space character) to block access to all users and groups, and also grant administrator access to its child "support" queue to the users "sherlock" and "pacioli" and the members of the "cfo- group" group:

```
<property>
 <name>yarn.scheduler.capacity.root.acl_administer_queue</name>
 <value> </value>
</property>

<property>
 <name>yarn.scheduler.capacity.root.support.acl_administer_queue</name>
 <value>sherlock,pacioli cfo-group</value>
</property>
```

Define Queue Mapping Policies

Administrators can define a default mapping policy to specify that applications submitted by users are automatically submitted to queues.

With a default mapping policy, users are not required to specify the queue name when submitting their applications. The default mapping policy can be configured to be overridden if the queue name is specified for the submitted application.

Queue mapping is defined using a comma-separated list of mapping assignments. The order of the mapping assignments list is important -- in cases where mulitple mapping assignments are used, the Capacity Scheduler processes the mapping assignments in left-to-right order to determine which mapping assignment to use first.

The Queue mapping assignment is defined using the yarn.scheduler.capacity.queue-mappings property in the capacity-scheduler.xml file. Queue mapping assignments can be defined for a user (using "u") or for a group of users (using "g"). Each mapping assignment type is described in the following sections.

Configure Queue Mapping for Users and Groups to Specific Queues

Specify that all applications submitted by a specific user are submitted to a specific queue.

Procedure

- To specify that all applications submitted by a specific user are submitted to a specific queue, use the following mapping assignment:

```
u:user1:queueA
```

This defines a mapping assignment for applications submitted by the "user1" user to be submitted to queue "queueA" by default.

To specify that all applications submitted by a specific group of users are submitted to a specific queue, use the following mapping assignment:

```
g:group1:queueB
```

This defines a mapping assignment for applications submitted by any user in the group "group1" to be submitted to queue "queueB" by default.

The Queue Mapping definition can consist of multiple assignments, in order of priority.

Consider the following example:

```
<property>
  <name>yarn.scheduler.capacity.queue-mappings</name>
  <value>u:maria:engineering,g:webadmins:weblog</value>
</property>
```

In this example there are two queue mapping assignments. The u:maria:engineering mapping will be respected first, which means all applications submitted by the user "maria" will be submitted to the "engineering" queue . The g:webadmins:weblog mapping will be processed after the first mapping -- thus, even if user "maria" belongs to the "webadmins" group, applications submitted by "maria" will still be submitted to the "engineering" queue.

Configure Queue Mapping for Users and Groups to Queues with the Same Name

Specify that all applications are submitted to the queue with the same name as a group.

Procedure

- To specify that all applications are submitted to the queue with the same name as a group, use this mapping assignment:

```
u:%user:%primary_group
```

Consider the following example configuration. On this cluster, there are two groups: "marketing" and "engineering". Each group has the following users:

In "marketing", there are 3 users: "angela", "rahul", and "dmitry".

In "engineering", there are 2 users: "maria" and "greg".

```
<property>
  <name>yarn.scheduler.capacity.queue-mappings</name>
  <value>u:%user:%primary_group</value>
</property>
```

With this queue mapping, any application submitted by members of the "marketing" group -- "angela", "rahul", or "dmitry" -- will be submitted to the "marketing" queue. Any application submitted by members of the "engineering" group -- "maria" or "greg" -- will be submitted to the "engineering" queue.

To specify that all applications are submitted to the queue with the same name as a user, use this mapping assignment:

```
u:%user:%user
```

This requires that queues are set up with the same name as the users. With this queue mapping, applications submitted by user "greg" will be submitted to the queue "greg".

Enable Override of Default Queue Mappings

You can override default queue mappings and submit applications that are specified for queues, other than those defined in the default queue mappings.

Procedure

- Override default queue mapping is disabled (set to false) by default.

```
<property>
    <name>yarn.scheduler.capacity.queue-mappings-override.enable</name>
    <value>false</value>
    <description>
      If a queue mapping is present and override is set to true, it will
 override the queue value specified
      by the user. This can be used by administrators to place jobs in
 queues
      that are different than the one specified by the user.
      The default is false - user can specify to a non-default queue.
    </description>
</property>
```

To enable queue mapping override, set the property to true in the capacity-scheduler.xml file.

Consider the following example in the case where queue mapping override has been enabled:

```
<property>
    <name>yarn.scheduler.capacity.queue-mappings</name>
    <value>u:maria:engineering,g:webadmins:weblog</value>
</property>
```

If user "maria" explicitly submits an application to the "marketing" queue, the default queue assignment of "engineering" is overridden, and the application is submitted to the "marketing" queue.

Configure Queue Mapping to use the user name from the application tag

You can configure queue mapping to use the user name from the application tag instead of the proxy user who submitted the job. For example, the runs Hive Queries submitted from HiveServer2 in the queue mapped from end user instead of hive user.

Procedure

1. Enable the application-tag-based-placement property to enable application placement based on the user ID passed using the application tags.

```
<property>
    <name>yarn.resourcemanager.application-tag-based-placement.enable</
name>
    <value>false</value>
    <description>
      Set to "true" to enable application placement based on the user ID
 passed using
```

```
      the application tags. When it is enabled, it checks for the
 userid=<userId> pattern
      and if found, the application will be placed onto the found user's
 queue, if the
      original user has the required rights on the passed user's queue.
    </description>
</property>
```

2. Add the list of whitelist users who can use application tag based placement. The applications when the submitting user is whitelisted, will be placed onto the queue defined in the yarn.scheduler.capacity.queue-mappings property defined for the user from the application tag. If there is no user defined, the submitting user will be used.

```
<property>
    <name>yarn.resourcemanager.application-tag-based-
placement.username.whitelist</name>
    <value></value>
    <description>
      Comma separated list of users who can use the application tag based
 placement, if
      "yarn.resourcemanager.application-tag-based-placement.enable" is
 enabled.
    </description>
</property>
```

Manage Cluster Capacity with Queues

You can manage your cluster capacity using queues to balance resource requirements of multiple applications from various users.

About this task

You can use the Capacity Scheduler to share cluster resources using FIFO (first-in, first-out) queues. YARN allows you to configure queues to own a fraction of the capacity of each cluster, and this specified queue capacity is fulfilled dynamically from the available nodes.

Users can submit applications to different queues at multiple levels in the queue hierarchy if the capacity is available on the nodes in the cluster. Because total cluster capacity can vary, capacity configuration values are expressed using percentages.

Procedure

- Specify the capacity property to allocate a floating-point percentage values of cluster capacity to a queue. The following properties divide the cluster resources between the Engineering, Support, and Marketing organizations in a 6:1:3 ratio (60%, 10%, and 30%).

  Property: yarn.scheduler.capacity.root.engineering.capacity

  Value: 60

  Property: yarn.scheduler.capacity.root.support.capacity

  Value: 10

  Property: yarn.scheduler.capacity.root.marketing.capacity

  Value: 30

  If you want the Engineering group to split its capacity between the Development and QA sub-teams in a 1:4 ratio. You can set the following property values:

  Property: yarn.scheduler.capacity.root.engineering.development.capacity

  Value: 20

Property: yarn.scheduler.capacity.root.engineering.qa.capacity

Value: 80

If you want the Engineering, Support, and Marketing organizations to use a specified absolute value for each resource type, you can set the following property values where the Engineering, Support, and Marketing queues are each allocated 10 GB of memory and 12 vcores, and 4 GPU cores:

Property: yarn.scheduler.capacity.root.engineering.capacity

Value: [memory=10240,vcores=12,yarn.io/gpu=4]

Property: yarn.scheduler.capacity.root.support.capacity

Value: [memory=10240,vcores=12,yarn.io/gpu=4]

Property: yarn.scheduler.capacity.root.marketing.capacity

Value: [memory=10240,vcores=12,yarn.io/gpu=4]

Note: The resource value of the parent queue is used if you do not provide a memory or a vcore value.

If you want to enable resource elasticity

- Specify the maximum capacity as a floating-point percentage value of resources allocated for a queue. You have to set the maximum capacity to be higher than or equal to the absolute capacity for each queue. Setting this value to -1 sets maximum capacity to 100%. In the following example, the maximum capacity of the Engineering queue is set as 70%.

  Property: yarn.scheduler.capacity.root.engineering.maximum-capacity

  Value: 70

Set Queue Priorities

For long-running applications and applications that required large containers, you must enable preemption for the YARN queue priorities to be properly applied.

About this task

Even with preemption enabled, there are some use cases where applications might not have access to cluster resources without setting priorities:

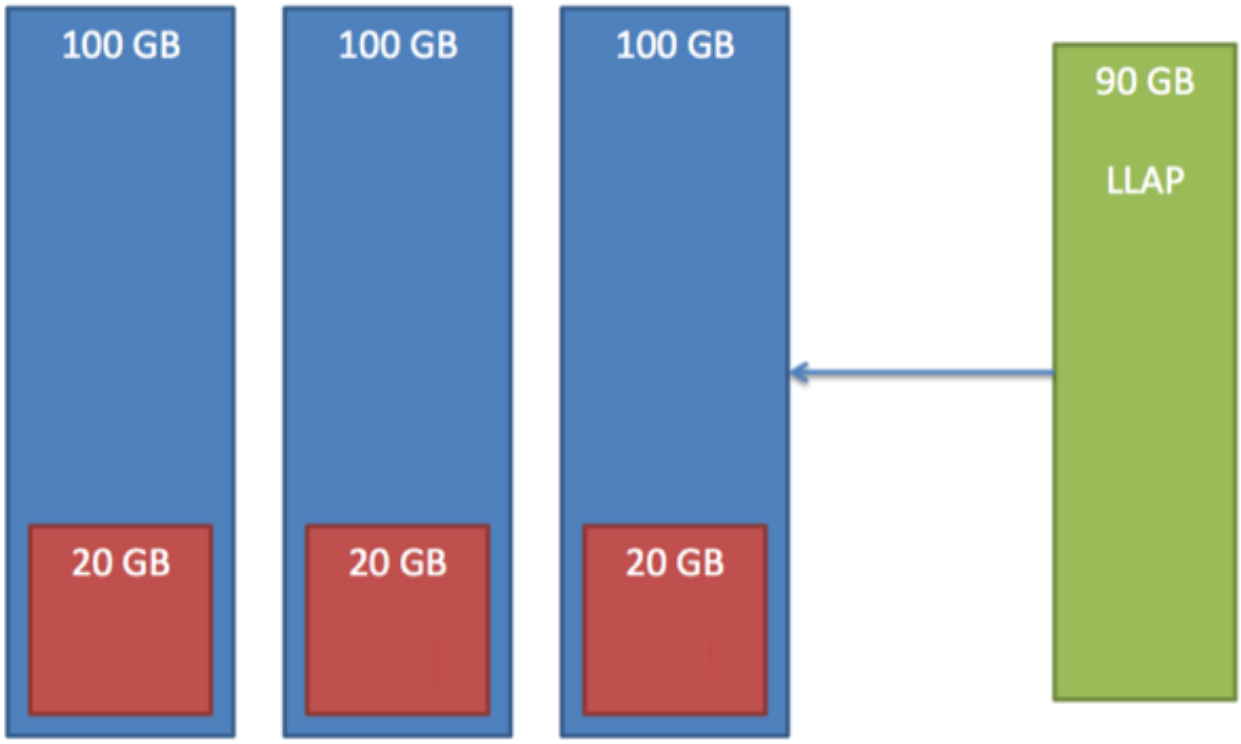
- Long-running applications – Without setting priorities, long-running applications in queues that are under capacity and with lower relative resource usage may not release cluster resources until they finish running.
- Applications that require large containers – The issue with long-running applications is exacerbated for applications that require large containers. With short-running applications, previous containers may eventually finish running and free cluster resources for applications with large containers. But with long-running services in the cluster, the large containers may never get sufficiently large resources on any nodes.
- Hive LLAP – Hive LLAP (Low-Latency Analytical Processing) enables you to run Hive queries with low-latency in near real-time. To ensure low-latency, you should set the priority of the queue used for LLAP to a higher priority, especially if your cluster includes long-running applications.

