Hortonworks Data Platform

YARN Resource Management

(Sep 4, 2015)

docs.hortonworks.com

Hortonworks Data Platform: YARN Resource Management

The Hortonworks Data Platform, powered by Apache Hadoop, is a massively scalable and 100% open source platform for storing, processing and analyzing large volumes of data. It is designed to deal with data from many sources and formats in a very quick, easy and cost-effective manner. The Hortonworks Data Platform consists of the essential set of Apache Hadoop projects including MapReduce, Hadoop Distributed File System (HDFS), HCatalog, Pig, Hive, HBase, ZooKeeper and Ambari. Hortonworks is the major contributor of code and patches to many of these projects. These projects have been integrated and tested as part of the Hortonworks Data Platform release process and installation and configuration tools have also been included.

Unlike other providers of platforms built using Apache Hadoop, Hortonworks contributes 100% of our code back to the Apache Software Foundation. The Hortonworks Data Platform is Apache-licensed and completely open source. We sell only expert technical support, training and partner-enablement services. **All of our technology is, and will remain, free and open source.**

For more information on Hortonworks technology, visit the Hortonworks Data Platform page. For more information on Hortonworks services, visit either the Support or Training page. Feel free to contact us directly to discuss your specific needs.

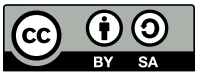


Table of Contents

1. Capacity Scheduler

This chapter describes how to 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. The following sections will describe how to configure these properties, and how Capacity Scheduler uses these properties to make various scheduling decisions.

1.1. Enabling Capacity Scheduler

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
```

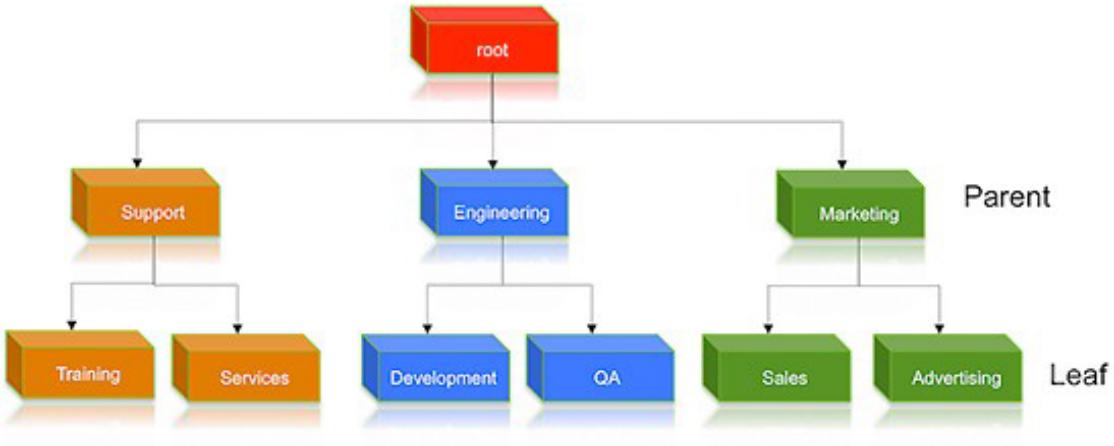
This command can only be run by cluster administrators. Administrator privileges are configured with the `yarn.admin.acl` property on the ResourceManager.

1.2. Setting up Queues

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. 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.

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 shoes 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>
```

1.3. Hierarchical Queue Characteristics

- 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.

1.4. 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.

1.5. Controlling Access to Queues with ACLs

Access-control lists (ACLs) can be used to restrict user and administrator access to 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.

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.

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:

```
<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>
```

1.6. Managing Cluster Capacity with Queues

The Capacity Scheduler is designed to allow organizations to share compute clusters using the very familiar notion of FIFO (first-in, first-out) queues. YARN does not assign entire nodes to queues. Queues own a fraction of the capacity of the cluster, and this specified q ueue capacity can be fulfilled from any number of nodes in a dynamic fashion.

Scheduling is the process of matching resource requirements – of multiple applications from various users, and submitted to different queues at multiple levels in the queue hierarchy – with the free capacity available on the nodes in the cluster. Because total cluster capacity can vary, capacity configuration values are expressed as percents.

The `capacity` property can be used by administrators to allocate a percentage of cluster capacity to a queue. The following properties would 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`

Now suppose that the Engineering group decides to split its capacity between the Development and QA sub-teams in a 1:4 ratio. That would be implemented by setting the following properties:

Property:`yarn.scheduler.capacity.root.engineering.development.capacity`

Value: `20`

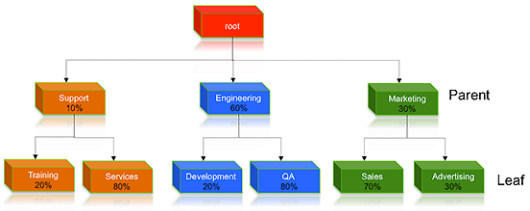
Property: `yarn.scheduler.capacity.root.engineering.qa.capacity`

Value: `80`

Note

The sum of capacities at any level in the hierarchy must equal 100%. Also, the capacity of an individual queue at any level in the hierarchy must be 1% or more (you cannot set a capacity to a value of 0).

The following image illustrates this cluster capacity configuration:



1.7. 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.

1.8. Resource Distribution Workflow Example

Suppose our cluster has 100 nodes, 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`.

1.9. Setting User Limits

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 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.

- 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`.

1.10. Application Reservations

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 utilized 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.

1.11. Starting and Stopping 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.

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.

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` as previously described.

1.12. Setting 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. The maximum-applications configuration property is used to set this 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`

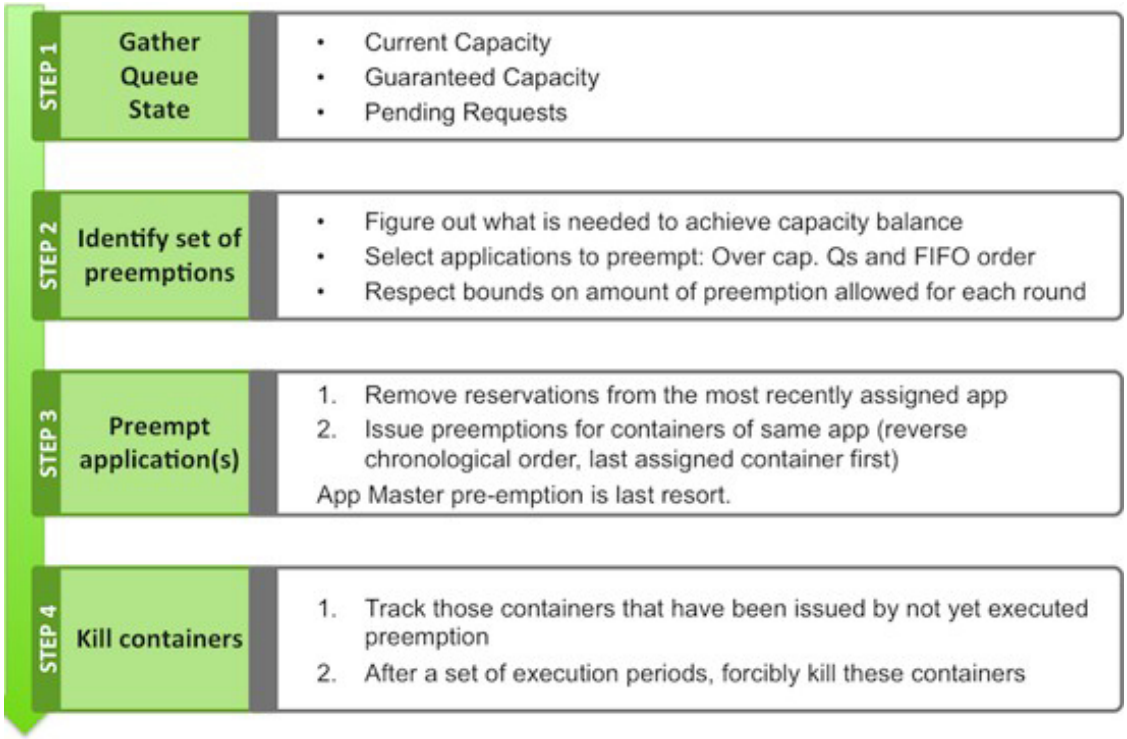
All of these limits ensure that no single application, user, or queue can cause catastrophic failure, or monopolize the cluster and cause excessive degradation of cluster performance.

1.13. Preemption

As mentioned previously, 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.

1.14. 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:



1.15. Preemption Configuration

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`

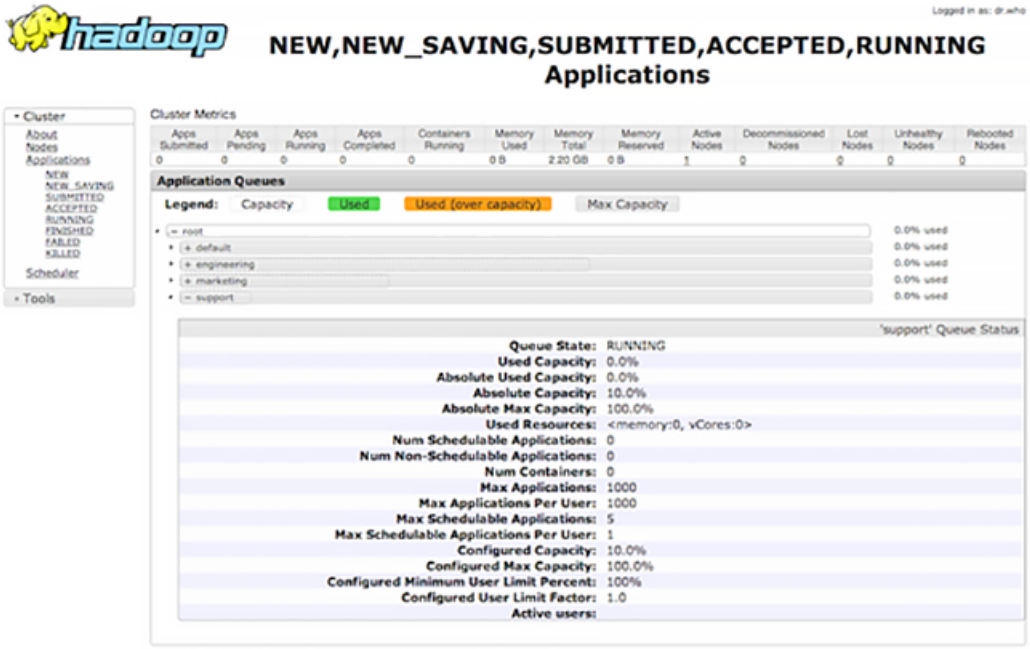
  Value: `0.1`

  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.

1.16. Scheduler User Interface

You can use the Scheduler page in the Hadoop User Interface (UI) page to view the status and settings of Capacity Scheduler queues. The following image show the Hadoop UI

Scheduler page (http://<hostname>:8088/cluster/scheduler) with the "support" queue selected:



2. CGroups

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.

2.1. Enabling CGroups

CGroups is a Linux kernel feature. Currently HDP supports CGroups on RHEL6 only. At this time there is no CGroups equivalent for Windows. CGroups are not enabled by default on HDP.

Enable CGroups

On Centos 6, CGroups are not set up by default. Run the following commands to set up CGroups:

```
yum install libcgroup
sudo mkdir /cgroup
mount -t cgroup -o cpu cpu /cgroup
```

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: `true`

Example:

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

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

Value: `/cgroup`

Example:

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

```
 <value>/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. To enable strict CPU limits, set the following property to `true`.

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

Value: `true`

Example:

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

2.2. Using CGroups

Strict CGroup CPU limits can be used 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 would be 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 could 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.

3. CPU Scheduling

This chapter describes how to allocate shared CPU and memory resources among users and groups in a Hadoop cluster.

As discussed in the Capacity Scheduler guide, 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 database structure, resource requirements, and access restrictions required by the various organizations, groups, and users that utilize cluster resources. When the default resource calculator (`DefaultResourceCalculator`) is used, resources are allocated based on memory alone.

CPU scheduling is enabled by using the Dominant Resource Calculator (`DominantResourceCalculator`) rather than the default resource calculator. The Dominant Resource Calculator is based on the Dominant Resource Fairness (DRF) model of resource allocation.

With the Dominant Resource Calculator, queues are still used to allocate cluster resources, but both CPU and memory are taken into consideration. In the DRF model, resource allocation takes into account the dominant resource required by a process. The Dominant Resource Calculator schedules both CPU-heavy and memory-heavy processes on the same node. 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.

CPU scheduling represents one aspect of YARN resource management capabilities that includes CGroups, node labels, archival storage, and memory as storage. CGroups should be used with CPU scheduling to constrain and manage CPU processes.

CPU scheduling is only recommended for Linux. Currently there is no isolation mechanism (CGroups equivalent) for Windows, so do not enable CPU scheduling on Windows.

3.1. Enabling CPU Scheduling

Enable CPU Scheduling in capacity-scheduler.xml

CPU scheduling is not enabled by default. To enable the CPU Scheduling, set the following property in the `/etc/hadoop/conf/capacity-scheduler.xml` file on the ResourceManager and NodeManager hosts:

Replace the `DefaultResourceCalculator` with the `DominantResourceCalculator`.

Property:`yarn.scheduler.capacity.resource-calculator`

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

```
<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>
```

Set Vcores in yarn-site.xml

In YARN, vcores (virtual cores) are used to normalize CPU resources across the cluster. The `yarn.nodemanager.resource.cpu-vcores` value sets the number of CPU cores that can be allocated for containers.

The number of vcores should be set 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>
```

It is also recommended that you enable CGroups along with CPU scheduling. CGroups are used as the isolation mechanism for CPU processes. With CGroups strict enforcement turned on, each CPU process gets only the resources it asks for. Without CGroups turned on, the DRF scheduler attempts to balance the load, but unpredictable behavior may occur.

Currently there is no isolation mechanism (CGroups equivalent) for Windows, so do not enable CPU scheduling on Windows.

3.2. Using CPU Scheduling

MapReduce Jobs Only

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

Mixed Workloads

One example of a mixed workload would be 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 a Storm jobs along with MapReduce jobs, the DRF scheduler does its best to balance memory and CPU resources, but you might start to see some degradation in performance. If you were to then add more CPU-intensive Storm jobs, individual jobs will start to take longer to run as the cluster CPU resources become consumed.

CGroups can be used along with CPU scheduling to help manage mixed workloads. 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 could also use node labels in conjunction with CPU scheduling and CGroups to restrict Storm-on-YARN jobs to a subset of cluster nodes.

