Design of pluggable YARN Auxiliary Services (YAS)

Overview

The propose is to create customizable auxiliary services infrastructure for all YARN services to provide abilities to extends or change functionality for all YARN demons.

Motivation

Currently, YARN has the ability to add auxiliary services only for NodeManager. All custom services limited by *AuxiliaryService* which include abstract methods such as *initializeApplication* or *stopApplication*. This class basically designed to implement custom shuffle service like *ShuffleHandler*. But in some cases we do not need to manage application, we only need to run some process within another daemon (e.g. node manager). For example, *PerNodeTimelineCollectorsAuxService* extends *AuxiliaryService* to be a part of NM demon but ignore methods mentioned above. So, this architecture is limited and basically intended for custom shuffle services.

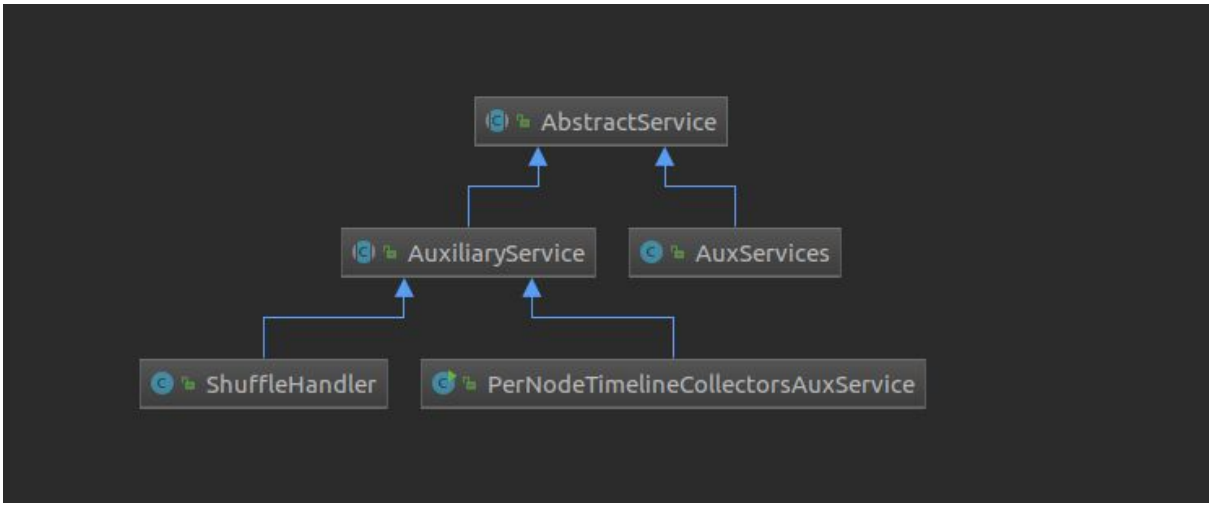
Also, there are some cases when we need to add auxiliary services for some other YARN demons. For example, in YARN-2267 proposed to add Auxiliary Service support in RM for implementation monitor plugins or some alarm services. We faced with one more case, we need to create/setup some storage for YARN demon data (for example for RM state store or AM intermediate data) before demon started. Also, will be good if we can use some auxiliary service in different YARN demons.

Usability improvements

The end users can use YAS to customize or extend the functionality of all YARN demons. We can provide an ability to create aux service both for one and for all demons (See *Architecture*). This will make easy creating of aux service for shuffle or just to up some service within another YARN daemon. We will get a unified and more extensible architecture for all aux services which may need to create in the future.