  Note:

  To set the queue used for Hive LLAP, select Hive > Config > Settings on the Ambari dashboard, then select a queue using the Interactive Query Queue drop-down. For more information, see the Hive Performance Tuning guide.

For example, the following figure shows a 3-node cluster with long-running 20 GB containers. The LLAP daemons require 90 GB of cluster resources, but preemption does not occur because the available queues are under capacity

with lower relative resource usage. With only 80 GB available on any of the nodes, LLAP must wait for the long-running applications to finish before it can access cluster resources.



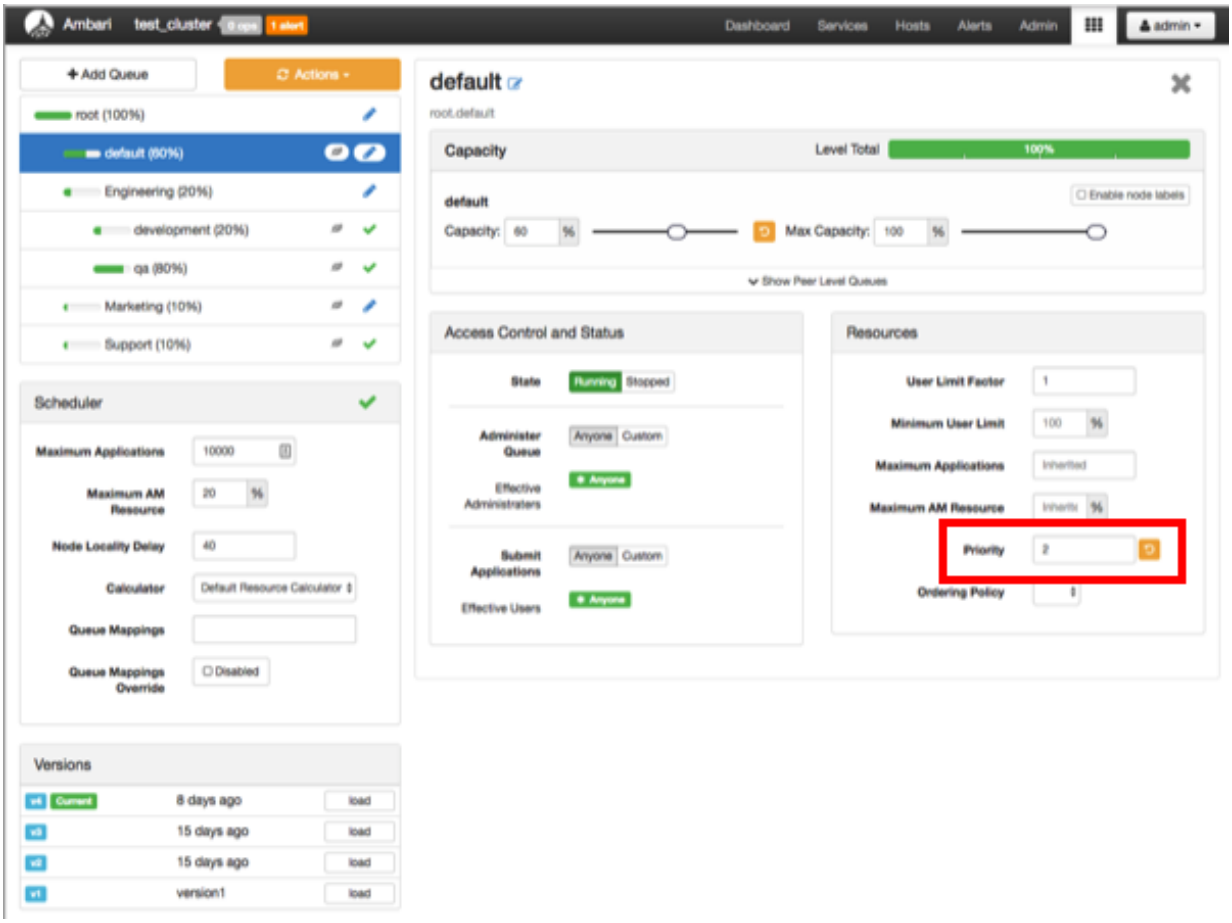
Prerequisites

Note:

In order for YARN Queue Priorities to be applied, you must enable preemption.

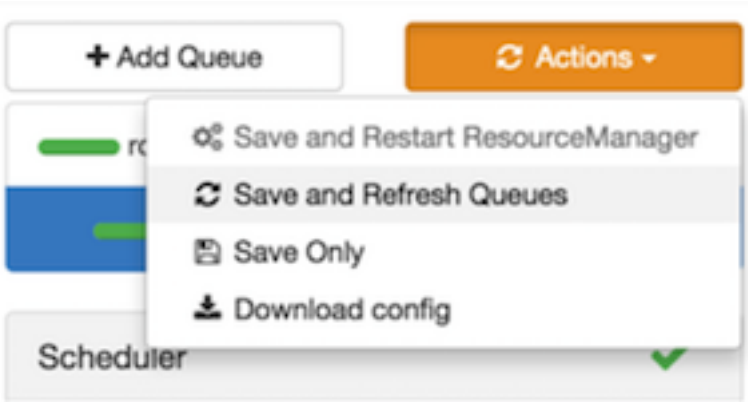
Procedure

1. On the YARN Queue Manager view, select a queue, then enter a priority in the Priority box under Resources. All queues are set to a priority of 0 by default. Higher numbers indicate higher



priority.

2. Select Actions > Save and Refresh Queues to save the priority



setting.

Resource Distribution Workflow

During scheduling, queues at any level in the hierarchy are sorted in the order of their current used capacity, and available resources are distributed among them starting with queues that are currently the most under-served.

With respect to capacities alone, the resource scheduling has the following workflow:

- The more under-served a queue is, the higher the priority it receives during resource allocation. The most under-served queue is the queue with the least ratio of used capacity as compared to the total cluster capacity.
  - The used capacity of any parent queue is defined as the aggregate sum of used capacity of all of its descendant queues, recursively.
  - The used capacity of a leaf queue is the amount of resources used by the allocated Containers of all of the applications running in that queue.
- Once it is decided to give a parent queue the currently available free resources, further scheduling is done recursively to determine which child queue gets to use the resources -- based on the previously described concept of used capacities.
- Further scheduling happens inside each leaf queue to allocate resources to applications in a FIFO order.
  - This is also dependent on locality, user level limits, and application limits.
  - Once an application within a leaf queue is chosen, scheduling also happens within the application. Applications may have different priorities of resource requests.
- To ensure elasticity, capacity that is configured but not utilized by any queue due to lack of demand is automatically assigned to the queues that are in need of resources.

Resource Distribution Workflow Example

To understand the resource distribution workflow, consider the example of a 100-node cluster, each with 10 GB of memory allocated for YARN containers, for a total cluster capacity of 1000 GB (1 TB).

According to the previously described configuration, the Engineering organization is assigned 60% of the cluster capacity, i.e., an absolute capacity of 600 GB. Similarly, the Support organization is assigned 100 GB, and the Marketing organization gets 300 GB.

Under the Engineering organization, capacity is distributed between the Development team and the QA team in a in a 1:4 ratio. So Development gets 120 GB, and 480 GB is assigned to QA.

Now consider the following timeline of events:

- Initially, the entire "engineering" queue is free with no applications running, while the "support" and "marketing" queues are utilizing their full capacities.
- Users Sid and Hitesh first submit applications to the "development" leaf queue. Their applications are elastic and can run with either all of the resources available in the cluster, or with a subset of cluster resources (depending upon the state of the resource-usage).
  - Even though the "development" queue is allocated 120 GB, Sid and Hitesh are each allowed to occupy 120 GB, for a total of 240 GB.
  - This can happen despite the fact that the "development" queue is configured to be run with a capacity of 120 GB. Capacity Scheduler allows elastic sharing of cluster resources for better utilization of available cluster resources. Since there are no other users in the "engineering" queue, Sid and Hitesh are allowed to use the available free resources.
- Next, users Jian, Zhijie and Xuan submit more applications to the "development" leaf queue. Even though each is restricted to 120 GB, the overall used capacity in the queue becomes 600 GB -- essentially taking over all of the resources allocated to the "qa" leaf queue.
- User Gupta now submits an application to the "qa" queue. With no free resources available in the cluster, his application must wait.

  - Given that the "development" queue is utilizing all of the available cluster resources, Gupta may or may not be able to immediately get back the guaranteed capacity of his "qa" queue -- depending upon whether or not preemption is enabled.
- As the applications of Sid, Hitesh, Jian, Zhijie, and Xuan finish running and resources become available, the newly available Containers will be allocated to Gupta's application.

This will continue until the cluster stabilizes at the intended 1:4 resource usage ratio for the "development" and "qa" queues.

From this example, you can see that it is possible for abusive users to submit applications continuously, and thereby lock out other queues from resource allocation until Containers finish running or get preempted. To avoid this scenario, Capacity Scheduler supports limits on the elastic growth of any queue. For example, to restrict the "development" queue from monopolizing the "engineering" queue capacity, an administrator can set a the maximum-capacity property:

Property: yarn.scheduler.capacity.root.engineering.development.maximum-capacity

Value: 40

Once this is set, users of the "development" queue can still go beyond their capacity of 120 GB, but they will not be allocated any more than 40% of the "engineering" parent queue's capacity (i.e., 40% of 600 GB = 240 GB).

The capacity and maximum-capacity properties can be used to control sharing and elasticity across the organizations and sub-organizations utilizing a YARN cluster. Administrators should balance these properties to avoid strict limits that result in a loss of utilization, and to avoid excessive cross-organization sharing.

Capacity and maximum capacity settings can be dynamically changed at run-time using yarn rmadmin -refreshQueues.

Set User Limits

Set a minimum percentage of resources allocated to each leaf queue user.

Procedure

- The minimum-user-limit-percent property can be used to set the minimum percentage of resources allocated to each leaf queue user. For example, to enable equal sharing of the "services" leaf queue capacity among five users, you would set the minimum-user-limit-percent property to 20%:

  Property: yarn.scheduler.capacity.root.support.services.minimum-user-limit-percent

  Value: 20

  This setting determines the minimum limit that any user's share of the queue capacity can shrink to. Irrespective of this limit, any user can come into the queue and take more than his or her allocated share if there are idle resources available.

  The following table shows how the queue resources are adjusted as users submit jobs to a queue with a minimum-user-limit-percent value of 20%:

yarn.scheduler.capacity.root.marketing.**minimum-user-limit-percent = 20**	
1 user submits jobs	Sole user gets 100% of queue capacity.
2 users submit jobs	Each user equally shares 50% of queue capacity.
3 users submit jobs	Each user equally shares 33.33% of queue capacity.
4 users submit jobs	Each user equally shares 25% of queue capacity.
5 users submit jobs	Each user equally shares 20% of queue capacity.
6th user submits job	6th user must wait for queue capacity to free up.

    - Queue resources are adjusted in the same manner for a single user submitting multiple jobs in succession. If no other users are requesting queue resources, the first job would receive 100% of the queue capacity. When the user submits a second job, each job receives 50% of queue capacity. When the user submits a third job, each job receives 33% of queue capacity. If a second user then submits a job, each job would receive 25% of queue capacity. When the number of jobs submitted by all users reaches a total of five, each job will receive 20% of queue capacity, and subsequent users must wait for queue capacity to free up (assuming preemption is not enabled).
    - The Capacity Scheduler also manages resources for decreasing numbers of users. As users' applications finish running, other existing users with outstanding requirements begin to reclaim that share.
    - Note that despite this sharing among users, the FIFO application scheduling order of Capacity Scheduler does not change. This guarantees that users cannot monopolize queues by submitting new applications continuously. Applications (and thus the corresponding users) that are submitted first always get a higher priority than applications that are submitted later.
- Capacity Scheduler's leaf queues can also use the user-limit-factor property to control user resource allocations. This property denotes the fraction of queue capacity that any single user can consume up to a maximum value, regardless of whether or not there are idle resources in the cluster.

  Property: yarn.scheduler.capacity.root.support.user-limit-factor

  Value: 1

  The default value of "1" means that any single user in the queue can at maximum only occupy the queue's configured capacity. This prevents users in a single queue from monopolizing resources across all queues in a cluster. Setting the value to "2" would restrict the queue's users to twice the queue's configured capacity. Setting it to a value of 0.5 would restrict any user from using resources beyond half of the queue capacity.