3.3. Dominant Resource Fairness (DRF)

The Dominant Resource Calculator (DominantResourceCalculator) is used to e nable CPU scheduling. The Dominant Resource Calculator is based on the Dominant Resource Fairness (DRF) model of resource allocation.

DRF uses the concept of the *dominant resource* to compare multi-dimensional resources. The idea is that in a multi-resource environment, resource allocation should be determined by the dominant share of an entity (user or queue), which is the maximum share that the entity has been allocated of any resource (memory or CPU). Essentially, the DRF seeks to maximize the minimum dominant share across all entities.

For example, if user A runs CPU-heavy tasks and user B runs memory-heavy tasks, the DRF attempts to equalize CPU share of user A with the memory share of user B. In this case, the DRF would allocate more CPU and less memory to the tasks run by user A, and allocate less CPU and more memory to the tasks run by user B. In the single resource case – where all jobs are requesting the same resources – the DRF reduces to max-min fairness for that resource.

For more information about DRF, see Dominant Resource Fairness: Fair Allocation of Mulitple Resources.

4. Log Aggregation for Long-running Applications

This chapter describes how to use log aggregation to collect logs for long-running YARN applications.

In Hadoop logs are collected for an application only when it finishes running. This works well for applications that only run for a short time, but is not ideal for long-running applications such as Storm and HBase running on YARN using Slider.

If an application runs for days or weeks, it is useful to collect log information while the application is running. This log information can be used to:

- Debug hardware and software issues
- Review and optimize application performance
- Review application resource utilization

A long-running application may generate a large amount of log information. The application can use log rotation to prevent the log file from becoming excessively large. The YARN NodeManager will periodically aggregate the latest completed log files and make them accessible.

4.1. Enable Log Aggregation

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

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

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

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

Note

Clients should roll over their logs using a log roll-over application such as log4j.

4.2. Including and Excluding Log Files in YARN Applications Running on Slider

For YARN applications running on Slider, log aggregation is specified in the `global` section of the `resources.json`: configuration file:

```
"global": { "yarn.log.include.patterns": "", "yarn.log.exclude.patterns": ""
 },
```

If `yarn.log.include.patterns` is empty, all container logs are included. You can specify the name(s) of log files (for example, `agent.log`) that you do not want to aggregate using `yarn.log.exclude.patterns`.

For example, the following `resources.json` property settings will aggregate all logs beginning with "std", but exclude the `stderr` log.

```
"global": { "yarn.log.include.patterns": "std*", "yarn.log.exclude.patterns":
 "stderr"
 },
```

The aggregated logs are stored in the HDFS `/app-logs/` directory. The following command can be used to retrieve the logs: `yarn logs -applicationId <app_id>`

Note

For applications deployed directly on YARN, you can use `LogAggregationContext` inside `ApplicationSubmissionContext` to specify the include and exclude patterns.

5. Node Labels

This chapter describes how to use Node labels to restrict YARN applications so that they run only on cluster nodes that have a specified node label.

As discussed in "Capacity Scheduler", 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 can be assigned to cluster nodes. You can then associate node labels with capacity scheduler queues to specify which node label each queue is allowed to access.

When a queue is associated with one or more node labels, all applications submitted by the queue run only on nodes with those specified 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.

Node labels represent one aspect of YARN resource management capabilities that includes CPU scheduling, CGroups, archival storage, and memory as storage.

5.1. Configuring Node Labels

To enable node labels, 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.manager-class</name>
      <value>org.apache.hadoop.yarn.server.resourcemanager.nodelabels.
RMNodeLabelsManager</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. To start or stop the ResourceManager, use the applicable commands in the "Controlling HDP Services Manually" section of the HDP Reference Guide.

4. Add and Assign Node Labels

For demonstration purposes, the following commands show how to use the `yarn rmadmin` client to add the node labels "x" and "y", but you can add your own 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 "x,y"
```

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
14/11/21 13:09:55 INFO impl.TimelineClientImpl: Timeline service address:
 http://node-1.example.com:8188/ws/v1/timeline/
14/11/21 13:09:55 INFO client.RMProxy: Connecting to ResourceManager at
 node-1.example.com/240.0.0.10:8032
Node Labels: x,y
[root@node-1 /]#
```

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.

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

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

5. Associating Node Labels with Queues

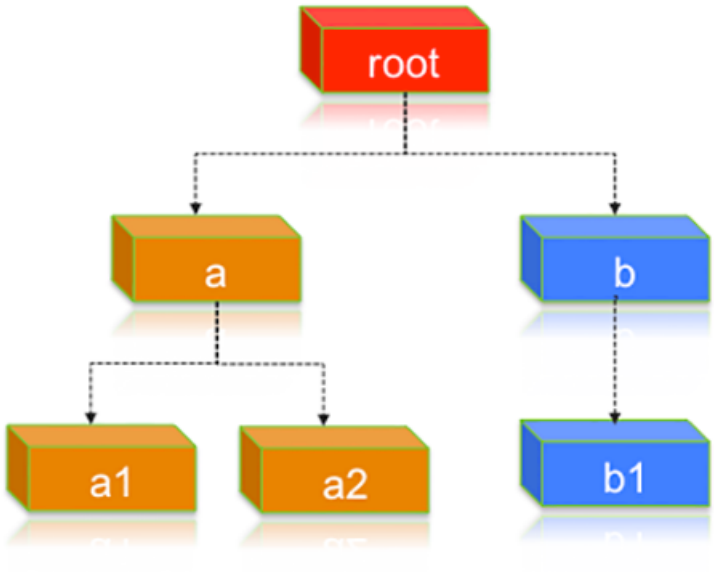
Now that we have created node labels, we can associate them with queues in the `/etc/hadoop/conf/capacity-scheduler.xml` file.

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>
```

6. 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
```

8. 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
[root@node-1 /]#
```

- List all node labels in the cluster: `yarn cluster --list-node-labels`

  Example:

```
[root@node-1 /]# yarn cluster --list-node-labels
14/11/21 13:09:55 INFO impl.TimelineClientImpl: Timeline service address:
 http://node-1.example.com:8188/ws/v1/timeline/
14/11/21 13:09:55 INFO client.RMProxy: Connecting to ResourceManager at
 node-1.example.com/240.0.0.10:8032
Node Labels: x,y
[root@node-1 /]#
```

- 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

[root@node-1 /]#
```

Node labels are also displayed in the ResourceManager UI on the Nodes and Scheduler pages.

Specifying 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>
```

Setting 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.

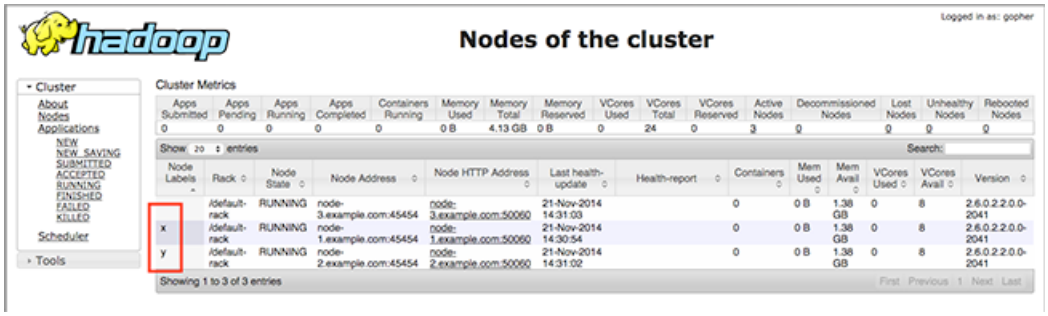
For example, to set "x"as the default node label for queue "b1", you would add the following property in the `capacity-scheduler.xml` file.

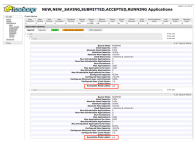
```
<property>
 <name>yarn.scheduler.capacity.root.b.b1.default-node-label-expression</name>
 <value>x</value>
</property>
```

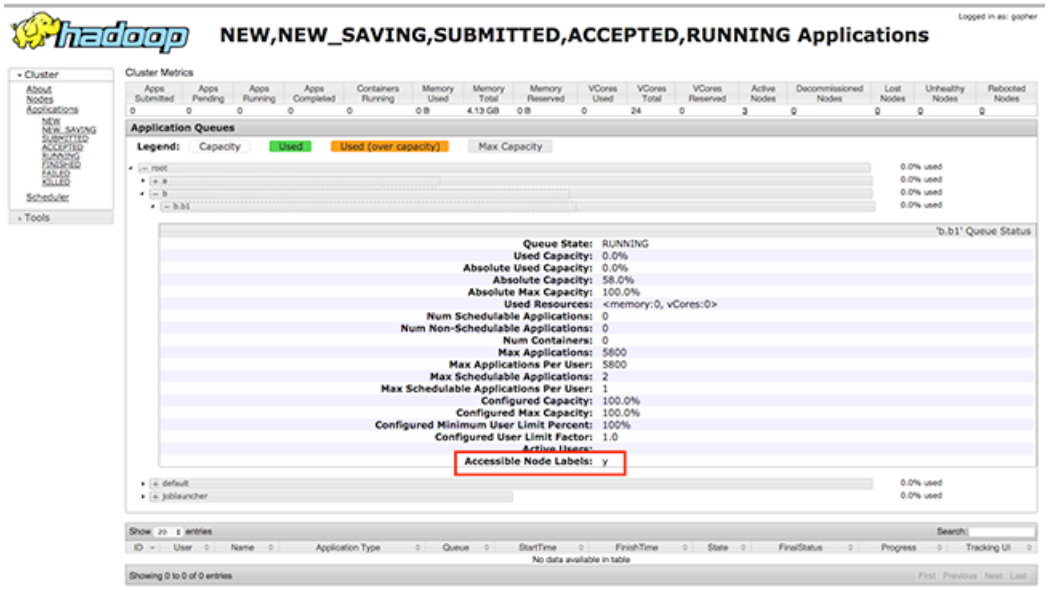
5.2. Using Node Labels

ResourceManger UI

The ResourceManager UI displays the node labels on the Nodes page, and also on the Scheduler page.







Setting Node Labels when Submitting Jobs

You can specify a label (using the `-node_label_expression` parameter) and 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.

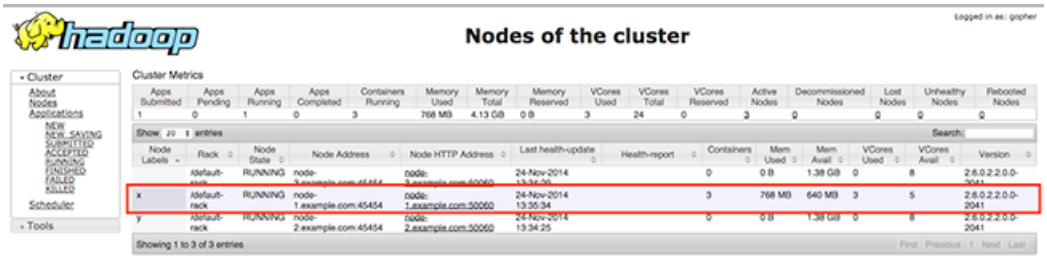
Note

You can only specify one node label in the `-node_label_expression`.

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 `-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 specifies 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.

6. Running Applications on YARN Using Slider

Apache Slider lets you deploy distributed applications across a Hadoop cluster. Slider leverages the YARN ResourceManager to allocate and distribute components of an application across a cluster.

Key Slider features:

- Run applications on YARN without changing the application code (as long as the application follows Slider developer guidelines). There is no need to develop a custom Application Master or other YARN code.
- Use the application registry for publishing and discovery of dynamic metadata.
- Run multiple instances of applications with different configurations or versions in one Hadoop cluster.
- Expand or shrink application component instances while an application is running.
- Transparently deploy applications in secure Kerberos clusters.
- Aggregate application logs from different containers.
- Run applications on a subset of cluster nodes using YARN node labels.
- Manage application, component, and container failures.

Slider leverages YARN capabilities to manage:

- Application recovery in cases of container failure
- Resource allocation (adding and removing containers)

6.1. System Requirements

Running Slider requires the following minimum system configuration:

- Hortonworks Data Platform 2.2
- Required Services: HDFS, YARN and ZooKeeper
- Oracle JDK 1.7 (64-bit)
- OpenSSL
- Python 2.6

6.2. Operating System Requirements

Slider is compatible with all operating systems supported by HDP.

Note

Accumulo on Slider is not supported on Windows, as Accumulo is not currently supported on Windows.

6.3. Installing Apache Slider

To install Apache Slider, follow the instructions in the Installing Apache Slider section in *Installing HDP Manually*.

6.4. Running Applications on Slider

The following sections describe how to configure, load, start, and verify Slider applications.

The Slider Application Specification

Configuring a Slider application includes two specification components: the Resource Specification (`resources.json`) and the Application Configuration (`appConfig.json`).

The Resource Specification

Slider needs to know what components (and how many components) are in an application package to deploy.

As Slider creates each instance of a component in its own YARN container, it also needs to know what to ask YARN for in terms of **memory** and **CPU** for those containers.

All this information goes into the **Resources Specification** file ("Resource Spec") named resources.json. The Resource Specification tells Slider how many instances of each component in the application to deploy, and also the parameters for YARN.

Resource Specification Example – Apache Memcached

```
{ "schema" : "http://example.org/specification/v2.0.0", "metadata" : {
 }, "global" : {
 }, "components": { "slider-appmaster": {
 }, "MEMCACHED": { "yarn.role.priority": "1", "yarn.component.instances": "1",
"yarn.vcores": "1", "yarn.memory": "256"
 }
 }
}
```

The Application Configuration

The Application Configuration file includes parameters that are specific to the application, rather than to YARN. The Application Configuration is applied to the default configuration provided by the application definition and then handed off to the associated component agent.

Application Configuration Example – memcached

```
{ "schema": "http://example.org/specification/v2.0.0", "metadata": {
 }, "global": { "application.def": "jmemcached-1.0.0.zip", "java_home": "/
usr/jdk64/jdk1.7.0_45", "site.global.additional_cp": "/usr/hdp/current/hadoop-
client/lib/*", "site.global.xmx_val": "256m", "site.global.xms_val": "128m",
```

```
 "site.global.memory_val": "200M", "site.global.listen_port": "${MEMCACHED.
ALLOCATED_PORT}{PER_CONTAINER}"
 }, "components": { "slider-appmaster": { "jvm.heapsize": "256M"
 }
 }
}
```

6.4.1. The Slider Application Package

The Slider application package includes all of the information needed to deploy a YARN application using Slider. It includes the application configuration files, as well as all application artifacts and the scripts required by the application.

Hortonworks provides the following application packages for Accumulo, HBase, and Storm on Linux:

centos6:

