Airship Integration Documentation

Release 0.1.0

Airship Authors

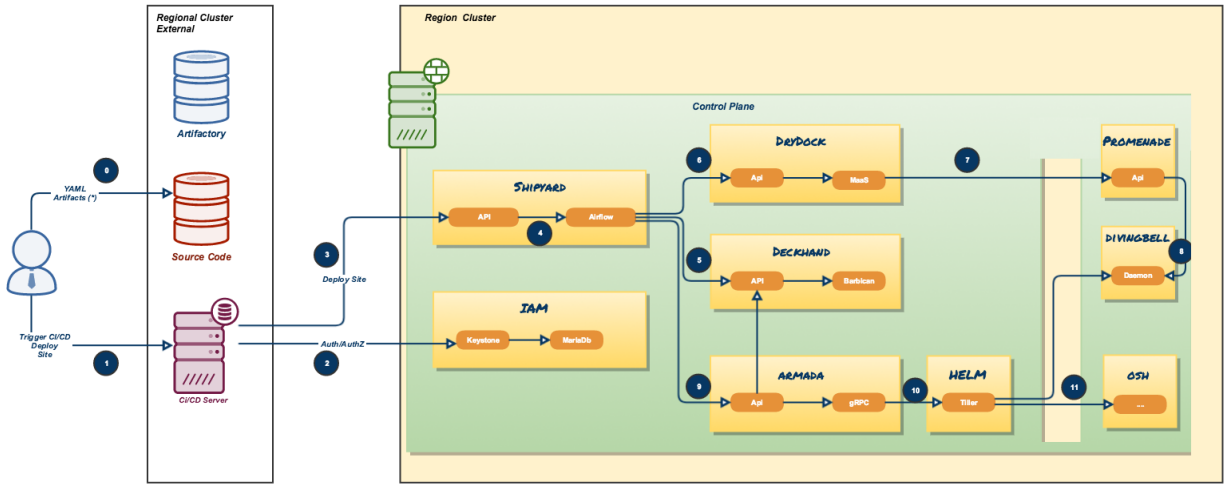
Jan 14, 2020

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:

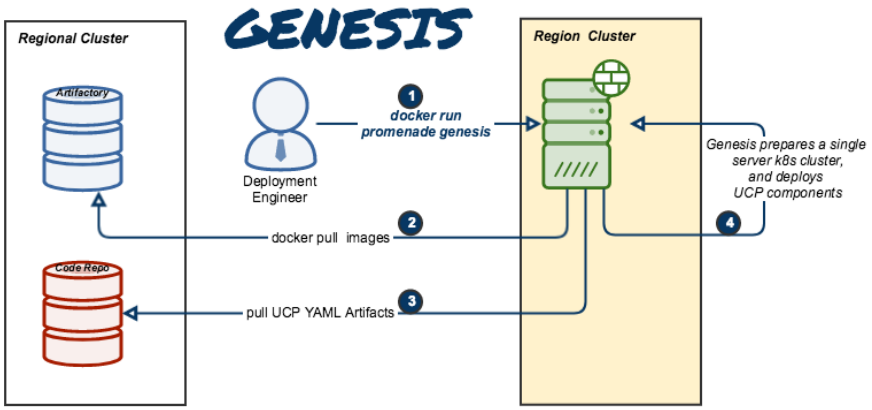
1. To plug gaps in day 1 tools (e.g., Drydock) for node configuration
2. To provide a day 2 solution for managing these configurations going forward
3. [Future] To provide a day 2 solution for system level host patching

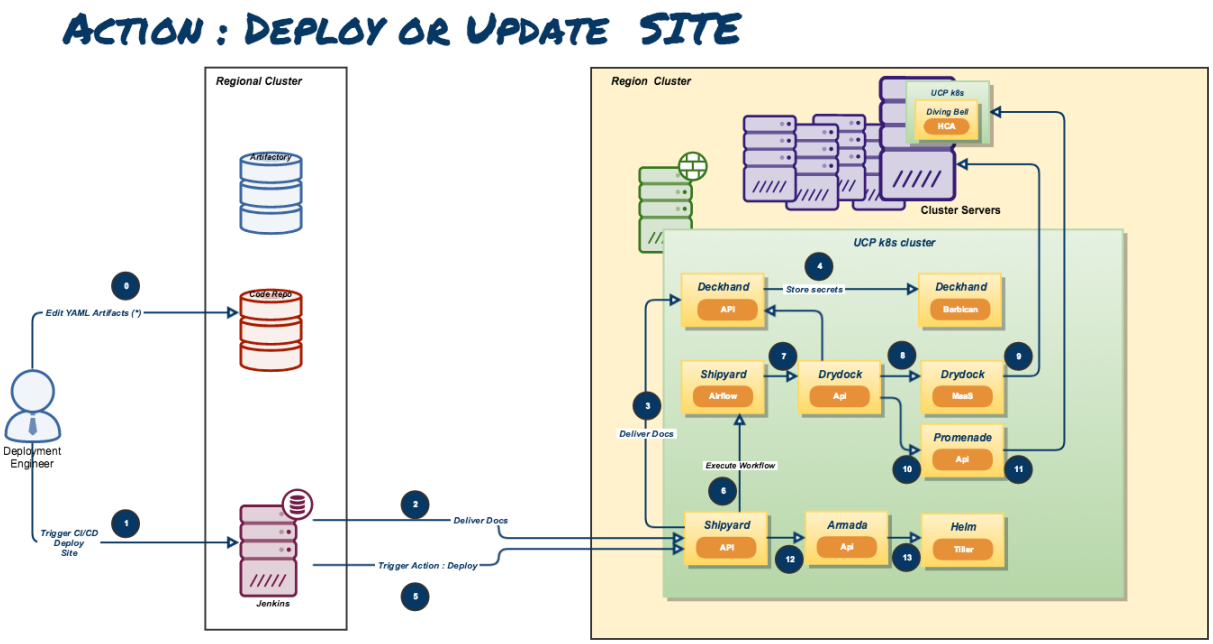
1.11 Berth

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 `seaworthy` reference site in the form of YAML comments.

2.1.1 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 Deckhand
  - Airship Divingbell
  - Airship Drydock
  - Airship MaaS
  - Airship Pegleg
  - Airship Promenade

- Airship Shipyard
- Airship Treasuremap

2.1.2 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.

Helm: (Helm) is a package manager for Kubernetes. Helm Charts help you define, install, and upgrade Kubernetes applications.

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 that deploy Kubernetes. This deployment 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.) configured to provide cloud services for customers.

