Promenade Documentation

Release 0.1.0

Promenade Authors

Sep 27, 2021

Contents

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

CHAPTER 1

User's Guide

1.1 Promenade Configuration Guide

1.1.1 Developer On-boarding

Overview

Functionality:

- Airship Bootstrapping
- Core Kubernetes Management
- Misc.

Code structure:

1. Jinja templates (`promenade/templates/**`).
2. Helm charts for Kubernetes components, etcd, CoreDNS, Promenade `charts/**`.
3. Python support code.
    - API
    - Config access object
    - CLI
    - Certificate generation code

Since Promenade is largely templates + charts, unit testing is not enough to provide confidence for code changes. Significant functional testing is required to test small changes to, e.g. the `etcd` or `calico` charts, which can completely break deployment (or break reboot recovery). Developers can run the functional tests locally:

```
./tools/setup_gate.sh  # Run once per machine you're on, DO NOT USE SUDO.
./tools/gate.sh        # Runs a 4 node resiliency test.
```

This runs the test defined in `tools/g2/manifests/resiliency.json`. There are a few additional test scenarios defined in adjacent files.

There are helpful tools for troubleshooting these gates in `tools/g2/bin/*`, including `tools/g2/bin/ssh.sh`, which will let you ssh directly to a node to debug it, e.g.:

```
./tools/g2/bin/ssh.sh n0
```

Running Resiliency Tests Behind Corporate Proxy

If your development environment is behind a corporate proxy, you will need to update following files to add your envrionment's proxy information, dns, or possibly your internal ntp servers, in order to deploy airship:

- *charts/coredns/values.yaml*: Update the upstream coredns nameserver IPs to your internal DNS addresses.
- *examples/basic/KubernetesNetwork.yaml*: Since resiliency manifest uses the examples/basic environment configuration, you will need to Update the kubernetes network configuration in this folder. Update the upstream nameserver IPs to your internal DNS addresses. Add the http(s) proxy URL and additional_no_proxy list. Also, if your enviornment requires that, update the ntp server list to your internal ntp server addresses for more reliable time sync.
- *tools/g2/templates/network-config.sub*: Update the upstream nameserver IPs to your internal DNS addresses.

Bootstrapping

Promenade is responsible for converting a vanilla Ubuntu 16.04 VM into a proper Airship.

How You Run It

Assuming you have a valid set of configuration. Generate *genesis.sh*, which is a self-contained script for bootstrapping the genesis node.

```
promenade build-all -o output-dir config/*.yaml
```

What `genesis.sh` does:

1. Basic host validation (always room for more).
2. Drops pre-templated files in place:
    - Manifests to run initial Kubernetes components `/etc/kubernetes/manifests`
        - Basic components (apiserver, scheduler, controller-manager)
        - Etcd
        - Auxiliary Etcd
    - Docker configuration
    - Kubelet configuration
    - Apt configuration (proxy)
    - Bootstrapping Armada configuration
        - Dedicated Tiller
        - Dedicated Kubernetes API server

    - API server points at auxiliary etcd.
3. Installs some apt packages (docker + user-defined)
4. Starts Docker and Kubernetes.
5. Waits for bootstrapping services to be up (healthy Kubernetes API).
6. Applies configured labels to node.
7. Waits for Armada to finish bootstrapping deployment.
8. Final host validation.

When it's done, you should have a working Airship deployed as defined by your configuration (e.g. with or without LMA, keystone, etc) with no configuration loaded into Deckhand (via Shipyard).

How It Works

The templates that get dropped in place generally live in `promenade/templates/**`. The genesis node gets everything under `roles/genesis/**` and `roles/common/**` directly in place. Note that the templates under `roles/join/**` are used instead of the files under `genesis` for joining nodes to the existing cluster.

The "real" work happens inside `kubelet` managed "static" pods (defined by flat files in `/etc/kubernetes/manifests`), primarily via Armada.

Charts do a bunch of work to take control of essentially everything behind the scenes. Trickiest is `etcd`, for which we run multiple server processes to keep the cluster happy throughout bootstrapping + initial node join.

Note that we deploy two separate etcd clusters: one for Kubernetes itself, and one for Calico. The Calico one is a bit less sensitive.

Anchor Pattern

To provide increased resiliency, we do something a bit unusual with the core components. We run a `DaemonSet` for them which simply copy static `Pod` definitions into the `/etc/kubernetes/manifests` directory on the hosts (along with any supporting files/configuration). This ensures that these workloads are present even when the Kubernetes API server is unreachable. We call this pattern the `Anchor` pattern.

The following components follow this pattern:

- Kubernetes core components
    - API server
    - Scheduler
    - Controller Manager
- Kubernetes etcd
- Calico etcd
- HAProxy (used for API server discovery)

The HAProxy `DaemonSet` runs on every machine in the cluster, but the others only run on "master" nodes.

Kubernetes Cluster Management

Promenade is responsible for managing the Kubernetes lifecycle of nodes. That primarily consists of "joining" them to the cluster and adding labels, but also includes label updates and node removal.