```
accumulo_pkg_url=http://public-repo-1.hortonworks.com/HDP/centos6/2.x/updates/
2.2.8.0/slider-app-packages/accumulo/slider-accumulo-app-package-1.6.1.2.2.8.
0-3150.zip
hbase_pkg_url=http://public-repo-1.hortonworks.com/HDP/centos6/2.x/updates/2.
2.8.0/slider-app-packages/hbase/slider-hbase-app-package-0.98.4.2.2.8.0-3150-
hadoop2.zip
storm_pkg_url=http://public-repo-1.hortonworks.com/HDP/centos6/2.x/updates/2.
2.8.0/slider-app-packages/storm/slider-storm-app-package-0.9.3.2.2.8.0-3150.
zip
```

centos5:

```
accumulo_pkg_url=http://public-repo-1.hortonworks.comHDP/centos5/2.x/updates/
2.2.8.0/slider-app-packages/accumulo/slider-accumulo-app-package-1.6.1.2.2.8.
0-3150.zip
hbase_pkg_url=http://public-repo-1.hortonworks.com/HDP/centos5/2.x/updates/2.
2.8.0/slider-app-packages/hbase/slider-hbase-app-package-0.98.4.2.2.8.0-3150-
hadoop2.zip
storm_pkg_url=http://public-repo-1.hortonworks.com/HDP/centos5/2.x/updates/2.
2.8.0/slider-app-packages/storm/slider-storm-app-package-0.9.3.2.2.8.0-3150.
zip
```

suse11sp3

```
accumulo_pkg_url=http://public-repo-1.hortonworks.com/HDP/suse11sp3/2.x/
updates/2.2.8.0/slider-app-packages/accumulo/slider-accumulo-app-package-1.6.
1.2.2.8.0-3150.zip
hbase_pkg_url=http://public-repo-1.hortonworks.com/HDP/suse11sp3/2.x/updates/
2.2.8.0/slider-app-packages/hbase/slider-hbase-app-package-0.98.4.2.2.8.
0-3150-hadoop2.zip
storm_pkg_url=http://public-repo-1.hortonworks.com/HDP/suse11sp3/2.x/updates/
2.2.8.0/slider-app-packages/storm/slider-storm-app-package-0.9.3.2.2.8.0-3150.
zip
```

sles11sp1

```
accumulo_pkg_url=http://public-repo-1.hortonworks.com/HDP/sles11sp1/2.x/
updates/2.2.8.0/slider-app-packages/accumulo/slider-accumulo-app-package-1.6.
1.2.2.8.0-3150.zip
hbase_pkg_url=http://public-repo-1.hortonworks.com/HDP/sles11sp1/2.x/updates/
2.2.8.0/slider-app-packages/hbase/slider-hbase-app-package-0.98.4.2.2.8.
0-3150-hadoop2.zip
```

```
storm_pkg_url=http://public-repo-1.hortonworks.com/HDP/sles11sp1/2.x/updates/
2.2.8.0/slider-app-packages/storm/slider-storm-app-package-0.9.3.2.2.8.0-3150.
zip
```

ubuntu12

```
accumulo_pkg_url=http://public-repo-1.hortonworks.com/HDP/ubuntu12/2.x/
updates/2.2.8.0/slider-app-packages/accumulo/slider-accumulo-app-package-1.6.
1.2.2.8.0-3150.zip
hbase_pkg_url=http://public-repo-1.hortonworks.com/HDP/ubuntu12/2.x/updates/2.
2.8.0/slider-app-packages/hbase/slider-hbase-app-package-0.98.4.2.2.8.0-3150-
hadoop2.zip
storm_pkg_url=http://public-repo-1.hortonworks.com/HDP/ubuntu12/2.x/updates/2.
2.8.0/slider-app-packages/storm/slider-storm-app-package-0.9.3.2.2.8.0-3150.
zip
```

debian6

```
accumulo_pkg_url=http://public-repo-1.hortonworks.com/HDP/debian6/2.x/updates/
2.2.8.0/slider-app-packages/accumulo/slider-accumulo-app-package-1.6.1.2.2.8.
0-3150.zip
hbase_pkg_url=http://public-repo-1.hortonworks.com/HDP/debian6/2.x/updates/2.
2.8.0/slider-app-packages/hbase/slider-hbase-app-package-0.98.4.2.2.8.0-3150-
hadoop2.zip
storm_pkg_url=http://public-repo-1.hortonworks.com/HDP/debian6/2.x/updates/2.
2.8.0/slider-app-packages/storm/slider-storm-app-package-0.9.3.2.2.8.0-3150.
zip
```

Note

Accumulo on Slider is not supported on Windows, as Accumulo is not currently supported on Windows.

For more information on Apache Slider application packaging, see this page. For more information on creating a Slider application package for Memcached, see this page.

The HDP .msi installer file for Windows includes the application packages for HBase and Storm on Windows.

6.4.2. Install the Application Package

Use the following command format to install the application package on YARN.

```
/usr/hdp/current/slider-client/bin/./slider install-package --name <name> --
package <package_path_on_disk>
```

6.4.3. Start the Application

Once the steps above are completed, the Slider Command Line Interface (CLI) can be used to start the application using the following command format:

```
/usr/hdp/current/slider-client/bin/./slider create <application_name> --
template appConfig.json --resources resources.json
```

Default files (`appConfig-default.json` and `resources-default.json`) files are included in each application package.

Note

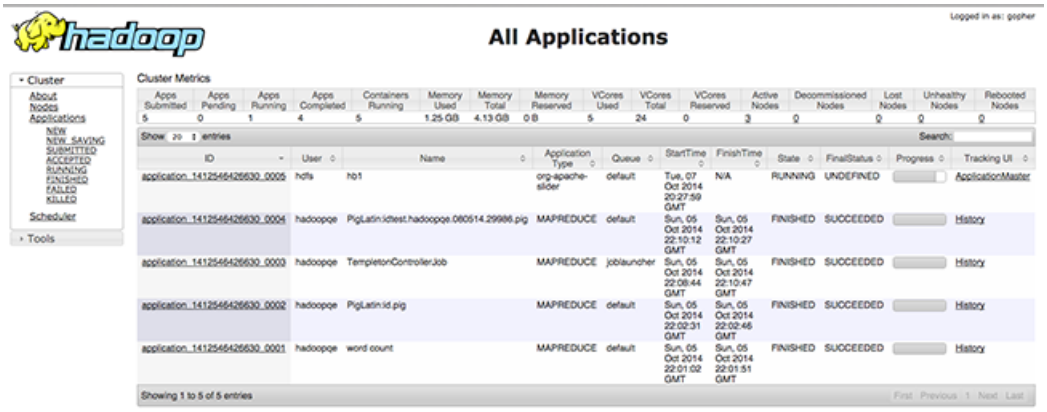
You can create multiple instances of an application by running the Slider `create` command with different application names. For example, you might change configuration settings in appConfig.json and create a separate instance to evaluate the new configuration.

6.4.4. Verify the Application

You can use the Slider `status` CLI command to verify the application launch:

```
/usr/hdp/current/slider-client/bin/./slider status <application_name>
```

You can also verify the successful launch of the application with the YARN Resource Manager Web UI. In most instances, this UI is accessible via a web browser at port 8088 of the Resource Manager Host:



The specific information for the running application is accessible via the "ApplicationMaster" link that can be seen in the far right column of the row associated with the running application (probably the top row):

6.4.5. Manage the Application Lifecycle

Once started, applications can be stopped, restarted, destroyed, and flexed as follows:

stop <name> [–force] [–wait time] [–message text]

Stops the application instance. The running application is stopped. Its settings are retained in HDFS.

The `--wait` argument can specify a time in seconds to wait for the application instance to be stopped.

The `--force` flag causes YARN asked directly to terminate the application instance. The `--message` argument supplies an optional text message to be used in the request: this will appear in the application's diagnostics in the YARN RM UI.

If an unknown (or already stopped) application instance is named, no error is returned.

Example:

```
/usr/hdp/current/slider-client/bin/./slider stop app1
```

start <name> [–wait milliseconds] [–out <filename>] Start a stopped application instance, recreating it from its previously saved state.

After the application is launched, if an `--out` argument specified a file, the "YARN application report" will be saved as a .json document into the file specified.

Example

```
/usr/hdp/current/slider-client/bin/./slider start app1
```

destroy <name> Destroy a (stopped) application instance. Important: This deletes all persistent data.

Example:

```
/usr/hdp/current/slider-client/bin/./slider destroy app1
```

flex <name> [–component component count]* Flex the number of workers in an application instance to the new value. If greater than before, new copies of the component will be requested. If less, component instances will be destroyed.

This operation has a return value of 0 if the change was submitted. Changes are not immediate and depend on the availability of resources in the YARN cluster

It returns -1 if there is no running application instance.

Examples:

```
slider flex app1 --component worker 8 --filesystem hdfs://<host>:<port>
slider flex app1 --component master 2 --filesystem hdfs://<host>:<port>
```

For more information on Slider CLI commands, see the Apache Slider documentation.

You can use the Slider `help` command to display a list of available commands.

```
[root@node-1 /]# /usr/hdp/current/slider-client/bin/./slider help
2014-10-31 09:20:10,852 [main] INFO client.SliderClient - Usage: slider
COMMAND [options] where COMMAND is one of
 am-suicide        Tell the Slider Application Master to simulate a
                   process failure by terminating itself
```

```
 build              Build a Slider cluster specification -but do not start it
 create             Create a live Slider application
 update             Update template for a Slider application
 destroy            Destroy a frozen Slider application)
 diagnostics        Diagnose the configuration of the running slider
                    application and slider client
 exists             Probe for an application running
 flex               Flex a Slider application
 stop               Stop a running application
 help               Print help information
 install-package    Install the application package in the home directory
                    under sub-folder packages
 install-keytab     Install the Kerberos keytab file in the sub-folder
                    'keytabs' of the user's Slider base directory
 kill-container     Kill a container in the application
 list               List running Slider applications
 registry           Query the registry of a YARN application
 resolve            Query the registry of a YARN application
 status             Get the status of an application
 start              Start a stopped application
 version            Print the Slider version information
Most commands print help when invoked without parameters
2014-10-31 09:20:10,856 [main] INFO util.ExitUtil - Exiting with status 0
```

Most commands display Help when invoked without parameters, similar to the usage statement shown below:

```
[root@node-1 /]# slider status
2014-10-31 09:45:11,817 [main] ERROR main.ServiceLauncher - org.apache.
slider.core.exceptions.BadCommandArgumentsException: Not enough arguments
for action: status Expected minimum 1 but got 0
Usage: slider status <application>
 --manager <manager>                Binding (usually hostname:port)
                                          of the YARN resource manager
                                          (optional)
 --basepath <basePath>              Slider base path on the filesystem
                                          (optional)
 --filesystem <filesystemBinding>   Filesystem Binding (optional)
 -D <definitions>                   Definitions (optional)
 --debug                                  Debug mode (optional)
 -S <sysprops>                      System properties in the form name
                                          value These are set after the JVM
 is
                                          started. (optional)
 --out <output>                     Output file for the configuration
                                          data (optional)

2014-10-31 09:45:11,826 [main] INFO util.ExitUtil - Exiting with status 40
[root@node-1 /]#
```

6.4.6. The Application Registry

Each application publishes several artifacts that can be used by an application administrator or an application client. Typical data published includes the applied configuration, links to application JMX endpoint or monitoring UI, and log folders.

All published data is available at the publisher endpoint that is hosted by Application Master launched by Slider. Here is an example of the publisher endpoint:

```
http://c6401.ambari.apache.org:47457/ws/v1/slider/publisher
```

From this endpoint, you can access several named property containing useful administrative information.

Publisher URI	Description
`{publisher-endpoint}/slider/quicklinks`	Named URLs that the application publishes.
`{publisher-endpoint}/slider/logfolders`	Log folders for the application components (YARN should be configured to retain logs).
`{publisher-endpoint}/slider/storm-site`	Applied configurations by the application (for example, storm-site, hbase-site).

Example output from `/slider/quicklinks`:

```
{
   "description": "QuickLinks",
   "entries": {
   "org.apache.slider.jmx": "http://c6401.ambari.apache.org:50154/api/cluster
         /summary",
   "org.apache.slider.metrics": "http://c6401.ambari.apache.org/cgi-bin/rrd.
         py?c=Application2",
   "org.apache.slider.monitor": "http://c6401.ambari.apache.org:41806",
   "org.apache.slider.ganglia": "http://c6401.ambari.apache.org/ganglia?c=
         Application2"
   },
   "updated": 0,
   "empty": false
}
```

6.5. Running HBase on YARN via Slider

Running HBase on YARN via Slider offers the following advantages:

- Easier installation and start-up.
- You can run mulitple HBase clusters on one Hadoop cluster.
- Process management (start/stop) is much easier with Slider.
- You can run different versions of HBase on the same Hadoop cluster.
- HBase cluster size is now a parameter that you can easily change.

6.5.1. Downloading and Installing the HBase Application Package

- **To install the HBase Application Package:**
- Make a working directory for the HBase-on-Slider application package:

  ```
  mkdir -p /usr/work/app-packages/hbase
  ```

- Download the HBase-on-Slider application package for your operating system to the `/usr/work/app-packages/hbase` directory. For example, use the following command to download the centos6 version:

```
cd /usr/work/app-packages/hbase
wget http://public-repo-1.hortonworks.com/HDP/centos6/2.x/updates/2.2.8.
0/slider-app-packages/hbase/slider-hbase-app-package-0.98.4.2.2.8.0-3150-
hadoop2.zip
```

- Use the following command format to install the application package on YARN:

```
su <user>
/usr/hdp/current/slider-client/bin/./slider install-package --name HBASE --
package <package path>
```

Where <user> is the user who will create the HBase application instance.

For example, if you downloaded the HBase package to the /usr/work/app-packages/hbase folder:

```
su <user>
/usr/hdp/current/slider-client/bin/./slider install-package --name HBASE --
package /usr/work/app-packages/hbase/slider-hbase-app-package-0.98.4.2.2.8.
0-3150-hadoop2.zip
```

Note that the --name (in this case, "HBASE") should match the value of the <name> element in the package metainfo.xml flle, and that the install destination in HDFS is in the user's home directory under ./slider/package/HBASE/*.zip

6.5.2. Configuring HBase on YARN via Slider

Accessing the HBase Configuration Files

The HBase application package includes default application and resource specification files. The package includes both non-secure (appConfig-default.json) and secure (appConfig-secured-default.json) versions of the application specification. You can save these files as another name, and then edit the files to customize the HBase configuration.

You can use the unzip command to extract the HBase application and resource specification files from the HBase-on-Slider application package. For example, you would use the following command to extract the files from the HBase application package in the /usr/work/app-packages/hbase directory:

```
unzip /usr/work/app-packages/hbase/slider-hbase-app-package-0.98.4.2.2.8.
0-3150-hadoop2.zip appConfig-default.json -d /usr/work/app-packages/hbase/
unzip /usr/work/app-packages/hbase/slider-hbase-app-package-0.98.4.2.2.8.
0-3150-hadoop2.zip resources-default.json -d /usr/work/app-packages/hbase/
```

You can use the following commands to copy and rename the default HBase application and resource specification files in the /usr/work/app-packages/hbase/ directory:

```
cp /usr/work/app-packages/hbase/appConfig-default.json /usr/work/app-packages/
hbase/appConfig.json
cp /usr/work/app-packages/hbase/resources-default.json /usr/work/app-packages/
hbase/resources.json
```

The default configuration files should work as-is, but you can also edit the files to adjust the HBase configuration.

6.5.3. Configuring HBase on YARN on Secure Clusters

As previously mentioned, the HBase-on-Slider application package includes both non-secure (`appConfig-default.json`) and secure (`appConfig-secured-default.json`) versions of the application specification.

On secure clusters, you should use the secure version of the application specification. The security-related entries in the `appConfig-secured.json` file are listed below. You will need to replace the values labeled "TODO" with the security settings for the cluster.

