Overview

Background

The Stack definitions can be found in the source tree at `/ambari-server/src/main/resources/stacks`. After you install the Ambari Server, the Stack definitions can be found at `/var/lib/ambari-server/resources/stacks`

Structure

The structure of a Stack definition is as follows:

```
|_ stacks
   |_ <stack_name>
      |_ <stack_version>
         metainfo.xml
         |_ hooks
         |_ repos
            repoinfo.xml
         |_ services
            |_ <service_name>
               metainfo.xml
               metrics.json
               |_ configuration
                  {configuration files}
               |_ package
                  {files, scripts, templates}
```

Defining a Service and Components

The `metainfo.xml` file in a Service describes the service, the components of the service and the management scripts to use for executing commands. A component of a service can be either a **MASTER**, **SLAVE** or **CLIENT** category. The <category> tells Ambari what default commands should be available to manage and monitor the component.

For each Component you specify the <commandScript> to use when executing commands. There is a defined set of default commands the component must support.

Component Category	Default Lifecycle Commands
MASTER	install, start, stop, configure, status
SLAVE	install, start, stop, configure, status
CLIENT	install, configure, status

Ambari supports different commands scripts written in **PYTHON**. The type is used to know how to execute the command scripts. You can also create **custom commands** if there are other commands beyond the default lifecycle commands your component needs to support.

For example, in the YARN Service describes the ResourceManager component as follows in `metainfo.xml`:

```
        <component>
          <name>RESOURCEMANAGER</name>
          <category>MASTER</category>
          <commandScript>
            <script>scripts/resourcemanager.py</script>
            <scriptType>PYTHON</scriptType>
            <timeout>600</timeout>
          </commandScript>
          <customCommands>
            <customCommand>
              <name>DECOMMISSION</name>
              <commandScript>
                <script>scripts/resourcemanager.py</script>
                <scriptType>PYTHON</scriptType>
                <timeout>600</timeout>
              </commandScript>
            </customCommand>
          </customCommands>
        </component>
```

The ResourceManager is a MASTER component, and the command script is `scripts/resourcemanager.py`, which can be found in the `services/YARN/package` directory. That command script is **PYTHON** and that script implements the default lifecycle commands as python methods. This is the **install** method for the default **INSTALL** command:

```
class Resourcemanager(Script):
  def install(self, env):
    self.install_packages(env)
    self.configure(env)
```

You can also see a custom command is defined **DECOMMISSION**, which means there is also a **decommission** method in that python command script:

```
  def decommission(self, env):
    import params

    ...

    Execute(yarn_refresh_cmd,
            user=yarn_user
    )
    pass
```

Using Stack Inheritance

Stacks can *extend* other Stacks in order to share command scripts and configurations. This reduces duplication of code across Stacks with the following:

- define repositories for the child Stack
- add new Services in the child Stack (not in the parent Stack)
- override command scripts of the parent Services
- override configurations of the parent Services

For example, the **HDP 2.1 Stack *extends* HDP 2.0.6 Stack** so only the changes applicable to **HDP 2.1 Stack** are present in that Stack definition. This extension is defined in the metainfo.xml for HDP 2.1 Stack:

```
<metainfo>
  <versions>
    <active>true</active>
  </versions>
  <extends>2.0.6</extends>
</metainfo>
```

Example: Implementing a Custom Service

In this example, we will create a custom service called "SAMPLESRV", add it to an existing Stack definition. This service includes MASTER, SLAVE and CLIENT components.

Create and Add the Service

1. On the Ambari Server, browse to the `/var/lib/ambari-server/resources/stacks/HDP/2.0.6/services` directory. In this case, we will browse to the HDP 2.0 Stack definition.

   ```
   cd /var/lib/ambari-server/resources/stacks/HDP/2.0.6/services
   ```

2. Create a directory named `/var/lib/ambari-server/resources/stacks/HDP/2.0.6/services/`**`SAMPLESRV`** that will contain the service definition for **SAMPLESRV**.

