# Airship Cookbook

v 0.72

# Contents

## Revision History

| Version | Details |
|---|---|
| **0.1** | Initial Draft |
| **0.2** | Preliminary details about airship added |
| **0.3** | Recommended Hardware & setup added |
| **0.4** | Details about deployment on various platforms added |
| **0.5** | Details of SR-IOV related setup added |
| **0.6** | FAQ section is added. |
| **0.7** | Added FAQ related to hardware, VLAN & SR-IOV. |
| **0.71** | Micro-cruiser BOM information added. |
| **0.72** | Simplified deployment process |

# Getting Started with Airship

## What is Airship?

Airship is a lifecycle manager for datacenter infrastructure. It leverages opensource tools as containers to automate cloud provisioning. The complete lifecycle of containers is production grade and provided by Kubernetes & helm by using declarative YAML documents. Although airship is envisioned to be a general-purpose infrastructure management tool; currently it's mainly used in containerization of Openstack.

Airship consists of a number of components. This is what each of the component does in a nutshell:

Kubernetes: A highly resilient opensource container life cycle manager and orchestrator by Google. It handles deployment & management of container workloads across several hosts in a cluster.

Helm: An installer for Kubernetes workloads – analogues to yum or apt-get in Linux. Any application can be installed in a repeatable manner using its own 'helm chart'.

Manifest files: Text files that contain Airship configurations in YAML format. Usually each component in airship has its own manifest file. These files are kept in a version control system like Gerrit for easier fallback and tracking.

Pegleg: Can aggregate several manifests files into one big one so that airship components can consume it.

Drydock: Before airship cluster can be deployed, drydock configures & provision everything you need on bare metal by consuming the manifest configuration files. It boots up your new nodes, provisions them, configure networks & make them ready for the Kubernetes deployment.

Promenade: Once drydock finished provisioning & setting up bare metal nodes, promenade installs a Kubernetes cluster on the nodes.

Shipyard: The entry point for any established airship cluster. It is an acyclic graph controller for managing full lifecycle of Kubernetes & Openstack. Any provisioning request you'll send to airship will be received by shipyard.

Deckhand: Stores configuration documents of all the airship cluster & provides several handy functions to easily clone, distribute and use config documents across several clusters.

Armada: Manager of multiple helm charts. Resolves all applications dependencies & centralize all configuration in a single YAML file.

## Where Can I get detailed documentation for Airship?

The detailed documentation of airship can be seen at https://airship-treasuremap.readthedocs.io/en/latest/

## What is the high-level architecture of Airship?

The figure below shows the collection of technologies that enable the Airship.

The orange highlights the Airship projects that facilitate the creation and life-cycle management of an undercloud platform that is used to enable OpenStack based Airship with the scale, speed, flexibility and operational predictability this infrastructure must deliver.

The blue highlights the Open Source projects that Airship leverages/ integrates with to deploy the undercloud platform.

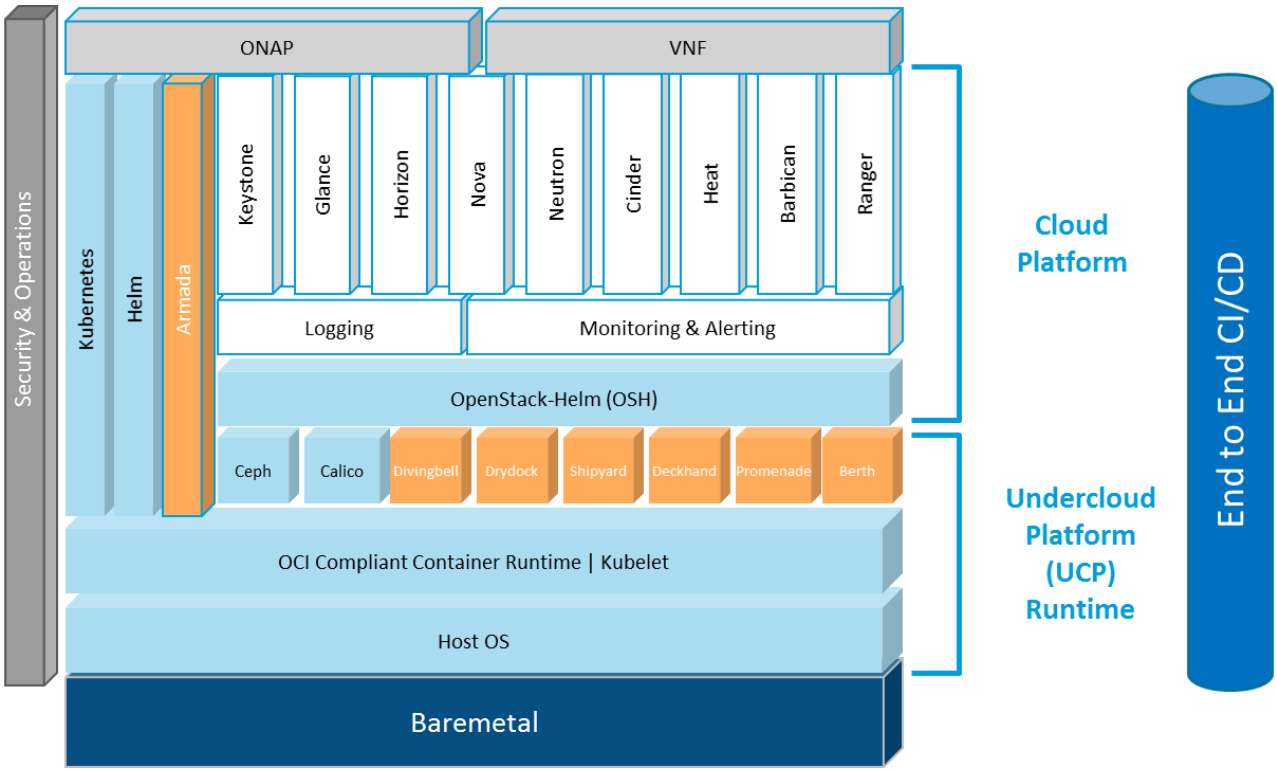
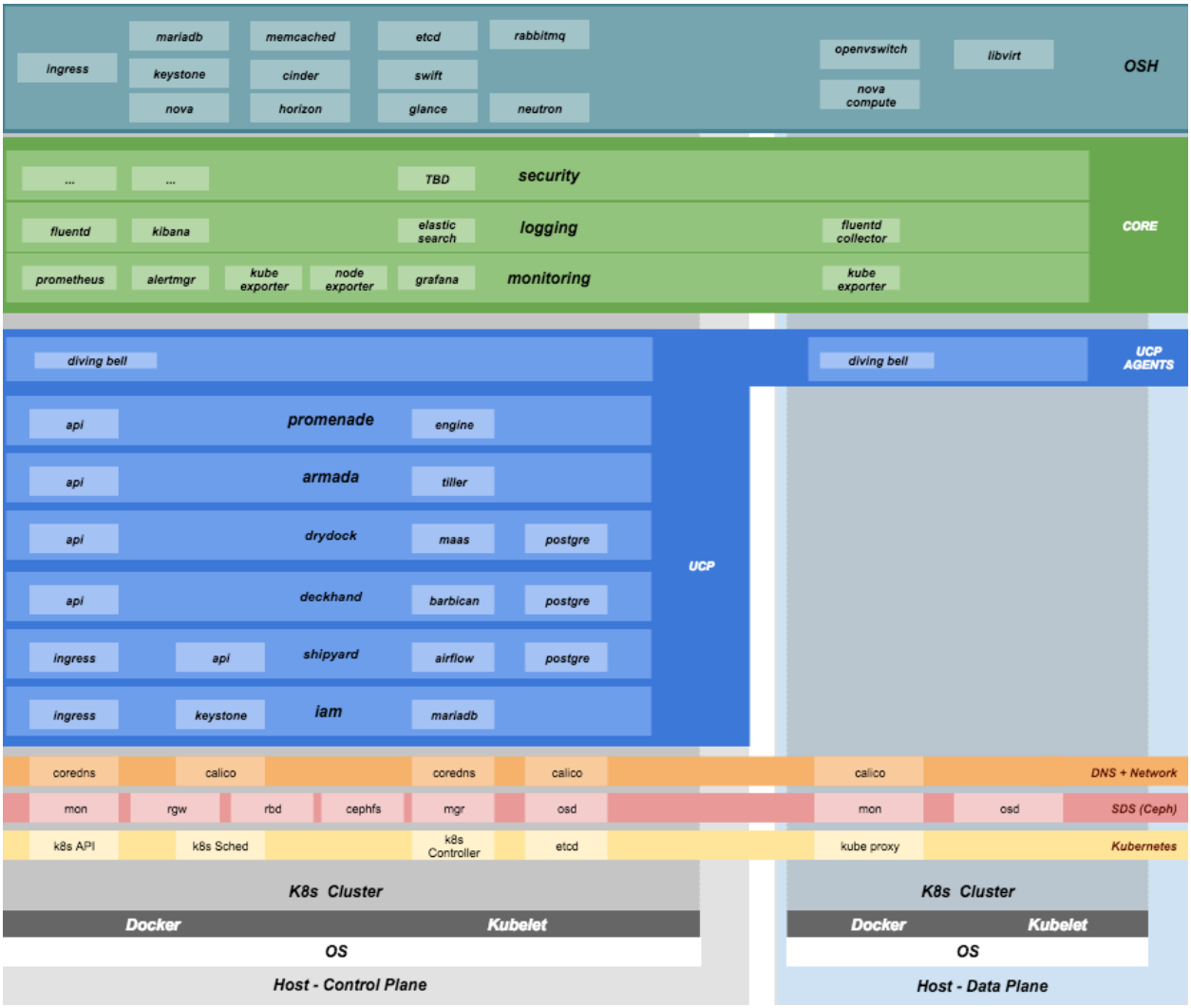



***Figure 1: Cloud w/ Airship - Reference Architecture***

The several components of airship are stacked the following way in a Kubernetes cluster.


ingress
mariadb
keystone
nova
memcached
cinder
horizon
etcd
swift
glance
rabbitmq
neutron
openvswitch
nova compute
libvirt
OSH
...
...
TBD
security
fluentd
kibana
elastic search
logging
fluentd collector
CORE
prometheus
alertmgr
kube exporter
node exporter
grafana
monitoring
kube exporter
diving bell
diving bell
UCP AGENTS
api
promenade
engine
api
armada
tiller
api
drydock
maas
postgre
api
deckhand
barbican
postgre
ingress
api
shipyard
airflow
postgre
ingress
keystone
iam
mariadb
UCP
coredns
calico
coredns
calico
calico
DNS + Network
mon
rgw
rbd
cephfs
mgr
osd
mon
osd
SDS (Ceph)
k8s API
k8s Sched
k8s Controller
etcd
kube proxy
Kubernetes
K8s Cluster
Docker
Kubelet
OS
Host - Control Plane
K8s Cluster
Docker
Kubelet
OS
Host - Data Plane

## Site Design and Repository Layout

The complete site design of an airship site is handled with text based manifest files. Due to this approach airship offers a lot of flexibility. With flexibility comes complexity, so to begin with, before making any changes, it is very important to understand:

- How site manifests are organized into different layers and put together with Pegleg
- How Deckhand performs basic operations like document replacement and substitution
- How this configuration code (site manifest) is divided between several git repositories
- How and why these repositories are branched, and what workflows are followed when working with these branches

Some sample manifest files can be seen here.

### Layers

There are several types of layer that can be used in airship. In some production environment, it is observed that having a three layered manifest files is beneficial. Please note that the names of these layers can be different based on the usage and the source of these manifest files, but they can be easily correlated. These layers being:

#### *Global Layer*

This is configuration data that applies to all deployments. This data will be applied to all the site. Some examples of the data which global manifest files contain are:

- General Airship and OpenStack settings (timeouts, Corporate LDAP server, Corporate mail servers, and so on)
- versions.yaml file containing pinned versions of all software used for deployments and updates
- The "golden/reference site" for each unique site-type (i.e. the versioned "template site" that is copied to deploy other sites of the same type)

#### *Type Layer*

This is configuration data that applies to specific types of deployments. Any site that belongs to a particular type of site would manifest data from the respective type manifest files.

- The number of control plane nodes
- The number of BGP peers.
- Any site using a particular type of networking solution.

#### *Site Layer*

This is configuration data that applies at the level of an individual site deployment.

- The site name
- OAM IP addresses

- Site-specific layering overrides (for example, if something from Global or Site-Type Layer was wrong for this site, we can override that config data)
- The specific branch, commit, or tag of other repositories that contain the remainder of the manifest data

For more information about layering and site authoring, please check the following resources:

- https://opendev.org/airship/pegleg/src/commit/025fa74939ecbbc2542e79e097caf48485d41e02/doc/source/authoring_strategy.rst
- https://opendev.org/airship/pegleg/src/commit/025fa74939ecbbc2542e79e097caf48485d41e02/doc/source/artifacts.rst

### Repositories

- **airship-security-manifests** – This repository is used to store secrets regardless of layer. This repository also have a 'global' and a 'site' layer where majority of the secrets for individual sites are stored. Some examples of security data stored in this layer are:
  - Passphrases, CAs, certs, and private server keys are stored at the Site layer in this repo
  - Passphrases and keys common to all sites (for example, artifactory and docker keys) are stored at the Global layer in this repo

- **airship-site-manifests repo** - All other Site layer information not falling under the security manifests repo above is stored in this repo. There are subfolders that correspond to every site deployed.
- **airship-manifests repo** - All other Global and Site-Type layer information not falling under the security manifests repo above is stored in this repo.

### Repo Branches and Tagging

- airship -manifests - This repo follows the Cactus git release model.
- airship-security-manifests and airship-site-manifests - Since any given site can only be one thing (one state) at a time (for example, Ocata vs. Pike), there is no need to branch site manifests or their secrets. Therefore these repos use the Master branch. Also note:
- Each of the *site-definition.yaml* in the *airship-site-manifests* repo reference the specific branch or tag to use from airship-manifests (global/type) repo. This is how a site "chooses" it's Airship release, by associating with a specific tag/release from the airship-manifests repo in *site-definition.yaml*. For example, one site may reference a 1.0.0 release, while another can reference a 1.0.1 release. Sites may also pin to specific versions of airship-security-manifests, which is independent of the official "AIRSHIP version" defined by tags of airship-manifests.

### How to commit changes

There are a number of factors which determine where changes are made and which processes are followed. The following set of questions aim to address those different situation-dependent cases.

### *Determine the 'Type' of Change*

#### Software version change

Software version for any application means updating its docker image. Once the updated docker image is placed in the binary repository (e.g. Artifactory), the *versions.yaml* manifest file in airship-manifest can be updated with the newer version of software. 'When' does this update takes place can follow the corporation release management schedule.

It is possible to change / update the software version of a single site by creating a site level 'override'. Site level manifests have a priority over the global and type manifest files and any data present in the site level manifest is used for choosing application version. however, it is not advisable to use site manifest for individual software versions. Overhead for managing each software version individually for sites can exponentially grow. Updates to software should be staged as a global change.

#### Configuration change

Any configuration changes to a the global or 'type' layers can be done by manipulating the relevant files in their directories. However, it's advisable to follow a release management cycle to perform updates. On the other hand, 'site' and 'security' manifest changes can be applied directly to the master branch of the respective site. Whenever possible, global & type configuration changes are preferred to the site level updates as they are more manageable. Site level changes should only be done in case of some special requirement which can't be achieved otherwise. Global and type level changes are also verified by CI/CD testing (mainly GF or BF upgrade testing), while site level changes are not put to any such validation.

Furthermore, it is highly advisable to always include a comprehensive commit message to the changes that are being pushed.

#### Quantifying the Scope of Change

A change to a manifest file can be of several types:

1. It can be applied to all sites or a particular site-type. For example a certain timeout value can be increased in the 'global' or 'type' site manifest, which will then be propagated to all the sites.
2. The change can be applied to all the sites, but *not generically*. For example rotating the RabbitMQ password for all sites. Each site has its own unique password; hence this password rotation will add this information to all site level parameters in the 'site' manifests.
3. The change can be a site specific change. Adding IP addresses for a particular site, creating a network policy specific to a site would fall under this category.

#### When should changes be applied?

Depending on the type of change, different strategies can be adopted. Changes that can wait a release can be applied via the 'global' or 'type' manifests. When the next release cycle of airship is applied, these changes will be a part of it. However, sometimes changes can't wait for the release cycles and must be applied immediately. For such situations, 'site' manifest files can be updated independently from the release cycles and applied to sites. These changes may be applied out-of-band apart from any planned releases.

## What are some useful links for Airship?

Detailed documentation of airship: https://airship-treasuremap.readthedocs.io/en/latest/

Opendev repository for airship: https://opendev.org/airship

## Installing a Local Airship Instance (aka Airship in a Bottle).

Airship in a bottle is an airship instance deployed in a single VM. This gives opportunity to a developer to easily setup airship environment locally for testing & development. In order to install airship in a bottle locally, you need to ensure that:

1. You have ubuntu 16.04 LTS installed on the host.
2. You have ssh access to the host.
3. You have enough resources (CPU/Mem/Disk) on the host to run virtual machine
4. Host machine have access to internet for package/iso downloads. Some corporate internet connection might block access to packages required for airship installation.
5. You have access rights to perform 'sudo' operations in host.

KVM install, VM creation and airship install process:

1. SSH into your host using your own account (ssh user@<host-ip>)
2. Install kvm and virt-manager on the host

```
 sudo apt-get install qemu-kvm libvirt-bin virtinst bridge-utils cpu-checker
```

3. Once installation is complete, validate kvm is ok

```
 kvm-ok
INFO: /dev/kvm exists
KVM acceleration can be used
```

4. Now download a ubuntu ISO on local server for VM install

```
#wget http://releases.ubuntu.com/16.04/ubuntu-16.04.6-server-amd64.iso
```

5. VM creation process on KVM

```
virt-install --name=<VM_Name> --vcpus=4 --memory=32768 --
cdrom=/<PATH_ISO>/ubuntu-16.04.6-server-amd64.iso --disk size=200 –graphics
vnc –os-variant=ubnutu
```

6. Configure vnc login from another terminal over ssh:

```
$ sudo virsh dumpxml <VM_Name> | grep vnc
<graphics type='vnc' port='5901' autoport='yes' listen='127.0.0.1'>
```

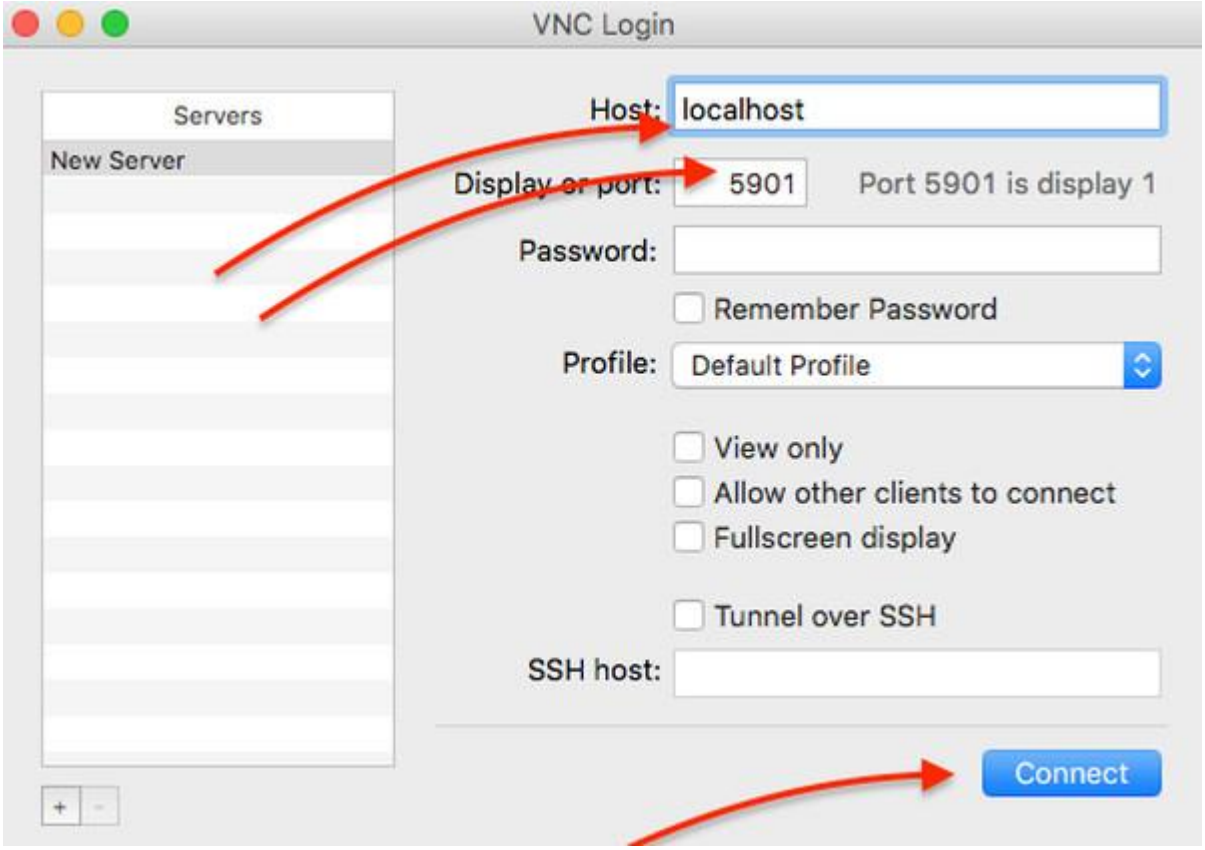
7. If you have trouble using the above-mentioned command, you can also use the following:

```
$ sudo virsh vncdisplay <VM_Name>
```

8. Please note down the port value (i.e. 5901). You need to use an SSH client to setup tunnel and a VNC client to access the remote vnc server. Type the following SSH port forwarding command from your client/desktop:

```
$ ssh username@<VM_Name> -L 5901:127.0.0.1:5901
```

9. Once you have ssh tunnel established, you can point your VNC client at your own 127.0.0.1 (localhost) address and port 5901 as follows:




And follow the instruction on the screen to finish the install process on the VM

10. Once you have VM up and running with ubuntu 16.04. login into the VM using ssh

```
#ssh <username>@<VM_IP>
```

11. Airship (all in one) install steps

```
#sudo -i
# mkdir -p /root/deploy && cd "$_"
#git clone https://opendev.org/airship/treasuremap/
Cloning into 'treasuremap'...
remote: Enumerating objects: 4382, done.
remote: Counting objects: 100% (4382/4382), done.
remote: Compressing objects: 100% (1778/1778), done.
remote: Total 4382 (delta 2624), reused 4038 (delta 2414)
```

```
Receiving objects: 100% (4382/4382), 1.77 MiB | 0 bytes/s, done.
Resolving deltas: 100% (2624/2624), done.
Checking connectivity... done.

cd /root/deploy/treasuremap/tools/deployment/aiab/

./airship-in-a-bottle.sh

Welcome to Airship in a Bottle

 /--------------------\
|                      \
|        |---|          \----
|        | x |               \
|        |---|                |
|          |                  /
|     \____|____/       /----
|                      /
 \--------------------/


A prototype example of deploying the Airship suite on a single VM.


This example will run through:
 - Setup
 - Genesis of Airship (Kubernetes)
 - Basic deployment of Openstack (including Nova, Neutron, and Horizon using
Openstack Helm)
 - VM creation automation using Heat

The expected runtime of this script is greater than 1 hour


The minimum recommended size of the Ubuntu 16.04 VM is 4 vCPUs, 20GB of RAM
with 32GB disk space.
Let's collect some information about your VM to get started.
Is your HOST IFACE eth0? (Y/n)<Y>
This example will run through:
 - Setup
 - Genesis of Airship (Kubernetes)
 - Basic deployment of Openstack (including Nova, Neutron, and Horizon using
Openstack Helm)
 - VM creation automation using Heat

The expected runtime of this script is greater than 1 hour

The minimum recommended size of the Ubuntu 16.04 VM is 4 vCPUs, 20GB of RAM
with 32GB disk space.
Let's collect some information about your VM to get started.
Is your HOST IFACE eth0? (Y/n) y
Is your LOCAL IP 10.0.0.4? (Y/n) y
++ hostname -s
+ export SHORT_HOSTNAME=allinone
+ SHORT_HOSTNAME=allinone
+ set +x
Updating /etc/hosts with: 10.0.0.4 allinone
```

```
10.0.0.4 allinone
+ export HOSTIP=10.0.0.4
+ HOSTIP=10.0.0.4
+ export HOSTCIDR=10.0.0.4/32
+ HOSTCIDR=10.0.0.4/32
+ export NODE_NET_IFACE=eth0
+ NODE_NET_IFACE=eth0
+ export TARGET_SITE=aiab
+ TARGET_SITE=aiab
+ set +x
Using DNS servers 168.63.129.16 and 168.63.129.16.
```

12. You should wait for 60-70 min before you see All in one airship is ready for you to consume.

```
NAMESPACE     NAME                                                  READY
STATUS            RESTARTS   AGE
kube-system   airship-coredns-test                                   0/1
Completed         0          1h
kube-system   airship-haproxy-haproxy-test                           0/1
Completed         0          1h
kube-system   airship-kubernetes-calico-etcd-etcd-test               0/1
Completed         0          1h
kube-system   airship-kubernetes-etcd-etcd-test                      0/1
Completed         0          1h
kube-system   auxiliary-etcd-allinone                                3/3
Running           0          1h
kube-system   bootstrap-armada-allinone                              4/4
Running           0          1h
kube-system   calico-etcd-allinone                                   1/1
Running           0          1h
kube-system   calico-etcd-anchor-mkqng                               1/1
Running           0          1h
kube-system   calico-kube-controllers-74b4879ddd-5vb96               1/1
Running           1          1h
kube-system   calico-node-j9lqv                                      1/1
Running           0          1h
kube-system   calico-settings-lkmsw                                  0/1
Completed         0          1h
kube-system   coredns-78b6d6db4-68c6z                                1/1
Running           0          1h
kube-system   coredns-78b6d6db4-7qxf9                                1/1
Running           0          1h
kube-system   coredns-78b6d6db4-9ml8f                                1/1
Running           0          1h
kube-system   haproxy-allinone                                       1/1
Running           1          1h
kube-system   haproxy-anchor-zfksm                                   1/1
Running           0          1h
kube-system   ingress-8db44f6cf-kfjhh                                1/1
Running           0          1h
kube-system   ingress-error-pages-55f95944fc-t9784                   1/1
Running           0          1h
```

```
kube-system   kubernetes-apiserver-allinone                          1/1
Running           0          1h
kube-system   kubernetes-apiserver-anchor-f2xhx                      1/1
Running           0          1h
kube-system   kubernetes-controller-manager-allinone                 1/1
Running           2          1h
kube-system   kubernetes-controller-manager-anchor-mp8d6             1/1
Running           0          1h
kube-system   kubernetes-etcd-allinone                               1/1
Running           0          1h
kube-system   kubernetes-etcd-anchor-266hg                           1/1
Running           0          1h
kube-system   kubernetes-proxy-85kzm                                 1/1
Running           0          1h
kube-system   kubernetes-scheduler-allinone                          1/1
Running           2          1h
kube-system   kubernetes-scheduler-anchor-2b6rl                      1/1
Running           0          1h
nfs           nfs-provisioner-6697458b7d-hw4sf                       1/1
Running           0          1h
```

13. At the end of successful install, you will see following message

```
OpenStack Horizon dashboard is available on this host at the following URL:

  http://<Internal_IP_Address_of_your_VM>:31724

Credentials:
  Domain: default
  Username: admin
  Password: <********>

OpenStack CLI commands could be launched via `./openstack` script, e.g.:
  # cd /root/deploy/treasuremap/..//treasuremap/tools/
  # ./openstack stack list
  ...

Other dashboards:

  MAAS: http:// <Internal_IP_Address_of_your_VM>/MAAS/ admin/<********>
  Airship Shipyard Airflow DAG: http:// <Internal_IP_Address_of_your_VM>/

Airship itself does not have a dashboard.

Other endpoints and credentials are listed in the following locations:
  /root/deploy/treasuremap/..//type/sloop/config/endpoints.yaml
  /root/deploy/treasuremap/..//treasuremap/site/aiab/secrets/passphrases/

Exposed ports of services can be listed with the following command:
  # kubectl get services --all-namespaces | grep -v ClusterIP
  ...

+ your_next_steps
```

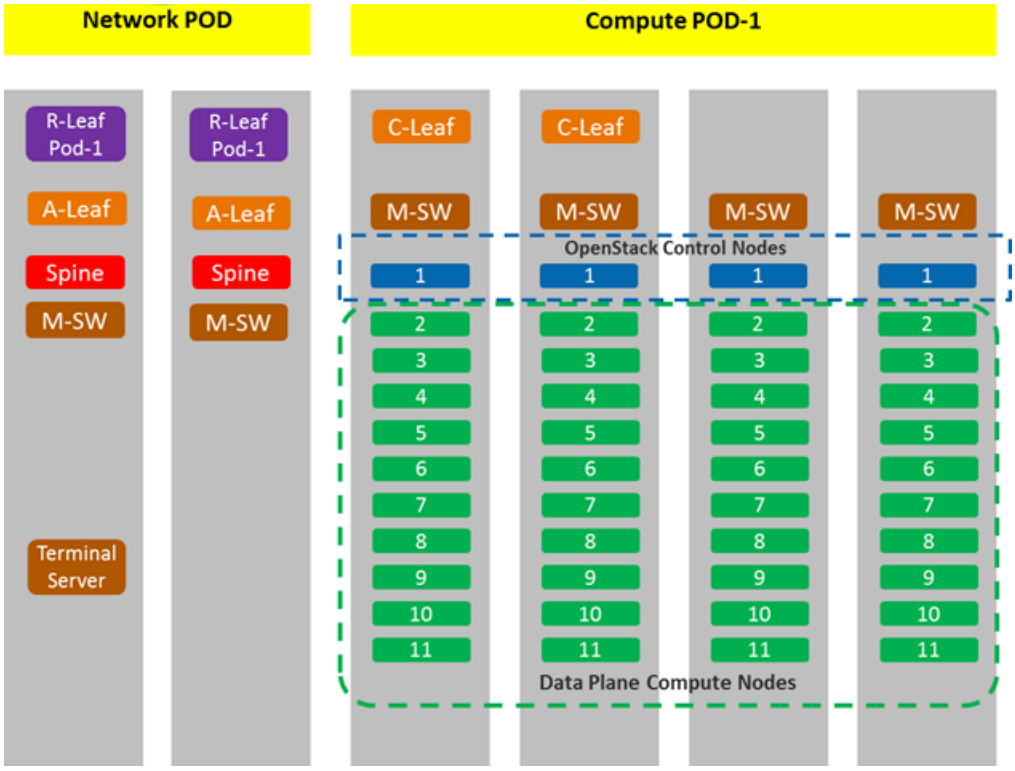
```
+ set +x

------------------------------------------------------------

Airship has completed deployment of OpenStack (OpenStack-Helm).

Explore Airship Treasuremap repository and documentation
available at the following URLs:

  https://opendev.org/airship/treasuremap/
  https://airship-treasuremap.readthedocs.io/

------------------------------------------------------------
```

## Deploying your first application in airship

# Working with Airship Clusters

## Physical Rack Layouts

Although there is no standard or hard requirement to size a given airship deployment a certain way; below mentioned is a configuration which has been used by some consumers in their environments and can be a starting point for dimensioning a particular airship cluster.




### Fabric devices

| Device | Description |
|---|---|
| Aggregation Leaf (Agg-Leaf) | Provides connectivity between WAN networks & fabric devices. |
| Compute Leaf (C-Leaf) | Provides connectivity to computes. |
| Compute/Aggregation Combo Leaf (C/Agg-Leaf) | This leaf combines the functions of an Agg-leaf and C-leaf. It provides connectivity to WAN networks & to compute servers. |
| Spine | Provides connectivity between the Agg-leaf or C/Agg-Leaf and the other C-leaves in the site. |

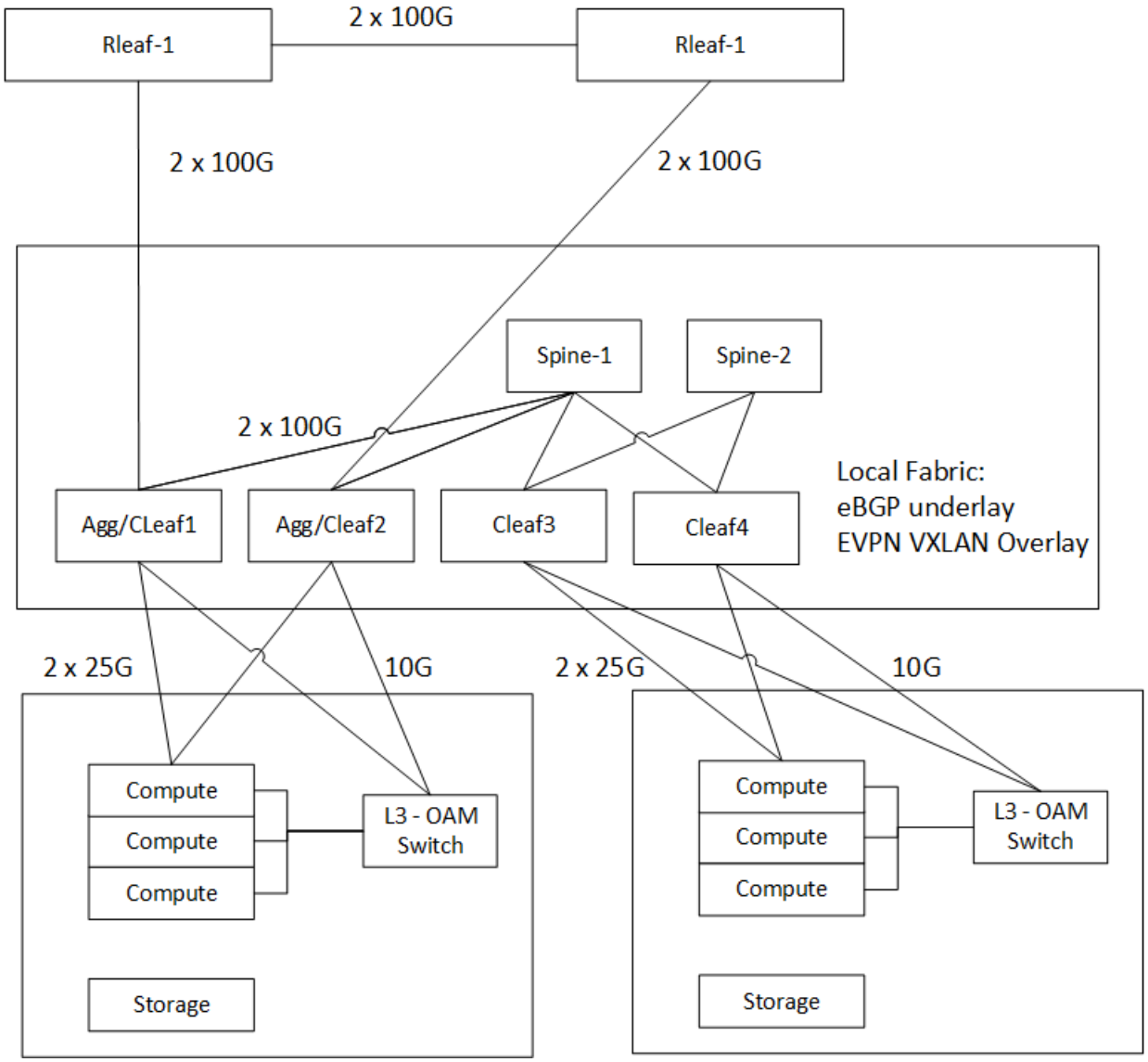
## Physical Connectivity Overview

At a high level, following are the devices connected to fabric devices:

• Each server will be connected with:
    4x25gige to a pair of C-leaves
    2x1gige copper to L2 OAM switch
• L2 OAM switches:
    Northbound: will be dual homed to a pair of C-leaf (using 10gige fiber)
    Southbound: will be connected to servers iLO/PXE and storage mgmt. interfaces (using 1gige copper)
• Agg Leaf:
  Each R-leaf will be single homed to Agg-leaf using multiple 100gige links (the Bandwidth varies between sites)

## Logical Rack Layout

An example of a logical rack layout for airship cluster can be as mentioned in this diagram:



## Minimal HW Setup – Micro Cruiser

Test environments sometimes don't need a full blown deployment of airship with several control and compute nodes. In a case where a complete performance testing is not desired, a minimal hardware setup containing 3 or (preferably) 4 control nodes & 4 compute (preferably 10) nodes for airship can be deployed. The control nodes shouldn't be reduced beyond 3 nodes as it would interfere with quorum requirements of some important components like etcd.