```
"site.hbase-site.hbase.coprocessor.master.classes": "org.apache.hadoop.hbase.
security.access.AccessController", "site.hbase-site.hbase.coprocessor.region.
classes" : "org.apache.hadoop.hbase.security.token.TokenProvider,org.apache.
hadoop.hbase.security.access.AccessController", "site.hbase-site.hbase.
regionserver.kerberos.principal": "${TODO-RS-PRINCIPAL}", "site.hbase-site.
hbase.regionserver.keytab.file": "${TODO-RS-KEYTAB}", "site.hbase-site.hbase.
master.kerberos.principal": "${TODO-MASTER-PRINCIPAL}", "site.hbase-site.
hbase.master.keytab.file": "${TODO-MASTER-KEYTAB}", "site.hbase-site.hbase.
rest.authentication.kerberos.keytab" : "${TODO-REST-AUTH-KEYTAB}", "site.
hbase-site.hbase.rest.kerberos.principal" : "${TODO-REST-PRINCIPAL}", "site.
hbase-site.hbase.rest.keytab.file" : "${TODO-REST-KEYTAB}", "site.hbase-site.
hbase.thrift.keytab.file" : "${TODO-THRIFT-KEYTAB}", "site.hbase-site.hbase.
thrift.kerberos.principal" : "${TODO-THRIFT-PRINCIPAL}", "site.hdfs-site.dfs.
namenode.kerberos.principal": "${TODO-NN-PRINCIPAL}", "site.hdfs-site.dfs.
namenode.kerberos.internal.spnego.principal": "${TODO-NN-SPNEGO-PRINCIPAL}",
 "site.hdfs-site.dfs.secondary.namenode.kerberos.principal": "${TODO-SNN-
PRINCIPAL}", "site.hdfs-site.dfs.secondary.namenode.kerberos.internal.spnego.
principal": "${TODO-SNN-SPNEGO-PRINCIPAL}", "site.hdfs-site.dfs.datanode.
kerberos.principal": "${TODO-DN-PRINCIPAL}",

Note: rest and thrift components are included above. The values in curly
 braces need to be filled out.
In components section: "slider-appmaster": { "jvm.heapsize": "256M", "slider.
am.keytab.local.path": "${TODO-HEADLESS-KEYTAB}", "slider.keytab.principal.
name": "${TODO-HEADLESS-PRINCIPAL}"
 },
```

Note

For more information on configuring HBase-on-Slider on secure clusters, including information about keytab-associated properties and the available keytab distribution options, see Apache Slider Security.

6.5.4. Components in HBase on YARN

You can specify the following components (also referred to as "roles") when deploying HBase on YARN via Slider:

- `HBASE_MASTER` # This corresponds to HBase master process.
- `HBASE_REGIONSERVER` # This corresponds to region server process.
- `HBASE_REST` # This corresponds to REST (aka Stargate) gateway.
- `HBASE_THRIFT` # This corresponds to Thrift gateway.
- `HBASE_THRIFT2` # This corresponds to Thrift2 gateway.

The following is a sample `resources.json` file with each of these roles configured:

```
{
   "schema": "http://example.org/specification/v2.0.0",
   "metadata": {
   },
   "global": {
          "yarn.log.include.patterns": "",
          "yarn.log.exclude.patterns": "",
          "yarn.log.interval": "0"
   },
   "components": {
          "HBASE_MASTER": {
          "yarn.role.priority": "1",
          "yarn.component.instances": "1"
          },
          "slider-appmaster": {
          },
          "HBASE_REGIONSERVER": {
          "yarn.role.priority": "2",
          "yarn.component.instances": "4"
          },
          "HBASE_THRIFT": {
          "yarn.role.priority": "4",
          "yarn.component.instances": "1",
          "yarn.memory": "256"
          },
          "HBASE_THRIFT2": {
          "yarn.role.priority": "5",
          "yarn.component.instances": "0",
          "yarn.memory": "256"
          },
          "HBASE_REST": {
          "yarn.role.priority": "3",
          "yarn.component.instances": "1",
          "yarn.memory": "256"
          }
     }
}
```

Note

Thrift and Thrift2 provide similar functionality. You should use either Thrift or Thrift2, but not both.

6.5.5. Launching an HBase Application Instance

Use the following command format to launch HBase:

```
su <user>
/usr/hdp/current/slider-client/bin/./slider create <hb_cluster_name> --
template appConfig.json --resources resources.json
```

Where `<user>` is the user who installed the HBase application package.

For example:

```
su <user>
```

```
/usr/hdp/current/slider-client/bin/./slider create hb1 --template /usr/work/
app-packages/hbase/appConfig.json --resources /usr/work/app-packages/hbase/
resources.json
```

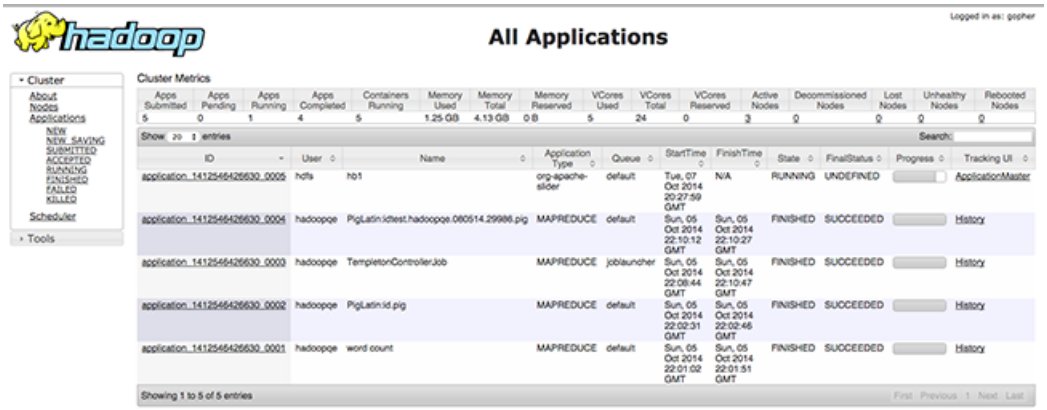
Note

Cluster names may not include uppercase characters.

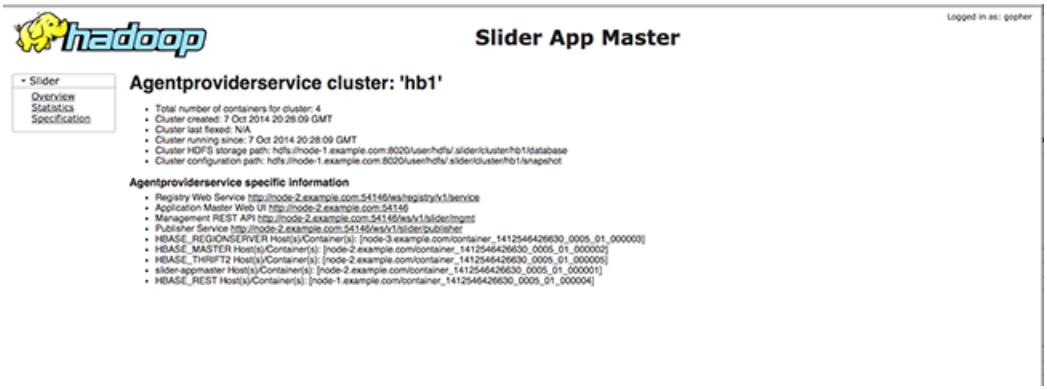
You can use the Slider CLI `status` command to verify the application launch:

```
/usr/hdp/current/slider-client/bin/./slider status <application_name>
```

You can also verify the successful launch of the HBase application with the YARN Resource Manager Web UI. In most instances, this UI is accessible via a web browser at port 8088 of the Resource Manager Host:



The specific information for the running application is accessible via the "ApplicationMaster" link that can be seen in the far right column of the row associated with the running application (probably the top row):



6.5.6. Deployment Considerations

For HA purposes, it is not advisable to deploy two Masters on the same host.

Thrift/Thrift2/REST gateways are likely to receive heavy traffic. They should be deployed on dedicated hosts.

Memory Considerations for Running One Component on a Node

You can adjust the amount of memory given to a component to achieve mutual exclusion of components, depending upon the NodeManager configuration on each node. Typically all nodes have the same value independent of the actual memory.

Assuming the memory capacity for each NodeManager is known (`yarn.nodemanager.resource.memory-mb`), you can configure the component to ask for 51% (basically more than half) of the maximum capacity. You also need to ensure that the maximum possible memory allocation (`yarn.scheduler.maximum-allocation-mb`) allows that value.

For example, if `yarn.nodemanager.resource.memory-mb` = `yarn.scheduler.maximum-allocation-mb` = 2048 Set `yarn.memory` = 1280 for the RegionServer.

Then set the HBase Master/RegionServer max memory to be 256 MB less than that to allow for the agent memory consumption # the agent should not be more than 100 MB but you can just assume that it consumes ~256 MB. So you can set the HBase Master/RegionServer variables for memory limit to 1024 MB.

Log Aggregation

This feature is backed by https://issues.apache.org/jira/browse/YARN-2468.

Log aggregation is specified in the `global` section of `resources.json`:

```
"global": {
                  "yarn.log.include.patterns": "",
                  "yarn.log.exclude.patterns": "",
                  "yarn.log.interval": "0"
    },
```

The `yarn.log.interval` unit is seconds.

You can specify the name(s) of log files (for example, `agent.log`) that you do not want to aggregate using `yarn.log.exclude.patterns`.

The aggregated logs are stored in the HDFS `/app-logs/` directory.

The following command can be used to retrieve the logs:

```
yarn logs -applicationId <app_id>
```

Reserving Nodes for HBase

You can use YARN node labels to reserve cluster nodes for applications and their components.

Node labels are specified with the `yarn.label.expression` property. If no label is specified, only non-labeled nodes are used when allocating containers for component instances.

If label expression is specified for slider-appmaster, then it also becomes the default label expression for all components. To take advantage of default label expression, leave out the property (see HBASE_REGIONSERVER in the example). A label expression with an empty string (`"yarn.label.expression":""`) is equivalent to nodes without labels.

For example, the following is a resources.json file for an HBase cluster that uses node labels. The label for the application instance is "hbase1", and the label expression for the HBASE_MASTER components is "hbase1_master". HBASE_REGIONSERVER instances will automatically use label "hbase1". Alternatively, if you specify (`"yarn.label.expression":""`) for HBASE_REGIONSERVER then the containers will only be allocated on nodes with no labels.

```
{ "schema": "http://example.org/specification/v2.0.0",
  "metadata": {
  },
  "global": {
  },
  "components": {
            "HBASE_MASTER": {
                  "yarn.role.priority": "1",
                  "yarn.component.instances": "1",
                  "yarn.label.expression":"hbase1_master"
           },
           "HBASE_REGIONSERVER": {
                  "yarn.role.priority": "1",
                  "yarn.component.instances": "1",
           },
           "slider-appmaster": {
                  "yarn.label.expression":"hbase1"
           }
     }
}
```

Specifically, for the above example you would need to:

- Create two node labels, "hbase1" and "hbase1_master" (using `yarn rmadmin` commands)
- Assign the labels to nodes (using `yarn rmadmin` commands)
- Create a queue by defining it in the `capacity-scheduler.xml` configuration file.
- Allow the queue to access to the labels and ensure that appropriate min/max capacity is assigned.
- Refresh the queues (`yarn rmadmin -refreshQueues`)
- Create the Slider application against the above queue using parameter `--queue` while creating the application.

Retrieving Effective hbase-site.xml

Once HBase is running on Slider, you can use the following command to retrieve `hbase-site.xml` so that your client can connect to the cluster:

```
slider registry --getconf hbase-site --name <cluster name> --format xml --
dest <path to local hbase-site.xml> --filesystem <hdfs namenode> --manager
 <resource manager>
```

Note that the `hbase.tmp.dir` in the returned file may be inside the YARN local directory for the container (which is local to the host where the container is running).

For example:

```
 <property>
 <name>hbase.tmp.dir</name> <value>/grid/0/hadoop/yarn/
local/usercache/test/appcache/application_1413942326640_0001/
container_1413942326640_0001_01_000005/work/app/tmp</value>
 <source/>
 </property>
```

If the client does not have access to this directory, changing the `hbase.tmp.dir` to a directory writable to the user would solve the permission issue.

Retrieving REST gateway for Slider HBase. You can retrieve quicklinks for Slider HBase first. The "org.apache.slider.hbase.rest" entry would show hostname and port for REST gateway.

Retrieving thrift gateway for Slider HBase You can retrieve quicklinks for Slider HBase first. The "org.apache.slider.hbase.thrift" entry would show hostname and port for thrift gateway.

Retrieving thrift2 gateway for Slider HBase You can retrieve quicklinks for Slider HBase first. The "org.apache.slider.hbase.thrift2" entry would show hostname and port for thrift2 gateway.

Workaround for HBase Client Issue

After you create an HBase application instance in Slider, you can use the `slider registry` command to retrieve the `hbase-site.xml` file:

```
slider registry --name hbase1 --getconf hbase-site --format xml --out hbase-
site.xml
```

The path of the `hbase.tmp.dir` property in the returned file will be inside the YARN local directory for the container (which is local to the host where the container is running):

```
 <property>
 <name>hbase.tmp.dir</name> <value>/grid/0/hadoop/yarn/
local/usercache/test/appcache/application_1413942326640_0001/
container_1413942326640_0001_01_000005/work/app/tmp</value>
 <source/>
 </property>
```

The HBase client (specifically the HBase shell) expects to see a directory named "jars" in this directory. If it is not there the HBase client attempts to create it, resulting in a failure to perform shell operations:

