Airship Integration Documentation

Release 0.1.0

Airship Authors

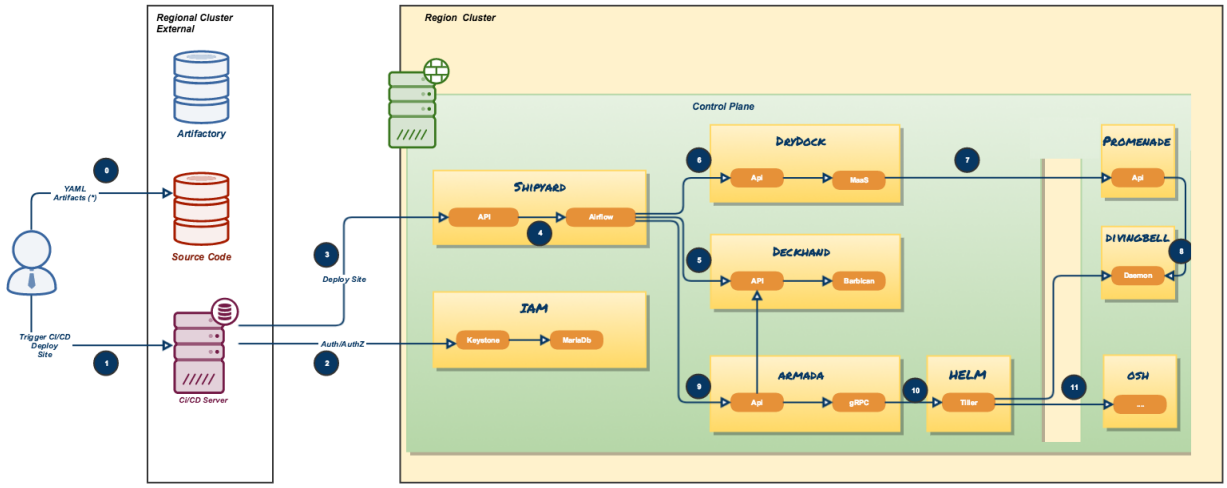
Jan 15, 2019

Contents

Airship is a collection of components that coordinate to form means of configuring and deploying and maintaining a Kubernetes environment using a declarative set of yaml documents.

More specifically, the current focus of this project is the implementation of OpenStack on Kubernetes (OOK).

ARCHITECTURE



CHAPTER 1

Component Projects

1.1 Pegleg

Pegleg is a document aggregator that provides early linting and validations via Deckhand, a document management micro-service within Airship.

1.2 Shipyard

Shipyard is the directed acyclic graph controller for Kubernetes and OpenStack control plane life cycle management.

Shipyard provides the entrypoint for the following aspects of the control plane:

1.2.1 Designs and Secrets

Site designs, including the configuration of bare metal host nodes, network design, operating systems, Kubernetes nodes, Armada manifests, Helm charts, and any other descriptors that define the build out of a group of servers enter the Airship via Shipyard. Secrets, such as passwords and certificates, use the same mechanism. The designs and secrets are stored in Airship's Deckhand, providing for version history and secure storage among other document-based conveniences.

1.2.2 Actions

Interaction with the site's control plane is done via invocation of actions in Shipyard. Each action is backed by a workflow implemented as a directed acyclic graph (DAG) that runs using Apache Airflow. Shipyard provides a mechanism to monitor and control the execution of the workflow.

1.3 Drydock

Drydock is a provisioning orchestrator for baremetal servers that translates a YAML-based declaritive site topology into a physical undercloud that can be used for building out a enterprise Kubernetes cluster. It uses plugins to leverage existing provisioning systems to build the servers allowing integration with the provisioning system that best fits the goals and environment of a site.

1.3.1 Capabilities

- Initial IPMI configuration for PXE booting new servers.
- Support for Canonical MAAS provisioning.
- Configuration of complex network topologies including bonding, tagged VLANs and static routes
- Support for running behind a corporate proxy
- Extensible boot action system for placing files and SystemD units on nodes for post-deployment execution
- Supports Keystone-based authentication and authorization

1.4 Deckhand

Deckhand is a document-based configuration storage service built with auditability and validation in mind.

1.4.1 Core Responsibilities

- layering - helps reduce duplication in configuration by applying the notion of inheritance to documents
- substitution - provides separation between secret data and other configuration data for security purposes and reduces data duplication by allowing common data to be defined once and substituted elsewhere dynamically
- revision history - maintains well-defined collections of documents within immutable revisions that are meant to operate together, while providing the ability to rollback to previous revisions
- validation - allows services to implement and register different kinds of validations and report errors
- secret management - leverages existing OpenStack APIs – namely Barbican – to reliably and securely store sensitive data

1.5 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.

1.5.1 Core Responsibilities

- Multiple Chart Deployments and Upgrades driven by Armada Manifests
- Manage multiple chart dependencies using Chart Groups
- Enhancing base Helm functionality

- Supports Keystone-based authentication and authorization

1.6 Kubernetes

Kubernetes is an open source system for managing containerized applications across multiple hosts, providing basic mechanisms for deployment, maintenance, and scaling of applications.

1.7 Promenade

Promenade is a tool for bootstrapping a resilient, self-hosted Kubernetes cluster and managing its life-cycle.

Bootstrapping begins by provisioning a single-node cluster with a complete, configurable Airship infrastructure. After hosts are added to the cluster, the original bootstrapping node can be re-provisioned to avoid subtle differences that could result in future issues.

Promenade provides cluster resiliency against both node failures and full cluster restarts. It does so by leveraging Helm charts to manage core Kubernetes assets directly on each host, to ensure their availability.

1.8 Helm

Helm is a package manager for Kubernetes. It helps you define, install, and upgrade even the most complex Kubernetes applications using Helm charts.

A chart is a collection of files that describe a related set of Kubernetes resources. Helm wraps up each chart's deployment into a concrete release, a tidy little box that is a collection of all the Kubernetes resources that compose that service, and so you can interact with a collection of Kubernetes resources that compose a release as a single unit, either to install, upgrade, or remove.

At its core, the value that Helm brings to the table – at least for us – is allowing us to templatize our experience with Kubernetes resources, providing a standard interface for operators or high-level software orchestrators to control the installation and life cycle of Kubernetes applications.

1.9 OpenStack-Helm

The OpenStack-Helm project provides a framework to enable the deployment, maintenance, and upgrading of loosely coupled OpenStack services and their dependencies individually or as part of complex environments.

OpenStack-Helm is essentially a marriage of Kubernetes, Helm, and OpenStack, and seeks to create Helm charts for each OpenStack service. These Helm charts provide complete life cycle management for these OpenStack services.

Users of OpenStack-Helm either deploy all or individual OpenStack components along with their required dependencies. It heavily borrows concepts from Stackanetes and complex Helm application deployments. Ideally, at the end of the day, this project is meant to be a collaborative project that brings OpenStack applications into a cloud-native model.

