Dynamically Loading Auxiliary Services

Billie Rinaldi, Wangda Tan, Vinod Kumar Vavilapalli

Use Cases

NM restart should not be needed when changing auxiliary services. In order of importance, the use cases we would like to support are:

1. Dynamically adding a new auxiliary service on all NMs
2. Dynamically stopping, draining and deleting an existing auxiliary service on all NMs
3. Dynamically updating an existing auxiliary service with a new version on all NMs

Supporting out-of-process auxiliary services (YARN-1593) is not a goal for this JIRA.

Implementation notes

Auxiliary Services will be loaded from a human readable manifest file in HDFS. Having a file specifying auxiliary service information will allow more flexibility and the ability to pass more information than reading auxiliary services from a filesystem directory or from the yarn-site.xml Configuration. We propose using a manifest file consisting of a list of YARN service specifications. This specification already exists, could easily contain all the information needed to load an auxiliary service, and would make it easier to move towards container-based auxiliary services in the future. Auxiliary services will not be launched using the YARN service framework, rather the service specification will be used instead of the yarn-site.xml Configuration to tell the NM which auxiliary services to load in the same in-process manner that it does currently.

The manifest file will be specified via a new Configuration property, *yarn.nodemanager.aux-services.manifest*. If this property is specified in the Configuration, NMs will read the manifest file on startup. When a manifest file is specified, they will not read auxiliary services from the Configuration (the old method). If no manifest file is specified, NMs will fall back on reading auxiliary services from the Configuration.

To support reloading an existing auxiliary service, we propose adding an optional version to the auxiliary service information. If the NM is already running an auxiliary service of the same name and the same version found in the manifest file, the NM will not reload the auxiliary service. If an auxiliary service found in the manifest file has the same name and a different version as an auxiliary servicre the NM is running, the NM will stop and remove its existing auxiliary service and initialize and start a new instance of the auxiliary service based on the information specified in the manifest file.

AuxiliaryService implementations may require modifications to support reloading. For example, the mapred ShuffleHandler registers a metrics source, but doesn't unregister it on service stop. This results in an error when another instance of ShuffleHandler is created. AuxiliaryService implementations should ensure that they perform any required cleanup on service stop.

Manifest file

The following information is needed for the NM to load auxiliary services:

- List of Aux Services on each NodeManager (required)
- Name of each Aux Service (required)
- Class name of each Aux Service (required)
- Jars forming the classpath of each Aux Service
- List of system classes that will not be loaded by the classloader for each Aux Service
- Aux Service specific configurations that are today put in NM configs (e.g. the Spark shuffle port)

We propose mapping auxiliary services information to the service specification as follows:

- List of Aux Services on each NodeManager (required) -> list of services
- Name of each Aux Service (required) -> service name
- Jars forming the classpath of each Aux Service -> configuration files

with the remaining information (class name, system classes, and custom configurations) to be mapped as configuration properties. Also, an optional version will be added to the auxiliary services information to allow the NM to determine if an existing Aux Service should be reloaded.

Example:

```
{
  "services": [
    {
      "name": <name>, // required
      "version": <version>,
      "configuration": {
        "properties": {
          "class.name": <className>, // required
          "system.classes": <systemClassesCsv>,
          <customAuxServiceProperty>: <customAuxServiceValue>,
          ...
        },
        "files": [ // optional files that will be localized
                   // and used to construct service classpath
        {
          "src_file": <jar file>,
          "type": "STATIC"
        },
```

```
          {
            "src_file": <tarball>,
            "type": "ARCHIVE"
          },
          ...
        ]
      }
    },
    ...
  ]
}
```

Instead of having a single classpath property as we did previously, we can use the service specification files to list multiple jars/tarballs that should be downloaded as local resources and added to the classpath for the service. Only STATIC/ARCHIVE file types may be specified for now. A dest_file property will be ignored, as the file will be downloaded to the auxiliary services directory with a special file name and will be added automatically to the classpath.

The configuration property names that were used to specify auxiliary service information in the Configuration are converted here to simpler names, so that *yarn.nodemanager.aux-services.<name>.class* becomes *class.name* and *yarn.nodemanager.aux-services.<name>.system-classes* becomes *system.classes* in the service specification. Note that in investigating this JIRA, I discovered that the aux services classloader has a bug where it is using the property *yarn.nodemanager.aux-services.<className>.system-classes* (which has the class name of the aux service instead of the short name for the aux service).

Reloading the manifest

There are multiple possible ways the NMs could receive updated manifest information. These options include:

1. NM polling the manifest file
2. RM polling the manifest file and using NM heartbeat to tell NM to reload manifest file
3. RMAdmin API call to tell RM to reload manifest file and then use NM heartbeat to tell NM to reload manifest file
    a. Asynchronous with a mechanism for retrieving auxiliary service status (NM REST API)
    b. RM would need to store manifest version or hash and send to NM and the NM would determine if it is already using that version or not. Or the RM could send a map of auxiliary service name to version and there wouldn't need to be a separate version/hash of the manifest.
4. Through a new NMAdmin API

    a. Refresh call indicating the manifest should be reloaded from HDFS vs. receiving entire manifest from admin call
    b. Synchronous with output of admin call indicating which NMs successfully applied the changes vs. asynchronous with some mechanism for retrieving auxiliary service status

For this ticket, we will implement option 1, dynamically loading auxiliary services from a manifest file through NM polling. Options 2 or 3 can be considered for future work, while option 4 does not seem worthwhile.

Future Work

Candidates for future work include:

- RM telling NMs when to reload aux services (through RM polling or RMAdmin API)
- Allowing different auxiliary services to be configured on different node partitions
- Supporting out-of-process auxiliary services (YARN-1593)