

In-Place Upgrade of Long-Running Applications in YARN

Background

Classic Slider supports upgrade/downgrade of application binaries and configurations as described in:

http://slider.incubator.apache.org/docs/slider_specs/application_pkg_upgrade.html

Yarn Services, which is the result of merging Slider capabilities in Yarn, needs to provide a way to upgrade/downgrade applications. This document talks about processing of upgrade action and Yarn Service API changes for supporting it.

Yarn Service Upgrade

Requirements

- Users can save a new version of the service.
- Users can upgrade all the components or selective components of the service to the new version.
- Users can upgrade a single component-instance or a range of component-instances of a component.
- Users can get a list of components running a specific version.
- User can get a list of component-instances running a specific version.

We will provide a run-book style list of steps to perform in-place upgrade of service. This will be similar to Slider upgrade support.

Prerequisites

- <https://issues.apache.org/jira/browse/YARN-7523>
Add a version field to Yarn Service.

Rest API

Changes from version 1 of Rest API are highlighted.

Metadata

```
public enum ServiceState {  
    ACCEPTED, STARTED, STABLE, STOPPED, FAILED, FLEX, UPGRADING, UPGRADING_AUTO_FINALIZE*  
}
```

*UPGRADING_AUTO_FINALIZE is explained [here](#).

```
public class Service {  
    ...  
    String version;  
    ...}
```

```
public enum ComponentState {  
    FLEXING, STABLE, UPGRADING, NEEDS_UPGRADE  
}
```

```
public enum ContainerState {  
    RUNNING_BUT_UNREADY, READY, STOPPED, UPGRADING  
}
```

Example Service

We will use the example of service1 to understand upgrade actions at each level.

```
service1 (v:1) STABLE  
  |__ compX (artifactId:'x1') STABLE  
    |_____complnsX1 (artifactId:'x1') READY  
    |_____complnsX2 (artifactId:'x1') READY  
  |__ compY (artifactId:'y1') STABLE  
    |_____complnsY1 (artifactId:'y1') READY  
    |_____complnsY2 (artifactId:'y1') READY
```

When does a Component need upgrade

A component will need an upgrade when either of the following fields change (first iteration):

1. Artifact id of the component changes
2. Configuration changes which includes:
 - a. Properties
 - b. Environment properties
 - c. Config files
3. Launch Command
4. Run privileged containers

For a subsequent iteration:

1. Dependencies, readiness check, number of containers, quicklinks: when these change, then the containers will not require a restart.
2. Resource and placement policy

3. Addition of a component: When new components are added then the current code in Service AM should automatically pick the difference and immediately start the containers for those components. This may not require the app-owner has to perform any additional steps as part of upgrade. Will have to test this.
4. Explicit removal of a component

In the following examples we will just use changes in the artifact id in order to keep the explanation concise.

Initiate Upgrade

Upgrade is initiated by the service level update API. This action saves the newer version of service definition and changes the state of service and components:

- The new service definition is saved as {service}_{version_no}. We will overwrite the older service definition with the newer one, once the upgrade is finalized. At any given point of time, there will be only 2 versions of service definitions.
- The ServiceState becomes UPGRADING.
- The ComponentState of all the components that have a different artifact id changes to NEEDS_UPGRADE.

The existing update endpoint for service can be used.

```
- PUT /v2/services/{service_name}
public Response updateService(@Context HttpServletRequest request,
    @PathParam(SERVICE_NAME) String appName,
    Service updateServiceData) {
    //...
    if (updateServiceData.getState() == ServiceState.UPGRADING) {
        //perform service upgrade
    }
    //...
}
```

Service upgrade is triggered when `updateServiceData.getState() == ServiceState.UPGRADING`.