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

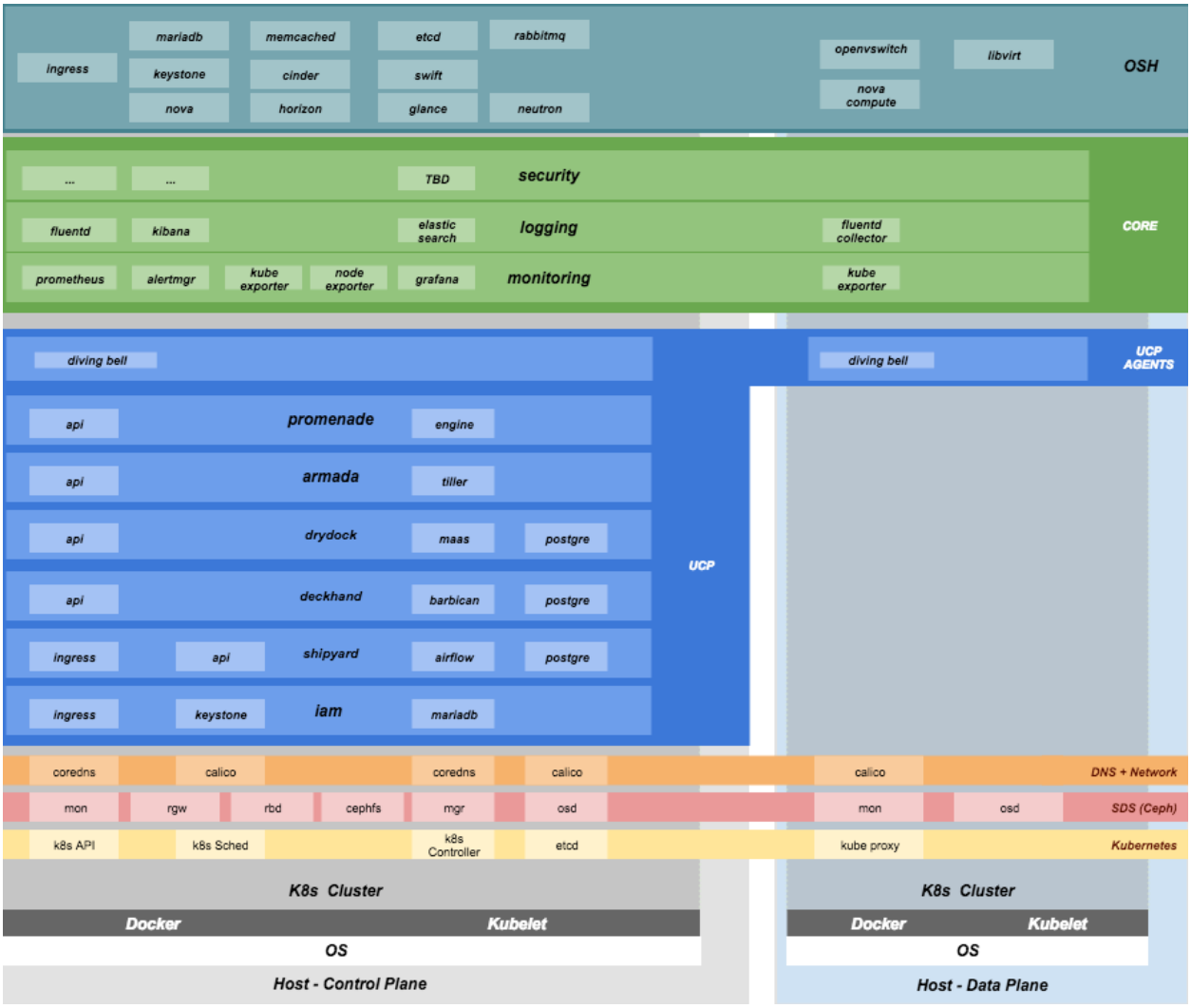
Host Profile: A host profile is a standard way of configuring a bare metal host. It encompasses items such as the number of bonds, bond slaves, physical storage mapping and partitioning, and kernel parameters.

2.1.3 Versioning

Airship reference manifests are delivered monthly as release tags in the Treasuremap.

The releases are verified by Seaworthy, Airsloop, and Airship-in-a-Bottle pipelines before delivery and are recommended for deployments instead of using the master branch directly.

2.1.4 Component Overview



2.1.5 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 / Master nodes**: The nodes that make up the control plane. (Note that the genesis node will be one of the controller nodes)
- **Compute / Worker Nodes**: The nodes that make up the data plane

2.1.6 Hardware Preparation

The Seaworthy site reference shows a production-worthy deployment that includes multiple disks, as well as redundant/bonded network configuration.

Airship hardware requirements are flexible, and the system can be deployed with very minimal requirements if needed (e.g., single disk, single network).

For simplified non-bonded, and single disk examples, see Airsloop.

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.

Disk

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

Network

1. You have a dedicated PXE interface on untagged/native VLAN, 1x1G interface (eno1)
2. You have VLAN segmented networks, 2x10G bonded interfaces (enp67s0f0 and enp68s0f1)
   - Management network (routed/OAM)
   - Calico network (Kubernetes control channel)
   - Storage network
   - Overlay network
   - Public network

See detailed network configuration in the `site/${NEW_SITE}/networks/physical/networks.yaml` configuration file.

2.1.7 Hardware sizing and minimum requirements

Node	Disk	Memory	CPU
Build (laptop)	10 GB	4 GB	1
Genesis/Control	500 GB	64 GB	24
Compute	N/A*	N/A*	N/A*

- Workload driven (determined by host profile)

See detailed hardware configuration in the `site/${NEW_SITE}/networks/profiles` folder.

2.1.8 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 the `treasuremap` git repo as follows

```
git clone https://opendev.org/airship/treasuremap.git
cd treasuremap && git checkout <release-tag>
```

2.1.9 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 that separates 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 the Pegleg documentation on this topic here.

The `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 `site` folder and copy the `seaworthy` site as follows:

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

Remove `seaworthy` specific certificates.

```
rm -f 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 more 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 comments. Search for all instances of `NEWSITE-CHANGEME` in your new site definition. Then 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

Register DNS names

Airship has two virtual IPs.

See `data.vip` in section of `site/${NEW_SITE}/networks/common-addresses.yaml` configuration file. Both are implemented via Kubernetes ingress controller and require FQDNs/DNS.

Register the following list of DNS names:

```
+---+---------------------------+-------------+
| A |              iam-sw.DOMAIN | ingress-vip |
| A |         shipyard-sw.DOMAIN | ingress-vip |
+---+---------------------------+-------------+
| A |  cloudformation-sw.DOMAIN | ingress-vip |
| A |          compute-sw.DOMAIN | ingress-vip |
| A |        dashboard-sw.DOMAIN | ingress-vip |
| A |          grafana-sw.DOMAIN | ingress-vip |
+---+---------------------------+-------------+
| A |         identity-sw.DOMAIN | ingress-vip |
| A |            image-sw.DOMAIN | ingress-vip |
| A |           kibana-sw.DOMAIN | ingress-vip |
| A |           nagios-sw.DOMAIN | ingress-vip |
| A |          network-sw.DOMAIN | ingress-vip |
| A | nova-novncproxy-sw.DOMAIN | ingress-vip |
| A |    object-store-sw.DOMAIN | ingress-vip |
| A |   orchestration-sw.DOMAIN | ingress-vip |
| A |        placement-sw.DOMAIN | ingress-vip |
| A |           volume-sw.DOMAIN | ingress-vip |
+---+---------------------------+-------------+
| A |             maas-sw.DOMAIN | maas-vip    |
| A |          drydock-sw.DOMAIN | maas-vip    |
+---+---------------------------+-------------+
```

Here `DOMAIN` is a name of ingress domain, you can find it in the `data.dns.ingress_domain` section of `site/${NEW_SITE}/secrets/certificates/ingress.yaml` configuration file.

Run the following command to get an up-to-date list of required DNS names:

```
grep -E 'host: .+DOMAIN' site/${NEW_SITE}/software/config/endpoints.yaml | \
    sort -u | awk '{print $2}'
```

Update Secrets

Replace public SSH key under `site/${NEW_SITE}/secrets/publickey/airship_ssh_public_key.yaml` with a lab specific SSH public key. This key is used for MAAS initial

deployment as well as the default user for Divingbell `site/${NEW_SITE}/software/charts/ucp/divingbell/divingbell.yaml`.

Add additional keys and Divingbell substitutions for any other users that require SSH access to the deployed servers. See more details at https://airship-divingbell.readthedocs.io/en/latest/.

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

- Passphrases 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))"
```

Configure certificates in `site/${NEW_SITE}/secrets/certificates/ingress.yaml`, they need to be issued for the domains configured in the `Register DNS names` section.

Caution: It is required to configure valid certificates. Self-signed certificates are not supported.