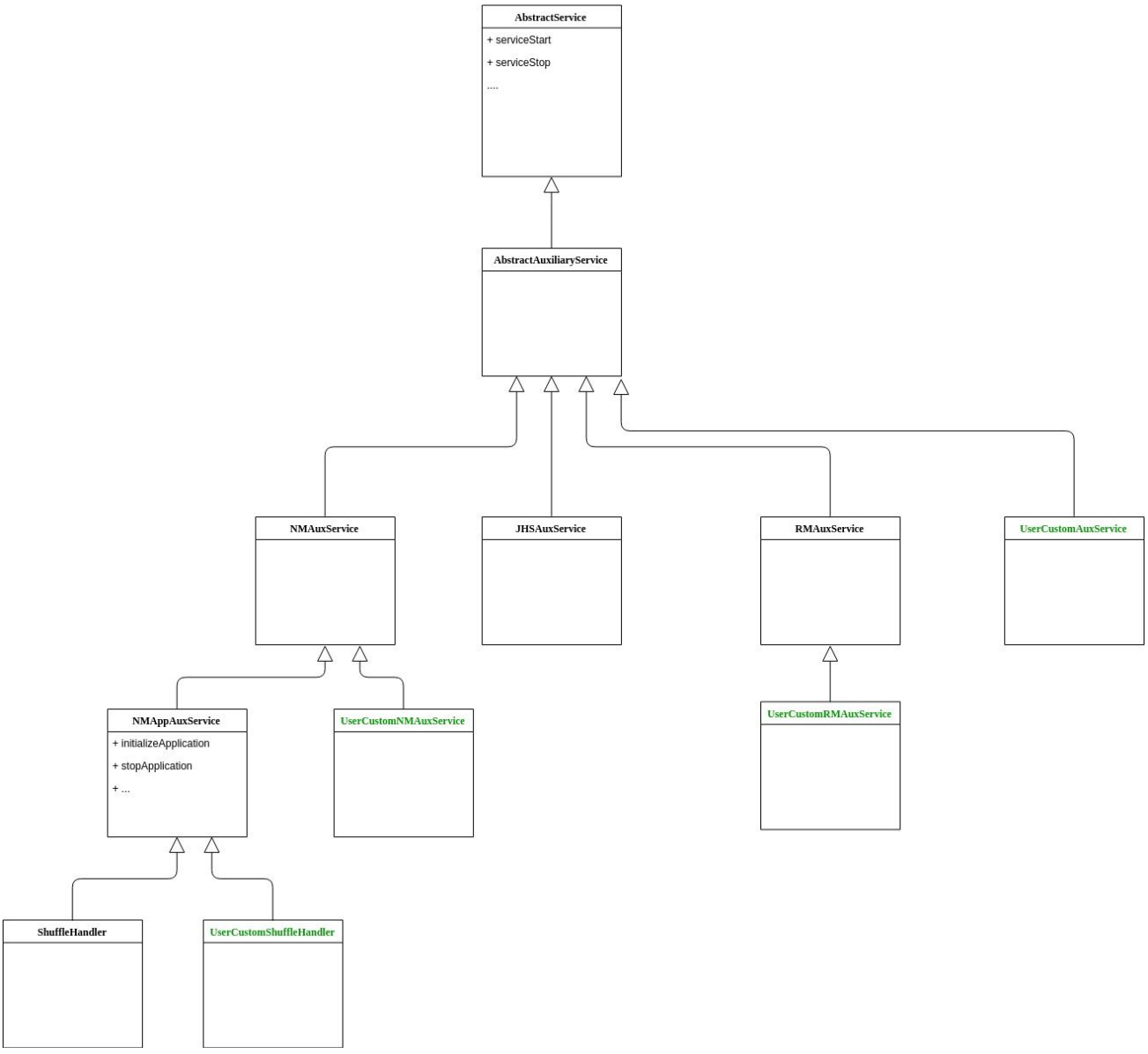
Architecture

Currently, we have the following structure:



AuxiliaryService is the base class which intended for extending by end users. This class contains methods which are used for application management. *ShuffleHandler* extends *AuxiliaryService* to provide the implementation of the shuffle phase for MapReduce application. *AuxServices* is class which is used for initializing auxiliary services which were specified in configurations. This class is created in *ContainerManagerImpl* class during *NodeManager* start and self-init and add all needed services.

In this documentation proposed to create the following structure:
Add base class *AbstractAuxiliaryService* which is intended for extends for other aux services. (**black** name of classes mark Hadoop side classes, green classes which were created by end user)
We can create per demon aux services. *NMAuxService*, *RMAuxService*, *JHSAuxService* etc. and extend these services for all our needs. For example, manage application in NM by *NMAppAuxService* which provide abstract methods which should be implemented by particular class such as *ShuffleHandler* or *UserCustomShuffleHandler*. Also, we can add in the future some other classes which extend *NMAuxService* from Hadoop side to provide some another functionality which can be changed.
The services which implement *AbstractAuxiliaryService* can be used in all YARN demons. For example, *UserCustomAuxService* will be run as part of all YARN services. *UserCustomRMAuxService* will be a part only of RM demon, *UserCustomNMAuxService* only of NM demons etc.

