Armada Documentation

Release 0.2.0

Armada Team

Sep 18, 2019

Contents:

CHAPTER 1

Armada

Armada is a tool for managing multiple Helm charts with dependencies by centralizing all configurations in a single Armada YAML and providing life-cycle hooks for all Helm releases.

Find more documentation for Armada on Read The Docs.

1.1 Overview

The Armada Python library and command line tool provide a way to synchronize a Helm (Tiller) target with an operator's intended state, consisting of several charts, dependencies, and overrides using a single file or directory with a collection of files. This allows operators to define many charts, potentially with different namespaces for those releases, and their overrides in a central place. With a single command, deploy and/or upgrade them where applicable.

Armada also supports fetching Helm chart source and then building charts from source from various local and remote locations, such as Git endpoints, tarballs or local directories.

It will also give the operator some indication of what is about to change by assisting with diffs for both values, values overrides, and actual template changes.

Its functionality extends beyond Helm, assisting in interacting with Kubernetes directly to perform basic pre- and post-steps, such as removing completed or failed jobs, running backup jobs, blocking on chart readiness, or deleting resources that do not support upgrades. However, primarily, it is an interface to support orchestrating Helm.

1.2 Components

Armada consists of two separate but complementary components:

1. CLI component (**mandatory**) which interfaces directly with Tiller.
2. API component (**optional**) which services user requests through a wsgi server (which in turn communicates with the Tiller server) and provides the following additional functionality:

- Role-Based Access Control.
- Limiting projects to specific Tiller functionality by leveraging project-scoping provided by Keystone.

1.3 Installation

1.3.1 Quick Start (via Container)

Armada can be most easily installed as a container, which requires Docker to be executed. To install Docker, please reference the following install guide.

Afterward, you can launch the Armada container by executing:

```
$ sudo docker run -d --net host -p 8000:8000 --name armada \
    -v ~/.kube/config:/armada/.kube/config \
    -v $(pwd)/examples/:/examples quay.io/airshipit/armada:latest-ubuntu_bionic
```

1.3.2 Manual Installation

For a comprehensive manual installation guide, please see Manual Install Guide.

1.3.3 Usage

To run Armada, simply supply it with your YAML-based intention for any number of charts:

```
$ armada apply examples/openstack-helm.yaml [ --debug ]
```

Which should output something like this:

```
$ armada apply examples/openstack-helm.yaml 2017-02-10 09:42:36,753

  armada INFO Cloning git:
  ...
```

For more information on how to install and use Armada, please reference: Armada Quickstart.

1.4 Integration Points

Armada CLI component has the following integration points:

- Tiller manages Armada chart installations.
- Deckhand is one of the supported control document sources for Armada.
- Prometheus exporter is provided for metric data related to application of charts and collections of charts. See metrics.

In addition, Armada's API component has the following integration points:

- Keystone (OpenStack's identity service) provides authentication and support for role-based authorization.

1.5 Further Reading

Airship.

CHAPTER 2

Developers Guide

2.1 Developer Install Guide

2.1.1 Quick Start (via Container)

Note: If actively developing new Armada functionality, it is recommended to proceed with *Manual Installation* instead.

To use the docker container to develop:

1. Clone the Armada repository.
2. `cd` into the cloned directory.

```
$ git clone https://opendev.org/airship/armada.git && cd armada
```

3. Next, run the following commands to install `tox`, generate sample policy and configuration files, and build Armada charts as well as the Armada container image. Armada Dockerfile.DISTRO files are located in images/armada. Supported DISTROs are ubuntu_bionic and opensuse_leap15. By default, DISTRO is ubuntu_bionic.

```
$ pip install tox

$ tox -e genconfig
$ tox -e genpolicy

$ export DISTRO=distro_name
$ docker build -f Dockerfile.${DISTRO} -t armada/latest

$ make images
```

4. Run the container via Docker:

```
$ docker run -d --name armada -v ~/.kube/:/armada/.kube/ -v $(pwd)/etc:/etc␣
↪armada:local
```

Note: The first build will take several minutes. Afterward, it will build much faster.

2.1.2 Manual Installation

Pre-requisites

Armada has many pre-requisites because it relies on Helm, which itself has pre-requisites. The guide below consolidates the installation of all pre-requisites. For help troubleshooting individual resources, reference their installation guides.

Armada requires a Kubernetes cluster to be deployed, along with kubectl, Helm client, and Tiller (the Helm server).

1. Install Kubernetes (k8s) and deploy a k8s cluster.

   Reference the *Kubernetes Cluster Management* section below for help.

2. Install and configure kubectl

3. Ensure that `~/.kube/config` exists and is properly configured by executing:

   ```
   $ kubectl config view
   ```

   If the file does not exist, please create it by running:

   ```
   $ kubectl
   ```

4. Install and configure the Helm client.

5. Install and configure Tiller (Helm server).

6. Verify that Tiller is installed and running correctly by running:

```
$ kubectl get pods -n kube-system
```

Kubernetes Cluster Management

To test Armada fixes/features a Kubernetes cluster must be installed.

Either software is recommended:

- Kubeadm
- Kubeadm-AIO

Armada CLI Installation

Follow the steps below to install the Armada CLI.

Note: Some commands below use `apt-get` as the package management software. Use whichever command corresponds to the Linux distro being used.

Warning: Armada is tested against a Ubuntu 16.04 and Opensuse(leap15.1)environment.

Clone the Armada repository, `cd` into it:

```
git clone https://opendev.org/airship/armada.git && cd armada
```

It is recommended that Armada be run inside a virtual environment. To do so:

```
$ virtualenv -p python3 venv
...
>> New python executable in <...>/venv/bin/python3
```

Afterward, `source` the executable:

```
source <...>/venv/bin/activate
```

Next, ensure that `pip` is installed.

$ apt-get install -y python3-pip $ pip3 install –upgrade pip

Finally, run (from inside the Armada root directory):

```
$ (venv) make build
```

The above command will install `pip` requirements and execute `python setup.py build` within the virtual environment.

Verify that the Armada CLI is installed:

```
$ armada --help
```

Which should emit:

```
>> Usage: armada [OPTIONS] COMMAND [ARGS]...
>>
>>  Multi Helm Chart Deployment Manager
...
```

Armada API Server Installation

The Armada API server is not required in order to use the Armada CLI, which in this sense is standalone. The Armada CLI communicates with the Tiller server and, as such, no API server needs to be instantiated in order for Armada to communicate with Tiller. The Armada API server and CLI interface have the exact same functionality. However, the Armada API server offers the following additional functionality:

- Role-Based Access Control, allowing Armada to provide authorization around specific Armada (and by extension) Tiller functionality.
- Keystone authentication and project scoping, providing an additional layer of security.

Before proceeding, ensure that the steps in *Armada CLI Installation* have been followed.

1. Determine where the Armada configuration/deployment files should be stored. The default location is `/etc/armada`. To override the default, run:

   ```
   $ export OS_ARMADA_CONFIG_DIR=<desired_path>
   ```

2. If the directory specified by `OS_ARMADA_CONFIG_DIR` is empty, run (from the Armada root directory):

```
$ cp etc/armada/* <OS_ARMADA_CONFIG_DIR>/
$ mv <OS_ARMADA_CONFIG_DIR>/armada.conf.sample <OS_ARMADA_CONFIG_DIR>/armada.conf
```

Install `uwsgi`:

```
$ apt-get install uwsgi -y
```

1. Ensure that port 8000 is available or else change the `PORT` value in `entrypoint.sh`.
2. From the root Armada directory, execute:

```
$ ./entrypoint.sh server
```

3. Verify that the Armada server is running by executing:

```
$ TOKEN=$(openstack token issue --format value -c id)
$ curl -i -X GET localhost:8000/versions -H "X-Auth-Token: $TOKEN"
```

Note that the port above uses the default value in `entrypoint.sh`.

2.1.3 Development Utilities

Armada comes equipped with many utilities useful for developers, such as unit test or linting jobs.

Many of these commands require that `tox` be installed. To do so, run:

```
$ pip3 install tox
```

To run the Python linter, execute:

```
$ tox -e pep8

or

$ make test-pep8
```

To lint Helm charts, execute:

```
$ make lint
```

To run unit tests, execute:

```
$ tox -e py35

or

$ make test-unit
```

To run the test coverage job:

```
$ tox -e coverage

or

$ make test-coverage
```

To run security checks via Bandit execute:

```
$ tox -e bandit

or

$ make test-bandit
```

To build the docker images:

```
$ make images
```

To build all Armada charts, execute:

```
$ make charts
```

To build a helm template for the charts:

```
$ make dry-run
```

To run lint, charts, and image targets all at once:

```
$ make all
```

To render any documentation that has build steps:

```
$ tox -e docs

or

$ make docs
```

To build armada's image:

```
$ make run_armada
```

To build all images:

```
$ make run_images
```

To generate sample configuration and policy files needed for Armada deployment, execute (respectively):

```
$ tox -e genconfig
$ tox -e genpolicy
```

2.1.4 Troubleshooting

The error messages are included in bullets below and tips to resolution are included beneath each bullet.

- "FileNotFoundError: [Errno 2] No such file or directory: '/etc/armada/api-paste.ini'"

  Reason: this means that Armada is trying to instantiate the server but failing to do so because it can't find an essential configuration file.

  Solution:

```
$ cp etc/armada/armada.conf.sample /etc/armada/armada.conf
```

This copies the sample Armada configuration file to the appropriate directory.

- For any errors related to `tox`:

  Ensure that `tox` is installed:

```
$ sudo apt-get install tox -y
```

- For any errors related to running `tox -e py35`:

  Ensure that `python3-dev` is installed:

```
$ sudo apt-get install python3-dev -y
```

2.2 Contribution Guidelines

If you would like to contribute to the development of OpenStack, you must follow the steps in this page:

https://docs.openstack.org/infra/manual/developers.html

If you already have a good understanding of how the system works and your OpenStack accounts are set up, you can skip to the development workflow section of this documentation to learn how changes to OpenStack should be submitted for review via the Gerrit tool:

https://docs.openstack.org/infra/manual/developers.html#development-workflow

Pull requests submitted through GitHub will be ignored.