Control Plane & Tenant Ceph Cluster Notes

Configuration variables for ceph control plane are located in:

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

Configuration variables for tenant ceph are located in:

- `site/${NEW_SITE}/software/charts/osh/openstack-tenant-ceph/ceph-osd.yaml`
- `site/${NEW_SITE}/software/charts/osh/openstack-tenant-ceph/ceph-client.yaml`

Configuration summary:

- 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 block device 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. Instead, they are configured as JBOD when connected through a RAID controller. If the 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 quorum, 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 performing a fresh install, disks are unlabeled or not labeled from a previous Ceph install, so that Ceph chart will not fail disk initialization.

Important: It is highly recommended to use SSD devices for Ceph Journal partitions.

If you have an operating system available on the target hardware, you can determine HDD and SSD devices 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).

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 Ceph Journals, you can pass in a specific partition (e.g., `/dev/sdb1`). Note that it's not required to pre-create these partitions. The Ceph chart will create journal partitions automatically if they don't exist. By default the size of every journal partition is 10G. Make sure there is enough space available to allocate all journal partitions.

Consider the following example where:

- /dev/sda is an operating system RAID-1 device (SSDs for OS root)
- /dev/sd[bc] are SSDs for ceph journals
- /dev/sd[efgh] are HDDs for OSDs

The data section of this file would look like:

```
data:
  values:
    conf:
      storage:
        osd:
          - data:
              type: block-logical
              location: /dev/sde
            journal:
              type: block-logical
              location: /dev/sdb1
          - data:
              type: block-logical
              location: /dev/sdf
            journal:
              type: block-logical
              location: /dev/sdb2
          - data:
              type: block-logical
              location: /dev/sdg
            journal:
              type: block-logical
              location: /dev/sdc1
          - data:
              type: block-logical
              location: /dev/sdh
            journal:
              type: block-logical
              location: /dev/sdc2
```

Manifest linting and combining layers

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

```
sudo tools/airship pegleg site -r /target lint $NEW_SITE
```

Note: `P001` and `P005` 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 P005` to the lint command.

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

```
sudo tools/airship pegleg site -r /target 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.

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

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

This output is required for 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 tools/airship pegleg site -r /target 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

Create an output directory for Promenade certs and run

```
mkdir ${NEW_SITE}_certs
sudo tools/airship promenade generate-certs \
  -o /target/${NEW_SITE}_certs /target/${NEW_SITE}_collected/*.yaml
```

Estimated runtime: About **1 minute**

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

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

Regenerate collected YAML files to include copied certificates:

```
sudo rm -rf ${NEW_SITE}_collected ${NEW_SITE}_certs
sudo tools/airship pegleg site -r /target collect $NEW_SITE \
    -s ${NEW_SITE}_collected
```

Finally, create the Promenade bundle:

```
mkdir ${NEW_SITE}_bundle
sudo tools/airship promenade build-all --validators \
  -o /target/${NEW_SITE}_bundle /target/${NEW_SITE}_collected/*.yaml
```

2.1.10 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 genesis node, the node you will use to seed the rest of your environment. Use standard Ubuntu ISO. Ensure to select the following:

- UTC timezone
- Hostname that matches the genesis hostname given in `data.genesis.hostname` in `site/${NEW_SITE}/networks/common-addresses.yaml`.
- At the `Partition Disks` screen, select `Manual` so that you can setup the same disk partitioning scheme used on the other control plane nodes that will be deployed by MaaS. Select the first logical device that corresponds to one of the RAID-1 arrays already setup in the hardware controller. On this device, setup partitions matching those defined for the `bootdisk` in your control plane host profile found in `site/${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.
- 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 `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.

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, as an alternative, update the genesis hostname in the site definition and then repeat the steps in the previous two sections, "Manifest linting and combining layers" and "Building the Promenade bundle".

Installing matching kernel version

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

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. Example:

```
platform:
  image: 'xenial'
  kernel: 'hwe-16.04'
  kernel_params:
    kernel_package: 'linux-image-4.15.0-46-generic'
```

It is recommended to install matching (and previously tested) kernel

```
sudo apt-get install linux-image-4.15.0-46-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 installed kernel is the latest available. Boot the genesis node using the installed kernel.

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 `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.

Promenade bootstrap

Copy the `${NEW_SITE}_bundle` directory 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: **1h**

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**

2.1.11 Deploy Site with Shipyard

Export valid login credentials for one of the Airship Keystone users defined for the site. Currently there are 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 `site/${NEW_SITE}/secrets/passphrases/ucp_shipyard_keystone_password.yaml`. Example:

```
export OS_AUTH_URL="https://iam-sw.DOMAIN:443/v3"

export OS_USERNAME=shipyard
export OS_PASSWORD=password123
```

Next, load collected site manifests to Shipyard

```
sudo -E tools/airship shipyard create configdocs ${NEW_SITE} \
  --directory=/target/${NEW_SITE}_collected

sudo tools/airship shipyard commit configdocs
```

Estimated runtime: **3m**

Now deploy the site with shipyard:

```
tools/airship shipyard create action deploy_site
```

Estimated runtime: **3h**

Check periodically for successful deployment:

```
tools/airship shipyard get actions
tools/airship shipyard describe action/<ACTION>
```

2.1.12 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`. Example:

```
PasswordAuthentication no
```

Then, restart the ssh service:

```
sudo systemctl restart ssh
```

2.2 Configuration Update Guide

The guide contains the instructions for updating the configuration of a deployed Airship environment. Please refer to Site Authoring and Deployment Guide if you do not have an Airship environment already deployed.

Update of an Airship environment consists of the following stages:

1. **Prepare the configuration**: before deploying any changes, a user should prepare and validate the manifests on a build node using Airship Pegleg.
2. **Deploy the changes**: during this stage, a user uploads the configuration to the Airship environment and starts the deployment using Airship Shipyard.

Note: This guide assumes you have Airship Pegleg and Airship Shipyard tools installed and configured; please refer to Site Authoring and Deployment Guide for the details.

2.2.1 Configuring Airship CLI

Clone the Airship Treasuremap repository and switch to correct version.

```
git clone https://opendev.org/airship/treasuremap
cd treasuremap/
# List available tags.
git tag --list
# Switch to the version your site is using.
git checkout {your-tag}
# Go back to a previous directory.
cd ..
```

Configure environment variables with the name of your site, and specify a path to the directory where site configuration is stored; for this example, we use Airship Seaworthy site:

```
export SITE=seaworthy
export SITE_PATH=treasuremap/site/seaworthy
```

2.2.2 Updating the manifests

Changing the configuration consists of the following steps:

1. Change site manifests.
2. Lint the manifests.
3. Collect the manifests.
4. Copy the manifests to the Airship environment.

Linting and collecting the manifests is done using Airship Pegleg.

For this example, we are going to update a debug level for keystone logs in a site layer.

Note: It is also possible to update the configuration in a global layer; for more details on Airship layering mechanism see Pegleg Definition Artifact Layout documentation.

Create an override file `${SITE_PATH}/software/charts/osh/openstack-keystone/keystone.yaml` with the following content:

```
---
schema: armada/Chart/v1
metadata:
  schema: metadata/Document/v1
  name: keystone
  replacement: true
  layeringDefinition:
    abstract: false
    layer: site
    parentSelector:
      name: keystone-global
    actions:
      - method: merge
        path: .
  storagePolicy: cleartext
data:
  values:
    conf:
      logging:
        logger_keystone:
          level: DEBUG
...
```

Check that the configuration is valid:

```
sudo ./treasuremap/tools/airship pegleg site -r treasuremap/ \
  lint ${SITE}
```

Collect the configuration:

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

Copy the configuration to a node that has the access to the site's Shipyard API, if the current node does not; this node can be one of your controllers:

```
scp -r ${SITE}_collected {genesis-ip}:/home/{user-name}/${SITE}_collected
```

2.2.3 Deploying the changes

After you copied the manifests, there are just a few steps needed to start the deployment:

1. Upload the changes to Airship Deckhand.
2. Start the deployment using Airship Shipyard.

Install Airship CLI as described in *Configuring Airship CLI* section.