1.10 Divingbell

Divingbell is a lightweight solution for:

1. Bare metal configuration management for a few very targeted use cases
2. Bare metal package manager orchestration

1.10.1 What problems does it solve?

The needs identified for Divingbell were:

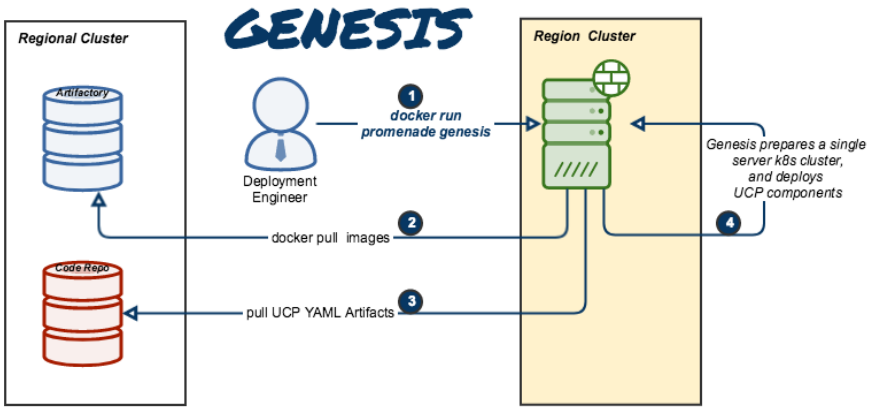
1. To plug gaps in day 1 tools (e.g., Drydock) for node configuration
2. To provide a day 2 solution for managing these configurations going forward
3. [Future] To provide a day 2 solution for system level host patching

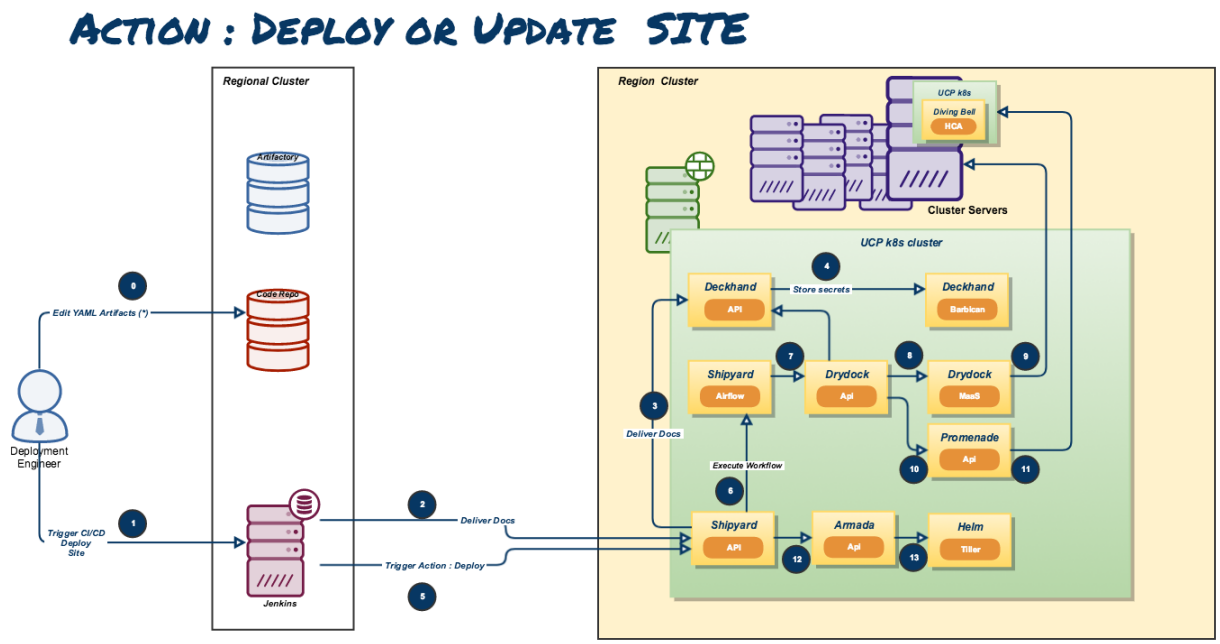
1.11 Berth

Berth is a deliberately minimalist VM runner for Kubernetes.

CHAPTER 2

Process Flows





2.1 Site Authoring and Deployment Guide

The document contains the instructions for standing up a greenfield Airship site. This can be broken down into two high-level pieces:

1. **Site authoring guide(s)**: Describes how to craft site manifests and configs required to perform a deployment. The primary site authoring guide is for deploying Airship sites, where OpenStack is the target platform deployed on top of Airship.
2. **Deployment guide(s)**: Describes how to apply site manifests for a given site.

This document is an "all in one" site authoring guide + deployment guide for a standard Airship deployment. For the most part, the site authoring guidance lives within `airship-seaworthy` reference site in the form of YAML comments.

2.1.1 Terminology

Cloud: A platform that provides a standard set of interfaces for IaaS consumers.

OSH: (OpenStack Helm) is a collection of Helm charts used to deploy OpenStack on kubernetes.

Undercloud/Overcloud: Terms used to distinguish which cloud is deployed on top of the other. In Airship sites, OpenStack (overcloud) is deployed on top of Kubernetes (undercloud).