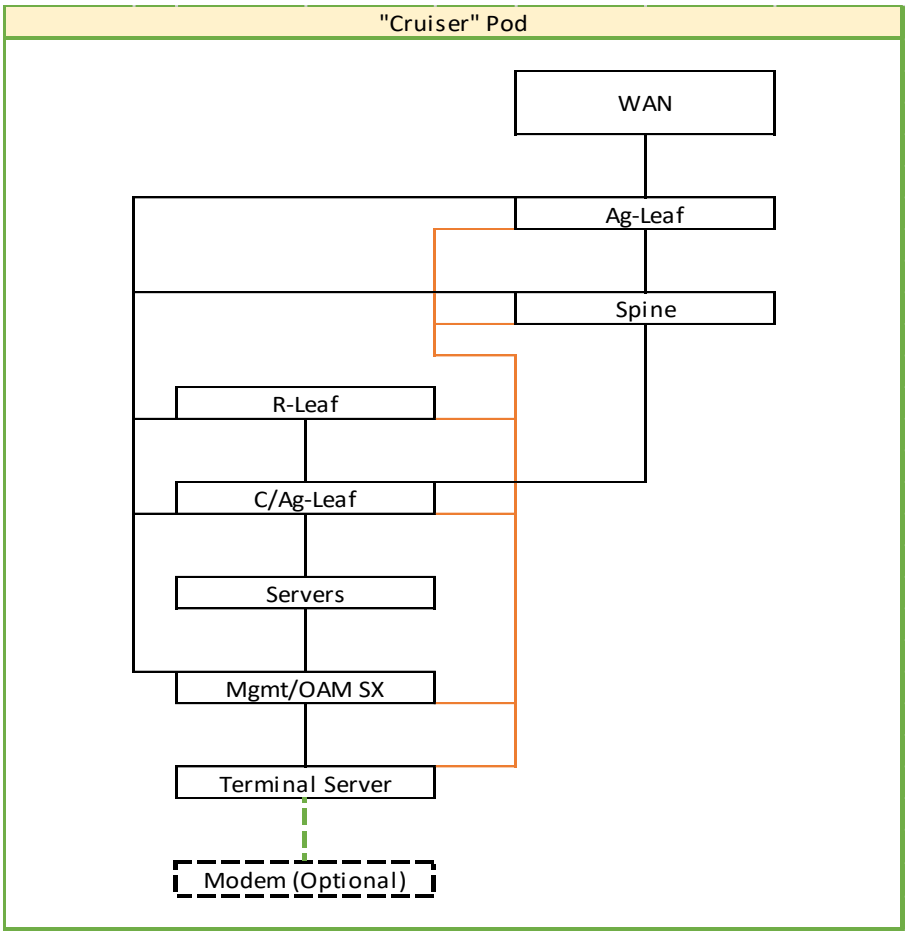
A suggested hardware BOM is mentioned under:

| | |
|---|---|
| Size of Lab Zone | Micro-Cruiser |
| Type of pod | N/A |
| Network Power | AC |
| Network Airflow | Back_to_Front |
| Server Power | AC |
| Server Vendor | Dell |
| R-Leaf | 5501 - 200G |
| C-Leaf Layout | Multi-Rack |
| No of C-Leaf pairs | 1 |
| C/Ag-Leaf to Spine distance | Short < 100m |
| Tenant Storage | N/A |
| Server Connectivity | SR - Fiber |
| Standalone Agg Leaf | N/A |
| No of racks with OAM SX | 4 |
| BOM for which pod | Initial |
| No of compute racks | 2 |
| No of network racks | 2 |
| No of WAN circuits | 2 |
| Speed of WAN circuits | 10G |
| No of control plane servers | 4 |
| No of tenant servers - GP | 8 |
| No of tenant servers - SSD storage optimized | N/A |
| No of tenant servers - HDD storage optimized | N/A |

| Key: |
|---|
| Data Entry |
| Fixed Value |
| Calculated Value |

## Layout

A logical outlook of the cruiser pod is shown under:




## Hardware

| Component | Function | Vendor | Model | Configuration | Qty | Unit Power (W) | Total Power (W) | Unit Size (RU) | Total Size (RU) | Comments |
|---|---|---|---|---|---|---|---|---|---|---|
| Servers | Control Compute - GP Server | Dell | R740XD | Dell 740XD: R740XD-NC1.5-25GB-AC-44CORE-384GBRAM-4X480GBSSD-6X2.4TBHDD-2X3.84TB SSD-2X1GB-4X10GB-4X25GB | 4 | 415 | 1660 | 2 | 8 | |

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| Servers | Tenant Compute - GP Server | Dell | R740XD | Dell 740XD: R740XD-NC1.5-25GB-AC-44CORE-384GBRAM-4X480GBSSD-6X2.4TBHDD-2X3.84TB SSD-2X1GB-4X10GB-4X25GB | 8 | 415 | 3320 | 2 | 16 | Minimum 4 computes required, maximum upto 40 if using multi-rack TOA C-Leaf deployment approach |
| Storage | Block - Tenants | Pure Storage | FA-M20R2-ETH-38TB-38/0 FA-M20R2-ETH-76TB-38/38 | HW Pure Storage FlashArray M20R2-ETH-30TB-38/0 - Lab HW PURE STORAGE FlashArray m20R2 ETH 76TB 38/38 - Prod | 0 | | | | | NOT REQUIRED IN EVERY EDGE LOCATION Future preference is to use tenant CEPH |
| Network | C/Ag-Leaf | Accton | AS-7816-64X-O-AC-B-US | 64x100G TH2, power-to-port airflow, AC | 2 | 600 | 1200 | 2 | 4 | Preference is to use power to port airflow model |
| Network | Ag-Leaf | Accton | AS-7816-64X-O-AC-B-US | 64x100G TH2, power-to-port airflow, AC | 2 | 600 | 1200 | 2 | 4 | Preference is to use power to port airflow model |
| Network | Spine | Accton | AS-7816-64X-O-AC-B-US | 64x100G TH2, power-to-port airflow, AC | 2 | 600 | 1200 | 2 | 4 | Required if more than 1 pair of leaf in a site, maximum 12 pairs |
| Network | R-Leaf | Cisco | NCS-5501-SE | 40x10G+4x100G Qumran - AC | 2 | 430 | 860 | 1 | 2 | Not required in all locations and for all use cases. Required in locations with L7 VNF's. |
| Network | Management Switch | Cisco | WS-C3650-48TQ-E AC power | 48x1G + 4x10G AC power | 4 | 85 | 340 | 1 | 4 | |

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| Network | Terminal Server | Cisco | ISR4431 AC power | Base + 16 Console ports AC power | 1 | 85 | 85 | 1 | 1 | |
| Network | Terminal Server | Cisco | NIM-16A + CAB-ASYNC-8 | Additional 16 console ports + Async Cable | 0 | 5 | 0 | 0 | 0 | Can insert maximum of 3 cards of which 1 comes with base |
| Modem | OOB Access | Current BAU models can be reused | | | 1 | 25 | 25 | 1 | 1 | |

## Cables

| Category | Component | Function | Vendor | Model | Configuration | Qty | Comments |
|---|---|---|---|---|---|---|---|
| Cables | Servers | Server Breakout | AMC Optics | 4x25G DAC 3m | 1m or 3m DAC - 100G to 4x25G | 0 | Breakout cable with transciever built-in |
| Cables | Network | Leaf Loopback | AMC Optics | 100G DAC 1m | 1m DAC - 100G to 100G | 0 | Cable with transceiver built-in |
| Cables | Network | C/Ag-Leaf to C/Ag-Leaf | | | 100G MPO Cable | 2 | |
| Cables | Network | Ag-Leaf to Ag-Leaf | | | 100G MPO Cable | 2 | |
| Cables | Network | Leaf Loopback | | | 100G MPO Cable | 6 | |
| Cables | Servers | Server to C/Ag-Leaf | | | MPO breakout 100G to 4x25G - SR | 12 | |
| Cables | Network | C/Ag-Leaf to Spine | | | 100G MPO Cable | 8 | |
| Cables | Network | C/Ag-Leaf to Spine | | | 100G SMF LC-LC | 0 | |
| Cables | Network | R-Leaf to C/Ag-Leaf | | | 100G MPO Cable | 8 | |
| Cables | Network | R-Leaf to C/Ag-Leaf | | | 100G SMF LC-LC | 0 | |
| Cables | Network | Mgmt ports | | | RJ-45 Cat 6e | 24 | |
| Cables | Network | Server iLO ports | | | RJ-45 Cat 6e | 12 | |
| Cables | Network | Console ports | | | RJ-45 Cat 6e | 13 | |
| Cables | Network | Storage Breakout | | | MPO breakout 40G to 4x10G - SR | 0 | |
| Cables | Network | Mgmt SX to Leaf | | | MPO breakout 40G to 4x10G - SR | 3 | |
| Cables | Network | Term Server to Mgmt SX | | | RJ-45 Cat 6e | 2 | |

## Optics

| Category | Component | Function | Vendor | Model | Configuration | Qty | Comments |
|---|---|---|---|---|---|---|---|
| Optics | Network | C/Ag-Leaf to Spine | Finisar | 100GSR4 | XCVR,QSFP28,100m,100GBASE-SR4,4x 850nm VCSEL,MMF,pull tab,MPO,ROHS compliant, 3.3V, 3.5W, 0/70°C operation | 16 | Requires MPO Cable |
| Optics | Network | C/Ag-Leaf to Spine | Finisar | 100GLR4 | XCVR,QSFP28,10km,100GBASE-LR4,4x 13XXnm LAN-WDM DFB,SMF,pull tab,LC,ROHS compliant, 3.3V, 3.5W, 0/70°C operation | 0 | |
| Optics | Network | C/Ag-Leaf to Spine | Finisar | 100GCWDM4 | XCVR,QSFP28,2km,100GE CWDM4,4X 13XXnm CWDM DFB,SMF,pull tab,LC,ROHS compliant, 3.3V, 3.5W, 0/70°C operation | 0 | |
| Optics | Network | Ag-Leaf to Spine | Finisar | 100GSR4 | XCVR,QSFP28,100m,100GBASE-SR4,4x 850nm VCSEL,MMF,pull tab,MPO,ROHS compliant, 3.3V, 3.5W, 0/70°C operation | 16 | Requires MPO Cable |
| Optics | Network | C/Ag-Leaf to C/Ag-Leaf | Finisar | 100GSR4 | XCVR,QSFP28,100m,100GBASE-SR4,4x 850nm VCSEL,MMF,pull tab,MPO,ROHS compliant, 3.3V, 3.5W, 0/70°C operation | 4 | Requires MPO Cable |
| Optics | Network | Ag-Leaf to Ag-Leaf | Finisar | 100GSR4 | XCVR,QSFP28,100m,100GBASE-SR4,4x 850nm VCSEL,MMF,pull tab,MPO,ROHS compliant, 3.3V, 3.5W, 0/70°C operation | 4 | Requires MPO Cable |
| Optics | Network | Leaf to Mgmt SX | Finisar | 40GSR4 | XCVR, QSFP, 4X 10.5GBPS, 40GBASE-SR4, 100/150m MMF, Limiting, XLPPI electrical interface, MPO, ROHS compliant, Gen4 | 3 | Requires MPO breakout cable |
| Optics | Network | Leaf to Storage | Finisar | 40GSR4 | XCVR, QSFP, 4X 10.5GBPS, 40GBASE-SR4, 100/150m MMF, Limiting, XLPPI electrical interface, MPO, ROHS compliant, Gen5 | 0 | Requires MPO breakout cable |
| Optics | Network | Network Uplinks | Finisar | 40GLR4 | XCVR, QSFP, 10km 40GBASE-LR4, 4x10G CWDM DFB, PIN, Limiting, XLPPI electrical interface, LC, ROHS, 3.3V, 3.5W, 0/70°C operation, Gen2 | 2 | Requires MPO breakout cable |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| Optics | Network | Network Uplinks | Finisar | 100GLR4 | XCVR,QSFP28,10km,100GBASE-LR4,4x 13XXnm LAN-WDM DFB,SMF,pull tab,LC,ROHS compliant, 3.3V, 3.5W, 0/70°C operation | 0 | |
| Optics | Network | R-Leaf to C/Ag-Leaf | Finisar | 100GSR4 | XCVR,QSFP28,100m,100GBASE-SR4,4x 850nm VCSEL,MMF,pull tab,MPO,ROHS compliant, 3.3V, 3.5W, 0/70°C operation | 16 | Requires MPO Cable |
| Optics | Network | R-Leaf to C/Ag-Leaf | Finisar | 100GCWDM4 | XCVR,QSFP28,2km,100GE CWDM4,4X 13XXnm CWDM DFB,SMF,pull tab,LC,ROHS compliant, 3.3V, 3.5W, 0/70°C operation | 0 | |
| Optics | Network | Server to C/Ag-Leaf | Finisar | 100GSR4 | XCVR,QSFP28,100m,100GBASE-SR4,4x 850nm VCSEL,MMF,pull tab,MPO,ROHS compliant, 3.3V, 3.5W, 0/70°C operation | 12 | |
| Optics | Servers | Server Uplinks | Finisar | 25GSR | 850nm Oxide VCSEL, 25GE, 25.78 Gb/s transceiver, RoHS compliant, multimode, pluggable SFP+ footprint, 3.3V, LC connector, bail, digital diagnostics, -40/85°C operation, 100m on OM4 fiber | 48 | |
| Optics | Network | Leaf Loopback | Finisar | 100GSR4 | XCVR,QSFP28,100m,100GBASE-SR4,4x 850nm VCSEL,MMF,pull tab,MPO,ROHS compliant, 3.3V, 3.5W, 0/70°C operation | 12 | |
| Optics | Network | Mgmt SX | Cisco | 10GESR | TLB_SFP-10G-SR. | 10 | MUST use Cisco 10GE optics |

## Software

| Category | Component | Function | Configuration | Qty |
|---|---|---|---|---|
| Software | Network | Leaf/Spine NOS | 1 license per leaf/spine | 6 |

## Some Recommendations for Hardware & its setup for Airship

Dell R740 has been used in some of the production grade deployments of airship. Some of the recommended hardware and software information is listed below:

### Firmware

| Dell R740xd | Version | ID | Comments |
|---|---|---|---|
| BIOS | 1.3.7 | MGGKF | |
| Storage Array Controller | 25.5.4.0006 | 8WK8N | PERC H730P Mini |
| iDRAC9 | 3.21.26.22 | FDMV1 | see note below |
| 14G Backplane Expander | 2.17 | NF3GR | |
| Solid State Disk SSDSC2KB480G7R 480Gb | SCV1 **DL58** | KRP53 | see note below |
| **Dell Network Adapters** | **Version** | **ID** | **Comments** |
| Intel X710 | 18.3.6 | GD64X | 10 Gb |
| Intel XXV710 | 18.3.6 | GD64X | 25 Gb |

### Notes

- iDRAC9 firmware 3.21.21.21 replaced version 3.15.17.15 3NCPY to address security vulnerabilities and to provide additional functionality. Virtual Console Plug-in changed from Java to HTML5.
- For details, please see: https://www.dell.com/support/home/us/en/04/drivers/driversdetails?driverId=387FW&osCode=WST14&productCode=poweredge-r740xd
- Firmware for Intel Youngsville 480Gb Solid State Disk (SSD) hard drive model SSDSC2KB480G7R must be at a revision level of at least DL57 to address rare case on power loss that can result in the drive failing upon power up. To remediate and for consistency, firmware should be upgraded to later firmware version DL58. Note that if upgrading firmware to DL58 from a version earlier than DL56, the server should be powered off for a full 2 minutes after the update due to a DRAM change. In all other cases, i.e. DL56 or DL57 to DL58, a system reboot will be sufficient.
- DL58 https://www.dell.com/support/home/us/en/04/drivers/driversdetails?driverid=krp53
- DL57 https://www.dell.com/support/home/us/en/04/drivers/driversdetails?driverid=wxjt4
- iDRAC9 firmware 3.21.26.22 replaced version 3.21.21.21 to address security vulnerabilities CVE-2018-15774 and CVE-2018-15774.

- TLS Protocol in iDRAC Settings / Services / Webserver changed from default of TLS 1.1 and Higher to TLS 1.2 Only to satisfy security scan requirement.

## BIOS Settings

Identify the NIC that will be used for PXE boot.

| NIC |
| --- |
| Integrated NIC 1 Port 4 (10 Gb) |

The following settings can be set manually by pressing F2 System Setup during boot or via the iDRAC by navigating to Configuration / BIOS Settings

**Note:** To allow the Dell BIOS GUI to work, you need to verify the Serial Communications are set to Auto otherwise you will get the BIOS screen in vt100 or text mode. Set under System BIOS, Serial Communication, Serial Communication set to Auto.

- System Setup / System BIOS Settings / Boot Settings / Boot mode **[BIOS]**
- System Setup / System BIOS Settings / Boot Settings / Boot Sequence Retry **[Disabled]**
- System Setup / System BIOS Settings / Integrated Devices **/** Internal USB Port **[Off]**
- System Setup / System BIOS Settings / Integrated Devices / SR-IOV Global Enable **[Enabled]**
- System Setup / System BIOS Settings / System Profile Settings /System Profile **[Performance]**
- System Setup / System BIOS Settings / System Security / AC Power Recovery **[On]**
- System Setup / System BIOS Settings / System Security / AC Power Recovery Delay **[Random]**
- System Setup / iDRAC Settings / Power Configuration / Redundancy Policy **[Input Power Redundant]**
- System Setup / iDRAC Settings / Power Configuration / Enable Hot Spare **[Disabled]**
- System Setup / iDRAC Settings / Network / Enable IPMI Over Lan **[Enabled]**
- System Setup / iDRAC Settings / Network / Common Settings / Static DNS Domain Name **[***DomainName***]** (ex: xxx.<subdomain>.<domain>.com)
- System Setup / iDRAC Settings / Lifecycle Controller **[Enabled]**

  The following settings can only be set manually by pressing F2 System Setup during boot:

- System Setup / Device Settings / ***NIC to be used for PXE*** / NIC Configuration / Legacy Boot Protocol **[PXE]**
- System Setup / Device Settings / ***All Remaining NICs*** / NIC Configuration / Legacy Boot Protocol **[none]**

For the network adapters in slots **2** and **7**

- System Setup / Device Settings / *Ports to utilize SR-IOV* / Device Level Configuration / Virtualization Mode **[SR-IOV]**

The following settings can only be set manually in the iDRAC User Interface:

- Select Configuration / Virtual Console / Max Sessions **[2]**
- Select Configuration / Virtual Console / Plug-in Type **[HTML5]**
- Select iDRAC Settings / Services / Web Server / SSL Encryption **[128-Bit or Higher]**
- Select iDRAC Settings / Services / Web Server / TLS Protocol **[TLS 1.2 Only]**
- Select iDRAC Settings / Services / SNMP Community Name / **null value** (blank)
- Select iDRAC Settings / Services / SNMP Agent / Enabled **[Disabled]**

### *RACADM Utility*

You can also use RACADM to validate some BIOS settings across multiple servers at once.

### *Installing RACADM*

Run the following to install racadm CLI (remember to export PROXY if your environment is behind one):

```
sudo echo "deb http://linux.dell.com/repo/community/ubuntu $(cat
/etc/apt/sources.list | grep '^deb' | head -1 | cut -d' ' -f3) openmanage" |
sudo tee -a /etc/apt/sources.list.d/linux.dell.com.sources.list
if [ -n "$PROXY" ]; then
export KEYSERVER_PROXY="--keyserver-options http-proxy=$PROXY"
sudo sh -c "echo 'Acquire::http::Proxy \"$PROXY\";
Acquire::https::Proxy \"$PROXY\";' > /etc/apt/apt.conf.d/20-proxy"
fi
sudo gpg --keyserver pool.sks-keyservers.net $KEYSERVER_PROXY --recv-key
1285491434D8786F
sudo gpg -a --export 1285491434D8786F | sudo apt-key add -
sudo apt update
sudo apt -y install srvadmin-all
```

### *Validating BIOS Settings*

The IPMI and BIOS settings may be validated for nodes in the environment with the following for each node according to its iDRAC IP address and iDRAC credentials (depending on racadm version and hardware vendor):

```
racadm -r $IDRAC_IP_ADDR -u $USER -p $PASSWORD --nocertwarn get
iDRAC.IPMILan.Enable|egrep '(iDRAC|Enable)'
racadm -r $IDRAC_IP_ADDR -u $USER -p $PASSWORD --nocertwarn get
iDRAC.Users.3|egrep '(IpmiLanPrivilege|User)'
racadm -r $IDRAC_IP_ADDR -u $USER -p $PASSWORD --nocertwarn get
BIOS.ProcSettings.ProcVirtualization
racadm -r $IDRAC_IP_ADDR -u $USER -p $PASSWORD --nocertwarn get
BIOS.BiosBootSettings.BootMode
racadm -r $IDRAC_IP_ADDR -u $USER -p $PASSWORD --nocertwarn get
BIOS.ProcSettings.LogicalProc
racadm -r $IDRAC_IP_ADDR -u $USER -p $PASSWORD --nocertwarn get
biOS.integratedDevices.sriovGlobalEnable
racadm -r $IDRAC_IP_ADDR -u $USER -p $PASSWORD --nocertwarn get
NIC.NICConfig.1.LegacyBootProto echo "FW/BIOS"
racadm -r $IDRAC_IP_ADDR -u $USER -p $PASSWORD --nocertwarn getsysinfo |
egrep '^(System Model|System BIOS Version|Firmware Version)' echo "PERC"
racadm -r $IDRAC_IP_ADDR -u $USER -p $PASSWORD --nocertwarn raid get
```

```
controllers -o | egrep '^( Name| FirmwareVersion)'
racadm -r $IDRAC_IP_ADDR -u $USER -p $PASSWORD --nocertwarn raid get vdisks -
o | egrep " Name| Size"
```

## Boot Sequence

For the Dell R740, Port 4 of the integrated card (Integrated NIC 1 Port 4) is used for PXE and should be first in the boot sequence. Note that if a different type of server is used, the PXE interface may change, but in all cases the PXE interface should be the first on boot sequence.

## RAID Configuration

- 2 x 480GB SSD – RAID 1: Operating System, Ceph Mon Ctrl Cluster, Ceph Mon Tenant Cluster
- 1 x 480GB SSD – RAID 0: Ceph Journal/Meta
- 1 x 480GB SSD – RAID 0: Ceph Journal/Meta
- 1 x 2.4 TB HDD – RAID 0: Ceph OSD
- 1 x 2.4 TB HDD – RAID 0: Ceph OSD
- 1 x 2.4 TB HDD – RAID 0: Ceph OSD
- 1 x 2.4 TB HDD – RAID 0: Ceph OSD
- 1 x 2.4 TB HDD – RAID 0: Ceph OSD
- 1 x 2.4 TB HDD – RAID 0: Ceph OSD

*Control Host Details:*

| Name | Layout | Size | Media Type | Read Policy | Write Policy | Stripe Size |
|---|---|---|---|---|---|---|
| OS, Ceph Mon Ctrl Cluster, Ceph Mon Tenant Cluster | RAID-1 | 446 GB | SSD | Read Ahead | Write Through | 64K |
| Ceph Journal | RAID-0 | 446 GB | SSD | Read Ahead | Write Back | 64K |
| Ceph Journal | RAID-0 | 446 GB | SSD | Read Ahead | Write Back | 64K |
| Ceph OSD | RAID-0 | 2235 GB | HDD | Read Ahead | Write Back | 64K |
| Ceph OSD | RAID-0 | 2235 GB | HDD | Read Ahead | Write Back | 64K |
| Ceph OSD | RAID-0 | 2235 GB | HDD | Read Ahead | Write Back | 64K |
| Ceph OSD | RAID-0 | 2235 GB | HDD | Read Ahead | Write Back | 64K |
| Ceph OSD | RAID-0 | 2235 GB | HDD | Read Ahead | Write Back | 64K |

| Name | Layout | Size | Media Type | Read Policy | Write Policy | Stripe Size |
|---|---|---|---|---|---|---|
| Ceph OSD | RAID-0 | 2235 GB | HDD | Read Ahead | Write Back | 64K |

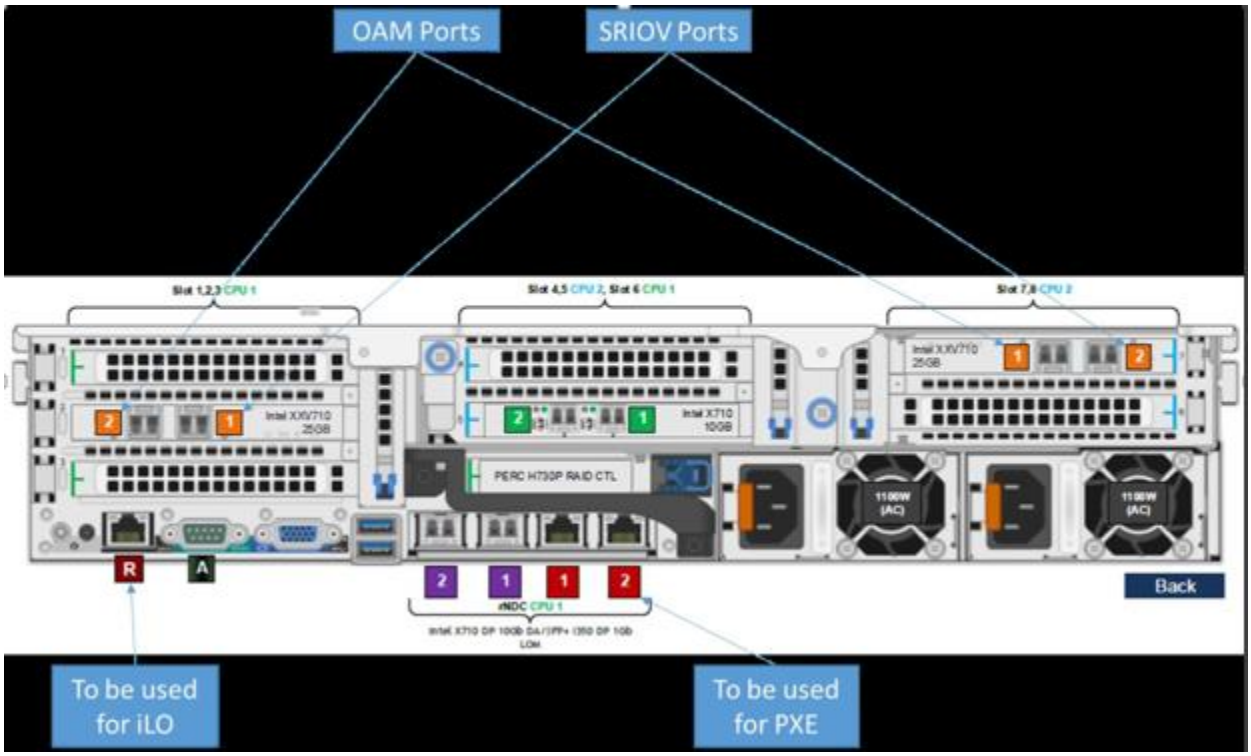
*Compute Hosts :*

- 2 x 480GB SSD – RAID 1: Operating System
- 1 x 480GB SSD – RAID 0: Ceph Journal/Meta
- 1 x 480GB SSD – RAID 0: Ceph Journal/Meta
- 4 x 2.4 TB HDD – RAID 10: OpenStack Ephemeral
- 1 x 2.4 TB HDD – RAID 0: Ceph OSD
- 1 x 2.4 TB HDD – RAID 0: Ceph OSD

*Compute Host Details:*

| Name | Layout | Size | Media Type | Read Policy | Write Policy | Stripe Size |
|---|---|---|---|---|---|---|
| OS | RAID-1 | 446 GB | SSD | Read Ahead | Write Through | 64K |
| Ceph Journal | RAID-0 | 446 GB | SSD | Read Ahead | Write Back | 64K |
| Ceph Journal | RAID-0 | 446 GB | SSD | Read Ahead | Write Back | 64K |
| OpenStack Ephemeral | RAID-10 | 4470 GB | HDD | Read Ahead | Write Back | 64K |
| Ceph OSD | RAID-0 | 2235 GB | HDD | Read Ahead | Write Back | 64K |
| Ceph OSD | RAID-0 | 2235 GB | HDD | Read Ahead | Write Back | 64K |

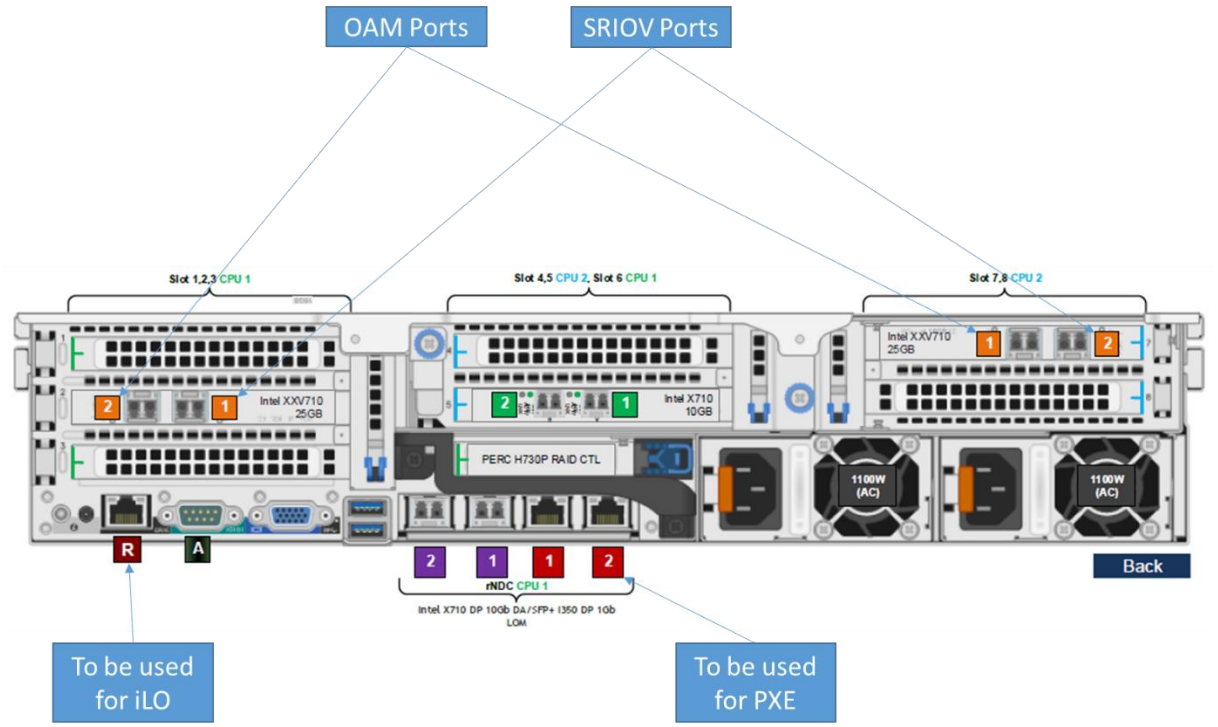
*Server Rear View*



## Host Profiles

Host profile is a personality given to any node in airship framework and is defined for all Control/Compute nodes. Additionally, it defines how the underlay c-leaf switch ports shall be configured for connected host.

There are some production deployments done using Dell Purley R740XD 2RU Server which will be considered in building a host profile. The diagram below shows the ports on Dell R740XD server to be used for various workloads:

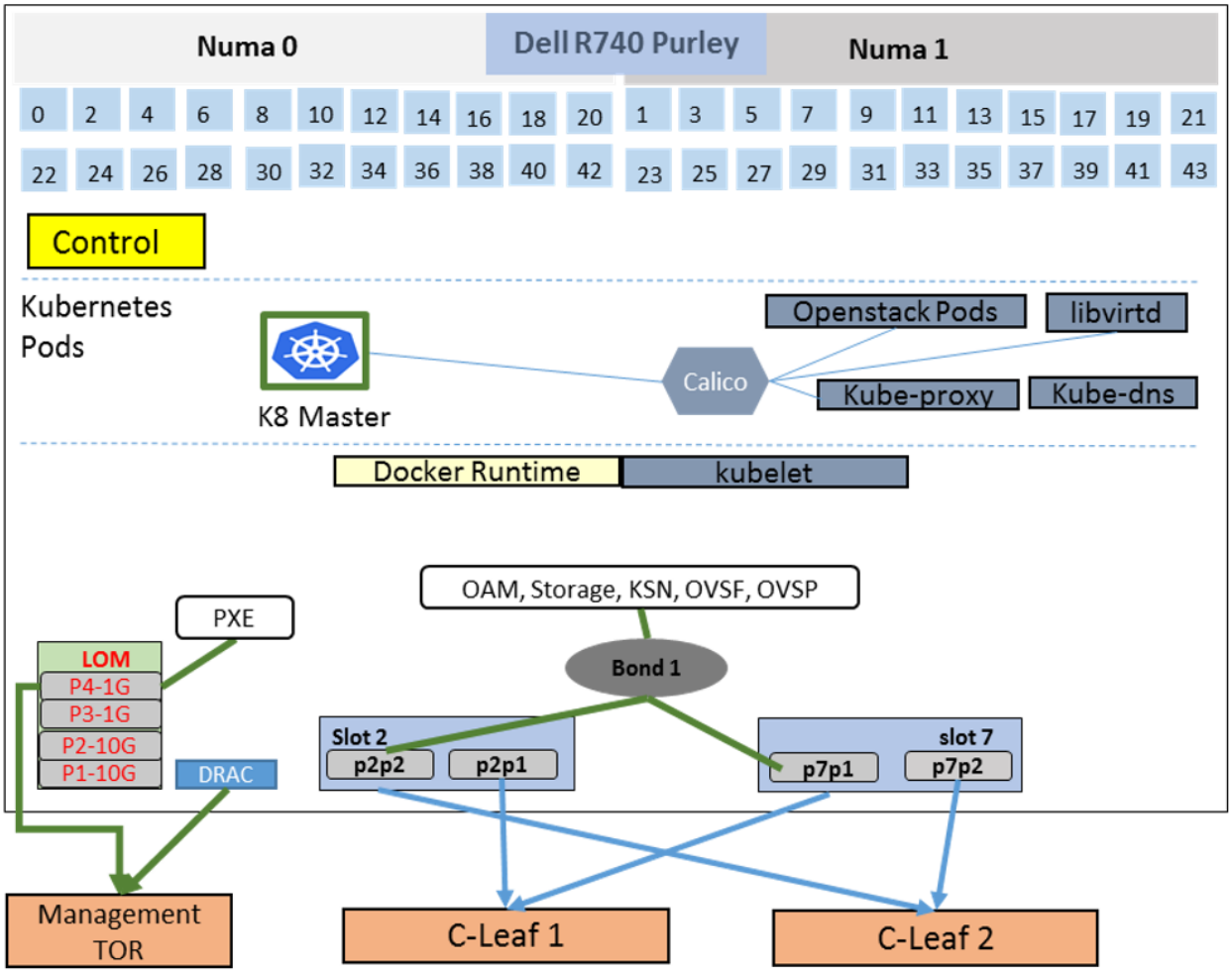



Two points to consider from the diagram above are:

1) NIC on slot 2 and slot 7 to be used for 25G cards connection
2) The NIC card in slot 2 is flipped

### Airship Control Plane Profile-CP

This profile is deployed specifically to host the control plane for the cloud. This includes the under-cloud platform and the OpenStack control plane.




| | | |
|---|---|---|
| Host Profile Title | airship-cp | |
| Hardware Profile | airship-Dell-r740-purley | |
| Compute | Server | Dell R740 [Purley] – 2RU |
| Network | Total Number of Data NIC ports | 4 |
| | Data NIC port capacity | 25 Gbps |
| | Numa 0 Data NICs | [Dell R740] p2p1, p2p2 |
| | Numa 0 Data NIC VF Promiscuous Mode | Disabled |
| | Numa 1 Data NICs | [Dell R740] p7p1, p7p2 |
| | Numa 1 Data NIC VF Promiscuous Mode | Disabled |
| | PXE (1G LOM) | Host PXEtraffic to go on 1G Broadcom LOM port connected to Management Switch. |

| | | |
|---|---|---|
| | Bond1 NIC Ports | p2p2 + p7p1 (2*25Gbps) |
| | Bond1 VLANs | Bond1 to carry Storage, Infra OAM, Kubernetes Service Network (KSN), OVSF (OpenStack Floating IP), OVSP (OpenStack Private) |
| | iDRAC port | “R” port as shown in figure #49 will be used for access to terminal console on the physical server. |
| Storage | Local storage | Disk: 4 x 480GB SSD<br><br>Disk 6 x 2.4 TB |
| | | |

*Notes*

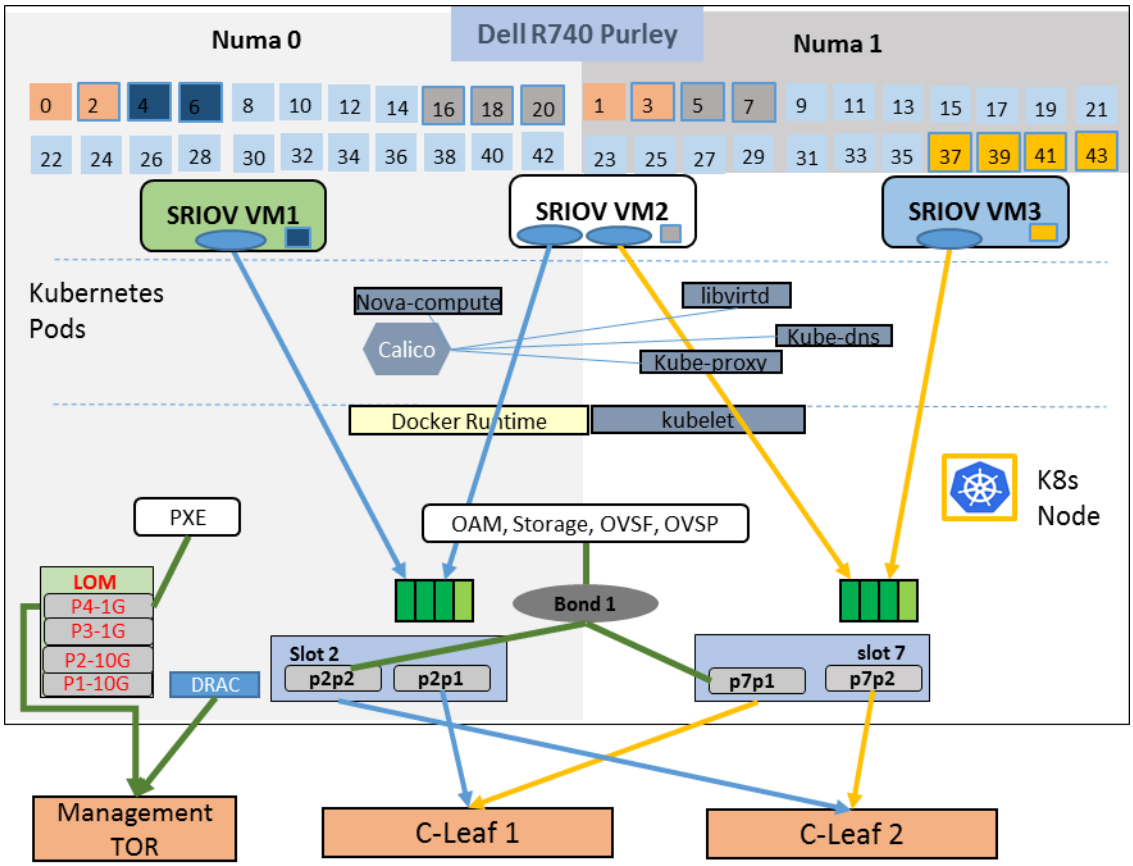
- First port from each dual port NIC card is attached to C-Leaf1
- Second port from each dual port NIC card is attached to C-Leaf2
- Bond1 (lacp-bonded, Mode=4) shall be formed on host server with the VLANs shown above

  Bond setting for LACP mode should be a default to following values: Validate these values from “cat /proc/net/binding/bond1”

  - Transmit Hash Policy: layer2 (0)
  - MII Polling Interval (ms): 100
  - Up Delay (ms): 300
- Down Delay (ms): 300All Kubernetes and Openstack services shall be containerized and spread across control servers
- 1G (marked as RED-2/P4)  Port on Onboard card shall be connected to Management TOR switch for PXE connectivity
- Refer link below for link connectivity detail:

### Airship Compute Profile – airship-P1

This host profile can be used for workloads that require low latency like 5G workloads. This profile works only with SR-IOV and is not for OVS-DPDK.




| | | |
|---|---|---|
| Host Profile Title | airship-p1 | |
| Hardware Profile | airship-Dell-r740-purley | |
| Compute | Server | Dell R740 [Purley] – 2RU |
| | CPU allocation ratio | 1:1 |
| | CPU pinning | Yes, enabled by default |
| | CPU Isolation | “isolcpus=<list>” on the kernel line of grub parameter |
| | CPU dedicated for Host | [Dell R740] 0, 44, 1, 45, 2, 46, 3, 47 |
| | CPU available for Guest VMs | [Dell R740] 4-43, 48-87 |
| | CPU HyperThreading [HT] | Enabled |
| | Hugepage support | Enabled for all tenants / 320 – 1G hugepages [Distributed equally on 2 NUMA boundaries – 160 on each NUMA] |
| | Max vCPUs available for tenant per NUMA | [Dell R740] 40 (w/ HT enabled) |

| | | |
|---|---|---|
| | Max vCPUs available for tenant per compute (incl. cores from both NUMA) | [Dell R740] 80 (w/ HT enabled) |
| Network | Number of Data NIC ports | 4 |
| | Data NIC port capacity | 25 Gbps |
| | Numa 0 Data NICs | [Dell R740] p2p1, p2p2 |
| | Numa 0 Data NIC VF Promiscuous Mode | Disabled |
| | Numa 1 Data NICs | [Dell R740] p7p1, p7p2 |
| | Numa 1 Data NIC VF Promiscuous Mode | Disabled |
| | PXE (1G LOM) | Host PXE/Admin traffic to go on 1G Broadcom LOM port connected to Management TOR. |
| | Bond1 NIC Ports | p2p2 + p7p1 (2*25Gbps) |
| | Bond1 VLANs | Bond1 to carry Storage, Infra OAM, Kubernetes Service Network (KSN), OVSF (OpenStack Floating IP), OVSP (OpenStack Private) |
| Storage | Local storage | Disk: 4 x 480GB SSD<br><br>Disk 6 x 2.4 TB |

*Notes:*

- HyperThreading (HT) enabled on all servers by default
- Cores 0, 1, 2, 3 and their HT cores 44, 45, 46 and 47 resp to be dedicated for Host and hypervisor needs. All other vCPUs shall be available for tenant usage.
- First port from each dual port NIC card is attached to C-Leaf1
- Second port from each dual port NIC card is attached to C-Leaf2
- Bond1 (lacp bonded) shall be formed with p2p2 and p7p1
- SRIOV shall be configured on ports p2p1 and p7p2
- 1G (marked as RED-2/P4) Port on Onboard card shall be connected to Management TOR switch for PXE connectivity
- Basic services like nova-compute, libvirt shall be containerized on compute node
- 1G Port on Onboard card shall be connected to Management TOR switch for PXE connectivity
- Differentiation between the SRIOV VMs running on same host profile are based on tenant flavor definitions, The small colored boxes in SRIOV VMs correlates with the core mapping for VM in respective NUMA

## VLAN & MTUs

Below mentioned are some recommended VLANs which need to be created.

| Function | Interface | VLAN | MTU |
|---|---|---|---|
| IPMI | | 1023 | 1500 |
| PXE | | 43 | 1500 |
| Physical interfaces to be used for bond1 | p2p2, p7p1 | | 9214 |
| Bond1 interface | Bond1 | | 9214 |
| OPS/Host OAM | Bond1.41 | 41 | 9000 |
| Storage | Bond1.42 | 42 | 9000 |
| Kubernetes | Bond1.44 | 44 | 9000 |
| OVS-P (OVS Tenant private network – private IP assigned to Openstack VMs) | Bond1.45 | 45 | 9150 |
| OVS-F (OVS Floating IP – Public IP assigned to Openstack VMs) | Bond1.{ovsf} | 50 | 9100 |
| SRIOV Interfaces | p2p1, p7p2 | | 1500 |

It has been confirmed in some production systems that though the MTU for SRIOV physical ports are configured with 1500 bytes, the Virtual Functions assigned to VMs can still override this setting with MTU set to jumbo frames. Intel Fortville i40e driver supports the VF MTU override over PF MTU.