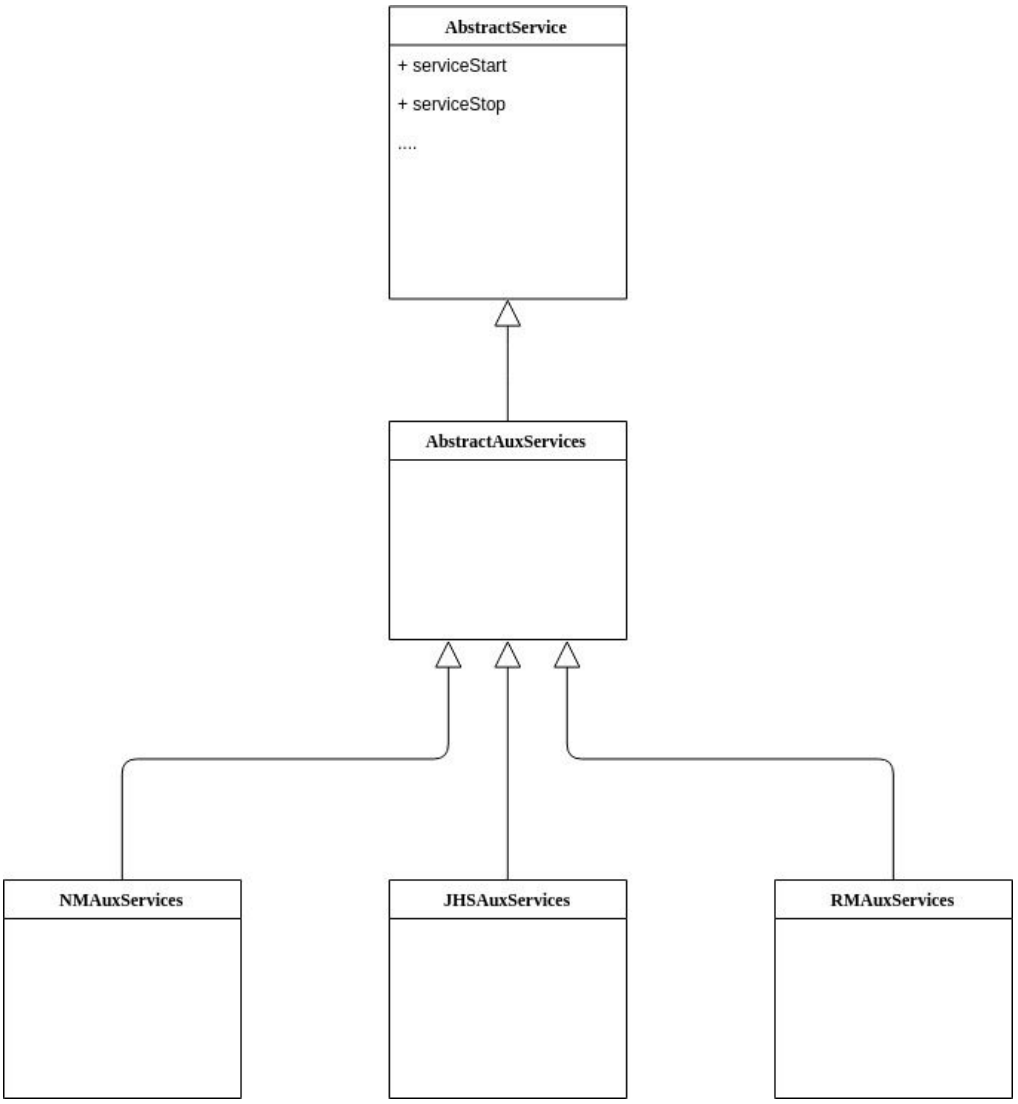


YARN Auxiliary Services

All these services will be initialized by particular *AuxServices* class. *AbstractAuxServices* class will initialize all common services which extend *AbstractAuxiliaryService*. Common service is the service which can be run an instance in all YARN demons and represent logic which can be used by all demons, this is **NOT** a single service/process which will be used by all YARN demons. This service needed when we have logic which acceptable for all YARN demons and we want to reuse it without creating a separated class for a different demon.

All particular classes will initialize services for the needed demon. For example, *NMAuxServices* class will use the method of *AbstractAuxServices* superclass to initialize common services and then initialize services which intended only for NM.

All these classes will be created in YARN demon classes such as *ResourceManager* and *NodeManager* in *serviceInit* method.



YARN Aux Services

Backward compatibility

We can change *AuxiliaryService* to extend *NMAppAuxService* to fit new architecture and save backward compatibility. Mark *AuxiliaryService* as Deprecated and add a hint to use *NMAppAuxService* instead of *AuxiliaryService* for new services. Therefore, all existing client code can run the same as before. In the future major version, we can remove *AuxiliaryService* at all. Existing *AuxServices* class will be renamed to *NMAuxServices* and we will change it to extend *AbstractAuxServices*.

Deployment

Configurations

To configure needed aux services we can just extend current configuration by the following properties:

Configuration Name
YARN Common
yarn.aux-services
yarn.aux-services.%s.classpath
yarn.aux-services.%s.remote-classpath
yarn.aux-services.%s.system-classes
NM
yarn.nodemanager.aux-services
yarn.nodemanager.aux-services.%s.classpath
yarn.nodemanager.aux-services.%s.remote-classpath
yarn.nodemanager.aux-services.%s.system-classes
RM
yarn.resourcemanager.aux-services
yarn.resourcemanager.aux-services.%s.classpath
yarn.resourcemanager.aux-services.%s.remote-classpath
yarn.resourcemanager.aux-services.%s.system-classes
...

Example:

```
<property>
   <name>yarn.resourcemanager.aux-services</name>
   <value>customRMAuxService</value>
</property>

<property>
   <name>yarn.resourcemanager.aux-services.customRMAuxService.classpath</name>
   <value>${local_dir_to_jar}/CustomRMAuxService.jar</value>
</property>

<property>
   <name>yarn.resourcemanager.aux-services.customRMAuxService.class</name>
```

<value>org.aux.CustomRMAuxService</value>
</property>

As the result, *AbstractAuxServices* class will initialize all services specified by *"yarn.aux-services"* property. These services can be run as part of all YARN demons.
NMAuxServices class will invoke initialization from parent *AbstractAuxServices* class and also initialize all services specified by *"yarn.nodemanager.aux-services"* property. RM's aux services which specified by *"yarn.resourcemanager.aux-services"* property will initialize by *RMAuxServices* class, the common services will be initialized the same as for NM.