Slider Rolling Upgrade/Downgrade

Phases

- YARN core (libraries and configurations) upgrade/downgrade
- Slider as a YARN application (libraries and configurations) upgrade/downgrade
- Applications deployed by Slider (libraries and configurations) upgrade/downgrade

Prerequisites

- YARN rolling upgrade/downgrade support with no need to restart Slider App Master(s)
- YARN to provide complete backward compatibility to running Slider App Master(s) which includes Application Submission Context, Container Launch Context, AM-RM protocol, etc.

Upgrade Applications Deployed by Slider

A run book style set of atomic steps exposed by Slider (orchestrated by Ambari or a shell script or executed manually in the right order)

First time an app owner bring a new package and creates a new instance

`slider package --install --name MyHBase_Facebook --version 1.0 --package ~/slider-hbase-app-package_v1.0.zip` **`(uploads new application package slider-hbase-app-package_v1.0.zip)`**

`slider create MyHBase_Facebook_Finance --template ~/myHBase_template_v1.0.json --resources ~/myHBase_resources_v1.0.json` **`(creates application MyHBase_Facebook_Finance)`**

`slider flex MyHBase_Facebook_Finance --component HBASE_MASTER 5` **`(application MyHBase_Facebook_Finance goes through its usual lifecycle)`**

The app owner can list all containers of an app instance with filtering options

```
slider list MyHBase_Facebook_Finance --containers

slider list MyHBase_Facebook_Finance --containers --components HBASE_MASTER
slider list MyHBase_Facebook_Finance --containers --components HBASE_MASTER
HBASE_REGIONSERVER --version 1.0

slider list MyHBase_Facebook_Finance --containers --version 1.0
```

At some point, the app owner has a newer version of the package and needs to upgrade

```
slider package --install --name MyHBase_Facebook --version 2.0 --package
~/slider-hbase-app-package_v2.0.zip
```
(uploads the newer version 2.0 of the app package)

```
slider upgrade MyHBase_Facebook_Finance --template ~/myHBase_template_v2.0.json
--resources ~/myHBase_resources_v2.0.json
```
(This command upgrades the internal state of Slider App Master with the new config and resource specifications. After this, upgrade command needs to be issued against all containers to upgrade them to the new app version e.g. v2.0 in this case. Note, it is recommended to not issue the flex command although Slider does not explicitly block it.)

At this point the upgrade orchestrator (Ambari or a script) will rely on the slider `list` command with `--containers` option (with additional filter options `--components` and/or `--version`) to find all the containers that are running for the application. The orchestrator needs to decide how it wants to span/schedule the upgrade commands of every single container. This will primarily depend on the type of the application. For e.g. say HBase might want to call upgrade on all HBase Master containers first before calling upgrade on any of the RegionServer containers. Storm might want to provide a 5 min delay between the upgrade of the Nimbus and Supervisor container(s).

```
slider upgrade MyHBase_Facebook_Finance --containers id1 id2 .. idn
```
(**upgrade all containers with ids id1, id2, .. idn. It is possible that all these n containers id1, id2, ... idn can actually be down at the same time. If this is not desired then issue n upgrade command with a single container id at a time.**)

```
e.g.
slider upgrade MyHBase_Facebook_Finance --containers
container_e03_1427412101162_0001_01_000002
container_e03_1427412101162_0001_01_000003

slider upgrade MyHBase_Facebook_Finance --components role1 role2 .. rolen
```
(**upgrade all containers of all roles role1, role2, .. rolen. Again all containers of all the n roles can be down at the same time. If this is not desired use the --containers option with one container at a time.**)

```
e.g.
```

```
slider upgrade MyHBase_Facebook_Finance --components HBASE_MASTER
HBASE_REGIONSERVER
```

When do we know that upgrade is complete?

The upgrade command need to be executed against all containers and upgrade can be considered complete only if the command `slider list MyHBase_Facebook_Finance --containers --version 1.0` returns no containers.

The app owner can also call `slider list MyHBase_Facebook_Finance --containers` to explicitly check that all expected containers are running and with the desired new version (e.g. 2.0 in this case).

If any issues are encountered in Slider and/or the application during upgrade, the application can be downgraded to its original version.

Downgrade Applications Deployed and Upgraded by Slider

A Slider deployed application might be required to be downgraded to its original version for any reason - whether the upgrade was completed in entirety or was in-flight. There are no new commands exposed to perform downgrade. It basically involves calling the `upgrade` command again with the older version (e.g. version 1.0 in this case) of the application config and resource specifications. It needs to be followed by upgrade commands issued against all containers which were upgraded to the newer version.

```
slider upgrade MyHBase_Facebook_Finance --template ~/myHBase_template_v1.0.json
--resources ~/myHBase_resources_v1.0.json (This command although called
upgrade, actually downgrades the internal state of Slider App Master with the
old v1.0 config and resource specifications)
```

```
slider list MyHBase_Facebook_Finance --containers --version 2.0
```

Take the list of all containers returned by the above command and issue upgrade command on all of them (in desired orchestrated fashion) -

```
slider upgrade MyHBase_Facebook_Finance --containers id1 id2 .. idn
```