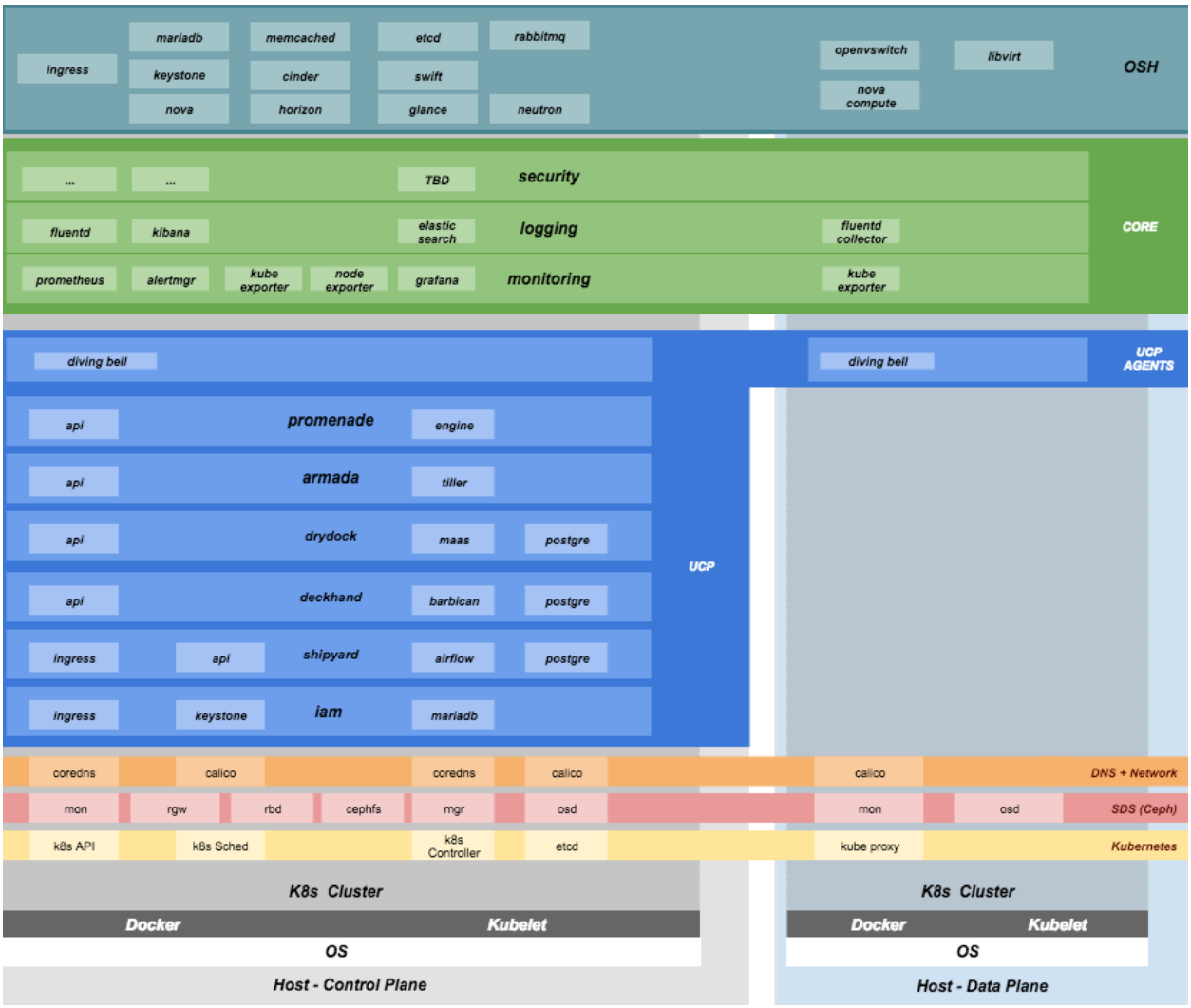
Airship: A specific implementation of OpenStack Helm charts onto kubernetes, the deployment of which is the primary focus of this document.

Control Plane: From the point of view of the cloud service provider, the control plane refers to the set of resources (hardware, network, storage, etc) sourced to run cloud services.

Data Plane: From the point of view of the cloud service provider, the data plane is the set of resources (hardware, network, storage, etc.) sourced to run consumer workloads. When used in this document, "data plane" refers to the data plane of the overcloud (OSH).

Host Profile: A host profile is a standard way of configuring a bare metal host. Encompasses items such as the number of bonds, bond slaves, physical storage mapping and partitioning, and kernel parameters.

Component Overview



Node Overview

This document refers to several types of nodes, which vary in their purpose, and to some degree in their orchestration / setup:

- **Build node**: This refers to the environment where configuration documents are built for your environment (e.g., your laptop)
- **Genesis node**: The "genesis" or "seed node" refers to a node used to get a new deployment off the ground, and is the first node built in a new deployment environment.
- **Control / Controller nodes**: The nodes that make up the control plane. (Note that the Genesis node will be one of the controller nodes.)
- **Compute nodes / Worker Nodes**: The nodes that make up the data plane

2.1.2 Support

Bugs may be viewed and reported at the following locations, depending on the component:

- OpenStack Helm: OpenStack Storyboard group
- Airship: Bugs may be filed using OpenStack Storyboard for specific projects in Airship group:
  - Airship Armada
  - Airship Berth
  - Airship Deckhand
  - Airship Divingbell
  - Airship Drydock
  - Airship MaaS
  - Airship Pegleg
  - Airship Promenade
  - Airship Shipyard
  - Airship in a Bottle
  - Airship Treasuremap

2.1.3 Hardware Prep

Disk

1. For servers that are in the control plane (including Genesis):
   - Two-disk RAID-1: Operating System
   - Two disks JBOD: Ceph Journal/Meta
   - Remaining disks JBOD: Ceph OSD
2. For servers that are in the tenant data plane (compute nodes):
   - Two-disk RAID-1: Operating System
   - Two disks JBOD: Ceph Journal/Meta
   - Two disks JBOD: Ceph OSD
   - Remaining disks need to be configured according to the host profile target for each given server (e.g. RAID-10 for OpenStack Ephemeral).

BIOS and IPMI

1. Virtualization enabled in BIOS
2. IPMI enabled in server BIOS (e.g., IPMI over LAN option enabled)
3. IPMI IPs assigned, and routed to the environment you will deploy into Note: Firmware bugs related to IPMI are common. Ensure you are running the latest firmware version for your hardware. Otherwise, it is recommended to perform an iLo/iDrac reset, as IPMI bugs with long-running firmware are not uncommon.
4. Set PXE as first boot device and ensure the correct NIC is selected for PXE

Network

1. You have a network you can successfully PXE boot with your network topology and bonding settings (dedicated PXE interace on untagged/native VLAN in this example)
2. You have (VLAN) segmented, routed networks accessible by all nodes for:
   (a) Management network(s) (k8s control channel)
   (b) Calico network(s)
   (c) Storage network(s)
   (d) Overlay network(s)
   (e) Public network(s)

2.1.4 HW Sizing and minimum requirements

Node	disk	memory	cpu
Build	10 GB	4 GB	1
Genesis	100 GB	16 GB	8
Control	10 TB	128 GB	24
Compute	N/A*	N/A*	N/A*

- Workload driven (determined by host profile)

2.1.5 Establishing build node environment

1. On the machine you wish to use to generate deployment files, install required tooling:

```
sudo apt -y install docker.io git
```

2. Clone and link the required git repos as follows:

```
git clone https://git.openstack.org/openstack/airship-pegleg
git clone https://git.openstack.org/openstack/airship-treasuremap
```

2.1.6 Building Site documents

This section goes over how to put together site documents according to your specific environment, and generate the initial Promenade bundle needed to start the site deployment.

Preparing deployment documents

In its current form, pegleg provides an organized structure for YAML elements, in order to separate common site elements (i.e., `global` folder) from unique site elements (i.e., `site` folder).

To gain a full understanding of the pegleg structure, it is highly recommended to read pegleg documentation on this here.

The `airship-seaworthy` site may be used as reference site. It is the principal pipeline for integration and continuous deployment testing of Airship.

Change directory to the `airship-treasuremap/site` folder and copy the `airship-seaworthy` site as follows:

```
NEW_SITE=mySite # replace with the name of your site
cd airship-treasuremap/site
cp -r airship-seaworthy $NEW_SITE
```

Remove `airship-seaworthy` specific certificates.

```
rm -f airship-treasuremap/site/${NEW_SITE}/secrets/certificates/certificates.yaml
```

You will then need to manually make changes to these files. These site manifests are heavily commented to explain parameters, and importantly identify all of the parameters that need to change when authoring a new site.