CHAPTER 3

Operations Guide

3.1 Document Authoring Guide

3.1.1 v1

v1 Authoring

armada/Manifest/v1

keyword	type	action
`release_prefix`	string	appends to the front of all charts released by the manifest in order to manage releases throughout their lifecycle
`chart_groups`	array	references ChartGroup document of all groups

Manifest Example

```
---
schema: armada/Manifest/v1
metadata:
  schema: metadata/Document/v1
  name: simple-armada
data:
  release_prefix: armada
  chart_groups:
    - chart_group
```

armada/ChartGroup/v1

keyword	type	action
description	string	description of chart set
chart_group	array	reference to chart document
sequenced	bool	enables sequenced chart deployment in a group
test_charts	bool	run pre-defined helm tests in a ChartGroup (DEPRECATED)

Danger: DEPRECATION: The `test_charts` key will be removed, as Armada will run helm tests for all charts by default.

Chart Group Example

```
---
schema: armada/ChartGroup/v1
metadata:
  schema: metadata/Document/v1
  name: blog-group
data:
  description: Deploys Simple Service
  sequenced: False
  chart_group:
    - chart
    - chart
```

armada/Chart/v1

Danger: DEPRECATION: `timeout` key-value will be removed timeout will be defined under `wait` object.

Chart

keyword	type	action
chart_name	string	name for the chart
release	string	name of the release (Armada will prepend with `release-prefix` during processing)
namespace	string	namespace of your chart
wait	object	See *Wait*.
protected	object	do not delete FAILED releases when encountered from previous run (provide the 'continue_processing' bool to continue or halt execution (default: halt))
test	object	See *Test*.
install	object	install the chart into your Kubernetes cluster
upgrade	object	upgrade the chart managed by the armada yaml
delete	object	See *Delete*.
values	object	override any default values in the charts
source	object	provide a path to a `git repo`, `local dir`, or `tarball url` chart
dependencies	object	(optional) Override the builtin chart dependencies with a list of Chart documents to use as dependencies instead. NOTE: Builtin ".tgz" dependencies are not yet supported.
timeout	int	time (in seconds) allotted for chart to deploy when 'wait' flag is set (DEPRECATED)

Wait

keyword	type	action
timeout	int	time (in seconds) to wait for chart to deploy
resources	array	Array of *Wait Resource* to wait on, with `labels` added to each item. Defaults to pods and jobs (if any exist) matching `labels`.
labels	object	Base mapping of labels to wait on. They are added to any labels in each item in the `resources` array.
native	boolean	See *Wait Native*.

Wait Resource

keyword	type	action
type	string	k8s resource type, supports: controllers ('deployment', 'daemonset', 'statefulset'), 'pod', 'job'
labels	object	mapping of kubernetes resource labels
min_ready	int string	Only for controller `type``s. Amount of pods in a controller which must be ready. Can be integer or percent string e.g. ``80%. Default `100%`.

Wait Native

Config for the native `helm (install|upgrade) --wait` flag.

keyword	type	action
enabled	boolean	defaults to true

Test

Run helm tests on the chart after install/upgrade.

keyword	type	action
enabled	bool	whether to enable/disable helm tests for this chart (default True)
timeout	int	time (in sec) to wait for completion of Helm tests. Default 300.
options	object	See *Test Options*.

Note: Armada will attempt to run helm tests by default. They may be disabled by setting the `enabled` key to `False`.

Danger: DEPRECATION: In addition to an object with the above fields, the `test` key currently also supports `bool`, which maps to `enabled`, but this is deprecated and will be removed. The `cleanup` option below is set to true in this case for backward compatibility.

Test Options

Test options to pass through directly to helm.

keyword	type	action
cleanup	bool	cleanup test pods after test completion, defaults to false

Note: If cleanup is `true` this prevents being able to debug a test in the event of failure.

Historically, the preferred way to achieve test cleanup has been to add a pre-upgrade delete action on the test pod.

This still works, however it is usually no longer necessary as Armada now automatically cleans up any test pods which match the `wait.labels` of the chart, immediately before running tests. Similar suggestions have been made for how `helm test --cleanup` itself ought to work (https://github.com/helm/helm/issues/3279).

Upgrade - Pre

keyword	type	action
pre	object	actions performed prior to updating a release

Upgrade - Actions

keyword	type	action
update	object	update daemonsets in pre-upgrade update actions
delete	sequence	delete jobs and pods in pre-upgrade delete actions

Upgrade - Actions - Update/Delete

keyword	type	action
name	string	name of action
type	string	type of Kubernetes workload to execute in scope for action
labels	object	k:v mapping of labels to select Kubernetes resources

Note: Update Actions only support type: 'daemonset'

Note: Delete Actions support type: 'pod', 'job', 'cronjob'

Chart Example

```
---
schema: armada/Chart/v1
metadata:
  schema: metadata/Document/v1
  name: blog-1
data:
  chart_name: blog-1
  release: blog-1
  namespace: default
  wait:
    timeout: 100
  protected:
    continue_processing: false
  test:
```

(continues on next page)

(continued from previous page)

```
    enabled: true
  install:
    no_hooks: false
  upgrade:
    no_hooks: false
    pre:
      update:
        - name: test-daemonset
          type: daemonset
          labels:
            foo: bar
            component: bar
            rak1: enabled
      delete:
        - name: test-job
          type: job
          labels:
            foo: bar
            component: bar
            rak1: enabled
  values: {}
  source:
    type: git
    location: https://github.com/namespace/repo
    subpath: .
    reference: master
```

Delete

keyword	type	action
timeout	integer	time (in seconds) to wait for chart to be deleted

Source

keyword	type	action
type	string	source to build the chart: `git`, `local`, or `tar`
location	string	`url` or `path` to the chart's parent directory
subpath	string	(optional) relative path to target chart from parent (`.` if not specified)
reference	string	(optional) branch, commit, or reference in the repo (`master` if not specified)
proxy_server	string	(optional) proxy server URL for downloading `git` or `tar` charts

Source Example

```
# type git
---
schema: armada/Chart/v1
metadata:
  schema: metadata/Document/v1
```

(continues on next page)

(continued from previous page)

```
  name: blog-1
data:
  chart_name: blog-1
  release: blog-1
  namespace: default
  wait:
    timeout: 100
    labels:
      component: blog
  install:
    no_hooks: false
  upgrade:
    no_hooks: false
  values: {}
  source:
    type: git
    location: https://github.com/namespace/repo
    subpath: .
    reference: master

# type local
---
schema: armada/Chart/v1
metadata:
  schema: metadata/Document/v1
  name: blog-1
data:
  chart_name: blog-1
  release: blog-1
  namespace: default
  wait:
    timeout: 100
  install:
    no_hooks: false
  upgrade:
    no_hooks: false
  values: {}
  source:
    type: local
    location: /path/to/charts
    subpath: chart
    reference: master

# type tar
---
schema: armada/Chart/v1
metadata:
  schema: metadata/Document/v1
  name: blog-1
data:
  chart_name: blog-1
  release: blog-1
  namespace: default
  wait:
    timeout: 100
  install:
    no_hooks: false
```

(continues on next page)

(continued from previous page)

```
  upgrade:
    no_hooks: false
  values: {}
  source:
    type: tar
    location: https://localhost:8879/charts/chart-0.1.0.tgz
    subpath: mariadb
    reference: null
    proxy_server: http://my.proxy.server:8888
```

Defining a Manifest

To define your Manifest you need to define a `armada/Manifest/v1` document, `armada/ChartGroup/v1` document, `armada/Chart/v1`. Following the definitions above for each document you will be able to construct an armada manifest.

Armada - Deploy Behavior

1. Armada will perform set of pre-flight checks to before applying the manifest - validate input manifest - check tiller service is Running - check chart source locations are valid
2. Deploying Armada Manifest
    1. If the chart is not found
        - we will install the chart
    3. If exist then
        - Armada will check if there are any differences in the chart
        - if the charts are different then it will execute an upgrade
        - else it will not perform any actions

Note: You can use references in order to build your charts, this will reduce the size of the chart definition will show example in multichart below

---

Simple Example

```
---
schema: armada/Chart/v1
metadata:
  schema: metadata/Document/v1
  name: blog-1
data:
  chart_name: blog-1
  release: blog-1
  namespace: default
  values: {}
  source:
    type: git
```

(continues on next page)

(continued from previous page)

```
    location: https://github.com/namespace/repo
    subpath: blog-1
    reference: new-feat
---
schema: armada/ChartGroup/v1
metadata:
  schema: metadata/Document/v1
  name: blog-group
data:
  description: Deploys Simple Service
  sequenced: False
  chart_group:
    - blog-1
---
schema: armada/Manifest/v1
metadata:
  schema: metadata/Document/v1
  name: simple-armada
data:
  release_prefix: armada
  chart_groups:
    - blog-group
```

Multichart Example

```
---
schema: armada/Chart/v1
metadata:
  schema: metadata/Document/v1
  name: blog-1
data:
  chart_name: blog-1
  release: blog-1
  namespace: default
  values: {}
  source:
    type: git
    location: https://github.com/namespace/repo
    subpath: blog1
    reference: master
---
schema: armada/Chart/v1
metadata:
  schema: metadata/Document/v1
  name: blog-2
data:
  chart_name: blog-2
  release: blog-2
  namespace: default
  values: {}
  source:
    type: tar
    location: https://github.com/namespace/repo/blog2.tgz
    subpath: blog2
```

(continues on next page)