Set the name of your site:

:: export SITE=seaworthy

Configure credentials for accessing Shipyard; the password is stored in `ucp_shipyard_keystone_password` secret, you can find it in `site/seaworthy/secrets/passphrases/ucp_shipyard_keystone_password.yaml` configuration file of your site.

```
export OS_USERNAME=shipyard
export OS_PASSWORD={shipyard_password}
```

Upload the changes to Airship Deckhand:

```
# Upload the configuration.
sudo -E ./treasuremap/tools/airship shipyard \
    create configdocs ${SITE} --replace --directory=${SITE}_collected

# Commit the configuration.
sudo -E ./treasuremap/tools/airship shipyard commit configdocs
```

Run the deployment:

```
sudo -E ./treasuremap/tools/airship shipyard create action update_site
```

You can also run `update_software` instead of `update_site` which skips hardware configuration and only applies the changes to services that are running on top of Kubernetes.

Now you can track the deployment progress using the following commands:

```
# Get all actions that were executed on you environment.
sudo -E ./treasuremap/tools/airship shipyard get actions

# Show all the steps within the action.
sudo -E ./treasuremap/tools/airship shipyard describe action/{action_id}
```

All steps will have status `success` when the update finishes.

2.3 Troubleshooting Guide

This guide provides information on troubleshooting of an Airship environment. Debugging of any software component starts with gathering more information about the failure, so the intention of the document is not to describe specific issues that one can encounter, but to provide a generic set of instructions that a user can follow to find the root cause of the problem.

For additional support you can contact the Airship team via IRC or mailing list, use Airship bug tracker to search and create issues.

2.3.1 Configuring Airship CLI

Many commands from this guide use Airship CLI, this section describes how to get it configured on your environment.

```
git clone https://opendev.org/airship/treasuremap
cd treasuremap/
# List available tags.
git tag --list
# Switch to the version your site is using.
git checkout {your-tag}
# Go back to a previous directory.
```

(continues on next page)

(continued from previous page)

```
cd ..
# Run it without arguments to get a help message.
sudo ./treasuremap/tools/airship
```

2.3.2 Manifests Preparation

When you do any configuration changes to the manifests, there are a few commands that you can use to validate the changes without uploading them to the Airship environment.

Run `lint` command for your site; it helps to catch the errors related to documents duplication, broken references, etc.

Example:

```
sudo ./treasuremap/tools/airship pegleg site -r airship-treasuremap/ \
    lint {site-name}
```

If you create configuration overrides or do changes to substitutions, it is recommended to run `render` command this command merges the layers and renders all substitutions. This allows finding what parameters are passed to Helm as overrides for Charts' defaults.

Example:

```
# Saves the result into rendered.txt file.
sudo ./treasuremap/tools/airship pegleg site -r treasuremap/ \
    render -o rendered.txt ${SITE}
```

2.3.3 Deployment Failure

During the deployment, it is important to identify a specific step where it fails, there are two major deployment steps:

1. **Drydock build**: deploys Operating System.
2. **Armada build**: deploys Helm Charts.

After *Configuring Airship CLI*, setup credentials for accessing Shipyard; the password is stored in `ucp_shipyard_keystone_password` secret, you can find it in `site/seaworthy/secrets/passphrases/ucp_shipyard_keystone_password.yaml` configuration file of your site.

```
export OS_USERNAME=shipyard
export OS_PASSWORD={shipyard_password}
```

Now you can use the following commands to access Shipyard:

```
# Get all actions that were executed on you environment.
sudo ./treasuremap/tools/airship shipyard get actions
# Show all the steps within the action.
sudo ./treasuremap/tools/airship shipyard describe action/{action_id}
# Get a bit more details on the step.
sudo ./treasuremap/tools/airship shipyard describe step/{action_id}/armada_build
# Print the logs from the step.
sudo ./treasuremap/tools/airship shipyard logs step/{action_id}/armada_build
```

After the failed step is determined, you can access the logs of a specific service (e.g., drydock-api/maas or armada-api) to get more information on the failure, note that there may be multiple pods of a single service running, you need to check all of them to find where the most recent logs are available.

Example of accessing Armada API logs:

```
# Get all pods running on the cluster and find a name of the pod you are
# interested in.
kubectl get pods -o wide --all-namespaces

# See the logs of specific pod.
kubectl logs -n ucp -f --tail 200 armada-api-d5f757d5-6z6nv
```

In some cases you want to restart your pod, there is no dedicated command for that in Kubernetes. However, you can delete the pod, it will be restarted by Kubernetes to satisfy replication factor.

```
# Restart Armada API service.
kubectl delete pod -n ucp armada-api-d5f757d5-6z6nv
```

2.3.4 Ceph

Many stateful services in Airship rely on Ceph to function correctly. For more information on Ceph debugging follow an official Ceph debugging guide.

Although Ceph tolerates failures of multiple OSDs, it is important to make sure that your Ceph cluster is healthy.

Example:

```
# Get a name of Ceph Monitor pod.
CEPH_MON=$(sudo kubectl get pods --all-namespaces -o=name | \
    grep ceph-mon | sed -n 1p | sed 's|pod/||')
# Get the status of the Ceph cluster.
sudo kubectl exec -it -n ceph ${CEPH_MON} -- ceph -s
```

Cluster is in a helthy state when `health` parameter is set to `HEALTH_OK`.

When the cluster is unhealthy, and some Placement Groups are reported to be in degraded or down states, determine the problem by inspecting the logs of Ceph OSD that is down using `kubectl`.

```
# Get a name of Ceph Monitor pod.
CEPH_MON=$(sudo kubectl get pods --all-namespaces -o=name | \
    grep ceph-mon | sed -n 1p | sed 's|pod/||')
# List a hierarchy of OSDs in the cluster to see what OSDs are down.
sudo kubectl exec -it -n ceph ${CEPH_MON} -- ceph osd tree
```

There are a few other commands that may be useful during the debugging:

```
# Get a name of Ceph Monitor pod.
CEPH_MON=$(sudo kubectl get pods --all-namespaces -o=name | \
    grep ceph-mon | sed -n 1p | sed 's|pod/||')

# Get a detailed information on the status of every Placement Group.
sudo kubectl exec -it -n ceph ${CEPH_MON} -- ceph pg dump

# List allocated block devices.
sudo kubectl exec -it -n ceph ${CEPH_MON} -- rbd ls
# See what client uses the device.
sudo kubectl exec -it -n ceph ${CEPH_MON} -- rbd status \
    kubernetes-dynamic-pvc-e71e65a9-3b99-11e9-bf31-e65b6238af01

# List all Ceph block devices mounted on a specific host.
mount | grep rbd
```

2.4 Seaworthy: Production-grade Airship

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

The site manifests are available at site/seaworthy.

2.4.1 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.

2.4.2 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.

Due to the limited capacity of a test environment, only Ubuntu-based images are used at the moment.

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

2.4.3 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

2.4.4 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.