*Note :*

- PXE interface on Control plane and compute nodes are untagged ports. They will not have any VLAN ID configured in host configuration.
- There is no bonding on SRIOV interfaces, neither is the fabric configured for any bonding configuration for these 2 interfaces. VMs may still use bond mode 1 (active-backup) for the VFs allocated to it from 2 different physical NICs.

### Host Aggregates and Placement

Airship cluster can be deployed with a single host aggregate. All compute servers shall be added to this host aggregate and VNF can follow the static placement policy to place the VMs on respective servers.

Metadata to be used for Host Aggregate creation: “p1=true”

Naming convention:

*{region}-{profile}-{HA#}*

For each of these components

- A dedicated set of compute nodes will be pre-allocated
- Initial deployment plan
    - {region}-p1-ha01 – All 40 compute nodes

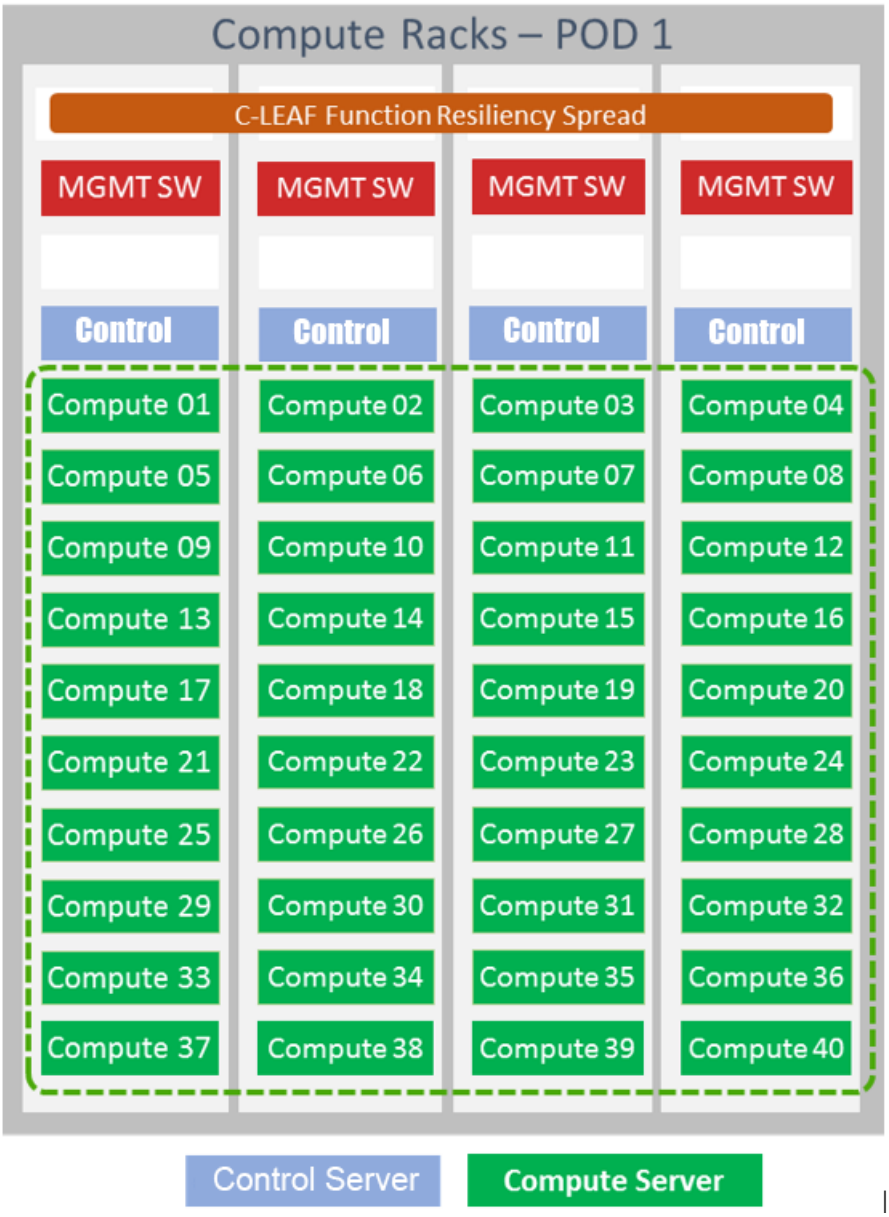
### Availability Zone

Availability Zones are the end-user visible logical abstraction for partitioning a cloud without knowing the physical infrastructure.

For airship, the suggestion is to have an AZ defined per Rack.

Naming conventions to be used for AZ:

{region}-{hypervisor}-{AZ#} e.g. testlab-kvm-az01




## Grub Parameters required per Host profile:

Some kernel configurations need to be applied as part of operating system grub boot options. Drydock can read these configurations from Host profile yaml attached to the server.

| Data path for Tenant traffic | Series Site | Grub Options |
|---|---|---|
| SRIOV ports | P1 | console=ttyS1,115200n8<br>intel_iommu=on<br>iommu=pt<br>hugepagesz=1G<br>hugepages=320 |
| Control Nodes | CP | console=ttyS1,115200n8 |

### Flavors

Here is a sample flavor which can now be used to deploy workloads.

| | Value | Flavor Metadata / Extra-specs |
|---|---|---|
| Flavor Name | p1 | “aggregate_instance_extra_specs:p1”: “true”,<br>“hw:mem_page_size”: ”large” [large here means 1G huge pages],<br>“hw:cpu_policy”: “dedicated” |
| # of vcpus | 40 | vcpus: "40" |
| RAM | 128 | ram: "131072", |
| Disk | 50 | disk: "50", |
| Ephemeral | 0 | ephemeral: "0", |
| NUMA boundaries [spread on 2 NUMA] | n0 | hw:numa_nodes: "2" |
| Instance Metadata | i2 | hw:pci_numa_affinity_policy: "dedicated" |

## SR-IOV Setup & Configuration

The SR-IOV specification defines a standardized mechanism to virtualize PCIe devices. This mechanism can virtualize a single PCIe Ethernet controller to appear as multiple PCIe devices. Each device can be directly assigned to an instance, bypassing the hypervisor and virtual switch layer. As a result, users are able to achieve low latency and near-line wire speed.

The following terms are used in this section:

| Term | Definition |
|---|---|
| PF | Physical Function. The physical Ethernet controller that supports SR-IOV. |
| VF | Virtual Function. The virtual PCIe device created from a physical Ethernet controller. |

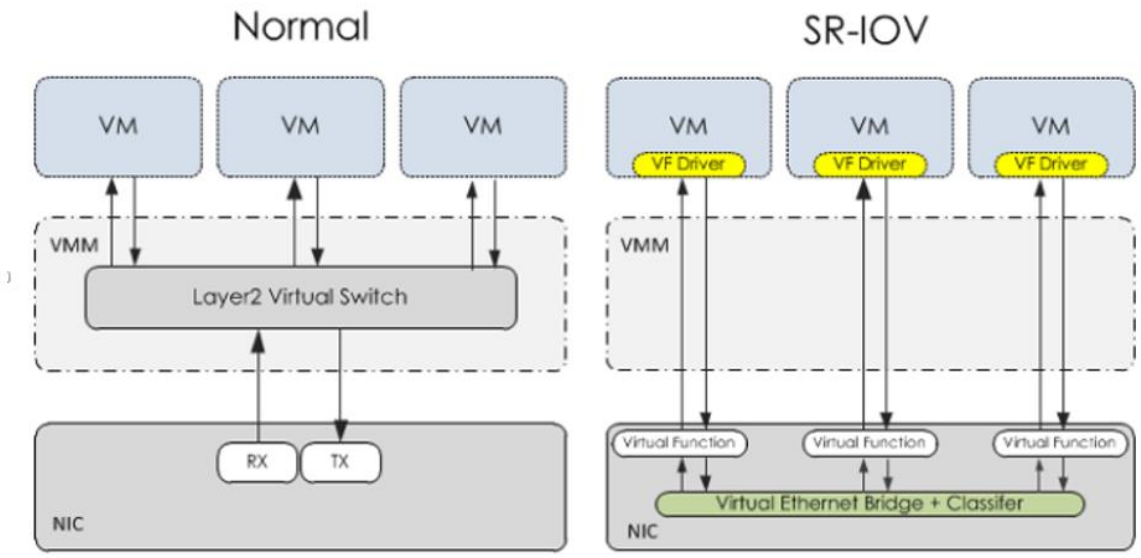



Figure: SR-IOV Architecture Comparison

The SR-IOV port is associated with a virtual function (VF) that is supported by the vNIC adaptor. In addition, the SR-IOV port may be extended to an upstream physical switch (IEEE 802.1br), and in such case, the port's configuration takes place in that switch.

### SR-IOV Agent

The SR-IOV agent allows you to set the admin state of ports, configure port security (enable and disable spoof checking), and configure QoS rate limiting. You must include the SR-IOV agent on each compute node using SR-IOV ports.

To Support SR-IOV Interfaces the following steps are required:

1. Create Virtual Functions (on Each Compute)
2. Whitelist PCI device in nova-compute (on each Compute)

3. Configure Neutron-Server (Controller)
4. Configure nova-scheduler (Controller)
5. Enable neutron SR-IOV Agent (Compute)

### Create Virtual Functions (Compute)

Ensure that the compute nodes are capable of PCI passthrough and SR-IOV. The hardware must provide VT-d an SR-IOV capabilities. SRIOV NICs shall be configured for total count of 64 VFs.

If the system supports VT-d, there should be DMAR related message exist in the dmesg log.

```
root@mdw2r01o001:~/sites/mdw2a# dmesg | grep -i dmar | more
[    0.000000] ACPI: DMAR 0x000000007B7E6000 00038A (v01 INTEL  INTEL ID
00000001 ?    00000001)
[    0.000000] DMAR: IOMMU enabled
[    0.102653] DMAR: Host address width 46
[    0.102655] DMAR: DRHD base: 0x000000fbffc000 flags: 0x0
[    0.102662] DMAR: dmar0: reg_base_addr fbffc000 ver 1:0 cap
8d2078c106f0466 ecap f020de
[    0.102666] DMAR: DRHD base: 0x000000c7ffc000 flags: 0x1
[    0.102671] DMAR: dmar1: reg_base_addr c7ffc000 ver 1:0 cap
8d2078c106f0466 ecap f020de
[    0.102674] DMAR: RMRR base: 0x00000079170000 end: 0x00000079172fff
```

If the system supports SR-IOV, there should be SR-IOV capability section for each NIC PF, and the total VFs should be non-zero.

```
root@mdw2r01o001:~/sites/mdw2a# lspci -vvv | grep -i "initial vf"

Initial VFs: 64, Total VFs: 64, Number of VFs: 64, Function Dependency Link:
00
Initial VFs: 64, Total VFs: 64, Number of VFs: 0, Function Dependency Link:
01
Initial VFs: 64, Total VFs: 64, Number of VFs: 0, Function Dependency Link:
00
Initial VFs: 64, Total VFs: 64, Number of VFs: 64, Function Dependency Link:
01
```

Enable iommu by modifying the “/etc/default/grub” file. Input–output memory management unit (IOMMU) is a memory management unit (MMU) that connects a direct-memory-access–capable (DMA-capable) I/O bus to the main memory. Like a traditional MMU, which translates CPU-visible virtual addresses to physical addresses, the IOMMU maps device-visible virtual addresses (also called device addresses or I/O addresses in this context) to physical addresses. Some units also provide memory protection from faulty or malicious devices. Add the following values to the end of ““GRUB_CMDLINE_LINUX_DEFAULT” parameter.

```
intel_iommu=on iommu=pt
```

Next, update grub and reboot the server to get the changes to take effect.

```
root@mdw2r01o001: update-grub
root@mdw2r01o001: reboot
```

After the server comes up, run the following command to make sure that iommu is enabled.

```
root@mdw2r01o001:~/sites/mdw2a# dmesg | grep -i "iommu.*enabled"
[    0.000000] DMAR: IOMMU enabled
```

Enable the number of virtual functions required on the SR-IOV interface.

```
echo '0' > /sys/class/net/ens2f0/device/sriov_numvfs
echo '0' > /sys/class/net/ens7f1/device/sriov_numvfs

echo '64' > /sys/class/net/ens2f0/device/sriov_numvfs
echo '64' > /sys/class/net/ens7f1/device/sriov_numvfs

ip link set ens2f0 up
ip link set ens7f1 up

if ! grep "sriov_numvfs" /etc/rc.local > /dev/null
then
  sed -i 's/exit 0//' /etc/rc.local
  echo "echo '64' > /sys/class/net/ens2f0/device/sriov_numvfs" >>
/etc/rc.local
  echo "echo '64' > /sys/class/net/ens7f1/device/sriov_numvfs" >>
/etc/rc.local
  echo "exit 0" >> /etc/rc.local
fi
```

Run the following command to make sure that VFs are created

```
root@mdw2r01o001:~/sites/mdw2a# lspci | grep -i virtual
01:00.4 USB controller: Hewlett-Packard Company Integrated Lights-Out
Standard Virtual USB Controller (rev 03)
08:10.0 Ethernet controller: Intel Corporation 82599 Ethernet Controller
Virtual Function (rev 01)
08:10.1 Ethernet controller: Intel Corporation 82599 Ethernet Controller
Virtual Function (rev 01)
08:10.2 Ethernet controller: Intel Corporation 82599 Ethernet Controller
Virtual Function (rev 01)
08:10.3 Ethernet controller: Intel Corporation 82599 Ethernet Controller
Virtual Function (rev 01)
```

### Whitelist PCI devices in nova-compute (Compute)

Nova-compute needs to know which PCI devices are allowed to be passed through to the VMs. Also, for SR-IOV PCI devices it needs to know to which physical network the VF belongs. This is done through the pci_passthrough_whitelist parameter under the default section in /etc/nova/nova.conf. Edit the nova.conf file:

```
[default]
pci_passthrough_whitelist = { "address": "0000:d8:0a.0", "physical_network":
"physnet2"}
```

The pci_passthrough_whitelist needs to be filled with all the VFs from both the ports which can be allocated to VMs.

### Configure neutron-server (Controller)

Add sriovnicswitch as mechanism driver. Edit the ml2_conf.ini file on each controller:

```
mechanism_drivers = openvswitch,sriovnicswitch
```

Add the ml2_conf_sriov.ini file as parameter to the neutron-server service. Edit the appropriate initialization script to configure the neutron-server service to load the SR-IOV configuration file:

```
--config-file /etc/neutron/neutron.conf
--config-file /etc/neutron/plugin.ini
--config-file /etc/neutron/plugins/ml2/ml2_conf_.ini
--config-file /etc/neutron/plugins/ml2/sriov_agent.ini
```

Restart the neutron-server service.

### Configure nova-scheduler (Controller)

On every controller node running the nova-scheduler service, add PciPassthroughFilter to enabled_filters to enable PciPassthroughFilterAn example can be seen blow.

```
[filter_scheduler]
enabled_filters = RetryFilter, AvailabilityZoneFilter, RamFilter,
ComputeFilter, ComputeCapabilitiesFilter, ImagePropertiesFilter,
ServerGroupAntiAffinityFilter, ServerGroupAffinityFilter,
PciPassthroughFilter, NUMATopologyFilter, DifferentHostFilter,
SameHostFilter, AggregateInstanceExtraSpecsFilter, AggregateCoreFilter,
AggregateRamFilter, AggregateMultiTenancyIsolation, JsonFilter, IoOpsFilter,
AggregateDiskFilter, AllHostsFilter ,IsolatedHostsFilter,
AggregateImagePropertiesIsolation, NumInstancesFilter,
AggregateNumInstancesFilter, MetricsFilter, SimpleCIDRAffinityFilter,
AggregateTypeAffinityFilter
```

Restart the nova-scheduler service.

### Enable neutron sriov-agent (Compute)

Install the SR-IOV agent.

Edit the sriov_agent.ini file on each compute node. For example:

```
[securitygroup]
firewall_driver = neutron.agent.firewall.NoopFirewallDriver

[sriov_nic]
physical_device_mappings = physnet1: ens2f0,physnet2: ens7f1 exclude_devices
=
```

*Note*

The physical_device_mappings parameter is not limited to be a 1-1 mapping between physical networks and NICs. This also enables to map the same physical network to more than one NIC. For example, if physnet2 is connected to eth3 and eth4, then physnet2:eth3,physnet2:eth4 is a valid option.

The exclude_devices parameter is empty, therefore, all the VFs associated with eth3 may be configured by the agent. To exclude specific VFs, add them to the exclude_devices parameter as follows:

```
exclude_devices =
eth1:0000:07:00.2;0000:07:00.3,eth2:0000:05:00.1;0000:05:00.2
```

Ensure the neutron sriov-agent runs successfully:

```
# neutron-sriov-nic-agent \
  --config-file /etc/neutron/neutron.conf \
  --config-file /etc/neutron/plugins/ml2/sriov_agent.ini
```

Enable the neutron sriov-agent service.

If installing from source, you must configure a daemon file for the init system manually.

### Launching instances with SR-IOV ports

Once configuration is complete, you can launch instances with SR-IOV ports.

Get the id of the network where you want the SR-IOV port to be created:

```
$ net_id=`neutron net-show net04 | grep "\ id\ " | awk '{ print $4 }'`
```

Create the SR-IOV port. vnic_type=direct is used here, but other options include normal, direct-physical, and macvtap:

```
$ port_id=`neutron port-create $net_id --name sriov_port --binding:vnic_type
direct | grep "\ id\ " | awk '{ print $4 }'`
```

Create the instance. Specify the SR-IOV port created in step two for the NIC:

```
$ openstack server create --flavor m1.large --image ubuntu_14.04 --nic port-
id=$port_id test-sriov
```

## Preparing the bare metal cluster for airship

Before starting to prepare hardware for airship deployment, it is important to know different types of machines / nodes used to deploy airship on a bare metal or multi-node cluster:

- **Build node:** This refers to the environment where deployment engineer builds configuration documents for airship environment (e.g. his laptop)
- **Genesis node:** The "genesis" or "seed node" refers to the first physical server used to get a new deployment off the ground. It is the first node built in a new deployment of airship environment.
- **Control / Controller nodes:** The nodes that make up the control plane. These nodes would host Kubernetes & Openstack control plane components. (Note - Genesis node will be one of the controller nodes)
- **Compute nodes / Worker Nodes:** The nodes that make up the data plane. The Openstack workloads are deployed on these nodes.

Airship configurations are kept in a collection of YAML files. There is a 'golden AIRSHIP reference site' YAML documents, which can be cloned and used as a starting point for authoring a new site's configuration documents.

## Minimum Hardware Requirements for Airship

The hardware used in airship deployment must adhere to the following minimum resource 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* |

It should be noted that these guidelines are workload dependent and derived from the host profiles used for the machines.

## RAID Configuration

Raid is required to track the changes to the disk layout; as it may impact other parts of the deployment procedure and the reference/golden manifest files. Data copy in raid helps to ensure that the documentation is in-sync with these manifests.

For servers that are in the control nodes disk (including Genesis):

- Two disks – RAID 1 configured for Host operating System
- Two disks – RAID 0 configured for CEPH journal and metadata (SSD preferred)
- Remaining disks as JBOD for control plane CEPH cluster

For servers that are in the tenant data plane (compute nodes):

- Two-disk - RAID-1 configured for operating system

- Two disks – RAID 0 configured for CEPH journal and metadata (SSD preferred)
- Two disks as JBOD for tenant CEPH
- Remaining disks need to be configured according to the host profile target for each given server (e.g., RAID-10) and to be used for Openstack ephemeral

### IPMI and BIOS

Make sure that the following configurations are enabled:

- Virtualization enabled in BIOS
- IPMI enabled in server BIOS (e.g., IPMI over LAN option enabled)
- IPMI IPs assigned and routed to the environment you will deploy airship in.
- Set PXE as first boot device and ensure the correct NIC is selected for PXE
- *Tip*: 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

If you run into problems that lead you to suspect wrong IPMI or BIOS settings, you may validate some of those settings with `racadm` across multiple servers.

### Network Configurations

- Ensure the physical server can be successfully PXE booted and the network topology and bonding settings allow it (dedicated PXE interface on untagged/native VLAN in this example)
- The following VLANs (segmented networks) are required to be created on all the nodes.
  - Management network(s) (k8s control channel, aka OAM network) - one shared network across all racks. This network is expected to be routed such that it can reach the internet, the out-of-band network, and resources like image repository (e.g. JFrog Artifactory) required for deployment.
  - Calico network - one shared network across all racks
  - Storage network(s) - one shared network across all racks
  - Overlay network(s) - one shared network across all racks
  - Public networks(s) - one or more shared networks across all racks

#### *Setting up Calico BGP peers*

BGP peers can be setup in the manifest file located at *'site / <site-name> / software / charts / kubernetes / container-networking /calico.yaml'*. In case commercial version of calico (Tigera Secure Enterprise Edition) is being used in the airship deployment, it would be tigera-secure-ee.yaml instead of calico.yaml. Below mentioned is an example of BGP peers setup in this file.

```
---
schema: armada/Chart/v1
metadata:
  schema: metadata/Document/v1
  replacement: true
  name: tigera-secure-ee
  layeringDefinition:
```

```
    abstract: false
    layer: site
    parentSelector:
      name: tigera-secure-ee
      layer: type
    actions:
      - method: replace
        path: .values.networking.bgp.ipv4.peers
      - method: merge
        path: .
      - method: delete
        path: .values.networking.policy
  substitutions:
    - src:
        schema: pegleg/CommonAddresses/v1
        name: common-addresses
        path: .calico.bgp.ipv4.public_service_cidr
      dest:
        path: .values.networking.bgp.ipv4.additional_cidrs
  storagePolicy: cleartext
data:
  values:
    networking:
      settings:
        mesh: "off"
        ippool:
          ipip:
            enabled: "false"
      bgp:
        asnumber: 64171
        ipv4:
          peers:
            - apiVersion: projectcalico.org/v3
              kind: BGPPeer
              metadata:
                name: labname-peer-0
              spec:
                asNumber: 64188
                peerIP: 172.20.20.1
                node: lab-node1
            - apiVersion: projectcalico.org/v3
              kind: BGPPeer
              metadata:
                name: labname-peer-1
              spec:
                asNumber: 64188
                peerIP: 172.20.20.2
                node: lab-node1
```

Similarly the BGP advertisement of airship elements can be done in the common-addresses.yaml at:

*site / <site-name>/ networks / common-addresses.yaml.* Here is an example.

```
---
schema: pegleg/CommonAddresses/v1
metadata:
  schema: metadata/Document/v1
  replacement: true
  name: common-addresses
  layeringDefinition:
    abstract: false
    layer: site
    parentSelector:
      name: common-addresses-global
    actions:
      - method: merge
        path: .
  storagePolicy: cleartext
data:
  calico:
    ip_autodetection_method: interface=bond1.2006
    etcd:
      service_ip: 10.10.10.136
    ip_rule:
      gateway: 172.10.10.3
      overlap_cidr: 10.98.0.0/15
    bgp:
      ipv4:
        public_service_cidr: 138.25.25.120/29
        ingress_vip: 138.25.25.121/32
        maas_vip: 172.10.10.135/32
        peers:
          - 0.0.0.0
          - 0.0.0.0
```

## Installing airship to a bare metal cluster

In order to install airship on a bare metal multi node cluster, it is needed to put together site documents according to your specific environment and generate the initial Promenade bundle for site deployment.

### Preparing deployment documents

In its current form, pegleg provides an organized structure for YAML elements. This allows 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-site-manifest/site' folder and copy the 'airship-seaworthy' site as follows:

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

Remove 'airship-seaworthy' specific certificates.

```
rm -rf airship-site-manifest/site/airship-
seaworthy/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 the parameters that need to change when authoring a new site.

**Important:** 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.

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

- site/$NEW_SITE/networks/physical/networks.yaml
- site/$NEW_SITE/baremetal/nodes.yaml
- site/$NEW_SITE/networks/common-addresses.yaml
- site/$NEW_SITE/pki/pki-catalog.yaml
- All other site files

## Configuring Control Plane CEPH cluster

**Assumptions:**

- 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.)
- Ceph disk mapping, disk layout, journal and OSD setup is the same across Ceph nodes, with only their role differing. Considering there are 4 nodes to be used for Control plane. Out of the 4 control plane nodes, it is expected 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)
- If doing a fresh install, disk should be unlabeled or not labeled from a previous Ceph install, so that Ceph chart will not fail disk initialization

Ceph environment parameters for the control plane can be located at:

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

Some configurations required here are:

- `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

This document covers two Ceph journal deployment paradigms:

- Servers with SSD/HDD mix (disregarding operating system disks)
- 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. The whole block device can also be used, however, that block device can't be passed to the Ceph chart like it can be 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
              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 like 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
```

### Generate passphrases

Generate the passphrases used in your environment as follows:

```
~/ airship-site-manifest/tools/generate_passphrases.sh "$NEW_SITE"
```

Wait to commit these changes to source control until certificates have been generated below.

### 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 lint \
  -p airship-site-manifest
```

**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 \
  -p airship-site-manifest \
  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:

```
mkdir -p ~/${NEW_SITE}_collected
sudo airship-pegleg/tools/pegleg.sh site \
  -p airship-site-manifest \
  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 \
  -p airship-site-manifest \
  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:

```
cd ~
 git clone https://opendev.org/airship/promenade.git
```

Refer to the `data/charts/ucp/promenade/reference` field in `~/airship-site-manifest/global/v4.0/software/config/versions.yaml`. If this is a pinned reference (i.e., any reference that's not `master`), then you should check-out 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-site-manifest/global/v4.0/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 `foo.example.com/promenade@sha256:d30397f...` in the versions file, then export the previously mentioned environment variable like so:

```
export IMAGE_PROMENADE=foo.example.com/promenade@sha256:d30397f...
```

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

```
mkdir ~/${NEW_SITE}_bundle
sudo ~/airship-promenade/tools/simple-deployment.sh
~/${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 repo. Ex:

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

## Genesis Node 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 using the Ubuntu ISO.

Ensure to select the following:

- UTC time zone
- Hostname that matches the Genesis hostname given in `/data/genesis/hostname` in `~/airship-site-manifest/site/$NEW_SITE/networks/common-addresses.yaml`.
- At the `Partition Disks` screen, select `Manual` so that you can setup the same disk partitioning scheme used on the other control plane nodes that will be deployed by MaaS. Select the first logical device that corresponds to one of the RAID-1 arrays already setup in the hardware controller. On this device, setup partitions matching those defined for the `bootdisk` in your control plane host profile found in `/airship-site-manifest/site/$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 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 `~/airship-site-manifest/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 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/v4.0/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, simple-stream image cache should be looked at that will be used by MaaS to deploy nodes with. Locate the `data/images/ucp/maas/maas_cache` key in within `~/airship-site-manifest/global/v4.0/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, install the `file` package. Define any proxy environment variables needed for your environment to reach public ubuntu package repos, then run:

```
apt -y install file
```

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`. Now install the matching kernel on the Genesis host (make sure to run on Genesis host, not 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-site-manifest/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.

## Manual Steps to Install Calico on Genesis

The equivalent MaaS boot actions must be applied manually to the Genesis node, since this is the one node that MaaS does not deploy. Specifically, the Calico script defined in the `calico-ip-rules` boot action should be manually applied as its own systemd unit on the Genesis node.

To do this, copy the `data` section of the file located at `~/airship-site-manifest/global/v4.0/scripts/configure-ip-rules.yaml` into the file `/opt/configure-ip-rules.sh` on the Genesis host as root (section beginning with `#!/bin/bash`).

Delete the leading spaces in the shebang for the script (i.e., `#!/bin/bash` should have no leading whitespace and appear on the first line of the script).

Then, `chmod 700 /opt/configure-ip-rules.sh`.

Next, copy the first `data` section of the `calico-ip-rules` boot action found at `~/airship-site-manifests/site/${NEW_SITE}/baremetal/bootactions/calico-ip-rules.yaml` into `/etc/systemd/system/configure-ip-rules.service` on the Genesis host as root (this is the section beginning with `[Unit]`. Remove all leading whitespace, then replace `DH_SUB_...` fields with the parameters they map to via deckhand substitutions. For example:

```
    - src:
        schema: pegleg/CommonAddresses/v1
        name: common-addresses
        path: .calico.ip_rule.gateway
      dest:
        path: .assets[0].data
        pattern: DH_SUB_GATEWAY_IP
```

`DH_SUB_GATEWAY_IP` is sourced from `.calico.ip_rule.gateway` in common-addresses. In the `${NEW_SITE}` site definition, this would be `172.29.141.3` as defined in `~/airship-site-manifest/site/${NEW_SITE}/networks/common-addresses.yaml`.

Do the same for the rest of the `DH_SUB...` parameters. Then, `chmod 444 /etc/systemd/system/configure-ip-rules.service`.

Lastly, run the associated systemctl commands to register, execute, and persist the script:

```
sudo systemctl daemon-reload
sudo systemctl start configure-ip-rules
sudo systemctl enable configure-ip-rules
```

## Manual steps for i40e driver installation on Genesis

To do this, copy `airship-site-manifest/tools/genesis-setup/i40e-dkms-install.service. sample` to Genesis host `/etc/systemd/system/i40e-dkms-install.service` as root.

Then, `chmod 444` the file.

Next, copy `airship-site-manifest/tools/genesis-setup/i40e-dkms-install.sh` script to Genesis host `/opt/i40e-dkms-install.sh` as root. Then, `chmod 700` the file.

Lastly, run the associated systemctl commands to register, execute, and persist the script:

```
sudo systemctl daemon-reload
sudo systemctl start i40e-dkms-install
sudo systemctl enable i40e-dkms-install
```

## 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 ~/${NEW_SITE}_bundle
sudo http_proxy=$PROXY https_proxy=$PROXY ./genesis.sh
```

Estimated runtime: **40m**

In the event of failures, refer to [genesis troubleshooting]
(https://promenade.readthedocs.io/en/latest/troubleshooting/genesis.html).

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**

## Deploy Site with Shipyard

Start by cloning the shipyard repository to the Genesis node:

```
git clone https://opendev.org/airship/shipyard.git
```

Refer to the `data/charts/ucp/shipyard/reference` field in `~/airship-site-manifest/global/v4.0/software/config/versions.yaml`.

If this is a pinned reference (i.e., any reference that's not `master`), then you should check-out 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-site-manifest/global/v4.0/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 `foo.example.com/shipyard@sha256:dfc25e1...` in the versions file, then export the previously mentioned environment variable:

```
export SHIPYARD_IMAGE=foo.example.com/shipyard@sha256:dfc25e1...
```

Export valid login credentials for one of the UCP 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, `shipyard` user can be used, with the password that was defined in `~/airship-site-manifest/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 ~/airship-shipyard/tools/deckhand_load_yaml.sh ${NEW_SITE}
~/${NEW_SITE}_collected
```

Estimated runtime: **3m**

Now deploy the site with shipyard:

```
sudo ~/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 UCP and OSH should be fully deployed.

