Cloudera Runtime 7.0.1

Managing Applications on Apache Hadhoop YARN

Date published: 2019-09-23
Date modified:

https://docs.cloudera.com/

Legal Notice

Contents

Application development

You can use the YARN REST APIs to manage applications. .

Use the YARN REST APIs to manage applications

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

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

Example: Get application data

- Without ?user.name=<user>:

```
curl http://localhost:19888/jobhistory/job/job_1516861688424_0001
Access denied: User null does not have permission to view job job_
1516861688424_0001
```

- With ?user.name=<user>:

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

Get an Application ID

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

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

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

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

Set Up an Application .json File

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

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

```
{
   "application-id":"application_1404203615263_0001",
   "application-name":"test",
   "am-container-spec":
   {
      "local-resources":
```

```
       {
          "entry":
         [
            {
               "key":"AppMaster.jar",
               "value":
               {
                  "resource":"hdfs://hdfs-namenode:9000/user/testuser/Dis
tributedShell/demo-app/AppMaster.jar",
                  "type":"FILE",
                  "visibility":"APPLICATION",
                  "size": "43004",
                  "timestamp": "1405452071209"
               }
            }
         ]
       },
       "commands":
       {
          "command":"{{JAVA_HOME}}/bin/java -Xmx10m org.apache.hadoop.yar
n.applications.distributedshell.ApplicationMaster --container_memory 10 --co
ntainer_vcores 1 --num_containers 1 --priority 0 1><LOG_DIR>/AppMaster.stdou
t 2><LOG_DIR>/AppMaster.stderr"
       },
       "environment":
       {
          "entry":
          [
             {
                "key": "DISTRIBUTEDSHELLSCRIPTTIMESTAMP",
                "value": "1405459400754"
             },
             {
                "key": "CLASSPATH",
                "value": "{{CLASSPATH}}<CPS>./*<CPS>{{HADOOP_CONF_DIR}}<C
PS>{{HADOOP_COMMON_HOME}}/share/hadoop/common/*<CPS>{{HADOOP_COMMON_HOME}}/s
hare/hadoop/common/lib/*<CPS>{{HADOOP_HDFS_HOME}}/share/hadoop/hdfs/*<CPS>{{
HADOOP_HDFS_HOME}}/share/hadoop/hdfs/lib/*<CPS>{{HADOOP_YARN_HOME}}/share/ha
doop/yarn/*<CPS>{{HADOOP_YARN_HOME}}/share/hadoop/yarn/lib/*<CPS>./log4j.pro
perties"
             },
             {
                "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/js
on"'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_001
2/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 -H 'Accept: application/json' -H 'Content-Type: application/j
son' -d '{"state": "KILLED"}' 'http://localhost:8088/ws/v1/cluster/apps/appl
ication_1409421698529_0012/state'
```

Managing applications

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

Manage long-running YARN applications

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

About this task
You can use the YARN Services API to deploy and manage the YARN services.

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

Related Information
Configure YARN for long-running applications

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

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

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

Configure YARN Services using Cloudera Manager
You can enable and configure the YARN Services feature using Cloudera Manager.

About this task

YARN Services is enabled by default to ensure that any program that is dependent on it, for example Hive LLAP, can be installed. However you can disable it using Cloudera Manager.

Before you begin

If you want to actively use the YARN Services feature, Cloudera recommends to use Capacity Scheduler, which is the default scheduler, as only that scheduler type can fully support this feature.

Procedure

1. In Cloudera Manager, select the YARN service.
2. Click the Configuration tab.
3. Select the YARN Services Management filter.
4. Ensure that Enable YARN Services is checked.
5. Configure the YARN Services Dependencies Path to specify the path where the YARN services dependencies tarball is uploaded.

   Cloudera recommends using the default path:

   ```
   /user/yarn/services/service-framework/${cdhVersion}/service-dep.tar.gz
   ```
6. Click Save Changes.

If you changed the YARN Services Dependencies path, do the following:

7. Click the Actions button.
8. Select Install YARN Services Dependencies.

9. Confirm that you want to run the Install YARN Servioces Dependencies comment by clicking the Install YARN Services Dependencies button.
10. Once the command is run successfully, close the status window.
11. Click the *Stale Service Restart* icon that is next to the service to invoke the cluster restart wizard.
12. Click Restart Stale Services.
13. Select Re-deploy client configuration.
14. Click Restart Now.

Deploying and Managing Services and Microservices on YARN

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

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

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

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

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

Launch the Service YARN file

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

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

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

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

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

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

Save the YARN file as Service App

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

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

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

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

Managing the YARN service life cycle through the REST API

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

Create a service

Use the following endpoint to create a service:

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

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

Update a service or upgrade the binary version of a service

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

Use the following endpoint to update the service:

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

Destroy a service

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

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

Get the details of a service

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

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

Set the number of instances of a component

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

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

YARN Services API Examples

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

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

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

  POST Request JSON

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

  GET Response JSON

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

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

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

```
                    "ip": "10.22.8.143",
                    "hostname": "myhost.local",
                    "state": "READY",
                    "launch_time": 1504051536450,
                    "bare_host": "10.22.8.143",
                    "component_name": "hello-1"
                }
            ],
            "launch_command": "./start_nginx.sh",
            "number_of_containers": 1,
            "run_privileged_container": false
        }
    ],
    "configuration": {
        "properties": {},
        "env": {},
        "files": []
    },
    "quicklinks": {}
}
```

- Update the lifetime of a service

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

  PUT Request JSON

  Note that irrespective of what the current lifetime value is, this update request will set the lifetime of the service to 3600 seconds (1 hour) from the time the request is submitted. Therefore, if a service has remaining lifetime of 5 minutes (for example) and would like to extend it to an hour, OR if an application has remaining lifetime of 5 hours (for example) and would like to reduce it down to one hour, then for both scenarios you would need to submit the following request.