(continued from previous page)

```
---
schema: armada/Chart/v1
metadata:
  schema: metadata/Document/v1
  name: blog-3
data:
  chart_name: blog-3
  release: blog-3
  namespace: default
  values: {}
  source:
    type: local
    location: /home/user/namespace/repo/blog3
---
schema: armada/ChartGroup/v1
metadata:
  schema: metadata/Document/v1
  name: blog-group-1
data:
  description: Deploys Simple Service
  sequenced: False
  chart_group:
    - blog-2
---
schema: armada/ChartGroup/v1
metadata:
  schema: metadata/Document/v1
  name: blog-group-2
data:
  description: Deploys Simple Service
  sequenced: False
  chart_group:
    - blog-1
    - blog-3
---
schema: armada/Manifest/v1
metadata:
  schema: metadata/Document/v1
  name: simple-armada
data:
  release_prefix: armada
  chart_groups:
    - blog-group-1
    - blog-group-2
```

Dependency Override Example

```
---
schema: armada/Chart/v1
metadata:
  schema: metadata/Document/v1
  name: blog-1
data:
  chart_name: blog-1
```

(continues on next page)

(continued from previous page)

```
  release: blog-1
  namespace: default
  values: {}
  source:
    type: git
    location: https://github.com/namespace/repo
    subpath: blog-1
    reference: new-feat
  dependencies:
    - blog-1-dep
---
schema: armada/Chart/v1
metadata:
  schema: metadata/Document/v1
  name: blog-1-dep
data:
  chart_name: blog-1-dep
  release: blog-1-dep
  namespace: default
  values: {}
  source:
    type: git
    location: https://github.com/namespace/dep-repo
    subpath: blog-1-dep
    reference: new-feat
---
schema: armada/ChartGroup/v1
metadata:
  schema: metadata/Document/v1
  name: blog-group
data:
  description: Deploys Simple Service
  sequenced: False
  chart_group:
    - blog-1
---
schema: armada/Manifest/v1
metadata:
  schema: metadata/Document/v1
  name: simple-armada
data:
  release_prefix: armada
  chart_groups:
    - blog-group
```

References

For working examples please check the examples in our repo here.

v1 Schemas

Below are the schemas Armada uses to validate Charts, Chart Groups, and Manifests.

Charts

Charts consist of the smallest building blocks in Armada. A `Chart` is comparable to a Helm chart. Charts consist of all the labels, dependencies, install and upgrade information, hooks and additional information needed to convey to Tiller.

Chart Groups

A `Chart Group` consists of a list of charts. `Chart Group` documents are useful for managing a group of `Chart` documents together.

Manifests

A `Manifest` is the largest building block in Armada. `Manifest` documents are responsible for managing collections of `Chart Group` documents.

Validation Schemas

Introduction

All schemas below are Deckhand DataSchema documents, which are essentially JSON schemas, with additional metadata useful for Deckhand to perform layering and substitution.

The validation schemas below are used by Armada to validate all ingested Charts, Chart Groups, and Manifests. Use the schemas below as models for authoring Armada documents.

Schemas

- `Chart` schema.

  JSON schema against which all documents with `armada/Chart/v1` `metadata.name` are validated.

Listing 1: Schema for `armada/Chart/v1` documents.

```
# NOTE: Do not modify this schema, it is deprecated.
---
schema: deckhand/DataSchema/v1
metadata:
  name: armada/Chart/v1
  schema: metadata/Control/v1
data:
  $schema: http://json-schema.org/schema#
  definitions:
    labels:
      type: object
      additionalProperties:
        type: string
    hook_action:
      type: array
      items:
        properties:
```

(continues on next page)

(continued from previous page)

```
        name:
          type: string
        type:
          type: string
        labels:
          $ref: '#/definitions/labels'
      required:
        - type
      additionalProperties: false
type: object
properties:
  release:
    type: string
  chart_name:
    type: string
  namespace:
    type: string
  values:
    type: object
  dependencies:
    type: array
    items:
      type: string
  protected:
    type: object
    properties:
      continue_processing:
        type: boolean
    additionalProperties: false
  test:
    anyOf:
      - type: boolean
      - type: object
        properties:
          enabled:
            type: boolean
          timeout:
            type: integer
          options:
            type: object
            properties:
              cleanup:
                type: boolean
            additionalProperties: false
        additionalProperties: false
  timeout:
    type: integer
  wait:
    type: object
    properties:
      timeout:
        type: integer
      resources:
        type: array
        items:
          properties:
            type:
```

(continues on next page)

(continued from previous page)

```
              type: string
            labels:
              $ref: '#/definitions/labels'
            min_ready:
              anyOf:
                - type: integer
                - type: string
          required:
            - type
          additionalProperties: false
      labels:
        $ref: "#/definitions/labels"
      # Config for helm's native `--wait` param.
      native:
        type: object
        properties:
          # TODO: Add separate timeout for native wait?
          enabled:
            type: boolean
        additionalProperties: false
    additionalProperties: false
  source:
    type: object
    properties:
      type:
        type: string
      location:
        type: string
      subpath:
        type: string
      reference:
        type: string
      proxy_server:
        type: string
      auth_method:
        type: string
    required:
      - location
      - subpath
      - type
  delete:
    type: object
    properties:
      timeout:
        type: integer
  install:
    # NOTE(sh8121att) Not clear that this key is actually used
    # in the code. Will leave it here for backward compatabilities
    # until an additional audit is done.
    type: object
  upgrade:
    type: object
    properties:
      no_hooks:
        type: boolean
      pre:
        type: object
```

(continues on next page)

(continued from previous page)

```
          additionalProperties: false
          properties:
            delete:
              $ref: '#/definitions/hook_action'
            update:
              $ref: '#/definitions/hook_action'
            create:
              $ref: '#/definitions/hook_action'
        post:
          type: object
          additionalProperties: false
          properties:
            create:
              $ref: '#/definitions/hook_action'
        options:
          type: object
          properties:
            force:
              type: boolean
            recreate_pods:
              type: boolean
          additionalProperties: false
      required:
        - no_hooks
      additionalProperties: false
  required:
    - namespace
    - chart_name
    - release
    - source
  additionalProperties: false
...
```

This schema is used to sanity-check all `Chart` documents that are passed to Armada.

- `Chart Group` schema.

  JSON schema against which all documents with `armada/Chart/v1` `metadata.name` are validated.

Listing 2: Schema for `armada/ChartGroup/v1` documents.

```
# NOTE: Do not modify this schema, it is deprecated.
---
schema: deckhand/DataSchema/v1
metadata:
  name: armada/ChartGroup/v1
  schema: metadata/Control/v1
data:
  $schema: http://json-schema.org/schema#
  properties:
    name:
      type: string
    description:
      type: string
    sequenced:
      type: boolean
    # TODO(MarshM): Deprecate `test_charts`, it is no longer useful
```

(continues on next page)

(continued from previous page)

```
    test_charts:
      type: boolean
    chart_group:
      type: array
      items:
        type: string
  required:
    - chart_group
  additionalProperties: false
...
```

This schema is used to sanity-check all `Chart Group` documents that are passed to Armada.

- `Manifest` schema.

  JSON schema against which all documents with `armada/Manifest/v1` `metadata.name` are validated.

Listing 3: Schema for `armada/Manifest/v1` documents.

```
# NOTE: Do not modify this schema, it is deprecated.
---
schema: deckhand/DataSchema/v1
metadata:
  name: armada/Manifest/v1
  schema: metadata/Control/v1
data:
  $schema: http://json-schema.org/schema#
  properties:
    release_prefix:
      type: string
    chart_groups:
      type: array
      items:
        type: string
  required:
    - chart_groups
    - release_prefix
  additionalProperties: false
...
```

This schema is used to sanity-check all `Manifest` documents that are passed to Armada.

Authoring Guidelines

All Armada documents must use the `deckhand/DataSchema/v1` schema.

3.1.2 v2 (EXPERIMENTAL!)

v2 Authoring

Danger: EXPERIMENTAL: *v2* docs are still experimental and WILL have breaking changes before they are finalized.

armada/Manifest/v2

keyword	type	action
`release_prefix`	string	appends to the front of all charts released by the manifest in order to manage releases throughout their lifecycle
`chart_groups`	array	A list of the `metadata.name` of each `ChartGroup` to deploy in order.

Manifest Example

```
---
schema: armada/Manifest/v2
metadata:
  schema: metadata/Document/v1
  name: simple-armada
data:
  release_prefix: armada
  chart_groups:
    - chart_group
```

armada/ChartGroup/v2

keyword	type	action
description	string	description of chart set
chart_group	array	A list of the `metadata.name` of each `Chart` to deploy.
sequenced	bool	If `true`, deploys each chart in sequence, else in parallel. Default `false`.

Chart Group Example

```
---
schema: armada/ChartGroup/v2
metadata:
  schema: metadata/Document/v1
  name: blog-group
data:
  description: Deploys Simple Service
  chart_group:
    - chart1
    - chart2
```

armada/Chart/v2

Chart

keyword	type	action
release	string	name of the release (Armada will prepend with `release-prefix` during processing)
namespace	string	namespace of your chart
wait	object	See *Wait*.
protected	object	do not delete FAILED releases when encountered from previous run (provide the 'continue_processing' bool to continue or halt execution (default: halt))
test	object	See *Test*.
upgrade	object	See *Upgrade*.
delete	object	See *Delete*.
values	object	(optional) override any default values in the charts
source	object	provide a path to a `git repo`, `local dir`, or `tarball url` chart
dependencies	object	(optional) Override the builtin chart dependencies with a list of Chart documents to use as dependencies instead. NOTE: Builtin ".tgz" dependencies are not yet supported.