These areas which must be updated for a new site are flagged with the label `NEWSITE-CHANGEME` in YAML commentary. Search for all instances of `NEWSITE-CHANGEME` in your new site definition, and follow the instructions that accompany the tag in order to make all needed changes to author your new Airship site.

Because some files depend on (or will repeat) information from others, the order in which you should build your site files is as follows:

1. site/$NEW_SITE/networks/physical/networks.yaml
2. site/$NEW_SITE/baremetal/nodes.yaml
3. site/$NEW_SITE/networks/common-addresses.yaml
4. site/$NEW_SITE/pki/pki-catalog.yaml
5. All other site files

Control Plane Ceph Cluster Notes

Environment Ceph parameters for the control plane are located in:

`site/${NEW_SITE}/software/charts/ucp/ceph/ceph.yaml`

Setting highlights:

- data/values/conf/storage/osd[*]/data/location: The block device that will be formatted by the Ceph chart and used as a Ceph OSD disk
- data/values/conf/storage/osd[*]/journal/location: The directory backing the ceph journal used by this OSD. Refer to the journal paradigm below.
- data/values/conf/pool/target/osd: Number of OSD disks on each node

Assumptions:

1. Ceph OSD disks are not configured for any type of RAID (i.e., they are configured as JBOD if connected through a RAID controller). (If RAID controller does not support JBOD, put each disk in its own RAID-0 and enable RAID cache and write-back cache if the RAID controller supports it.)
2. Ceph disk mapping, disk layout, journal and OSD setup is the same across Ceph nodes, with only their role differing. Out of the 4 control plane nodes, we expect to have 3 actively participating in the Ceph quorom, and the remaining 1 node designated as a standby Ceph node which uses a different control plane profile (cp_*-secondary) than the other three (cp_*-primary).
3. If doing a fresh install, disk are unlabeled or not labeled from a previous Ceph install, so that Ceph chart will not fail disk initialization

This document covers two Ceph journal deployment paradigms:

1. Servers with SSD/HDD mix (disregarding operating system disks).
2. Servers with no SSDs (disregarding operating system disks). In other words, exclusively spinning disk HDDs available for Ceph.

If you have an operating system available on the target hardware, you can determine HDD and SSD layout with:

```
lsblk -d -o name,rota
```

where a `rota` (rotational) value of `1` indicates a spinning HDD, and where a value of `0` indicates non-spinning disk (i.e. SSD). (Note - Some SSDs still report a value of `1`, so it is best to go by your server specifications).

In case #1, the SSDs will be used for journals and the HDDs for OSDs.

For OSDs, pass in the whole block device (e.g., `/dev/sdd`), and the Ceph chart will take care of disk partitioning, formatting, mounting, etc.

For journals, divide the number of journal disks as evenly as possible between the OSD disks. We will also use the whole block device, however we cannot pass that block device to the Ceph chart like we can for the OSD disks.

Instead, the journal devices must be already partitioned, formatted, and mounted prior to Ceph chart execution. This should be done by MaaS as part of the Drydock host-profile being used for control plane nodes.

Consider the follow example where:

- /dev/sda is an operating system RAID-1 device (SSDs for OS root)
- /dev/sdb is an operating system RAID-1 device (SSDs for ceph journal)
- /dev/sd[cdef] are HDDs

Then, the data section of this file would look like:

```
data:
  values:
    conf:
      storage:
        osd:
          - data:
              type: block-logical
              location: /dev/sdd
            journal:
              type: directory
              location: /var/lib/openstack-helm/ceph/journal/journal-sdd
          - data:
              type: block-logical
              location: /dev/sde
            journal:
              type: directory
              location: /var/lib/openstack-helm/ceph/journal/journal-sde
          - data:
              type: block-logical
              location: /dev/sdf
            journal:
              type: directory
              location: /var/lib/openstack-helm/ceph/journal/journal-sdf
          - data:
              type: block-logical
              location: /dev/sdg
            journal:
              type: directory
```

(continues on next page)

(continued from previous page)

```
            location: /var/lib/openstack-helm/ceph/journal/journal-sdg
      pool:
        target:
          osd: 4
```

where the following mount is setup by MaaS via Drydock host profile for the control-plane nodes:

```
/dev/sdb is mounted to /var/lib/openstack-helm/ceph/journal
```

In case #2, Ceph best practice is to allocate journal space on all OSD disks. The Ceph chart assumes this partitioning has been done beforehand. Ensure that your control plane host profile is partitioning each disk between the Ceph OSD and Ceph journal, and that it is mounting the journal partitions. (Drydock will drive these disk layouts via MaaS provisioning). Note the mountpoints for the journals and the partition mappings. Consider the following example where:

- /dev/sda is the operating system RAID-1 device
- /dev/sd[bcde] are HDDs

Then, the data section of this file will look similar to the following:

```
data:
  values:
    conf:
      storage:
        osd:
          - data:
              type: block-logical
              location: /dev/sdb2
            journal:
              type: directory
              location: /var/lib/openstack-helm/ceph/journal0/journal-sdb
          - data:
              type: block-logical
              location: /dev/sdc2
            journal:
              type: directory
              location: /var/lib/openstack-helm/ceph/journal1/journal-sdc
          - data:
              type: block-logical
              location: /dev/sdd2
            journal:
              type: directory
              location: /var/lib/openstack-helm/ceph/journal2/journal-sdd
          - data:
              type: block-logical
              location: /dev/sde2
            journal:
              type: directory
              location: /var/lib/openstack-helm/ceph/journal3/journal-sde
      pool:
        target:
          osd: 4
```

where the following mounts are setup by MaaS via Drydock host profile for the control-plane nodes:

```
/dev/sdb1 is mounted to /var/lib/openstack-helm/ceph/journal0
/dev/sdc1 is mounted to /var/lib/openstack-helm/ceph/journal1
/dev/sdd1 is mounted to /var/lib/openstack-helm/ceph/journal2
/dev/sde1 is mounted to /var/lib/openstack-helm/ceph/journal3
```

Update Passphrases

Replace passphrases under `site/${NEW_SITE}/secrets/passphrases/` with random generated ones:

- Passpharses generation `openssl rand -hex 10`
- UUID generation `uuidgen` (e.g. for Ceph filesystem ID)
- Update `secrets/passphrases/ipmi_admin_password.yaml` with IPMI password
- Update `secrets/passphrases/ubuntu_crypt_password.yaml` with password hash:

```
python3 -c "from crypt import *; print(crypt('<YOUR_PASSWORD>', METHOD_SHA512))"
```

Manifest linting and combining layers

After constituent YAML configurations are finalized, use Pegleg to lint your manifests, and resolve any issues that result from linting before proceeding:

```
sudo airship-pegleg/tools/pegleg.sh repo \
  -r airship-treasuremap lint
```

Note: `P001` and `P003` linting errors are expected for missing certificates, as they are not generated until the next section. You may suppress these warnings by appending `-x P001 -x P003` to the lint command.