Let's say service1 is upgraded to version 2:

CURRENT	DESIRED
service1 (v:1)	service1 (v:2)
<div> <div>compX (artifactId:'x1')</div> <div> <div>complsX1 (artifactId:'x1')</div> <div>complsX2 (artifactId:'x1')</div> </div> </div>	<div> <div>compX (artifactId:'x2')</div> <div> <div>complsX1 (artifactId:'x2')</div> <div>complsX2 (artifactId:'x2')</div> </div> </div>
<div> <div>compY (artifactId:'y1')</div> <div> <div>complsY1 (artifactId:'y1')</div> <div>complsY2 (artifactId:'y1')</div> </div> </div>	<div> <div>compY (artifactId:'y1')</div> <div> <div>complsY1 (artifactId:'y1')</div> <div>complsY2 (artifactId:'y1')</div> </div> </div>

1. service1_v2 is persisted.
2. State of service1 is changed to UPGRADING.

3. Only compX is affected since the desired artifact is 'x2' and its current artifact is 'x1'.
The state of compX is changed to NEEDS_UPGRADE.

At the end of service upgrade call, the service state will be:

CURRENT

service1 (v:1) UPGRADING

```
└─ compX (artifactId:'x1') NEEDS_UPGRADE
    └─ _____complnsX1 (artifactId:'x1') READY
    └─ _____complnsX2 (artifactId:'x1') READY
└─ compY (artifactId:'y1') STABLE
    └─ _____complnsY1 (artifactId:'y1') READY
    └─ _____complnsY2 (artifactId:'y1') READY
```

No upgrades are performed by this call. It is just an indication that the user is going to perform component/component instances upgrade now.

Upgrade Component

Single Component

Calling upgrade on a component will upgrade all running component instances (or containers) of that component. The app owner should not assume they will be upgraded in any particular order. In fact all containers could be upgrading at the exact same time, thereby reaching a temporary intermediate state where none of them are available. If this is not desired, then app owners should consider using "Upgrade Component Instance" API to get fine grained control of upgrading containers in batches, maybe even with sufficient delay between every 2 batches to assure component and hence service availability.

The existing update endpoint for component can be used.

```
- PUT /v2/services/{service_name}/components/{component_name}
public Response updateComponent(@Context HttpServletRequest request,
    @PathParam(SERVICE_NAME) String appName,
    @PathParam(COMPONENT_NAME) String componentName, Component component) {
    //...
    if (component.getState() == ComponentState.UPGRADING) {
        //perform component upgrade
    }
    //...
}
```

Component upgrade is triggered when `component.getState() == ComponentState.UPGRADING`.

When triggering a component upgrade, the artifact that a component is upgraded to is the one present in the newer version of service definition which is persisted. The artifact id in the request body is completely ignored.

Component upgrade is only triggered when the current state of Component is “NEEDS_UPGRADE” otherwise it is a no-op.

Let’s say compX is upgraded

CURRENT	DESIRED
service1 (v:1) UPGRADING	
___ compX (artifactId:'x1') NEEDS_UPGRADE	compX (artifactId:'x2')
_____complnsX1 (artifactId:'x1')	_____complnsX1 (artifactId:'x2')
_____complnsX2 (artifactId:'x1')	_____complnsX2 (artifactId:'x2')
___ compY (artifactId:'y1') STABLE	
_____complnsY1 (artifactId:'y1') READY	
_____complnsY2 (artifactId:'y1') READY	

1. The state of compX is changed to UPGRADING.
2. The instances of compX are asynchronously upgraded.
3. Once all the instances are upgraded, the state of compX is changed to STABLE and its artifact id will be ‘x2’.

The service state right after the API call will look like:

CURRENT
service1 (v:1) UPGRADING
___ compX (artifactId:'x1') UPGRADING
_____complnsX1 (artifactId:'x1') READY
_____complnsX2 (artifactId:'x1') READY
___ compY (artifactId:'y1') STABLE
_____complnsY1 (artifactId:'y1') READY
_____complnsY2 (artifactId:'y1') READY

The service state once the upgrade of component instances has initiated but not all instances have completely upgraded:

CURRENT
service1 (v:1) UPGRADING
___ compX (artifactId:'x1') UPGRADING
_____complnsX1 (artifactId:'x1') UPGRADING
_____complnsX2 (artifactId:'x2') READY
___ compY (artifactId:'y1') STABLE
_____complnsY1 (artifactId:'y1') READY
_____complnsY2 (artifactId:'y1') READY

The service state once all the instances have completely upgraded:

CURRENT
service1 (v:1) UPGRADING
___ compX (artifactId:'x2') STABLE
_____complnsX1 (artifactId:'x2') READY
_____complnsX2 (artifactId:'x2') READY
___ compY (artifactId:'y1') STABLE
_____complnsY1 (artifactId:'y1') READY
_____complnsY2 (artifactId:'y1') READY

~~We can add a field “target_service_version” to Component metadata.~~

~~If there is a “target_service_vesion” field in the component, then that will indicate creating a newer version of service definition instead of updating the current version.~~

Multiple Components

This will upgrade all the components that are marked as upgrading. The order of upgrades across component can be decided by the AM based on component dependencies. The steps in upgrading an individual component will be same as described in the upgrade of single component [section](#).

```
- PUT /v2/services/{service_name}/components
public Response updateComponents(@Context HttpServletRequest request,
    @PathParam(SERVICE_NAME) String appName, List<Component> components) {
    //...
    components.forEach(component -> {
        if (component.getState() == ComponentState.UPGRADING) {
            //add to components that need upgrade
        }
    });
    // perform upgrade of the required components based on components dependency.
    //...
}
```

Upgrade Component Instance

Single Component Instance

```
PUT
/v2/services/{service_name}/components/{component_name}/component-instances/{component_instance_name}
public Response updateComponentInstance(@Context HttpServletRequest request,
    @PathParam(SERVICE_NAME) String appName,
    @PathParam(COMPONENT_NAME) String componentName,
    @PathParam(COMPONENT_INSTANCE_NAME) String componentInstanceName,
    Container container) {

    //...
    if (container.getState() == ContainerState.UPGRADING) {
        //perform container upgrade
    }
    //...
}
```

Since, component instance names are unique, we will support shorter version for upgrading component instance as well:

```
PUT /v2/services/{service_name}/component-instances/{component_instance_name}
public Response updateComponentInstance(@Context HttpServletRequest request,
    @PathParam(SERVICE_NAME) String appName,
    @PathParam(COMPONENT_INSTANCE_NAME) String componentInstanceName,
    Container container) {
```

```

//...
if (container.getState() == ContainerState.UPGRADING) {
    //perform container upgrade
}
//...
}

```

Component instance is triggered when `container.getState() == ContainerState.UPGRADING`. The artifact that a component instance is upgraded to is the artifact id of the component present in the newer version of the service definition which is persisted. Any artifact id in the request body is completely ignored.

Component instance upgrade is only triggered when the current state of the Component is "NEEDS_UPGRADE".

Let's say `complsX1` is upgraded.

CURRENT	DESIRED
service1 (v:1) UPGRADING	
__ compX (artifactId:'x1') NEEDS_UPGRADE	
_____complsX1 (artifactId:'x1')	complsX1 (artifactId:'x2')
_____complsX2 (artifactId:'x1')	
__ compY (artifactId:'y1') STABLE	
_____complsY1 (artifactId:'y1') READY	
_____complsY2 (artifactId:'y1') READY	

1. The state of `complsX1` is changed to UPGRADING.
2. `complsX1` is upgraded asynchronously.
3. Once the `complsX1` is upgraded, its state will change to READY and its artifact will be 'x2'.

The service state right after the API call will look like:

```

CURRENT
service1 (v:1) UPGRADING
    |__ compX (artifactId:'x1') NEEDS_UPGRADE
        |_____complsX1 (artifactId:'x1') UPGRADING
        |_____complsX2 (artifactId:'x1') READY
    |__ compY (artifactId:'y1') STABLE
        |_____complsY1 (artifactId:'y1') READY
        |_____complsY2 (artifactId:'y1') READY

```

The service state once the container has finished upgrading:

```

CURRENT
service1 (v:1) UPGRADING
    |__ compX (artifactId:'x1') NEEDS_UPGRADE
        |_____complsX1 (artifactId:'x2') READY
        |_____complsX2 (artifactId:'x1') READY
    |__ compY (artifactId:'y1') STABLE
        |_____complsY1 (artifactId:'y1') READY
        |_____complsY2 (artifactId:'y1') READY

```

When each of the component instances are in READY state with the newer version, then the state of the state of the component will be changed to STABLE.