Wait

keyword	type	action
timeout	int	time (in seconds) to wait for chart to deploy
resources	dict | array	*Wait Resource* s to wait on. Defaults to all supported resource types (see *Wait Resource* `.type`), with `required: false`. **dict** - Maps resource types to one of: • *Wait Resource* without `.type` (single config) • list of *Wait Resource* without `.type` (multiple configs) • `false` (disabled) Any resource type not overridden retains its default config mentioned above. **array** - Lists all *Wait Resource* s to use, completely overriding the default. Can be set to `[]` to disable all resource types. See also *Wait Resources Examples*.
labels	object	Base mapping of labels to wait on. They are added to any labels in each item in the `resources` array.
native	boolean	See *Wait Native*.

Wait Resource

keyword	type	action
type	string	K8s resource type, supports: 'deployment', 'daemonset', 'statefulset', 'pod', 'job'. NOTE: Omit when *Wait* `.resources` is a dict, as then the dict key is used instead.
labels	object	Kubernetes labels specific to this resource. *Wait* `.labels` are included with these, so only define this if additional labels are needed to identify the targeted resources.
min_ready	int | string	Only for controller `type` s. Amount of pods in a controller which must be ready. Can be integer or percent string e.g. `80%`. Default `100%`.
allow_async_updates	boolean	Only for `daemonset` and `statefulset` types. Whether to wait for async update strategies, i.e. OnDelete or partitioned RollingUpdate. Defaults to `false` in order to fail fast in cases where the async update is not expected to complete until same point later on.
required	boolean	Whether to require the resource to be found. Defaults to `true` for explicit items in `wait.resources`. See `wait.resources` for its overall defaults.

Wait Resources Examples

```
wait:
  # ...
  # Disable all waiting.
  resources: []
```

```
wait:
  # ...
  # Disable waiting for a given type (job).
  resources:
    job: false
```

```
wait:
  # ...
  # Use min_ready < 100%.
  resources:
    daemonset:
      min_ready: 80%
```

```
wait:
  resources:
    # Multiple configs for same type.
    daemonset:
      - labels:
          component: one
        min_ready: 80%
      - labels:
          component: two
        min_ready: 50%
```

```
wait:
  # ...
  resources:
    - type: daemonset
      labels:
        component: critical
      min_ready: 100%
    - type: daemonset
      labels:
        component: best_effort
      min_ready: 80%
    # ... (re-include any other resource types needed when using list)
```

Wait Native

Config for the native `helm (install|upgrade) --wait` flag.

keyword	type	action
enabled	boolean	defaults to `false`

Test

Run helm tests on the chart after install/upgrade.

keyword	type	action
enabled	bool	whether to enable/disable helm tests for this chart (default True)
timeout	int	time (in sec) to wait for completion of Helm tests. Default 300.
options	object	See *Test Options*.

Note: Armada will attempt to run helm tests by default. They may be disabled by setting the `enabled` key to `false`.

Test Options

Test options to pass through directly to helm.

keyword	type	action
cleanup	bool	Same as Helm CLI.

Note: If cleanup is `true` this prevents being able to debug a test in the event of failure.

Historically, the preferred way to achieve test cleanup has been to add a pre-upgrade delete action on the test pod.

This still works, however it is usually no longer necessary as Armada now automatically cleans up any test pods which match the `wait.labels` of the chart, immediately before running tests. Similar suggestions have been made for how `helm test --cleanup` itself ought to work (https://github.com/helm/helm/issues/3279).

Upgrade

keyword	type	action
options	object	See *Upgrade - Options*.
pre	object	See *Upgrade - Pre*.

Upgrade - Options

Upgrade options to pass through directly to helm.

keyword	type	action
no_hooks	boolean	Same as Helm CLI.
force	boolean	Same as Helm CLI.
recreate_pods	boolean	Same as Helm CLI.

Upgrade - Pre

keyword	type	action
delete	array	List of *Upgrade - Pre - Delete*.

Upgrade - Pre - Delete

keyword	type	action
type	string	type of kubernetes resource to delete supported types are: 'pod', 'job', 'cronjob'.
labels	object	k:v mapping of labels to select Kubernetes resources

Chart Example

```
---
schema: armada/Chart/v2
metadata:
  schema: metadata/Document/v1
  name: blog-1
data:
  release: blog-1
  namespace: default
  wait:
    timeout: 100
  protected:
    continue_processing: false
  test:
    enabled: true
  upgrade:
    pre:
      delete:
        - name: test-job
          type: job
          labels:
            foo: bar
            component: bar
            rak1: enabled
  source:
    type: git
    location: https://github.com/namespace/repo
    reference: master
```

Delete

keyword	type	action
timeout	integer	time (in seconds) to wait for chart to be deleted

Source

keyword	type	action
type	string	source to build the chart: `git`, `local`, or `tar`
location	string	`url` or `path` to the chart's parent directory
subpath	string	(optional) relative path to target chart from parent (`.` if not specified)
reference	string	(optional) branch, commit, or reference in the repo (`master` if not specified)
proxy_server	string	(optional) proxy server URL for downloading `git` or `tar` charts

Source Example

```
# type git
---
schema: armada/Chart/v2
metadata:
  schema: metadata/Document/v1
  name: blog-1
data:
  release: blog-1
  namespace: default
  wait:
    timeout: 100
    labels:
      component: blog
  source:
    type: git
    location: https://github.com/namespace/repo
    proxy_server: http://my.proxy.server:8888

# type local
---
schema: armada/Chart/v2
metadata:
  schema: metadata/Document/v1
  name: blog-1
data:
  release: blog-1
  namespace: default
  wait:
    timeout: 100
  source:
    type: local
    location: /path/to/charts
    subpath: chart
    reference: master

# type tar
---
schema: armada/Chart/v2
metadata:
  schema: metadata/Document/v1
  name: blog-1
data:
  release: blog-1
```

(continues on next page)

(continued from previous page)

```
  namespace: default
  wait:
    timeout: 100
  source:
    type: tar
    location: https://localhost:8879/charts/chart-0.1.0.tgz
    subpath: mariadb
```

Simple Example

```
---
schema: armada/Chart/v2
metadata:
  schema: metadata/Document/v1
  name: blog-1
data:
  release: blog-1
  namespace: default
  source:
    type: git
    location: https://github.com/namespace/repo
    subpath: blog-1
    reference: new-feat
---
schema: armada/ChartGroup/v2
metadata:
  schema: metadata/Document/v1
  name: blog-group
data:
  description: Deploys Simple Service
  chart_group:
    - blog-1
---
schema: armada/Manifest/v2
metadata:
  schema: metadata/Document/v1
  name: simple-armada
data:
  release_prefix: armada
  chart_groups:
    - blog-group
```

Multichart Example

```
---
schema: armada/Chart/v2
metadata:
  schema: metadata/Document/v1
  name: blog-1
data:
  release: blog-1
  namespace: default
```

(continues on next page)

(continued from previous page)

```
  source:
    type: git
    location: https://github.com/namespace/repo
    subpath: blog1
    reference: master
---
schema: armada/Chart/v2
metadata:
  schema: metadata/Document/v1
  name: blog-2
data:
  release: blog-2
  namespace: default
  source:
    type: tar
    location: https://github.com/namespace/repo/blog2.tgz
    subpath: blog2
---
schema: armada/Chart/v2
metadata:
  schema: metadata/Document/v1
  name: blog-3
data:
  release: blog-3
  namespace: default
  source:
    type: local
    location: /home/user/namespace/repo/blog3
---
schema: armada/ChartGroup/v2
metadata:
  schema: metadata/Document/v1
  name: blog-group-1
data:
  description: Deploys Simple Service
  chart_group:
    - blog-2
---
schema: armada/ChartGroup/v2
metadata:
  schema: metadata/Document/v1
  name: blog-group-2
data:
  description: Deploys Simple Service
  chart_group:
    - blog-1
    - blog-3
---
schema: armada/Manifest/v2
metadata:
  schema: metadata/Document/v1
  name: simple-armada
data:
  release_prefix: armada
  chart_groups:
    - blog-group-1
    - blog-group-2
```

Dependency Override Example

```
---
schema: armada/Chart/v2
metadata:
  schema: metadata/Document/v1
  name: blog-1
data:
  release: blog-1
  namespace: default
  source:
    type: git
    location: https://github.com/namespace/repo
    subpath: blog-1
    reference: new-feat
  dependencies:
    - blog-dep-1
---
schema: armada/Chart/v2
metadata:
  schema: metadata/Document/v1
  name: blog-1-dep
data:
  release: blog-1-dep
  namespace: default
  source:
    type: git
    location: https://github.com/namespace/dep-repo
    subpath: blog-1-dep
    reference: new-feat
---
schema: armada/ChartGroup/v2
metadata:
  schema: metadata/Document/v1
  name: blog-group
data:
  description: Deploys Simple Service
  chart_group:
    - blog-1
---
schema: armada/Manifest/v2
metadata:
  schema: metadata/Document/v1
  name: simple-armada
data:
  release_prefix: armada
  chart_groups:
    - blog-group
```

References

For working examples please check the examples in our repo here.

v2 Schemas

Below are the schemas Armada uses to validate Charts, Chart Groups, and Manifests.

Charts

Charts consist of the smallest building blocks in Armada. A `Chart` is comparable to a Helm chart. Charts consist of all the labels, dependencies, install and upgrade information, hooks and additional information needed to convey to Tiller.

Chart Groups

A `Chart Group` consists of a list of charts. `Chart Group` documents are useful for managing a group of `Chart` documents together.

Manifests

A `Manifest` is the largest building block in Armada. `Manifest` documents are responsible for managing collections of `Chart Group` documents.

Validation Schemas

Introduction

All schemas below are Deckhand DataSchema documents, which are essentially JSON schemas, with additional metadata useful for Deckhand to perform layering and substitution.

The validation schemas below are used by Armada to validate all ingested Charts, Chart Groups, and Manifests. Use the schemas below as models for authoring Armada documents.