Next, use pegleg to perform the merge that will yield the combined global + site type + site YAML:

```
sudo sh airship-pegleg/tools/pegleg.sh site \
  -r airship-treasuremap \
  collect $NEW_SITE
```

Perform a visual inspection of the output. If any errors are discovered, you may fix your manifests and re-run the `lint` and `collect` commands.

After you have an error-free output, save the resulting YAML as follows:

```
sudo airship-pegleg/tools/pegleg.sh site \
  -r airship-treasuremap \
  collect $NEW_SITE -s ${NEW_SITE}_collected
```

It is this output which will be used in subsequent steps.

Lastly, you should also perform a `render` on the documents. The resulting render from Pegleg will not be used as input in subsequent steps, but is useful for understanding what the document will look like once Deckhand has performed all substitutions, replacements, etc. This is also useful for troubleshooting, and addressing any Deckhand errors prior to submitting via Shipyard:

```
sudo airship-pegleg/tools/pegleg.sh site \
  -r airship-treasuremap \
  render $NEW_SITE
```

Inspect the rendered document for any errors. If there are errors, address them in your manifests and re-run this section of the document.

Building the Promenade bundle

Clone the Promenade repo, if not already cloned:

```
git clone https://github.com/openstack/airship-promenade
```

Refer to the `data/charts/ucp/promenade/reference` field in `airship-treasuremap/global/software/config/versions.yaml`. If this is a pinned reference (i.e., any reference that's not `master`), then you should checkout the same version of the Promenade repository. For example, if the Promenade reference was `86c3c11...` in the versions file, checkout the same version of the Promenade repo which was cloned previously:

```
(cd airship-promenade && git checkout 86c3c11)
```

Likewise, before running the `simple-deployment.sh` script, you should refer to the `data/images/ucp/promenade/promenade` field in `~/airship-treasuremap/global/software/config/versions.yaml`. If there is a pinned reference (i.e., any image reference that's not `latest`), then this reference should be used to set the `IMAGE_PROMENADE` environment variable. For example, if the Promenade image was pinned to `quay.io/airshipit/promenade:d30397f...` in the versions file, then export the previously mentioned environment variable like so:

```
export IMAGE_PROMENADE=quay.io/airshipit/promenade:d30397f...
```

Now, create an output directory for Promenade bundles and run the `simple-deployment.sh` script:

```
mkdir ${NEW_SITE}_bundle
sudo -E airship-promenade/tools/simple-deployment.sh ${NEW_SITE}_collected ${NEW_SITE}
↪_bundle
```

Estimated runtime: About **1 minute**

After the bundle has been successfully created, copy the generated certificates into the security folder. Ex:

```
mkdir -p airship-treasuremap/site/${NEW_SITE}/secrets/certificates
sudo cp ${NEW_SITE}_bundle/certificates.yaml \
  airship-treasuremap/site/${NEW_SITE}/secrets/certificates/certificates.yaml
```

2.1.7 Genesis node

Initial setup

Before starting, ensure that the BIOS and IPMI settings match those stated previously in this document. Also ensure that the hardware RAID is setup for this node per the control plane disk configuration stated previously in this document.

Then, start with a manual install of Ubuntu 16.04 on the node you wish to use to seed the rest of your environment standard Ubuntu ISO. Ensure to select the following:

- UTC timezone
- Hostname that matches the Genesis hostname given in `/data/genesis/hostname` in `airship-treasuremap/site/${NEW_SITE}/networks/common-addresses.yaml`.

- At the `Partition Disks` screen, select `Manual` so that you can setup the same disk partitioning scheme used on the other control plane nodes that will be deployed by MaaS. Select the first logical device that corresponds to one of the RAID-1 arrays already setup in the hardware controller. On this device, setup partitions matching those defined for the `bootdisk` in your control plane host profile found in `airship-treasuremap/site/${NEW_SITE}/profiles/host`. (e.g., 30G for /, 1G for /boot, 100G for /var/log, and all remaining storage for /var). Note that the volume size syntax looking like `>300g` in Drydock means that all remaining disk space is allocated to this volume, and that volume needs to be at least 300G in size.
- Ensure that OpenSSH and Docker (Docker is needed because of miniMirror) are included as installed packages
- When you get to the prompt, "How do you want to manage upgrades on this system?", choose "No automatic updates" so that packages are only updated at the time of our choosing (e.g. maintenance windows).
- Ensure the grub bootloader is also installed to the same logical device as in the previous step (this should be default behavior).

After installation, ensure the host has outbound internet access and can resolve public DNS entries (e.g., `nslookup google.com`, `curl https://www.google.com`).

Ensure that the deployed Genesis hostname matches the hostname in `data/genesis/hostname` in `airship-treasuremap/site/${NEW_SITE}/networks/common-addresses.yaml`. If it does not match, then either change the hostname of the node to match the configuration documents, or re-generate the configuration with the correct hostname. In order to change the hostname of the deployed node, you may run the following:

```
sudo hostname $NEW_HOSTNAME
sudo sh -c "echo $NEW_HOSTNAME > /etc/hostname"
sudo vi /etc/hosts # Anywhere the old hostname appears in the file, replace
                   # with the new hostname
```

Or to regenerate manifests, re-run the previous two sections with the after updating the genesis hostname in the site definition.

Installing matching kernel version

Install the same kernel version on the Genesis host that MaaS will use to deploy new baremetal nodes.

In order to do this, first you must determine the kernel version that will be deployed to those nodes. Start by looking at the host profile definition used to deploy other control plane nodes by searching for `control-plane: enabled`. Most likely this will be a file under `global/profiles/host`. In this file, find the kernel info - e.g.:

```
platform:
    image: 'xenial'
    kernel: 'hwe-16.04'
```

In this case, it is the hardware enablement kernel for 16.04. To find the exact kernel version that will be deployed, we must look into the simple-stream image cache that will be used by MaaS to deploy nodes with. Locate the `data/images/ucp/maas/maas_cache` key in within `airship-treasuremap/global/software/config/versions.yaml`. This is the image that you will need to fetch, using a node with docker installed that has access and can reach the site/location hosting the image. For example, from the **build node**, the command would take the form:

```
sudo docker pull YOUR_SSTREAM_IMAGE
```

Then, create a container from that image:

```
cd ~
sudo sh -c "$(docker images | grep sstream-cache | head -1 | awk '{print $1}' > image_
↪name)"
sudo docker create --name sstream $(cat image_name)
```

Then use the container ID returned from the last command as follows:

```
sudo docker start sstream
sudo docker exec -it sstream /bin/bash
```

In the container, `cd` to the following location (substituting for the platform image and platform kernel identified in the host profile previously, and choosing the folder corresponding to the most current date if more than one are available) and run the `file` command on the `boot-kernel` file:

```
cd /var/www/html/maas/images/ephemeral-v3/daily/PLATFORM_IMAGE/amd64/LATEST_DATE/
↪PLATFORM_KERNEL/generic
file boot-kernel
```

This will produce the complete kernel version. E.g.:

```
Linux kernel x86 boot executable bzImage, version 4.13.0-43-generic (buildd@lcy01-
↪amd64-029) #48~16.04.1-Ubuntu S, RO-rootFS, swap_dev 0x7, Normal VGA
```

In this example, the kernel version is `4.13.0-43-generic`. Define any proxy environment variables needed for your environment to reach public ubuntu package repos, and install the matching kernel on the Genesis host (make sure to run on Genesis host, not on the build host):

```
sudo apt -y install 4.13.0-43-generic
```

Check the installed packages on the genesis host with `dpkg --list`. If there are any later kernel versions installed, remove them with `sudo apt remove`, so that the newly install kernel is the latest available.