  These settings can also be dynamically changed at run-time using yarn rmadmin - refreshQueues.

Application Reservations

For a resource-intensive application, the Capacity Scheduler creates a reservation on a cluster node if the node's free capacity can meet the particular application's requirements. This ensures that the resources are utilized only by that particular application until the application reservation is fulfilled.

The Capacity Scheduler is responsible for matching free resources in the cluster with the resource requirements of an application. Many times, a scheduling cycle occurs such that even though there are free resources on a node, they are not sized large enough to satisfy the application waiting for a resource at the head of the queue. This typically happens with high-memory applications whose resource demand for Containers is much larger than the typical application running in the cluster. This mismatch can lead to starving these resource-intensive applications.

The Capacity Scheduler reservations feature addresses this issue as follows:

- When a node reports in with a finished Container, the Capacity Scheduler selects an appropriate queue to utilize the newly available resources based on capacity and maximum capacity settings.
- Within that selected queue, the Capacity Scheduler looks at the applications in a FIFO order along with the user limits. Once a needy application is found, the Capacity Scheduler tries to see if the requirements of that application can be met by the node's free capacity.
- If there is a size mismatch, the Capacity Scheduler immediately creates a reservation on the node for the application's required Container.
- Once a reservation is made for an application on a node, those resources are not used by the Capacity Scheduler for any other queue, application, or Container until the application reservation is fulfilled.
- The node on which a reservation is made reports back when enough Containers finish running such that the total free capacity on the node now matches the reservation size. When that happens, the Capacity Scheduler marks the reservation as fulfilled, removes it, and allocates a Container on the node.
- In some cases another node fulfills the resources required by the application, so the application no longer needs the reserved capacity on the first node. In this situation, the reservation is simply cancelled.

To prevent the number of reservations from growing in an unbounded manner, and to avoid any potential scheduling deadlocks, the Capacity Scheduler maintains only one active reservation at a time on each node.

Set Flexible Scheduling Policies

Set FIFO (First-In, First-Out) or Fair scheduling policies in Capacity Scheduler depending on your requirements.

The default ordering policy in Capacity Scheduler is FIFO (First-In, First-Out). FIFO generally works well for predicable, recurring batch jobs. but sometimes not as well for on-demand or exploratory workloads. For these types of jobs, Fair Sharing is often a better choice. Flexible scheduling policies enable you to assign FIFO or Fair ordering polices for different types of workloads on a per-queue basis.

Examples of FIFO and Fair Sharing Policies

Both FIFO (First-In, First-Out) and Fair scheduling policies work differently in batch jobs and ad hoc jobs.

Batch Example

In this example, two queues have the same resources available. One uses the FIFO ordering policy, and the other uses the Fair Sharing policy. A user submits three jobs to each queue one right after another, waiting just long enough for each job to start. The first job uses 6x the resource limit in the queue, the second 4x, and last 2x.

- In the FIFO queue, the 6x job would start and run to completion, then the 4x job would start and run to completion, and then the 2x job. They would start and finish in the order 6x, 4x, 2x.
- In the Fair queue, the 6x job would start, then the 4x job, and then the 2x job. All three would run concurrently, with each using 1/3 of the available application resources. They would typically finish in the following order: 2x, 4x, 6x.

Ad Hoc Plus Batch Example

In this example, a job using 10x the queue resources is running. After the job is halfway complete, the same user starts a second job needing 1x the queue resources.

- In the FIFO queue, the 10x job will run until it no longer uses all queue resources (map phase complete, for example), and then the 1x job will start.
- In the Fair queue, the 1x job will start, run, and complete as soon as possible – picking up resources from the 10x job by attrition.

Configure Queue Ordering Policies

You can configure the property for queue ordering policies to fifo or fair in capacity-scheduler.xml.

Procedure

- Ordering policies are configured in capacity-scheduler.xml. To specify ordering policies on a per-queue basis, set the following property to fifo or fair. The default setting is fifo.

```
<property>
  <name>yarn.scheduler.capacity.<queue-path>.ordering-policy</name>
  <value>fair</value>
</property>
```

You can use the following property to enable size-based weighting of resource allocation. When this property is set to true, queue resources are assigned to individual applications based on their size, rather than providing an equal share of queue resources to all applications regardless of size. The default setting is false.

```
<property>
<name>yarn.scheduler.capacity.<queue-path>
.ordering-policy.fair.enable-size-based-weight</name>
<value>true</value>
</property>
```

Best Practices for Ordering Policies

You must consider factors related to applications and resource availability in queues while configuring ordering policies.

- Ordering policies are configured on a per-queue basis, with the default ordering policy set to FIFO. Fairness is usually best for on-demand, interactive, or exploratory workloads, while FIFO can be more efficient for predictable, recurring batch processing. You should segregate these different types of workloads into queues configured with the appropriate ordering policy.
- In queues supporting both large and small applications, large applications can potentially "starve" (not receive sufficient resources). To avoid this scenario, use different queues for large and small jobs, or use size-based weighting to reduce the natural tendency of the ordering logic to favor smaller applications.
- Use the yarn.scheduler.capacity.<queue-path>.maximum-am-resource-percent property to restrict the number of concurrent applications running in the queue to avoid a scenario in which too many applications are running simultaneously. Limits on each queue are directly proportional to their queue capacities and user limits. This property is specified as a float, for example: 0.5 = 50%. The default setting is 10%. This property can be set for all queues using the yarn.scheduler.capacity.maximum-am-resource-percent property, and can also be overridden on a per-queue basis using the yarn.scheduler.capacity.<queue-path>.maximum-am-resource-percent property.

Start and Stop Queues

Queues in YARN can be in two states: RUNNING or STOPPED. A RUNNING state indicates that a queue can accept application submissions, and a STOPPED queue does not accept application submissions. The default state of any configured queue is RUNNING.

About this task

In Capacity Scheduler, parent queues, leaf queues, and the root queue can all be stopped. For an application to be accepted at any leaf queue, all the queues in the hierarchy all the way up to the root queue must be running. This means that if a parent queue is stopped, all of the descendant queues in that hierarchy are inactive, even if their own state is RUNNING.

Procedure

- The following example sets the value of the state property of the "support" queue to RUNNING:

  Property: yarn.scheduler.capacity.root.support.state

  Value: RUNNING

  Administrators can use the ability to stop and drain applications in a queue for a number of reasons, such as when decommissioning a queue and migrating its users to other queues. Administrators can stop queues at run-time, so that while current applications run to completion, no new applications are admitted. Existing applications can continue until they finish running, and thus the queue can be drained gracefully without any end-user impact.

  Administrators can also restart the stopped queues by modifying the state configuration property and then refreshing the queue using yarn rmadmin -refreshQueues.

Set Application Limits

To avoid system-thrash due to an unmanageable load -- caused either by malicious users, or by accident -- the Capacity Scheduler enables you to place a static, configurable limit on the total number of concurrently active (both running and pending) applications at any one time.

Procedure

- The maximum-applications configuration property is used to set the limit, with a default value of 10,000.

  Property: yarn.scheduler.capacity.maximum-applications

Value: 10000

The limit for running applications in any specific queue is a fraction of this total limit, proportional to its capacity. This is a hard limit, which means that once this limit is reached for a queue, any new applications to that queue will be rejected, and clients will have to wait and retry later. This limit can be explicitly overridden on a per-queue basis with the following configuration property:

Property: yarn.scheduler.capacity.<queue-path>.maximum-applications

Value: absolute-capacity * yarn.scheduler.capacity.maximum-applications

There is another resource limit that can be used to set a maximum percentage of cluster resources allocated specifically to ApplicationMasters. The maximum-am-resource-percent property has a default value of 10%, and exists to avoid cross-application deadlocks where significant resources in the cluster are occupied entirely by the Containers running ApplicationMasters. This property also indirectly controls the number of concurrent running applications in the cluster, with each queue limited to a number of running applications proportional to its capacity.

Property: yarn.scheduler.capacity.maximum-am-resource-percent

Value: 0.1

As with maximum-applications, this limit can also be overridden on a per-queue basis:

Property: yarn.scheduler.capacity.<queue-path>.maximum-am-resource-percent

Value: 0.1

All of these limits ensure that no single application, user, or queue can cause catastrophic failure, or monopolize the cluster and cause excessive degradation of cluster performance.

Enable Preemption

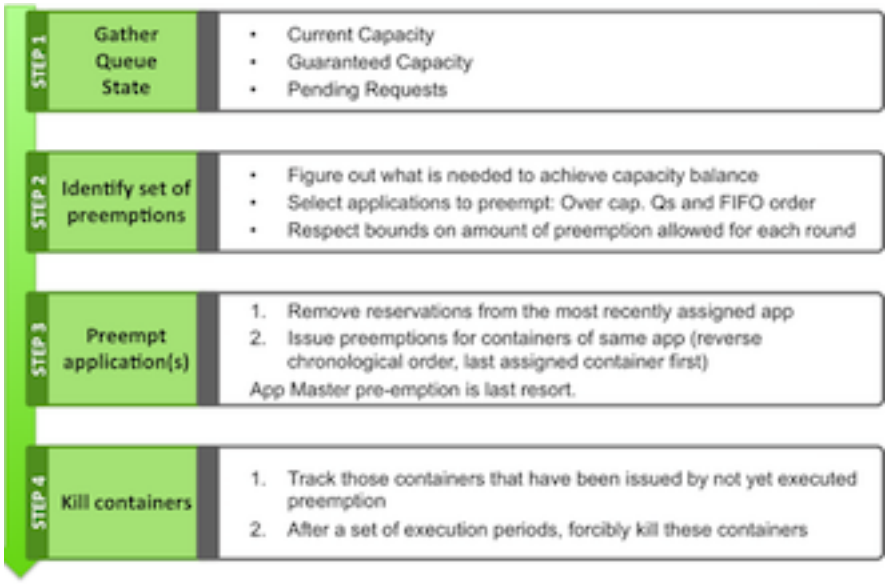
Capacity Scheduler Preemption allows higher-priority applications to preempt lower-priority applications.

A scenario can occur in which a queue has a guaranteed level of cluster resources, but must wait to run applications because other queues are utilizing all of the available resources. If Preemption is enabled, higher-priority applications do not have to wait because lower priority applications have taken up the available capacity. With Preemption enabled, under-served queues can begin to claim their allocated cluster resources almost immediately, without having to wait for other queues' applications to finish running.

Preemption Workflow

Preemption is governed by a set of capacity monitor policies, which must be enabled by setting the yarn.resourcemanager.scheduler.monitor.enable property to true. These capacity monitor policies apply Preemption in configurable intervals based on defined capacity allocations, and in as graceful a manner as possible. Containers are only killed as a last resort.

The following image demonstrates the Preemption workflow:



Related Tasks

Configure Preemption

Related Information

Better SLAS Via Resource Preemption in the YARN Capacity Scheduler

Configure Preemption

Configure various properties in yarn-site.xml to set application preemption in the Capacity Scheduler.

Procedure

- The following properties in the /etc/hadoop/conf/yarn-site.xml file on the ResourceManager host are used to enable and configure Preemption.
  - Property: yarn.resourcemanager.scheduler.monitor.enable

    Value: true

    Description: Setting this property to "true" enables Preemption. It enables a set of periodic monitors that affect the Capacity Scheduler. This default value for this property is "false" (disabled).
  - Property: yarn.resourcemanager.scheduler.monitor.policies

    Value:

    org.apache.hadoop.yarn.server.resourcemanager.monitor.capacity.ProportionalCapacityPreemptionPolicy

    Description: The list of SchedulingEditPolicy classes that interact with the scheduler. The only policy currently available for preemption is the “ProportionalCapacityPreemptionPolicy”.
  - Property: yarn.resourcemanager.monitor.capacity.preemption.monitoring_interval

    Value: 3000

    Description: The time in milliseconds between invocations of this policy. Setting this value to a longer time interval will cause the Capacity Monitor to run less frequently.
  - Property: yarn.resourcemanager.monitor.capacity.preemption.max_wait_before_kill

    Value: 15000

    Description: The time in milliseconds between requesting a preemption from an application and killing the container. Setting this to a higher value will give applications more time to respond to preemption requests and gracefully release Containers.
  - Property: yarn.resourcemanager.monitor.capacity.preemption.total_preemption_per_round

Value: 0.1

Description: The maximum percentage of resources preempted in a single round. You can use this value to restrict the pace at which Containers are reclaimed from the cluster. After computing the total desired preemption, the policy scales it back to this limit. This should be set to (memory-of-one-NodeManager)/(total-cluster-memory). For example, if one NodeManager has 32 GB, and the total cluster resource is 100 GB, the total_preemption_per_round should set to 32/100 = 0.32. The default value is 0.1 (10%).