Schemas

- `Chart` schema.

  JSON schema against which all documents with `armada/Chart/v2` `metadata.name` are validated.

Listing 4: Schema for `armada/Chart/v2` documents.

```
# JSON schema for validating Armada charts.
---
schema: deckhand/DataSchema/v1
metadata:
  name: armada/Chart/v2
  schema: metadata/Control/v1
data:
  $schema: http://json-schema.org/schema#
  definitions:
    labels:
      type: object
      additionalProperties:
```

(continues on next page)

(continued from previous page)

```
        type: string
    hook_action:
      type: array
      items:
        properties:
          type:
            type: string
          labels:
            $ref: '#/definitions/labels'
        required:
          - type
        additionalProperties: false
    wait_resource_type_config:
      properties:
        labels:
          $ref: '#/definitions/labels'
        min_ready:
          anyOf:
            - type: integer
            - type: string
        required:
          type: boolean
  type: object
  properties:
    release:
      type: string
    namespace:
      type: string
    values:
      type: object
    # TODO: Remove this, and just read dependencies out of `chart` dir as helm
    # CLI does.
    dependencies:
      type: array
      items:
        type: string
    protected:
      type: object
      properties:
        continue_processing:
          type: boolean
      additionalProperties: false
    test:
      type: object
      properties:
        enabled:
          type: boolean
        timeout:
          type: integer
        options:
          type: object
          properties:
            cleanup:
              type: boolean
          additionalProperties: false
      additionalProperties: false
    wait:
```

(continues on next page)

(continued from previous page)

```
      type: object
      properties:
        timeout:
          type: integer
        resources:
          anyOf:
            - additionalProperties:
                anyOf:
                  - $ref: '#/definitions/wait_resource_type_config'
                  - type: array
                    items:
                      $ref: '#/definitions/wait_resource_type_config'
            - type: array
              items:
                allOf:
                  - $ref: '#/definitions/wait_resource_type_config'
                  - properties:
                      type:
                        type: string
                    required:
                      - type
        labels:
          $ref: "#/definitions/labels"
        # Config for helm's native `--wait` param.
        native:
          type: object
          properties:
            enabled:
              type: boolean
          additionalProperties: false
      additionalProperties: false
    source:
      type: object
      properties:
        type:
          type: string
        location:
          type: string
        subpath:
          type: string
        reference:
          type: string
        proxy_server:
          type: string
        auth_method:
          type: string
      required:
        - location
        - type
    delete:
      type: object
      properties:
        timeout:
          type: integer
    upgrade:
      type: object
      properties:
```

(continues on next page)

(continued from previous page)

```
        pre:
          type: object
          additionalProperties: false
          properties:
            delete:
              $ref: '#/definitions/hook_action'
        options:
          type: object
          properties:
            force:
              type: boolean
            recreate_pods:
              type: boolean
            no_hooks:
              type: boolean
          additionalProperties: false
      additionalProperties: false
  required:
    - namespace
    - release
    - source
  additionalProperties: false
...
```

This schema is used to sanity-check all `Chart` documents that are passed to Armada.

- `Chart Group` schema.

  JSON schema against which all documents with `armada/Chart/v2` `metadata.name` are validated.

Listing 5: Schema for `armada/ChartGroup/v2` documents.

```
# JSON schema for validating Armada chart groups.
---
schema: deckhand/DataSchema/v1
metadata:
  name: armada/ChartGroup/v2
  schema: metadata/Control/v1
data:
  $schema: http://json-schema.org/schema#
  properties:
    name:
      type: string
    description:
      type: string
    sequenced:
      type: boolean
    chart_group:
      type: array
      items:
        type: string
  required:
    # TODO: Rename to `charts`?
    - chart_group
  additionalProperties: false
...
```

This schema is used to sanity-check all `Chart Group` documents that are passed to Armada.

- `Manifest` schema.

  JSON schema against which all documents with `armada/Manifest/v2` `metadata.name` are validated.

Listing 6: Schema for `armada/Manifest/v2` documents.

```
# JSON schema for validating Armada manifests.
---
schema: deckhand/DataSchema/v1
metadata:
  name: armada/Manifest/v2
  schema: metadata/Control/v1
data:
  $schema: http://json-schema.org/schema#
  properties:
    release_prefix:
      type: string
    chart_groups:
      type: array
      items:
        type: string
  required:
    - chart_groups
    - release_prefix
  additionalProperties: false
...
```

This schema is used to sanity-check all `Manifest` documents that are passed to Armada.

Authoring Guidelines

All Armada documents must use the `deckhand/DataSchema/v1` schema.

3.1.3 v1-v2 Migration

The following migrations must be done when moving from *v1* to *v2* docs.

Chart

change	migration
`chart_name` removed	Remove. It was redundant with `metadata.name` while at the same time not guaranteeing uniqueness. Log messages now reference `metadata.name` for improved grep-ability.
`test` as a boolean removed	*test* must now be an object.
`timeout` removed	Use `wait.timeout` instead.
`install` removed	Remove. Previously unused.
`upgrade.post` removed	Remove.
`upgrade.pre.update` removed	Remove.
`upgrade.pre.create` removed	Remove.
`upgrade.pre.delete[*].name` removed	Remove.
`upgrade.pre.delete[*]` with `type: job` no longer deletes cronjobs	If you have an item in `upgrade.pre.delete` and `type: job` and you also want to delete cronjobs, add another item with `type: cronjob` and same labels.
`upgrade.no_hooks` moved to `upgrade.options.no_hooks`, and now optional	Remove as desired, otherwise move to the new location.
`source.subpath` now optional, deafults to no subpath.	Remove as desired.
`wait` improvements	See *Wait Improvements*.

Wait Improvements

The *v2 wait API* includes the following changes.

Breaking changes

1. `wait.resources` now defaults to all supported resource types, currently `job`, `daemonset`, `statefulset`, `deployment`, and `pod`, with `required` (a new option) set to `false`. The previous default was the equivalent of pods with `required=true`, and jobs with `required=false`.
2. `type: pod` waits now exclude pods owned by other resources, such as controllers, as one should instead wait directly on the controller itself, which per 1. is now the default.
3. Waits are no longer retried due to resources having been modified. This was mildly useful before as an indicator of whether all targeted resources were accounted for, but with 1. and 2. above, we are now tracking top-level resources directly included in the release, rather than generated resources, such as controller-owned pods, so there is no need to wait for them to come into existence.
4. `wait.native.enabled` is now disabled by default. With the above changes, this is no longer useful as a backup mechanism. Having both enabled leads to ambiguity in which wait would fail in each case. More importantly, this must be disabled in order to use the `min_ready` functionality, otherwise tiller will wait for 100% anyway. So this prevents accidentally leaving it enabled in that case. Also when the tiller native wait times out, this caused the release to be marked FAILED by tiller, which caused it to be purged and re-installed (unless protected), even though the wait criteria may have eventually succeeded, which is already validated by armada on a retry.

New features

Per-resource-type overrides

`wait.resources` can now be a dict, mapping individual resource types to wait configurations (or lists thereof), such that one can keep the default configuration for the other resource types, and also disable a given resource type, by mapping it to `false`.

The ability to provide the entire explicit list for `wait.resources` remains in place as well.

required

A `required` field is also exposed for items/values in `wait.resources`.

allow_async_updates

An `allow_async_updates` field is added to daemonset and statefulset type items/values in `wait.resources`.

ChartGroup

change	migration
`test_charts` removed	Use the Chart schema's *test.enabled* instead.

Manifest

No changes.

3.2 Configuring Armada

Armada uses an INI-like standard oslo_config file. A sample file can be generated via tox

```
$ tox -e genconfig
$ tox -e genpolicy
```

Customize your configuration based on the information below

3.2.1 Keystone Integration

Armada requires a service account to use for validating API tokens

---

Note: If you do not have a keystone already deploy, then armada can deploy a keystone services:

$ armada apply keystone-manifest.yaml

```
$ openstack domain create 'ucp'
$ openstack project create --domain 'ucp' 'service'
$ openstack user create --domain ucp --project service --project-domain 'ucp' --
↪password armada armada
$ openstack role add --project-domain ucp --user-domain ucp --user armada --project␣
↪service admin

# OR

$ ./tools/keystone-account.sh
```

The service account must then be included in the armada.conf

```
[keystone_authtoken]
auth_type = password
auth_uri = https://<keystone-api>:5000/
auth_url = https://<keystone-api>:35357/
auth_version = 3
delay_auth_decision = true
password = armada
project_domain_name = ucp
project_name = service
user_domain_name = ucp
user_name = armada
```

3.3 Armada - Troubleshooting

3.3.1 Debugging Pods

Before starting to work in armada we need to check that the tiller pod is active and running.

```
kubectl get pods -n kube-system | grep tiller
```

```
armada tiller --status
```

3.3.2 Checking Logs

In order to check the logs the logs file will be in *~/.armada* directory.

When running Armada in the container you can execute docker logs to retrieve logs

```
docker logs [container-name | container-id]
```

3.3.3 Errors/Exceptions

A guide for interpreting errors/exceptions can be found here.

3.3.4 Working with SSL

You might run into SSL error with armada if you are not using the correct versions of SSL.

Debugging Checklist:

1. python -c "import ssl; print ssl.OPENSSL_VERSION"

   If the version that appears is less than 1.0, then problems will occur. Please update to current or use our docker container solve this issue

2. check your urllib3 version, you could run into urllib3 issues. older versions of this lib can cause SSL errors run `pip install --upgrade urllib3` and it should solve this issue

3.3.5 Issue

If the issue that you are having does not appear here please check the Armada issues on StoryBoard. If the issue does not exist, please create an issue.