Let's say complnsX2 is also upgraded, then the service state right after the upgrade call will look like:

CURRENT

service1 (v:1) UPGRADING

```
|__ compX (artifactId:'x1') NEEDS_UPGRADE
    |_____complnsX1 (artifactId:'x2') READY
    |_____complnsX2 (artifactId:'x1') UPGRADING
|__ compY (artifactId:'y1') STABLE
    |_____complnsY1 (artifactId:'y1') READY
    |_____complnsY2 (artifactId:'y1') READY
```

The service state once the complnsX2 has finished upgrading:

CURRENT

service1 (v:1) UPGRADING

```
|__ compX (artifactId:'x1') STABLE
    |_____complnsX1 (artifactId:'x2') READY
    |_____complnsX2 (artifactId:'x2') READY
|__ compY (artifactId:'y1') STABLE
    |_____complnsY1 (artifactId:'y1') READY
    |_____complnsY2 (artifactId:'y1') READY
```

Multiple Component Instances

```
PUT /v2/services/{service_name}/component-instances
```

```
public Response updateComponentInstances(@Context HttpServletRequest request,
    @PathParam(SERVICE_NAME) String appName,
    List<Container> componentInstances) {
    //...
    componentInstances.forEach(container -> {
        if (container.getState() == ContainerState.UPGRADING) {
            //perform container upgrade
        }
    });
    //...
}
```

The steps in upgrading an individual component instance will be same as described in the upgrade of single component instance [section](#).

Multiple Component Instances of a Component

```
PUT /v2/services/{service_name}/components/{component_name}/component-instances
```

```
public Response updateComponentInstances(@Context HttpServletRequest request,
    @PathParam(SERVICE_NAME) String appName,
    @PathParam(COMPONENT_NAME) String componentName,
    List<Container> componentInstances) {
    //...
    componentInstances.forEach(container -> {
        if (container.getState() == ContainerState.UPGRADING) {
            //perform container upgrade
        }
    })
}
```



```
});
//...
}
```

The steps in upgrading an individual component will be same as described in the upgrade of single component [section](#).

Finalize Upgrade

Finalize upgrade consists of 2 actions:

1. Overwriting the older version of service spec with the newer version.
2. Changing the service state to STABLE.

We provide users with 2 finalize options-

1. They perform the upgrade manually by invoking the service level update API.
This is helpful if the user needs to perform validations after the upgrade and if these validations fail, the user may want to abort the upgrade. The older service spec will not be lost.
2. They want the service to auto finalize.

Manual Finalize

When the upgrade was initiated with ServiceState=UPGRADING, then the user needs to manually finalize the upgrade.

To manually finalize the upgrade, the user needs to invoke the service level update API with **ServiceState = STARTED**.

This call will overwrite the older service spec with the newer version and the service state will be changed to STABLE.

The service state after the API call will look like:

```
CURRENT
service1 (v:1) STABLE
  |__ compX (artifactId:'x1') STABLE
      |_____complnsX1 (artifactId:'x2') READY
      |_____complnsX2 (artifactId:'x2') READY
  |__ compY (artifactId:'y1') STABLE
      |_____complnsY1 (artifactId:'y1') READY
      |_____complnsY2 (artifactId:'y1') READY
```

Auto Finalize

When the upgrade was initiated with **ServiceState = UPGRADING_AUTO_FINALIZE**, then the service will finalize the upgrade automatically. Once all the components are in STABLE state with the newer version, then

- The older service definition will be overwritten with the newer version.
- The service state will change to STABLE.

Abort/Upgrade during an upgrade

We let the user upgrade to a previous/newer version when the service is already upgrading.

NOTE: This will not require any special handling. It will fall in place with the regular processing of upgrades where the state of service and components are changed by comparing with the latest desired spec.