## 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
```

## Establishing tenant OpenStack environment

In the future, it is expected to created OpenStack resources via Ranger or Helm charts. However, before that option is available, it is currently necessary to perform some OpenStack initialization of tenant-driven resources:

- The initial creation of tenant project(s)
- The initial creation of tenant-driven host aggregates and host aggregate
- metadata (e.g., availability zones)
- The initial creation of tenant-driven nova flavors
- The initial creation of tenant-driven shared/floating IP networks
- etc

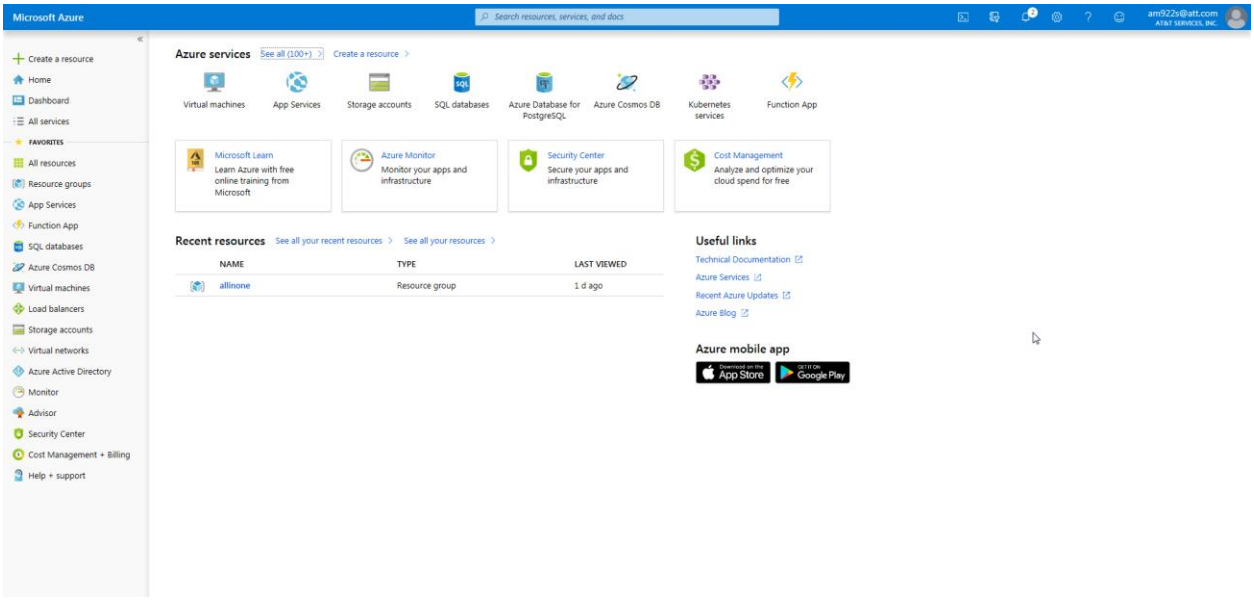
Consult your engineering package for details of how these resources should be configured for your specific environment.

## Install airship on Azure

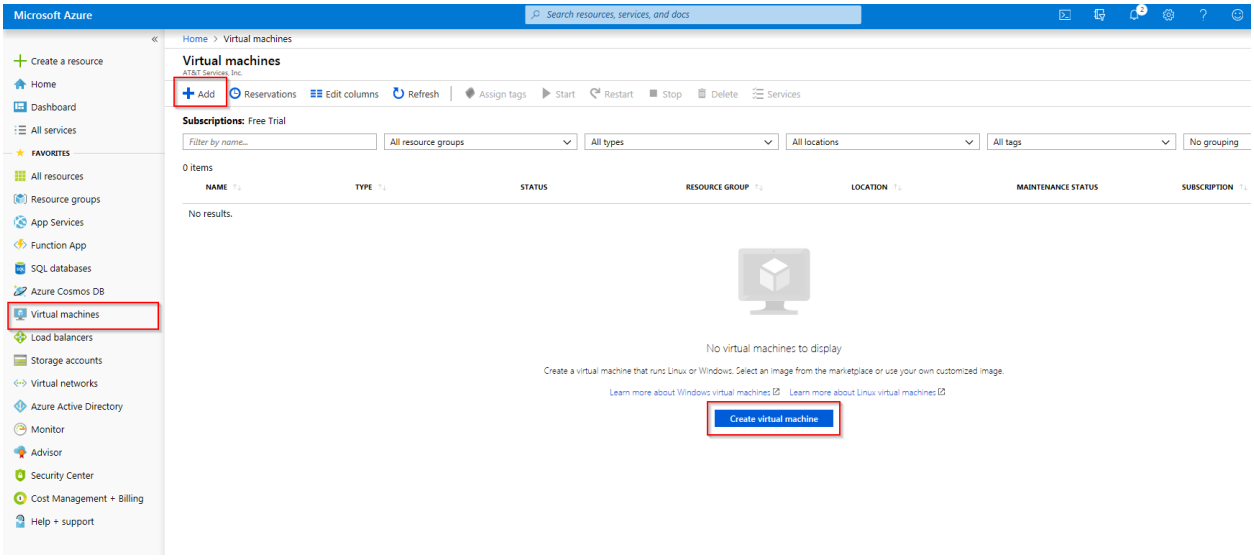
Airship can be installed easily in a hosted VM in Microsoft Azure. The method described below assumes you already have a free (or commercial) account with Azure.

1. Sign in to your azure account using your credentials at https://portal.azure.com

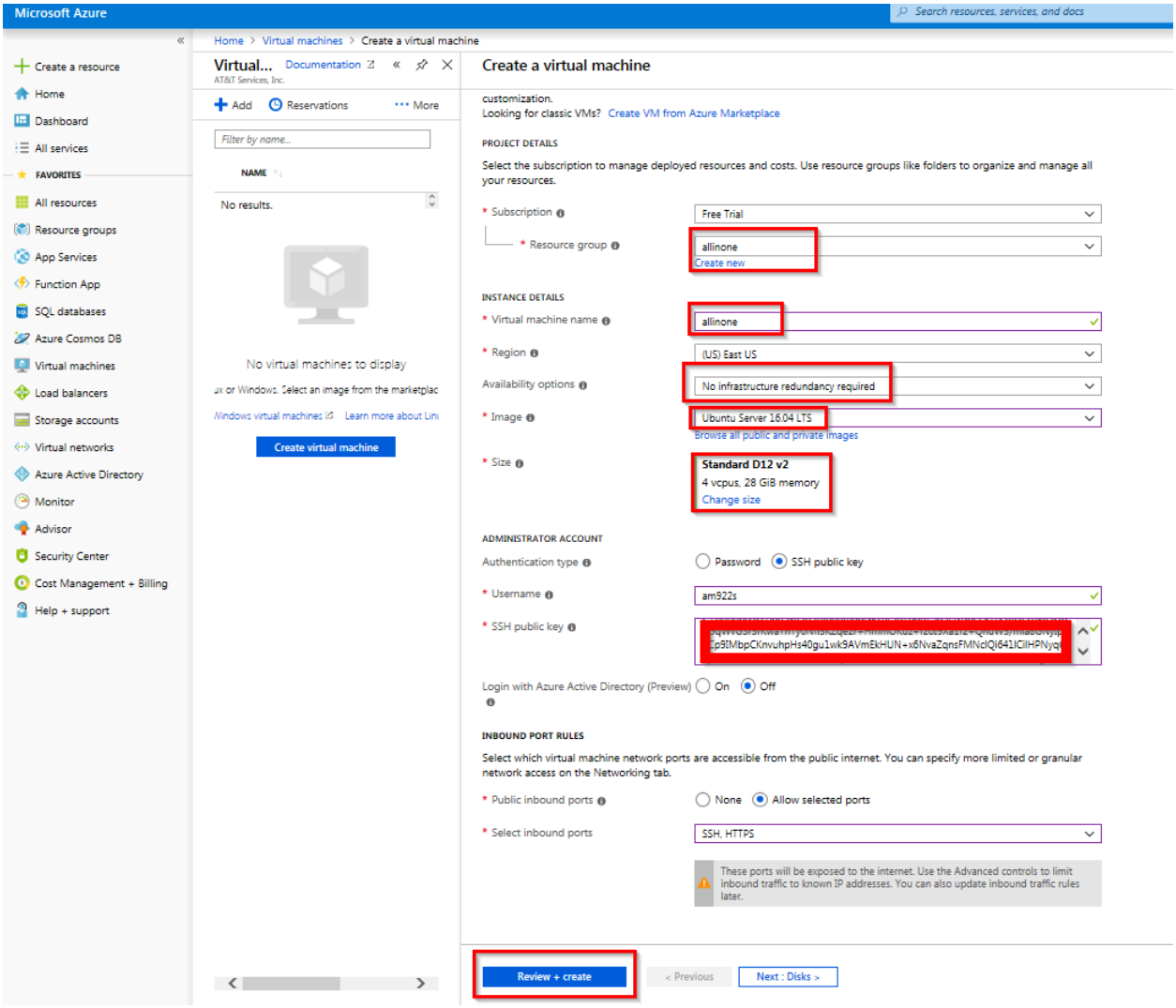
   Once logged in, you will land on to Dashboard view



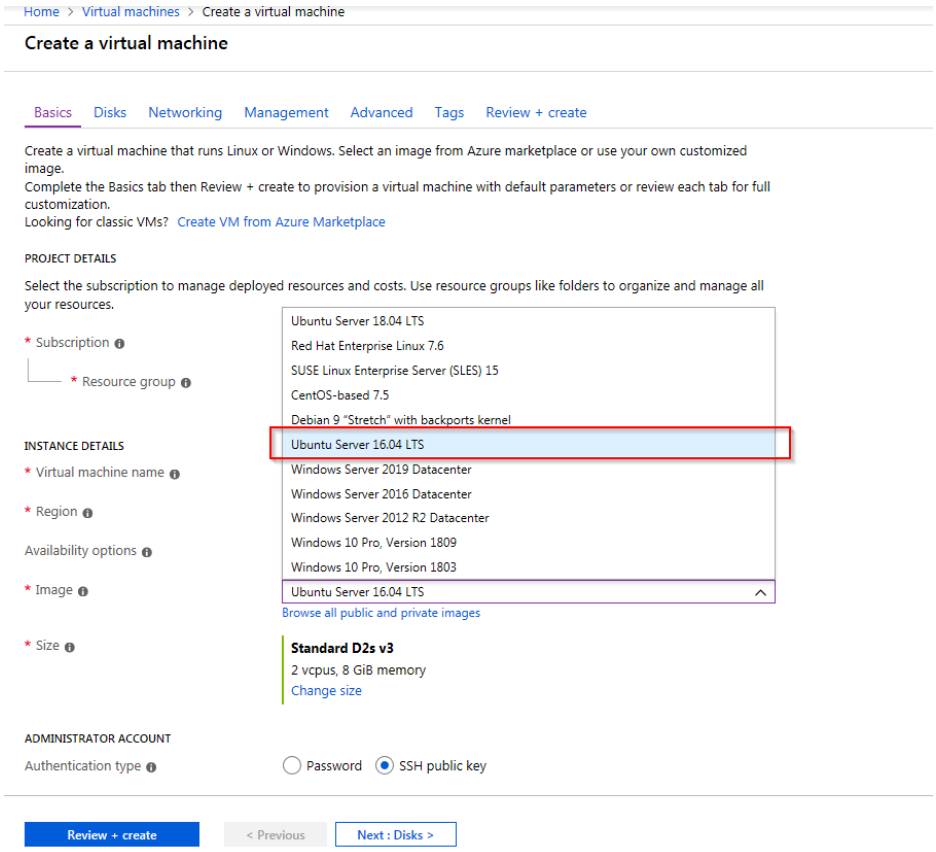

2. Click on “Virtual Machine” on left pane to add/create a new virtual machine.
   Click on “Add” or “Create Virtual Machine”



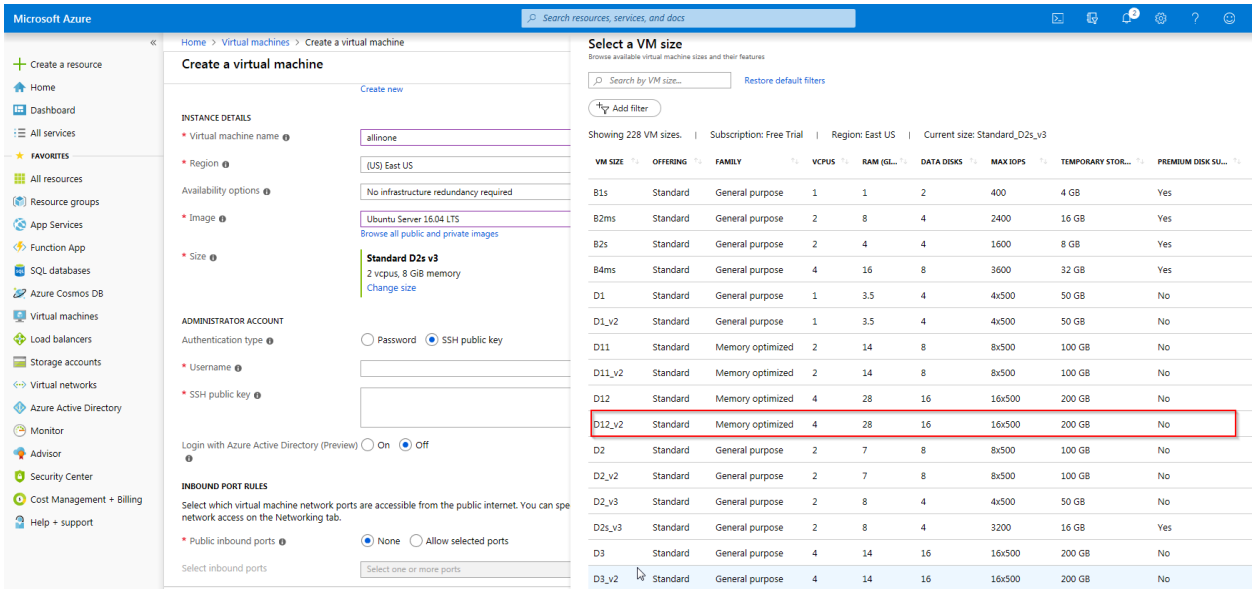
3. Fill in the highlighted information to create new VM



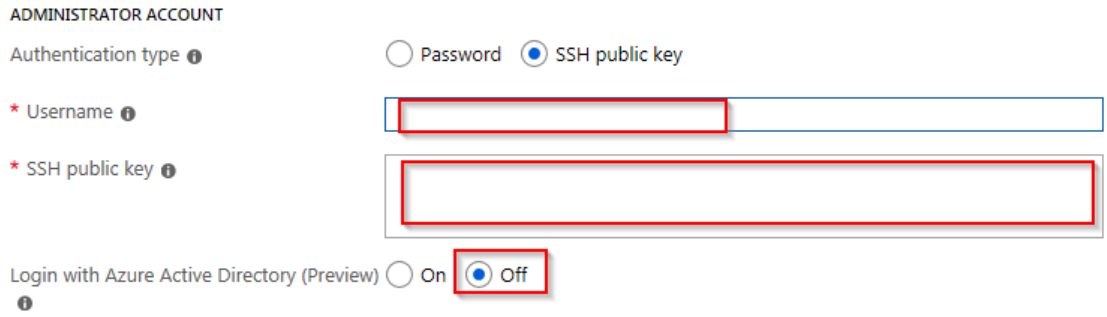

1. Subscription : “Free Tier” or use your organization subscription ID#
2. Resource Group
    a. Name of existing resource or select new one
3. Instance Details
    a. Virtual machine Name : <Mention the name of the VM>
    b. Region : <wherever you want to have your VM located>
    c. Availability Options : <No infrastructure redundancy required>
    d. Image : <Ubuntu Server 16.04 LTS



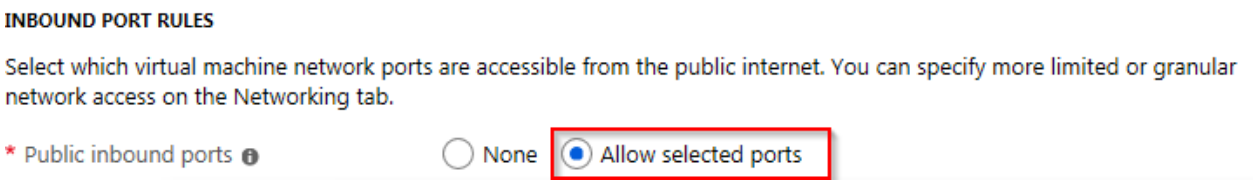

e. Size: Instance configuration. Click on "Change Size" which will open another window to select a flavor.
Select flavor "D12_v2" which meets the minimum requirement to run airship in a single VM.



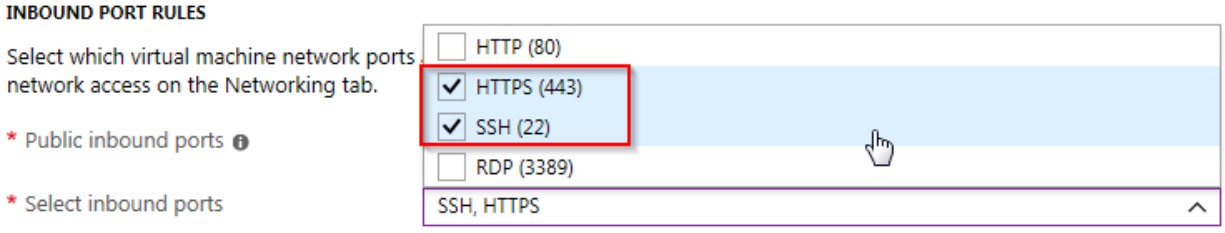
4. Set the administrative account setting be use “SSH public key” only.
    a. Mention the username which will be used for accessing the VM
    b. Copy your ssh public key (rsa/dsa) in to the box
    c. Select Active directory setting “Off”



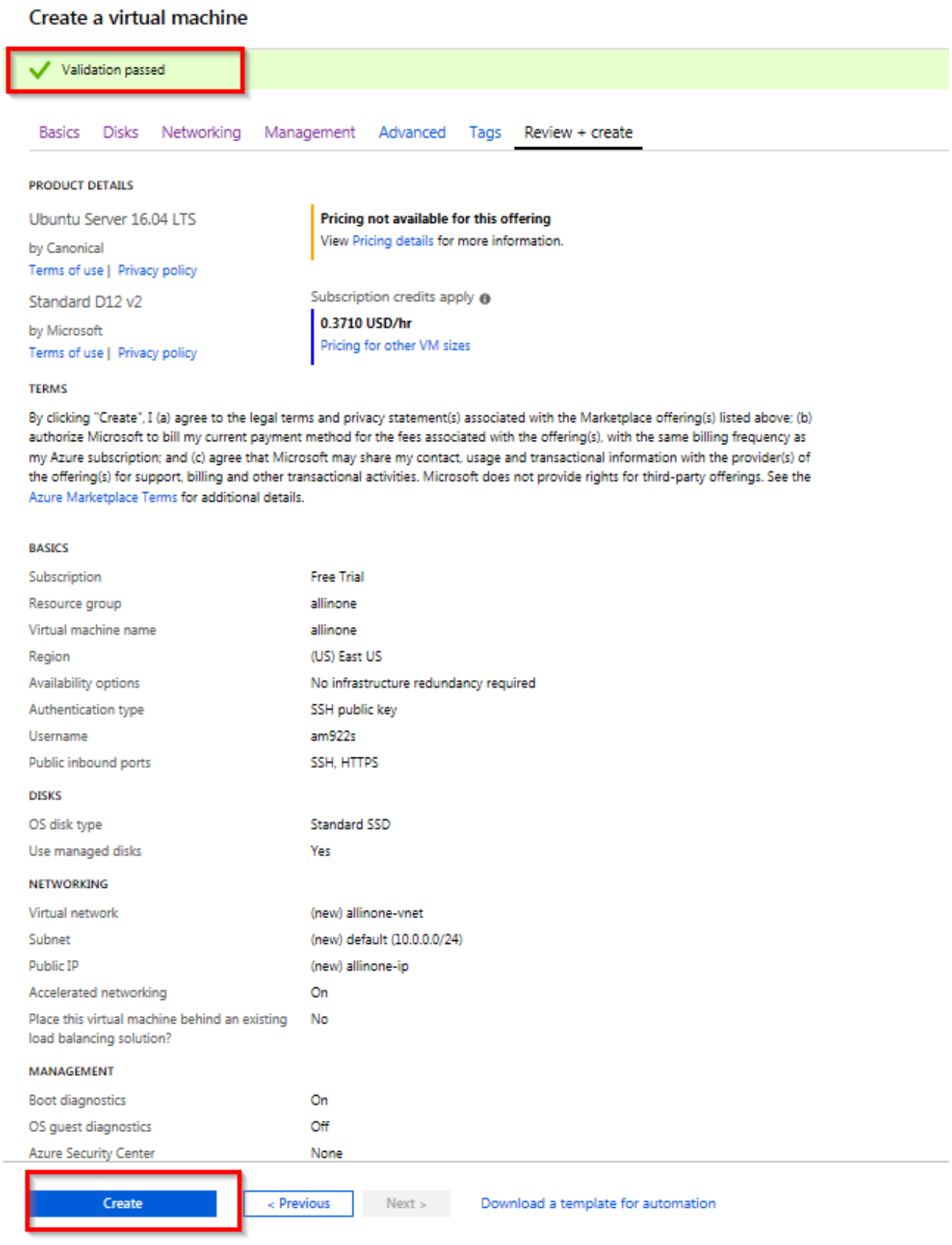

5. Next, setup the inbound port rules
    a. Select “Allow select ports” and enable port 22 (ssh) and port 443 (https).




    b. Select port “22” and “443” as allowed port on public network

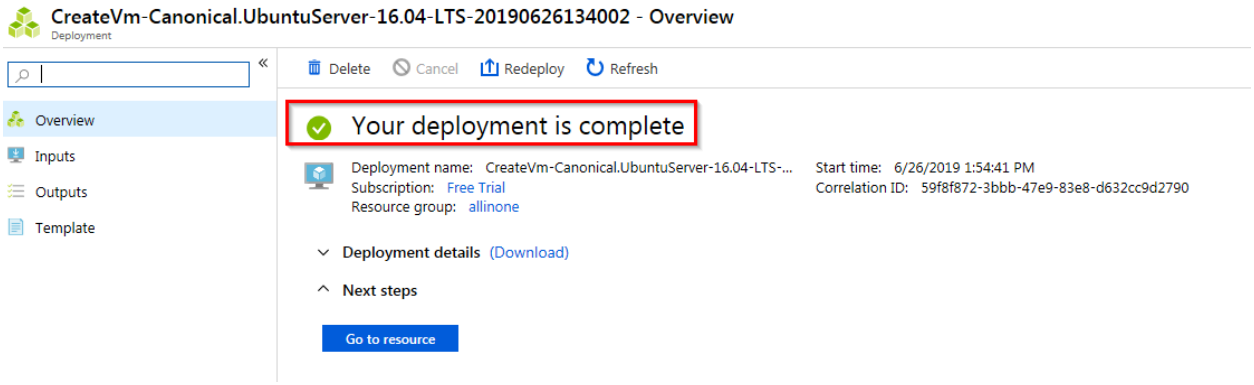



6. Now click on “Review + Create” tab. You will be taken to confirmation if validation for this VM is passed or not

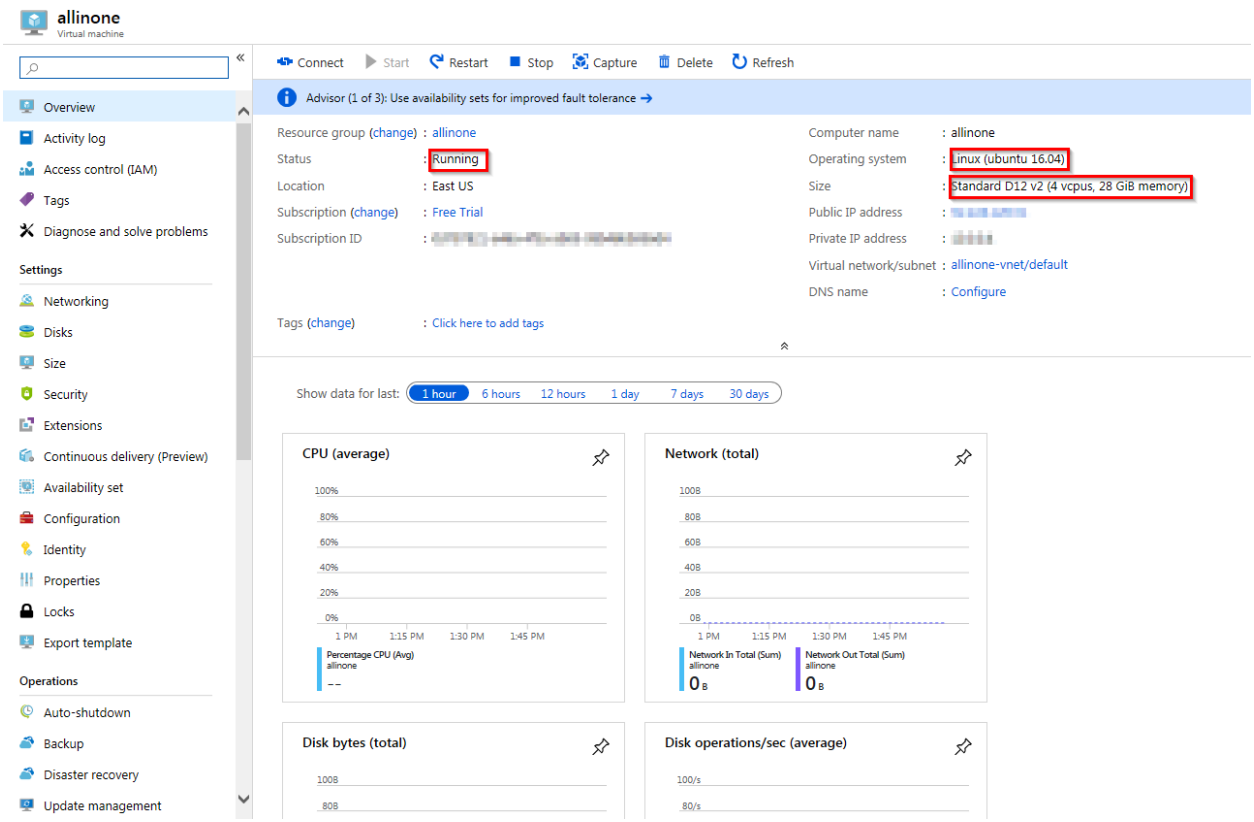



Click on “Create”. This will start the VM creation process.

7. Wait for some time. After VM creation is done, you should see the similar screen.




8. Once your VM is ready click on the “Virtual Machine” and expand VM. You should to see VM status like:




9. Note the public IP address of the VM and connect using preferred ssh client. Use your SSH private key to access the VM

```
#ssh -I <Private_ssh_key_file> <username>@<Public_IP>
```

10. Once you logged in make sure SWAP is off on the host

```
# free -m
              total        used        free      shared  buff/cache
available
Mem:          28114         279       27390           8         444
27390
Swap:             0           0           0
```

11. Run the following commands to setup airship in this VM:

```
#sudo -i
# mkdir -p /root/deploy && cd "$_"
#git clone https://opendev.org/airship/treasuremap/
Cloning into 'treasuremap'...
remote: Enumerating objects: 4382, done.
remote: Counting objects: 100% (4382/4382), done.
remote: Compressing objects: 100% (1778/1778), done.
remote: Total 4382 (delta 2624), reused 4038 (delta 2414)
Receiving objects: 100% (4382/4382), 1.77 MiB | 0 bytes/s, done.
Resolving deltas: 100% (2624/2624), done.
Checking connectivity... done.

cd /root/deploy/treasuremap/tools/deployment/aiab/

./airship-in-a-bottle.sh

Welcome to Airship in a Bottle

 /--------------------\
|                      \
|         |---|         \----
|         | x |              \
|         |---|               |
|           |                 /
|     \____|____/       /----
|                      /
 \--------------------/


A prototype example of deploying the Airship suite on a single VM.


This example will run through:
 - Setup
 - Genesis of Airship (Kubernetes)
 - Basic deployment of Openstack (including Nova, Neutron, and Horizon using
Openstack Helm)
 - VM creation automation using Heat

The expected runtime of this script is greater than 1 hour


The minimum recommended size of the Ubuntu 16.04 VM is 4 vCPUs, 20GB of RAM
with 32GB disk space.
```

```
Let's collect some information about your VM to get started.
Is your HOST IFACE eth0? (Y/n)<Y>
This example will run through:
 - Setup
 - Genesis of Airship (Kubernetes)
 - Basic deployment of Openstack (including Nova, Neutron, and Horizon using
Openstack Helm)
 - VM creation automation using Heat

The expected runtime of this script is greater than 1 hour


The minimum recommended size of the Ubuntu 16.04 VM is 4 vCPUs, 20GB of RAM
with 32GB disk space.
Let's collect some information about your VM to get started.
Is your HOST IFACE eth0? (Y/n) y
Is your LOCAL IP 10.0.0.4? (Y/n) y
++ hostname -s
+ export SHORT_HOSTNAME=allinone
+ SHORT_HOSTNAME=allinone
+ set +x
Updating /etc/hosts with: 10.0.0.4 allinone
10.0.0.4 allinone
+ export HOSTIP=10.0.0.4
+ HOSTIP=10.0.0.4
+ export HOSTCIDR=10.0.0.4/32
+ HOSTCIDR=10.0.0.4/32
+ export NODE_NET_IFACE=eth0
+ NODE_NET_IFACE=eth0
+ export TARGET_SITE=aiab
+ TARGET_SITE=aiab
+ set +x
Using DNS servers 168.63.129.16 and 168.63.129.16.
```

12. The setup can generally run for 60 – 80 minutes. Once the setup is complete you can check for the status of all airship components.

```
NAMESPACE     NAME                                          READY   STATUS
RESTARTS   AGE

kube-system   airship-coredns-test                           0/1     Completed
0          1h

kube-system   airship-haproxy-haproxy-test                   0/1     Completed
0          1h

kube-system   airship-kubernetes-calico-etcd-etcd-test       0/1     Completed
0          1h

kube-system   airship-kubernetes-etcd-etcd-test              0/1     Completed
0          1h

kube-system   auxiliary-etcd-allinone                        3/3     Running
0          1h
```

```
kube-system   bootstrap-armada-allinone                   4/4    Running
0          1h

kube-system   calico-etcd-allinone                        1/1    Running
0          1h

kube-system   calico-etcd-anchor-mkqng                    1/1    Running
0          1h

kube-system   calico-kube-controllers-74b4879ddd-5vb96    1/1    Running
1          1h

kube-system   calico-node-j9lqv                           1/1    Running
0          1h

kube-system   calico-settings-lkmsw                       0/1    Completed
0          1h

kube-system   coredns-78b6d6db4-68c6z                     1/1    Running
0          1h

kube-system   coredns-78b6d6db4-7qxf9                     1/1    Running
0          1h

kube-system   coredns-78b6d6db4-9ml8f                     1/1    Running
0          1h

kube-system   haproxy-allinone                            1/1    Running
1          1h

kube-system   haproxy-anchor-zfksm                        1/1    Running
0          1h

kube-system   ingress-8db44f6cf-kfjhh                     1/1    Running
0          1h

kube-system   ingress-error-pages-55f95944fc-t9784        1/1    Running
0          1h

kube-system   kubernetes-apiserver-allinone               1/1    Running
0          1h

kube-system   kubernetes-apiserver-anchor-f2xhx           1/1    Running
0          1h

kube-system   kubernetes-controller-manager-allinone      1/1    Running
2          1h

kube-system   kubernetes-controller-manager-anchor-mp8d6  1/1    Running
0          1h

kube-system   kubernetes-etcd-allinone                    1/1    Running
0          1h

kube-system   kubernetes-etcd-anchor-266hg                1/1    Running
0          1h

kube-system   kubernetes-proxy-85kzm                      1/1    Running
0          1h

kube-system   kubernetes-scheduler-allinone               1/1    Running
2          1h
```

```
kube-system   kubernetes-scheduler-anchor-2b6rl           1/1    Running
0          1h
nfs           nfs-provisioner-6697458b7d-hw4sf            1/1    Running
0          1h
```

13. Look for the following successful install message.

```
OpenStack Horizon dashboard is available on this host at the following URL:

  http://<Internal_IP_Address_of_your_VM>:31724

Credentials:
  Domain: default
  Username: admin
  Password: <********>

OpenStack CLI commands could be launched via `./openstack` script, e.g.:
  # cd /root/deploy/treasuremap/..//treasuremap/tools/
  # ./openstack stack list
  ...

Other dashboards:

  MAAS: http:// <Internal_IP_Address_of_your_VM>/MAAS/ admin/<********>
  Airship Shipyard Airflow DAG: http:// <Internal_IP_Address_of_your_VM>/

Airship itself does not have a dashboard.

Other endpoints and credentials are listed in the following locations:
  /root/deploy/treasuremap/..//type/sloop/config/endpoints.yaml
  /root/deploy/treasuremap/..//treasuremap/site/aiab/secrets/passphrases/

Exposed ports of services can be listed with the following command:
  # kubectl get services --all-namespaces | grep -v ClusterIP
  ...

+ your_next_steps
+ set +x

----------------------------------------------------------------

Airship has completed deployment of OpenStack (OpenStack-Helm).

Explore Airship Treasuremap repository and documentation
available at the following URLs:

  https://opendev.org/airship/treasuremap/
  https://airship-treasuremap.readthedocs.io/

----------------------------------------------------------------
```

# Airship Troubleshooting

## Health Checks

Perform the health checks in this section to verify the health of an Airship site.

### Verify Peering

Verify that peering is established.

```
% sudo /opt/cni/bin/calicoctl node status
```

Example:

```
% sudo /opt/cni/bin/calicoctl node status
Calico process is running.

IPv4 BGP status

+--------------+-----------+-------+------------+-------------+
| PEER ADDRESS | PEER TYPE | STATE |   SINCE    |    INFO     |
+--------------+-----------+-------+------------+-------------+
| 172.29.0.2   | global    | up    | 2018-05-22 | Established |
| 172.29.0.3   | global    | up    | 2018-05-22 | Established |
+--------------+-----------+-------+------------+-------------+

IPv6 BGP status
No IPv6 peers found.
```

Verify that STATE is `up` and INFO is `Established`. However, if STATE is `start` and INFO is `Connect`, peering has failed.

For more information on Calico troubleshooting, Visit the latest publication of: https://docs.projectcalico.org/v3.4/usage/troubleshooting/

### Kubernetes Health Checks

Check Kubernetes health:

| Command | Description |
|---|---|
| `% kubectl get nodes` | Verify that for all nodes, STATE is `Ready`.<br><br>**Note:** After a reboot, it may take as long as 30 minutes for a node to stabilize and reach a `Ready` condition. |

| Command | Description |
|---|---|
| `% kubectl get pods --all-namespaces \| grep Running \| grep 0/` | Verify that liveness probes for all pods are working. This command exposes pods whose liveness probe is failing. |
| `% kubectl get pods --all-namespaces \| grep -v Running \| grep -v Completed` | Verify that all pods are in the `Running` or `Completed` state. This command exposes pods that are not running or completed. |
| `% kubectl get pods --all-namespaces -o wide \| grep Crash` | Look for crashed pods. |
| `% kubectl get pods --all-namespaces -o wide \| grep core`<br>`% kubectl get services --all-namespaces \| grep core` | Check the health of core services. |
| `% kubectl get pods --all-namespaces -o wide \| grep proxy` | Check the health of proxy services. |
| `% kubectl get pods --all-namespaces -o wide -w` | Get all pod details. |
| `% kubectl get jobs –all-namespaces -o wide \| grep -v "1 1"` | Look for failed jobs. |

## OpenStack Health Checks

Check OpenStack health by issuing the following commands at the terminal:

| Command | Description |
|---|---|
| `% openstack token issue` | Verifies Keystone by requesting a token. |
| `% openstack network list` | Verify networks. |
| `% openstack subnet list` | Verify subnets. |
| `% openstack server list` | Verify VMs. |

| Command | Description |
| --- | --- |
| `% openstack hypervisor list` | Verify compute hypervisors. |

### Check Ceph Status

Check Ceph status:

```
% MON_POD=$(sudo kubectl get --no-headers pods -n=ceph -
l="application=ceph,component=mon" | awk '{ print $1; exit }')
% echo $MON_POD
% sudo kubectl exec -n ceph ${MON_POD} -- ceph -s
```

For troubleshooting information, see Ceph Troubleshooting section of this book.

### Check for kube-proxy iptables NAT Issues

Check the IP tables and make sure the IP addresses are the same:

```
% iptables -n -t nat -L | grep coredns
% kubectl -n kube-system get -o wide pod | grep coredns
```

### Release Details

Details about each Airship site deployment are available to operators. These details include `git` commit IDs, branches, and tags for the release as well as deployment completion status. This information is useful for troubleshooting and for mapping observed issues and behavior to the software version (`git` branch and tag) to identify root cause and potential fixes.

Release details are specific to each site. The release details are available through Kubernetes and also in a file on each node.

- In Kubernetes, the release details are in the `deployment-status` file in the ConfigMap resource. Use this command:

```
kubectl –n ucp get cm deployment-status
```

- On each node, the release details are in the `airship-release` file:

```
cat /etc/airship-release
```

#### *Notes*

- The `deployment:status` value indicates whether the site deployment process completed, but it does not indicate whether the deployment was successful.

- The `dirty` value for each repository indicates whether the repository contains uncommitted document changes.
- The `airship-release` file is updated whenever the ConfigMap changes.

*Example*

The example `airship-release` file below indicates that the deployment process completed and that the repositories (site, global, and secrets) contain no uncommitted documents.

```
$ cat "/etc/airship-release"
deployment:
  status: completed
  context-marker: c7a9c65c886641029ce84724ba7a999f8542a7c5
  action: 01BZZK07NF04XPC5F4SCTHNPKN
  document_url: http(s)://deckhand_url/doc/revision/123/
  user: ab1234
  date: 2019-09-11 12:34:56 UTC
documents:
  site-repository:
    commit: 37260deff6a213e30897fc284a993c791336a99d
    tag: master
    dirty: false
  globals:
    commit: 971aa120762527302100fddb4f53be071874e504
    tag: v1.3
    dirty: false
  secrets:
    commit: 23e7265aee4843301807d649036f8e860fda0cda
    tag: master
    dirty: false
```

## Resolving Common Issues

This section provides solutions for some common issues. In some cases, failures are indicated on LMA dashboards, but some job and pod failures may not appear on dashboards. In general, follow these guidelines:

- Kubernetes jobs: In job listings, the DESIRED and SUCCESSFUL values should match.
- Kubernetes pods: In pod listings, all replicas should report ready (READY values of 1/1, 2/2, or similar).

### elastic-curator Job Failure

The `elastic-curator` job can fail sometimes. In the example job listing below, the DESIRED field shows 1 replica and SUCCESSFUL has a value of 0, which indicates failure.

```
% kubectl get jobs
NAMESPACE  NAME                          DESIRED   SUCCESSFUL   AGE  CON
TAINERS  IMAGES              SELECTOR
...
osh-infra  elastic-curator-
1551657600  1         0            22h  curator     docker-open-
airship.<laburl>.com/upstream-  local/bobrik/curator@sha256:ab2fae9cd5
22a56207719529fe865be819307799963f1a8eb955f1d6f64a018d
```

To resolve the issue, delete the job:

```
# kubectl delete jobs -n osh-infra elastic-curator-<REPLICA-ID>
```

The `elastic-curator` cron job runs every 6 hours. When it runs again per schedule, verify that it succeeded.

**Note:** The `elasticsearch-snapshot` job may also fail. List the job and verify that it is successful. Delete it as well if it failed.

### exporter-create-sql-user Job Failure

When an `exporter-create-sql-user` job fails, restart the failed job. The following example shows pods and jobs listings for a failed `exporter-create-sql-user` failure. Note the READY column (column 3), indicating 0 out of 1 pod is ready.