3.4 Armada Install & Usage Guide

3.4.1 Prerequisites

Kubernetes Cluster

Tiller Service

Armada documents

Note: Need to have provided a storage system prior(ceph, nfs)

3.4.2 Usage

Note: The apply command performs two main actions: installing and updating define charts in the armada manifest

1. Pull or Build the Armada Docker Images:

```
Pull:

docker pull quay.io/airshipit/armada:latest-ubuntu_bionic

Build:

git clone https://opendev.org/airship/armada.git && cd armada/
docker build . -t quay.io/airshipit/armada:latest-ubuntu_bionic
```

2. Running Armada

   a. docker container

Note: Make sure to mount your kubeconfig into `/armada/.kube/config` in the container

Note: To run you custom Armada.yamls you need to mount them into the container as shown below. This example is using `examples/` directory in armada repo.

```
docker run -d --net host -p 8000:8000 --name armada -v $(pwd)/etc/:/etc/ -v ~/.kube/:/
↪armada/.kube/ -v $(pwd)/examples/:/examples quay.io/airshipit/armada:latest-ubuntu_
↪bionic
docker exec armada armada --help
```

b. Helm Install

Note: To install Armada via the Helm chart please make sure to provide a Keystone endpoint

```
make charts

helm install <registry>/armada --name armada --namespace armada
```

3. Check that tiller is Available

```
docker exec armada armada tiller --status
```

4. If tiller is up then we can start deploying our armada yamls

```
docker exec armada armada apply /examples/openstack-helm.yaml [ --debug ]
```

5. Upgrading charts: modify the armada yaml or chart source code and run `armada apply` above

```
docker exec armada armada apply /examples/openstack-helm.yaml [ --debug ]
```

6. To check deployed releases:

```
docker exec armada armada tiller --releases
```

7. Testing Releases:

```
docker exec armada armada test --release=armada-keystone

OR

docker exec armada armada test --file=/examples/openstack-helm.yaml
```

8. Rolling back Releases:

```
docker exec armada armada rollback --release=armada-keystone
```

3.4.3 Overriding Manifest Values

It is possible to override manifest values from the command line using the –set and –values flags. When using the set flag, the document type should be specified first, with the target values following in this manner:

```
armada apply --set [ document_type ]:[ document_name ]:[ data_value ]=[ value ]

Example:
```

(continues on next page)

(continued from previous page)

```
armada apply --set chart:blog-1:release="new-blog"
armada apply --set chart:blog-1:values.blog.new="welcome"
```

Note: When overriding values using the set flag, new values will be inserted if they do not exist. An error will only occur if the correct pattern is not used.

There are three types of override types that can be specified: - chart - chart_group - manifest

An example of overriding the location of a chart:

```
armada apply --set chart:[ chart_name ]:source.location=test [ FILE ]

Example:

armada apply --set chart:blog-1:release=test [ FILE ]
```

An example of overriding the description of a chart group:

```
armada apply --set chart_group:[ chart_group_name ]:description=test [ FILE ]

Example:

armada apply examples/simple.yaml --set chart_group:blog-group:description=test
```

An example of overriding the release prefix of a manifest:

```
armada apply --set manifest:[ manifest_name ]:release_prefix=[ value ] [ FILE ]

Example:

armada apply example/simple.yaml --set manifest:simple-armada:release_prefix=armada-2
```

Note: The –set flag can be used multiple times.

It is also possible to override manifest values using values specified in a yaml file using the –values flag. When using the –values flag, a path to the yaml file should be specified in this format:

```
armada apply --values [ path_to_yaml ] [ FILE ]

Example:

armada apply examples/simple.yaml --values examples/simple-ovr-values.yaml
```

Note: The –values flag, like the –set flag, can be specified more than once. The –set and –values flag can also be specified at the same time; however, overrides specified by the –set flag take precedence over those specified by the –values flag.

When creating a yaml file of override values, it should be the same as creating an armada manifest overriding documents with the same schema and metadata name for example:

```
---
schema: armada/Chart/v1
metadata:
  schema: metadata/Document/v1
  name: blog-1
data:
  release: chart-example
  namespace: blog-blog
---
schema: armada/Chart/v1
metadata:
  schema: metadata/Document/v1
  name: blog-2
data:
  release: chart-example-2
  namespace: blog-blog
---
schema: armada/ChartGroup/v1
metadata:
  schema: metadata/Document/v1
  name: blog-group
data:
  description: Change value deploy
  chart_group:
    - blog-1
```

3.4.4 User bearer token

It is possible to pass the user bearer token from the armada CLI to interact with a kubernetes cluster that has been configured with an external Auth-backend like openstack-keystone.

```
Example:

armada apply --bearer-token [ TOKEN ] --values [ path_to_yaml ] [ FILE ]

armada tiller --bearer-token [ TOKEN ] --status
```

Note: The bearer token option is available for the following commands

armada apply, armada delete, armada tiller, armada rollback

3.5 Metrics

Armada exposes metric data, for consumption by Prometheus.

3.5.1 Exporting

Metric data can be exported via:

- API: Prometheus exporter in the */metrics* endpoint. The Armada chart includes the appropriate Prometheus scrape configurations for this endpoint.

- CLI: *–metrics-output=<path>* of *apply* command. The node exporter text file collector can then be used to export the produced text files to Prometheus.

3.5.2 Metric Names

Metric names are as follows:

*armada_* + <action> + _ + <metric>

3.5.3 Supported <action>s

The below tree of <action>s are measured. Supported prometheus labels are noted. Labels are inherited by sub-actions except as noted.

- *apply*:
  - description: apply a manifest
  - labels: *manifest*
  - sub-actions:
    - *chart_handle*:
      - description: fully handle a chart (see below sub-actions)
      - labels:
      - *chart*
      - *action* (install|upgrade|noop) (not included in sub-actions)
      - sub-actions:
      - *chart_download*
      - *chart_deploy*
      - *chart_test*
    - *chart_delete*:
      - description: delete a chart (e.g. due to *FAILED* status)
      - labels: *chart*

3.5.4 Supported <metric>s

- *failure_total*: total failed attempts
- *attempt_total*: total attempts
- *attempt_inprogress*: total attempts in progress
- *duration_seconds*: duration of each attempt

Timeouts

The *chart_handle* and *chart_test* actions additionally include the following metrics:

- *timeout_duration_seconds*: configured chart timeout duration in seconds
- *timeout_usage_ratio*: = *duration_seconds / timeout_duration_seconds*

These can help identify charts whose timeouts may need to be changed to avoid potential failures or to acheive faster failures.

Chart concurrency

The *chart_handle* action additionally includes the following metric:

- *concurrency_count*: count of charts being handled concurrently

This can help identify opportunities for greater chart concurrency.

3.6 Exceptions Guide

3.6.1 Armada Exceptions

API Exceptions

Armada Exceptions

Base Exceptions

Chartbuilder Exceptions

Kubernetes Exceptions

Manifest Exceptions

Override Exceptions

Source Exceptions

Tiller Exceptions

Lint (Validate) Exceptions

3.7 Armada Plugin

The armada plugin extends all the functionality of Armada to be used as a plugin with Helm.

3.7.1 Install Plugin

Install directly from the repository

```
helm plugin install https://opendev.org/airship/armada.git
```

Clone and install locally

```
git clone https://opendev.org/airship/armada.git ~/.helm/plugins/
helm plugin install ~/.helm/plugins/armada
```

3.7.2 Usage

helm <Name> <action> [options]

```
helm armada tiller --status
helm armada apply ~/.helm/plugins/armada/examples/simple.yaml
```

3.8 Sample Configuration File

The following is a sample Armada configuration for adaptation and use. It is auto-generated from Armada when this documentation is built, so if you are having issues with an option, please compare your version of Armada with the version of this documentation.

The sample configuration can also be viewed in file form.

```
[DEFAULT]

#
# From oslo.log
#

# If set to true, the logging level will be set to DEBUG instead of the default
# INFO level. (boolean value)
# Note: This option can be changed without restarting.
#debug = false

# The name of a logging configuration file. This file is appended to any
# existing logging configuration files. For details about logging configuration
# files, see the Python logging module documentation. Note that when logging
# configuration files are used then all logging configuration is set in the
# configuration file and other logging configuration options are ignored (for
# example, log-date-format). (string value)
# Note: This option can be changed without restarting.
# Deprecated group/name - [DEFAULT]/log_config
#log_config_append = <None>

# Defines the format string for %%(asctime)s in log records. Default:
# %(default)s . This option is ignored if log_config_append is set. (string
# value)
#log_date_format = %Y-%m-%d %H:%M:%S

# (Optional) Name of log file to send logging output to. If no default is set,
# logging will go to stderr as defined by use_stderr. This option is ignored if
```

(continues on next page)

(continued from previous page)

```
# log_config_append is set. (string value)
# Deprecated group/name - [DEFAULT]/logfile
#log_file = <None>

# (Optional) The base directory used for relative log_file  paths. This option
# is ignored if log_config_append is set. (string value)
# Deprecated group/name - [DEFAULT]/logdir
#log_dir = <None>

# Uses logging handler designed to watch file system. When log file is moved or
# removed this handler will open a new log file with specified path
# instantaneously. It makes sense only if log_file option is specified and
# Linux platform is used. This option is ignored if log_config_append is set.
# (boolean value)
#watch_log_file = false

# Use syslog for logging. Existing syslog format is DEPRECATED and will be
# changed later to honor RFC5424. This option is ignored if log_config_append
# is set. (boolean value)
#use_syslog = false

# Enable journald for logging. If running in a systemd environment you may wish
# to enable journal support. Doing so will use the journal native protocol
# which includes structured metadata in addition to log messages.This option is
# ignored if log_config_append is set. (boolean value)
#use_journal = false

# Syslog facility to receive log lines. This option is ignored if
# log_config_append is set. (string value)
#syslog_log_facility = LOG_USER

# Use JSON formatting for logging. This option is ignored if log_config_append
# is set. (boolean value)
#use_json = false

# Log output to standard error. This option is ignored if log_config_append is
# set. (boolean value)
#use_stderr = false

# Log output to Windows Event Log. (boolean value)
#use_eventlog = false

# The amount of time before the log files are rotated. This option is ignored
# unless log_rotation_type is setto "interval". (integer value)
#log_rotate_interval = 1

# Rotation interval type. The time of the last file change (or the time when
# the service was started) is used when scheduling the next rotation. (string
# value)
# Possible values:
# Seconds - <No description provided>
# Minutes - <No description provided>
# Hours - <No description provided>
# Days - <No description provided>
# Weekday - <No description provided>
# Midnight - <No description provided>
#log_rotate_interval_type = days
```