   ```
   mkdir /var/lib/ambari-server/resources/stacks/HDP/2.0.6/services/SAMPLESRV
   cd /var/lib/ambari-server/resources/stacks/HDP/2.0.6/services/SAMPLESRV
   ```

3. Browse to the newly created `SAMPLESRV` directory, create a `metainfo.xml` file that describes the new service. For example:

```
<?xml version="1.0"?>
<metainfo>
    <schemaVersion>2.0</schemaVersion>
    <services>
        <service>
            <name>SAMPLESRV</name>
            <displayName>New Sample Service</displayName>
            <comment>A New Sample Service</comment>
            <version>1.0.0</version>
            <components>
                <component>
                    <name>SAMPLESRV_MASTER</name>
                    <displayName>Sample Srv Master</displayName>
                    <category>MASTER</category>
                    <cardinality>1</cardinality>
                    <commandScript>
                        <script>scripts/master.py</script>
                        <scriptType>PYTHON</scriptType>
                        <timeout>600</timeout>
                    </commandScript>
                </component>
                <component>
                    <name>SAMPLESRV_SLAVE</name>
                    <displayName>Sample Srv Slave</displayName>
                    <category>SLAVE</category>
                    <cardinality>1+</cardinality>
                    <commandScript>
                        <script>scripts/slave.py</script>
                        <scriptType>PYTHON</scriptType>
                        <timeout>600</timeout>
                    </commandScript>
                </component>
                <component>
                    <name>SAMPLESRV_CLIENT</name>
                    <displayName>Sample Srv Client</displayName>
                    <category>CLIENT</category>
                    <cardinality>1+</cardinality>
                    <commandScript>
                        <script>scripts/sample_client.py</script>
                        <scriptType>PYTHON</scriptType>
                        <timeout>600</timeout>
                    </commandScript>
                </component>
            </components>
            <osSpecifics>
                <osSpecific>
                    <osFamily>any</osFamily>  <!-- note: use osType rather than osFamily for Ambari
1.5.0 and 1.5.1 -->
                </osSpecific>
            </osSpecifics>
        </service>
    </services>
</metainfo>
```

4. In the above, my service name is "**SAMPLESRV**", and it contains:
    - one **MASTER** component "**SAMPLESRV_MASTER**"
    - one **SLAVE** component "**SAMPLESRV_SLAVE**"
    - one **CLIENT** component "**SAMPLESRV_CLIENT**"
5. Next, let's create that command script. Create a directory for the command script `/var/lib/ambari-server/resources/stacks/HDP/2.0.6/services/SAMPLESRV/`**`package/scripts`** that we designated in the service metainfo.

```
mkdir -p /var/lib/ambari-server/resources/stacks/HDP/2.0.6/services/SAMPLESRV/package/scripts
cd /var/lib/ambari-server/resources/stacks/HDP/2.0.6/services/SAMPLESRV/package/scripts
```

6. Browse to the scripts directory and create the `.py` command script files.

   For example `master.py` file:

```
import sys
from resource_management import *
class Master(Script):
  def install(self, env):
    print 'Install the Sample Srv Master';
  def stop(self, env):
    print 'Stop the Sample Srv Master';
  def start(self, env):
    print 'Start the Sample Srv Master';

  def status(self, env):
    print 'Status of the Sample Srv Master';
  def configure(self, env):
    print 'Configure the Sample Srv Master';
if __name__ == "__main__":
  Master().execute()
```

For example `slave.py` file:

```
import sys
from resource_management import *
class Slave(Script):
  def install(self, env):
    print 'Install the Sample Srv Slave';
  def stop(self, env):
    print 'Stop the Sample Srv Slave';
  def start(self, env):
    print 'Start the Sample Srv Slave';
  def status(self, env):
    print 'Status of the Sample Srv Slave';
  def configure(self, env):
    print 'Configure the Sample Srv Slave';
if __name__ == "__main__":
  Slave().execute()
```