- Property: yarn.resourcemanager.monitor.capacity.preemption.natural_termination_factor

  Value: 1.0

  Description: Similar to total_preemption_per_round, you can apply this factor to slow down resource preemption after the preemption target is computed for each queue (for example, “give me 5 GB back from queue-A”). For example, if 5 GB is needed back, in the first cycle preemption takes back 1 GB (20% of 5GB), 0.8 GB (20% of the remaining 4 GB) in the next, 0.64 GB (20% of the remaining 3.2 GB) next, and so on. You can increase this value to speed up resource reclamation. The recommended value for this parameter is 1.0, meaning that 100% of the target capacity is preempted in a cycle.
- Property: yarn.resourcemanager.monitor.capacity.preemption.max_ignored_over_capacity

  Value: 0.1

  Description: The maximum amount of resources above the target capacity ignored for preemption. When configured to a value x >= 0, Resource Manager will wait till a queue uses resources amounting to configured_capacity * (1 + x) before starting to preempt containers from it. By default, it is 0.1, which means Resource Manager will start preemption for a queue only when it goes 10% above its guaranteed capacity. This avoids resource-joggling and aggressive preemption.

Related Concepts

Preemption Workflow

Related Information

Better SLAS Via Resource Preemption in the YARN Capacity Scheduler

Enable Priority Scheduling

You can use Priority Scheduling to run YARN applications at higher priority, regardless of other applications that are already running in the cluster. YARN allocates more resources to applications running at a higher priority over those running at a lower priority. Priority Scheduling enables you to set an application's priority both at the time of submission and dynamically at run time.

About this task

Priority Scheduling works only with the FIFO (first-in, first-out) ordering policy. FIFO is the default Capacity Scheduler ordering policy.

Procedure

1. Set the cluster maximum and leaf-queue level priorities.
   - Cluster maximum priority: Set the following property in the yarn-site.xml file to define the maximum priority for an application in the cluster:

     ```
     yarn.cluster.max-application-priority
     ```

     Any application submitted with a priority greater than this setting has its priority reset to the yarn.cluster.max-application-priority value.

- Leaf queue-level priority: Set the following property in the capacity-scheduler.xml file to define the default application priority in a leaf queue:

```
yarn.scheduler.capacity.root.<leaf-queue-path>.default-application-
priority
```

  The default application priority is used for any application submitted without a specified priority.

2. Use either the yarn application -appID command-line option or the Cluster Application REST API to set the priority for already running applications.
   - yarn application -appID <appID> -updatePriority <priority>
   - Cluster Application Priority API

Related Tasks

Configure ACLs for Application Priorities

Related Information

YARN User Commands

Configure ACLs for Application Priorities

Configure Priority ACLs to ensure that only select users can submit applications with a specified priority to a queue. You must configure these Priority ACLs at the leaf queue-level.

About this task

If ACLs are already configured for user access to a leaf queue, then the Priority ACLs for the queue can include only those users with access to that queue.

Procedure

- Set the following property in the capacity-scheduler.xml file to set the Priority ACLs:

```
<property>
    <name>yarn.scheduler.capacity.<leaf-queue-
path>.acl_application_max_priority</name>
    <value>[user={username} group={groupname} max_priority={priority}
 default_priority={priority}]
    </value>
    <description>
      The ACL of users who can submit applications with configured
 priority.
    </description>
</property>
```

  The following example shows how you can configure Priority ACLs for a user maria and for the users of a group hadoop:

```
yarn.scheduler.capacity.root.queue1.acl_application_max_priority=[user=maria
 group=hadoop max_priority=7 default_priority=4]
```

  The user maria and the users of the hadoop group can submit applications with a maximum priority of 7.

Related Tasks

Enable Priority Scheduling

Enable Intra-Queue Preemption

Intra-queue preemption helps in effective distribution of resources within a queue based on configured user limits or application priorities.

Intra-queue preemption prevents resource imbalances in a queue by preventing the following situations from occurring:

- Lower-priority applications consuming all the available resources on the queue and, thereby, starving higher-priority applications of resources.
- A few users consuming the entire queue capacity and, thereby, depriving other users from submitting higher-priority applications. This situation could occur in spite of all the users being eligible for the queue's resources based on configured limits.

Properties for Configuring Intra-Queue Preemption

Intra-queue preemption is enabled by default for YARN queues. In addition, you can configure the order of intra-queue preemption either by application priorities or configured user limits.

You can configure the values of the following properties in yarn-site.xml for intra-queue preemption:

Property	Description
yarn.resourcemanager.monitor.capacity.preemption.intra-queue-preemption.enabled	Specifies whether intra-queue preemption is enabled or disabled for queues. The default value is true.
yarn.resourcemanager.monitor.capacity.preemption.intra-queue-preemption.preemption-order-policy	Specifies the order in which a queue can preempt resources. Based on your requirements, you can configure this property to either of the following values: • userlimit-first, to initiate intra-queue preemption based on configured user limits. This is the default value. • priority-first, to initiate intra-queue preemption based on application priorities.

Intra-Queue Preemption Based on Application Priorities

Enabling preemption on a queue depending on application priorities ensures that higher-priority applications can preempt resources from lower-priority applications when required.

Example of resource consumption on a queue without preemption

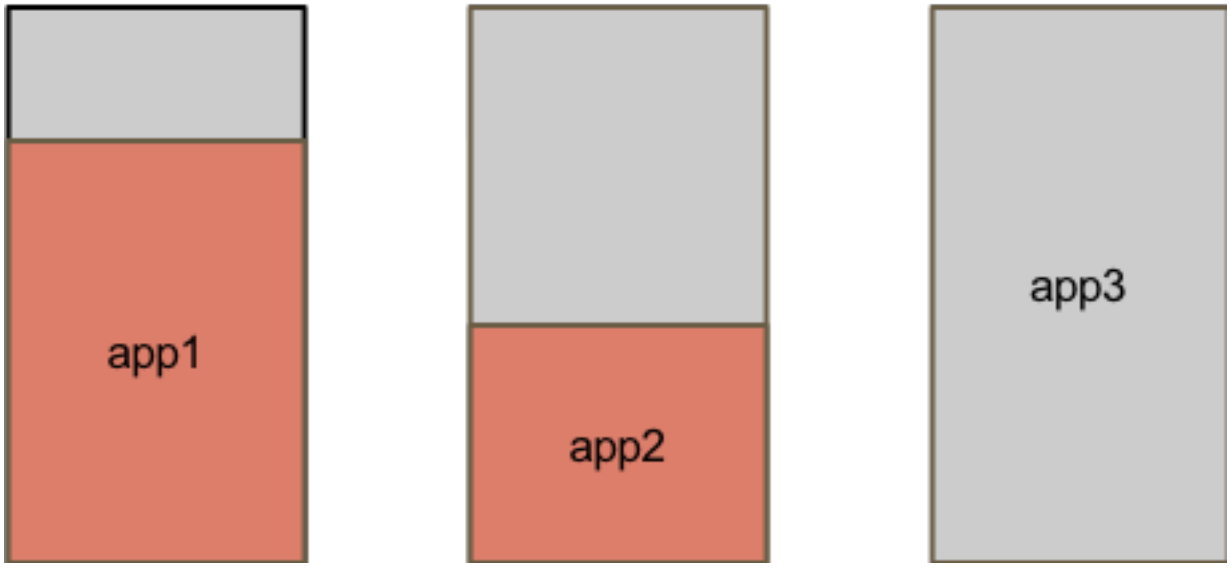
Consider a queue qA with root.qA.capacity configured at 70%. Consider applications submitted in the following order:

1. A user submits an application app1 with priority p1. Because no other application is running on the queue, app1 uses a majority of the resources available on the queue.
2. Shortly after app1 starts running, the user submits another application app2 with the same priority as app1. In this situation, app2 uses the resources that remain on the queue.
3. The user submits a third application app3 with a higher priority p3.

If preemption is not enabled on the queue, the lower priority applications app1 and app2 consume all of the queue's available capacity leaving the higher priority application app3 starved of resources.

The following table explains the resource distribution between the three applications:

Application	Priority	Consumed Resources	Pending Resources
app1	p1	50	20
app2	p1	20	20
app3	p3	0	80



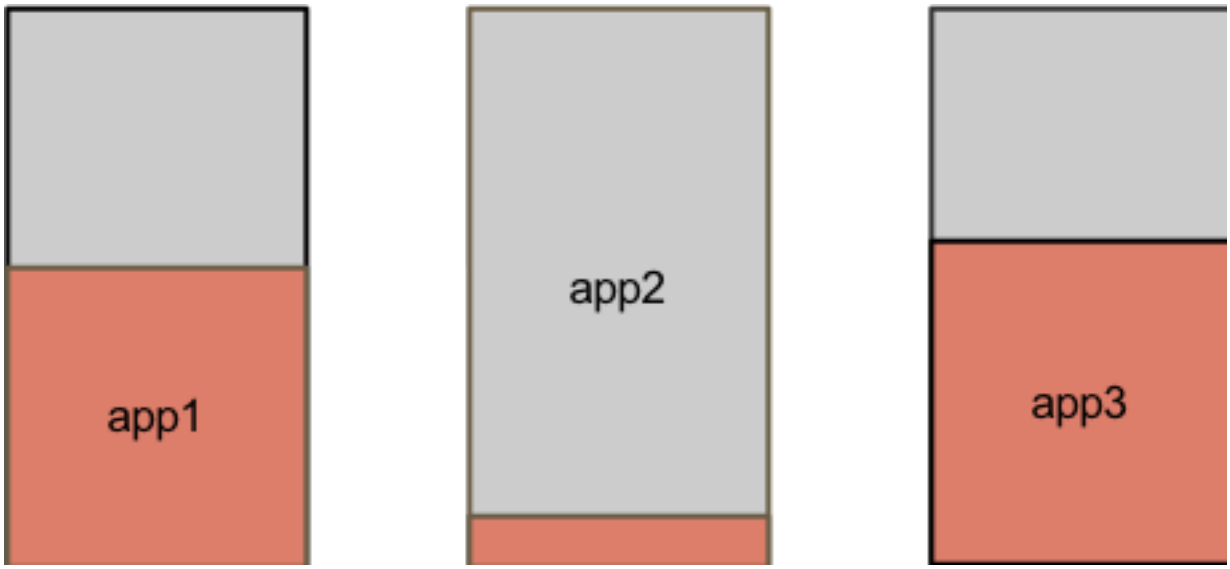
Example of resource consumption on a queue with preemption

Consider the same queue and applications with priorities as the previous example.

If preemption based on application priority is enabled on the queue, resources are preempted from app1 and app2 for the higher-priority app3 to run.

The following table explains the resource distribution between the three applications when preemption is enabled:

Application	Priority	Preempted Resources	Consumed Resources	Pending Resources
app1	p1	16	34	36
app2	p1	19	1	39
app3	p3	0	35	45



Intra-Queue Preemption based on User Limits

Enabling preemption on a queue based on user limits ensures that resources are uniformly distributed among all users who submit applications to the particular queue.

Example of resource consumption on a queue without preemption

Consider a queue qA with root.qA.capacity configured at 100% and minimum-user-limit-percent configured at 33%. This implies that the first three users submitting applications to the queue can each use a minimum of 33% of the queue's resources. If the three users are already consuming the queue's resources as specified, then any additional user must wait for resources to be allocated before submitting applications.