```
# kubectl get pods --all-namespaces -o wide | grep -v Completed |
egrep "0\/|1\/2"
openstack        prometheus-mysql-exporter-f6cfb8f9b-k6pww
0/1       Init:0/1    0          1h        10.97.1.170
auk51r05o001
ucp              airship-drydock-auth-test
0/1       Error       0          22h       10.97.124.142
auk51r03o001
# kubectl get jobs -n openstack | grep exporter-create-sql-user
exporter-create-sql-user                           1         0
56m
```

### maas-rack Pod Failure

When a `maas-rack` pod fails, delete the failed pod. Kubernetes will then restart the pod.

**Important:** Be sure to delete the failed pod before the next run of Drydock. If Drydock runs, and the pod is inactive, Drydock will attempt to re-provision the node where the failed pod runs, resulting in further operational issues.

### Stuck nova-service-cleaner Job

When a `nova-service-cleaner` job is stuck, restart the job. This should fix the nova-service-cleaner job and resume normal operations.

### MariaDB Pod Failure

If the MariaDB pod fails, delete `/var/lib/mysql/*` from the MariaDB pod. Then restart the pod.

### Tenant Ceph in HEALTH_WARN State

A defect in the calculation of number of placement groups (PGs) can place tenant Ceph in the `HEALTH_WARN` state. Confirm this condition as shown in the following example, substituting the proper `ceph-mon` pod name for `ceph-mon-54pzm`:

```
# kubectl exec -it -n tenant-ceph ceph-mon-54pzm -- ceph -s
  cluster:
    id:     6fed55f6-614e-4190-8547-debfcfa1a7da
    health: HEALTH_WARN
            1 pools have many more objects per pg than average
services:
    mon: 3 daemons, quorum auk51r03o001,auk51r04o001,auk51r06o001
    mgr: auk51r05o001(active), standbys: auk51r04o001, auk51r03o001
    osd: 32 osds: 32 up, 32 in
    rgw: 3 daemons active  data:
    pools:   17 pools, 1032 pgs
    objects: 11.64 k objects, 17 GiB
    usage:   54 GiB used, 70 TiB / 70 TiB avail
    pgs:     1032 active+clean
```

Fix this condition by running the following script.

```
  #!/bin/bash
  set -x
  NAMESPACE="tenant-ceph"
  PG_TARGET=128
  PGP_TARGET=128
  MON=$(kubectl get pod -n ${NAMESPACE} | grep ceph-mon | head -1
  | awk '{print $1}');
  for POOL_NAME in default.rgw.log;
  do
    echo "POOL_NAME: '${POOL_NAME}'"
    CURRENT_PG_VALUE=$(kubectl exec -i $MON -n ${NAMESPACE} -- ceph osd
  pool get "${POOL_NAME}" pg_num | awk "/^pg_num:/ { print \$NF }")
    CURRENT_PGP_VALUE=$(kubectl exec -i $MON -n ${NAMESPACE} -- ceph
  osd pool get "${POOL_NAME}" pgp_num | awk "/^pgp_num:/ { print \$NF
  }")
    echo "PG_NUM: ${CURRENT_PG_VALUE}"
    echo "PGP_NUM: ${CURRENT_PGP_VALUE}"
    if [[ "${CURRENT_PG_VALUE}" -eq 8 ]]; then
      echo "Adjusting PG_NUM=${PG_TARGET} for POOL=${POOL_NAME}"
      kubectl exec -i $MON -n ${NAMESPACE} -- ceph osd
  pool set "${POOL_NAME}" nosizechange false
      kubectl exec -i $MON -n ${NAMESPACE} -- ceph osd
  pool set "${POOL_NAME}" nopgchange false
      kubectl exec -i $MON -n ${NAMESPACE} -- ceph osd
  pool set "${POOL_NAME}" pg_num ${PG_TARGET}
      kubectl exec -i $MON -n ${NAMESPACE} -- ceph osd
  pool set "${POOL_NAME}" nosizechange true
      kubectl exec -i $MON -n ${NAMESPACE} -- ceph osd
  pool set "${POOL_NAME}" nopgchange true
```

```
  fi;
  if [[ "${CURRENT_PGP_VALUE}" -eq 8 ]]; then
    echo "Adjusting PGP_NUM=${PGP_TARGET} for POOL=${POOL_NAME}"
    kubectl exec -i $MON -n ${NAMESPACE} -- ceph osd
pool set "${POOL_NAME}" nopgchange false
    kubectl exec -i $MON -n ${NAMESPACE} -- ceph osd
pool set "${POOL_NAME}" pgp_num ${PGP_TARGET}
    kubectl exec -i $MON -n ${NAMESPACE} -- ceph osd
pool set "${POOL_NAME}" nopgchange true
  fi;
done;
```

### Authentication Failures Due to Incorrect Mail Configuration

Authentication failures occur when SSH keys are unavailable. This problem can occur when the Genesis node is incorrectly attempting to send mail through `sendmail` rather than `heirloom-mailx`, as indicated in `/var/log/mail.log`. To resolve this issue, reinstall mail packages. Follow these steps:

1. Purge `ssmtp`, `sendmail`, and `heirloom-mailx`.
2. Install `ssmtp`.
3. Configure the mail hub in `ssmtp.conf`, ensuring it has a DNS-resolvable hostname.
4. Install `heirloom-mailx`.
5. Send yourself a test mail message. Monitor the process in `/var/log/mail.log`, and make sure the mail arrives in your inbox. Use this command:

```
mail -s "test mail" your_ID@provider.com < /dev/null
```

## Docker Failure Causing Various Problems

A Docker failure can cause a variety of problems including a node being stuck in the `NotReady` status, failed services, and slow startup times for services. To determine if a docker failure is the cause, follow these steps.

1. Check to see if any nodes are in `NotReady` state.

```
# kubectl get nodes | grep NotReady
auk51r10c002   NotReady  <none>    15d       v1.11.6
```

2. If a node is `NotReady`, list pods in the node. Look for pods in the `Unknown` or `NodeLost` state.

```
# kubectl get pods | grep 'Unknown|NodeLost'
ceph-osd-default-83945928-
fwsbd            1/1       NodeLost           0          49m
ceph-rbd-provisioner-cd77dbb65-
6ncqr       1/1       Unknown            0          49m
```

3. If pods are in the `Unknown` or `NodeLost` state, check the load average for the node. Use the top utility.

```
# top
top - 18:17:50 up 15 days, 14:49,  3 users,  load average: 57.63,
51.40, 48.39
Tasks: 3037 total,   3 running, 2517 sleeping,   0 stopped,   1 zombie
%Cpu(s): 12.4 us,  4.9 sy,  0.0 ni, 81.3 id,  1.1 wa,  0.0 hi,  0.2
si,  0.0 st
KiB Mem : 39490604+total, 19300115+free, 12485539+used, 77049496
buff/cache
KiB Swap:        0 total,        0 free,        0 used. 25199379+avail
Mem
    PID USER      PR  NI    VIRT    RES    SHR S  %CPU %MEM     TIME+
COMMAND
 543957 root      20   0 25.902g 2.337g  23244 S 327.8  0.6  27968:07
kubelet
 686360 systemd+  20   0  0.638t 0.013t 1.632g S 198.4  3.6   4595:49
java
 686373 mfe       20   0 46.404g 350924   7216 S 148.9  0.1   1704:06
beam.smp
3723123 3000      20   0 3872920 3.654g   2556 R 100.0  1.0 879:40.93
nagios
4073711 mfe       20   0 46.187g 178208   7268 S  83.5  0.0   1448:40
beam.smp
 612434 mfe       20   0 46.290g 285156   7228 S  76.1  0.1   1570:22
beam.smp
```

4. A load average greater than 50 indicates problems with kubelet and docker. Examine the docker and kubelet logs for indications of problems.

```
# journalctl -r -u docker
# journalctl -r -u kubelet
```

5. If the logs show that kubelet was unable to connect to docker or that docker commands take more than 5 seconds to respond, execute the following:

```
# systemctl stop kubelet
# systemctl restart docker
(...wait 5 minutes...)
# systemctl start kubelet
```

6. If these steps do not resolve the issues, execute the steps a second time, and if necessary, a third time. If executing them 3 times does not resolve the issue, reboot the node.

## Kubernetes apiserver Pods Crash Loop

Under some conditions, Kubernetes `apiserver` pods crash loop with the following log message:

```
failed to initialize admission: failed to read plugin config: unable
to read admission control configuration from
"/etc/kubernetes/apiserver/acconfig.yaml" [open
/etc/kubernetes/apiserver/acconfig.yaml: no such file or directory]
```

To resolve this issue, follow these steps.

1. Create this file: `/etc/kubernetes/apiserver/eventconfig.yaml`
2. Edit the file, inserting this content:

```
apiVersion: eventratelimit.admission.k8s.io/v1alpha1
kind: Configuration
limits:
- burst: 1000
  qps: 100
  type: Server
```

3. Create this file: `/etc/kubernetes/apiserver/acconfig.yaml`
4. Edit the file, inserting this content:

```
apiVersion: apiserver.k8s.io/v1alpha1
kind: AdmissionConfiguration
plugins:
- name: EventRateLimit
  path: eventconfig.yaml
```

5. Once these files are created, an apiserver and its anchor should be created, restoring access to Kubernetes.
6. Delete the second apiserver anchor pod and wait for the anchor and apiserver pods to be recreated.
7. Delete the third apiserver anchor pod and wait for the anchor and apiserver pods to be recreated.
8. Delete the apiserver anchor pod which was fixed manually by creating the files in the preceding steps and wait for the anchor and apiserver pods to be recreated.
9. Ensure all apiserver and apiserver anchor pods have been restored.

### IPMI Commands Fail to Make Active Connection

IPMI connection problems maybe indicated by failures as mentioned below:

```
# ipmitool -I lanplus -H <IPMI_IP> -U <Username> -P <Password> power
status
Error: Unable to establish IPMI v2 / RMCP+ session
```

To resolve the issue, reset the IPMI password to the same password:

```
# /opt/dell/srvadmin/sbin/racadm -r <IPMI_IP> -u <Username> -
p <Password> set iDRAC.Users.3.Password <Password>
```

**Important:** The password should be the same to which was already in use. Do not use a new password.

### maas-rack Pods Not Ready

Under certain circumstances, maas-rack pods may not become ready due to register duplicate errors. Example:

```
ucp maas-rack-0 0/1 Running 0 53m 172.29.0.14 auk51r06o001
ucp maas-rack-1 0/1 Running 0 1h 172.29.0.11 auk51r03o001
```

Retrieve logs by accessing the maas-rack pod via exec to execute this command:

```
journalctl -u register-rack-controller
```

Example log:

```
Mar 25 17:13:54 auk51r05o001 register-rack-controller.sh[1488]: + '['
1 ']'
Mar 25 17:13:54 auk51r05o001 register-rack-controller.sh[1488]: + maas
local rack-controller delete xgt8xb
Mar 25 17:13:54 auk51r05o001 register-rack-controller.sh[1488]: + [[ 2
-ne 0 ]]
Mar 25 17:13:54 auk51r05o001 register-rack-controller.sh[1488]: + echo
'Could not delete rack controller.'
Mar 25 17:13:54 auk51r05o001 register-rack-controller.sh[1488]: Could
not delete rack controller.
Mar 25 17:13:54 auk51r05o001 register-rack-controller.sh[1488]: +
sleep 10
```

To resolve the issue, delete the `maas-rack` pod. It will be recreated. After approximately 5 minutes, it should be in the `Ready` state.

## Diagnosing a Problem

Once it has been determined that a problem is occurring, follow these steps to diagnose the problem.

1. Use Nagios to determine if there are alerts on any hosts.
2. Use Grafana to analyze site metrics. Grafana shows some health metrics, such as ceph, but mostly it is a way to visualize metrics such as I/O spikes and memory pressure. These metrics provide important insight into the health of the hosts, but Grafana cannot identify problems or otherwise detect suspicious performance. Expertise and careful analysis of the data are required to identify problems on the hosts.
3. Validate in Kubernetes that you have the right number of control plane hosts and compute hosts visible. For example:

```
  user@lab:~$ sudo kubectl get nodes -o wide
```

4. You can use Kubernetes to extract the logs from any pod of interest. For example:

```
# find all pods that seem unhealthy in the environment (normally, 0 pods
unhealthy) - this is focusing only on openstack namespace
user@lab:~$ sudo kubectl get pod -n openstack | grep -v Running | grep -v
Completed
# describe the pod to figure out which container within the pod is actually
crashing
user@lab:~$ sudo kubectl describe pod neutron-sriov-agent-default-rshs9 -n
openstack
# extract logs from a particular container in the pod - this pod has 3
containers, focus is on one of the init containers
user@lab:~$ sudo kubectl logs neutron-sriov-agent-default-rshs9  -n openstack
-c neutron-sriov-agent-init
```

5. Use Kibana to drill into the platform logs. It provides both an integrated view of logs and also detail at the host, container, service, and application levels. In addition to the provided filters, you can create your own custom filters.
6. Examine ceph:
    - Verify general cluster health. Look for blocked requests and unhealthy components.
    - Check ceph OSD, MON, PG status

Further useful information can also be found in the Kubernetes documentation:

https://kubernetes.io/docs/tasks/debug-application-cluster/debug-application-introspection/

### Listing a Cluster

To get a list of all the pods and their status, run:

```
# sudo kubectl get pods --all-namespaces
```

If there are pods **not** in a Running state, this may indicate an issue inside the container which caused the service to exit (e.g., CrashLoopBackOff), or it may indicate a Kubernetes issue before even trying to start the container (e.g., ImagePullBackOff).

### Pod Logs, Pod Descriptions, and Pod Exec

To troubleshoot the failure of a containerized service, display the logs of the failing service as follows:

```
sudo kubectl logs <POD_NAME> --namespace=<NAME_SPACE>
```

where <POD_NAME> is the name of the failing pod, and <NAME_SPACE> is the namespace of that pod (as reported in the output of `sudo kubectl get pods --all-namespaces`).

To troubleshoot a Kubernetes issue with the pod, use the following command:

```
sudo kubectl describe pod <POD_NAME> --namespace=<NAME_SPACE>
```

This command returns details of Kubernetes-related failures.

To execute a command inside a pod, use this command:

```
kubectl exec -it –namespace <NAME_SPACE> <POD_NAME> --
<SHELL or COMMAND>
```

Example:

```
# kubectl exec -it -n openstack ro-api-5f9bfc95f6-5x55c -- sh
```

### Changing the Log Level for OpenStack Components

By default, the log level for OpenStack components is set to `INFO`. For troubleshooting purposes, however, it may be necessary to generate more detailed logs by setting the log level to `DEBUG`. Changing the log level involves cloning the appropriate repo, editing the YAML file for the appropriate component, merging the patch set, and triggering a site update. Follow these steps.

1. Identify the appropriate repo and clone it. If most cases it should be the ‘site manifest’ repo.

```
git clone ssh://<user>@<Gerrit_repo_url>:29418/airship-site-manifests
# <user> is your user ID

cd airship-site-manifests
```

2. Locate the YAML file for the component you are troubleshooting. In the path names below, <site-name> is the name of your site.

- Nova:

```
site/<site-name>/software/charts/osh/openstack-compute-kit/nova.yaml
#example: site/auk5/software/charts/osh/openstack-compute-
kit/nova.yaml
```

- Neutron:

```
site/<site-name>/software/charts/osh/openstack-compute-
kit/neutron.yaml
#example: site/auk5/software/charts/osh/openstack-compute-
kit/neutron.yaml
```

- Heat, horizon, cinder, glance or keystone:

```
site/<site-name>/software/charts/osh/openstack-
<component>/<component>.yaml
#example: site/auk5/software/charts/osh/openstack-heat/heat.yaml
```

3. Find the level key of the corresponding logger and set its value to DEBUG. The YAML file may be more complicated than the example below due to nested dictionaries. Be careful to change only the value of the level key of the appropriate logger(s). Please note that Neutron has two loggers to update.

```
data:
  values:
    conf:
      logging:
        logger_neutron:
          level: DEBUG
        logger_neutron_taas:
          level: DEBUG
```

4. Save the YAML file and push the change:

```
git add <file-name-with-path>
git commit
git push origin HEAD:refs/for/master
```

Example output, where *<user>* is your user ID:

```
ubuntu@ubuntu:~/repos/airship-site-
manifests/site/mtn52a/software/charts/osh$ git push origin
HEAD:refs/for/master
Counting objects: 20, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (15/15), done.
Writing objects: 100% (20/20), 1.88 KiB | 0 bytes/s, done.
Total 20 (delta 8), reused 0 (delta 0)
remote: Resolving deltas: 100% (8/8)
remote: Processing changes: refs: 1, done
remote: ERROR: [ec298b4] missing Change-Id in commit message footer
remote:
remote: Hint: To automatically insert Change-Id, install the hook:
remote:   gitdir=$(git rev-parse --git-dir); scp -p -P
29418 <user>@<Gerrit ip>:hooks/commit-msg ${gitdir}/hooks/
remote: And then amend the commit:
remote:   git commit --amend
remote:
To ssh://<user>@<Gerrit ip>:29418/airship-site-manifests
! [remote rejected] HEAD -> refs/for/master ([ec298b4] missing Change-
Id in commit message footer)
error: failed to push some refs to 'ssh://<user>@<Gerrit
ip>:29418/airship-site-manifests' # see below if you get this error
```

If you get the above error, perform these actions:

```
#Set gitdir variable as shown in the output above, to install the
hook:
gitdir=$(git rev-parse --git-dir); scp -p -P 29418 <user>@<Gerrit
ip>:hooks/commit-msg ${gitdir}/hooks/
git commit –amend
```

Then push again:

```
git push origin HEAD:refs/for/master
```

5. Once the patch set is reviewed and approved, it can be merged.
6. After the merge, trigger a site update to deploy the log level update. Please refer to the Upgrade Procedures here:
7. Collect the DEBUG log(s) and troubleshoot the issue.
8. After troubleshooting, set log levels back to INFO to prevent flooding the node with debug output. Repeat the steps in this section to set log level to INFO.
9. Verify the log level is set back to INFO.

## Node Provisioning Troubleshooting

A likely area for problems during a deployment is a failure to provision some portion of the nodes in the environment, usually due to hardware issues or other basic problems with PXE or networking in a new environment.

After executing the site deployment, logon to the MaaS dashboard to monitor node deployment progress. MaaS has several deployment states to be aware of, where the node can become stuck for different reasons.

### Node Discovery

On the first successful PXE boot to the MaaS server, the bare metal node will boot a discovery OS and register itself with the MaaS database. Most of the time, a node that can PXE successfully will also succeed in registering with the MaaS database. Failures to register may be caused by a missing NIC driver in the discovery OS kernel, or a failure in one of the MaaS components.

Also during this phase, the `freeipmi-tools` package is installed in order to run the ipmipower setup (ipmi-locate). This attempts to discover the IPMI information for this host and populate this in the MaaS database. If you encounter problems with IPMI, verify from the MaaS dashboard that the IPMI connection details for the node are correct, and that `IPMI over LAN` or equivalent option has been enabled in the node's BIOS. `IPMI over LAN` option allows the IPMI service to listen for IPMI connections on the iDRAC/iLO network interface.

After discovery, the node is promptly powered off by MaaS via IPMI.

**Note: A powered off node is the expected end-state for this phase of the MaaS lifecycle.**

This phase is an ideal opportunity to validate that all nodes are discovered. Look over the list of nodes in MaaS to identify any missing hardware. The expectation is that you should see a number of nodes registered in MaaS that equal the number of `BaremetalNode` defined in your site definition. Failure to discover nodes is usually due to a PXE boot problem or misconfiguration:

1. Basic network fabric communication failure or misconfiguration. The standard AIRSHIP design assumes PXE is untagged (native VLAN) on a dedicated PXE interface.
2. Portfast/Edgeport not configured. In this case it takes 30 seconds between the time the port gets a link established and the time it enters forwarding state. By then, the PXE agent has timed out and given up trying. You can test this by going to the iDRAC/iLO console for a node like Genesis that already has Ubuntu installed (or attach a virtual "live" Ubuntu CD otherwise), and run:

    - `$ ping $GATEWAY_IP` # Confirm you can ping any IP outside of this node, then Ctrl+C
    - `$ sudo ifdown $PXE_INTERFACE` # take down the PXE interface, then wait a few seconds
    - `$ sudo ifup $PXE_INTERFACE && time ping $GATEWAY_IP` # bring the PXE interface back up and run the ping again.

    As soon as you see pings go through, Ctrl+C and observe reported time. If the amount of time to ping is greater than 1 second, then your network fabric is not properly configured (it should be under one second). The PXE timeout depends on the NIC ROM installed, so an `ifup` time approaching 30 seconds is certainly a problem for PXE, whereas anything between 1 and 29 seconds is a cause for concern and investigation, but not necessarily going to cause outright

PXE failure. Intermittent PXE failure can also occur when the PXE timeout is too close or matching the `ifup` time, causing an inconsistent PXE success rate depending on your luck with timing, which is another reason to fix this issue even if you are able to PXE boot nodes after enough retries.

3. LACP fallback not configured. In environments where PXE traffic rides on the same interfaces that will participate in an LACP bond, the `LACP fallback` or equivalent setting must be enabled.
4. Server PXE booting off the wrong network interface. Check iDRAC/iLO console to see which NIC is used for PXE during boot.
5. Server not configured for PXE boot or was configured for PXE "next boot" but not persistently for subsequent boots. Check the iDRAC/iLO console to see what boot device the server is attempting.

## Node Commissioning

After a node has been discovered, it is then commissioned. Commissioning can only be initiated when the node is off. If MaaS will not allow the node to be commissioned because it is powered on, this suggests an IPMI problem (e.g., the node power-off that should have happened after step 1 above may not have occurred). Refer to the Troubleshooting IPMI Issues section in this cookbook to get more information.

When working properly, MaaS will power-on the node via IPMI and PXE boot it again. Again the discovery OS is loaded, however this time executing a set of MaaS commissioning scripts. Commissioning activities are recorded in MaaS logs for each node, and include:

- A one-time ntpdate to get a somewhat reasonably synced time (this is done so that log timestamps are roughly accurate, so it does not need to be precise).
- Gathering and reporting information to MaaS about the hardware (number of CPUs, number of disks and their sizes/partitions, number of NICs, etc.)
- Installing the Ubuntu package for LLDP
- Executing LLDP
- Possibly other custom commissioning scripts in the future (e.g., burn-in testing).

If a node fails commissioning, look in the MaaS commission logs of the failing node to determine the reason for commission failure. The following guidance is provided for problems related to the above-listed commissioning activities:

- If the node fails at `ntpdate`, ensure the right NTP server is configured and that it is reachable from the network you are using. The best test is to try ntpdate to the same time source from another node in the cluster, to verify that the right ports are open. For example, on Genesis: `sudo service ntp stop; ntpdate $MAAS_NTP_SERVER_IP; sudo service ntp restart`
- If the node fails in gathering hardware information, this could be due to a number of reasons. Possible reasons include problems between the discovery kernel and the hardware which manifest in kernel panics or hung kernel. Another possible reason could be corrupt or broken disks that stall the inventory process.
- If nodes systematically cannot install the Ubuntu package for LLDP, ensure that the upstream apt sources are correct and reachable from the environment. Compare with the package source list on Genesis. Ensure that your proxy settings are correct for your environment. If nodes randomly cannot install the Ubuntu package for LLDP, then it is possible there is a loading issue if there are too many nodes concurrently commissioning and/or deploying. In the case of suspected loading issues, try commissioning the nodes manually a few at a time from the MaaS dashboard.
- If the execution of LLDP fails, there could be an issue with the NIC drivers used by the discovery kernel. Examine the commission logs of the node for more details.

If there are commission failures with no node logs available, the basic network may not be functioning in the discovery OS used for commissioning. In this case, you should login to the iDRAC/iLO belonging to the node and watch the console as the commission attempt is made. Most of the commission output should be posted to the console. At this time, Drydock does not support configuring a custom MaaS pre-seed which would allow setting a password, so currently the only information you will have access to in this scenario is that which appears (sometimes very briefly) on the console.

If there are random commission timeouts on different nodes, this could be an indication of a loading issue. In this case, try to manually commission a smaller set of nodes to see if the problem persists. The total number of nodes being discovered, commissioned, and deployed by MaaS at any one time add to the total MaaS load.

In some cases, commission timeouts may still occur due to slow network (taking too long to fetch packages) or some other part of the commissioning process that is exceeding the default timeout (15 minutes). Unfortunately, MaaS does not presently allow configuration of this timeout, so it cannot be extended at this time.

One very good commissioning sanity check to perform is to look at the entire list of nodes, and compare the reported resources for CPU, memory, disk, and network. In particular, look for nodes with resource values that don't match the others, as this is a likely indicator of hardware problems that should be investigated now instead of when they cause unexpected problems later. Note the following:

- If you have nodes with non-matching CPU counts, then you likely have a heterogeneous mix of different types of hardware with different types of CPUs. In this case you should perform a careful audit of the hardware to ensure that you are booting the right nodes, as its expected all nodes to have matching specifications in Airship. If you are certain this is a valid exception, and you do intend to leverage multiple hardware profiles, you should ensure at this stage that the hardware profiles and allocated node labels in your site manifests match the resource reports from MaaS.
- Ensure that the `isolcpus` and `vcpu_pin_set` in your site manifests "agree" with the number of CPUs reported for data plane nodes in your environment. You should have a greater number of reported CPUs than the number of `isolcpus` listed for your data plane host profile(s). If you are using more than one data plane host profile, ensure that all profiles agree with MaaS resource reporting.
- If you have a small number of nodes reporting less RAM than the others, it is likely you have some bad RAM modules that need replacement. You can verify this by logging into the iDRAC/iLO of affected nodes and viewing the RAM health information. Report the nodes with bad RAM to data center personnel so they can be replaced. Nodes with less RAM than expected by the hardware profile will cause failure of hugepage allocation, which expects a specific number of 1G pages to be available. If you have a larger number of nodes with disagreeing RAM values, you may have non-homogeneous hardware; see CPU count disagreements above.
- Ensure that the number of `hugepages` in your site manifests "agree" with the amount of RAM reported for data plane nodes in your environment. You should have a greater amount of RAM reported (in GB) than the number of hugepages. If you are using more than one data-plane host profile, ensure that all profiles agree with MaaS resource reporting.
- If you have a small number of nodes with non-matching disk counts or disk capacity, it is likely that hardware RAID may be configured differently (or not configured) or that there are disk failures on these nodes. Login to the iDRAC/iLO of these nodes to check their disk health and have data center personnel replace any failed disks. (Ceph in particular will have issues if the expected number of disks are not present.) Also check the RAID configuration, and make any changes required to reconfigure RAID settings consistently for nodes of each kind. Note: Control plane nodes are expected to have different disk configuration from the data plane nodes, so you should see two different disk layouts. Ensure that your site manifest agrees with the nodes selected for control plane versus the ones selected for data plane based on the disk configurations (control plane nodes will have HDDs in JBOD for Ceph, whereas data plane nodes will have the HDDs in one hardware RAID array).

- Ensure that the disk configurations for hardware profiles in site manifests match the available disk resources reported by MaaS.
- If you have a small number of nodes with non-matching NIC counts, it is likely that these nodes may have NIC failures, or (in case of blades) that the blade chassis needs to be reseated. Login to iDRAC/iLO for these nodes to examine NIC health and report any failed or link-down NIC to data center personnel. Missing NICs should not cause a provisioning failure as long as it is an unused NIC or part of a bond that has at least one operational link.

Also note that nodes that report "0" for a resource likely have not completed commissioning, have failed commissioning, were released/rediscovered, or may have encountered some other commissioning-related problem. Updates to resource totals are made incrementally (e.g., first CPU, then memory, etc.) with some delay between each update for each node. In these cases you may need to wait for commissioning to complete or may want to try re-commissioning the node again to see if any erroneously reported resource information corrects itself.

By default, again MaaS will power off the node after successful commissioning.

## Node Deployment

After a node is successfully commissioned, it can then be deployed. When deploying a node, MaaS will PXE boot the node for the third time, deploy the final target OS, and do one final reboot that boots to local disk.

Deployments can fail for some of the same reasons as commissioning (see loading problems in previous section; also apt mirror availability). MaaS provides a node deployment log which contains more details of specific deployment failures.

## Hugepages Troubleshooting

If tenant instances (VMs) requiring hugepages are not scheduled and enabled as required, verify that Fuel hardware profiles (flavors) are configured correctly for hugepages.

**Note:** Operators and tenants cannot modify hugepages parameters. Operators and tenants can only determine whether a VM instance requires scheduling on a hugepage-enabled server.

### Specifying and Verifying Hugepages

#### *Hugepage Options in Boot Image*

Use this command to view the hugepages options used at boot time.

```
# cat /proc/cmdline
BOOT_IMAGE=/vmlinuz-4.15.0-34-generic root=UUID=9f426a59-1213-4c6d-
979e-84ff85c93327 ro hugepagesz=1G
hugepages=320 console=ttyS1,115200n8 cgroup_disable=hugetlb iommu=pt
transparent_hugepage=never isolcpus=4-43,48-
87 default_hugepagesz=1Gamd_iommu=on kernel_package=linux-image-
4.15.0-34-generic intel_iommu=on
```

This command shows what kernel command line was used to boot the system. Note, however, that the hugepage settings can be tweaked during runtime (but not by the operator or tenant) and may no longer reflect the boot time settings. To display the hugepages settings that are currently in effect, use the `sysctl` or `meminfo` utility.

#### *Specify Hugepages in Flavor*

Set hugepages in flavor:

```
# openstack flavor set [FLAVOR_ID] --property hw:mem_page_size=large
# openstack image set [IMAGE_ID]  --property hw_mem_page_size=1GB
```

Verify the flavor and image properties/settings:

```
# openstack flavor show [FLAVOR_ID]
# openstack image show [IMAGE_ID]
```

Verify the mem_page_size settings, disk, ram and vcpus:

- Disk, ram, and vcpus values must be numeric without any spaces or alpha characters. Values will be converted to integers.
- Ram and swap must be multiples of 1024 (swap can be empty).

For more information, see the OpenStack Hugepages Documentation.

#### *Verify Hugepage Usage*

```
# cat /proc/meminfo | grep Huge
AnonHugePages:         0 kB
ShmemHugePages:        0 kB
```

```
HugePages_Total:     320
HugePages_Free:      318
HugePages_Rsvd:        0
HugePages_Surp:        0
Hugepagesize:    1048576 kB
```

### *Verify Number of Hugepages*

```
# cat /proc/sys/vm/nr_hugepages
320
# sysctl vm.nr_hugepages
 = 320
```

### *Hugeadm Utility*

In addition, operators and tenants can use the `hugeadm` utility to list information related to hugepages. The `hugeadm` options are:

```
--pool-list
--list-all-mounts
--verbose <level>, -v
```

## Isolating Hugepage Problems

If you are having problems with hugepages, you may be able to isolate the problem by performing the following steps on the data plane node(s) which are suspected to have issues.

1. SSH to the bare metal node(s) where hugepages are expected to work
2. Run `sudo apt -y install hugepages`
3. Run `hugeadm --pool-list`. Sample output:

```
Size          Minimum    Current    Maximum    Default
1073741824        300        300        300          *
```

Then verify:

a. That the table contains one (and only one) row after the headers. If you have more than one, this means you have multiple page sizes configured and you need to adjust your kernel boot parameters such that it only defines one instance of `hugepagesz` that matches the page size for the Drydock hardware profile.
b. That the `Maximum` field above (number of pages) is greater than zero and matches the number of pages defined in the hardware profile.
c. That the hugepages maximum memory allocation does not exceed the amount of memory available on the node. Run `free -b` to see the amount of memory that the node has and compare with the hugepages requested output from the above table: `Size` x `Maximum` = Number of bytes allocated to hugepages. If the byte count for hugepages exceeds the amount of memory available on the node, it is likely you are either using the wrong hardware profile, the hardware profile needs to be updated with the hugepage count for the amount of RAM on your nodes, or this node has failed RAM modules or a failed CPU socket which has reduced the amount of total RAM available on this node. Another indication of insufficient memory is a `Current` page count less than the `Maximum`.

d. That the `Default` field in the above output displays an asterisk (`*`).
e. Ensure the mount job for hugepages is green on the node: `sudo systemctl status dev-hugepages.mount` and `hugeadm --list-all-mounts`. Expected output for 1G pagesize:

```
Mount Point        Options
/dev/hugepages     rw,relatime,pagesize=1024M
```

4. If `docker exec` is available for the `libvirt` pod (depends on current security policies), exec into the `libvirt` pod to verify:
a. `/etc/default/qemu-kvm` has `KVM_HUGEPAGES` set to a value of `1`.
b. `/dev/hugepages` exists and you have permissions to write there: `touch /dev/hugepages/foo; rm /dev/hugepages/foo`
c. You have quota to write a hugepage as follows (assuming 1G pagesize):

```
fallocate -o0 -l $((1024*1024)) /dev/hugepages/foo; rm
/dev/hugepages/foo
```

## Troubleshooting Time Sync Issues

On MaaS-provisioned nodes, ntpd is installed on the bare metal and can be checked by running the following:

```
ntpq -p
```

Re-run as needed until the `reach` is 377 for all of the time sources (i.e., the last eight polls to the NTP server were successful). A reach of 0 indicates a failed NTP server connection, whereas a reach greater than 0 and less than 377 likely indicates network flapping or a similar network reliability issue that should be resolved before proceeding.

For time sources with a `reach` of 377, ensure that the `offset` and `jitter` fields are less than 10.000 (milliseconds). For example:

```
.    remote           refid      st t when poll
reach   delay   offset  jitter
==============================================================================
========
+time.tritn.com  63.145.169.3     2
u   48   64  377   54.875    3.533   2.392
+mis.wci.com     216.218.254.202  2 u   53   64  377   73.954   -
2.089   2.538
*97-127-86-125.m .PPS.            1
u   43   64  377   24.638    0.122   2.686
```

If the offset is outside tolerance (10.000 ms) but the jitter is not, then the system time should eventually converge with UTC, and the offset should slowly reduce over this period.

However, if the jitter is outside tolerance, then this may indicate a problem with the time sources and/or the network connection to them, and the system time will not converge nor the offset be reduced. However, in production airship deployment, the jitter has converged by itself within ~2 hours from several hundred milliseconds down to <= 10ms. So, if this happens in your deployments, first try waiting to see if the issue resolves itself within a few hours.

Previously in cases where there was a large reported offset (exceeding the 1,000,000 ms ntpd panic threshold), a one-time correction with ntpdate can be executed. However, with the default Ubuntu configuration of ntpd, this is unnecessary because Canonical defaults **NTPD_OPTS='-g'** in **/etc/default/ntp**, which instructs the NTP daemon to do the same (ignore the panic threshold) on the initial startup of the NTP daemon. Generally speaking, with high round-trip latencies to time servers, trying to execute ntpdate-style time jumps where the **offset**s are less than twice the reported delays is counter-productive, and are better served by waiting for the NTP algorithm to converge to accurate time on its own, potentially over several hours.

## Troubleshooting IPMI Issues

### Manual Validation

Drydock already performs validation of IPMI credentials for bare metal nodes for which it has configuration data. However, if Drydock is not installed yet, or you want to perform a manual validation, follow these steps. This section also provides troubleshooting steps for how to resolve issues with non-working IPMI functionality on bare metal nodes.

First install needed utilities on the jump host:

```
    sudo apt -y install ipmitool nmap
```

Run `ipmitool` against each out-of-band interface defined in your site manifests, substituting the IP address, username, and password that are specified in them (**Note:** This assumes you can route to out-of-band IPs from the jump host):

```
    ipmitool -I lanplus -H <OOB_IP_ADDR> -U <USER> -P <PASSWORD> chassis
status
```

If successful, an output similar to the following should be received:

```
System Power         : on
Power Overload       : false
Power Interlock      : inactive
Main Power Fault     : false
Power Control Fault  : false
Power Restore Policy : always-off
Last Power Event     : command
Chassis Intrusion    : inactive
Front-Panel Lockout  : inactive
Drive Fault          : false
Cooling/Fan Fault    : false
Sleep Button Disable : not allowed
Diag Button Disable  : allowed
Reset Button Disable : not allowed
Power Button Disable : allowed
Sleep Button Disabled: false
Diag Button Disabled : true
Reset Button Disabled: false
Power Button Disabled: false
```

In this case, IPMI connection and authentication is working properly on this node. However, if unsuccessful, an output similar to the following may be received:

```
    Error: Unable to establish IPMI v2 / RMCP+ session
```

If this happens, check the access to the IPMI port on the target server as follows:

```
    sudo nmap -sU -p 623 $OOB_IP_ADDR
```

If `nmap` reports that the port is open, there is an authorization issue. Ensure your IPMI credentials are correct and that the node is running the latest firmware. If the issue persists, try performing an iDRAC/iLO restart, followed by a password reset.

**Important:** UDP scanning is prone to false positives.

If `nmap` reports that the port is closed, ensure that `IPMI over LAN` or equivalent is enabled for the target server. (This allows the IPMI service to bind to the same IP address used for iDRAC/iLO, but on a different port - UDP 623). Ensure the node is running the latest firmware. If the issue persists, try performing an iDRAC/iLO reset, and toggling the `IPMI over LAN` option off and on again.

### Other ipmitool Commands

```
# /usr/bin/ipmitool -I lanplus -H $ip  -U$user -P$password chassis
status
# /usr/bin/ipmitool -I lanplus -H $ip  -U$user -P$password sdr
# /usr/bin/ipmitool -I lanplus -H $ip  -U$user -P$password sel elist
# /usr/bin/ipmitool -I lanplus -H $ip  -U$user -P$password power
status
```

## Control Node Fails Ungracefully

If a control node fails ungracefully, meaning it is hard down outside of a teardown workflow, Kubernetes pods hosted on the node should automatically recover to other control nodes in the cluster. The exception to this is when Kubernetes does not automatically reschedule StatefulSet pods or pods backed by PVCs. Below are steps required to force Kubernetes to reschedule stateful pods.

**Note:** Kubernetes uses timers to decide when to start recovering lost pods so that a short-lived network blip does not cause significant churn in the cluster. When a control node fails, wait until it is marked `NotReady` in `kubectl get nodes` before beginning recovery efforts.

### Ensure Ceph Health

A defect in the Ceph chart can cause Ceph to enter an unhealthy state after a control node failure. Follow these steps.

1..Search for any `ceph-mon` pod with `NodeLost` status. Example:

```
root@n1:/var/log# kubectl get pods -n ceph | grep ceph-mon | grep
NodeLost
ceph-mon-
g59s2                               1/1       NodeLost           0
    49m
```

2. Force delete the pod if any. Example:

```
root@n1:/var/log# kubectl delete pod -n ceph ceph-mon-g59s2 --grace-
period=0 --force
```

*Example*

```
root@n1:/var/log# kubectl delete pod -n ceph ceph-mon-g59s2 --grace-
period=0 --force
warning: Immediate deletion does not wait for confirmation that the
running resource has
been terminated. The resource may continue to run on the cluster
indefinitely.
pod "ceph-mon-g59s2" deleted
root@n2:~# sleep 600 # Give ceph time to recover
root@n2:~# kubectl get pods -n ceph -o wide
NAME                                          READY     STATUS
  RESTARTS   AGE       IP              NODE
ceph-cephfs-provisioner-7dcdf7f4fb-
8qq5g   1/1       Running            0          17h       10.97.217.2
    n2
ceph-cephfs-provisioner-7dcdf7f4fb-
b2n6k   1/1       Unknown            0          18h       10.97.26.5
    n0
ceph-cephfs-provisioner-7dcdf7f4fb-
rckj9   1/1       Unknown            0          18h       10.97.26.4
    n0
ceph-cephfs-provisioner-7dcdf7f4fb-
```

```
smjhc   1/1       Running            0          17h       10.97.217.4
    n2
ceph-mds-5f94dd77f5-
pvdjw                  1/1       Unknown            0          18h
   10.97.26.24     n0
ceph-mds-5f94dd77f5-
xmmwn                  1/1       Running            3          17h
   10.97.217.6     n2
ceph-mgr-7d7fbff7bb-
5ffdx                  0/1       CrashLoopBackOff   212        17h
   192.168.77.12   n2
ceph-mgr-7d7fbff7bb-
scbw2                  1/1       Unknown            1          18h
   192.168.77.10   n0
ceph-mon-check-5777b7fcb4-
c4zj7            1/1       Unknown            0          18h       10.
97.26.22     n0
ceph-mon-check-5777b7fcb4-
djkf7            1/1       Running            0          17h       10.
97.217.15    n2
ceph-mon-
gsqtm                             1/1       Running            12
    18h       192.168.77.11   n1
ceph-mon-
gwhl2                             0/1       Pending            0
    17h       <none>          n0
ceph-mon-
xrblp                             1/1       Running            2
    18h       192.168.77.12   n2
ceph-osd-default-83945928-
fnz5c            1/1       Running            0          18h       192
.168.77.11   n1
ceph-osd-default-83945928-
fwsbd            1/1       NodeLost           0          18h       192
.168.77.10   n0
ceph-osd-default-83945928-
thb7n            1/1       Running            0          18h       192
.168.77.12   n2
ceph-rbd-provisioner-cd77dbb65-
6ncqr       1/1       Unknown            0          18h       10.97.26
.10     n0
ceph-rbd-provisioner-cd77dbb65-
brc86       1/1       Unknown            0          18h       10.97.26
.15     n0
ceph-rbd-provisioner-cd77dbb65-
sqvhb       1/1       Running            0          17h       10.97.21
7.3     n2
ceph-rbd-provisioner-cd77dbb65-
w9hhf       1/1       Running            0          17h       10.97.21
7.9     n2
```

The `ceph-mgr CrashLoop` should not impact the ability to recover PVC-backed pods. Below is a short script that should output a list of pods that will need to be manually deleted.