For example `sample_client.py` file:

```
import sys
from resource_management import *
class SampleClient(Script):
  def install(self, env):
    print 'Install the Sample Srv Client';
  def configure(self, env):
    print 'Configure the Sample Srv Client';
if __name__ == "__main__":
  SampleClient().execute()
```

7. Now, restart Ambari Server for this new service definition to be distributed to all the Agents in the cluster.

```
ambari-server restart
```

Install the Service (via Ambari Web "Add Services")

> ⚠ The ability to add custom services via Ambari Web is new as of Ambari 1.7.0.

1. In Ambari Web, browse to Services and click the **Actions** button in the Service navigation area on the left.
2. The "Add Services" wizard launches. You will see an option to include "My Sample Service" (which is the `<displayName>` of the service as defined in the service `metainfo.xml` file).
3. Select "My Sample Service" and click Next.
4. Assign the "Sample Srv Master" and click Next.
5. Select the hosts to install the "Sample Srv Client" and click Next.
6. Once complete, the "My Sample Service" will be available Service navigation area.
7. If you want to add the "Sample Srv Client" to any hosts, you can browse to Hosts and navigate to a specific host and click "+ Add".

Example: Implementing a Custom Client-only Service

In this example, we will create a custom service called "TESTSRV", add it to an existing Stack definition and use the Ambari APIs to install/configure the service. This service is a CLIENT so it has two commands: install and configure.

Create and Add the Service

1. On the Ambari Server, browse to the `/var/lib/ambari-server/resources/stacks/HDP/2.0.6/services` directory. In this case, we will browse to the HDP 2.0 Stack definition.

```
cd /var/lib/ambari-server/resources/stacks/HDP/2.0.6/services
```

2. Create a directory named `/var/lib/ambari-server/resources/stacks/HDP/2.0.6/services/`**TESTSRV** that will contain the service definition for **TESTSRV**.

```
mkdir /var/lib/ambari-server/resources/stacks/HDP/2.0.6/services/TESTSRV
cd /var/lib/ambari-server/resources/stacks/HDP/2.0.6/services/TESTSRV
```

3. Browse to the newly created `TESTSRV` directory, create a `metainfo.xml` file that describes the new service. For example:

```
<?xml version="1.0"?>
<metainfo>
    <schemaVersion>2.0</schemaVersion>
    <services>
        <service>
            <name>TESTSRV</name>
            <displayName>New Test Service</displayName>
            <comment>A New Test Service</comment>
            <version>0.1.0</version>
            <components>
                <component>
                    <name>TEST_CLIENT</name>
                    <displayName>New Test Client</displayName>
                    <category>CLIENT</category>
                    <cardinality>1+</cardinality>
                    <commandScript>
                        <script>scripts/test_client.py</script>
                        <scriptType>PYTHON</scriptType>
                        <timeout>600</timeout>
                    </commandScript>
                    <customCommands>
                      <customCommand>
                        <name>SOMETHINGCUSTOM</name>
                        <commandScript>
                          <script>scripts/test_client.py</script>
                          <scriptType>PYTHON</scriptType>
                          <timeout>600</timeout>
                        </commandScript>
                      </customCommand>
                    </customCommands>
                </component>
            </components>
            <osSpecifics>
                <osSpecific>
                    <osFamily>any</osFamily>  <!-- note: use osType rather than osFamily for Ambari
1.5.0 and 1.5.1 -->
                </osSpecific>
            </osSpecifics>
        </service>
    </services>
</metainfo>
```

4. In the above, my service name is "**TESTSRV**", and it contains one component "**TEST_CLIENT**" that is of component category "**CLIENT**". That client is managed via the command script `scripts/test_client.py`. Next, let's create that command script.
5. Create a directory for the command script `/var/lib/ambari-server/resources/stacks/HDP/2.0.6/services/TESTSRV/`**`package/scripts`** that we designated in the service metainfo.

```
mkdir -p /var/lib/ambari-server/resources/stacks/HDP/2.0.6/services/TESTSRV/package/scripts
cd /var/lib/ambari-server/resources/stacks/HDP/2.0.6/services/TESTSRV/package/scripts
```