```
[test@ip-10-0-0-66 hbase]$ hbase --config ./conf shell
...
hbase(main):001:0> status 'simple'
...
ERROR: Failed to create local dir /grid/0/hadoop/yarn/local/
usercache/test/appcache/application_1413942326640_0001/
container_1413942326640_0001_01_000005/work/app/tmp/local/jars,
 DynamicClassLoader failed to init
```

Workaround: Change the `hbase.tmp.dir` to a directory that is writable for the user running "hbase shell" or to a directory which already has the "jars" directory.

6.6. Running Storm on YARN via Slider

Running Storm on YARN via Slider offers the following advantages.

- Easier installation and start-up.
- You can run multiple instances of Storm on one Hadoop cluster.
- Process management (start/stop) is much easier with Slider.
- You can run different versions of Storm on the same Hadoop cluster.

Note

You can run Storm stand-alone and Storm on Slider on the same cluster, but only if they use different topologies.

6.6.1. Downloading and Installing the Storm Application Package

- **To install the Storm Application Package:**
- Make a working directory for the Storm-on-Slider application package:

```
mkdir -p /usr/work/app-packages/storm
```

- Download the Storm-on-Slider application package for your operating system to the `/usr/work/app-packages/storm` directory. For example, use the following command to download the centos6 version:

```
cd /usr/work/app-packages/storm
wget http://public-repo-1.hortonworks.com/HDP/centos6/2.x/updates/2.2.8.0/
slider-app-packages/storm/slider-storm-app-package-0.9.3.2.2.8.0-3150.zip
```

- Use the following command format to install the application package on YARN:

```
su <user>
/usr/hdp/current/slider-client/bin/./slider install-package --name STORM --
package <package path>
```

Where `<user>` is the user who will create the Storm application instance.

For example, if you downloaded the Storm package to the `/usr/work/app-packages/storm` folder:

```
su <user>
/usr/hdp/current/slider-client/bin/./slider install-package --name STORM --
package /usr/work/app-packages/storm/slider-storm-app-package-0.9.3.2.2.8.
0-3150.zip
```

Note that the `--name` (in this case, "STORM") should match the value of the `<name>` element in the package `metainfo.xml` flle, and that the install destination in HDFS is in the user's home directory under ./slider/package/STORM/*.zip

6.6.2. Configuring Storm on YARN

Accessing the Storm Configuration Files

The Storm application package includes default application and resource specification files. The package includes both non-secure (`appConfig-default.json`) and secure (`appConfig-secured-default.json`) versions of the application specification. You can save these files as a another name, and then edit the files to customize the Storm configuration.

You can use the `unzip` command to extract the Storm application and resource specification files from the Storm-on-Slider application package. For example, you would use the following command to extract the files from the Storm application package in the `/usr/work/app-packages/storm` directory:

```
unzip /usr/work/app-packages/storm/slider-storm-app-package-0.9.3.2.2.8.
0-3150.zip appConfig-default.json -d /usr/work/app-packages/storm
unzip /usr/work/app-packages/storm/slider-storm-app-package-0.9.3.2.2.8.
0-3150.zip resources-default.json -d /usr/work/app-packages/storm
```

You can use the following commands to copy and rename the default Storm application and resource specification files in the `/usr/work/app-packages/storm` directory:

```
cp /usr/work/app-packages/storm/appConfig-default.json /usr/work/app-packages/
storm/appConfig.json
cp /usr/work/app-packages/storm/resources-default.json /usr/work/app-packages/
storm/resources.json
```

Application Configuration for Storm on YARN

The basic properties to adjust are the heapsize parameters for Storm daemons such as nimbus, supervisor, UI, and worker childopts to fit your system. Memory considerations are discussed in the "Deployment Considerations" section of this guide.

The following is an example of a Storm `appConfig.json` file:

```
{ "schema": "http://example.org/specification/v2.0.0",
   "metadata": {
 },
"global": {
 "application.def": ".slider/package/STORM/slider-storm-app-package-0.9.3.2.2.
8.0-3150.zip",
 "java_home": "/usr/jdk64/jdk1.7.0_67",
 "create.default.zookeeper.node": "true",
 "system_configs": "core-site",
 "site.global.app_user": "yarn",
 "site.global.app_root": "${AGENT_WORK_ROOT}/app/install/apache-storm-0.9.3.2.
2.8.0-3150",
 "site.global.user_group": "hadoop",
 "site.global.ganglia_server_host": "${NN_HOST}",
 "site.global.ganglia_server_id": "Application2",
 "site.global.ganglia_enabled":"true",
 "site.global.ganglia_server_port": "8668",

 "site.storm-site.storm.log.dir" : "${AGENT_LOG_ROOT}",
 "site.storm-site.storm.zookeeper.servers": "['${ZK_HOST}']",
```

```
 "site.storm-site.nimbus.thrift.port": "${NIMBUS.ALLOCATED_PORT}",
 "site.storm-site.storm.local.dir": "${AGENT_WORK_ROOT}/app/tmp/storm",
 "site.storm-site.transactional.zookeeper.root": "/transactional",
 "site.storm-site.storm.zookeeper.port": "2181",
 "site.storm-site.nimbus.childopts": "-Xmx1024m -javaagent:${AGENT_WORK_ROOT}/
app/install/apache-storm-0.9.3.2.2.8.0-3150/external/storm-jmxetric/lib/
jmxetric-1.0.4.jar=host=${@//site/global/ganglia_server_host},port=${@//
site/global/ganglia_server_port},wireformat31x=true,mode=multicast,config=
${AGENT_WORK_ROOT}/app/install/apache-storm-0.9.3.2.2.0.0-908/external/storm-
jmxetric/conf/jmxetric-conf.xml,process=Nimbus_JVM",
 "site.storm-site.worker.childopts": "-Xmx768m -javaagent:${AGENT_WORK_ROOT}/
app/install/apache-storm-0.9.3.2.2.8.0-3150/external/storm-jmxetric/lib/
jmxetric-1.0.4.jar=host=${@//site/global/ganglia_server_host},port=${@//
site/global/ganglia_server_port},wireformat31x=true,mode=multicast,config=
${AGENT_WORK_ROOT}/app/install/apache-storm-0.9.3.2.2.0.0-908/external/storm-
jmxetric/conf/jmxetric-conf.xml,process=Worker_%ID%_JVM",
 "site.storm-site.dev.zookeeper.path": "${AGENT_WORK_ROOT}/app/tmp/dev-storm-
zookeeper",
 "site.storm-site.drpc.invocations.port": "0",
 "site.storm-site.storm.zookeeper.root": "${DEFAULT_ZK_PATH}",
 "site.storm-site.transactional.zookeeper.port": "null",
 "site.storm-site.nimbus.host": "${NIMBUS_HOST}",
 "site.storm-site.ui.port": "${STORM_UI_SERVER.ALLOCATED_PORT}",
 "site.storm-site.supervisor.slots.ports": "[${SUPERVISOR.ALLOCATED_PORT}
{PER_CONTAINER},${SUPERVISOR.ALLOCATED_PORT}{PER_CONTAINER}]",
 "site.storm-site.supervisor.childopts": "-Xmx256m -Dcom.sun.management.
jmxremote -Dcom.sun.management.jmxremote.ssl=false -Dcom.sun.management.
jmxremote.authenticate=false -Dcom.sun.management.jmxremote.port=0 -javaagent:
${AGENT_WORK_ROOT}/app/install/apache-storm-0.9.3.2.2.8.0-3150/external/storm-
jmxetric/lib/jmxetric-1.0.4.jar=host=${NN_HOST},port=8668,wireformat31x=true,
mode=multicast,config=${AGENT_WORK_ROOT}/app/install/apache-storm-0.9.3.2.2.0.
0-908/external/storm-jmxetric/conf/jmxetric-conf.xml,process=Supervisor_JVM",
 "site.storm-site.drpc.port": "0",
 "site.storm-site.logviewer.port": "${SUPERVISOR.ALLOCATED_PORT}
{PER_CONTAINER}"
 },
 "components": {
   "slider-appmaster": {
   "jvm.heapsize": "256M"
   }
  }
 }
```

Note

The user name for the `site.global.app_user` property should be the user name you will use to launch the Storm application instance.

Resource Components in Storm on YARN

You can specify the following components (also referred to as "roles") when deploying Storm on YARN via Slider:

- `NIMBUS` # Storm Nimbus process.
- `STORM_UI_SERVER` # Storm Web UI process.
- `DRPC_SERVER` # Storm DRPC Server.

- SUPERVISOR # Storm Supervisor process.

The following is an example of a Storm `resources.json` file with these roles configured:

```
{
"schema" : "http://example.org/specification/v2.0.0",
"metadata" : {
 },
"global" : {
  "yarn.log.include.patterns": "",
  "yarn.log.exclude.patterns": "",
  "yarn.log.interval": "0"
 },
 "components": {
   "slider-appmaster": {
    },
   "NIMBUS": {
     "yarn.role.priority": "1",
     "yarn.component.instances": "1",
     "yarn.memory": "2048",
     "yarn.label.expression":"storm1_nimbus_label"
    },
    "STORM_UI_SERVER": {
    "yarn.role.priority": "2",
    "yarn.component.instances": "1",
    "yarn.memory": "1278",
    "yarn.label.expression":"storm1_nimbus_label"
    },
    "DRPC_SERVER": {
      "yarn.role.priority": "3",
      "yarn.component.instances": "1",
      "yarn.memory": "1278",
     “"yarn.label.expression":"storm1_nimbus_label"
    },
   "SUPERVISOR": {
    "yarn.role.priority": "4",
    "yarn.component.instances": "1",
    "yarn.memory": "3072",
    "yarn.label.expression":"storm1_supervisor_label"
    }
   }
 }
```

The memory and number of instances of SUPERVISOR should be adjusted for your system and desired application instance size. By default SUPERVISOR has two worker ports. If you allocate more workers per SUPERVISOR you should also adjust the `yarn.memory` vaule in the SUPERVISOR section of the `resources.json` file.

For all of the other roles (`NIMBUS`, `DRPC_SERVER`, `STORM_UI_SERVER`), you should configure only one instance.

6.6.3. Configuring Storm on YARN on Secure Clusters

1. Generate and Distribute Keytab Files for Storm

On the KDC host:

- Log in as root.

- Create a Storm server principal for each NodeManager host:

```
kadmin.local -q "addprinc -randkey <server_principal_name>/
<node_manager_host_name>@EXAMPLE.COM"
```

- Create a Storm client principal:

```
kadmin.local -q "addprinc -randkey <client_principal_name>@EXAMPLE.COM"
```

- Export the principals to a keytab file:
  - For each NodeManager host:

```
kadmin.local -q "xst -norandkey -k /etc/security/keytabs/nimbus.keytab
 <server_principal_name>/<node_manager_host_name>@EXAMPLE.COM"
```

  - For the Storm client principal:

```
kadmin.local -q "xst -norandkey -k /etc/security/keytabs/storm.keytab
 <client_principal_name>@EXAMPLE.COM"
```

- Distribute the keytab file to the NodeManager hosts on which the application components will be launched. Be sure to set the permissions so that the runtime elements are allowed to access the keytab files, for example:

```
sudo su -
chown root:users <keytab_file>
chmod 440 <keytab_file>
```

2. Add an OS User for HDFS Access

You need to add an OS user for proper HDFS access (user and group availability) to the secure Storm deployment:

- Create system users with the same short names as the generated server principal and client principal:

```
useradd -n <storm server or client principal short name>
passwd <storm server or client principal short name>
```

  You must specify a password in order to activate the user account.

- Associate the users to the appropriate user groups:

```
usermod -a -G hadoop <storm server or client principal short name>
```

Edit the Secure Version of the Application Configuration Files

As previously mentioned, the Storm-on-Slider application package includes both non-secure (`appConfig-default.json`) and secure (`appConfig-secured-default.json`) versions of the application specification.

On secure clusters, you should use the secure version of the application specification. The security-related entries in the `appConfig-secured.json` file are listed below.

```
"site.storm-site.nimbus.authorizer": "backtype.storm.security.auth.
authorizer.SimpleACLAuthorizer", "site.storm-site.storm.thrift.transport":
 "backtype.storm.security.auth.kerberos.KerberosSaslTransportPlugin",
 "site.storm-site.java.security.auth.login.config": "${AGENT_WORK_ROOT}/
app/install/apache-storm-0.9.3.2.2.8.0-3150/conf/storm_jaas.conf", "site.
storm-site.storm.principal.tolocal": "backtype.storm.security.auth.
KerberosPrincipalToLocal", "site.storm-site.storm.zookeeper.superACL":
 "sasl:storm", "site.storm-site.nimbus.admins": "['jon', 'storm']", "site.
storm-site.nimbus.supervisor.users": "['storm']", "site.storm-site.nimubs.
authorizer": "backtype.storm.security.auth.authorizer.SimpleACLAuthorizer",
 "site.storm-site.storm.thrift.transport": "backtype.storm.security.auth.
kerberos.KerberosSaslTransportPlugin", "site.storm-site.storm.principal.
tolocal": "backtype.storm.security.auth.KerberosPrincipalToLocal", "site.
storm-site.ui.filter": "org.apache.hadoop.security.authentication.server.
AuthenticationFilter", "site.storm-site.ui.filter.params": "{'type':
 'kerberos', 'kerberos.principal': 'HTTP/_HOST', 'kerberos.keytab': '/etc/
security/keytabs/spnego.service.keytab', 'kerberos.name.rules': 'RULE:[2:$1@
$0]([jt]t@.*EXAMPLE.COM)s/.*/$MAPRED_USER/ RULE:[2:$1@$0]([nd]n@.*EXAMPLE.
COM)s/.*/$HDFS_USER/DEFAULT'}", "site.storm-env.kerberos_domain": "EXAMPLE.
COM", "site.storm-env.storm_client_principal_name": "storm@EXAMPLE.COM",
 "site.storm-env.storm_server_principal_name": "storm_server/_HOST@EXAMPLE.
COM", "site.storm-env.storm_client_keytab": "/etc/security/keytabs/storm.
keytab", "site.storm-env.storm_server_keytab": "/etc/security/keytabs/nimbus.
keytab"
```

some key points regarding these configuration properties:

- The properties assume the use of the Kerberos domain "EXAMPLE.COM". Change the domain name to match the name configured for your environment.
- The JAAS configuration (`storm_jaas.conf`) path will be dependent on the version of the Storm distribution you are using (for example, version apache-storm-0.9.3.2.2.8.0-3150 as shown above).
- The `superACL` property should point to the client principal short name.
- The `nimbus.admins` property values should include both the Storm client principal short name and the principal associated with the Slider user who launches the application.
- The `supervisor.users` property should be set to the short name of the Storm client principal.
- The `ui.filter.params` property requires an HTTP/Web principal. This principle can be found in the `spnego.service.keytab` file.
- The `storm-env` properties are fairly straightforward – simply provide the server principal, client principal, and keytab file locations.

At this point you should be ready to launch a Storm cluster using the Slider `create` command. You will need to authenticate against Kerberos and obtain a TGT using the `kinit` command prior to invoking the Slider `create` command:

```
kinit <user name>
```

Note

For more information on configuring Storm-on-Slider on secure clusters, including information about keytab-associated properties and the available keytab distribution options, see Apache Slider Security.

6.6.4. Launching a Storm Application Instance

Note

The application instance can only be launched by the user specified in the `site.global.app_user` in the Storm `appConfig.json` file.

Use the following command format to launch Storm:

```
su <user>
/usr/hdp/current/slider-client/bin/./slider create <storm_cluster_name> --
template appConfig.json --resources resources.json
```

Where `<user>` is the user who installed the Storm application package.

For example:

```
su <user>
/usr/hdp/current/slider-client/bin/./slider create storm1 --template /usr/
work/app-packages/storm/appConfig.json --resources /usr/work/app-packages/
storm/resources.json
```

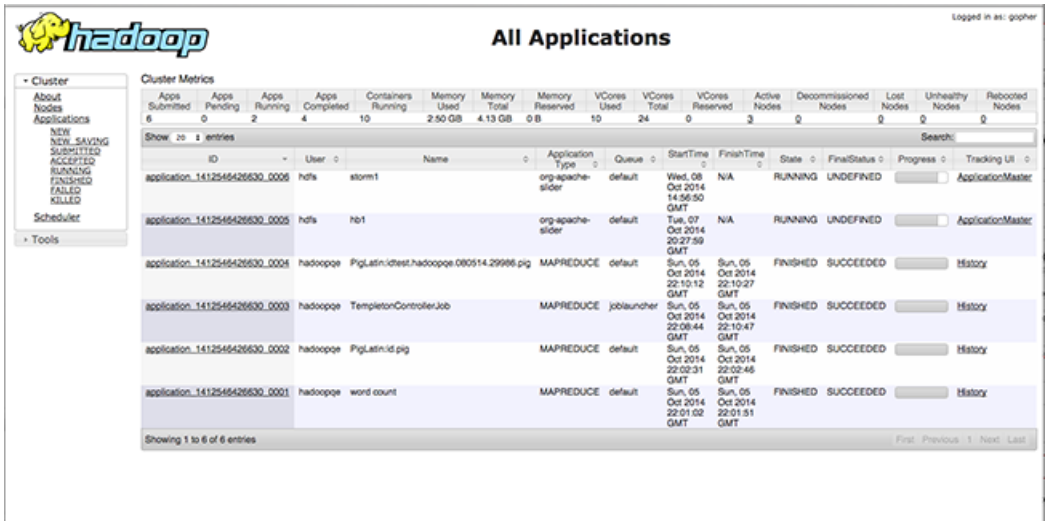
Note

Cluster names may not contain uppercase characters.

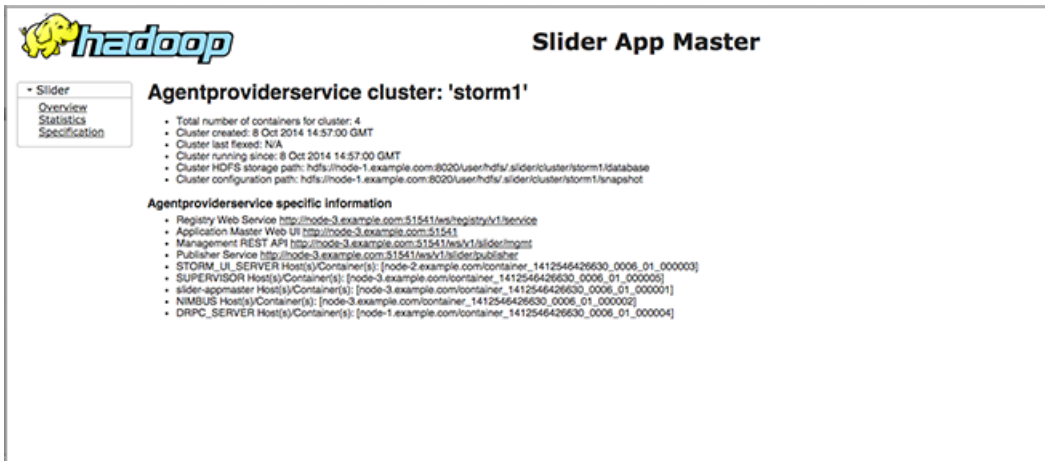
You can use the Slider CLI `status` command to verify the application launch:

```
/usr/hdp/current/slider-client/bin/./slider status <application_name>
```

The successful launch of the Storm application can also be verified via the YARN Resource Manager Web UI. This UI is usually accessible via a Web browser at port 8088 of the Resource Manager Host:



The specific information for the running application is accessible via the "ApplicationMaster" link that can be seen in the far right column of the row associated with the running application (probably the top row):



6.6.5. Deployment Considerations

Memory Considerations for Running One Component on a Node

You can adjust the amount of memory given to a component to achieve mutual exclusion of components, depending upon the NodeManager configuration on each node. Typically all nodes have the same value independent of the actual memory.

Assuming the memory capacity for each NodeManager is known (`yarn.nodemanager.resource.memory-mb`), you can configure the component to ask for 51% (basically more than half) of the maximum capacity. You also need to ensure that the maximum possible memory allocation (`yarn.scheduler.maximum-allocation-mb`) allows that value.

For example, if `yarn.nodemanager.resource.memory-mb` = `yarn.scheduler.maximum-allocation-mb` = 2048 Set `yarn.memory` = 1280 for the `nimbus.childopts` property in the appConfig.json file.

For SUPERVISOR allocate 1/4th of 1280 to `supervisor.childopts` and depending on how many workers you plan to run, divide the rest of the available container memory by the number of workers and adjust `worker.childopts` accordingly.

Log Aggregation

This feature is backed by https://issues.apache.org/jira/browse/YARN-2468.

Log aggregation is specified in the `global` section of `resources.json`:

```
"global": { "yarn.log.include.patterns": "", "yarn.log.exclude.patterns": "",
 "yarn.log.interval": "0"
 },
```

The `yarn.log.interval` unit is seconds.

You can specify the name(s) of log files (for example, `agent.log`) that you do not want to aggregate using `yarn.log.exclude.patterns`.

The aggregated logs are stored in the HDFS `/app-logs/` directory.

The following command can be used to retrieve the logs:

```
yarn logs -applicationId <app_id>
```

For Storm you should exclude all active logs. Any file under `storm.log.dir/*.log` and the `storm.log.dir/metadata` directory should be excluded. You should be collecting the rolled-over logs. So any file with `*.log.%i` is ready to be collected for log aggregation.

The following is an example of a Storm `log4j.properties` file:

```
<configuration scan="true" scanPeriod="60 seconds">
 <appender name="A1" class="ch.qos.logback.core.rolling.RollingFileAppender">
 <file>${storm.log.dir}/${logfile.name}</file>
 <rollingPolicy class="ch.qos.logback.core.rolling.FixedWindowRollingPolicy">
 <fileNamePattern>${storm.log.dir}/${logfile.name}.%i</fileNamePattern>
 <minIndex>1</minIndex>
 <maxIndex>9</maxIndex>
 </rollingPolicy>

 <triggeringPolicy class="ch.qos.logback.core.rolling.
SizeBasedTriggeringPolicy">
 <maxFileSize>100MB</maxFileSize>
 </triggeringPolicy>

 <encoder>
 <pattern>%d{yyyy-MM-dd HH:mm:ss} %c{1} [%p] %m%n</pattern>
 </encoder>
 </appender>

 <appender name="ACCESS" class="ch.qos.logback.core.rolling.
RollingFileAppender">
 <file>${storm.log.dir}/access.log</file>
 <rollingPolicy class="ch.qos.logback.core.rolling.FixedWindowRollingPolicy">
 <fileNamePattern>${storm.log.dir}/access.log.%i</fileNamePattern>
 <minIndex>1</minIndex>
 <maxIndex>9</maxIndex>
 </rollingPolicy>

 <triggeringPolicy class="ch.qos.logback.core.rolling.
SizeBasedTriggeringPolicy">
 <maxFileSize>100MB</maxFileSize>
 </triggeringPolicy>

 <encoder>
 <pattern>%d{yyyy-MM-dd HH:mm:ss} %c{1} [%p] %m%n</pattern>
 </encoder>
 </appender>

 <appender name="METRICS" class="ch.qos.logback.core.rolling.
RollingFileAppender">
 <file>${storm.log.dir}/metrics.log</file>
 <rollingPolicy class="ch.qos.logback.core.rolling.FixedWindowRollingPolicy">
 <fileNamePattern>${storm.log.dir}/logs/metrics.log.%i</fileNamePattern>
 <minIndex>1</minIndex>
 <maxIndex>9</maxIndex>
 </rollingPolicy>
```

```
 <triggeringPolicy class="ch.qos.logback.core.rolling.
SizeBasedTriggeringPolicy">
 <maxFileSize>2MB</maxFileSize>
 </triggeringPolicy>

 <encoder>
 <pattern>%d %-8r %m%n</pattern>
 </encoder>
 </appender>

 <root level="INFO">
 <appender-ref ref="A1"/>
 </root>

 <logger name="backtype.storm.security.auth.authorizer" additivity="false">
 <level value="INFO" />
 <appender-ref ref="ACCESS" />
 </logger>

 <logger name="backtype.storm.metric.LoggingMetricsConsumer" additivity=
"false" >
 <level value="INFO"/>
 <appender-ref ref="METRICS"/>
 </logger>

</configuration>
```

Reserving Nodes for Storm

You can use YARN node labels to reserve cluster nodes for applications and their components. You could use node labels to reserve cluster nodes for Storm to ensure that NIMBUS and SUPERVISOR provide a consistent performance level.

Node labels are specified with the `yarn.label.expression` property. If no label is specified, only non-labeled nodes are used when allocating containers for component instances.

A brief summary is that you could label particular YARN nodes for Storm, say with labels “storm1_nimbus” for nimbus and “storm1_supervisor” for supervisors, and create a separate queue for assigning containers to these nodes. “STORM_UI_SERVER” can run in the same label as storm1_nimbus. “storm1_drpc” label should be for DRPC. To use these labeled nodes, you would add yarn.label.expression parameters to the Storm components in your resources.json file (including the slider-appmaster), e.g. "yarn.label.expression": "storm1_supervisor". When you run the “slider create” command for your Storm cluster, you would add the parameter “–queue <queue name>”.

Running Example Topologies on Storm-on-YARN

Installing the Storm-Slider Client

- Make sure you are using HDP-2.2
- `yum search storm`
- `storm_<latest_version>-slider-client`
- Pick the latest package

- `yum install storm_<latest_version>-slider-client`
- edit `/etc/storm-slider-client/conf/storm-slider-env.sh`
- `export SLIDER_HOME=/usr/hdp/current/slider`
- `export JAVA_HOME=/usr/jdk64/jdk1.7`

Using Quicklinks to Access the Storm UI

Use the following command format:

```
storm-slider --app <yarn-app-name> quicklinks
```

For example:

```
/usr/hdp/<hdp_version>/storm-slider-client/bin/storm-slider --app cl2
 quicklinks
```

Example output:

```
{ "org.apache.slider.jmx" : "http://ec2-54-172-39-242.compute-1.amazonaws.
com:34579/api/v1/cluster/summary", "org.apache.slider.metrics" : "http://
ec2-54-172-207-11.compute-1.amazonaws.com/cgi-bin/rrd.py?c=Application2",
 "nimbus.host_port" : "http://ec2-54-172-39-242.compute-1.amazonaws.
com:38818", "org.apache.slider.monitor" : "http://ec2-54-172-39-242.
compute-1.amazonaws.com:34579", "org.apache.slider.metrics.ui" : "http://
ec2-54-172-207-11.compute-1.amazonaws.com/ganglia?c=Application2"
}
```

`org.apache.slider.metrics.monitor` points to the Storm UI.

Deploy the Storm Word Count Topology

Use the following command to deploy the Storm wordcount topology:

```
/usr/hdp/current/storm-slider-client/bin/storm-slider --app <cluster_name>
 jar /usr/hdp/current/storm-slider-client/contrib/storm-starter/storm-starter-
topologies-*.jar storm.starter.WordCountTopology wordcount
```

Replace <`cluster_name`> with the cluster name deployed using Slider.

Tests

- Deploy wordcount topology and go to the Storm UI. Check to see if wordcount topology is listed under topologies. Click on wordcount topology and check to see if emitted and transferred numbers are increasing.
- Start multiple Storm clusters, then deploy wordcount topologies on each cluster and check to see if they are running.
- Check to see if the labeling of torm components is working by listing out the NIMBUS process on the storm1_nimbus labeled node.

6.7. Running Accumulo on YARN via Slider

Running Accumulo on YARN via Slider offers the following advantages:

- Easier installation and start-up.

- You can run multiple instances of Accumulo on one Hadoop cluster.
- Process management (start/stop) is much easier with Slider.
- You can run different versions of Accumulo on the same Hadoop cluster.
- You can take advantage of YARN features such as container restart and allocation flexibility.

6.7.1. Downloading and Installing the Accumulo Application Package

- **To install the Accumulo Application Package:**
- Make a working directory for the Accumulo-on-Slider application package:

```
mkdir -p /usr/work/app-packages/accumulo
```

- Download the Accumulo-on-Slider application package for your operating system to the `/usr/work/app-packages/accumulo` directory. For example, use the following command to download the centos6 version:

```
cd /usr/work/app-packages/accumulo
wget http://public-repo-1.hortonworks.com/HDP/centos6/2.x/updates/2.2.8.0/
slider-app-packages/accumulo/slider-accumulo-app-package-1.6.1.2.2.8.0-3150.
zip
```

- Use the following command format to install the application package on YARN:

```
su <user>
/usr/hdp/current/slider-client/bin/./slider install-package --name ACCUMULO
 --package <package path>
```

Where `<user>` is the user who will create the Accumulo application instance.

For example, if user "hw_qa" downloaded the Accumulo package to the `/usr/work/app-packages/accumulo` folder:

```
su hw_qa
/usr/hdp/current/slider-client/bin/./slider install-package --name ACCUMULO
 --package /usr/work/app-packages/accumulo/slider-accumulo-app-package-1.6.
1.2.2.8.0-3150.zip
```

Note that the `--name` (in this case, "ACCUMULO") should match the value of the `<name>` element in the package `metainfo.xml` flle, and that the install destination in HDFS is in the user's home directory under ./slider/package/ACCUMULO/*.zip

Note

Accumulo on Slider is not supported on Windows, as Accumulo is not currently supported on Windows.

6.7.2. Configuring Accumulo on YARN

Accessing the Accumulo Configuration Files

The Accumulo application package includes default application and resource specification files. The package includes both non-secure (`appConfig-default.json`) and secure (`appConfig-secured-default.json`) versions of the application specification. You can save these files as a another name, and then edit the files to customize the Accumulo configuration.

You can use the `unzip` command to extract the Accumulo application and resource specification files from the Accumulo-on-Slider application package. For example, you would use the following command to extract the files from the Accumulo application package in the `/usr/work/app-packages/accumulo` directory:

```
unzip /usr/work/app-packages/accumulo/slider-accumulo-app-package-1.6.1.2.2.8.
0-3150.zip appConfig-default.json -d /usr/work/app-packages/accumulo
unzip /usr/work/app-packages/accumulo/slider-accumulo-app-package-1.6.1.2.2.8.
0-3150.zip resources-default.json -d /usr/work/app-packages/accumulo
```

You can use the following commands to copy and rename the default Storm application and resource specification files in the `/usr/work/app-packages/accumulo` directory:

```
cp /usr/work/app-packages/accumulo/appConfig-default.json /usr/work/app-
packages/accumulo/appConfig.json
cp /usr/work/app-packages/accumulo/resources-default.json /usr/work/app-
packages/accumulo/resources.json
```

Application Configuration for Accumulo on YARN

The following is an example of an `appConfig.json` file for Accumulo on YARN via Slider. The basic properties to adjust for your system are the heap size, the Accumulo memory properties, and the location of JAVA_HOME. The directories and classpaths are configured properly for HDP in the default `appConfig-default.json` file, but you must set the JAVA_HOME value in the "global" section of the `appConfig.json` file to match your system JAVA_HOME setting.