Pre and post upgrade hooks:

- **Pre-upgrade hook (optional):** The pre-upgrade steps, if provided, will allow applications to execute simple housekeeping tasks before Slider actually calls stop operation in an upgrade scenario (specifically if they need to be performed in every single container and 1000s of them are running). An example could be to send a message to a queue that the current instance of memcached is going down so that the load balancer rules can be dynamically updated. Performing long-running tasks or tasks that needs to be executed by only 1 instance of a component (n instances of which are running) should be performed manually before calling `upgrade` (or maybe before calling the `upgrade` command). The pre-upgrade hook is not a good candidate for such operations. Additional parameters might have to be exposed, like timeout, which can be used to wait at most n seconds (say) after which Slider will call the application stop hook even if the pre-upgrade operation is not completed.

- **Post-upgrade hook (optional):** This allows applications to perform simple housekeeping tasks prior to calling start on the new version of the application component. This is helpful only if such tasks are required to be performed in every single container and 1000s of them are running. This is required only in the upgrade scenario and should not be called on new containers created using flex up in non-upgrade scenarios. This is tricky, how will Slider determine this? I don't think we need a post-upgrade hook? Clarify with Rakesh and remove?

Upgrade support of simple apps with no packages:

- Upgrade of such simple apps does not involve binaries
- Upgrade of such simple apps basically means re-configuration only
- First class support is out of scope in Dal as the following can be done to perform the reconfiguration (or upgrade if you prefer to call so)
    - Download the app package that was internally created by Slider from HDFS (First class Slider command-line support will not be provided for download. So refer to documentation or get Slider/Yarn admin support to find the app package location in HDFS.)
    - Make necessary changes to metainfo.json and repackage the app
    - Make necessary updates to appConfig.json and resources.json
    - Follow the upgrade steps provided in this document to upgrade the simple app

Assumptions/Recommendations:

- Slider App Master is restarted during upgrade, which as per YARN is a new app attempt. Is this acceptable? Is there a way to tell YARN that it is an app master initiated restart and not to count it as a failure? If YARN provides this support even if it generates a new app attempt id, it is probably fine with Slider. TODO: Talk to Vinod.

- Let's take the scenario when the upgrade command has been called (with new app config and resources) and the orchestrator is in the middle of calling upgrade command on all containers in a certain order/schedule. At this point, it is important to understand that if a container (on which the upgrade command was not called yet) fails, then its replacement container will come up with the newer version of the application.
- It is recommended to not call `flex` (to increase or decrease the no of containers assigned to a specific role) after the upgrade command has been issued and until upgrade command is issued on all currently running containers (as per current resource specification). Basically avoid calling flex until "`slider list --containers --version <old_ver>`" returns no more containers. Note, flex command is not explicitly blocked by Slider, hence unexpected behavior of the application might occur if called.

Slider Internals:

- Slider will maintain at most 1 older version of app config and resource specification files in its internal HDFS folders. If v1.0 of an application is currently running and an app owner calls upgrade with v2.0 specs, Slider will retain v1.0 files. When upgrade is called again with v3.0 specs, all v1.0 files will be deleted. Only v2.0 files will be retained at that point. Note, Slider will not perform any checks to verify the current version of the containers running for a given application. Let's say if v1.0 of an application is running. The app owner calls upgrade command with v2.0 specs and then immediately follows it with upgrade command with v3.0 specs (without actually upgrading the v1.0 containers to v2.0). In this case the desire of the app owner is to upgrade v1.0 directly to v3.0. However note, that all the v1.0 specs have been deleted by Slider since the upgrade command was called twice. To be on the safe side, it is recommended to download the v1.0 app config and resource files prior to calling upgrade command. Currently there is no plan to expose a first class command line options to download these files, hence documentation/support might be required here.

Questions/TODOs:

- What if a new component (say THRIFT3 for HBase) gets introduced in v2.0 of a package which was non-existent in v1.0? Does it need to be flexed up to its desired no of container instances after the upgrade of all currently running containers are completed?

- What if an existing component gets deleted in v2.0 which existed in v1.0? Should the component be flexed down to zero before performing the upgrade? Or should the upgrade be performed as usual and then flexed down to zero post successful upgrade?
- Is it possible to embed an application version within the package of an application (say in metainfo.xml or similar) such that there is no need to specify the option `--version` during install of an app package?
- TODO: When `slider list <app> --containers` is invoked on a FINISHED/STOPPED application, Slider will dump the role/placement history of all roles of the application. Can this be done for DaI?
- What happens to role history when components are added/removed between upgrades? SLIDER-600 (RoleHistory to support reloads with different # of roles from current set) covers this topic and will be used to track from upgrade perspective as well.
- If pre-upgrade fails, today Slider will continue with the stop cmd of the app and termination of the current container. What other options can be exposed to the app owners here?

Testcases (Must have test scenarios)

- Tests on different states of the application (now that we have a fairly complex state machine when the app enters into upgrade mode)
- Tests around roles added/deleted between v1 and v2 are complex and needs to be covered. Role-history and priority makes things even more interesting and complex to write tests.
-