Lastly if you wish to cleanup your build node, you may run the following:

```
exit # (to quit the container)
cd ~
sudo docker stop sstream
sudo docker rm sstream
sudo docker image rm $(cat image_name)
sudo rm image_name
```

Install ntpdate/ntp

Install and run ntpdate, to ensure a reasonably sane time on genesis host before proceeding:

```
sudo apt -y install ntpdate
sudo ntpdate ntp.ubuntu.com
```

If your network policy does not allow time sync with external time sources, specify a local NTP server instead of using `ntp.ubuntu.com`.

Then, install the NTP client:

```
sudo apt -y install ntp
```

Add the list of NTP servers specified in `data/ntp/servers_joined` in file `airship-treasuremap/site/${NEW_SITE}/networks/common-address.yaml` to `/etc/ntp.conf` as follows:

```
pool NTP_SERVER1 iburst
pool NTP_SERVER2 iburst
(repeat for each NTP server with correct NTP IP or FQDN)
```

Then, restart the NTP service:

```
sudo service ntp restart
```

If you cannot get good time to your selected time servers, consider using alternate time sources for your deployment.

Disable the apparmor profile for ntpd:

```
sudo ln -s /etc/apparmor.d/usr.sbin.ntpd /etc/apparmor.d/disable/
sudo apparmor_parser -R /etc/apparmor.d/usr.sbin.ntpd
```

This prevents an issue with the MaaS containers, which otherwise get permission denied errors from apparmor when the MaaS container tries to leverage libc6 for /bin/sh when MaaS container ntpd is forcefully disabled.

Setup Ceph Journals

Until genesis node reprovisioning is implemented, it is necessary to manually perform host-level disk partitioning and mounting on the genesis node, for activites that would otherwise have been addressed by a bare metal node provision via Drydock host profile data by MaaS.

Assuming your genesis HW matches the HW used in your control plane host profile, you should manually apply to the genesis node the same Ceph partitioning (OSDs & journals) and formatting + mounting (journals only) as defined in the control plane host profile. See `airship-treasuremap/global/profiles/host/base_control_plane.yaml`.

For example, if we have a journal SSDs `/dev/sdb` on the genesis node, then use the `cfdisk` tool to format it:

```
sudo cfdisk /dev/sdb
```

Then:

1. Select `gpt` label for the disk
2. Select `New` to create a new partition
3. If scenario #1 applies in site/$NEW_SITE/software/charts/ucp/ceph/ceph.yaml_, then accept default partition size (entire disk). If scenario #2 applies, then only allocate as much space as defined in the journal disk partitions mounted in the control plane host profile.
4. Select `Write` option to commit changes, then `Quit`
5. If scenario #2 applies, create a second partition that takes up all of the remaining disk space. This will be used as the OSD partition (`/dev/sdb2`).

Install package to format disks with XFS:

```
sudo apt -y install xfsprogs
```

Then, construct an XFS filesystem on the journal partition with XFS:

```
sudo mkfs.xfs /dev/sdb1
```

Create a directory as mount point for `/dev/sdb1` to match those defined in the same host profile ceph journals:

```
sudo mkdir -p /var/lib/ceph/cp
```

Use the `blkid` command to get the UUID for `/dev/sdb1`, then populate `/etc/fstab` accordingly. Ex:

```
sudo sh -c 'echo "UUID=01234567-ffff-aaaa-bbbb-abcdef012345 /var/lib/ceph/cp xfs␣
↪defaults 0 0" >> /etc/fstab'
```

Repeat all preceeding steps in this section for each journal device in the Ceph cluster. After this is completed for all journals, mount the partitions:

```
sudo mount -a
```

Promenade bootstrap

Copy the `${NEW_SITE}_bundle` and `${NEW_SITE}_collected` directories from the build node to the genesis node, into the home directory of the user there (e.g., `/home/ubuntu`). Then, run the following script as sudo on the genesis node:

```
cd ${NEW_SITE}_bundle
sudo ./genesis.sh
```

Estimated runtime: **40m**

Following completion, run the `validate-genesis.sh` script to ensure correct provisioning of the genesis node:

```
cd ${NEW_SITE}_bundle
sudo ./validate-genesis.sh
```

Estimated runtime: **2m**

2.1.8 Deploy Site with Shipyard

Start by cloning the shipyard repository to the Genesis node:

```
git clone https://github.com/openstack/airship-shipyard
```

Refer to the `data/charts/ucp/shipyard/reference` field in `airship-treasuremap/global/software/config/versions.yaml`. If this is a pinned reference (i.e., any reference that's not `master`), then you should checkout the same version of the Shipyard repository. For example, if the Shipyard reference was `7046ad3...` in the versions file, checkout the same version of the Shipyard repo which was cloned previously:

```
(cd airship-shipyard && git checkout 7046ad3)
```

Likewise, before running the `deckhand_load_yaml.sh` script, you should refer to the `data/images/ucp/shipyard/shipyard` field in `airship-treasuremap/global/software/config/versions.yaml`. If there is a pinned reference (i.e., any image reference that's not `latest`), then this reference should be used to set the `SHIPYARD_IMAGE` environment variable. For example, if the Shipyard image was pinned to `quay.io/airshipit/shipyard@sha256:dfc25e1...` in the versions file, then export the previously mentioned environment variable:

```
export SHIPYARD_IMAGE=quay.io/airshipit/shipyard@sha256:dfc25e1...
```

Export valid login credentials for one of the Airship Keystone users defined for the site. Currently there is no authorization checks in place, so the credentials for any of the site-defined users will work. For example, we can use

the `shipyard` user, with the password that was defined in `airship-treasuremap/site/${NEW_SITE}/secrets/passphrases/ucp_shipyard_keystone_password.yaml`. Ex:

```
export OS_USERNAME=shipyard
export OS_PASSWORD=46a75e4...
```

(Note: Default auth variables are defined here, and should otherwise be correct, barring any customizations of these site parameters).