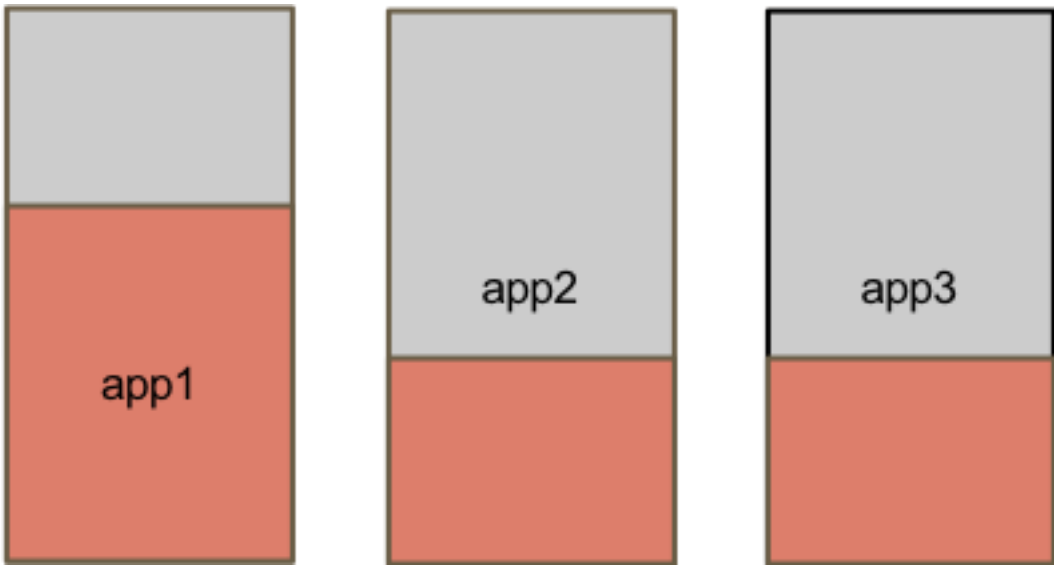
Consider applications submitted in the following order:

1. The user u1 submits an application app1 with priority p1. Because no other application is running on the queue, app1 uses a majority of the resources available on the queue.
2. Shortly after app1 starts running, users u2 and u3 respectively submit applications app2 and app3 around the same time with priority p1. In this situation, app2 and app3 use the resources that remain on the queue.

   If preemption is not enabled on the queue, app2 and app3 cannot consume their share of resources on the queue in spite of having the same priority as app1.

The following table explains the resource distribution between the three applications:

Application	Users	Consumed Resources	Pending Resources
app1	u1	60	30
app2	u2	20	25
app3	u3	20	25



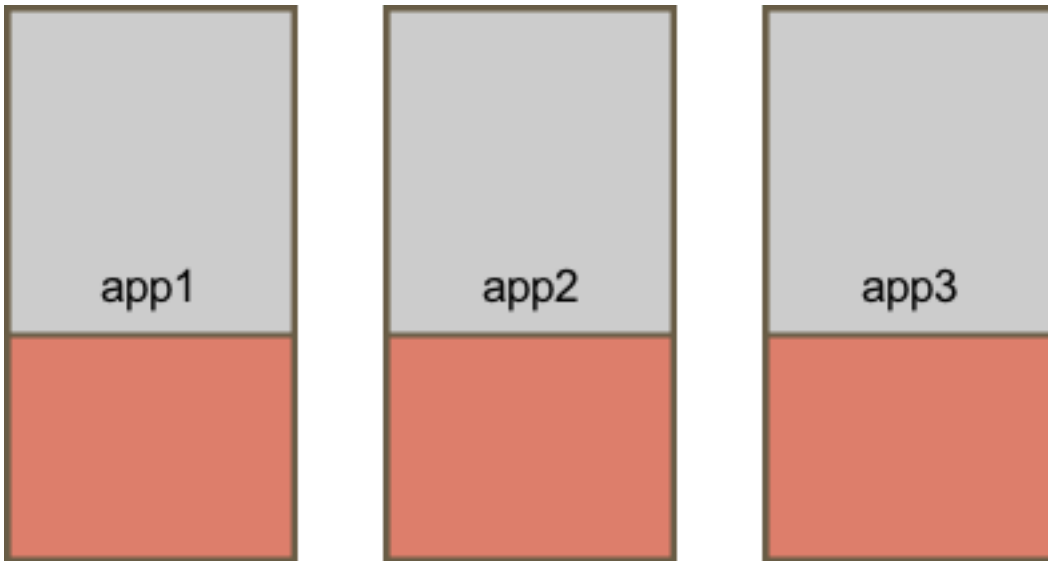
Example of resource consumption on a queue with preemption

Consider the same queue and applications with priorities as the previous example.

If preemption based on user limits is enabled on the queue, resources are preempted from app1 for app2 and app3 to run.

The following table explains the resource distribution between the three applications when preemption is enabled:

Application	Priority	Preempted Resources	Consumed Resources	Pending Resources
app1	u1	26	34	56
app2	u2	0	33	12
app3	u3	0	33	12



Monitoring Clusters using YARN Web User Interface

Using the YARN Web User Interface, you can monitor clusters, queues, applications, services, flow activities, and nodes. You can also view the configuration details and check the logs for various applications and services.

Accessing YARN Web User Interface

Access the YARN Web User Interface to monitor clusters, queues, applications, services, and flow activities.

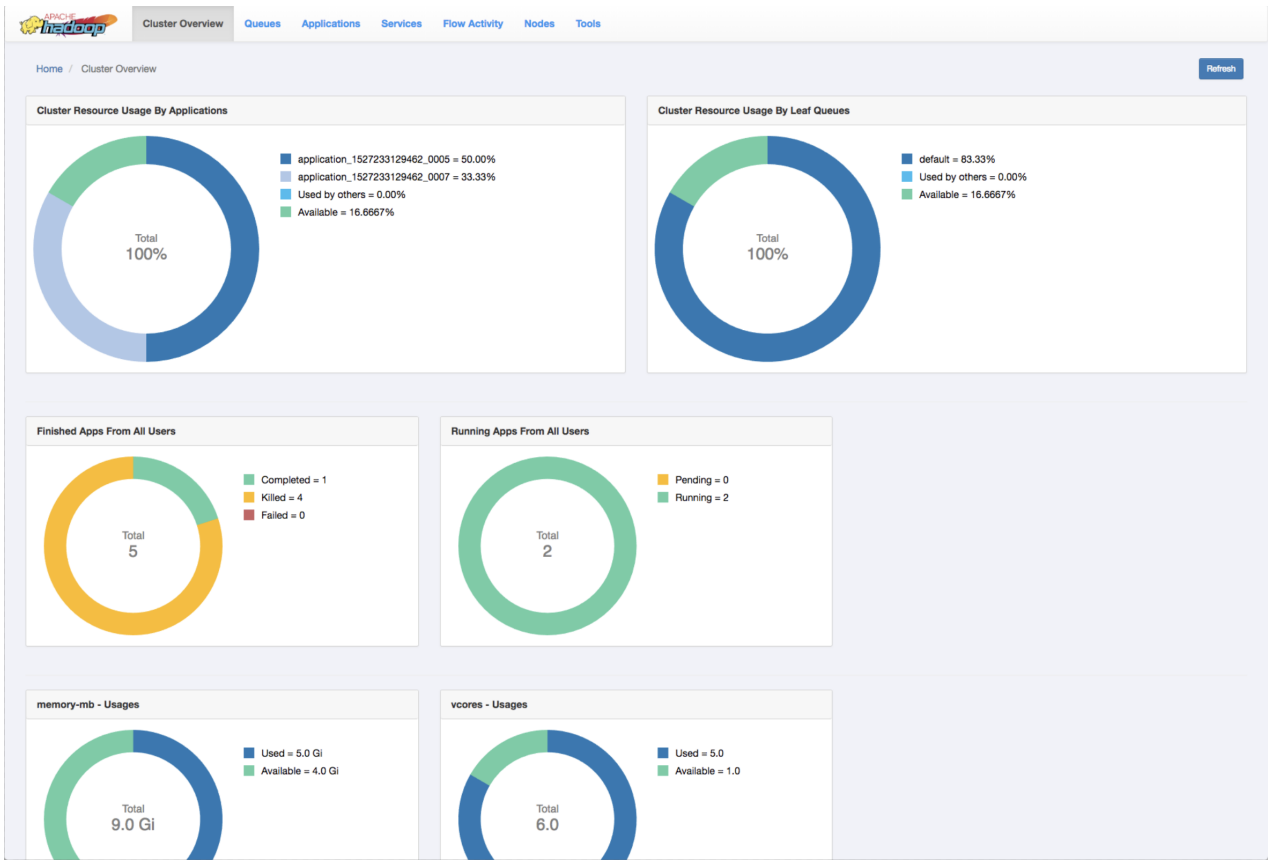
Procedure

1. In Ambari Web, browse to **Services** > **> YARN** > **> Summary**.
2. Under **Quick Links**, click **ResourceManager UI**.

   The YARN Web User Interface loads on a separate browser window.

Monitoring Clusters

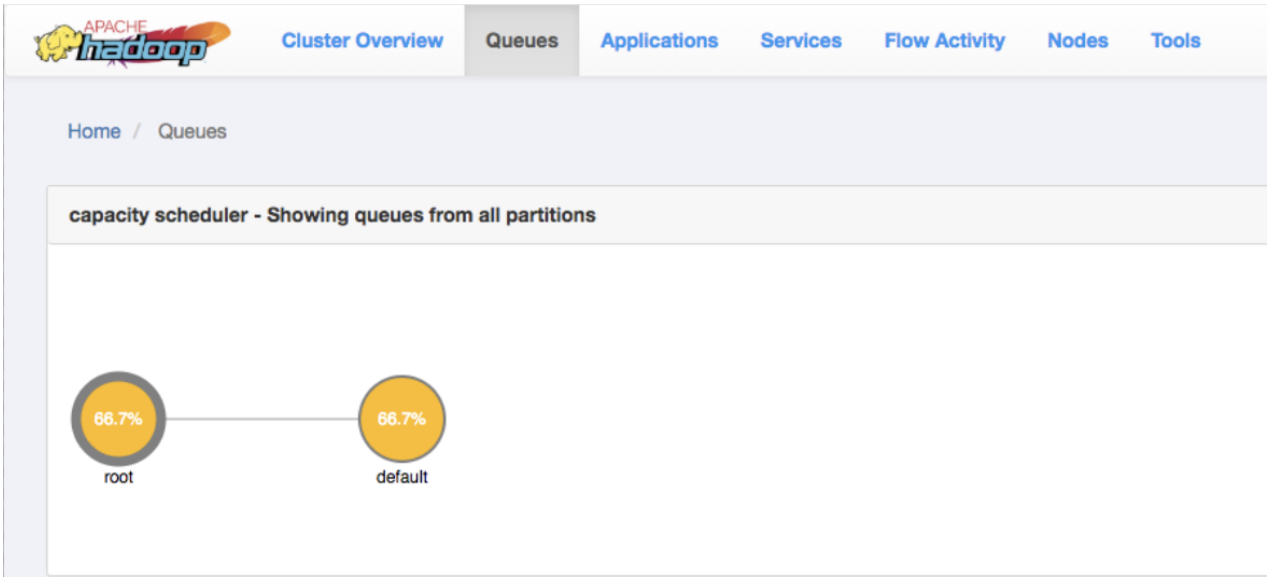
The **Cluster Overview** page shows cluster resource usage by applications and queues, information about finished and running applications, and usage of memory and vCores in the cluster.