(continues on next page)

(continued from previous page)

```
# Maximum number of rotated log files. (integer value)
#max_logfile_count = 30

# Log file maximum size in MB. This option is ignored if "log_rotation_type" is
# not set to "size". (integer value)
#max_logfile_size_mb = 200

# Log rotation type. (string value)
# Possible values:
# interval - Rotate logs at predefined time intervals.
# size - Rotate logs once they reach a predefined size.
# none - Do not rotate log files.
#log_rotation_type = none

# Format string to use for log messages with context. Used by
# oslo_log.formatters.ContextFormatter (string value)
#logging_context_format_string = %(asctime)s.%(msecs)03d %(process)d %(levelname)s
↪%(name)s [%(request_id)s %(user_identity)s] %(instance)s%(message)s

# Format string to use for log messages when context is undefined. Used by
# oslo_log.formatters.ContextFormatter (string value)
#logging_default_format_string = %(asctime)s.%(msecs)03d %(process)d %(levelname)s
↪%(name)s [-] %(instance)s%(message)s

# Additional data to append to log message when logging level for the message
# is DEBUG. Used by oslo_log.formatters.ContextFormatter (string value)
#logging_debug_format_suffix = %(funcName)s %(pathname)s:%(lineno)d

# Prefix each line of exception output with this format. Used by
# oslo_log.formatters.ContextFormatter (string value)
#logging_exception_prefix = %(asctime)s.%(msecs)03d %(process)d ERROR %(name)s
↪%(instance)s

# Defines the format string for %(user_identity)s that is used in
# logging_context_format_string. Used by oslo_log.formatters.ContextFormatter
# (string value)
#logging_user_identity_format = %(user)s %(tenant)s %(domain)s %(user_domain)s
↪%(project_domain)s

# List of package logging levels in logger=LEVEL pairs. This option is ignored
# if log_config_append is set. (list value)
#default_log_levels = amqp=WARN,amqplib=WARN,boto=WARN,qpid=WARN,sqlalchemy=WARN,
↪suds=INFO,oslo.messaging=INFO,oslo_messaging=INFO,iso8601=WARN,requests.packages.
↪urllib3.connectionpool=WARN,urllib3.connectionpool=WARN,websocket=WARN,requests.
↪packages.urllib3.util.retry=WARN,urllib3.util.retry=WARN,keystonemiddleware=WARN,
↪routes.middleware=WARN,stevedore=WARN,taskflow=WARN,keystoneauth=WARN,oslo.
↪cache=INFO,oslo_policy=INFO,dogpile.core.dogpile=INFO

# Enables or disables publication of error events. (boolean value)
#publish_errors = false

# The format for an instance that is passed with the log message. (string
# value)
#instance_format = "[instance: %(uuid)s] "

# The format for an instance UUID that is passed with the log message. (string
```

(continues on next page)

(continued from previous page)

```
# value)
#instance_uuid_format = "[instance: %(uuid)s] "

# Interval, number of seconds, of log rate limiting. (integer value)
#rate_limit_interval = 0

# Maximum number of logged messages per rate_limit_interval. (integer value)
#rate_limit_burst = 0

# Log level name used by rate limiting: CRITICAL, ERROR, INFO, WARNING, DEBUG
# or empty string. Logs with level greater or equal to rate_limit_except_level
# are not filtered. An empty string means that all levels are filtered. (string
# value)
#rate_limit_except_level = CRITICAL

# Enables or disables fatal status of deprecations. (boolean value)
#fatal_deprecations = false


[cors]

#
# From oslo.middleware
#

# Indicate whether this resource may be shared with the domain received in the
# requests "origin" header. Format: "<protocol>://<host>[:<port>]", no trailing
# slash. Example: https://horizon.example.com (list value)
#allowed_origin = <None>

# Indicate that the actual request can include user credentials (boolean value)
#allow_credentials = true

# Indicate which headers are safe to expose to the API. Defaults to HTTP Simple
# Headers. (list value)
#expose_headers =

# Maximum cache age of CORS preflight requests. (integer value)
#max_age = 3600

# Indicate which methods can be used during the actual request. (list value)
#allow_methods = OPTIONS,GET,HEAD,POST,PUT,DELETE,TRACE,PATCH

# Indicate which header field names may be used during the actual request.
# (list value)
#allow_headers =


[healthcheck]

#
# From oslo.middleware
#

# DEPRECATED: The path to respond to healtcheck requests on. (string value)
# This option is deprecated for removal.
# Its value may be silently ignored in the future.
```

(continues on next page)

(continued from previous page)

```
#path = /healthcheck

# Show more detailed information as part of the response. Security note:
# Enabling this option may expose sensitive details about the service being
# monitored. Be sure to verify that it will not violate your security policies.
# (boolean value)
#detailed = false

# Additional backends that can perform health checks and report that
# information back as part of a request. (list value)
#backends =

# Check the presence of a file to determine if an application is running on a
# port. Used by DisableByFileHealthcheck plugin. (string value)
#disable_by_file_path = <None>

# Check the presence of a file based on a port to determine if an application
# is running on a port. Expects a "port:path" list of strings. Used by
# DisableByFilesPortsHealthcheck plugin. (list value)
#disable_by_file_paths =


[keystone_authtoken]

#
# From keystonemiddleware.auth_token
#

# Complete "public" Identity API endpoint. This endpoint should not be an
# "admin" endpoint, as it should be accessible by all end users.
# Unauthenticated clients are redirected to this endpoint to authenticate.
# Although this endpoint should  ideally be unversioned, client support in the
# wild varies.  If you're using a versioned v2 endpoint here, then this  should
# *not* be the same endpoint the service user utilizes  for validating tokens,
# because normal end users may not be  able to reach that endpoint. (string
# value)
#auth_uri = <None>

# API version of the admin Identity API endpoint. (string value)
#auth_version = <None>

# Do not handle authorization requests within the middleware, but delegate the
# authorization decision to downstream WSGI components. (boolean value)
#delay_auth_decision = false

# Request timeout value for communicating with Identity API server. (integer
# value)
#http_connect_timeout = <None>

# How many times are we trying to reconnect when communicating with Identity
# API Server. (integer value)
#http_request_max_retries = 3

# Request environment key where the Swift cache object is stored. When
# auth_token middleware is deployed with a Swift cache, use this option to have
# the middleware share a caching backend with swift. Otherwise, use the
# ``memcached_servers`` option instead. (string value)
```

(continues on next page)

(continued from previous page)

```
#cache = <None>

# Required if identity server requires client certificate (string value)
#certfile = <None>

# Required if identity server requires client certificate (string value)
#keyfile = <None>

# A PEM encoded Certificate Authority to use when verifying HTTPs connections.
# Defaults to system CAs. (string value)
#cafile = <None>

# Verify HTTPS connections. (boolean value)
#insecure = false

# The region in which the identity server can be found. (string value)
#region_name = <None>

# Directory used to cache files related to PKI tokens. (string value)
#signing_dir = <None>

# Optionally specify a list of memcached server(s) to use for caching. If left
# undefined, tokens will instead be cached in-process. (list value)
# Deprecated group/name - [keystone_authtoken]/memcache_servers
#memcached_servers = <None>

# In order to prevent excessive effort spent validating tokens, the middleware
# caches previously-seen tokens for a configurable duration (in seconds). Set
# to -1 to disable caching completely. (integer value)
#token_cache_time = 300

# Determines the frequency at which the list of revoked tokens is retrieved
# from the Identity service (in seconds). A high number of revocation events
# combined with a low cache duration may significantly reduce performance. Only
# valid for PKI tokens. (integer value)
#revocation_cache_time = 10

# (Optional) If defined, indicate whether token data should be authenticated or
# authenticated and encrypted. If MAC, token data is authenticated (with HMAC)
# in the cache. If ENCRYPT, token data is encrypted and authenticated in the
# cache. If the value is not one of these options or empty, auth_token will
# raise an exception on initialization. (string value)
# Possible values:
# None - <No description provided>
# MAC - <No description provided>
# ENCRYPT - <No description provided>
#memcache_security_strategy = None

# (Optional, mandatory if memcache_security_strategy is defined) This string is
# used for key derivation. (string value)
#memcache_secret_key = <None>

# (Optional) Number of seconds memcached server is considered dead before it is
# tried again. (integer value)
#memcache_pool_dead_retry = 300

# (Optional) Maximum total number of open connections to every memcached
```

(continues on next page)