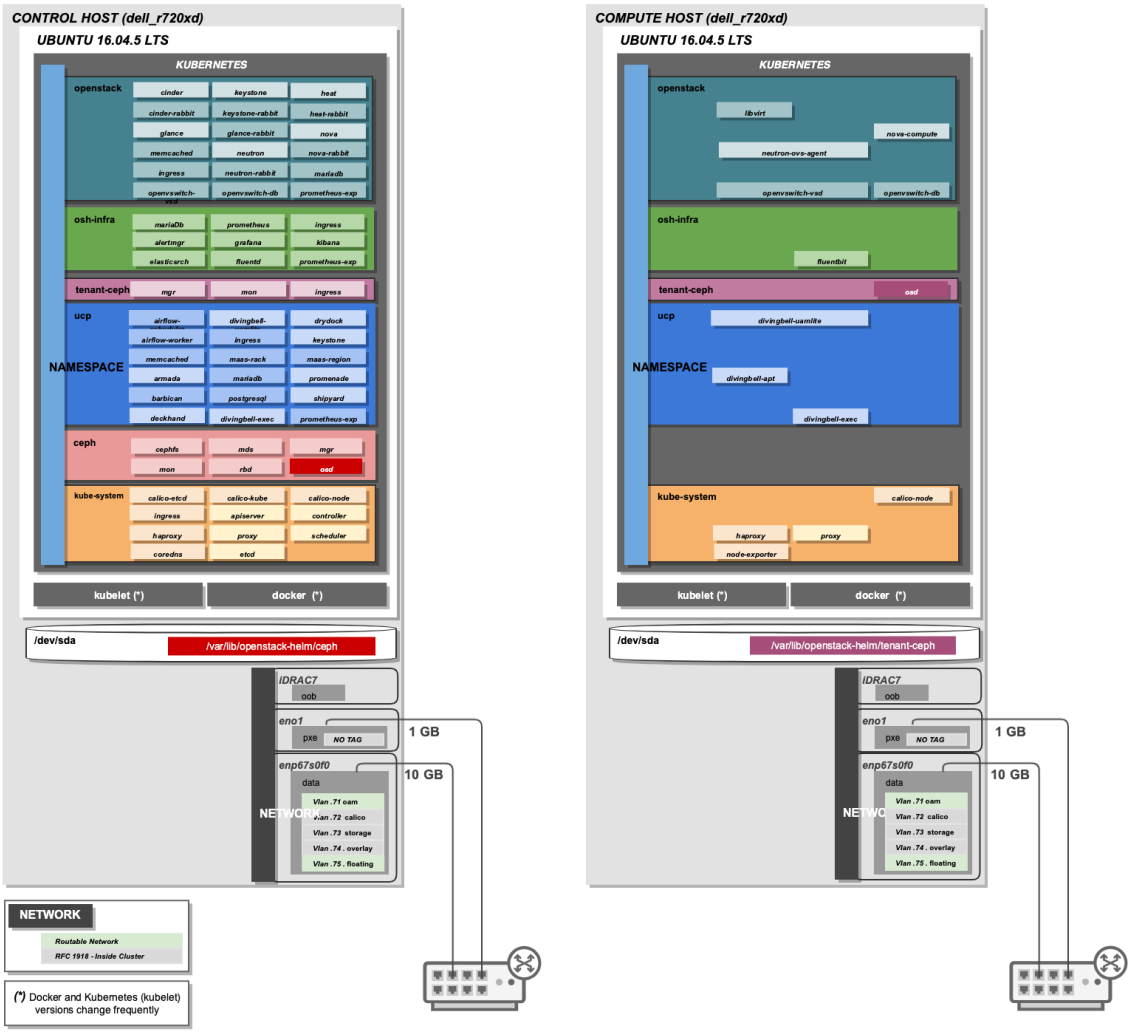
BGP peering is supported but not enabled in this setup, see Calico chart.

2.5 Airsloop: Simple Bare-Metal Airship

Airsloop is a two bare-metal server site deployment reference.

The goal of this site is to be used as a reference for simplified Airship deployments with one control and one or more compute nodes.

It is recommended to get familiar with the Site Authoring and Deployment Guide documentation before deploying Airsloop in the lab. Most steps and concepts including setting up the Genesis node are the same.



Various resiliency and security features are tuned down via configuration.

- Two bare-metal server setup with 1 control, and 1 compute. Most components are scaled to a single replica and doesn't carry any HA as there is only a single control plane host.
- No requirements for DNS/certificates. HTTP and internal cluster DNS is used.
- Ceph set to use the single disk. This generally provides minimalistic no-touch Ceph deployment. No replication of Ceph data (single copy).
- Simplified networking (no bonding). Two network interfaces are used by default (flat PXE, and DATA network with VLANs for OAM, Calico, Storage, and OpenStack Overlay).

- Generic hostnames used (airsloop-control-1, airsloop-compute-1) that simplifies generation of k8s certificates.

Airsloop site manifests are available at site/airsloop.

2.5.1 Hardware

While HW configuration is flexible, Airsloop reference manifests reflect a single control and a single compute node. The aim of this is to create a minimalistic lab/demo reference environment.

Increasing the number of compute nodes will require site overrides to align parts of the system such as Ceph OSDs, etcd, etc.

See host profiles for the servers here.

Node	Hostnames
control	airsloop-control-1
compute	airsloop-compute-1

2.5.2 Network

Physical (underlay) networks are described in Drydock site configuration here.

It defines OOB (iLO/IPMI), untagged PXE, and multiple tagged general use networks. Also no bonded interfaces are used in Airsloop deployment.

The networking reference is simplified compared to Airship Seaworthy site. There are only two NICs required (excluding oob), one for PXE and another one for the rest of the networks separated using VLAN segmentation.

Below is the reference network configuration:

NICs	VLANs	Names	CIDRs
oob	N/A	oob	10.22.104.0/24
pxe	N/A	pxe	10.22.70.0/24
data	71	oam	10.22.71.0/24
	72	calico	10.22.72.0/24
	73	storage	10.22.73.0/24
	74	overlay	10.22.74.0/24

Calico overlay for k8s POD networking uses IPIP mesh.

2.5.3 Storage

Because Airsloop is a minimalistic deployment the required number of disks is just one per node. That disk is not only used by the OS but also by Ceph Journals and OSDs. The way that this is achieved is by using directories and not extra disks for Ceph storage. Ceph OSD configuration can be changed in a Ceph chart override.

The following Ceph chart configuration is used:

```
osd:
  - data:
      type: directory
      location: /var/lib/openstack-helm/ceph/osd/osd-one
    journal:
```

(continues on next page)

(continued from previous page)

```
      type: directory
      location: /var/lib/openstack-helm/ceph/osd/journal-one
```

2.5.4 Host Profiles

Host profiles in Airship are tightly coupled with the hardware profiles. That means every disk or interface which is described in host profiles should have a corresponding reference to the hardware profile which is being used.

Airship always identifies every NIC or disk by its PCI or SCSI address and that means that the interfaces and the disks that are defined in host and hardware profiles should have the correct PCI and SCSI addresses objectively.

Let's give an example by following the host profile of Airsloop site.

In this Host Profile is defined that the slave interface that will be used for the pxe boot will be the pxe_nic01. That means a corresponding entry should exist in this Hardware Profile which it does. So when drydock and maas try to deploy the node it will identify the interface by the PCI address that is written in the Hardware profile.

A simple way to find out which PCI or SCSI address corresponds to which NIC or Disk is to use the lshw command. More information about that command can be found Here.

2.5.5 Extend Cluster

This section describes what changes need to be made to the existing manifests of Airsloop for the addition of an extra compute node to the cluster.

First and foremost the user should go to the nodes.yaml file and add an extra section for the new compute node.

The next step is to add a similar section as the existing airsloop-compute-1 section to the pki-catalog.yaml. This is essential for the correct generation of certificates and the correct communication between the nodes in the cluster.

Also every time the user adds an extra compute node to the cluster then the number of OSDs that are managed by this manifest Ceph-client should be increased by one.

Last step is to regenerate the certificates which correspond to this certificates.yaml file so the changes in the pki-catalog.yaml file takes place. This can be done through the promenade CLI.

2.5.6 Getting Started

Update Site Manifests.