Cluster Overview
Queues
Applications
Services
Flow Activity
Nodes
Tools
Home / Cluster Overview
Refresh
Cluster Resource Usage By Applications
application_1527233129462_0005 = 50.00%
application_1527233129462_0007 = 33.33%
Used by others = 0.00%
Available = 16.6667%
Total
100%
Cluster Resource Usage By Leaf Queues
default = 83.33%
Used by others = 0.00%
Available = 16.6667%
Total
100%
Finished Apps From All Users
Completed = 1
Killed = 4
Failed = 0
Total
5
Running Apps From All Users
Pending = 0
Running = 2
Total
2
memory-mb - Usages
Used = 5.0 Gi
Available = 4.0 Gi
Total
9.0 Gi
vcores - Usages
Used = 5.0
Available = 1.0
Total
6.0

Cluster Resource Usage by Applications	Displays the percentage of cluster resources in use by applications and the percentage available for usage.
Cluster Resource Usage by Leaf Queues	Displays the percentage of cluster resources in use by leaf queues and the percentage available for usage.
Finished Apps From All Users	Displays the number of completed, killed, and failed applications.
Monitor Running Apps	Displays the number of pending and running applications.
memory-mb - Usages	Displays the amount of used and available memory.
vcores - Usages	Displays the number of used and available virtual cores.
Monitor Node Managers	Displays the status of the Node Managers under the following categories: • Active • Unhealthy • Decommissioning • Decommissioned

Monitoring Queues

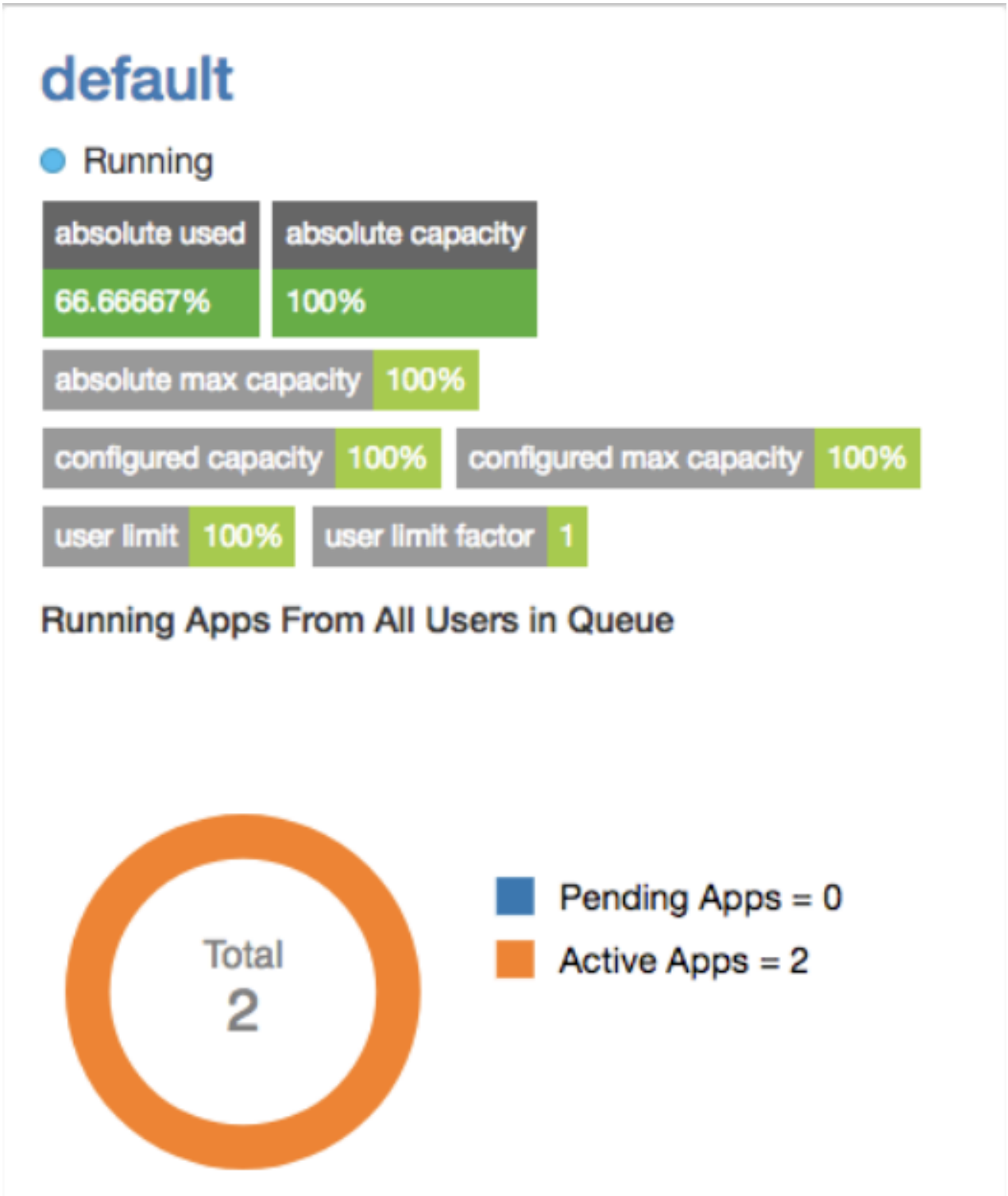
The **Queues** page displays details of YARN queues. You can either view queues from all the partitions or filter to view queues of a partition.



View Queue Details

In the capacity scheduler view, click the circle that represents a particular queue. The right column of the page gets updated with details of that queue.

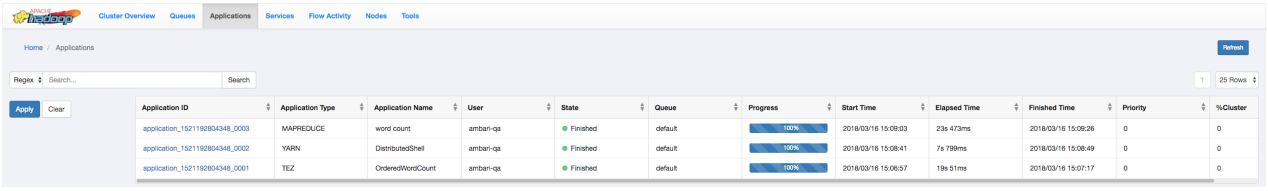
The following example shows the details of a queue:



You can double-click the queue to view its details on a separate page. You can also view details of any application submitted to that queue by clicking its corresponding Application ID.

Monitoring Applications

You can search for applications and view their details using the YARN Web User Interface.



The **Applications** page displays details of the YARN applications in a tabular form.

- Application ID: The identifier for the application.
- Application Type: Specifies the application type for Mapreduce, YARN, TEZ, or other applications.
- Application Name: Provides the name of the application
- User: The name of the user who is the owner of the application.
- State: The running state of the application.
- Queue: Specifies the name of the queue to which the application belongs.
- Progress: The progress of the application in a percentage display.
- Start Time: The time when an application run started.
- Elapsed Time: The time taken for the application run.
- Finished Time: The time of completion of the application run.
- Priority: The priority of running the application.
- %Cluster: The percentage of cluster resources used by the application run.

Searching an application

The **Applications** page displays the list of YARN applications in a tabular form. You can apply search filters on this list to show only those applications that match the search criteria.

About this task

You can specify the search criteria either as regular expressions or SQL statements.

Procedure

1. On the **Applications** page, select either **Regex** or **SQL** from the drop-down list depending on the type of search you want to perform.
2. In the Search box, specify the search criteria.

Search Criteria	Description
Regex	Lists the applications whose details match the search criterion specified as a regular expression. For example, if you want to view application runs longer than an hour, mention the regular expression ^hso that the YARN UI shows only those applications that mention the **Elapsed Time** value in terms of hours, minutes, and seconds.
SQL	Lists the applications whose details match the search criterion specified as a SQL statement. For example, if you want to view all the applications submitted by the user yarn, specify "User"='yarn' as the search criterion.

3. Click **Search** to view details of the applications matching the search criteria.

Note:

- Apart from specifying search criteria to filter the list of applications, you can also select or clear the **State** and **Queue** check-boxes to view a specific set of applications depending on your requirements.
- You can sort the application entries in ascending or descending order by clicking the corresponding arrow next to any column header in the table.

Related Concepts

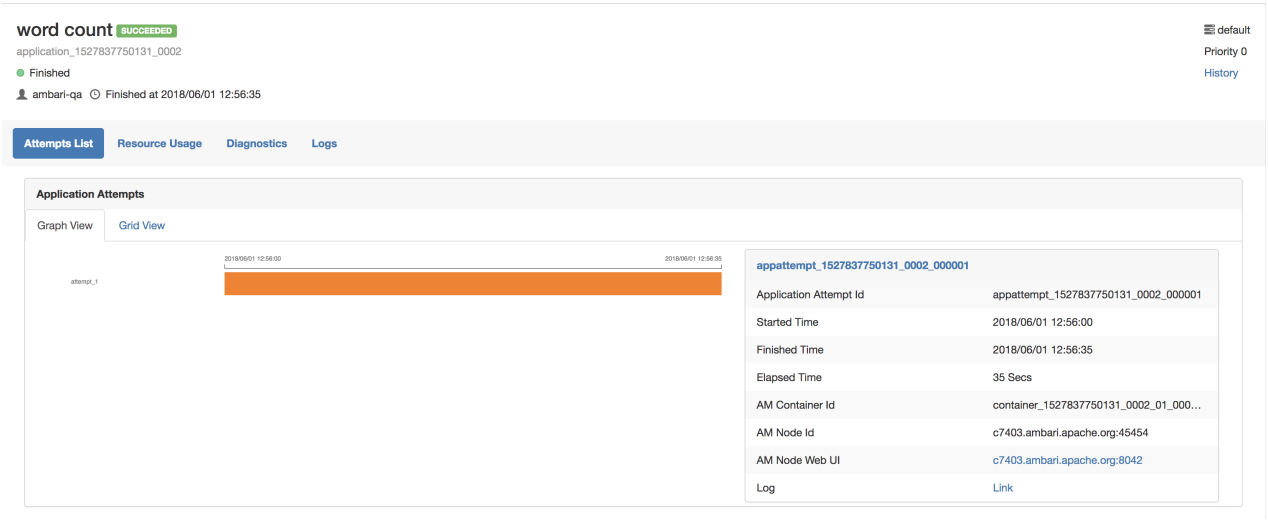
Managing and Monitoring Services

Viewing application details

Clicking a YARN application on the **Applications** page displays its details.

You can view the following details for the selected application:

- Application Attempts
- Resource Usage
- Diagnostics
- Logs


word count SUCCEEDED
application_1527837750131_0002
Finished
ambari-qa Finished at 2018/06/01 12:56:35
default
Priority 0
History
Attempts List
Resource Usage
Diagnostics
Logs
Application Attempts
Graph View
Grid View
2018/06/01 12:56:00
2018/06/01 12:56:35
attempt_1
appattempt_1527837750131_0002_000001
Application Attempt Id
appattempt_1527837750131_0002_000001
Started Time
2018/06/01 12:56:00
Finished Time
2018/06/01 12:56:35
Elapsed Time
35 Secs
AM Container Id
container_1527837750131_0002_01_000...
AM Node Id
c7403.ambari.apache.org:45454
AM Node Web UI
c7403.ambari.apache.org:8042
Log
Link

Application Attempts

You can view the attempts in Graph View and Grid View.

Graph View	A graph displays the start time and finish time of the attempt. You can also find the details of the attempt such as application attempt ID, started time, finished time, elapsed time, AM Container ID, and AM Node ID in the form of a table. You can access the Node UI using the AM Node Web UI. You can also view the log by clicking on the log link.
Grid View	A table displays the details of the application attempts. You can find the details of the attempt such as application attempt ID, started time, finished time, elapsed time, AM Container ID, and AM Node ID. You can access the Node UI using the AM Node Web UI. You can also view the log by clicking on the log link.

Resource Usage

This tab displays the resources used by the application attempts.

Diagnostics

Use this tab to view the diagnostics details of the application attempts. You can view any outstanding resource requests, and scheduling information.

Logs

Use this tab to view logs specific to containers. Select an attempt from the dropdown list and select the specific container to view the desired logs.

Managing and Monitoring Services

You can create new YARN services and view the list of existing services on the **Services** page.

The **Services** page lists the services in a tabular form. Clicking an individual service displays its details. You can filter the list of services the same way as you filter applications on the **Applications** page.

Related Tasks

Searching an application

Create New Services

The YARN Web User Interface enables you to define new services. You can either create standard services by providing their details or custom services by using JSON files containing the required definitions.

Create a standard service

You can create a standard service as a template and deploy it based on your requirements.

Procedure

1. On the **Services** page of the YARN Web User Interface, click **New Service**.
2. In the **User name for service** field, specify the name of the user who launches the service.
3. Enter the service definition details.
   - **Service Name**: Enter a unique name for the application.
   - **Queue Name**: Enter the name of the YARN queue to which this application should belong.

