Cloudera Runtime 7.2.6

Managing Apache Hadoop YARN Services

Date published: 2020-02-11
Date modified: 2020-12-01

https://docs.cloudera.com/

Legal Notice

Contents

Configure YARN Services API to Manage Long-running 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

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.

Running YARN Services

You can use YARN Services API to manage YARN services. 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.

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.

Deploy and manage services 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 a YARN service

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 a YARN service definition

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.

Create new YARN services using UI

The YARN Web User Interface enables you to define new services. You can either create standard services by providing their details or custom services by using JSON files containing the required definitions.

Create a standard YARN service

You can create a standard service as a template and deploy it based on your requirements.

Procedure

1. On the Services page of the YARN Web User Interface, click New Service.
2. In the User name for service field, specify the name of the user who launches the service.
3. Enter the service definition details.
   - Service Name: Enter a unique name for the application.
   - Queue Name: Enter the name of the YARN queue to which this application should belong.
   - Service Lifetime: Life time of the application from the time it is in STARTED state till the time it is automatically destroyed by YARN. If you want to have unlimited life time, do not enter any value.
   - Service Components: Enter the details of the service components such as component name, CPU required, memory, number of containers, artifact ID, and launch command. If it is an application like HBase, the

components can be a simple role like master or RegionServer. For a complex application, the components can be other nested applications with their own details.

- Service Configurations: Set of configuration properties that can be ingested into the application components through environments, files, and custom pluggable helper docker containers. You can upload files of several formats such as properties files, JSON files, XML files, YAML files, and template files.
- File Configurations: Set of file configurations that needs to be created and made available as a volume in an application component container. You can upload JSON file configurations to add to the service.

4. Click Save.
5. Specify a name for the new service and click Add.

   The newly created service is added to the list of saved templates.

   Note: Click Reset if you do not want to save your changes and specify the service details again.

6. Select the service and click Deploy to deploy it.

Create a custom YARN service

You can define a service in JSON format and save it as a template.

Procedure

1. On the Services page of the YARN Web User Interface, click New Service.
2. Click the Custom tab.
3. In the User name for service field, specify the name of the user who launches the service.
4. In the Service Definition field, specify the service definition in JSON format.

   The following example shows the sleeper service template definition.

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

5. Click Save.
6. Specify a name for the new service and click Add.

   The newly created service is added to the list of saved templates.

   Note: Click Reset if you do not want to save your changes and specify the service details again.

7. Select the service and click Deploy to deploy it.

Manage 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"
  }
}
```