Next, run the deckhand_load_yaml.sh script as follows:

```
sudo -E airship-shipyard/tools/deckhand_load_yaml.sh ${NEW_SITE} ${NEW_SITE}_collected
```

Estimated runtime: **3m**

Now deploy the site with shipyard:

```
sudo -E airship-shipyard/tools/deploy_site.sh
```

Estimated runtime: **1h30m**

The message `Site Successfully Deployed` is the expected output at the end of a successful deployment. In this example, this means that Airship and OSH should be fully deployed.

2.1.9 Disable password-based login on Genesis

Before proceeding, verify that your SSH access to the Genesis node is working with your SSH key (i.e., not using password-based authentication).

Then, disable password-based SSH authentication on Genesis in `/etc/ssh/sshd_config` by uncommenting the `PasswordAuthentication` and setting its value to `no`. Ex:

```
PasswordAuthentication no
```

Then, restart the ssh service:

```
sudo systemctl restart ssh
```

2.2 Airship Seaworthy

Airship Seaworthy is a multi-node site deployment reference and continuous integration pipeline.

The site manifests are available at site/airship-seaworthy.

2.2.1 Pipeline

Airship Seaworthy pipeline automates deployment flow documented in Site Authoring and Deployment Guide.

The pipeline is implemented as Jenkins Pipeline (Groovy), see code for the pipeline at Jenkinsfile.

2.2.2 Versions

The manifest overrides (versions.yaml) are setup to deploy OpenStack Ocata.

The versions are kept up to date via updater.py, a utility that updates versions.yaml latest charts and (selected) images.

The pipeline attempts to uplift and deploy latest versions on daily bases.

2.2.3 Hardware

While HW configuration is flexible, Airship Seaworthy reference manifests reflect full HA deployment, similar to what might be expected in production.

Reducing number of control/compute nodes will require site overrides to align parts of the system such as Ceph replication, etcd, etc.

Airship Seaworthy site has 6 DELL R720xd bare-metal servers: 3 control, and 3 compute nodes. See host profiles for the servers here.

Control (masters)

- cab23-r720-11
- cab23-r720-12
- cab23-r720-13

Compute (workers)

- cab23-r720-14
- cab23-r720-17
- cab23-r720-19

2.2.4 Network

Physical (underlay) networks are described in Drydock site configuration here. It defines OOB (iLO/IPMI), untagged PXE, and multiple tagged general use networks.

Calico overlay for k8s POD networking uses IPIP mesh.

BGP peering is supported but not enabled in this setup, see Calico chart.

2.3 Airskiff

- Skiff (n): a shallow, flat-bottomed, open boat
- Airskiff (n): a learning development, and gating environment for Airship

2.3.1 What is Airskiff

Airskiff is an easy way to get started with the software delivery components of Airship:

- Armada
- Deckhand
- Pegleg

- Shipyard

Airskiff is packaged with a set of deployment scripts modeled after the OpenStack-Helm project for seamless developer setup.

These scripts:

- Download, build, and containerize the Airship components above from source.
- Deploy a Kubernetes cluster using KubeADM.
- Deploy Armada, Deckhand, and Shipyard using the latest Armada image.
- Deploy OpenStack using the Airskiff site and charts from the OpenStack-Helm project.

Warning: Airskiff is not safe for production use. These scripts are only intended to deploy a minimal development environment.

2.3.2 Common Deployment Requirements

This section covers actions that may be required for some deployment scenarios.

Passwordless sudo

Airskiff relies on scripts that utilize the `sudo` command. Throughout this guide the assumption is that the user is: `ubuntu`. It is advised to add the following lines to `/etc/sudoers`:

```
root    ALL=(ALL) NOPASSWD: ALL
ubuntu  ALL=(ALL) NOPASSWD: ALL
```

Proxy Configuration

Note: This section assumes you have properly defined the standard `http_proxy`, `https_proxy`, and `no_proxy` environment variables and have followed the Docker proxy guide to create a systemd drop-in unit.

In order to deploy Airskiff behind proxy servers, define the following environment variables:

```
export USE_PROXY=true
export PROXY=${http_proxy}
export no_proxy=${no_proxy},172.17.0.1,.svc.cluster.local
export NO_PROXY=${NO_PROXY},172.17.0.1,.svc.cluster.local
```

Note: The `.svc.cluster.local` address is required to allow the OpenStack client to communicate without being routed through proxy servers. The IP address `172.17.0.1` is the advertised IP address for the Kubernetes API server. Replace the addresses if your configuration does not match the one defined above.

2.3.3 Deploy Airskiff

Deploy Airskiff using the deployment scripts contained in the `tools/deployment/airskiff` directory of the airship-treasuremap repository.

Note: Scripts should be run from the root of `airship-treasuremap` repository.

Install required packages

```
#!/bin/bash
set -xe

sudo apt-get update
sudo apt-get install --no-install-recommends -y \
        ca-certificates \
        git \
        make \
        jq \
        nmap \
        curl \
        uuid-runtime \
        apt-transport-https \
        ca-certificates \
        software-properties-common

# Purge Docker and install Docker CE
systemctl unmask docker.service
sudo apt-get remove --no-install-recommends -y docker docker-engine docker.io
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -
sudo apt-add-repository \
        "deb [arch=amd64] https://download.docker.com/linux/ubuntu \
        $(lsb_release -cs) \
        stable"

sudo apt-get update
sudo apt-get install --no-install-recommends -y docker-ce

# Add $USER to docker group
# NOTE: This requires re-authentication. Restart your shell.
sudo adduser "$(whoami)" docker
```

Alternatively, this step can be performed by running the script directly:

```
./tools/deployment/airskiff/developer/000-install-packages.sh
```

Restart your shell session

At this point, restart your shell session to complete adding `$USER` to the `docker` group.

Build Airship components

```
#!/bin/bash
set -xe

# NOTE(drewwalters96): Build Airship components locally, for use as a
#   as a development environment.

CURRENT_DIR="$(pwd)"
: "${INSTALL_PATH:="../"}"

cd ${INSTALL_PATH}
git clone https://git.openstack.org/openstack/airship-armada.git
cd airship-armada
# NOTE(drewwalters96): Temporarily pin Armada to K8s v2.11.0.
git checkout dd75474d78e69f3634eb94c8ea6730b898e83277
make images IMAGE_TAG=dd75474d78e69f3634eb94c8ea6730b898e83277

cd ${INSTALL_PATH}
git clone https://git.openstack.org/openstack/airship-deckhand.git
cd airship-deckhand && make

cd ${INSTALL_PATH}
git clone https://git.openstack.org/openstack/airship-shipyard.git
cd airship-shipyard && make

cd ${INSTALL_PATH}
git clone https://git.openstack.org/openstack/airship-pegleg.git

# Clone dependencies
git clone https://git.openstack.org/openstack/openstack-helm-infra.git

cd openstack-helm-infra
git checkout 8662018a4dceb82a5d699d9e118caff9e5edb156

cd "${CURRENT_DIR}"
```

Alternatively, this step can be performed by running the script directly:

```
./tools/deployment/airskiff/developer/005-make-airship.sh
```