Carefully review site manifests (site/airsloop) and update the configuration to match the hardware, networking setup and other specifics of the lab.

See more details at Site Authoring and Deployment Guide.

Note: Many manifest files (YAMLs) contain documentation in comments that instruct what changes are required for specific sections.

---

1. Build Site Documents

```
tools/airship pegleg site -r /target collect airsloop -s collect

mkdir certs
```

(continues on next page)

(continued from previous page)

```
tools/airship promenade generate-certs -o /target/certs /target/collect/*.yaml

mkdir bundle
tools/airship promenade build-all -o /target/bundle /target/collect/*.yaml /target/
↪certs/*.yaml
```

See more details at Building Site documents, use site `airsloop`.

2. Deploy Genesis

Deploy the Genesis node, see more details at Genesis node.

Genesis is the first node in the cluster and serves as a control node. In Airsloop configuration Genesis is the only control node (airsloop-control-1).

Airsloop is using non-bonded network interfaces:

```
auto lo
iface lo inet loopback

auto eno1
iface eno1 inet static
  address 10.22.70.21/24

auto enp67s0f0
iface enp67s0f0 inet manual

auto enp67s0f0.71
iface enp67s0f0.71 inet static
  address 10.22.71.21/24
  gateway 10.22.71.1
  dns-nameservers 8.8.8.8 8.8.4.4
  vlan-raw-device enp67s0f0
  vlan_id 71

auto enp67s0f0.72
iface enp67s0f0.72 inet static
  address 10.22.72.21/24
  vlan-raw-device enp67s0f0
  vlan_id 72

auto enp67s0f0.73
iface enp67s0f0.73 inet static
  address 10.22.73.21/24
  vlan-raw-device enp67s0f0
  vlan_id 73

auto enp67s0f0.74
iface enp67s0f0.74 inet static
  address 10.22.74.21/24
  vlan-raw-device enp67s0f0
  vlan_id 74
```

Execute Genesis bootstrap script on the Genesis server.

```
sudo ./genesis.sh
```

3. Deploy Site

```
tools/airship shipyard create configdocs design --directory=/target/collect
tools/airship shipyard commit configdocs

tools/airship shipyard create action deploy_site

tools/shipyard get actions
```

See more details at Deploy Site with Shipyard.

2.5.7 Deploying Behind a Proxy

The following documents show the main differences you need to make in order to have airsloop run behind a proxy.

Note: The "-" sign refers to a line that needs to be omitted (replaced), and the "+" sign refers to a line replacing the omitted line, or simply a line that needs to be added to your yaml.

Under site/airsloop/software/charts/osh/openstack-glance/ create a glance.yaml file as follows:

```
---
schema: armada/Chart/v1
metadata:
  schema: metadata/Document/v1
  replacement: true
  name: glance
  layeringDefinition:
    abstract: false
    layer: site
    parentSelector:
      name: glance-type
    actions:
      - method: merge
        path: .
  storagePolicy: cleartext
data:
  test:
    enabled: false
...
```

Under site/airsloop/software/config/ create a versions.yaml file in the following format:

```
---
data:
  charts:
    kubernetes:
      apiserver:
        proxy_server: proxy.example.com:8080
      apiserver-htk:
        proxy_server: proxy.example.com:8080
      calico:
        calico:
          proxy_server: proxy.example.com:8080
        calico-htk:
          proxy_server: proxy.example.com:8080
        etcd:
```

(continues on next page)

(continued from previous page)

```
        proxy_server: proxy.example.com:8080
      etcd-htk:
        proxy_server: proxy.example.com:8080
    controller-manager:
      proxy_server: proxy.example.com:8080
    controller-manager-htk:
      proxy_server: proxy.example.com:8080
    coredns:
      proxy_server: proxy.example.com:8080
    coredns-htk:
      proxy_server: proxy.example.com:8080
    etcd:
      proxy_server: proxy.example.com:8080
    etcd-htk:
      proxy_server: proxy.example.com:8080
    haproxy:
      proxy_server: proxy.example.com:8080
    haproxy-htk:
      proxy_server: proxy.example.com:8080
    ingress:
      proxy_server: proxy.example.com:8080
    ingress-htk:
      proxy_server: proxy.example.com:8080
    proxy:
      proxy_server: proxy.example.com:8080
    proxy-htk:
      proxy_server: proxy.example.com:8080
    scheduler:
      proxy_server: proxy.example.com:8080
    scheduler-htk:
      proxy_server: proxy.example.com:8080
  osh:
    barbican:
      proxy_server: proxy.example.com:8080
    cinder:
      proxy_server: proxy.example.com:8080
    cinder-htk:
      proxy_server: proxy.example.com:8080
    glance:
      proxy_server: proxy.example.com:8080
    glance-htk:
      proxy_server: proxy.example.com:8080
    heat:
      proxy_server: proxy.example.com:8080
    heat-htk:
      proxy_server: proxy.example.com:8080
    helm_toolkit:
      proxy_server: proxy.example.com:8080
    horizon:
      proxy_server: proxy.example.com:8080
    horizon-htk:
      proxy_server: proxy.example.com:8080
    ingress:
      proxy_server: proxy.example.com:8080
    ingress-htk:
      proxy_server: proxy.example.com:8080
    keystone:
```

(continues on next page)

(continued from previous page)

```
      proxy_server: proxy.example.com:8080
    keystone-htk:
      proxy_server: proxy.example.com:8080
    libvirt:
      proxy_server: proxy.example.com:8080
    libvirt-htk:
      proxy_server: proxy.example.com:8080
    mariadb:
      proxy_server: proxy.example.com:8080
    mariadb-htk:
      proxy_server: proxy.example.com:8080
    memcached:
      proxy_server: proxy.example.com:8080
    memcached-htk:
      proxy_server: proxy.example.com:8080
    neutron:
      proxy_server: proxy.example.com:8080
    neutron-htk:
      proxy_server: proxy.example.com:8080
    nova:
      proxy_server: proxy.example.com:8080
    nova-htk:
      proxy_server: proxy.example.com:8080
    openvswitch:
      proxy_server: proxy.example.com:8080
    openvswitch-htk:
      proxy_server: proxy.example.com:8080
    rabbitmq:
      proxy_server: proxy.example.com:8080
    rabbitmq-htk:
      proxy_server: proxy.example.com:8080
    tempest:
      proxy_server: proxy.example.com:8080
    tempest-htk:
      proxy_server: proxy.example.com:8080
  osh_infra:
    elasticsearch:
      proxy_server: proxy.example.com:8080
    fluentbit:
      proxy_server: proxy.example.com:8080
    fluentd:
      proxy_server: proxy.example.com:8080
    grafana:
      proxy_server: proxy.example.com:8080
    helm_toolkit:
      proxy_server: proxy.example.com:8080
    kibana:
      proxy_server: proxy.example.com:8080
    nagios:
      proxy_server: proxy.example.com:8080
    nfs_provisioner:
      proxy_server: proxy.example.com:8080
    podsecuritypolicy:
      proxy_server: proxy.example.com:8080
    prometheus:
      proxy_server: proxy.example.com:8080
    prometheus_alertmanager:
```

(continues on next page)