6. Browse to the scripts directory and create the `test_client.py` file. For example:

```
import sys
from resource_management import *

class TestClient(Script):
  def install(self, env):
    print 'Install the client';
  def configure(self, env):
    print 'Configure the client';
  def somethingcustom(self, env):
    print 'Something custom';

if __name__ == "__main__":
  TestClient().execute()
```

7. Now, restart Ambari Server for this new service definition to be distributed to all the Agents in the cluster.

```
ambari-server restart
```

Install the Service (via the Ambari REST API)

1. Add the Service to the Cluster.

```
POST
/api/v1/clusters/MyCluster/services

{
"ServiceInfo": {
  "service_name":"TESTSRV"
  }
}
```

2. Add the Components to the Service. In this case, add TEST_CLIENT to TESTSRV.

```
POST
/api/v1/clusters/MyCluster/services/TESTSRV/components/TEST_CLIENT
```

3. Install the component on all target hosts. For example, to install on `c6402.ambari.apache.org` and `c6403.ambari.apache.org`, first create the host_component resource on the hosts using POST.

```
POST
/api/v1/clusters/MyCluster/hosts/c6402.ambari.apache.org/host_components/TEST_CLIENT

POST
/api/v1/clusters/MyCluster/hosts/c6403.ambari.apache.org/host_components/TEST_CLIENT
```

4. Now have Ambari install the components on all hosts. In this single command, you are instructing Ambari to install all components related to the service. This call the `install()` method in the command script on each host.

```
PUT
/api/v1/clusters/MyCluster/services/TESTSRV

{
  "RequestInfo": {
    "context": "Install Test Srv Client"
  },
  "Body": {
    "ServiceInfo": {
      "state": "INSTALLED"
    }
  }
}
```

5. Alternatively, instead of installing all components at the same time, you can explicitly install each host component. In this example, we will explicitly install the TEST_CLIENT on `c6402.ambari.apache.org`:

```
PUT
/api/v1/clusters/MyCluster/hosts/c6402.ambari.apache.org/host_components/TEST_CLIENT

{
  "RequestInfo": {
    "context":"Install Test Srv Client"
  },
  "Body": {
    "HostRoles": {
      "state":"INSTALLED"
    }
  }
}
```

6. Use the following to configure the client on the host. This will end up calling the `configure()` method in the command script.

```
POST
/api/v1/clusters/MyCluster/requests

{
  "RequestInfo" : {
    "command" : "CONFIGURE",
    "context" : "Config Test Srv Client"
  },
  "Requests/resource_filters": [{
    "service_name" : "TESTSRV",
    "component_name" : "TEST_CLIENT",
    "hosts" : "c6403.ambari.apache.org"
  }]
}
```

7. If you want to see which hosts the component is installed.

```
GET
/api/v1/clusters/MyCluster/components/TEST_CLIENT
```

Install the Service (via Ambari Web "Add Services")

The ability to add custom services via Ambari Web is new as of Ambari 1.7.0.

1. In Ambari Web, browse to Services and click the **Actions** button in the Service navigation area on the left.
2. The "Add Services" wizard launches. You will see an option to include "My Test Service" (which is the `<displayName>` of the service as defined in the service `metainfo.xml` file).
3. Select "My Test Service" and click Next.
4. Select the hosts to install the "New Test Client" and click Next.

5. Once complete, the "My Test Service" will be available Service navigation area.
6. If you want to add the "New Test Client" to any hosts, you can browse to Hosts and navigate to a specific host and click "+ Add".