- **Service Lifetime**: Life time of the application from the time it is in STARTED state till the time it is automatically destroyed by YARN. If you want to have unlimited life time, do not enter any value.
- **Service Components**: Enter the details of the service components such as component name, CPU required, memory, number of containers, artifact ID, and launch command. If it is an application like HBase, the components can be a simple role like master or RegionServer. For a complex application, the components can be other nested applications with their own details.
- **Service Configurations**: Set of configuration properties that can be ingested into the application components through environments, files, and custom pluggable helper docker containers. You can upload files of several formats such as properties files, JSON files, XML files, YAML files, and template files.
- **File Configurations**: Set of file configurations that needs to be created and made available as a volume in an application component container. You can upload JSON file configurations to add to the service.

4. Click **Save**.
5. Specify a name for the new service and click **Add**.

   The newly created service is added to the list of saved templates.

   Note: Click **Reset** if you do not want to save your changes and specify the service details again.

6. Select the service and click **Deploy** to deploy it.

Create a custom service

You can define a service in JSON format and save it as a template.

Procedure

1. On the **Services** page of the YARN Web User Interface, click **New Service**.
2. Click the **Custom** tab.
3. In the **User name for service** field, specify the name of the user who launches the service.
4. In the **Service Definition** field, specify the service definition in JSON format.

   The following example shows the sleeper service template definition.

```
{
  "name": "sleeper-service",
  "version": "1.0.0",
  "components" :
  [
    {
      "name": "sleeper",
      "number_of_containers": 2,
      "launch_command": "sleep 900000",
      "resource": {
        "cpus": 1,
        "memory": "256"
      }
    }
  ]
}
```

5. Click **Save**.
6. Specify a name for the new service and click **Add**.

   The newly created service is added to the list of saved templates.

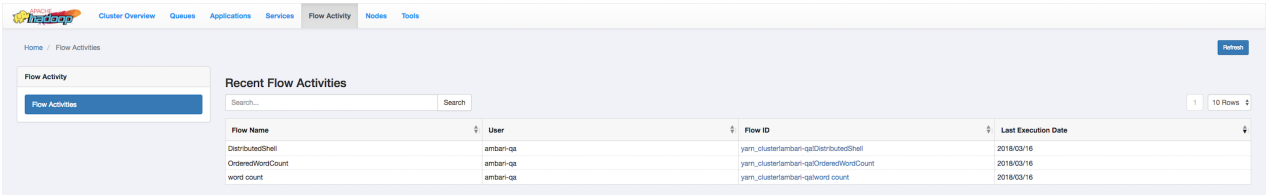
   Note: Click **Reset** if you do not want to save your changes and specify the service details again.

7. Select the service and click **Deploy** to deploy it.

Monitoring Flow Activity

You can view information about application flows from the **Flow Activities** page.

The page displays information about the recent flow activities in a tabular form. Clicking a Flow ID displays details of the corresponding flow on a separate **Flow Information** page.



The **Flow Information** page displays the following details for a selected flow activity:

Flow Info	Displays details of the selected flow such as its name, the associated user, the ID, the time when the first run started, the time when the last run finished, and the last execution date of the flow.
Flow Runs	Provides details of all the flow runs such as Run ID, Run Duration, CPU VCores, memory used, creation time, and end time. You can also view graphs of metrics such as Flow Run Vs Run Duration, Flow Run Vs CPU VCores, and Flow Run Vs Memory Used. Clicking a Run ID displays its specific Flow Run Information page. On this page, you can view all the details of the flow info and list of applications associated with this flow. Clicking an Application ID displays details of the application. The **Flow Runs** page also displays general metrics used by the particular flow run with the selected Flow Run ID. You can search for specific metrics by entering the details in the search bar.

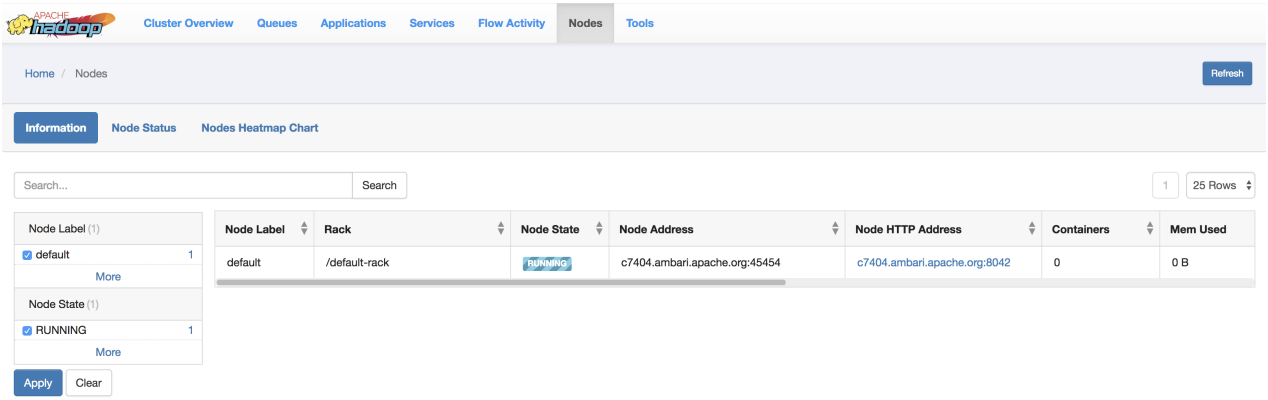
Monitoring Nodes

The **Nodes** page on the YARN Web User Interface enables you to view information about the cluster nodes on which the NodeManagers are running.

The **Nodes** page displays details under the following headers: Information, Node Status and Nodes Heatmap Chart.

Information

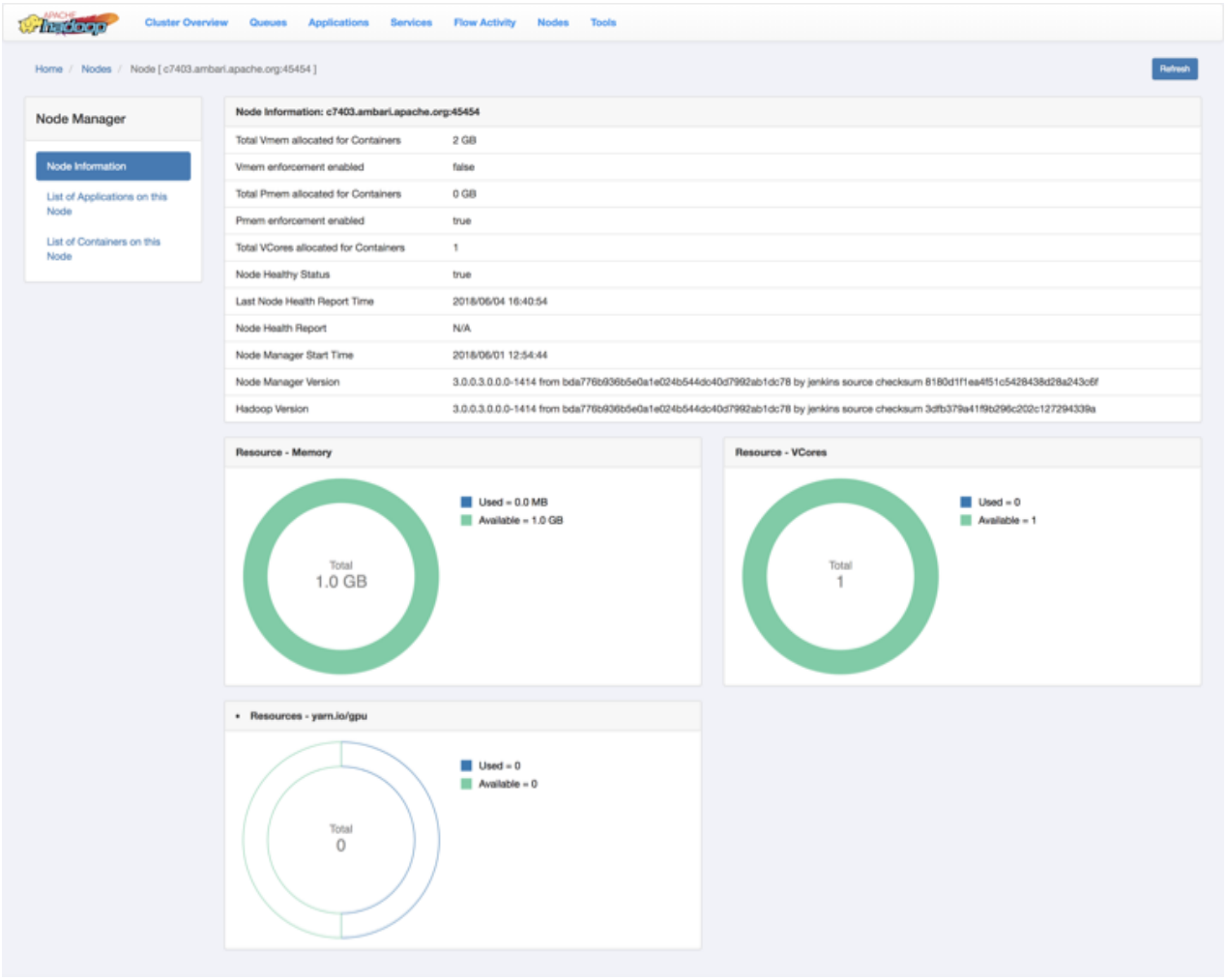
The **Information** tab displays the node details as shown in the following figure:


Cluster Overview
Queues
Applications
Services
Flow Activity
Nodes
Tools
Home / Nodes
Refresh
Information
Node Status
Nodes Heatmap Chart
Search...
Search
1
25 Rows
Node Label (1)
default 1
More
Node State (1)
RUNNING 1
More
Apply
Clear
Node Label
Rack
Node State
Node Address
Node HTTP Address
Containers
Mem Used
default
/default-rack
RUNNING
c7404.ambari.apache.org:45454
c7404.ambari.apache.org:8042
0
0 B

You can sort through any of the columns to view the details of the required nodes. You can also search to find the specific node labels from the entire list.

Node Manager Page

To view the Node Manager page of any node label, click the corresponding Node HTTP address. The **Node Manager** page displays details of a node as shown in the following figure:



You can also view the resource usage in the following categories in a pie-chart representation:

1. Memory
2. VCores
3. yarn-io/GPU

Node Status

This tab displays the node managers in a pictorial representation. It displays details such as the number of active nodes, number of unhealthy nodes, decommissioning nodes, and the number of decommissioned node managers.

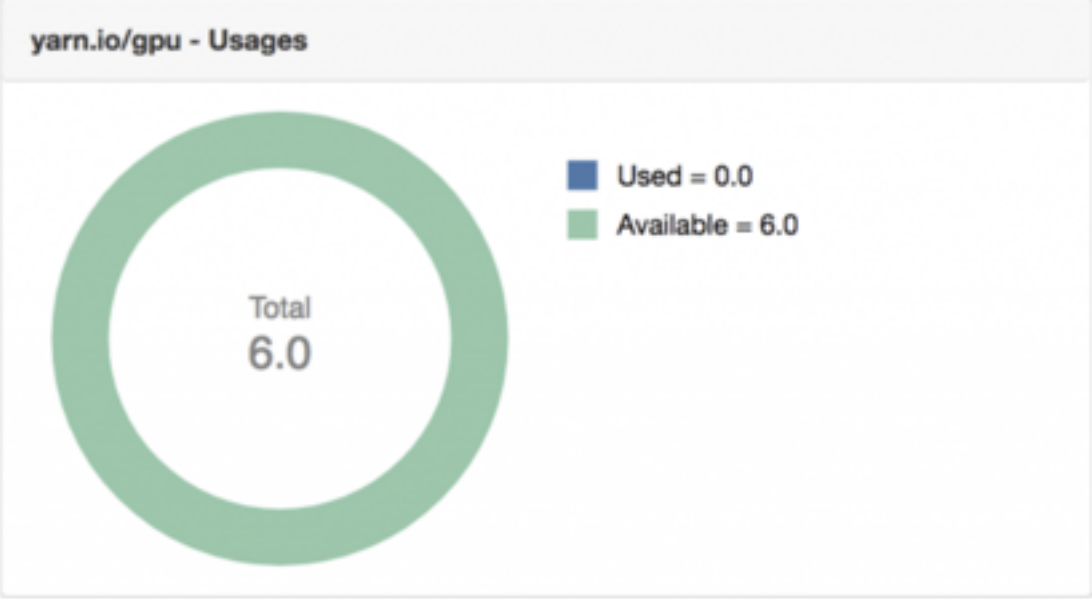
Nodes Heatmap Chart

This tab graphically displays nodes on the basis of their usage of memory. You can enter host or rack details in the search bar to filter nodes.