```
OIFS=$IFS
IFS=$'\n'
NODENAME='n0' # Set to the genesis node hostname
for pod in $(kubectl get pods --all-namespaces -o wide | grep -
v NAMESPACE | tr -s ' ')
do
  pod_name=$(echo $pod | cut -d ' ' -f 2)
  pod_ns=$(echo $pod | cut -d ' ' -f 1)
  pod_node=$(echo $pod | cut -d ' ' -f 8)
  if [[ $pod_node == $NODENAME ]]
  then
    pod_valid=$(kubectl describe pod -n $pod_ns $pod_name | grep -
E '(Type: +PersistentVolumeClaim)|(Created By: +StatefulSet)')
    if [[ ! -z $pod_valid ]]
    then
      echo "${pod_ns}/${pod_name}"
    fi
  fi
done
IFS=$OIFS
```

Example output:

```
    ucp/airflow-worker-0 ucp/airflow-worker-1 ucp/maas-region-0
ucp/mariadb-0 ucp/postgresql-0 ucp/ucp-rabbitmq-rabbitmq-0
```

For each of the above pods, they will need to be force deleted using the `` `kubectl delete pod --grace-period=0 --force ...` `` command. After deletion, Kubernetes recreates the pod on an available node with the correct labels. If a newly created pod stays in `Pending` state, that likely means that there are no available nodes with the correct labels. The exception to this is `` `maas-region-0` `` which currently cannot recover to another node. It should be left in `Unknown` state. Node deployments will be unavailable until the Genesis node is recovered. Due to dependencies, databases should be deleted first. Then delete the message bus RabbitMQ. Once they have been rescheduled and are `Ready`, the rest of the pods can be deleted.

```
root@n2:~# kubectl delete pod -n ucp postgresql-0 --grace-period=0 --
force
warning: Immediate deletion does not wait for confirmation that the
running resource has
been terminated. The resource may continue to run on the cluster
indefinitely.
pod "postgresql-0" deleted
root@n2:~# kubectl delete pod -n ucp mariadb-0 --grace-period=0 --
force
warning: Immediate deletion does not wait for confirmation that the
running resource has
been terminated. The resource may continue to run on the cluster
indefinitely.
pod "mariadb-0" deleted
root@n2:~# kubectl get pods -n ucp postgresql-0 -o wide
NAME           READY     STATUS    RESTARTS   AGE       IP
```

```
NODE
postgresql-
0   1/1       Running   0          1m        10.97.40.131   n1
root@n2:~# kubectl get pods -n ucp mariadb-0 -o wide
NAME        READY     STATUS    RESTARTS   AGE       IP             NO
DE
mariadb-0   1/1       Running   0          1m        10.97.40.132   n1
root@n2:~# kubectl delete pod -n ucp ucp-rabbitmq-rabbitmq-0 --grace-
period=0 --force
warning: Immediate deletion does not wait for confirmation that the
running resource has
been terminated. The resource may continue to run on the cluster
indefinitely.
pod "ucp-rabbitmq-rabbitmq-0" deleted
root@n2:~# kubectl get pods -o wide -n ucp ucp-rabbitmq-rabbitmq-0
NAME                     READY     STATUS    RESTARTS   AGE       IP
           NODE
ucp-rabbitmq-rabbitmq-
0   1/1       Running   0          57s       10.97.40.133   n1
root@n2:~# kubectl delete pod -n ucp airflow-worker-0 --grace-period=0
--force
warning: Immediate deletion does not wait for confirmation that the
running resource has
been terminated. The resource may continue to run on the cluster
indefinitely.
pod "airflow-worker-0" deleted
root@n2:~# kubectl get pods -n ucp airflow-worker-0 -o wide
NAME              READY     STATUS    RESTARTS   AGE       IP
     NODE
airflow-worker-
0   2/2       Running   0          12m       10.97.40.130   n1
root@n2:~# kubectl delete pod -n ucp airflow-worker-1 --grace-period=0
--force
warning: Immediate deletion does not wait for confirmation that the
running resource has
been terminated. The resource may continue to run on the cluster
indefinitely.
pod "airflow-worker-1" deleted
root@n2:~# kubectl get pods -n ucp airflow-worker-1 -o wide
NAME              READY     STATUS    RESTARTS   AGE       IP
     NODE
airflow-worker-
1   2/2       Running   0          1m        10.97.40.134   n1
```

At this point all the UCP services (excluding MaaS) begin to recover as the StatefulSets have been recovered.

```
root@n2:~# kubectl get pods -n ucp -o wide
NAME                                          READY     STATUS     RE
STARTS   AGE       IP              NODE
airflow-flower-8cc44cc6d-
79x44                1/1       Running    0          18h       10.97.
217.8     n2
```

```
airflow-flower-8cc44cc6d-
j2nkj                 1/1       Unknown    0          19h       10.97.
26.17     n0
airflow-flower-8cc44cc6d-
sj44d                 1/1       Unknown    0          19h       10.97.
26.40     n0
airflow-flower-8cc44cc6d-
vzfll                 1/1       Running    0          18h       10.97.
217.16    n2
airflow-scheduler-f8468f569-
nnfwt               1/1       Unknown    0          19h       10.97.26.
37     n0
airflow-scheduler-f8468f569-
qg4dm               1/1       Running    0          18h       10.97.217
.19    n2
airflow-scheduler-f8468f569-
wdsr7               1/1       Running    0          18h       10.97.217
.10    n2
airflow-scheduler-f8468f569-
xwwzx               1/1       Unknown    0          19h       10.97.26.
20     n0
airflow-web-58fd6b6c59-
7t4cv                   1/1       Unknown    0          19h       10.9
7.26.32     n0
airflow-web-58fd6b6c59-
glwxb                   1/1       Running    0          18h       10.9
7.217.22    n2
airflow-web-58fd6b6c59-
t6sr8                   1/1       Running    0          18h       10.9
7.217.11    n2
airflow-web-58fd6b6c59-
zjvqc                   1/1       Unknown    0          19h       10.9
7.26.45     n0
airflow-worker-
0                               2/2       Running    0          20m
    10.97.40.130    n1
airflow-worker-
1                               2/2       Running    0          6m
    10.97.40.134    n1
armada-api-67fcd6c847-
khm6t                    1/1       Unknown    0          19h       10.
97.26.57     n0
armada-api-67fcd6c847-
pqhks                    1/1       Running    0          18h       10.
97.217.12    n2
barbican-api-57bbb49445-
llz2f                   1/1       Running    0          18h       10.97
.217.14    n2
barbican-api-57bbb49445-
w224k                   1/1       Unknown    0          19h       10.97
.26.35     n0
deckhand-54f5d99769-
```

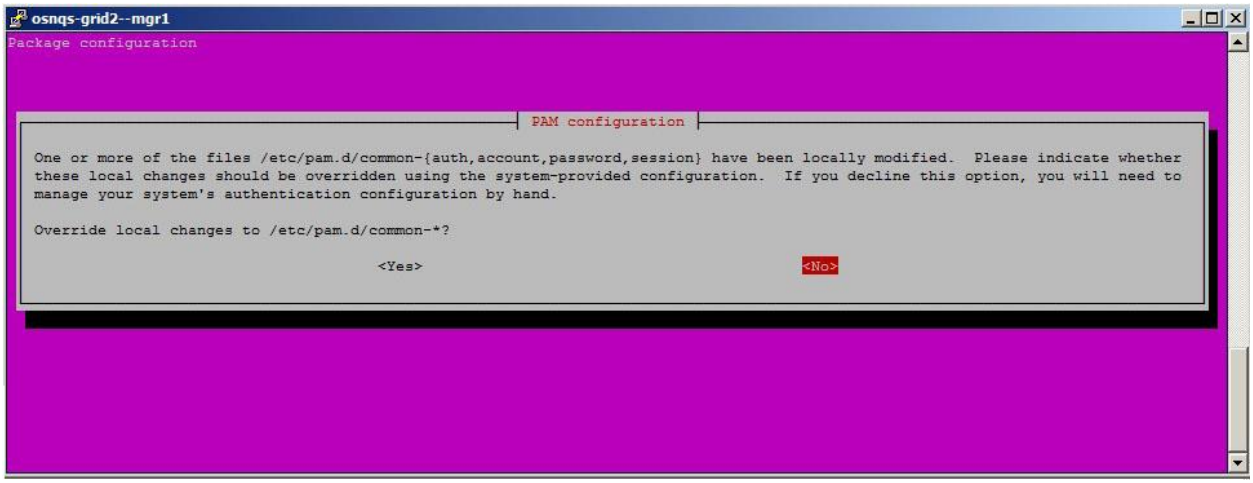
```
79bcj                        1/1       Unknown    0          19h       1
0.97.26.59     n0
deckhand-54f5d99769-
spwgl                        1/1       Running    0          18h       1
0.97.217.17    n2
drydock-api-658b4b4647-
6b9xj                    1/1       Running    0          18h       10.9
7.217.26    n2
drydock-api-658b4b4647-
6t6sn                    1/1       Unknown    0          19h       10.9
7.26.12     n0
drydock-api-658b4b4647-
b268n                    1/1       Running    0          18h       10.9
7.217.18    n2
drydock-api-658b4b4647-
kvbhx                    1/1       Unknown    0          19h       10.9
7.26.61     n0
ingress-5cd9d498b-
6nq9h                        1/1       Running    0          18h
10.97.217.20    n2
ingress-5cd9d498b-
cvq8j                        1/1       Unknown    0          19h
10.97.26.43     n0
ingress-error-pages-576687ff74-
7x72s           1/1       Running    0          18h       10.97.217.27
    n2
ingress-error-pages-576687ff74-
hn7kk           1/1       Unknown    0          19h       10.97.26.36
    n0
keystone-api-5cffcf87d-
qxxwk                    1/1       Unknown    0          19h       10.9
7.26.28     n0
keystone-api-5cffcf87d-
x7jxw                    1/1       Running    0          18h       10.9
7.217.21    n2
maas-rack-5b4b8896db-
chrpz                       1/1       Unknown    0          19h       19
2.168.77.10   n0
maas-rack-5b4b8896db-
frkp8                       0/1       Init:0/1   0          18h       19
2.168.77.12   n2
maas-region-
0                                  1/1       Unknown    0          19h
       10.97.26.42     n0
mariadb-
0                                     1/1       Running    0
11m       10.97.40.132    n1
postgresql-
0                                   1/1       Running    0          15
m       10.97.40.131    n1
promenade-api-6958b66f5c-
nbstq                1/1       Unknown    0          19h       10.97.
```

```
26.8      n0
promenade-api-6958b66f5c-
sr8ts                 1/1       Running    0          18h       10.97.
217.23    n2
shipyard-6559fd7c58-
7l5lp                       1/1       Unknown    0          19h       1
0.97.26.33     n0
shipyard-6559fd7c58-
fxmx5                       1/1       Running    0          18h       1
0.97.217.28    n2
shipyard-6559fd7c58-
mr4g7                       1/1       Unknown    0          19h       1
0.97.26.51     n0
shipyard-6559fd7c58-
zpnz8                       1/1       Running    0          18h       1
0.97.217.24    n2
ucp-rabbitmq-rabbitmq-
0                         1/1       Running    0          10m       10.
97.40.133    n1
ucp-ucp-memcached-memcached-5b4dc6fd85-
kgzts   1/1       Running    0          18h       10.97.217.25    n2
ucp-ucp-memcached-memcached-5b4dc6fd85-
s4nqw   1/1       Unknown    0          19h       10.97.26.26     n0
```

If there is a service layer (such as Openstack Helm) also running, continue deleting the pods from that namespace that the script above listed.

## Troubleshooting apt on BareMetal Nodes

Occasionally, some scripts that attempt `apt` operations on BareMetal nodes get stuck and never complete because they are waiting for human input to the following prompt:




This is a result of gstools modifications made to `/etc/pam.d` settings on these nodes.

When this happens, open a bug report against the component that is performing the `apt` or `apt-get` operation. They need to use/set **DEBIAN_FRONTEND=noninteractive** in order for this prompt to be dismissed, and for `apt` not to become stuck waiting for user input.

## Ceph Troubleshooting

### Initial Troubleshooting

When a Ceph cluster is up and running, there are many built-in features that allow for I/O operations to continue in the event of a software or hardware failure. However, there may be occurrences where administrators need to troubleshoot issues affecting the Ceph cluster.

Many of the commands in this document require the name of the Ceph monitor pod. Use the following shell command to assign the pod name to the `MON_POD` environment variable.

```
MON_POD=$(sudo kubectl get --no-headers pods -n=ceph -
l="application=ceph,component=mon" | awk '{ print $1; exit }')
```

### Identifying Problems

To determine possible causes of an  error with  Ceph, answer the following question:

- Certain problems can arise when using unsupported configurations. Are you running a configuration that is supported ?
- Do you know which Ceph component caused the problem?

### Diagnosing the Health of a Ceph Storage Cluster

This procedure lists basic steps to diagnose the health of a Ceph Storage Cluster.

Check the overall status of the cluster.

```
kubectl -n ceph exec ${MON_POD} ceph health detail
```

Check the Ceph logs for any error messages listed.

If the logs do not include sufficient amount of information, increase the debugging level and try to reproduce the action that failed.

### Understanding the Output of the ceph health Command

The following command returns information about the status of the Ceph Storage Cluster:

```
kubectl -n ceph exec ${MON_POD} ceph health
```

Health indicators are:

- HEALTH_OK indicates that the cluster is healthy.
- HEALTH_WARN indicates a warning but all the data stored in the cluster remains accessible. In some cases, the Ceph status returns to HEALTH_OK automatically, for example when Ceph finishes the rebalancing process.
- HEALTH_ERR indicates a more serious problem that requires your immediate attention as a part or all of your data has become inaccessible.

Use these commands to get more detailed information:

```
kubectl -n ceph exec ${MON_POD} ceph health detail
kubectl exec -n ceph ${MON_POD} ceph -s
```

Messages related to monitors

- "mon.X is down (out of quorum)"
- "clock skew"
- "store is getting too big!"

Messages related to osds

- "full osds"
- "nearfull osds"
- "osds are down"
- "requests are blocked"
- "slow requests"

Messages related to PGS

- "pgs down"
- "pgs inconsistent"
- "scrub errors"
- "pgps stale"
- "unfound"

## Understanding Ceph Logs

By default, Ceph stores its logs in the /var/log/ceph/ directory.

The <cluster-name>.log is the main cluster log file that includes the global cluster events. By default, this log is named ceph.log. Only the Monitor hosts include the main cluster log.

The audit.log is the cluster audit log file that keeps the trace of all commands issued against the cluster and received by the Monitors. Only the Monitor hosts include the main cluster log.

Each OSD and Monitor has its own log file, named <cluster-name>-osd.<number>.log and <cluster-name>-mon.<hostname>.log. When you increase debugging level for Ceph subsystems, Ceph generates a new log files for those subsystems as well.

The following list contains the most common Ceph log error messages related to Monitors and OSDs.

### *Common Error Messages in Ceph Logs Related to Monitors*

- clock skew
- clocks not synchronized
- Corruption: 1 missing files
- Caught signal (Bus error)

### *Common Error Messages in Ceph Logs Related to OSDs*

- heartbeat_check: no reply from osd.X
- wrongly marked me down
- osds have slow requests

- FAILED assert(!m_filestore_fail_eio)
- FAILED assert(0 == "hit suicide timeout")

### *Enabling Rados Debug Logging*

To enable debug logs in Rados gateway, add the following in /etc/ceph/ceph.conf of Rados gateway node and restart the rgw service.

```
debug ms = 1
debug rgw = 20
```

The logs are written to /var/log/radosgw/ on the Rados Gateway node.

## Displaying Rados Block Device Size

To display RBD size, use this command:

```
rbd ls | while read rbd; do echo $rbd; rbd info $rbd | grep size; done
```

To display usage details based on RBD size, use this command:

```
rbd ls | while read rbd; do echo $rbd; rbd info $rbd | grep size; done
| grep -B1 "1.8 TiB" | grep kubernetes | while read rbd; do rbd disk-
usage $rbd; done
```

## Troubleshooting Monitors

You should keep in mind that losing a monitor, or a bunch of them, doesn't necessarily mean that your cluster is down, as long as a majority of them remains up, running and with a formed quorum. Regardless of how bad the situation is, the first thing you should do is to calm down, take a breath and try answering our initial troubleshooting script.

### Initial Troubleshooting

#### *Are the monitors running?*

##### Diagnostic

```
kubectl -n ceph get pods -l application=ceph,component=mon
```

##### Action if a MON is down

- Verify the node labels
- Inspect the POD log
- Inspect the MON log in /var/log/ceph/ceph-mon.$id.log

#### *Are you able to connect to the monitor's servers?*

##### Diagnostic

Repeat this command for all MONs to find out about each of them:

```
kubectl -n ceph exec {utility-POD} ceph -m {IP_of_MON} -s
```

##### Action if a MON cannot be contacted

1. Verify the network interfaces on the MON node
2. Verify the firewall rules on the MON node
3. Inspect the MON log in /var/log/ceph/ceph-mon.$id.log
4. Verify that the MON is running

#### Is a Quorum Present on the Cluster?

To determine if a quorum is up and running on the cluster, execute this command:

```
kubectl -n ceph exec ${MON_POD} ceph -s
```

If the cluster replies to this command, the cluster is up and running. The monitors will only answer to a status request if there is a formed quorum. If the cluster does not respond or returns a lot of fault messages, see below.

#### *What if ceph -s Blocks or Returns Fault Messages?*

If the command in the preceding section blocks or returns a lot of fault messages, then it is likely that your monitors are either down completely or just a portion is up -- a portion that is not enough to form a quorum (keep in mind that a quorum if formed by a majority of monitors).

If you haven't gone through all the steps so far, please go back and complete them.

For those running on Emperor 0.72-rc1 and forward, you will be able to contact each monitor individually asking them for their status, regardless of a quorum being formed. This can be achieved using `ceph ping mon.ID`, ID being the monitor's identifier. You should perform this for each monitor in the cluster.

## Understanding mon_status

mon_status can be obtained through the ceph tool when you have a formed quorum. This command will output a multitude of information about the monitor, including the same output you would get with quorum_status.

Take the following example of mon_status:

```
{ "name": "c",
  "rank": 2,
  "state": "peon",
  "election_epoch": 38,
  "quorum": [
        1,
        2],
  "outside_quorum": [],
  "extra_probe_peers": [],
  "sync_provider": [],
  "monmap": { "epoch": 3,
      "fsid": "5c4e9d53-e2e1-478a-8061-f543f8be4cf8",
      "modified": "2013-10-30 04:12:01.945629",
      "created": "2013-10-29 14:14:41.914786",
      "mons": [
            { "rank": 0,
              "name": "a",
              "addr": "127.0.0.1:6789\/0"},
            { "rank": 1,
              "name": "b",
              "addr": "127.0.0.1:6790\/0"},
            { "rank": 2,
              "name": "c",
              "addr": "127.0.0.1:6795\/0"}]}}
```

A couple of things are obvious: there are three monitors in the monmap (*a*, *b* and *c*), the quorum is formed by only two monitors, and *c* is in the quorum as a *peon*.

Which monitor is out of the quorum?

The answer would be **a**.

Why?

Take a look at the quorum set. There are two monitors in this set: *1* and *2*. These are not monitor names. These are monitor ranks, as established in the current monmap. The monitor with rank 0 is missing and according to the monmap that would be mon.a.

By the way, how are ranks established?

Ranks are (re)calculated whenever you add or remove monitors and follow a simple rule: the **greater** the IP:PORT combination, the **lower** the rank is. In this case, considering that 127.0.0.1:6789 is lower than all the remaining IP:PORT combinations, mon.a has rank 0.

## Most Common Monitor Issues

### *Have Quorum but at least one Monitor is down*

When this happens, depending on the version of Ceph you are running, you should be seeing something similar to:

```
$  kubectl -n ceph exec  ${MON_POD}  ceph health detail[snip]mon.a (rank 0) addr 127.0.0.1:6789/0 is down (out of quorum)
```

#### How to troubleshoot this?

First, make sure mon.a is running. Secondly, make sure you are able to connect to mon.a's server from the other monitors' servers. Check the ports as well. Check iptables on all your monitor nodes and make sure you are not dropping/rejecting connections.

If this initial troubleshooting doesn't solve your problems, then it's time to go deeper.

First, check the problematic monitor's mon_status.

Considering the monitor is out of the quorum, its state should be one of probing, electing or synchronizing. If it happens to be either leader or peon, then the monitor believes to be in quorum, while the remaining cluster is sure it is not; or maybe it got into the quorum while troubleshooting of monitor goes on, so check you ceph -s again just to make sure. Proceed if the monitor is not yet in the quorum.

#### What if the state is probing?

This means the monitor is still looking for the other monitors. Every time you start a monitor, the monitor will stay in this state for some time while trying to find the rest of the monitors specified in the monmap. The time a monitor will spend in this state can vary. For instance, when on a single-monitor cluster, the monitor will pass through the probing state almost instantaneously, since there are no other monitors around. On a multi-monitor cluster, the monitors will stay in this state until they find enough monitors to form a quorum -- this means that if you have 2 out of 3 monitors down, the one remaining monitor will stay in this state indefinitely until you bring one of the other monitors up.

If you have a quorum, however, the monitor should be able to find the remaining monitors pretty fast, as long as they can be reached. If your monitor is stuck probing and you have gone through with all the communication troubleshooting, then there is a fair chance that the monitor is trying to reach the other monitors on a wrong address. mon_status outputs the monmap known to the monitor: check if the other monitor's locations match reality.

#### What if state is electing?

This means the monitor is in the middle of an election. These should be fast to complete, but at times the monitors can get stuck electing. This is usually a sign of a clock skew among the monitor nodes. If all your clocks are properly synchronized, it is best if you prepare some logs and reach out to the community. This is not a state that is likely to persist and aside from (*really*) old bugs there is not an obvious reason besides clock skews on why this would happen.

#### What if state is synchronizing?

This means the monitor is synchronizing with the rest of the cluster in order to join the quorum. The synchronization process is as faster as smaller your monitor store is, so if you have a big store it may take a while. Don't worry, it should be finished soon enough.

However, if you notice that the monitor jumps from synchronizing to electing and then back to synchronizing, then you do have a problem: the cluster state is advancing (i.e., generating new

maps) way too fast for the synchronization process to keep up. This used to be a thing in early Cuttlefish, but since then the synchronization process was quite refactored and enhanced to avoid just this sort of behavior.

### What if state is leader or peon?

This should not happen. There is a chance this might happen however, and it has a lot to do with clock skews

### Recovering a Monitor's Broken monmap

This is how a monmap usually looks like, depending on the number of monitors:

```
epoch 3
fsid 5c4e9d53-e2e1-478a-8061-f543f8be4cf8
last_changed 2013-10-30 04:12:01.945629
created 2013-10-29 14:14:41.914786
0: 127.0.0.1:6789/0 mon.a
1: 127.0.0.1:6790/0 mon.b
2: 127.0.0.1:6795/0 mon.c
```

This may not be what you have however. You may end up with a monitor with a severely outdated monmap, thus being unable to find the remaining monitors (e.g., say mon.c is down; you add a new monitor mon.d, then remove mon.a, then add a new monitor mon.e and remove mon.b; you will end up with a totally different monmap from the one mon.c knows).

In this sort of situations, you have two possible solutions:

- Scrap the monitor and create a new one. See Removing a Monitor. See Adding a Monitor
- Inject a valid monmap into the monitor. Not practical in a Kubernetes environment (for now unless I find a way to stop and restart the MON).

### *Clock Skews*

Monitors can be severely affected by significant clock skews across the monitor nodes. This usually translates into weird behavior with no obvious cause. To avoid such issues, you should run a clock synchronization tool on your monitor nodes.

#### What's the maximum tolerated clock skew?

By default, the monitors will allow clocks to drift up to 0.05 seconds.

#### Can I increase the maximum tolerated clock skew?

This value is configurable via the mon-clock-drift-allowed option, and although you *CAN* it doesn't mean you *SHOULD*. The clock skew mechanism is in place because clock skewed monitor may not properly behave. We, as developers and QA aficionados, are comfortable with the current default value, as it will alert the user before the monitors get out hand. Changing this value without testing it first may cause unforeseen effects on the stability of the monitors and overall cluster healthiness, although there is no risk of data loss.

#### How do I know there's a clock skew?

The monitors will warn you in the form of a HEALTH_WARN. ceph health detail should show something in the form of:

```
mon.c addr 10.10.0.1:6789/0 clock skew 0.08235s > max 0.05s (latency 0.0045s)
```

That means that mon.c has been flagged as suffering from a clock skew.

#### What should I do if there's a clock skew?

Synchronize your clocks. Running an NTP client may help. If you are already using one and you hit this sort of issues, check if you are using some NTP server remote to your network and consider hosting your own NTP server on your network. This last option tends to reduce the amount of issues with monitor clock skews.

### *Client Can't Connect or Mount*

Check your IP tables. Some OS install utilities add a REJECT rule to iptables. The rule rejects all clients trying to connect to the host except for ssh. If your monitor host's IP tables have such a REJECT rule in place, clients connecting from a separate node will fail to mount with a timeout error. You need to address iptables rules that reject clients trying to connect to Ceph daemons. For example, you would need to address rules that look like this appropriately:

```
REJECT all -- anywhere anywhere reject-with icmp-host-prohibited
```

You may also need to add rules to iptables on your Ceph hosts to ensure that clients can access the ports associated with your Ceph monitors (i.e., port 6789 by default) and Ceph OSDs (i.e., 6800 through 7300 by default). For example:

```
iptables -A INPUT -m multiport -p tcp -s {ip-address}/{netmask} --dports
6789,6800:7300 -j ACCEPT
```

## Monitor Store Failures

### *Symptoms of store corruption*

Ceph monitor stores the cluster map in a key/value store. Versions up to Jewel will use LevelDB while starting with Luminous the default is RocksDB. Both defaults are for freshly installed clusters. If a monitor fails due to a key/value store corruption, the following error messages might be found in the monitor log:

```
Corruption: error in middle of record
```

or:

```
Corruption: 1 missing files; e.g.:
/var/lib/ceph/mon/mon.0/store.db/1234567.ldb
```

### *Recovery using healthy monitor(s)*

If there is any survivors, we can always replace the corrupted one with a new one. And after booting up, the new joiner will sync up with a healthy peer, and once it is fully synchronized, it will be able to serve the clients.

### *Everything Failed! Now What?*

### *Preparing your logs*

Monitor logs are, by default, kept in /var/log/ceph/ceph-mon.$id.log*. We may want them. However, your logs may not have the necessary information. If you don't find your monitor logs at their default location, you can check where they should be by running:

```
kubectl exec -n ceph ${MON_POD} -- ceph-conf --name mon.FOO --show-config-
value log_file
```

The amount of information in the logs are subject to the debug levels being enforced by your configuration files. If you have not enforced a specific debug level then Ceph is using the default levels and your logs may not contain important information to track down you issue. A first step in getting relevant information into your logs will be to raise debug levels. In this case we will be interested in the information from the monitor. Similarly, to what happens on other components, different parts of the monitor will output their debug information on different subsystems.

You will have to raise the debug levels of those subsystems more closely related to your issue. This may not be an easy task for someone unfamiliar with troubleshooting Ceph. For most situations, setting the following options on your monitors will be enough to pinpoint a potential source of the issue:

```
debug mon = 10debug ms = 1
```

If we find that these debug levels are not enough, there's a chance we may ask you to raise them or even define other debug subsystems to obtain information from – but at least we started off with some useful information, instead of a massively empty log without much to go on with.

### *Do I need to restart a monitor to adjust debug levels?*

No. You may do it in one of two ways:

#### You have quorum

Either inject the debug option into the monitor you want to debug or into all monitors at once. To inject into only the one monitor, execute this command:

```
kubectl -n ceph exec  ${MON_POD} ceph tell mon.{id} config set debug_mon
10/10
```

To inject into all monitors at once, execute this command:

```
kubectl -n ceph exec  ${MON_POD}  ceph tell mon.* config set debug_mon 10/10
```

#### No quorum

Use the monitor's admin socket and directly adjust the configuration options:

```
kubectl -n ceph exec  ${MON_POD}  ceph daemon mon.{id} config set debug_mon
10/10
```

Going back to default values is as easy as rerunning the above commands using the debug level 1/10 instead. You can check your current values using the admin socket and the following commands:

```
kubectl -n ceph exec  ${MON_POD}  ceph daemon mon.{id} config show
```

or

```
kubectl -n ceph exec  ${MON_POD}  ceph daemon mon.FOO config get
'OPTION_NAME'
```

### *Reproduced the problem with appropriate debug levels. Now what?*

Ideally you would send to support only the relevant portions of your logs. Your support organization knows that figuring out the corresponding portion may not be the easiest of tasks. Therefore, they won't hold it to you if you provide the full log, but common sense should be employed. If your log has hundreds of thousands of lines, it may get tricky to go through the whole thing, especially if support is not aware at which point, whatever your issue is, happened. For instance, when reproducing, keep

in mind to write down current time and date and to extract the relevant portions of your logs based on that.

## Split-Brain

Since a monitor operates in the quorum, more than half of the total configured monitors should always be available to maintain access to the cluster and prevent split-brain problems using an odd number of configured monitors. Out of all the cluster monitors, one of them operates as the leader. The other monitor nodes are entitled to become leaders if the leader monitor is unavailable. A production cluster must have at least three monitor nodes to provide high availability and prevent split brain situations. Moreover, if you shut down all monitors at the same time, Ceph clients cannot read or write data.

In the event a monitor needs to be added, please see instructions for adding a Ceph monitor.

## Troubleshooting OSDs

Before troubleshooting your OSDs, check your monitors and network first. If you execute ceph health or ceph -s on the command line and Ceph returns a health status, it means that the monitors have a quorum. If you don't have a monitor quorum or if there are errors with the monitor status, address the monitor issues first. Check your networks to ensure they are running properly, because networks may have a significant impact on OSD operation and performance.

### Obtaining Data About OSDs

A good first step in troubleshooting your OSDs is to obtain information in addition to the information you collected while monitoring your OSDs (e.g., ceph osd tree).

### Ceph Logs

If you haven't changed the default path, you can find Ceph log files at /var/log/ceph:

```
kubectl -n ceph exec  ${MON_POD}  ls /var/log/ceph
```

If you don't get enough log detail, you can change your logging level.

### Admin Socket

Use the admin socket tool to retrieve runtime information. For details, list the sockets for your Ceph processes:

```
kubectl -n ceph exec  ${MON_POD}  ls /var/run/ceph
```

Then, execute the following, replacing {daemon-name} with an actual daemon (e.g., osd.0):

```
kubectl -n ceph exec  {DAEMON_POD} ceph daemon osd.0 help
```

Alternatively, you can specify a {socket-file} (e.g., something in /var/run/ceph):

```
kubectl -n ceph exec  {OSD-POD} ceph daemon {socket-file} help
```

The admin socket, among other things, allows you to:

- List your configuration at runtime
- Dump historic operations
- Dump the operation priority queue state
- Dump operations in flight
- Dump perfcounters

### Stopping without Rebalancing

Periodically, you may need to perform maintenance on a subset of your cluster or resolve a problem that affects a failure domain (e.g., a rack). If you do not want CRUSH to automatically rebalance the cluster as you stop OSDs for maintenance, set the cluster to noout first

```
kubectl exec  -n ceph ${MON_POD} ceph osd set noout
```

Once the cluster is set to noout, you can begin stopping the OSDs within the failure domain that requires maintenance work.

```
kubectl exec  -n ceph ${MON_POD}  stop ceph-osd id={num}
```

**Note:** Placement groups within the OSDs you stop will become degraded while you are addressing issues within the failure domain.

Once you have completed your maintenance, restart the OSDs.

```
kubectl exec  -n ceph ${MON_POD} start ceph-osd id={num}
```

Finally, you must unset the cluster from noout.