(continued from previous page)

```
# server. (integer value)
#memcache_pool_maxsize = 10

# (Optional) Socket timeout in seconds for communicating with a memcached
# server. (integer value)
#memcache_pool_socket_timeout = 3

# (Optional) Number of seconds a connection to memcached is held unused in the
# pool before it is closed. (integer value)
#memcache_pool_unused_timeout = 60

# (Optional) Number of seconds that an operation will wait to get a memcached
# client connection from the pool. (integer value)
#memcache_pool_conn_get_timeout = 10

# (Optional) Use the advanced (eventlet safe) memcached client pool. The
# advanced pool will only work under python 2.x. (boolean value)
#memcache_use_advanced_pool = false

# (Optional) Indicate whether to set the X-Service-Catalog header. If False,
# middleware will not ask for service catalog on token validation and will not
# set the X-Service-Catalog header. (boolean value)
#include_service_catalog = true

# Used to control the use and type of token binding. Can be set to: "disabled"
# to not check token binding. "permissive" (default) to validate binding
# information if the bind type is of a form known to the server and ignore it
# if not. "strict" like "permissive" but if the bind type is unknown the token
# will be rejected. "required" any form of token binding is needed to be
# allowed. Finally the name of a binding method that must be present in tokens.
# (string value)
#enforce_token_bind = permissive

# If true, the revocation list will be checked for cached tokens. This requires
# that PKI tokens are configured on the identity server. (boolean value)
#check_revocations_for_cached = false

# Hash algorithms to use for hashing PKI tokens. This may be a single algorithm
# or multiple. The algorithms are those supported by Python standard
# hashlib.new(). The hashes will be tried in the order given, so put the
# preferred one first for performance. The result of the first hash will be
# stored in the cache. This will typically be set to multiple values only while
# migrating from a less secure algorithm to a more secure one. Once all the old
# tokens are expired this option should be set to a single value for better
# performance. (list value)
#hash_algorithms = md5

# Authentication type to load (string value)
# Deprecated group/name - [keystone_authtoken]/auth_plugin
#auth_type = <None>

# Config Section from which to load plugin specific options (string value)
#auth_section = <None>


[oslo_middleware]
```

(continues on next page)

(continued from previous page)

```
#
# From oslo.middleware
#

# The maximum body size for each  request, in bytes. (integer value)
# Deprecated group/name - [DEFAULT]/osapi_max_request_body_size
# Deprecated group/name - [DEFAULT]/max_request_body_size
#max_request_body_size = 114688

# DEPRECATED: The HTTP Header that will be used to determine what the original
# request protocol scheme was, even if it was hidden by a SSL termination
# proxy. (string value)
# This option is deprecated for removal.
# Its value may be silently ignored in the future.
#secure_proxy_ssl_header = X-Forwarded-Proto

# Whether the application is behind a proxy or not. This determines if the
# middleware should parse the headers or not. (boolean value)
#enable_proxy_headers_parsing = false


[oslo_policy]

#
# From oslo.policy
#

# This option controls whether or not to enforce scope when evaluating
# policies. If ``True``, the scope of the token used in the request is compared
# to the ``scope_types`` of the policy being enforced. If the scopes do not
# match, an ``InvalidScope`` exception will be raised. If ``False``, a message
# will be logged informing operators that policies are being invoked with
# mismatching scope. (boolean value)
#enforce_scope = false

# The relative or absolute path of a file that maps roles to permissions for a
# given service. Relative paths must be specified in relation to the
# configuration file setting this option. (string value)
#policy_file = policy.json

# Default rule. Enforced when a requested rule is not found. (string value)
#policy_default_rule = default

# Directories where policy configuration files are stored. They can be relative
# to any directory in the search path defined by the config_dir option, or
# absolute paths. The file defined by policy_file must exist for these
# directories to be searched.  Missing or empty directories are ignored. (multi
# valued)
#policy_dirs = policy.d

# Content Type to send and receive data for REST based policy check (string
# value)
# Possible values:
# application/x-www-form-urlencoded - <No description provided>
# application/json - <No description provided>
#remote_content_type = application/x-www-form-urlencoded
```

(continues on next page)

(continued from previous page)

```
# server identity verification for REST based policy check (boolean value)
#remote_ssl_verify_server_crt = false

# Absolute path to ca cert file for REST based policy check (string value)
#remote_ssl_ca_crt_file = <None>

# Absolute path to client cert for REST based policy check (string value)
#remote_ssl_client_crt_file = <None>

# Absolute path client key file REST based policy check (string value)
#remote_ssl_client_key_file = <None>
```

3.9 Sample Policy File

The following is a sample Armada policy file for adaptation and use. It is auto-generated from Armada when this documentation is built, so if you are having issues with an option, please compare your version of Armada with the version of this documentation.

The sample policy file can also be viewed in file form.

CHAPTER 4

Commands Guide

4.1 Armada - Apply

4.1.1 Commands

```
Usage: armada apply [OPTIONS] [LOCATIONS]...

  This command installs and updates charts defined in Armada manifest.

  The apply argument must be relative path to Armada Manifest. Executing
  apply command once will install all charts defined in manifest. Re-
  executing apply command will execute upgrade.

  To see how to create an Armada manifest:     https://airship-
  armada.readthedocs.io/en/latest/operations/

  To install or upgrade charts, run:

          $ armada apply examples/simple.yaml

  To override a specific value in a Manifest, run:

          $ armada apply examples/simple.yaml --set manifest:simple-armada:release=
↪"wordpress"

  Or to override several values in a Manifest, reference a values.yaml-
  formatted file:

          $ armada apply examples/simple.yaml --values examples/simple-ovr-values.yaml

Options:
  --api                         Contacts service endpoint.
  --disable-update-post         Disable post-update Tiller operations.
```

(continues on next page)

(continued from previous page)

```
  --disable-update-pre          Disable pre-update Tiller operations.
  --enable-chart-cleanup        Clean up unmanaged charts.
  --metrics-output TEXT         The output path for metric data
  --use-doc-ref                 Use armada manifest file reference.
  --set TEXT                    Use to override Armada Manifest values.
                                Accepts overrides that adhere to the format
                                <path>:<to>:<property>=<value> to specify a
                                primitive or
                                <path>:<to>:<property>=<value1>,...,<valueN>
                                to specify a list of values.
  --tiller-host TEXT            Tiller host IP.
  --tiller-port INTEGER         Tiller host port.
  -tn, --tiller-namespace TEXT  Tiller namespace.
  --timeout INTEGER             Specifies time to wait for each chart to fully
                                finish deploying.
  -f, --values TEXT             Use to override multiple Armada Manifest
                                values by reading overrides from a
                                values.yaml-type file.
  --wait                        Force Tiller to wait until all charts are
                                deployed, rather than using each charts
                                specified wait policy. This is equivalent to
                                sequenced chartgroups.
  --target-manifest TEXT        The target manifest to run. Required for
                                specifying which manifest to run when multiple
                                are available.
  --bearer-token TEXT           User Bearer token
  --debug                       Enable debug logging.
  --help                        Show this message and exit.
```

4.1.2 Synopsis

The apply command will consume an armada manifest which contains group of charts that it will deploy into the tiller service in your Kubernetes cluster. Executing the `armada apply` again on existing armada deployment will start an update of the armada deployed charts.

```
armada apply armada-manifest.yaml [--debug]
```

If you remove `armada/Charts/v1` from the `armada/ChartGroups/v1` in the armada manifest and execute an `armada apply` with the `--enable-chart-cleanup` flag. Armada will remove undefined releases with the armada manifest's `release_prefix` keyword.

4.2 Armada - Rollback

4.2.1 Commands

```
Usage: armada rollback [OPTIONS]

This command performs a rollback on the specified release.

To rollback a release, run:

    $ armada rollback --release my_release
```

(continues on next page)

(continued from previous page)

```
Options:
  --release TEXT                Release to rollback.
  --tiller-host TEXT            Tiller Host IP
  --tiller-port INTEGER         Tiller Host Port
  -tn, --tiller-namespace TEXT  Tiller Namespace
  --timeout INTEGER             Tiller Host IP
  --version INTEGER             Version of release to rollback to. 0 represents the␣
↪previous release
  --wait                        Version of release to rollback to. 0 represents the␣
↪previous release
  --bearer-token                User bearer token
  --help                        Show this message and exit.
```

4.2.2 Synopsis

The rollback command will perform helm rollback on the release.

4.3 Armada - Test

4.3.1 Commands

```
Usage: armada test [OPTIONS]

  This command test deployed charts

  The tiller command uses flags to obtain information from tiller services.
  The test command will run the release chart tests either via a
  manifest or by targeting a release.

  To obtain armada deployed releases:

      $ armada test --file examples/simple.yaml

  To test release:

      $ armada test --release blog-1

Options:
  --cleanup                     Delete test pods after test completion
  --enable-all                  Run disabled chart tests
  --file TEXT                   armada manifest
  --release TEXT                helm release
  --tiller-host TEXT            Tiller Host IP
  --tiller-port INTEGER         Tiller Host Port
  -tn, --tiller-namespace TEXT  Tiller Namespace
  --target-manifest TEXT        The target manifest to run. Required for
                                specifying which manifest to run when multiple
                                are available.
  --help                        Show this message and exit.
```

4.3.2 Synopsis

The test command will perform helm test defined on the release. Test command can test a single release or a manifest.

4.4 Armada - Tiller

4.4.1 Commands

```
Usage: armada tiller [OPTIONS]

  This command gets tiller information

  The tiller command uses flags to obtain information from tiller services

  To obtain armada deployed releases:

      $ armada tiller --releases

  To obtain tiller service status/information:

      $ armada tiller --status

Options:
  --tiller-host TEXT            Tiller host ip
  --tiller-port INTEGER         Tiller host port
  -tn, --tiller-namespace TEXT  Tiller namespace
  --releases                    list of deployed releses
  --status                      Status of Armada services
  --bearer-token                User bearer token
  --help                        Show this message and exit.
```

4.4.2 Synopsis

The tiller command will perform command directly with tiller to check if tiller in the cluster is running and the list of releases in tiller cluster.

4.5 Armada - Validate

4.5.1 Commands

```
Usage: armada validate [OPTIONS] FILENAME

  This command validates Armada Manifest

  The validate argument must be a relative path to Armada manifest

      $ armada validate examples/simple.yaml

Options:
  --help  Show this message and exit.
```

4.5.2 Synopsis

The validate command will take in an Armada manifest and will validate if it is correctly defined and consumable.

CHAPTER 5

Indices and tables

- genindex
- modindex
- search