(continued from previous page)

```
      proxy_server: proxy.example.com:8080
    prometheus_kube_state_metrics:
      proxy_server: proxy.example.com:8080
    prometheus_node_exporter:
      proxy_server: proxy.example.com:8080
    prometheus_openstack_exporter:
      proxy_server: proxy.example.com:8080
    prometheus_process_exporter:
      proxy_server: proxy.example.com:8080
  ucp:
    armada:
      proxy_server: proxy.example.com:8080
    armada-htk:
      proxy_server: proxy.example.com:8080
    barbican:
      proxy_server: proxy.example.com:8080
    barbican-htk:
      proxy_server: proxy.example.com:8080
    ceph-client:
      proxy_server: proxy.example.com:8080
    ceph-htk:
      proxy_server: proxy.example.com:8080
    ceph-mon:
      proxy_server: proxy.example.com:8080
    ceph-osd:
      proxy_server: proxy.example.com:8080
    ceph-provisioners:
      proxy_server: proxy.example.com:8080
    ceph-rgw:
      proxy_server: proxy.example.com:8080
    deckhand:
      proxy_server: proxy.example.com:8080
    deckhand-htk:
      proxy_server: proxy.example.com:8080
    divingbell:
      proxy_server: proxy.example.com:8080
    divingbell-htk:
      proxy_server: proxy.example.com:8080
    drydock:
      proxy_server: proxy.example.com:8080
    drydock-htk:
      proxy_server: proxy.example.com:8080
    ingress:
      proxy_server: proxy.example.com:8080
    ingress-htk:
      proxy_server: proxy.example.com:8080
    keystone:
      proxy_server: proxy.example.com:8080
    keystone-htk:
      proxy_server: proxy.example.com:8080
    maas:
      proxy_server: proxy.example.com:8080
    maas-htk:
      proxy_server: proxy.example.com:8080
    mariadb:
      proxy_server: proxy.example.com:8080
    mariadb-htk:
```

(continues on next page)

(continued from previous page)

```
        proxy_server: proxy.example.com:8080
      memcached:
        proxy_server: proxy.example.com:8080
      memcached-htk:
        proxy_server: proxy.example.com:8080
      postgresql:
        proxy_server: proxy.example.com:8080
      postgresql-htk:
        proxy_server: proxy.example.com:8080
      promenade:
        proxy_server: proxy.example.com:8080
      promenade-htk:
        proxy_server: proxy.example.com:8080
      rabbitmq:
        proxy_server: proxy.example.com:8080
      rabbitmq-htk:
        proxy_server: proxy.example.com:8080
      shipyard:
        proxy_server: proxy.example.com:8080
      shipyard-htk:
        proxy_server: proxy.example.com:8080
      tenant-ceph-client:
        proxy_server: proxy.example.com:8080
      tenant-ceph-htk:
        proxy_server: proxy.example.com:8080
      tenant-ceph-mon:
        proxy_server: proxy.example.com:8080
      tenant-ceph-osd:
        proxy_server: proxy.example.com:8080
      tenant-ceph-provisioners:
        proxy_server: proxy.example.com:8080
      tenant-ceph-rgw:
        proxy_server: proxy.example.com:8080
      tiller:
        proxy_server: proxy.example.com:8080
      tiller-htk:
        proxy_server: proxy.example.com:8080
metadata:
  name: software-versions
  replacement: true
  layeringDefinition:
    abstract: false
    layer: site
    parentSelector:
      name: software-versions-global
    actions:
      - method: merge
        path: .
  storagePolicy: cleartext
  schema: metadata/Document/v1
schema: pegleg/SoftwareVersions/v1
...
```

Update site/airsloop/networks/common-addresses.yaml to add the proxy information as follows:

```
  # settings are correct and reachable in your environment; otherwise update
  # them with the correct values for your environment.
```

(continues on next page)

(continued from previous page)

```
   proxy:
-    http: ""
-    https: ""
-    no_proxy: []
+    http: "proxy.example.com:8080"
+    https: "proxy.example.com:8080"
+    no_proxy:
+      - 127.0.0.1
```

Under site/airsloop/software/charts/ucp/ create the file maas.yaml with the following format:

```
---
# This file defines site-specific deviations for MaaS.
schema: armada/Chart/v1
metadata:
  schema: metadata/Document/v1
  replacement: true
  name: ucp-maas
  layeringDefinition:
    abstract: false
    layer: site
    parentSelector:
      name: ucp-maas-type
    actions:
      - method: merge
        path: .
  storagePolicy: cleartext
data:
    values:
      conf:
        maas:
          proxy:
            proxy_enabled: true
            peer_proxy_enabled: true
            proxy_server: 'http://proxy.example.com:8080'
...
```

Under site/airsloop/software/charts/ucp/ create a promenade.yaml file in the following format:

```
---
# This file defines site-specific deviations for Promenade.
schema: armada/Chart/v1
metadata:
  schema: metadata/Document/v1
  replacement: true
  name: ucp-promenade
  layeringDefinition:
    abstract: false
    layer: site
    parentSelector:
      name: ucp-promenade-type
    actions:
      - method: merge
        path: .
  storagePolicy: cleartext
data:
  values:
```

(continues on next page)

(continued from previous page)

```
    pod:
      env:
        promenade_api:
          - name: http_proxy
            value: http://proxy.example.com:8080
          - name: https_proxy
            value: http://proxy.example.com:8080
          - name: no_proxy
            value: "127.0.0.1,localhost,kubernetes,kubernetes.default,kubernetes.
↪default.svc,kubernetes.default.svc.cluster.local,.cluster.local"
          - name: HTTP_PROXY
            value: http://proxy.example.com:8080
          - name: HTTP_PROXY
            value: http://proxy.example.com:8080
          - name: HTTPS_PROXY
            value: http://proxy.example.com:8080
          - name: NO_PROXY
            value: "127.0.0.1,localhost,kubernetes,kubernetes.default,kubernetes.
↪default.svc,kubernetes.default.svc.cluster.local,.cluster.local"
...
```

2.6 Airskiff: Lightweight Airship for Dev

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

2.6.1 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 Minikube.
- 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.

2.6.2 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},10.0.2.15,.svc.cluster.local
export NO_PROXY=${NO_PROXY},10.0.2.15,.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 `10.0.2.15` is the advertised IP address of the minikube Kubernetes cluster. Replace the addresses if your configuration does not match the one defined above.

2.6.3 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 `treasuremap` repository.

Clone Dependencies

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

CURRENT_DIR="$(pwd)"
: "${INSTALL_PATH:="../"}"
: "${OSH_INFRA_COMMIT:="8ba46703ee9fab0115e4b7f62ea43e0798c36872"}"
: "${CLONE_ARMADA:=true}"
: "${CLONE_DECKHAND:=true}"
: "${CLONE_SHIPYARD:=true}"
```

(continues on next page)

(continued from previous page)

```
cd ${INSTALL_PATH}

# Clone Airship projects
if [[ ${CLONE_ARMADA} = true ]] ; then
    git clone https://opendev.org/airship/armada.git
fi
if [[ ${CLONE_DECKHAND} = true ]] ; then
    git clone https://opendev.org/airship/deckhand.git
fi
if [[ ${CLONE_SHIPYARD} = true ]] ; then
    git clone https://opendev.org/airship/shipyard.git
fi

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

cd openstack-helm-infra
git checkout "${OSH_INFRA_COMMIT}"

cd "${CURRENT_DIR}"
```

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

```
./tools/deployment/airskiff/developer/000-clone-dependencies.sh
```

Deploy Kubernetes with Minikube

```
#!/bin/bash

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 K8s with Minikube
cd "${OSH_INFRA_PATH}"
bash -c "./tools/deployment/common/005-deploy-k8s.sh"

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

# Add user to Docker group
# NOTE: This requires re-authentication. Restart your shell.
sudo adduser "$(whoami)" docker
sudo su - "$USER" -c bash <<'END_SCRIPT'
if echo $(groups) | grep -qv 'docker'; then
    echo "You need to logout to apply group permissions"
    echo "Please logout and login"
fi
END_SCRIPT

cd "${CURRENT_DIR}"
```

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

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

Restart your shell session

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

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: 'password123'
      project_name: 'admin'
      project_domain_name: 'default'
      user_domain_name: 'default'
      auth_url: 'http://keystone-api.ucp.svc.cluster.local:5000/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-api.openstack.svc.cluster.local:5000/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

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

(continues on next page)

(continued from previous page)