Abort

Let's say the current state of the service is:

CURRENT

service1 (v:1) UPGRADING

```
└─ compX (artifactId:'x1') NEEDS UPGRADE
    └─ complnsX1 (artifactId:'x2') READY
    └─ complnsX2 (artifactId:'x1') READY
└─ compY (artifactId:'y1') STABLE
    └─ complnsY1 (artifactId:'y1') READY
    └─ complnsY2 (artifactId:'y1') READY
```

The user can abort the upgrade by issuing another service level upgrading request with ServiceState=UPGRADING and the desired spec as:

DESIRED (version v1 of the service)

service1 (v:1)

```
└─ compX (artifactId:'x1')
    └─ complnsX1 (artifactId:'x1')
    └─ complnsX2 (artifactId:'x1')
└─ compY (artifactId:'y1')
    └─ complnsY1 (artifactId:'y1')
    └─ complnsY2 (artifactId:'y1')
```

Upgrade to a different version during an upgrade

Similarly to abort, users can trigger upgrade to a different version in middle of an existing upgrade.

CURRENT

service1 (v:1) UPGRADING

```
└─ compX (artifactId:'x1') NEEDS UPGRADE
    └─ complnsX1 (artifactId:'x2') READY
    └─ complnsX2 (artifactId:'x1') READY
└─ compY (artifactId:'y1') STABLE
    └─ complnsY1 (artifactId:'y1') READY
    └─ complnsY2 (artifactId:'y1') READY
```

DESIRED

service1 (v:3)

```
└─ compX (artifactId:'x2')
    └─ complnsX1 (artifactId:'x2')
    └─ complnsX2 (artifactId:'x2')
└─ compY (artifactId:'y2')
    └─ complnsY1 (artifactId:'y2')
    └─ complnsY2 (artifactId:'y2')
```

Get Endpoints

All the below endpoints are new.

Get components

List components with query params as:

- Artifact Id
- List of component states

```
GET /v2/services/{service_name}/components
public List<Component> getComponents(@PathParam(SERVICE_NAME) String appName,
    @QueryParam("artifactId") String artifactId,
    @QueryParam("componentStates") List<Enum<ComponentState>> componentStates) {
}
```

Get Component Instances

List component-instances with query params as:

- List of component names
- Artifact Id
- List of container states

```
GET /v2/services/{service_name}/component-instances
public List<ComponentInstance> getComponentInstances(@PathParam(SERVICE_NAME) String appName,
    @QueryParam("componentNames") List<String> componentNames,
    @QueryParam("artifactId") String artifactId,
    @QueryParam("containerStates") List<Enum<ContainerState>> containerStates) {
}
```

ServiceClient API

API for upgrades

```
public int actionUpgrade(String appName, String version)
```

Upgrade the service to requested version.

```
public int actionUpgrade(String appName, List<Component> components)
```

Upgrade the selected components.

```
public int actionUpgrade(String appName, List<Container> componentInstances)
```

This will allow upgrade of a selected component instances.

API for listing components and component instances

```
public List<Component> getComponents(String appName, EnumSet<ComponentState> states, String artifactId)
```

This will list all the components of the service which match the requested states and artifactId.

```
public List<Container> getComponentInstances(String appName, Set<String> componentNames,
    EnumSet<ContainerState> states, String artifactId)
```

This will list all the containers of the service which match the requested components/states/artifactId.

Open Items

1. Gour's comments:

For components which do not have artifacts, but `launch_command` only, we have to understand that they rely on some system level binaries available on all hosts. The host binaries may have been upgraded (outside the scope of YS) or maybe the service is being upgraded with a newer `launch_command`. For such artifact-less components, we have to think of a strategy to decipher in which of the 3 states they are in -> pre-upgrade, post-upgrade or no-upgrade.

Yarn Service Downgrade

No additional API for downgrade. It can be achieved by calling the upgrade with the older version.

Follow-up Items

1. Jian's Suggestion: Roll back by version id. We can maintain a history of versions - a concept similar to git.

Appendix

Upgrade Support in Yarn

<https://issues.apache.org/jira/browse/YARN-4726>

Upgrade and Restart container API exposed in `ContainerManagementProtocol` via <https://issues.apache.org/jira/browse/YARN-5609>

Related Jiras

<https://issues.apache.org/jira/browse/YARN-1040>

<https://issues.apache.org/jira/browse/YARN-4726>

<https://issues.apache.org/jira/browse/YARN-7512>