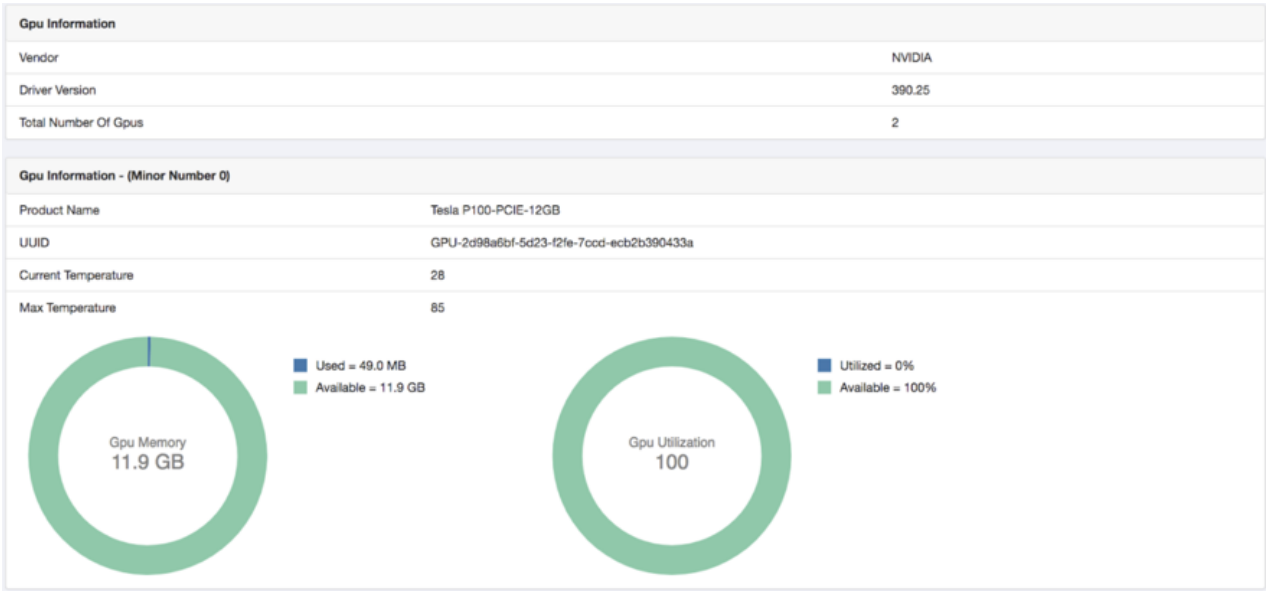
Monitoring GPU metrics

You can view the total used and available GPU resources using the YARN Web User Interface.

On the ResourceManager page, you can see the available GPU resources across the cluster.



On each NodeManager page, you can see the the per-GPU device usage and metrics.



Tools

You can view the YARN configuration and YARN Daemon logs on the **Tools** page.

YARN Configuration

You can see the values of the properties defined in the following configuration files:

1. Core Configuration: Details of properties defined in the core-default.xml file.
2. YARN Configuration: Details of properties defined in the yarn-default.xml file.
3. MapReduce Configuration: Details of the properties defined in the mapred-default.xml file.

YARN Daemon Logs

You can view the list of log files. Click on a log file to view its contents in another tab of your browser.

Fault Tolerance

Configuring the ResourceManager or the NodeManager for work-preserving restarts makes YARN more fault-tolerant as the work of running applications is preserved in the event of a ResourceManager or a NodeManager restart.

Configure Work-preserving Restart

Configure YARN to preserve the work of running applications in the event of a ResourceManager or NodeManager restart.

Work-preserving ResourceManager and NodeManager restart ensures that node restart or fail-over is completely transparent to end-users, with minimal impact to running applications.

Configure the ResourceManager for Work-preserving Restart

Configure YARN to preserve the work of running applications in the event of a ResourceManager restart.

Configure ResourceManager Work-preserving Restart.

Work-preserving ResourceManager restart ensures that applications continuously function during a ResourceManager restart with minimal impact to end-users. The overall concept is that the ResourceManager preserves application queue state in a pluggable state store, and reloads that state on restart. While the ResourceManager is down, ApplicationMasters and NodeManagers continuously poll the ResourceManager until it restarts. When the ResourceManager comes back online, the ApplicationMasters and NodeManagers re-register with the newly started ResourceManger. When the ResourceManager restarts, it also recovers container information by absorbing the container statuses sent from all NodeManagers. Thus, no work will be lost due to a ResourceManager crash-reboot event.

To configure work-preserving restart for the ResourceManager, set the following properties in the yarn-site.xml file.

Property:

yarn.resourcemanager.recovery.enabled

Value:

true

Description:

Enables ResourceManager restart. The default value is false. If this configuration property is set to true, running applications will resume when the ResourceManager is restarted.

Example:

```
<property>
    <name>yarn.resourcemanager.recovery.enabled</name>
    <value>true</value>
 </property>
```

Property:

yarn.resourcemanager.store.class

Value:

<specified_state_store>

Description:

Specifies the state-store used to store application and application-attempt state and other credential information to enable restart. The available state-store implementations are:

- org.apache.hadoop.yarn.server.resourcemanager.recovery.FileSystemRMStateStore – a state-store implementation persisting state to a file system such as HDFS. This is the default value.
- org.apache.hadoop.yarn.server.resourcemanager.recovery.LeveldbRMStateStore – a LevelDB-based state-store implementation.
- org.apache.hadoop.yarn.server.resourcemanager.recovery.ZKRMStateStore – a ZooKeeper-based state-store implementation.

Example:

```
<property>
    <name>yarn.resourcemanager.store.class</name>

 <value>org.apache.hadoop.yarn.server.resourcemanager.recovery.FileSystemRMStateStore</
value>
 </property>
```

FileSystemRMStateStore Configuration

The following properties apply only if org.apache.hadoop.yarn.server.resourcemanager.recovery.FileSystemRMStateStore has been specified as the state-store in the yarn.resourcemanager.store.class property.

Property:

yarn.resourcemanager.fs.state-store.uri

Value:

<hadoop.tmp.dir>/yarn/system/rmstore

Description:

The URI pointing to the location of the file system path where the RM state will be stored (e.g. hdfs://localhost:9000/rmstore). The default value is <hadoop.tmp.dir>/yarn/system/rmstore.

Example:

```
<property>
    <name>yarn.resourcemanager.fs.state-store.uri</name>
    <value>hdfs://localhost:9000/rmstore</value>
 </property
```

Property:

yarn.resourcemanager.fs.state-store.retry-policy-spec

Value:

2000, 500

Description:

The Hadoop FileSystem client retry policy specification. Hadoop FileSystem client retry is always enabled. This is specified in pairs of sleep-time and number-of-retries i.e. (t0, n0), (t1, n1), ..., the first n0 retries sleep t0 milliseconds on average, the following n1 retries sleep t1 milliseconds on average, and so on. The default value is (2000, 500).

Example:

```
<property>
    <name>yarn.resourcemanager.fs.state-store.retry-policy-spec</name>
    <value>2000, 500</value>
```

```
 </property
```

LeveldbRMStateStore Configuration

The following properties apply only if org.apache.hadoop.yarn.server.resourcemanager.recovery.LeveldbRMStateStore has been specified as the state-store in the yarn.resourcemanager.store.class property.

Property:

yarn.resourcemanager.leveldb-state-store.path

Value:

<hadoop.tmp.dir>/yarn/system/rmstore

Description:

The local path where the RM state will be stored.

Example:

```
<property>
    <name>yarn.resourcemanager.leveldb-state-store.path</name>
    <value><hadoop.tmp.dir>/yarn/system/rmstore</value>
 </property
```

ZKRMStateStore Configuration

The following properties apply only if org.apache.hadoop.yarn.server.resourcemanager.recovery.ZKRMStateStore has been specified as the state-store in the yarn.resourcemanager.store.class property.

Property:

yarn.resourcemanager.zk-address

Value:

<host>:<port>

Description:

A comma-separated list of <host>:<port> pairs, each corresponding to a server in a ZooKeeper cluster where the ResourceManager state will be stored.

Example:

```
<property>
    <name>yarn.resourcemanager.zk-address</name>
    <value>127.0.0.1:2181</value>
 </property
```

Property:

yarn.resourcemanager.zk-state-store.parent-path

Value:

/rmstore

Description:

The full path of the root znode where RM state will be stored. The default value is /rmstore.

Example:

```
<property>
```

```
    <name>yarn.resourcemanager.zk-state-store.parent-path</name>
    <value>/rmstore</value>
 </property
```

Property:

yarn.resourcemanager.zk-num-retries

Value:

500

Description:

The number of times the ZooKeeper-client running inside the ZKRMStateStore tries to connect to ZooKeeper in case of connection timeouts. The default value is 500.

Example:

```
<property>
    <name>yarn.resourcemanager.zk-num-retries</name>
    <value>500</value>
 </property
```

Property:

yarn.resourcemanager.zk-retry-interval-ms

Value:

2000

Description:

The interval in milliseconds between retries when connecting to a ZooKeeper server. The default value is 2 seconds.

Example:

```
<property>
    <name>yarn.resourcemanager.zk-retry-interval-ms</name>
    <value>2000</value>
 </property
```

Property:

yarn.resourcemanager.zk-timeout-ms

Value:

10000

Description:

The ZooKeeper session timeout in milliseconds. This configuration is used by the ZooKeeper server to determine when the session expires. Session expiration happens when the server does not hear from the client (i.e. no heartbeat) within the session timeout period specified by this property. The default value is 10 seconds.

Example:

```
<property>
    <name>yarn.resourcemanager.zk-timeout-ms</name>
    <value>10000</value>
 </property
```

Property:

yarn.resourcemanager.zk-acl

Value:

world:anyone:rwcda

Description:

The ACLs to be used for setting permissions on ZooKeeper znodes. The default value isworld:anyone:rwcda.

Example

```
<property>
    <name>yarn.resourcemanager.zk-acl</name>
    <value>world:anyone:rwcda</value>
</property>
```

Configure NodeManagers for Work-preserving Restart

Configure YARN to preserve the work of running applications in the event of a NodeManager restart.

About this task

NodeManager work-preserving enables a NodeManager to be restarted without losing the active containers running on the node. At a high level, the NodeManager stores any necessary state to a local state store as it processes container management requests. When the NodeManager restarts, it recovers by first loading the state for various subsystems, and then lets those subsystems perform recovery using the loaded state.

Procedure

- To configure work-preserving restart for NodeManagers, set the following properties in the yarn-site.xml file on all NodeManagers in the cluster.

  Property:

  yarn.nodemanager.recovery.enabled

  Value:

  true

  Description:

  Enables the NodeManager to recover after a restart.

  Example:

  ```
  <property
      <name>yarn.nodemanager.recovery.enabled</name>
      <value>true</value>
   </property>
  ```

  Property:

  yarn.nodemanager.recovery.dir

  Value:

  <yarn_log_dir_prefix>/nodemanager/recovery-state

  Description:

  The local file system directory in which the NodeManager will store state information when recovery is enabled.

  Example:

  ```
  <property>
      <name>yarn.nodemanager.recovery.dir</name>
  ```

```
    <value><yarn_log_dir_prefix>/nodemanager/recovery-state</value>
 </property>
```

You should also confirm that the yarn.nodemanager.address port is set to a non-zero value, e.g. 45454:

```
<property>
    <name>yarn.nodemanager.address</name>
    <value>0.0.0.0:45454</value>
 </property>
```