```
kubectl exec  -n ceph ${MON_POD} ceph osd unset noout
```

## OSD Not Running

Under normal circumstances, simply restarting the ceph-osd daemon will allow it to rejoin the cluster and recover.

### *An OSD Won't Start*

If you start your cluster and an OSD won't start, check the following:

- **Configuration File:** If you were not able to get OSDs running from a new installation, check your configuration file to ensure it conforms (e.g., host not hostname, etc.).
- **Check Paths:** Check the paths in your configuration, and the actual paths themselves for data and journals. If you separate the OSD data from the journal data and there are errors in your configuration file or in the actual mounts, you may have trouble starting OSDs. If you want to store the journal on a block device, you should partition your journal disk and assign one partition per OSD.
- **Check Max Thread count:** If you have a node with a lot of OSDs, you may be hitting the default maximum number of threads (e.g., usually 32k), especially during recovery. You can increase the number of threads using sysctl to see if increasing the maximum number of threads to the maximum possible number of threads allowed (i.e., 4194303) will help. For example:

```
    sysctl -w kernel.pid_max=4194303
```

If increasing the maximum thread count resolves the issue, you can make itpermanent by including a ``kernel.pid_max`` setting in the``/etc/sysctl.conf`` file. For example:

```
    kernel.pid_max = 4194303
```

- **Kernel Version:** Identify the kernel version and distribution you are using. Ceph uses some third party tools by default, which may be buggy or may conflict with certain distributions and/or kernel versions (e.g., Google perftools)..
- **Segment Fault:** If there is a segment fault, turn your logging up (if it is not already), and try again. If it segment faults again, contact the ceph-devel email list and provide your Ceph configuration file, your monitor output and the contents of your log file(s).

### *An OSD Failed*

When a ceph-osd process dies, the monitor will learn about the failure from surviving ceph-osd daemons and report it via the ceph health command

```
kubectl exec  -n ceph  ${MON_POD} ceph health
HEALTH_WARN 1/3 in osds are down
```

Specifically, you will get a warning whenever there are ceph-osd processes that are marked in and down. You can identify which ceph-osds are down with

```
kubectl exec -n ceph ${MON_POD} ceph health detail
HEALTH_WARN 1/3 in osds are down
osd.0 is down since epoch 23, last address 192.168.106.220:6800/11080
```

If there is a disk failure or other fault preventing ceph-osd from functioning or restarting, an error message should be present in its log file in /var/log/ceph.

If the daemon stopped because of a heartbeat failure, the underlying kernel file system may be unresponsive. Check dmesg output for disk or other kernel errors.

### *Low OSD Free Space*

Ceph prevents you from writing to a full OSD so that you don't lose data. In an operational cluster, you should receive a warning when your cluster is getting near its full ratio. The mon osd full ratio defaults to 0.95, or 95% of capacity before it stops clients from writing data. The mon osd backfill ratio defaults to 0.90, or 90 % of capacity when it blocks backfills from starting. The OSD nearfull ratio defaults to 0.85, or 85% of capacity when it generates a health warning.

Changing it can be done using:

```
kubectl exec -n ceph ${MON_POD} -- ceph osd set-nearfull-ratio <float[0.0-
1.0]>
```

Full cluster issues usually arise when testing how Ceph handles an OSD failure on a small cluster. When one node has a high percentage of the cluster's data, the cluster can easily eclipse its nearfull and full ratio immediately. If you are testing how Ceph reacts to OSD failures on a small cluster, you should leave ample free disk space and consider temporarily lowering the OSD full ratio, OSD backfill ratio and OSD nearfull ratio using these commands:

```
kubectl exec  -n ceph ${MON_POD} -- ceph osd set-nearfull-ratio <float[0.0-
1.0]>
kubectl exec  -n ceph ${MON_POD} -- ceph osd set-full-ratio <float[0.0-1.0]>
kubectl exec  -n ceph ${MON_POD} -- ceph osd set-backfillfull-ratio
<float[0.0-1.0]>
```

Full and near-full OSDs will be reported by ceph health:

```
kubectl exec -n ceph ${MON_POD} -- ceph health
HEALTH_WARN 1 nearfull osd(s)
```

or:

```
kubectl exec  -n ceph ${MON_POD} -- ceph health detail
HEALTH_ERR 1 full osd(s); 1 backfillfull osd(s); 1 nearfull osd(s)
osd.3 is full at 97%
osd.4 is backfill full at 91%
osd.2 is near full at 87%
```

The best way to deal with a full cluster is to add new ceph-osds, allowing the cluster to redistribute data to the newly available storage.

If you cannot start an OSD because it is full, you may delete some data by deleting some placement group directories in the full OSD.

**Important:** If you choose to delete a placement group directory on a full OSD, **DO NOT** delete the same placement group directory on another full OSD, or **YOU MAY LOSE DATA**.
You **MUST** maintain at least one copy of your data on at least one OSD.

### *Drive Configuration*

A storage drive should only support one OSD. Sequential read and sequential write throughput can bottleneck if other processes share the drive, including journals, operating systems, monitors, other OSDs and non-Ceph processes.

Ceph acknowledges writes *after* its journal has been written, so fast SSDs are an attractive option to accelerate the response time of wrote workloads.

**Note:** Partitioning a drive does not change its total throughput or sequential read/write limits. Running a journal in a separate partition may help, but you should prefer a separate physical drive.

### *Bad Sectors / Fragmented Disk*

Check your disks for bad sectors and fragmentation. This can cause total throughput to drop substantially.

### Logging Levels

If you turned logging levels up to track an issue and then forgot to turn logging levels back down, the OSD may be writing a lot of logs onto the disk. If you intend to keep logging levels high, you may consider mounting a drive to the default path for logging (i.e., /var/log/ceph/$cluster-$name.log).

### OSD Recovery

When a ceph-osd process dies, the monitor learns about the failure from surviving OSD daemons and reports it via the ceph health command:

```
$ ceph health
HEALTH_WARN 1/3 in osds are down
```

Specifically, you get a warning whenever there are ceph-osd processes that are marked in and down. You can identify which ceph-osds are down with this command:

```
$ ceph health detail
HEALTH_WARN 1/3 in osds are down
osd.0 is down since epoch 23, last address 192.168.106.220:6800/11080
```

Under normal circumstances, simply restarting the ceph-osd daemon allows it to rejoin the cluster and recover. If there is a disk failure or other fault preventing ceph-osd from functioning or restarting, an error message should be present in its log file in /var/log/ceph.

If the daemon has been marked as ‘down’ because of a heartbeat failure, the underlying kernel file system may be unresponsive. Check dmesg output for disk or other kernel errors and verify the OSD daemon did not crash or stop.

As soon an OSD is marked out, Ceph initiates recovery operations. The grace period before marking OSDs out automatically is set by the optional ceph.conf tunable mon_osd_down_out_interval, which defaults to 300 seconds (5 minutes). During recovery Ceph moves or copies all data that was hosted on the OSD devices that failed.

Since CRUSH replicates data to multiple OSDs, replicated copies survive and are read during the recovery process. As CRUSH develops the requisite new mapping of PGs to OSDs to populate the CRUSH map, it minimizes the amount of data that must move to other, still-functioning hosts and OSD devices to complete recovery. This helps degraded objects (those that lost their copies due to failures), and thus the cluster as a whole becomes healthy once again as the replication demanded by the configured CRUSH rule policy is restored.

### Recovery Throttling

Depending upon your configuration, Ceph may reduce recovery rates to maintain performance or it may increase recovery rates to the point that recovery impacts OSD performance. Check to see if the OSD is recovering.

### Filesystem Issues

Currently, it is recommend deploying clusters with XFS.

### Insufficient RAM

4GB of RAM is recommended per OSD daemon. You may notice that during normal operations, the OSD only uses a fraction of that amount (e.g., 100-200MB). Unused RAM makes it tempting to use the excess RAM for co-resident applications, VMs and so forth. However, when OSDs go into recovery mode, their memory utilization spikes. If there is no RAM available, the OSD performance will slow considerably.

### Slow Requests

If a ceph-osd daemon is slow to respond to a request, it will generate log messages complaining about requests that are taking too long. The warning threshold defaults to 30 seconds and is configurable via the osd op complaint time option. When this happens, the cluster log will receive messages such as the one below.

```
{date} {osd.num} [WRN] 1 slow requests, 1 included below; oldest blocked for
> 30.005692 secs
{date} {osd.num}  [WRN] slow request 30.005692 seconds old, received at
{date-time}: osd_op(client.4240.0:8 benchmark_data_ceph-1_39426_object7
[write 0~4194304] 0.69848840) v4 currently waiting for subops from [610]
```

Possible causes include:

- A bad drive (check dmesg output)
- A bug in the kernel file system (check dmesg output)
- An overloaded cluster (check system load, iostat, etc.)
- A bug in the ceph-osd daemon.

Possible solutions:

- Upgrade kernel
- Upgrade Ceph
- Restart OSDs

### Debugging Slow Requests

If you run either of the following commands, you will see a set of operations and a list of events each operation went through.

```
kubectl exec -n ceph {OSD-POD} -- ceph daemon osd.<id> dump_historic_ops
kubectl exec -n ceph {OSD-POD} -- ceph daemon osd.<id> dump_ops_in_flight
```

These events are briefly described below.

#### *Events from the Messenger layer*

- header_read: When the messenger first started reading the message off the wire.

- throttled: When the messenger tried to acquire memory throttle space to read the message into memory.
- all_read: When the messenger finished reading the message off the wire.
- dispatched: When the messenger gave the message to the OSD.
- initiated: This is identical to header_read. The existence of both is a historical oddity.

*Events from the OSD as it prepares operations*

- queued_for_pg: The op has been put into the queue for processing by its PG.
- reached_pg: The PG has started doing the op.
- waiting for *: The op is waiting for some other work to complete before it can proceed (e.g. a new OSDMap; for its object target to scrub; for the PG to finish peering; all as specified in the message).
- started: The op has been accepted as something the OSD should do and is now being performed.
- waiting for subops from: The op has been sent to replica OSDs.

*Events from the FileStore*

- commit_queued_for_journal_write: The op has been given to the FileStore.
- write_thread_in_journal_buffer: The op is in the journal's buffer and waiting to be persisted (as the next disk write).
- journaled_completion_queued: The op was journaled to disk and its callback queued for invocation.

*Events from the OSD after stuff has been given to local disk*

- op_commit: The op has been committed (i.e. written to journal) by the primary OSD.
- op_applied: The op has been written to the backing FS (i.e. applied in memory but not flushed out to disk) on the primary.
- sub_op_applied: op_applied, but for a replica's "subop".
- sub_op_committed: op_commit, but for a replica's subop (only for EC pools).
- sub_op_commit_rec/sub_op_apply_rec from <X>: The primary marks this when it hears about the above, but for a particular replica (i.e. <X>).
- commit_sent: a reply back to the client (or primary OSD, for sub ops).

Many of these events are seemingly redundant, but they cross important boundaries in the internal code (such as passing data across locks into new threads).

```
up      # prevent OSDs from getting marked upceph osd set nodown    # prevent
OSDs from getting marked down
```

These flags are recorded in the osdmap structure:

```
ceph osd dump | grep flagsflags no-up,no-down
```

You can clear the flags with:

```
ceph osd unset noupceph osd unset nodown
```

Two other flags are supported, noin and noout, which prevent booting OSDs from being marked in(allocated data) or protect OSDs from eventually being marked out (regardless of what the current value for mon_osd_down_out_interval is).

**Note:** The noup, noout, and nodown flags are temporary in the sense that once the flags are cleared, the action they were blocking should occur shortly after. The noin flag, on the other hand,

prevents OSDs from being marked in on boot, and any daemons that started while the flag was set will remain that way.

### Failed Assert

Luminous OSD(s) can crash due to a failed assert in pi_compact_rep::add_interval(). This is an unresolved bug in Luminous, which is the main reason Airship has moved to Mimic, which sometimes causes OSDs to crash with log entries and a call stack similar to the following:

```
   -34> 2017-08-26 00:38:00.505114 7f14556b4d00  0 osd.299 1085665 load_pgs
opened 455 pgs
   -33> 2017-08-26 00:38:00.505787 7f14556b4d00 10 osd.299 1085665 19.f1e
needs 1050342-1085230
   -32> 2017-08-26 00:38:00.505814 7f14556b4d00  1 osd.299 1085665
build_past_intervals_parallel over 1050342-1085230
   -31> 2017-08-26 00:38:00.505818 7f14556b4d00 10 osd.299 1085665
build_past_intervals_parallel epoch 1050342
   -30> 2017-08-26 00:38:00.505824 7f14556b4d00 20 osd.299 0 get_map 1050342
- loading and decoding 0x7f14b3dfb0c0
   -29> 2017-08-26 00:38:00.506245 7f14556b4d00 10 osd.299 0 add_map_bl
1050342 780781 bytes
   -28> 2017-08-26 00:38:00.508539 7f14556b4d00 10 osd.299 1085665
build_past_intervals_parallel epoch 1050342 pg 19.f1e first map, acting [80]
up [80], same_interval_since = 1050342
   -27> 2017-08-26 00:38:00.508547 7f14556b4d00 10 osd.299 1085665
build_past_intervals_parallel epoch 1050343
   -26> 2017-08-26 00:38:00.508550 7f14556b4d00 20 osd.299 0 get_map 1050343
- loading and decoding 0x7f14b3dfad80
   -25> 2017-08-26 00:38:00.508997 7f14556b4d00 10 osd.299 0 add_map_bl
1050343 781371 bytes
   -24> 2017-08-26 00:38:00.511176 7f14556b4d00 10 osd.299 1085665
build_past_intervals_parallel epoch 1050344
   -23> 2017-08-26 00:38:00.511196 7f14556b4d00 20 osd.299 0 get_map 1050344
- loading and decoding 0x7f14b3dfb740
   -22> 2017-08-26 00:38:00.511625 7f14556b4d00 10 osd.299 0 add_map_bl
1050344 782446 bytes
   -21> 2017-08-26 00:38:00.513813 7f14556b4d00 10 osd.299 1085665
build_past_intervals_parallel epoch 1050345
   -20> 2017-08-26 00:38:00.513820 7f14556b4d00 20 osd.299 0 get_map 1050345
- loading and decoding 0x7f14b3dfba80
   -19> 2017-08-26 00:38:00.514260 7f14556b4d00 10 osd.299 0 add_map_bl
1050345 782071 bytes
   -18> 2017-08-26 00:38:00.516463 7f14556b4d00 10 osd.299 1085665
build_past_intervals_parallel epoch 1050346
   -17> 2017-08-26 00:38:00.516488 7f14556b4d00 20 osd.299 0 get_map 1050346
- loading and decoding 0x7f14b79c4000
   -16> 2017-08-26 00:38:00.516927 7f14556b4d00 10 osd.299 0 add_map_bl
1050346 781955 bytes
   -15> 2017-08-26 00:38:00.519047 7f14556b4d00 10 osd.299 1085665
build_past_intervals_parallel epoch 1050347
   -14> 2017-08-26 00:38:00.519054 7f14556b4d00 20 osd.299 0 get_map 1050347
- loading and decoding 0x7f14b79c4340
   -13> 2017-08-26 00:38:00.519500 7f14556b4d00 10 osd.299 0 add_map_bl
1050347 781930 bytes
   -12> 2017-08-26 00:38:00.521612 7f14556b4d00 10 osd.299 1085665
build_past_intervals_parallel epoch 1050348
   -11> 2017-08-26 00:38:00.521619 7f14556b4d00 20 osd.299 0 get_map 1050348
- loading and decoding 0x7f14b79c4680
```

```
   -10> 2017-08-26 00:38:00.522074 7f14556b4d00 10 osd.299 0 add_map_bl
1050348 784883 bytes
    -9> 2017-08-26 00:38:00.524245 7f14556b4d00 10 osd.299 1085665
build_past_intervals_parallel epoch 1050349
    -8> 2017-08-26 00:38:00.524252 7f14556b4d00 20 osd.299 0 get_map 1050349
- loading and decoding 0x7f14b79c49c0
    -7> 2017-08-26 00:38:00.524706 7f14556b4d00 10 osd.299 0 add_map_bl
1050349 785081 bytes
    -6> 2017-08-26 00:38:00.526854 7f14556b4d00 10 osd.299 1085665
build_past_intervals_parallel epoch 1050350
    -5> 2017-08-26 00:38:00.526861 7f14556b4d00 20 osd.299 0 get_map 1050350
- loading and decoding 0x7f14b79c4d00
    -4> 2017-08-26 00:38:00.527330 7f14556b4d00 10 osd.299 0 add_map_bl
1050350 785948 bytes
    -3> 2017-08-26 00:38:00.529505 7f14556b4d00 10 osd.299 1085665
build_past_intervals_parallel epoch 1050351
    -2> 2017-08-26 00:38:00.529512 7f14556b4d00 20 osd.299 0 get_map 1050351
- loading and decoding 0x7f14b79c5040
    -1> 2017-08-26 00:38:00.529979 7f14556b4d00 10 osd.299 0 add_map_bl
1050351 788650 bytes
     0> 2017-08-26 00:38:00.534373 7f14556b4d00 -1 /home/jenkins-
build/build/workspace/ceph-
build/ARCH/x86_64/AVAILABLE_ARCH/x86_64/AVAILABLE_DIST/centos7/DIST/centos7/M
ACHINE_SIZE/huge/release/12.1.4/rpm/el7/BUILD/ceph-
12.1.4/src/osd/osd_types.cc: In function 'virtual void
pi_compact_rep::add_interval(bool, const PastIntervals::pg_interval_t&)'
thread 7f14556b4d00 time 2017-08-26 00:38:00.532119
/home/jenkins-build/build/workspace/ceph-
build/ARCH/x86_64/AVAILABLE_ARCH/x86_64/AVAILABLE_DIST/centos7/DIST/centos7/M
ACHINE_SIZE/huge/release/12.1.4/rpm/el7/BUILD/ceph-
12.1.4/src/osd/osd_types.cc: 3205: FAILED assert(interval.last > last)
ceph version 12.1.4 (a5f84b37668fc8e03165aaf5cbb380c78e4deba4) luminous (rc)
1: (ceph::__ceph_assert_fail(char const*, char const*, int, char
const*)+0x110) [0x7f145612f420]
2: (pi_compact_rep::add_interval(bool, PastIntervals::pg_interval_t
const&)+0x3b2) [0x7f1455e030b2]
3: (PastIntervals::check_new_interval(int, int, std::vector<int,
std::allocator<int> > const&, std::vector<int, std::allocator<int> > const&,
int, int, std::vector<int, std::allocator<int> > const&, std::vector<int,
std::allocator<int> > const&, unsigned int, unsigned int,
std::shared_ptr<OSDMap const>, std::shared_ptr<OSDMap const>, pg_t,
IsPGRecoverablePredicate*, PastIntervals*, std::ostream*)+0x380)
[0x7f1455de8ab0]
4: (OSD::build_past_intervals_parallel()+0xa8f) [0x7f1455bbc71f]
5: (OSD::load_pgs()+0x503) [0x7f1455bbef13]
6: (OSD::init()+0x2179) [0x7f1455bd7779]
7: (main()+0x2def) [0x7f1455add56f]
8: (__libc_start_main()+0xf5) [0x7f1451d14b35]
9: (()+0x4ac8a6) [0x7f1455b7b8a6]
```

In this case the log indicates that the OSD was attempting to build past intervals for PG 19.f1e when it crashed. There isn't a fix for this issue in Luminous, but there is a workaround, which is to delete the past intervals for the affected PG offline and allow the OSD to attempt to rebuild them.

In order to delete the past intervals for the affected PG, a few things are needed. The ceph-osd pod for the affected OSD must be stopped so it won't continually try to restart and use the OSD's object

store while this operation is in process. Also, the object store for the affected OSD must be mounted in a running ceph-osd pod so ceph-objectstore-tool can be used to delete the past intervals for the affected PG. When these steps are complete, the crashed ceph-osd pod should be able to come up and function normally.

```
# kubectl label nodes --all ceph_recover_window=inactive
# kubectl label nodes {target host} —overwrite ceph_recover_window=active
# kubectl patch -n ceph ds {crashed ceph-osd daemon-set} -
p='{"spec":{"template":{"spec":{"nodeSelector":{"ceph-
osd":"enabled","ceph_recover_window":"inactive"}}}}}'
# kubectl -n ceph exec -it {running ceph-osd-pod} — /bin/bash
# mkdir /var/lib/ceph/osd/ceph-{OSD ID}
# mount {crashed OSD partition} /var/lib/ceph/osd/ceph-{OSD ID}
# ceph-objectstore-tool —data-path /var/lib/ceph/osd/ceph-{OSD ID} —pgid {PG
ID} —op rm-past-intervals
# umount /var/lib/ceph/osd/ceph-{OSD ID}
# rmdir /var/lib/ceph/osd/ceph-{OSD ID}
# exit
# kubectl patch -n ceph ds {crashed ceph-osd daemon-set} -
p='{"spec":{"template":{"spec":{"nodeSelector":{"ceph-
osd":"enabled","ceph_recover_window":"active"}}}}}'
```

These steps really just facilitate a retry for building the PG's past intervals. It is entirely possible that the OSD could crash again in that process. This bug appears to be due to a race condition that has been removed in Mimic. If the race condition is hit again and the OSD crashes, repeat the steps again and attempt another retry.

## Troubleshooting Flapping OSDs

There may be occasions when a single osd or multiple osds bounce back and forth between up and down status. If such events are occuring in the cluster the following procedures should be taken to begin troubleshooting.

```
kubectl exec -n ceph ${MON_POD} -- ceph -w | grep osds
```

The  command shows OSDs repeatedly as down and then up again within a short period of time:

```
#  kubectl exec -n ceph ${MON_POD} -- ceph -w | grep osds2017-04-05
06:27:20.810535 mon.0 [INF] osdmap e609: 9 osds: 8up, 9 in2017-04-05
06:27:24.120611 mon.0 [INF] osdmap e611: 9 osds: 7up, 9 in2017-04-05
06:27:25.975622 mon.0 [INF] HEALTH_WARN; 118 pgsstale; 2/9 in osds are
down2017-04-05 06:27:27.489790 mon.0 [INF] osdmap e614: 9 osds: 6up, 9
in2017-04-05 06:27:36.540000 mon.0 [INF] osdmap e616: 9 osds: 7up, 9 in2017-
04-05 06:27:39.681913 mon.0 [INF] osdmap e618: 9 osds: 8up, 9 in2017-04-05
06:27:43.269401 mon.0 [INF] osdmap e620: 9 osds: 9up, 9 in2017-04-05
06:27:54.884426 mon.0 [INF] osdmap e622: 9 osds: 8up, 9 in2017-04-05
06:27:57.398706 mon.0 [INF] osdmap e624: 9 osds: 7up, 9 in2017-04-05
06:27:59.669841 mon.0 [INF] osdmap e625: 9 osds: 6up, 9 in2017-04-05
06:28:07.043677 mon.0 [INF] osdmap e628: 9 osds: 7up, 9 in2017-04-05
06:28:10.512331 mon.0 [INF] osdmap e630: 9 osds: 8up, 9 in2017-04-05
06:28:12.670923 mon.0 [INF] osdmap e631: 9 osds: 9up, 9 in
```

In addition the Ceph log contains error messages similar to the following ones:

```
2016-07-25 03:44:06.510583 osd.50 127.0.0.1:6801/149046 18992 :cluster [WRN]
map e600547 wrongly marked me down
2016-07-25 19:00:08.906864 7fa2a0033700 -1 osd.254 609110heartbeat_check: no
reply from osd.2 since back
```

```
2016-07-2519:00:07.444113 front
2016-07-25 18:59:48.311935 (cutoff 2016-07-25 18:59:48.906862)
```

*What This Means*

The main causes for OSDs to flap are:

- Certain cluster operations, such as scrubbing or recovery, take an abnormal amount of time, for example if you perform these operations on objects with a large index or if your cluster contains large placement groups. Usually, after these operations finish, the flapping OSD problem will solve by itself.
- Problems with the underlying physical hardware. In this case, the ceph health detail command usually returns slow requests error message.
- Problems with the network

OSDs cannot handle well the situation when the cluster (back-end) network fails or develops significant latency while the public (front-end) network operates optimally.

OSDs use the cluster network for sending heartbeat packets to each other to indicate that they are up and in. If the cluster network does not work properly, OSDs are unable to send and receive the heartbeat packets. As a consequence, they report each other as being down to the Monitors, while marking themselves as up when they receive an OSD map update.

**Note:** The flapping OSDs scenario does not include the situation when the OSD processes are started and then immediately killed.

*To Troubleshoot This Problem*

Check the output of the ceph health detail command again.

```
# ceph health detail
HEALTH_WARN 30 requests are blocked > 32 sec; 3 osds have slow requests
30 ops are blocked > 268435 sec
1 ops are blocked > 268435 sec on osd.11
1 ops are blocked > 268435 sec on osd.18
28 ops are blocked > 268435 sec on osd.39
3 osds have slow requests
```

Determine which OSDs are marked as down and on what nodes they reside:

```
#  kubectl exec -n ceph ${MON_POD} -- ceph osd tree | grep down
```

On the nodes containing the flapping OSDs, troubleshoot and fix any networking problems.

Alternatively, you can temporarily force Monitors to stop marking the OSDs as down and up by setting the noup and nodown flags:

```
#  kubectl exec -n ceph ${MON_POD} -- ceph osd set noup
#  kubectl exec -n ceph ${MON_POD} -- ceph osd set nodown
```

**Important:** Using the noup and nodown flags does not fix the root cause of the problem but only prevents OSDs from flapping. Open a support ticket, if you are unable to fix and troubleshoot the error by yourself.

## Troubleshooting OSDs Slow/Blocked Requests

The ceph-osd daemon is slow to respond to a request and the ceph health detail command returns an error message similar to the following one

```
HEALTH_WARN 30 requests are blocked > 32 sec; 3 osds have slow requests
30 ops are blocked > 268435 sec
1 ops are blocked > 268435 sec on osd.11
1 ops are blocked > 268435 sec on osd.18
28 ops are blocked > 268435 sec on osd.39
3 osds have slow requests
```

In addition, the Ceph logs include an error message similar to the following ones:

```
2015-08-24 13:18:10.024659 osd.1 127.0.0.1:6812/3032 9 : cluster[WRN] 6 slow
requests, 6 included below; oldest blocked for >61.758455 secs
2016-07-25 03:44:06.510583 osd.50 [WRN] slow request 30.005692seconds old,
received at {date-time}: osd_op(client.4240.0:8benchmark_data_ceph-
1_39426_object7 [write 0~4194304]0.69848840) v4 currently waiting for subops
from [610]
```

### What This Means

The presence of slow requests indicate an OSD is not able to service the I/O operations it has received within the time defined by the osd_op_complaint_time parameter. By default, this parameter is set to 30 seconds.

The main causes of OSDs having slow requests are:

- Problems with the underlying hardware, such as disk drives, hosts, racks, or network switches Problems with network. These problems are usually connected with flapping OSDs. See the section called “Flapping OSDs” for details. System load The following table shows the types of slow requests. Use the dump_historic_ops administration socket command to determine the type of a slow request.

### To Troubleshoot This Problem

Determine if the OSDs with slow or blocked requests share a common piece of hardware, for example a journal drive, a host, a rack, or a network switch.

If the OSDs share a journal drive:

- Use the smartmontools utility to check the health of the disk or the logs to determine any errors on the disk.
- Use the iostat utility to get the I/O wait report (%iowait and %utilization) on the OSD disk to determine if the disk is under heavy load or experiencing an unusual wait time servicing IO requests

**Note:** If the OSDs share a host:

1. Check the RAM and CPU utilization
2. Use the netstat utility to see the network statistics on the Network Interface Controllers (NICs) and troubleshoot any networking issues.

If the OSDs share a rack, check the network switch for the rack. For example, if you use jumbo frames, verify that the NIC in the path has jumbo frames set.

### Troubleshooting RGW

The best way to start troubleshooting RADOS Gateway problems is to inspect the logs in /var/log/ceph/ceph-client.rgw.{hostname}.log.

### *HTTP Request Errors*

Examining the access and error logs for the web server itself is probably the first step in identifying what is going on. If there is a 500 error, that usually indicates a problem communicating with the radosgw daemon. Ensure the daemon is running, its socket path is configured, and that the web server is looking for it in the proper location.

### *Crashed RadosGW Process*

If the built-in HTTP server thread dies, you will normally get  a 500 error. In that situation, simply restarting radosgw will restore service.To diagnose the cause of the crash, check the log in /var/log/ceph and/or the core file (if one was generated).Blcoked RadosGW Requests
If some (or all) radosgw requests appear to be blocked, you can get some insight into the internal state of the radosgw daemon via its admin socket. By default, there will be a socket configured to reside in /var/run/ceph, and the daemon can be queried with:

```
ceph daemon /var/run/ceph/client.rgw help
help                list available commands
objecter_requests   show in-progress osd requests
perfcounters_dump   dump perfcounters value
perfcounters_schema dump perfcounters schema
version             get protocol version
```

Of particular interest:

```
ceph daemon /var/run/ceph/client.rgw objecter_requests
...
```

will dump information about current in-progress requests with the RADOS cluster. This allows one to identify if any requests are blocked by a non-responsive OSD. For example, one might see:

```
{ "ops": [
      { "tid": 1858,
        "pg": "2.d2041a48",
        "osd": 1,
        "last_sent": "2012-03-08 14:56:37.949872",
        "attempts": 1,
        "object_id": "fatty_25647_object1857",
        "object_locator": "@2",
        "snapid": "head",
        "snap_context": "0=[]",
        "mtime": "2012-03-08 14:56:37.949813",
        "osd_ops": [
              "write 0~4096"]},
      { "tid": 1873,
        "pg": "2.695e9f8e",
        "osd": 1,
        "last_sent": "2012-03-08 14:56:37.970615",
        "attempts": 1,
        "object_id": "fatty_25647_object1872",
        "object_locator": "@2",
        "snapid": "head",
        "snap_context": "0=[]",
        "mtime": "2012-03-08 14:56:37.970555",
        "osd_ops": [
              "write 0~4096"]}],
"linger_ops": [],
"pool_ops": [],
"pool_stat_ops": [],
```

```
"statfs_ops": []}
```

In this dump, two requests are in progress. The last_sent field is the time the RADOS request was sent. If this is a while ago, it suggests that the OSD is not responding. For example, for request 1858, you could check the OSD status with:

```
ceph pg map 2.d2041a48osdmap e9 pg 2.d2041a48 (2.0) -> up [1,0] acting [1,0]
```

This tells us to look at osd.1, the primary copy for this PG:

```
ceph --admin-daemon /var/run/ceph/osd.1.asok
{ "num_ops": 651,
"ops": [
       { "description": "osd_op(client.4124.0:1858 fatty_25647_object1857
[write 0~4096] 2.d2041a48)",
         "received_at": "1331247573.344650",
         "age": "25.606449",
         "flag_point": "waiting for sub ops",
         "client_info": { "client": "client.4124",
             "tid": 1858}},
...
```

The flag_point field indicates that the OSD is currently waiting for replicas to respond, in this case,as we are looking at OSD 1 and the only other OSD protecting this PG is OSD 0 according to the ceph pg map command.

## Troubleshooting Placement Groups

It is normal for placement groups to enter states like "degraded" or "peering" following a failure. Normally these states indicate the normal progression through the failure recovery process. However, if a placement group stays in one of these states for a long time this may be an indication of a larger problem. For this reason, the monitor will warn when placement groups get "stuck" in a non-optimal state. Specifically:

- inactive - The placement group has not been active for too long (i.e., it hasn't been able to service read/write requests).
- unclean - The placement group has not been clean for too long (i.e., it hasn't been able to completely recover from a previous failure).
- stale - The placement group status has not been updated by a ceph-osd, indicating that all nodes storing this placement group may be down.

You can explicitly list stuck placement groups with one of:

```
kubectl exec -n ceph ${MON_POD} -- ceph pg dump_stuck stale
kubectl exec  -n ceph ${MON_POD} -- ceph pg dump_stuck inactive
kubectl exec  -n ceph ${MON_POD} -- ceph pg dump_stuck unclean
```

For stuck stale placement groups, it is normally a matter of getting the right ceph-osd daemons running again. For stuck inactive placement groups, it is usually a peering problem (see failures-osd-peering). For stuck unclean placement groups, there is usually something preventing recovery from completing, like unfound objects (see failures-osd-unfound).

## Placement Group Down - Peering Failure

In certain cases, the ceph-osd Peering process can run into problems, preventing a PG from becoming active and usable. For example, ceph health might report:

```
kubectl exec  -n ceph ${MON_POD} -- ceph health detail|
HEALTH_ERR 7 pgs degraded; 12 pgs down; 12 pgs peering; 1 pgs recovering; 6
```

```
pgs stuck unclean; 114/3300 degraded (3.455%); 1/3 in osds are down
...
pg 0.5 is down+peering
pg 1.4 is down+peering
...
osd.1 is down since epoch 69, last address 192.168.106.220:6801/8651
```

Query the cluster to determine exactly why the PG is marked down with:

```
kubectl exec  -n ceph ${MON_POD} -- ceph pg 0.5 query
```

Results:

```
{ "state": "down+peering",
  ...
  "recovery_state": [
        { "name": "Started\/Primary\/Peering\/GetInfo",
          "enter_time": "2012-03-06 14:40:16.169679",
          "requested_info_from": []},
        { "name": "Started\/Primary\/Peering",
          "enter_time": "2012-03-06 14:40:16.169659",
          "probing_osds": [
                0,
                1],
          "blocked": "peering is blocked due to down osds",
          "down_osds_we_would_probe": [
                1],
          "peering_blocked_by": [
                { "osd": 1,
                  "current_lost_at": 0,
                  "comment": "starting or marking this osd lost may let us
proceed"}]},
        { "name": "Started",
          "enter_time": "2012-03-06 14:40:16.169513"}
    ]
}
```

The recovery_state section tells us that peering is blocked due to down ceph-osd daemons, specifically osd.1. In this case, start the ceph-osd and things will recover.

Alternatively, if there is a catastrophic failure of osd.1 (e.g., disk failure), we can tell the cluster that it is lost and to cope the best it can.

**Important:** This is dangerous in that the cluster cannot guarantee that the other copies of the data are consistent and up to date.

To instruct Ceph to continue anyway:

```
kubectl exec  -n ceph ${MON_POD} -- ceph osd lost 1
```

Recovery will proceed.

### Unfound Object

Under certain combinations of failures Ceph may complain about unfound objects:

```
kubectl exec  -n ceph ${MON_POD} -- ceph health detail
HEALTH_WARN 1 pgs degraded; 78/3778 unfound (2.065%)
pg 2.4 is active+degraded, 78 unfound
```

This means that the storage cluster knows that some objects (or newer copies of existing objects) exist, but it hasn't found copies of them. One example of how this might come about for a PG whose data is on ceph-osds 1 and 2:

- 1 goes down
- 2 handles some writes, alone
- 1 comes up
- 1 and 2 re-peer, and the objects missing on 1 are queued for recovery.
- Before the new objects are copied, 2 goes down.

Now 1 knows that these object exist, but there is no live ceph-osd with a valid copy of this object. In this case, IO to those objects will block, and the cluster will hope that the failed node comes back soon; this is assumed to be preferable to returning an IO error to the user.

First, you can identify which objects are unfound with:

```
kubectl exec  -n ceph ${MON_POD} -- ceph pg 2.4 list_missing [starting
offset, in json]
```

Results:

```
{ "offset": { "oid": "",
     "key": "",
     "snapid": 0,
     "hash": 0,
     "max": 0},
 "num_missing": 0,
 "num_unfound": 0,
 "objects": [
    { "oid": "object 1",
      "key": "",
      "hash": 0,
      "max": 0 },
    ...
 ],
 "more": 0}
```

If there are too many objects to list in a single result, the more field will be true and you can query for more.

You can then identify which OSDs have been probed or might contain data:

```
kubectl exec  -n ceph ${MON_POD} -- ceph pg 2.4 query
```

Results:

```
"recovery_state": [
     { "name": "Started\/Primary\/Active",
       "enter_time": "2012-03-06 15:15:46.713212",
       "might_have_unfound": [
             { "osd": 1,
               "status": "osd is down"}]},
```

In this case, for example, the cluster knows that osd.1 might have data, but it is down. The full range of possible states include:

- already probed
- querying

- OSD is down
- not queried (yet)

Sometimes it simply takes some time for the cluster to query possible locations.

It is possible that there are other locations where the object may exist that are not listed. For example, if a ceph-osd is stopped and taken out of the cluster, the cluster fully recovers, and due to some future set of failures ends up with an unfound object, it won't consider the long-departed ceph-osd as a potential location. (This scenario, however, is unlikely.)

If all possible locations have been queried and objects are still lost, you may have to give up on the lost objects. This, again, is possible given unusual combinations of failures that allow the cluster to learn about writes that were performed before the writes themselves are recovered. To mark the "unfound" objects as "lost":

```
kubectl exec -n ceph ${MON_POD} -- ceph pg 2.5 mark_unfound_lost
revert|delete
```

The final argument specifies how the cluster should deal with lost objects.

The "delete" option will forget about them entirely.

The "revert" option  will either roll back to a previous version of the object or (if it was a new object) forget about it entirely. Use this with caution, as it may confuse applications that expected the object to exist.

### Homeless Placement Groups

It is possible for all OSDs that had copies of a given placement groups to fail. If that's the case, that subset of the object store is unavailable, and the monitor will receive no status updates for those placement groups. To detect this situation, the monitor marks any placement group whose primary OSD has failed as stale. For example:

```
kubectl exec -n ceph ${MON_POD} -- ceph health
HEALTH_WARN 24 pgs stale; 3/300 in osds are down
```

You can identify which placement groups are stale, and what the last OSDs to store them were, with:

```
kubectl exec -n ceph ${MON_POD} -- ceph health detail
HEALTH_WARN 24 pgs stale; 3/300 in osds are down...pg 2.5 is stuck
stale+active+remapped, last acting [2,0]
...
osd.10 is down since epoch 23, last address 192.168.106.220:6800/11080
osd.11 is down since epoch 13, last address 192.168.106.220:6803/11539
osd.12 is down since epoch 24, last address 192.168.106.220:6806/11861
```

In order to get placement group 2.5 back online, for example, given it was last managed by osd.0 and osd.2, restarting those ceph-osd daemons will allow the cluster to recover that placement group (and, presumably, many others).

### Only a Few OSDs Receive Data

If you have many nodes in your cluster and only a few of them receive data,`check`_ the number of placement groups in your pool. Since placement groups get mapped to OSDs, a small number of placement groups will potentially not distribute placements groups across all the OSDs of your cluster. Try creating a pool with a placement group count that is a multiple of the number of OSDs and a power of 2 for better distribution.

### Can't Write Data

If your cluster is up, but some OSDs are down and you cannot write data, check to ensure that you have the minimum number of OSDs running for the placement group. If you don't have the minimum number of OSDs running, Ceph will not allow you to write data because there is no guarantee that Ceph can replicate your data.

### PGs Inconsistent

If you observe an active + clean + inconsistent PG, this is likely due to an error during scrubbing. As always, we can identify the inconsistent placement group(s) with:

```
$ kubectl exec  -n ceph ${MON_POD} -- ceph health detail
HEALTH_ERR 1 pgs inconsistent; 2 scrub errors
pg 0.6 is active+clean+inconsistent, acting [0,1,2]
2 scrub errors
```

Or if you prefer inspecting the output in a programmatic way:

```
$ kubectl exec  -n ceph ${MON_POD} -- rados list-inconsistent-pg <pool-name>
["0.6"]
```

There is only one consistent state, but in the worst case, we could have different inconsistencies in multiple perspectives found in more than one objects. If an object named foo in PG 0.6 is truncated, we will have:

```
$ kubectl exec  -n ceph ${MON_POD} -- rados list-inconsistent-obj <pgid> --
format=json-pretty
```

Results:

```
{
    "epoch": 14,
    "inconsistents": [
        {
            "object": {
                "name": "foo",
                "nspace": "",
                "locator": "",
                "snap": "head",
                "version": 1
            },
            "errors": [
                "data_digest_mismatch",
                "size_mismatch"
            ],
            "union_shard_errors": [
                "data_digest_mismatch_info",
                "size_mismatch_info"
            ],
            "selected_object_info": "0:602f83fe:::foo:head(16'1
client.4110.0:1 dirty|data_digest|omap_digest s 968 uv 1 dd e978e67f od
ffffffff alloc_hint [0 0 0])",
            "shards": [
                {
                    "osd": 0,
                    "errors": [],
                    "size": 968,
                    "omap_digest": "0xffffffff",
                    "data_digest": "0xe978e67f"
```

```
                },
                {
                    "osd": 1,
                    "errors": [],
                    "size": 968,
                    "omap_digest": "0xffffffff",
                    "data_digest": "0xe978e67f"
                },
                {
                    "osd": 2,
                    "errors": [
                        "data_digest_mismatch_info",
                        "size_mismatch_info"
                    ],
                    "size": 0,
                    "omap_digest": "0xffffffff",
                    "data_digest": "0xffffffff"
                }
            ]
        }
    ]
}
```

In this case, we can learn from the output:

- The only inconsistent object is named foo, and it is its head that has inconsistencies.
- The inconsistencies fall into two categories:
- errors: these errors indicate inconsistencies between shards without a determination of which shard(s) are bad. Check for the errors in the shards array, if available, to pinpoint the problem.
- data_digest_mismatch: the digest of the replica read from OSD.2 is different from the ones of OSD.0 and OSD.1
- size_mismatch: the size of the replica read from OSD.2 is 0, while the size reported by OSD.0 and OSD.1 is 968.
- union_shard_errors: the union of all shard specific errors in shards array. The errors are set for the given shard that has the problem. They include errors like read_error. The errors ending in oi indicate a comparison with selected_object_info. Look at the shards array to determine which shard has which error(s).
- data_digest_mismatch_info: the digest stored in the object-info is not 0xffffffff, which is calculated from the shard read from OSD.2
- size_mismatch_info: the size stored in the object-info is different from the one read from OSD.2. The latter is 0.

You can repair the inconsistent placement group by executing:

```
kubectl exec -n ceph ${MON_POD} -- ceph pg repair {placement-group-ID}
```

Which overwrites the bad copies with the authoritative ones. In most cases, Ceph is able to choose authoritative copies from all available replicas using some predefined criteria. But this does not always work. For example, the stored data digest could be missing, and the calculated digest will be ignored when choosing the authoritative copies. So, please use the above command with caution.

If read_error is listed in the errors attribute of a shard, the inconsistency is likely due to disk errors. You might want to check your disk used by that OSD.

If you receive active + clean + inconsistent states periodically due to clock skew, you may consider configuring your `NTP` daemons on your monitor hosts to act as peers.

## PGs Incomplete

A PG that is stuck incomplete is likely caused by OSDs going up and down at different times, usually in conjunction with one or more OSDs subsequently becoming permanently lost before recovery is complete. The incomplete PG(s) can be identified as follows:

```
# ceph health detail | grep incomplete
pg 6.399 is incomplete, acting [18,19,9]
pg 3.16f is incomplete, acting [20,8,21]
pg 3.c2 is incomplete, acting [30,25,4]
pg 6.84 is incomplete, acting [28,21,8]
pg 3.6a is incomplete, acting [30,27,8]
```

An incomplete PG is one for which the PG is aware of writes that have happened, but the data associated with those writes is not available on any active OSDs. Essentially every available copy of the PG knows that it is behind or incomplete.

There are two options for resolving incomplete PGs. Either the data required to complete the PG must become available somehow, or an available, incomplete copy of the PG must be marked as complete manually. Marking an incomplete copy as complete involves some level of data loss and should only be done in cases where there is no other option or the associated data is determined to be disposable.

In order to determine which unavailable OSD(s) contain the missing data, the PG must be queried via ‘ceph pg query {PG ID}’. The output of this command is lengthy and should probably be viewed with ‘less’ or some other mechanism that allows the output to be scrolled and searched. In this case look for the section under ‘down_osds_we_would_probe’ to see which unavailable OSDs are causing the PG to be stuck incomplete similar to the following.

This is an example of a PG that is stuck peering, not incomplete, but the ‘down_osds_we_would_probe’ list is still relevant.