Deploy Kubernetes with KubeADM

```
#!/bin/bash
set -xe

CURRENT_DIR="$(pwd)"
: "${OSH_INFRA_PATH:="../openstack-helm-infra"}"

# Configure proxy settings if $PROXY is set
if [ -n "${PROXY}" ]; then
  . tools/deployment/airskiff/common/setup-proxy.sh
fi

# Deploy a kubeadm-administered cluster.
cd ${OSH_INFRA_PATH}
```

(continues on next page)

(continued from previous page)

```
make dev-deploy setup-host
make dev-deploy k8s
cd "${CURRENT_DIR}"

kubectl label nodes --all --overwrite ucp-control-plane=enabled
```

Alternatively, this step can be performed by running the script directly:

```
./tools/deployment/airskiff/developer/010-deploy-k8s.sh
```

Setup OpenStack Client

```
#!/bin/bash
set -xe

# Install OpenStack client and create OpenStack client configuration file.
sudo -H -E pip install "cmd2<=0.8.7"
sudo -H -E pip install python-openstackclient python-heatclient

sudo -H mkdir -p /etc/openstack
sudo -H chown -R "$(id -un)": /etc/openstack
tee /etc/openstack/clouds.yaml << EOF
clouds:
  airship:
    region_name: RegionOne
    identity_api_version: 3
    auth:
      username: 'admin'
      password: 'password'
      project_name: 'admin'
      project_domain_name: 'default'
      user_domain_name: 'default'
      auth_url: 'http://keystone.ucp.svc.cluster.local/v3'
  openstack:
    region_name: RegionOne
    identity_api_version: 3
    auth:
      username: 'admin'
      password: 'password123'
      project_name: 'admin'
      project_domain_name: 'default'
      user_domain_name: 'default'
      auth_url: 'http://keystone.openstack.svc.cluster.local/v3'
EOF
```

Alternatively, this step can be performed by running the script directly:

```
./tools/deployment/airskiff/developer/020-setup-client.sh
```

Deploy Airship components using Armada

```
#!/bin/bash
set -xe
```

(continues on next page)

(continued from previous page)

```
: "${INSTALL_PATH:="$(pwd)/../"}"

# Download latest Armada image and deploy Airship components
# NOTE(drewwalters96): Temporarily pin Armada to K8s v2.11.0.
docker run --rm --net host -p 8000:8000 --name armada \
    -v ~/.kube/config:/armada/.kube/config \
    -v "$(pwd)"/tools/deployment/airskiff/manifests/:/manifests \
    -v "${INSTALL_PATH}":/airship-components \
    quay.io/airshipit/armada:dd75474d78e69f3634eb94c8ea6730b898e83277 \
    apply /manifests/airship.yaml
```

Alternatively, this step can be performed by running the script directly:

```
./tools/deployment/airskiff/developer/030-armada-bootstrap.sh
```

Deploy OpenStack using Airship

```
#!/bin/bash
set -xe

# Deploy OpenStack using Airship
CURRENT_DIR="$(pwd)"
: "${PL_PATH:="../airship-pegleg"}"
: "${SY_PATH:="../airship-shipyard"}"

# NOTE: Image to use for all Pegleg operations
: "${PL_IMAGE:=quay.io/airshipit/pegleg:latest}"

# Lint deployment documents
: "${PEGLEG:="${PL_PATH}/tools/pegleg.sh"}"
: "${PL_SITE:="airskiff"}"

# NOTE(drewwalters96): Disable Pegleg linting errors P001 and P009; a
#  a cleartext storage policy is acceptable for non-production use cases
#  and maintain consistency with other treasuremap sites.
IMAGE=${PL_IMAGE} TERM_OPTS=" " ${PEGLEG} site -r . lint "${PL_SITE}" -x P001 -x P009

# Collect deployment documents
: "${PL_OUTPUT:="peggles"}"
mkdir -p ${PL_OUTPUT}

IMAGE=${PL_IMAGE} TERM_OPTS="-l info" ${PEGLEG} site -r . collect ${PL_SITE} -s ${PL_
↪OUTPUT}
cp -rp "${CURRENT_DIR}"/${PL_OUTPUT} ${SY_PATH}

# Deploy Airskiff site
cd ${SY_PATH}
: "${SHIPYARD:="./tools/shipyard.sh"}"

${SHIPYARD} create configdocs airskiff-design \
            --replace \
            --directory=/target/${PL_OUTPUT}

${SHIPYARD} commit configdocs
```

(continues on next page)

(continued from previous page)

```
${SHIPYARD} create action update_software --allow-intermediate-commits
cd "${CURRENT_DIR}"
```

Alternatively, this step can be performed by running the script directly:

```
./tools/deployment/airskiff/developer/100-deploy-osh.sh
```

2.3.4 Use Airskiff

The Airskiff deployment scripts install and configure the OpenStack client for usage on your host machine.

Airship Examples

To use Airship services, set the `OS_CLOUD` environment variable to `airship`.

```
export OS_CLOUD=airship
```

List the Airship service endpoints:

```
openstack endpoint list
```

Note: `${SHIPYARD}` is the path to a cloned Shipyard repository.

Run Helm tests for all deployed releases:

```
${SHIPYARD}/tools/shipyard.sh create action test_site
```

List all Shipyard actions:

```
${SHIPYARD}/tools/shipyard.sh get actions
```

For more information about Airship operations, see the Shipyard actions documentation.

OpenStack Examples

To use OpenStack services, set the `OS_CLOUD` environment variable to `openstack`:

```
export OS_CLOUD=openstack
```

List the OpenStack service endpoints:

```
openstack endpoint list
```

List `Glance` images:

```
openstack image list
```

Issue a new `Keystone` token:

```
openstack token issue
```

Note: Airskiff deploys identity, network, cloudformation, placement, compute, orchestration, and image services. You can deploy more services by adding chart groups to `site/airskiff/software/manifests/full-site.yaml`. For more information, refer to the site authoring and deployment guide.

2.3.5 Develop with Airskiff

Once you have successfully deployed a running cluster, changes to Airship and OpenStack components can be deployed using Shipyard actions or the Airskiff deployment scripts.

This example demonstrates deploying Armada changes using the Airskiff deployment scripts.

Note: `${ARMADA}` is the path to your cloned Armada repository that contains the changes you wish to deploy. `${TREASUREMAP}` is the path to your cloned Treasuremap repository.

Build Armada:

```
cd ${ARMADA}
make images
```

Update Airship components:

```
cd ${TREASUREMAP}
./tools/deployment/developer/airskiff/030-armada-bootstrap.sh
```

2.3.6 Troubleshooting

This section is intended to help you through the initial troubleshooting process. If issues persist after following this guide, please join us on IRC: #airshipit (freenode)

`Missing value auth-url required for auth plugin password`

If this error message appears when using the OpenStack client, verify your client is configured for authentication:

```
# For Airship services
export OS_CLOUD=airship

# For OpenStack services
export OS_CLOUD=openstack
```