```
: "${PEGLEG:="./tools/airship pegleg"}"
: "${PL_SITE:="airskiff"}"
: "${TARGET_MANIFEST:="cluster-bootstrap"}"

# Render documents
${PEGLEG} site -r . render "${PL_SITE}" -o airskiff.yaml

# Set permissions o+r, beacause these files need to be readable
# for Armada in the container
AIRSKIFF_PERMISSIONS=$(stat --format '%a' airskiff.yaml)
KUBE_CONFIG_PERMISSIONS=$(stat --format '%a' ~/.kube/config)

sudo chmod 0644 airskiff.yaml
sudo chmod 0644 ~/.kube/config

# In the event that this docker command fails, we want to continue the script
# and reset the file permissions.
set +e

# Download latest Armada image and deploy Airship components
docker run --rm --net host -p 8000:8000 --name armada \
    -v ~/.kube/config:/armada/.kube/config \
    -v "$(pwd)"/airskiff.yaml:/airskiff.yaml \
    -v "${INSTALL_PATH}":/airship-components \
    quay.io/airshipit/armada:latest-ubuntu_bionic \
    apply /airskiff.yaml --target-manifest $TARGET_MANIFEST

# Set back permissions of the files
sudo chmod "${AIRSKIFF_PERMISSIONS}" airskiff.yaml
sudo chmod "${KUBE_CONFIG_PERMISSIONS}" ~/.kube/config
```

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

# Lint deployment documents
: "${AIRSHIP_PATH:="./tools/airship"}"
: "${PEGLEG:="${AIRSHIP_PATH} pegleg"}"
: "${SHIPYARD:="${AIRSHIP_PATH} shipyard"}"
: "${PL_SITE:="airskiff"}"

# Source OpenStack credentials for Airship utility scripts
. tools/deployment/airskiff/common/os-env.sh

# 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.
${PEGLEG} site -r . lint "${PL_SITE}" -x P001 -x P009

# Collect deployment documents
```

(continues on next page)

(continued from previous page)

```
: "${PL_OUTPUT:="peggles"}"
mkdir -p ${PL_OUTPUT}

TERM_OPTS="-l info" ${PEGLEG} site -r . collect ${PL_SITE} -s ${PL_OUTPUT}

# Start the deployment
${SHIPYARD} create configdocs airskiff-design \
             --replace \
             --directory=${PL_OUTPUT}
${SHIPYARD} commit configdocs
${SHIPYARD} create action update_software --allow-intermediate-commits
```

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

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

2.6.4 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.

2.6.5 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/airskiff/developer/030-armada-bootstrap.sh
```

2.6.6 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
```

2.7 Development Guide

2.7.1 Welcome

Thank you for your interest in Airship. Our community is eager to help you contribute to the success of our project and welcome you as a member of our community!

We invite you to reach out to us at any time via the Airship mailing list or #airshipit IRC channel on freenode.

Welcome aboard!

2.7.2 Getting Started

Airship is a collection of open source tools for automating cloud provisioning and management. Airship provides a declarative framework for defining and managing the life cycle of open infrastructure tools and the underlying hardware. These tools include OpenStack for virtual machines, Kubernetes for container orchestration, and MaaS for bare metal, with planned support for OpenStack Ironic.

We recommend that new contributors begin by reading the high-level architecture overview included in our treasuremap documentation. The architectural overview introduces each Airship component, their core responsibilities, and their integration points.

2.7.3 Deep Dive

Each Airship component is accompanied by its own documentation that provides an extensive overview of the component. With so many components, it can be challenging to find a starting point.

We recommend the following:

Try an Airship environment

Airship provides two single-node environments for demo and development purpose.

Airship-in-a-Bottle is a set of reference documents and shell scripts that stand up a full Airship environment with the execution of a script.

Airskiff is a light-weight development environment bundled with a set of deployment scripts that provides a single-node Airship environment. Airskiff uses minikube to bootstrap Kubernetes, so it does not include Drydock, MaaS, or Promenade.

Additionally, we provide a reference architecture for easily deploying a smaller, demo site.

Airsloop is a fully-authored Airship site that can be quickly deployed as a baremetal, demo lab.

Focus on a component

When starting out, focusing on one Airship component allows you to become intricately familiar with the responsibilities of that component and understand its function in the Airship integration. Because the components are modeled after each other, you will also become familiar with the same patterns and conventions that all Airship components use.

Airship source code lives in the OpenDev Airship namespace. To clone an Airship project, execute the following, replacing <*component*> with the name of the Airship component you want to clone.

Refer to the component's documentation to get started. A list of each component's documentation is listed below for reference:

- Armada
- Deckhand
- Divingbell
- Drydock
- Pegleg
- Promenade
- Shipyard

Find a Storyboard task or story

Airship work items are tracked using Storyboard. A board of items can be found here.

Once you find an item to work on, simply assign the item to yourself or leave a comment that you plan to provide implementation for the item.

2.7.4 Testing Changes

Testing of Airship changes can be accomplished several ways:

1. Standalone, single component testing
2. Integration testing
3. Linting, unit, and functional tests/linting

Note: Testing changes to charts in Airship repositories is best accomplished using the integration method describe below.

Standalone Testing

Standalone testing of Airship components, i.e. using an Airship component as a Python project, provides the quickest feedback loop of the three methods and allows developers to make changes on the fly. We recommend testing initial code changes using this method to see results in real-time.

Each Airship component written in Python has pre-requisites and guides for running the project in a standalone capacity. Refer to the documentation listed below.

- Armada
- Deckhand
- Drydock
- Pegleg
- Promenade
- Shipyard

Integration Testing

While each Airship component supports individual usage, Airship components have several integration points that should be exercised after modifying functionality.

We maintain several environments that encompass these integration points:

1. Airskiff: Integration of Armada, Deckhand, Shipyard, and Pegleg
2. *Airship-in-a-Bottle Multinode*: Full Airship integration

For changes that merely impact software delivery components, exercising a full Airskiff deployment is often sufficient. Otherwise, we recommend using the Airship-in-a-Bottle Multinode environment.

Each environment's documentation covers the process required to build and test component images.

Final Checks

Airship projects provide Makefiles to run unit, integration, and functional tests as well as lint Python code for PEP8 compliance and Helm charts for successful template rendering. All checks are gated by Zuul before a change can be merged. For more information on executing these checks, refer to project-specific documentation.

Third party CI tools, such as Jenkins, report results on Airship-in-a-Bottle patches. These can be exposed using the "Toggle CI" button in the bottom left-hand page of any gerrit change.

2.7.5 Pushing code

Airship uses the OpenDev gerrit for code review. Refer to the OpenStack Contributing Guide for a tutorial on submitting changes to Gerrit code review.

2.7.6 Next steps

Upon pushing a change to gerrit, Zuul continuous integration will post job results on your patch. Refer to the job output by clicking on the job itself to determine if further action is required. If it's not clear why a job failed, please reach out to a team member in IRC. We are happy to assist!

Assuming all continuous integration jobs succeed, Airship community members and core developers will review your patch and provide feedback. Many patches are submitted to Airship projects each day. If your patch does not receive feedback for several days, please reach out using IRC or the Airship mailing list.

2.7.7 Merging code

Like most OpenDev projects, Airship patches require two +2 code review votes from core members to merge. Once you have addressed all outstanding feedback, your change will be merged.

2.7.8 Beyond

Congratulations! After your first change merges, please keep up-to-date with the team. We hold two weekly meetings for project and design discussion:

Our weekly #airshipit IRC meeting provides an opportunity to discuss project operations.

Our weekly design call provides an opportunity for in-depth discussion of new and existing Airship features.

For more information on the times of each meeting, refer to the Airship wiki.