```
{
    "schema": "http://example.org/specification/v2.0.0",
    "metadata": {
    },
    "global": { "application.def": ".slider/package/ACCUMULO/slider-accumulo-
app-package-1.6.1.2.2.8.0-3150.zip",
    "java_home": "/usr/hadoop-jdk1.6.0_31",
    "site.global.app_root": "${AGENT_WORK_ROOT}/app/install/accumulo-1.6.1.2.
2.8.0-3150",
    "site.global.app_user": "${USER}",
    "site.global.user_group": "hadoop",

    "site.accumulo-env.java_home": "${JAVA_HOME}",
    "site.accumulo-env.tserver_heapsize": "256m",
    "site.accumulo-env.master_heapsize": "128m",
    "site.accumulo-env.monitor_heapsize": "64m",
    "site.accumulo-env.gc_heapsize": "64m",
    "site.accumulo-env.other_heapsize": "128m",
    "site.accumulo-env.hadoop_prefix": "/usr/hdp/current/hadoop-client",
    "site.accumulo-env.hadoop_conf_dir": "/etc/hadoop/conf",
    "site.accumulo-env.zookeeper_home": "${zk.dir}",

    "site.client.instance.name": "${USER}-${CLUSTER_NAME}",

    "site.global.accumulo_root_password": "NOT_USED",
```

```
    "site.global.ssl_cert_dir": "ssl",
    "site.global.monitor_protocol": "http",

    "site.accumulo-site.instance.volumes": "${DEFAULT_DATA_DIR}/data",
    "site.accumulo-site.instance.zookeeper.host": "${ZK_HOST}",
    "site.accumulo-site.instance.security.authenticator": "org.apache.slider.
accumulo.CustomAuthenticator",

    "site.accumulo-site.general.security.credential.provider.paths": "jceks://
hdfs/user/${USER}/accumulo-${CLUSTER_NAME}.jceks",
    "site.accumulo-site.instance.rpc.ssl.enabled": "false",
    "site.accumulo-site.instance.rpc.ssl.clientAuth": "false",
    "site.accumulo-site.general.kerberos.keytab": "",
    "site.accumulo-site.general.kerberos.principal": "",

    "site.accumulo-site.tserver.memory.maps.native.enabled": "false",
    "site.accumulo-site.tserver.memory.maps.max": "80M",
    "site.accumulo-site.tserver.cache.data.size": "7M",
    "site.accumulo-site.tserver.cache.index.size": "20M",
    "site.accumulo-site.tserver.sort.buffer.size": "50M",
    "site.accumulo-site.tserver.walog.max.size": "40M",

    "site.accumulo-site.trace.user": "root",

    "site.accumulo-site.master.port.client": "0",
    "site.accumulo-site.trace.port.client": "0",
    "site.accumulo-site.tserver.port.client": "0",
    "site.accumulo-site.gc.port.client": "0",
    "site.accumulo-site.monitor.port.client": "${ACCUMULO_MONITOR.
ALLOCATED_PORT}",
    "site.accumulo-site.monitor.port.log4j": "0",
    "site.accumulo-site.master.replication.coordinator.port": "0",
    "site.accumulo-site.replication.receipt.service.port": "0",

    "site.accumulo-site.general.classpaths": "$ACCUMULO_HOME/lib/accumulo-
server.jar,\n$ACCUMULO_HOME/lib/accumulo-core.jar,\n$ACCUMULO_HOME/lib/
accumulo-start.jar,\n$ACCUMULO_HOME/lib/accumulo-fate.jar,\n$ACCUMULO_HOME/
lib/accumulo-proxy.jar,\n$ACCUMULO_HOME/lib/[^.].*.jar,\n$ZOOKEEPER_HOME/
zookeeper[^.].*.jar,\n$HADOOP_CONF_DIR,\n$HADOOP_PREFIX/[^.].*.jar,\n
$HADOOP_PREFIX/lib/[^.].*.jar,\n$HADOOP_PREFIX/share/hadoop/common/.*.jar,\n
$HADOOP_PREFIX/share/hadoop/common/lib/.*.jar,\n$HADOOP_PREFIX/share/hadoop/
hdfs/.*.jar,\n$HADOOP_PREFIX/share/hadoop/mapreduce/.*.jar,\n$HADOOP_PREFIX/
share/hadoop/yarn/.*.jar,\n/usr/hdp/current/hadoop-client/.*.jar,\n/usr/hdp/
current/hadoop-client/lib/.*.jar,\n/usr/hdp/current/hadoop-hdfs-client/.*.jar,
\n/usr/hdp/current/hadoop-mapreduce-client/.*.jar,\n/usr/hdp/current/hadoop-
yarn-client/.*.jar,"
 },
 "credentials": {
    "jceks://hdfs/user/${USER}/accumulo-${CLUSTER_NAME}.jceks": ["root.
initial.password", "instance.secret", "trace.token.property.password"]
 },
 "components": {
    "slider-appmaster": {
       "jvm.heapsize": "256M",
       "slider.am.keytab.local.path": "",
       "slider.keytab.principal.name": ""
    }
  }
}
```

Resource Components in Accumulo on YARN

You can specify the following components (also referred to as "roles") when deploying Accumulo on YARN via Slider:

- ACCUMULO_MASTER # Accumulo master process.
- ACCUMULO_TSERVER # Accumulo tablet server process.
- ACCUMULO_MONITOR # Accumulo monitor web UI
- ACCUMULO_GC # Accumulo garbage collector process
- ACCUMULO_TRACER # Accumulo trace collector process

The following is an example of an Accumulo `resources.json` file with these roles configured:

```
{
   "schema": "http://example.org/specification/v2.0.0",
   "metadata": {
    },
    "global": {
       "yarn.log.include.patterns": "",
       "yarn.log.exclude.patterns": ""
    },
    "components": {
       "ACCUMULO_MASTER": {
          "yarn.role.priority": "1",
          "yarn.component.instances": "1",
          "yarn.memory": "256"
       },
       "slider-appmaster": {
       },
       "ACCUMULO_TSERVER": {
          "yarn.role.priority": "2",
          "yarn.component.instances": "1",
          "yarn.memory": "512"
       },
       "ACCUMULO_MONITOR": {
          "yarn.role.priority": "3",
          "yarn.component.instances": "1",
          "yarn.memory": "128"
       },
       "ACCUMULO_GC": {
          "yarn.role.priority": "4",
          "yarn.component.instances": "1",
          "yarn.memory": "128"
       },
       "ACCUMULO_TRACER": {
          "yarn.role.priority": "5",
          "yarn.component.instances": "1",
          "yarn.memory": "256"
      }
   }
}
```

The memory and number of instances of each component should be adjusted for your system and desired application instance size. You typically only need to request

one instance of the ACCUMULO_MONITOR, ACCUMULO_GC, and ACCUMULO_TRACER processes. For HA (High Availability) purposes, you will generally want two instances of ACCUMULO_MASTER, and enough instances of ACCUMULO_TSERVER to support your application.

6.7.3. Configuring Accumulo on YARN on Secure Clusters

As previously mentioned, the Accumulo-on-Slider application package includes both non-secure (`appConfig-default.json`) and secure (`appConfig-secured-default.json`) versions of the application specification. On secure clusters, you should use the secure version of the application specification.

To configure Accumulo for Kerberos, the following properties should be changed in the `appConfig.json` file to specify a keytab and principal for Accumulo processes to use. You must generate the keytabs on all the hosts and provide their location to Accumulo in the keytab property. If the keytab is not headless, the recommended form for the principal is `accumulo/_HOST@<realm>`, with the realm for your system. The Slider Application Master also needs a keytab and a principal, which can be (but does not need to be) the same one used for the Accumulo processes.

```
"global": {
     "site.accumulo-site.general.kerberos.keytab": "",
     "site.accumulo-site.general.kerberos.principal": "",
   },
   "components": {
      "slider-appmaster": {
         "slider.am.keytab.local.path": "",
         "slider.keytab.principal.name": ""
     }
   }
```

should be changed to:

```
"global": {
   "site.accumulo-site.general.kerberos.keytab": <keytab file abs path>,
   "site.accumulo-site.general.kerberos.principal": <principal name>,
   },
   "components": {
      "slider-appmaster": {
         "slider.am.keytab.local.path": <keytab file abs path>,
         "slider.keytab.principal.name": <principal name>
    }
  }
```

Note

For more information on configuring Accumulo-on-Slider on secure clusters, including information about keytab-associated properties and the available keytab distribution options, see Apache Slider Security.

6.7.4. Launching an Accumulo Application Instance

1. Iniatialize Accumulo Passwords

Before starting an Accumulo cluster on YARN, you must initialize the passwords needed by Accumulo. These passwords are specified in the credentials section of the `appConfig.json` file. By default they are the `root.initial.password` (the initial password for the root user, which can be changed later using the Accumulo API), the `instance.secret` (a shared secret needed by Accumulo server processes), and the `trace.token.property.password` (the password for the user that collects traces, which is the root user in the default configuration — so it should match the `root.initial.password`).

These passwords can be created using the following commands:

```
/usr/hdp/current/hadoop-client/bin/hadoop credential create root.initial.
password -provider jceks://hdfs/user/<user>/accumulo-<accumulo_cluster_name>.
jceks -v <root password>
/usr/hdp/current/hadoop-client/bin/hadoop credential create instance.secret
 -provider jceks://hdfs/user/<user>/accumulo-<accumulo_cluster_name>.jceks -v
 <instance secret>
/usr/hdp/current/hadoop-client/bin/hadoop credential create trace.
token.property.password -provider jceks://hdfs/user/<user>/accumulo-
<accumulo_cluster_name>.jceks -v <root password again>
```

Where <user> is the user who will create the Accumulo application instance and <accumulo_cluster_name> is the name of the Accumulo cluster that will be used in the Slider `create` command.

For example, using the user "hw_qa" and the Accumulo cluster name "accumulo1":

```
/usr/hdp/current/hadoop-client/bin/hadoop credential create root.initial.
password -provider jceks://hdfs/user/hw_qa/accumulo-accumulo1.jceks -v
 testRootPassword1
/usr/hdp/current/hadoop-client/bin/hadoop credential create instance.secret -
provider jceks://hdfs/user/hw_qa/accumulo-accumulo1.jceks -v instanceSecret1
/usr/hdp/current/hadoop-client/bin/hadoop credential create trace.token.
property.password -provider jceks://hdfs/user/hw_qa/accumulo-accumulo1.jceks -
v testRootPassword1
```

2. Launch the Accumulo Application Instance

Use the following command format to launch Accumulo:

```
su <user>
/usr/hdp/current/slider-client/bin/./slider create <accumulo_cluster_name> --
template appConfig.json --resources resources.json
```

Where <user> and <accumulo_cluster_name> are the user and cluster name that were used when the Accumulo passwords were created.

For example:

```
su hw_qa
/usr/hdp/current/slider-client/bin/./slider create accumulo1 --template /usr/
work/app-packages/accumulo/appConfig.json --resources /usr/work/app-packages/
accumulo/resources.json
```

Note

The cluster name may not include uppercase characters.

You can use the Slider CLI `status` command to verify the application launch:

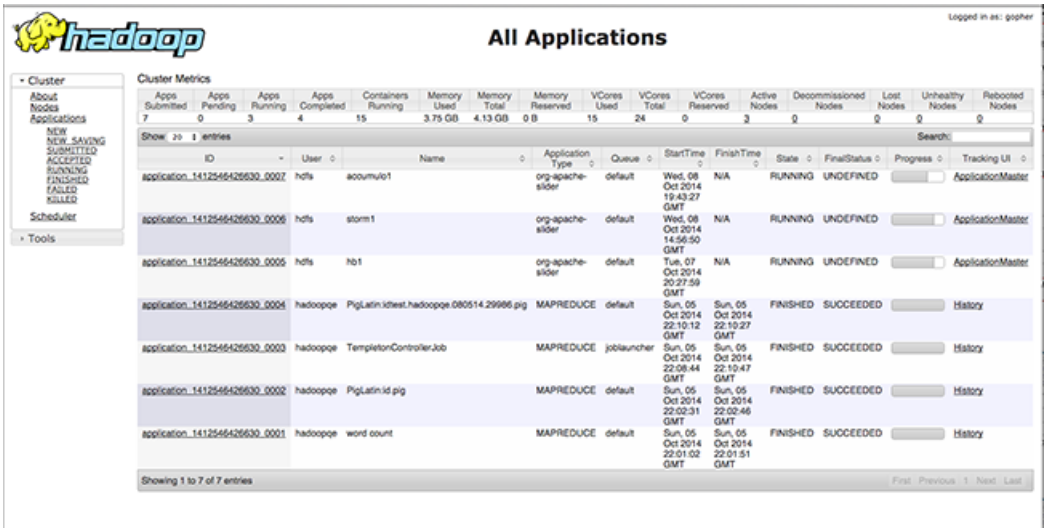
```
/usr/hdp/current/slider-client/bin/./slider status <accumulo_cluster_name>
```

You can also use the Slider CLI `list` command to verify the application launch:

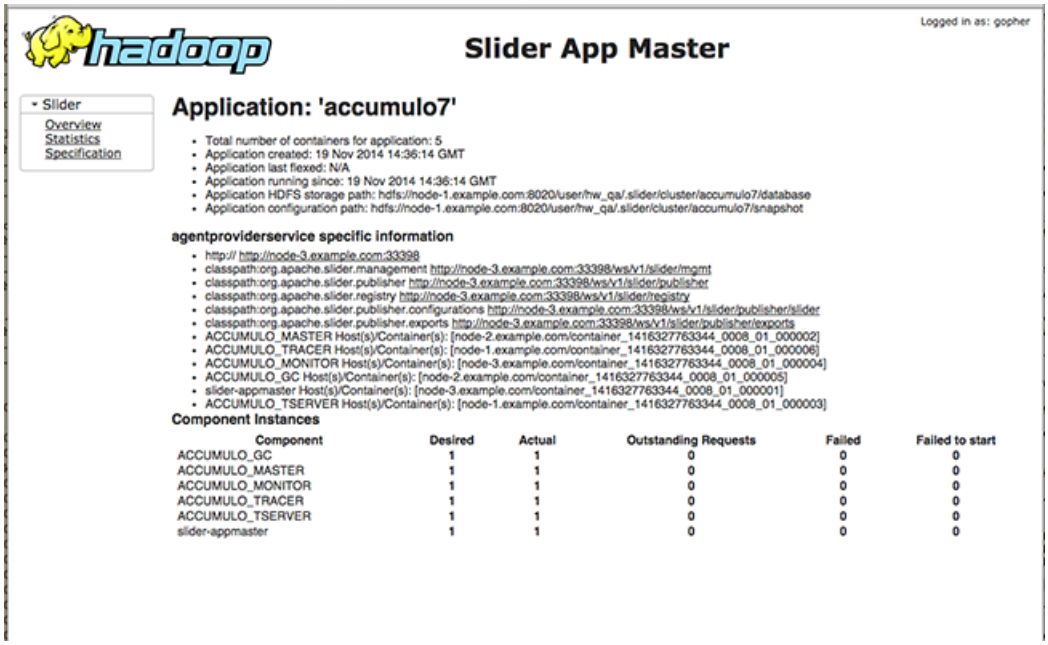
```
/usr/hdp/current/slider-client/bin/./slider list <accumulo_cluster_name>
```

The Accumulo instance name will have the format `<user name>-<accumulo_cluster_name>`.

The successful launch of the Accumulo cluster can also be verified via the YARN Resource Manager Web UI. This UI is usually accessible via a Web browser at port 8088 of the Resource Manager Host:



The specific information for the running Accumulo cluster is accessible via the "ApplicationMaster" link that can be seen in the far right column of the row associated with the running application (probably the top row):



Accesssing the Accumulo Monitor

On the Slider App Master page, one of the links should end with `publisher/slider`, for example:

```
http://node-1.example.com:52413/ws/v1/slider/publisher/slider
```

Add "quicklinks" to that URL, for example:

```
http://node-1.example.com:52413/ws/v1/slider/publisher/slider/
quicklinks
```

That page will list some .json code. The value of the "org.apache.slider.monitor" key is the URL of the Accumulo monitor page. If you go to the quicklinks page right after the application instance is created, it will not be ready right away, because the monitor has to come up and register itself. But if you keep reloading the quicklinks URL, it will come up eventually (if the cluster is healthy).

6.7.5. Client Connections to Accumulo and Retrieving Effective `accumulo-site.xml`

The most efficient method for obtaining a client for Accumulo would be to install the Accumulo `.rpm` (`yum install accumulo`) and the Accumulo configuration .rpm (`yum install accumulo-conf-standalone`). Then copy all of the Accumulo default configuration files from the `/usr/hdp/<version_number>/etc/accumulo/conf.dist/templates/` directory into your desired Accumulo configuration directory (e.g. `/etc/accumulo/conf`).

The next step is to obtain some configuration files from the registry for your Slider Accumulo cluster to enable your client to connect to the instance of Accumulo you created. Once Slider Accumulo is running, you can use the following commands to retrieve the

necessary configurations from the registry into the Accumulo `conf` directory you are setting up.

```
slider registry --getconf accumulo-site --name <cluster name> --format xml --
dest <accumulo_conf_dir>/accumulo-site.xml
slider registry --getconf client --name <cluster name> --format properties --
dest <accumulo_conf_dir>/client.conf
slider registry --getconf accumulo-env --name <cluster name> --format json --
dest accumulo-env.json
python -c "import json; file = open('accumulo-env.json'); content = json.
load(file); file.close(); print content['content']" > <accumulo_conf_dir>/
accumulo-env.sh
```

You should now be able to connect to Accumulo with the following command:

```
accumulo shell -u root
```

If there are multiple Slider Accumulo clusters running, you can connect to any of them from the same client installation. You still must download the configuration files from the registry for one running cluster, but you only need to do that once, and can then connect to other clusters with the following command:

```
accumulo shell -zh <zookeeper host(s)> -zi <instance name> -u root
```

Remember that the Accumulo instance name will be `<user name>-<slider cluster name>` if you are using the default setting in the `appConfig.json` file (which is strongly recommended).

6.7.6. Deployment Considerations

Memory Considerations for Running One Component on a Node

You can adjust the amount of memory given to a component to achieve mutual exclusion of components, depending upon the NodeManager configuration on each node. Typically all nodes have the same value independent of the actual memory.

Assuming the memory capacity for each NodeManager is known (`yarn.nodemanager.resource.memory-mb`), you can configure the component to ask for 51% (basically more than half) of the maximum capacity. You also need to ensure that the maximum possible memory allocation (`yarn.scheduler.maximum-allocation-mb`) allows that value.

For example, if `yarn.nodemanager.resource.memory-mb` = `yarn.scheduler.maximum-allocation-mb` = 2048 Set `yarn.memory` = 1280 for the ACCUMULO_MASTER and ACCUMULO_TSERVER properties in the `resources.json` file.

Then in the `appConfig.json` file, set the ACCUMULO_MASTER and ACCUMULO_TSERVER heap sizes (including the ACCUMULO_TSERVER off-heap memory properties, if native maps are enabled) to be 256 MB less than the memory requested for the YARN containers to allow for the agent memory consumption – the agent should not use more than 100 MB, but you can assume that it consumes ~256 MB. So you can set the ACCUMULO_MASTER and ACCUMULO_TSERVER variables for memory limit to fit within 1024 MB.

Log Aggregation

This feature is backed by https://issues.apache.org/jira/browse/YARN-2468.

Log aggregation is specified in the `yarn-site.xml` file. The `yarn.log-aggregation-enable` property enables log aggregation for running applications. If a monitoring interval is also set, it will aggregate logs while an application is running, with the specified interval. The minimum i nterval is 3600 seconds.

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

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

Log aggregation is specified in the `global` section of `resources.json`:

```
 "global": {
           "yarn.log.include.patterns": "",
           "yarn.log.exclude.patterns": ""
    },
```

If `yarn.log.include.patterns` is empty, all container logs are included. You can specify the name(s) of log files (for example, `agent.log`) that you do not want to aggregate using `yarn.log.exclude.patterns`.

The aggregated logs are stored in the HDFS `/app-logs/` directory. The following command can be used to retrieve the logs:

```
yarn logs -applicationId <app_id>
```

Reserving Nodes for Accumulo

You can use YARN node labels to reserve cluster nodes for applications and their components. You could use node labels to reserve cluster nodes for Accumulo to ensure that ACCUMULO_MASTER and ACCUMULO_TSERVER provide a consistent performance level.

Node labels are specified with the `yarn.label.expression` property. If no label is specified, only non-labeled nodes are used when allocating containers for component instances.

A brief summary is that you could label particular YARN nodes for Accumulo, say with labels “accumulo1” and “accumulo1_master”, and create a separate queue for assigning containers to these nodes. To use these labeled nodes, you would add `yarn.label.expression` parameters to the Accumulo components in your `resources.json` file (including the slider-appmaster), e.g. `"yarn.label.expression": "accumulo1_master"`. When you run the `slider create` command for your Accumulo cluster, you would add the parameter “`--queue <queue_name>`”.

Configuring Accumulo for SSL

Accumulo can be configured to use SSL (Secure Sockets Layer) for its internal RPC communications, and its monitor web UI can also be configured to use SSL. The Slider Accumulo application package is set up to use the same SSL certs for both, although the features can be enabled independently.

The SSL certificates must be provided. To distribute the certificates in HDFS, upload them to the directory specified in "`site.global.ssl_cert_dir`" (by default, `/user/<user name>/ssl`). There should be a `truststore.jks` file and a `.jks` file for each host named `<hostname>.jks`. You must add the passwords for the certificates to the credential provider by adding them to the list in the "credentials" section of the `appConfig.json` file as shown below. To turn on SSL, set the Accumulo SSL properties below to `true`. To turn on https for the monitor UI, change the `monitor_protocol` to `https`.

The properties are as follows:

```
"global": {
     "site.global.ssl_cert_dir": "ssl",
     "site.global.monitor_protocol": "http",
     "site.accumulo-site.instance.rpc.ssl.enabled": "false",
     "site.accumulo-site.instance.rpc.ssl.clientAuth": "false",
  },
  "credentials": {
     "jceks://hdfs/user/${USER}/accumulo-${CLUSTER_NAME}.jceks": ["root.
initial.password", "instance.secret", "trace.token.property.password"]
  },
```

Change these to:

```
"global": {
     "site.global.ssl_cert_dir": "ssl",
     "site.global.monitor_protocol": "https",
     "site.accumulo-site.instance.rpc.ssl.enabled": "true",
     "site.accumulo-site.instance.rpc.ssl.clientAuth": "true",
   },
   "credentials": {
      "jceks://hdfs/user/${USER}/accumulo-${CLUSTER_NAME}.jceks": ["root.
initial.password", "instance.secret", "trace.token.property.password", "rpc.
javax.net.ssl.keyStorePassword", "rpc.javax.net.ssl.trustStorePassword",
 "monitor.ssl.keyStorePassword", "monitor.ssl.trustStorePassword"]
   },
```

If you would like to distribute the certs yourself rather than through HDFS, simply set the following properties to the locations of the `.jks` files in the local files ystem (the keystore file should have the same name on all hosts for this configuration).

```
"global": {
   "site.accumulo-site.rpc.javax.net.ssl.keyStore": "<keystore file>",
   "site.accumulo-site.rpc.javax.net.ssl.trustStore": "<truststore file>",
  },
```

If `clientAuth` is enabled, you must have a `client.conf` file in your client Accumulo conf directory, or a `.accumulo/config` file in your home directory. Your `keystore.jks` and `truststore.jks` SSL certs for the client can be

placed in an ssl directory in the Accumulo `conf` directory (or their locations can be specified by also specifying the `rpc.javax.net.ssl.keyStore` and `rpc.javax.net.ssl.trustStore` properties). If the client user is the same user that created the Accumulo cluster, it can use the same credential provider as the `app, jceks://hdfs/user/<user name>/accumulo-<cluster name>.jceks`, but otherwise the client user will have to create their own credential provider using the hadoop credential command. The user must set the credential provider in their `client.conf` file, and make sure the specified credential provider contains the `rpc.javax.net.ssl.keyStorePassword` and `rpc.javax.net.ssl.trustStorePassword`.

A `client.conf` file for the Accumulo instance can be retrieved with the following command:

```
slider registry --getconf client --name <cluster name> --format properties --
dest <path to local client.conf>
```

Building Accumulo Native Libraries

Accumulo performs better when it uses a native in-memory map for newly written data. To build the native libraries for your system, perform the following steps to add them to your application package. Then set the "`site.accumulo-site.tserver.memory.maps.native.enabled`" property to `true` in your `appConfig.json` file, and be sure to adjust the ACCUMULO_TSERVER heapsize parameter so that it no longer includes the `tserver.memory.maps.max` memory.

```
unzip <app package name>.zip package/files/accumulo*gz
cd package/files/
gunzip accumulo-<version>-bin.tar.gz
tar xvf accumulo-<version>-bin.tar
accumulo-1.6.0/bin/build_native_library.sh
tar uvf accumulo-<version>-bin.tar accumulo-<version>
rm -rf accumulo-<version>
gzip accumulo-<version>-bin.tar
cd ../../
zip <app package name>.zip -r package
rm -rf package
```

7. Running Multiple MapReduce Versions Using the YARN Distributed Cache

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 lifecycle. 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.

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, `mr-framework` is specified as the alias:

```
<property>
 <name>mapreduce.application.framework.path</name>
 <value>hdfs:/mapred/framework/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 the following exception:

```
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.

8. Timeline Server

This chapter describes how to configure and run the Timeline Server, which 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 via REST APIs that enable rendering by application/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.

8.1. Configuring the Timeline Server

Required Properties

Only one property needs to be specified in the `etc/hadoop/conf/yarn-site.xml` file in order to enable the Timeline Server: `yarn.timeline-service.hostname`. The host name of the Timeline Server web application.

Example:

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

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>
```

8.2. Enabling Generic Data Collection

- **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>
```

8.3. Configuring Per-Framework Data Collection

- **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>
```

8.4. Configuring the Timeline Server Store

- **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>
```

8.5. Configuring Timeline Server Security

Configuring 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>
```

Configuring 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>
```

Configuring 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>
```

Note

You should also configure YARN SSL settings.

8.6. Running the Timeline Server

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
```

8.7. Accessing Generic Data from the Command-Line

You can use the following commands to access application generic history data from the command-line. Note that these same commands can be used 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>
```

8.8. Publishing 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 via `TimelineClient`. For example:

```
// 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();
```

9. Using the YARN REST APIs to Manage Applications

This chapter describes how to use the YARN REST APIs to submit, monitor, and kill applications.

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 submitting 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'
```

Access the Apache YARN REST API Specification

For more information, see the Apache YARN REST APIs documentation.

10. Work-Preserving Restart

This chapter describes how to 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.

10.1. Configuring the ResourceManager for 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 pecified 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 is `world:anyone:rwcda`. **Example**

```
<property>
    <name>yarn.resourcemanager.zk-acl</name>
    <value>world:anyone:rwcda</value>
</property>
```

10.2. Configuring NodeManagers for Work-Preserving Restart

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.

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>
```