Example: Implementing a Custom Client-only Service (with Configs)

In this example, we will create a custom service called "TESTCONFIGSRV" and add it to an existing Stack definition. This service is a CLIENT so it has two commands: install and configure. And the service also includes a configuration type "test-config".

Create and Add the Service to the Stack

1. On the Ambari Server, browse to the `/var/lib/ambari-server/resources/stacks/HDP/2.0.6/services` directory. In this case, we will browse to the HDP 2.0 Stack definition.

```
cd /var/lib/ambari-server/resources/stacks/HDP/2.0.6/services
```

2. Create a directory named `/var/lib/ambari-server/resources/stacks/HDP/2.0.6/services/`**`TESTCONFIGSRV`** that will contain the service definition for TESTCONFIGSRV.

```
mkdir /var/lib/ambari-server/resources/stacks/HDP/2.0.6/services/TESTCONFIGSRV
cd /var/lib/ambari-server/resources/stacks/HDP/2.0.6/services/TESTCONFIGSRV
```

3. Browse to the newly created `TESTCONFIGSRV` directory, create a `metainfo.xml` file that describes the new service. For example:

```
<?xml version="1.0"?>
<metainfo>
    <schemaVersion>2.0</schemaVersion>
    <services>
        <service>
            <name>TESTCONFIGSRV</name>
            <displayName>New Test Config Service</displayName>
            <comment>A New Test Config Service</comment>
            <version>0.1.0</version>
            <components>
                <component>
                    <name>TESTCONFIG_CLIENT</name>
                    <displayName>New Test Config Client</displayName>
                    <category>CLIENT</category>
                    <cardinality>1+</cardinality>
                    <commandScript>
                        <script>scripts/test_client.py</script>
                        <scriptType>PYTHON</scriptType>
                        <timeout>600</timeout>
                    </commandScript>
                </component>
            </components>
            <osSpecifics>
                <osSpecific>
                    <osFamily>any</osFamily>  <!-- note: use osType rather than osFamily for Ambari
1.5.0 and 1.5.1 -->
                </osSpecific>
            </osSpecifics>
        </service>
    </services>
</metainfo>
```

4. In the above, my service name is "**TESTCONFIGSRV**", and it contains one component "**TESTCONFIG_CLIENT**" that is of component category "**CLIENT**". That client is managed via the command script `scripts/test_client.py`. Next, let's create that command script.
5. Create a directory for the command script `/var/lib/ambari-server/resources/stacks/HDP/2.0.6/services/TESTCONFIGSRV/`**`package/scripts`** that we designated in the service metainfo `<commandScript>`.

```
mkdir -p /var/lib/ambari-server/resources/stacks/HDP/2.0.6/services/TESTCONFIGSRV/package/scripts
cd /var/lib/ambari-server/resources/stacks/HDP/2.0.6/services/TESTCONFIGSRV/package/scripts
```

6. Browse to the scripts directory and create the `test_client.py` file. For example:

```
import sys
from resource_management import *

class TestClient(Script):
  def install(self, env):
    print 'Install the config client';
  def configure(self, env):
    print 'Configure the config client';

if __name__ == "__main__":
  TestClient().execute()
```

7. Now let's define a config type for this service. Create a directory for the configuration dictionary file `/var/lib/ambari-server/resources/stacks/HDP/2.0.6/services/TESTCONFIGSRV/`**`configuration`**.

```
mkdir -p /var/lib/ambari-server/resources/stacks/HDP/2.0.6/services/TESTCONFIGSRV/configuration
cd /var/lib/ambari-server/resources/stacks/HDP/2.0.6/services/TESTCONFIGSRV/configuration
```

8. Browse to the configuration directory and create the `test-config.xml` file. For example:

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>

<configuration>
  <property>
    <name>some.test.property</name>
    <value>this.is.the.default.value</value>
    <description>This is a kool description.</description>
 </property>
</configuration>
```

9. Now, restart Ambari Server for this new service definition to be distributed to all the Agents in the cluster.

```
ambari-server restart
```