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

- Stop a service

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

  PUT Request JSON

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

- Start a service

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

  PUT Request JSON

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

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

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

  PUT Request JSON

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

```
}
```

- Destroy a service

  DELETE URL – http://localhost:8088/app/v1/services/hello-world
- Create a complicated service – HBase

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

```
{
  "name": "hbase-app-1",
  "lifetime": "3600",
  "version": "2.0.0.3.0.0.0",
  "artifact": {
    "id": "hbase:2.0.0.3.0.0.0",
    "type": "DOCKER"
  },
  "configuration": {
    "env": {
      "HBASE_LOG_DIR": "<LOG_DIR>"
    },
    "files": [
      {
        "type": "TEMPLATE",
        "dest_file": "/etc/hadoop/conf/core-site.xml",
        "src_file": "core-site.xml"
      },
      {
        "type": "TEMPLATE",
        "dest_file": "/etc/hadoop/conf/hdfs-site.xml",
        "src_file": "hdfs-site.xml"
      },
      {
        "type": "XML",
        "dest_file": "/etc/hbase/conf/hbase-site.xml",
        "properties": {
          "hbase.cluster.distributed": "true",
          "hbase.zookeeper.quorum": "${CLUSTER_ZK_QUORUM}",
          "hbase.rootdir": "${SERVICE_HDFS_DIR}/hbase",
          "zookeeper.znode.parent": "${SERVICE_ZK_PATH}",
          "hbase.master.hostname": "hbasemaster-0.${SERVICE_NAME}.${USER}.
${DOMAIN}",
          "hbase.master.info.port": "16010"
        }
      }
    ]
  },
  "components": [
    {
      "name": "hbasemaster",
      "number_of_containers": 1,
      "launch_command": "sleep 15;/opt/cloudera/parcels/CDH-<version>/bin/
hbase master start",
      "resource": {
        "cpus": 1,
        "memory": "2048"
      },
      "configuration": {
        "env": {
          "HBASE_MASTER_OPTS": "-Xmx2048m -Xms1024m"
        }
      }
    },
```

```
    {
      "name": "regionserver",
      "number_of_containers": 1,
      "launch_command": "sleep 15; /opt/cloudera/parcels/CDH-<version>/bi
n/hbase master start",
      "dependencies": [
        "hbasemaster"
      ],
      "resource": {
        "cpus": 1,
        "memory": "2048"
      },
      "configuration": {
        "files": [
          {
            "type": "XML",
            "dest_file": "/etc/hbase/conf/hbase-site.xml",
            "properties": {
              "hbase.regionserver.hostname": "${COMPONENT_INSTANCE_NAME}.
${SERVICE_NAME}.${USER}.${DOMAIN}"
            }
          }
        ],
        "env": {
          "HBASE_REGIONSERVER_OPTS": "-XX:CMSInitiatingOccupancyFraction=
70 -Xmx2048m -Xms1024m"
        }
      }
    },
    {
      "name": "hbaseclient",
      "number_of_containers": 1,
      "launch_command": "sleep infinity",
      "resource": {
        "cpus": 1,
        "memory": "1024"
      }
    }
  ],
  "quicklinks": {
    "HBase Master Status UI": "http://hbasemaster-0.${SERVICE_NAME}.${US
ER}.${DOMAIN}:16010/master-status"
  }
}
```

Configure Cross-Origin Support on YARN

Cross-Origin Resource Sharing (CORS) is enabled by default on YARN so that the corresponding services accept cross-origin requests from only selected domains. Enabling CORS also helps the YARN UI fetch data endpoints from the browser.

About this task

CORS is enabled at cluster level and then on individual components such as YARN. If you want to set other values than the default ones, use safety valves in Cloudera Manager to configure the CORS configuration properties.

Procedure

1. In the Cloudera Manager, select the YARN service.
2. Click the Configuration tab.

3. Search for core-site.xml.
4. Find YARN Service Advanced Configuration Snippet (Safety Valve) for core-site.xml.
5. Click on the plus icon to add and configure the following properties :

Property	Default value	Description
hadoop.http.cross-origin.allowed-origins	regex:.*[.]subdomain[.]*example*[.]com(;\d+)?.*	Comma-separated list of origins allowed for cross-origin support. The default value is *. Mention only specific origins so that the services do not accept all the cross-origin requests.
hadoop.http.cross-origin.allowed-methods	GET, PUT, POST, OPTIONS, HEAD, DELETE	Comma-separated list of methods allowed for cross-origin support.
hadoop.http.cross-origin.allowed-headers	X-Requested-With, Content-Type, Accept, Origin, WWW-Authenticate, Accept-Encoding, Transfer-Encoding	Comma-separated list of headers allowed for cross-origin support.
hadoop.http.cross-origin.max-age	180	Number of seconds until when a preflight request can be cached.

6. Click Save Changes.
7. In the Cloudera Manager, select the YARN service.
8. Click the Configuration tab.
9. Search for yarn-site.xml.
10. Find the YARN Service Advanced Configuration Snippet (Safety Valve) for yarn-site.xml.
11. Click on the plus icon to add and configure the cross-origin.enabled properties:

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

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

12. Click Save Changes.
13. Restart the HDFS and YARN services.