```
# ceph pg 0.5 query
{ "state": "down+peering",
  ...
  "recovery_state": [
        { "name": "Started\/Primary\/Peering\/GetInfo",
          "enter_time": "2012-03-06 14:40:16.169679",
          "requested_info_from": []},
        { "name": "Started\/Primary\/Peering",
          "enter_time": "2012-03-06 14:40:16.169659",
          "probing_osds": [
                0,
                1],
          "blocked": "peering is blocked due to down osds",
          "down_osds_we_would_probe": [
                1],
          "peering_blocked_by": [
                { "osd": 1,
                  "current_lost_at": 0,
                  "comment": "starting or marking this osd lost may let us
proceed"}]},
        { "name": "Started",
          "enter_time": "2012-03-06 14:40:16.169513"}
    ]
}
```

The ideal scenario would be to bring the down OSDs back up and in and allow them to complete the PG. If this is possible, the situation should resolve itself and no further action is needed.

If the down OSDs are lost and not able to come back up and in but the data still exists on their disks, then the PG can be manually exported from a dead OSD's object store and imported to a healthy OSD using ceph-objectstore-tool to make the missing data become available. For this operation the PG data should ideally be imported to an OSD that doesn't currently have a copy of the PG. This ensures that the import won't be destructive in any way to existing data for the PG. Ceph will figure out where the data is supposed to be and move it appropriately.

Choose a random OSD that isn't listed as a peer under the 'peer_info' in the PG query for each OSD listed in the 'down_osds_we_would_probe' list. These could be different OSDs for each PG or, disk space permitting, a single OSD on which to import all exported PGs from different failed OSDs. For each OSD pair (one down and one active), do the following:

Install ceph-objectstore-tool on the host(s) with the failed OSD(s) or mount the failed OSD disk(s) in a running ceph-osd pod on the same host.

```
# ceph-objectstore-tool —data-path {path to dead OSD mount} —pgid {PG ID} —op
export —file {export file}
```

Copy the exported files to a location that can be accessed from within ceph-osd pods and import to one or more healthy OSDs using ceph-objectstore-tool. The target OSD must not be running, which means that the pod needs to be deactivated so it won't restart automatically if the ceph-osd process stops. For this reason it is advantageous to use a single import OSD if possible regardless of the number of PGs being imported.

Note that the target OSD pod for import is not the same as the OSD which is importing the PGs. The import target OSD must have its pod stopped in order to mount and write to its object store, but the tools need to be run on a running ceph-osd pod. The running pod will not be affected and is simply a stage with the proper tools to perform the required operations on the import target OSD.

```
# kubectl label nodes --all ceph_recover_window=inactive
# kubectl label nodes {target host} —overwrite ceph_recover_window=active
# kubectl patch -n ceph ds {import target ceph-osd daemon-set} -
p='{"spec":{"template":{"spec":{"nodeSelector":{"ceph-
osd":"enabled","ceph_recover_window":"inactive"}}}}}'
# kubectl -n ceph exec -it {running ceph-osd-pod} — /bin/bash
# mkdir /var/lib/ceph/osd/ceph-{OSD ID}
# mount {OSD partition} /var/lib/ceph/osd/ceph-{OSD ID}
# ceph-objectstore-tool —data-path /var/lib/ceph/osd/ceph-{OSD ID} —pgid {PG
ID} —op import —file {exported file}
# umount /var/lib/ceph/osd/ceph-{OSD ID}
# rmdir /var/lib/ceph/osd/ceph-{OSD ID}
# exit
# kubectl patch -n ceph ds {target ceph-osd daemon-set} -
p='{"spec":{"template":{"spec":{"nodeSelector":{"ceph-
osd":"enabled","ceph_recover_window":"active"}}}}}'
```

The ceph-osd pod should start up with the newly imported PG data, which will naturally be moved to the proper OSDs and the cluster should return to health.

If the dead OSD disks are not available or are not usable, then some level of data loss has been experienced and the best that can be done is to find the most up-to-date copy of the PG and mark it complete to allow the cluster to recover. The process is very similar to the import process above with

one difference. In this case ceph-objectstore-tool will be used to mark an incomplete copy of the PG as complete rather than to import a complete copy from another OSD's object store.

Find the most up-to-date copy of the PG by examining the peers in the PG query from above. Each peer will list an epoch and a timestamp for its most recent update for its copy of the PG. There will likely be multiple OSDs with the same epoch and timestamp. Any OSD will suffice, but ideally choose one that is currently acting as a peer for the PG and target it for marking the PG complete.

The same note applies here as for the import above. The target PG to be marked complete must have its OSD stopped and mounted on a separate, running ceph-osd pod in order to utilize the tools there for these operations.

```
# kubectl label nodes --all ceph_recover_window=inactive
# kubectl label nodes {target host} —overwrite ceph_recover_window=active
# kubectl patch -n ceph ds {mark-complete target ceph-osd daemon-set} -
p='{"spec":{"template":{"spec":{"nodeSelector":{"ceph-
osd":"enabled","ceph_recover_window":"inactive"}}}}}'
# kubectl -n ceph exec -it {running  ceph-osd-pod} — /bin/bash
# mkdir /var/lib/ceph/osd/ceph-{OSD ID}
# mount {OSD partition} /var/lib/ceph/osd/ceph-{OSD ID}
# ceph-objectstore-tool —data-path /var/lib/ceph/osd/ceph-{OSD ID} —pgid {PG
ID} —op mark-complete
# umount /var/lib/ceph/osd/ceph-{OSD ID}
# rmdir /var/lib/ceph/osd/ceph-{OSD ID}
# exit
# kubectl patch -n ceph ds {target ceph-osd daemon-set} -
p='{"spec":{"template":{"spec":{"nodeSelector":{"ceph-
osd":"enabled","ceph_recover_window":"active"}}}}}'
```

Naturally repeat these operations for as many imports or mark-completes as required to get all of the incomplete PGs into a complete state. Once this is accomplished, the cluster will backfill as necessary and return to health.

## Troubleshooting Networking Issues

Ceph Storage depends heavily on a reliable network connection. Ceph nodes use the network for communicating with each other. Networking issues can cause many problems with OSDs, such as flapping OSD, or OSD incorrectly reported as down. Networking issues can also cause Monitor clock skew errors. In addition, packet loss, high latency, or limited bandwidth can impact the cluster performance and stability.

### Procedure: Basic Networking Troubleshooting

1. Verify that the cluster_network and public_network parameters in the Ceph configuration file include correct values.
2. Verify that the network interfaces are up.
3. Verify that the Ceph nodes are able to reach each other using their short host names.
4. If you use a firewall, ensure that Ceph nodes are able to reach other on their appropriate ports.
5. Verify that there are no errors on the interface counters and that the network connectivity between hosts has expected latency and no packet loss.
6. For performance issues, in addition to the latency checks, also use the iperf utility to verify the network bandwidth between all nodes of the cluster.
7. Ensure that all hosts have equal speed network interconnects, otherwise slow attached nodes could slow down the faster connected ones. Also, ensure that the inter switch links can handle the aggregated bandwidth of the attached nodes.

## Procedure: Basic NTP Troubleshooting

Verify that the ntpd daemon is running on the Monitor hosts:

```
systemctl status ntpd
```

If ntpd is not running, enable and start it:

```
systemctl enable ntpdsystemctl start ntpd
```

Ensure that ntpd is synchronizing the clocks correctly:

```
ntpq -p
```

### Scripts

### Check MGR log

To verify the log of a Manager, verify the log sent by the Manager daemon to Kubernetes:

```
userid@mtn16r11o001:~$ kubectl -n ceph get pod -l
application=ceph,component=mgr -o wide
NAME              READY          STATUS    RESTARTS       AGE      IP
    NODE
{pod}             2/2            Running   1              2d       135.91.204.212
   mtn16r11o001
ceph-mgr-
nvj5p   2/2           Running   3             2d      135.91.204.228    mtn16
r12o001
...
userid@mtn16r11o001:~$ kubectl -n ceph  logs {pod}
```

### Check RADOSGW log

To verify the log of a RADOS Gateway, verify the log sent by the RADOS Gateway daemon to Kubernetes

```
userid@mtn16r11o001:~$ kubectl -n ceph  get pod -l
application=ceph,component=rgw -o wide
NAME               READY          STATUS    RESTARTS       AGE      IP
     NODE
{pod}              1/1            Running   0              3h       10.97.88.225
     mtn16r11o001
```

### Check MON log

To verify the log of a Monitor, verify the log sent by the daemon to Kubernetes:

```
userid@mtn16r11o001:~$ kubectl -n ceph  get pod -l
application=ceph,component=mon -o wide
NAME              READY          STATUS    RESTARTS       AGE      IP
   NODE
{pod}             2/2            Running   1              2d       135.91.204.212
   mtn16r11o001
{pod1}            2/2            Running   1              2d       135.91.204.228
   mtn16r12o002
{pod2}            2/2            Running   1              2d       135.91.204.229
   mtn16r12o001
...
userid@mtn16r11o001:~$ kubectl -n ceph  logs {pod} -c ceph-mon
```

### Check OSD log

To verify the log of an OSD, verify the log sent by the daemon to Kubernetes:

```
userid@mtn16r11o001:~$ kubectl -n ceph  get pod -l
application=ceph,component=osd -o wide
NAME              READY         STATUS    RESTARTS      AGE     IP
   NODE
{POD}             2/2           Running   0             4h      135.91.204.212
   mtn16r11o001
. . .
userid@mtn16r11o001:~$ kubectl -n ceph logs {POD} -c ceph-osd-default
```

## Failure Impact Analysis

### Ceph-Monitor Failure

Any single ceph-mon failure should not take down the entire monitor cluster as long as a majority of the monitor pods are available. If that is the case–the remaining pods are able to form a quorum–the ceph health command will report any problems:

```
kubectl -n ceph exec ${MON_POD} --  ceph health
HEALTH_WARN 1 mons down, quorum 0,2
```

and:

```
kubectl -n ceph exec -it ${MON_POD} -- ceph health detail
HEALTH_WARN 1 mons down, quorum 0,2
mon.b (rank 1) addr 192.168.106.220:6790/0 is down (out of quorum)
```

In the event a ceph-monitor pod fails, kubernetes will automatically attempt to restart the pod on the same kubernetes host, if that host is unavailable, kubernetes will attempt to deploy a ceph-mon pod on a kubernetes host with the ceph-mon label. Generally speaking, once the monitor pod restarts and enters into a running state, monitor services and functionality should return to a HEALTH_OK state. As long as there are sufficient Monitors running to have a quorate partition, there is not impact on the client IO traffic.

If there are not enough monitor pods to form a quorum, the ceph command will block trying to reach the cluster. In this situation, you need to get enough ceph-mon daemons running to form a quorum before doing anything else with the cluster.

### Impact of RGW Failure

The failure of an RGW service will have little impact on cluster operations as users will be interfacing with RGW behind a load-balancer. End-users may experience a temporary drop in performance drop in s3/swift operations as all client requests will now go through a smaller number of rados gateway instances.

In the event a single ceph-rgw pod fails, kubernetes will automatically attempt to restart the pod on the same kubernetes host, if that host is unavailable, kubernetes will attempt to deploy a ceph-rgw pod on a kubernetes host with the ceph-rgw label.

### Impact of Failed OSD/Rack

When any component within a cluster fails, be it a single OSD device, a host's worth of OSDs, or a larger bucket like a rack, Ceph waits for a short grace period before it marks the failed OSDs out. This state is then updated in the CRUSH map. As soon as an OSD is marked out, Ceph initiates recovery and backfill operations. During this process, Ceph moves or copies all data that was hosted on the OSD devices that failed using a valid copy that still exist in the cluster.

Since CRUSH replicates data to multiple OSDs, replicated copies survive and are read during recovery. As CRUSH chooses a new mapping for the PGs that were hosted on the failed

component, it will minimize the amount of data that must move to other, still-functioning hosts and OSD devices in order to complete recovery. This helps degraded objects (those that lost their copies due to failures), and thus the cluster as a whole becomes healthy once again as the replication demanded by the configured CRUSH rule policy is restored.

Once the failed OSD is replaced and added to the Ceph cluster, CRUSH will start renewing the cluster state, identify how the new PG placement should take place for all existing PGs and move the new OSD to Up Set (and subsequently Acting Set). Once an OSD is added to the Up Set, Ceph will start rebalancing data on the new OSD for the PGs it's supposed to hold. The data is moved from older hosts or OSDs into our new OSDs. Rebalancing operations are necessary to keep all the OSDs equally utilized and thus the placement and workload uniform. Multiple OSDs might be employed for pulling relevant data out for the PGs they own, but they will work in parallel to quickly move the data over.

The amount of data movement a new OSD or a new server can cause is directly proportional to the percentage of the ratio of its CRUSH weight to the original CRUSH weight of the rest of the cluster. Rebalancing should never cause the cluster to go offline, but it can cause significant slowdowns by saturating system and network resources, which to some end-users can feel like a system outage.

**-h, --help**

Show this help message and exit

**-e *EXE*, --exe *EXE***

Ceph executable [**/usr/bin/ceph**]

**-c *CONF*, --conf *CONF***

Alternative Ceph configuration file

**-m *MONADDRESS*, --monaddress *MONADDRESS***

Ceph monitor address [**:port**]

**-i *ID*, --id *ID***

Ceph client ID

**-n *NAME*, --name *NAME***

Ceph client name

**-k *KEYRING*, --keyring *KEYRING***

Ceph client keyring file

**-p *POOL*, --pool *POOL***

Ceph pool name

**-d, --detail**

Show pool details on WARN and CRITICAL alerts

**-W *WARN*, --warn *WARN***

Generate WARN alert when RAW USED exceeds this percent

**-C *CRITICAL*, --critical *CRITICAL***

Generate CRITICAL alert when RAW USED exceeds this percent

**-V, --version**

Show version and exit

# Airship Operations

## Rolling Upgrade

For performing a rolling upgrade, the manifest of the component that needs to be upgraded should be overwritten in the site manifests. Let's consider a simple example of increasing the replicas of promenade api. Now, create a manifest at site/<site-name>/software/charts/ucp/promenade/promenade.yaml with the increased set of replicas. A sample manifest is as follows –

```
schema: armada/Chart/v1
metadata:
  schema: metadata/Document/v1
  name: ucp-promenade-scaled
  layeringDefinition:
    abstract: false
    layer: site
    replacement: true
    parentSelector:
      name: ucp-promenade-scaled-type
    actions:
    - method: merge
      path: .
  storagePolicy: cleartext

data:
  values:
    pod:
      replicas:
        api: 6
```

The initial replica size of promenade-api is 2. Now it has been overwritten 6 in site manifests. After the updates to manifest has been done, generate a new bundle with the changes.

```
sh -c pegleg/tools/pegleg.sh site -r <site-manifest-dir-name> -e
secrets=<security-manifest-name> -e global=<global-manifest-name> collect
<site-name> -s <site_name_collected>
```

Now, create the configdocs from the newly generated bundle.

```
docker run -v < site_name_collected>:/target -e
'OS_AUTH_URL=http://<user>@domain.com' -e 'OS_PASSWORD=<shipyard password> -e
'OS_PROJECT_DOMAIN_NAME=default' -e 'OS_PROJECT_NAME=service' -e
'OS_USERNAME=shipyard' -e 'OS_USER_DOMAIN_NAME=default' -e
'OS_IDENTITY_API_VERSION=3' --rm --net=host <shipyard image> create
configdocs --replace --directory=/target <site name>
```

After creation, commit these docs.

```
docker run -v < site_name_collected>:/target -e
'OS_AUTH_URL=http://<user>@domain.com' -e 'OS_PASSWORD=<shipyard password> -e
'OS_PROJECT_DOMAIN_NAME=default' -e 'OS_PROJECT_NAME=service' -e
'OS_USERNAME=shipyard' -e 'OS_USER_DOMAIN_NAME=default' -e
'OS_IDENTITY_API_VERSION=3' --rm --net=host <shipyard image> commit
configdocs
```

Once the above 2 commands are executed without any errors, proceed to updating the software.

```
docker run -v < site_name_collected>:/target -e
'OS_AUTH_URL=http://<user>@domain.com' -e 'OS_PASSWORD=<shipyard password> -e
'OS_PROJECT_DOMAIN_NAME=default' -e 'OS_PROJECT_NAME=service' -e
'OS_USERNAME=shipyard' -e 'OS_USER_DOMAIN_NAME=default' -e
'OS_IDENTITY_API_VERSION=3' --rm --net=host <shipyard image> create action
update_software --allow-intermediate-commits
```

This will update the software in the cluster and increase the replicas of the promenade-api to 6.

```
kubectl get po -n ucp |grep promenade-api
promenade-api-5497bc7cf4-4f8d4                                    1/1
Running     0          10m
promenade-api-5497bc7cf4-5lc6v                                    1/1
Running     0          10m
promenade-api-5497bc7cf4-3e9a5                                    1/1
Running     0          10m
promenade-api-5497bc7cf4-27r1z                                    1/1
Running     0          10m
promenade-api-5497bc7cf4-6w0u1                                    1/1
Running     0          10m
promenade-api-5497bc7cf4-75q8z                                    1/1
Running     0          10m
```

## Deckhand manifest viewing

You can use shipyard cli to fetch the config docs and view the manifest files stored in deckhand. Issue the following command to view the raw config docs from deckhand.

```
docker run -v < site_name_collected>:/target -e
'OS_AUTH_URL=http://<user>@domain.com' -e 'OS_PASSWORD=<shipyard password> -e
'OS_PROJECT_DOMAIN_NAME=default' -e 'OS_PROJECT_NAME=service' -e
'OS_USERNAME=shipyard' -e 'OS_USER_DOMAIN_NAME=default' -e
'OS_IDENTITY_API_VERSION=3' --rm --net=host <shipyard image> get configdocs –
collection=<collection-name>
```

In order to get rendered config docs i.e. the config docs where all the overwriting has already been done using all the manifest layers, use the following command.

```
docker run -v < site_name_collected>:/target -e
'OS_AUTH_URL=http://<user>@domain.com' -e 'OS_PASSWORD=<shipyard password> -e
'OS_PROJECT_DOMAIN_NAME=default' -e 'OS_PROJECT_NAME=service' -e
'OS_USERNAME=shipyard' -e 'OS_USER_DOMAIN_NAME=default' -e
'OS_IDENTITY_API_VERSION=3' --rm --net=host <shipyard image> get
renderedconfigdocs –collection=<collection-name>
```

## Create workload

Before any workload can be deployed, python Openstack client needs to be installed. kolla/ubuntu-source-keystone:3.0.3 docker image can be used for this purpose. In order to use this image for executing the openstack client commands we need to pass the following arguments as environment variables to the image.

```
OS_AUTH_URL – the url of the keystone public endpoint
OS_PROJECT_DOMAIN_NAME – the domain name under which the project is created
OS_USER_DOMAIN_NAME – the domain name under which the user is created
OS_PROJECT_NAME – the project name which the user is authorized to access
OS_USERNAME – the user name
OS_PASSWORD – the respective password of the user.
OS_IDENTITY_API_VERSION=3
```

Let's deploy a sample workload. To do that, first create a SR-IOV port using the following heat template

```
---
heat_template_version: 2015-04-30
parameters:
  sriov_port:
    default: <default port name>
    description: port
    type: string
  sriov_provider_net_id:
    default: <default net id>
    description: "Network ID for the server"
    type: string
resources:
  server1_port:
    properties:
      ? "binding:vnic_type"
      : direct
      name:
        get_param: sriov_port
      network_id:
        get_param: sriov_provider_net_id
    type: "OS::Neutron::Port"
```

Now create the port using the openstack client and the above yaml.

```
docker run --rm --net=host -v <above yaml file path>:/tmp/test.yaml -e
OS_AUTH_URL=http://<user>@<example domain> -e OS_PROJECT_DOMAIN_NAME=default
-e OS_USER_DOMAIN_NAME=default -e OS_PROJECT_NAME=admin -e OS_USERNAME=admin
-e OS_PASSWORD=<admin password> -e OS_IDENTITY_API_VERSION=3 kolla/ubuntu-
source-keystone:3.0.3 openstack stack create -f /tmp/test.yaml testport
```

**Note:** If Openstack client is installed on the host then export the environment variables mentioned above and run the openstack cli commands directly on host

```
openstack stack create -f /tmp/test.yaml testport

 openstack stack list
+--------------------------------------+------------+-----------------+------
---------------+
| id                                   | stack_name | stack_status    |
creation_time        |
+--------------------------------------+------------+-----------------+------
---------------+
| 433e261a-bac4-43e3-b854-c2d8fd462329 | testport   | CREATE_COMPLETE | 2019-
08-07T10:23:31Z |
+--------------------------------------+------------+-----------------+------
---------------+
```

Now a VM can be created using the above mentioned port.

```
openstack server create Test_VM --image=d4f41b4f-95f5-41b1-97e7-050964426145
--flavor=51786bb98fda46368e17010c7b6931a8 --nic port-id=18558cad-2042-440d-
9e83-8e039211c783


+--------------------------------------+--------------------------------------
-----------+
| Property                             | Value
|
+--------------------------------------+--------------------------------------
-----------+
| OS-DCF:diskConfig                    | MANUAL
|
| OS-EXT-AZ:availability_zone          | nova
|
| OS-EXT-SRV-ATTR:host                 | -
|
| OS-EXT-SRV-ATTR:hypervisor_hostname  | -
|
| OS-EXT-SRV-ATTR:instance_name        | instance-000006
|
| OS-EXT-STS:power_state               | 0
|
| OS-EXT-STS:task_state                | scheduling
|
| OS-EXT-STS:vm_state                  | building
|
| OS-SRV-USG:launched_at               | -
|
| OS-SRV-USG:terminated_at             | -
|
| accessIPv4                           |
|
| accessIPv6                           |
|
```

```
| adminPass                                | <admin pass>
|
| config_drive                             |
|
| created                                  | 2019-08-07T10:25:03Z
|
| flavor                                   | test
(51786bb98fda46368e17010c7b6931a8) |
| hostId                                   |
|
| id                                       | 4fdfded4-55d7-45c8-9396-5ddd0da5ebf0
|
| image                                    | TestVM (d4f41b4f-95f5-41b1-97e7-
050964426145)  |
| key_name                                 | -
|
| metadata                                 | {}
|
| name                                     | Test_VM
|
| os-extended-volumes:volumes_attached | []
|
| progress                                 | 0
|
| scheduler_hints                          | {}
|
| security_groups                          | default
|
| status                                   | BUILD
|
| tenant_id                                | 03775f6dcdf446df920652d5e4d696e3
|
| updated                                  | 2019-08-07T10:25:03Z
|
| user_id                                  | gj6620
|
+------------------------------------------+--------------------------------------
-----------+
```

## Delete Workload

With the python Openstack client pre-installed; kolla/ubuntu-source-keystone:3.0.3 docker image can be used for deleting a workload.

```
docker run --rm --net=host -e OS_AUTH_URL=http://<user>@<example domain> -e
OS_PROJECT_DOMAIN_NAME=default -e OS_USER_DOMAIN_NAME=default -e
OS_PROJECT_NAME=admin -e OS_USERNAME=admin -e OS_PASSWORD=<admin password> -e
OS_IDENTITY_API_VERSION=3 kolla/ubuntu-source-keystone:3.0.3 openstack server
delete Test_VM
Request to delete server Test_VM has been accepted.
```

You can then check the progress using the below mentioned command.

```
docker run --rm --net=host -e OS_AUTH_URL=http://<user>@<example domain> -e
OS_PROJECT_DOMAIN_NAME=default -e OS_USER_DOMAIN_NAME=default -e
OS_PROJECT_NAME=admin -e OS_USERNAME=admin -e OS_PASSWORD=<admin password> -e
OS_IDENTITY_API_VERSION=3 kolla/ubuntu-source-keystone:3.0.3 openstack stack-
delete testport
+--------------------------------------+------------+--------------------+---
------------------+
| id                                   | stack_name | stack_status       |
creation_time        |
+--------------------------------------+------------+--------------------+---
------------------+
| 433e261a-bac4-43e3-b854-c2d8fd462329 | testport   | DELETE_IN_PROGRESS |
2019-08-07T10:23:31Z |
+--------------------------------------+------------+--------------------+---
------------------+
```

## Use diving bell to install / upgrade host level packages

In order to manipulate host level packages, diving bell manifest can be used. This process is similar to rolling upgrade process mentioned earlier. The diving bell manifest is overwritten and necessary host packages information is added. Afterwards an update_software is performed for the site.

Consider the following sample diving bell site manifest.

```
---
schema: armada/Chart/v1
metadata:
  schema: metadata/Document/v1
  name: ucp-divingbell
  layeringDefinition:
    replacement: True
    abstract: false
    layer: site
    parentSelector:
      name: ucp-divingbell-global
    actions:
    - method: merge
      path: .
  storagePolicy: cleartext


values:
     conf:
       apt:
         blacklistpkgs: []
         packages:
           - name: vim
```

Once the changes have been performed to the diving bell manifest, generate a new bundle with these changes.

```
sh -c pegleg/tools/pegleg.sh site -r site-manifests -e secrets= security-
manifests -e global=manifests collect <site-name> -s <site_name_collected>
```

Now, create and commit the configdocs from the newly generated bundle.

```
docker run -v < site_name_collected>:/target -e
'OS_AUTH_URL=http://<user>@domain.com' -e 'OS_PASSWORD=<shipyard password> -e
'OS_PROJECT_DOMAIN_NAME=default' -e 'OS_PROJECT_NAME=service' -e
'OS_USERNAME=shipyard' -e 'OS_USER_DOMAIN_NAME=default' -e
'OS_IDENTITY_API_VERSION=3' --rm --net=host <shipyard image> create
configdocs --replace --directory=/target <site name>


docker run -v < site_name_collected>:/target -e
'OS_AUTH_URL=http://<user>@domain.com' -e 'OS_PASSWORD=<shipyard password> -e
'OS_PROJECT_DOMAIN_NAME=default' -e 'OS_PROJECT_NAME=service' -e
```

```
'OS_USERNAME=shipyard' -e 'OS_USER_DOMAIN_NAME=default' -e
'OS_IDENTITY_API_VERSION=3' --rm --net=host <shipyard image> commit
configdocs
```

Once the above 2 commands are executed without errors, proceed to updating the software.

```
docker run -v < site_name_collected>:/target -e
'OS_AUTH_URL=http://<user>@domain.com' -e 'OS_PASSWORD=<shipyard password> -e
'OS_PROJECT_DOMAIN_NAME=default' -e 'OS_PROJECT_NAME=service' -e
'OS_USERNAME=shipyard' -e 'OS_USER_DOMAIN_NAME=default' -e
'OS_IDENTITY_API_VERSION=3' --rm --net=host <shipyard image> create action
update_software --allow-intermediate-commits
```

After the above action is completed, the package installation can be verified on the host as follows –

```
dpkg -l|grep -E "vim|nmap"
ii  vim                              2:7.4.1689-3ubuntu1.2
amd64        Vi IMproved - enhanced vi editor
ii  vim-common                       2:7.4.1689-3ubuntu1.2
amd64        Vi IMproved - Common files
ii  vim-runtime                      2:7.4.1689-3ubuntu1.2
all          Vi IMproved - Runtime files
ii  vim-tiny                         2:7.4.1689-3ubuntu1.2
amd64        Vi IMproved - enhanced vi editor - compact version
```

# Frequently Asked Questions

## What is a good starting point to develop understanding of airship?

Firstly, a good understanding of each airship component should be attained. Afterwards, familiarity with the YAML manifest files of airship is highly recommended. Airship treasuremap contains the 'seaworthy' manifest files, which can be explored to develop firm understanding of how configuration of airship is done.

## What is the minimal environment hardware for airship?

For starters, minimal hardware setup can be 4 control + 3 workers node or 3 control + 4 worker nodes. However, workers sizing should be done by keeping VNF resource requirements in mind, so that the workers have enough resources to provide once all the components of VNF are deployed, there is no resource starvation. VNFs have been observed running out of resources in test environments because of shortfalls in planning. Deployment engineers can skip on aspects like redundancy in C-leafs etc to reduce cost for test environments.

## Can commodity hardware be used for deploying airship?

Technically any commodity hardware can be used to deploy hardware. However, to follow the production environment deployed by operators like AT&T, It is highly recommended that lab environments should be built using the hardware that AT&T uses. The details of the hardware are mentioned earlier in the book. Using any other hardware would require re-certification and extra troubleshooting effort to get things working. This can delay the onboarding process. Furthermore, for SR-IOV workloads, the recommended hardware uses 25G cards. Any other hardware like 10G wouldn't yield the optimal test results.. This is important that the test environments uses 25G cards at minimum.

## Does airship support the use of tenant networks?

All the SR-IOV networks are provider networks.

## Is using SR-IOV in tenant network mandatory?

Currently only provider network is used for SR-IOV and OVS. In current deployments of airship, there is support for SR-IOV and OVS kernel. Tenants should use SR-IOV as a data path for tenant vlans. They can still use OvS for OAM connectivity. In future version of airship, there will be SR-IOV and OvS-DPDK as data path technologies on same host & tenants would be able to choose any of those technologies. But again for high performance/network thirsty applications still should use SR-IOV over OvS-DPDK.

## Is using SR-IOV mandatory on a provide network that is not performance critical?

Both SR-IOV and OvS-DPDK are supported. Tenant can attach to either of them. OvS should be used for less performance critical apps. For high performance its suggested to use SR-IOV.

## The airship Calico BGP peers with the AIRSHIP Network infrastructure. Where does the calico peer to?

Calico BGP peering is with the C-leaf.

## Is there VRRP setup between c-leaf's to serve as the redundant gateway's for the dell servers?

Yes, there is VRRP between the c-leaf's to provide redundancy.

## What kind of spanning tree protocol is used?

There is no STP or VSTP to do per VLAN STP. Instead EVPN Multihoming is used.

## Are 25 GB ports configured with any special LACP options to handle the bonding?

The recommended configuration is as follows:

2 x 25G for SR-IOV/OVS–DPDK (Data Path). – There is no LACP or Bonding used here.

A second 2x25G for OAM/Hypervisor access/Storage –LACP can be used on these two ports.

## While working with SR-IOV networks in Openstack, what kind of VLAN creation options are there?

Traffic for a VNF enters the C leaf and it gets switched at layer 2 towards the said VNF, hence its VLAN based. Each tenant has its own separate VLAN.

While provisioning SR-IOV networks in Openstack the platform supports a variety of configurations. 2 flat network topologies can be created using VLAN 0 in Openstack so that the virtual function has no filter. Furthermore, there can also be 2 VLAN based networks using provider network with SR-IOV NIC and use them in guest.

## While provisioning SR-IOV ports what kind of IP addresses are used at Openstack level?

Fake IP addresses in Openstack has been used in some production deployments. This is how some of the sensitive tenants like Mobility use the SRIOV networks. A common way to assign fake IP address is to use the 169.x.x.x IP network. A deployment engineer can view this fake IP range in the Openstack dashboard for the SR-IOV networks & ports. This fake IP addresses mean nothing and are used only to provide Openstack with some view of the IPs. The real IPs and subnets of SRIOV networks are completely different and provisioned on the VMs, and BGP communicated by VMs to the R-leaf.

## The Azure deployment setup described in this book is testing only or can also be used for production deployment?

The azure deployment process outlined in the document is for testing and not for a production scale. The current scope of airship is for on-prem setup and not public cloud.

## Is there any option for VNF (tenant only and/or SR-IOV) to talk BGP/ECMP to the edge routers to manage traffic flows for redundancy.

Yes. The gateway VNFs can have BGP relations to the R leaf and advertise VNF availability through BGP route advertisement.

## What is the scope / expectation of IPv6 support?

If the underlying network fabric is IPv6 capable then the expectation is the VNF should be able to support the bearer and OAM traffic using IPv6 addresses.

## What are the options to spread traffic across multiple racks & AZs in one airship deployments.

Each rack in AIRSHIP is a part of 1 AZ, so a total of 4 AZ exist in one deployment of AIRSHIP. There is 1 aggregate spanning across these 4 AZs. Tenant can span over multiple Racks / AZs and get better resiliency.

## Is there an expectation of L2 networking and/or VLANs to span multiple airship instances?

No. The VLAN ranges / ip subnet spaces are all contained within one AIRSHIP instance. There is no req to span L2 to multiple airship instances.

## What are some recommended hardware specs?

CPU model: Dell R740 Purley Servers - 44 cores/88 vCPUs with HT enabled

CPU sockets per servers? 2 CPU sockets per server

NUMA nodes per socket? 2 NUMA total, 1 per socket

Hyperthreading enabled & part of the sizing calculations.

NIC: 25G NIC dual port - XXV710-DA2. See host profile for more details.

For details see the recommended hardware section.

## How SR-IOV and VLAN tagging is handled in airship?

Airship today supports SR-IOV as the data path agent and allows Virtual Functions (VF) to be attach to a tenant VM. The traffic destined to VM is forwarded to attached VF as it is passed on to the network layer (i.e. with the Vlan tags attached to packet). Compute host/fabric doesn't perform any strip/un-strip operations on the packets. There can be a Q-in-Q packet where the forwarding shall be done on basis of outer tag. All these operations have to be performed by VM. Furthermore, the inner tag shall not be configured anywhere on the data path (neither on neutron port/VF/fabric). Today there is a limit with i40e driver on number of VLAN filters that can be set (max 8 VLAN filters), if using trunk support on VF.

It is in the roadmap to enable AIRSHIP release to allow setting up trunk function on SR-IOV VF. It can be done today using custom i40e driver sysfs commands for testing purposes. The support to enable configuration on VF from OpenStack Neutron code in being worked out.