Node Join

This is done via a self-contained script that is obtained by Drydock querying the Promenade API `GET /api/v1.0/join-scripts` (and providing a configuration link to Deckhand originally specified by Shipyard).

The join script is delivered to the node by Drydock and executed via a systemd unit. When it runs, it follows a similar pattern to `genesis.sh`, but naturally does not use any Kubernetes bootstrapping components or run Armada:

1. Basic host validation (always room for more).
2. Drops pre-templated files in place:
    - Docker configuration
    - Kubelet configuration
    - Apt configuration (proxy)
3. Installs some apt packages (docker + user-defined)
4. Starts Docker and Kubernetes.
5. Waits for node to be recognized by Kubernetes.
6. Applies configured labels to node.
7. Final host validation.

After the node has successfully joined, the systemd unit disables itself so that it is not run again on reboot (though it would be safe to do so).

Other Management Features

Re-labeling and node removal API development has been delayed for other priorities, but is recently underway. While changing labels is generally easy, there are a few trickier bits around Kubelet and etcd management.

It is currently possible to fully de-label and remove a node from the cluster using a script that gets placed on each node (it requires `kubectl` so that must be in place), but that work is not exposed via API yet. The resiliency gate exercises this to reprovision the genesis node as a normal node.

Miscellaneous

Promenade does a few bits of additional work that's hard to classify, and probably don't belong in scope long term. Most notably is certificate generation.

Certificate generation is configured by the `PKICatalog` configuration document, which specifies the details for each certificate (CN, groups, hosts). Promenade then translates those requirements into calls to `cfssl`. The following will create a `certificates.yaml` file in `output-dir` containing all the generated certs:

```
promenade generate-certs -o output-dir config/*.yaml
```

If there are existing certs in `config/*.yaml`, then they will be used if applicable.

Troubleshooting

The context for this section is the functional gates described above. You can run them with:

```
./tools/gate.sh <gate_name>
```

When something goes wrong with this, you can ssh into individual nodes for testing (the nodes are named `n0` through `n3`):

```
./tools/g2/bin/ssh.sh <node_name>
```

When you get into a node and see various failures, or have an Armada error message saying a particular chart deployment failed, it is important to assess the overall cluster rather than just digging into the first thing you see. For example, if there is a problem with `etcd`, it could manifest as the Kubernetes API server pods failing.

Here is an approximate priority list of what to check for health (i.e. things higher up in the list break things lower down):

1. Kubernetes etcd
2. Kubernetes API Server
3. Other Kubernetes components (scheduler, controller-manager, kubelet).
4. Kubernetes proxy
5. Calico etcd
6. Calico node
7. DNS (CoreDNS)

For almost any other application, all of the above must be healthy before they will function properly.

1.1.2 Design

Promenade is a Kubernetes cluster deployment tool with the following goals:

- Resiliency in the face of node loss and full cluster reboot.
- Bare metal node support without external runtime dependencies.
- Providing a fully functional single-node cluster to allow cluster-hosted tooling to provision the remaining cluster nodes.
- Helm chart managed component life-cycle.
- API-managed cluster life-cycle.

Cluster Bootstrapping

The cluster is bootstrapped on a single node, called the genesis node. This node goes through a short-lived bootstrapping phase driven by static pod manifests consumed by `kubelet`, then quickly moves to chart-managed infrastructure, driven by Armada.

During the bootstrapping phase, the following temporary components are run as static pods which are configured directly from Promenade's configuration documents:

- Kubernetes core components
  - `apiserver`
  - `controller-manager`
  - `scheduler`
- Etcd for use by the Kubernetes `apiserver`
- Helm's server process `tiller`

- CoreDNS to be used for Kubernetes `apiserver` discovery

With these components up, it is possible to leverage Armada to deploy Helm charts to manage these components (and additional components) going forward.

Though completely configurable, a typical Armada manifest should specify charts for:

- Kubernetes components
  - `apiserver`
  - `controller-manager`
  - `proxy`
  - `scheduler`
- Cluster DNS (e.g. CoreDNS)
- Etcd for use by the Kubernetes `apiserver`
- A CNI provider for Kubernetes (e.g. Calico)
- An initial under-cloud system to allow cluster expansion, including components like Armada, Deckhand, Drydock and Shipyard.

Once these charts are deployed, the cluster is validated (currently, validation is limited to resolving DNS queries and verifying basic Kubernetes functionality including `Pod` scheduling log collection), and then the genesis process is complete. Additional nodes can be added to the cluster using day 2 procedures.

After additional master nodes are added to the cluster, it is possible to remove the genesis node from the cluster so that it can be fully re-provisioned using the same process as for all the other nodes.

Life-cycle Management

There are two sets of resources that require life-cycle management: cluster nodes and Kubernetes control plane components. These two sets of resources are managed differently.

Node Life-Cycle Management

Node life-cycle management tools are provided via an API to be consumed by other tools like Drydock and Shipyard.

The life-cycle operations for nodes are:

1. Adding a node to the cluster
2. Removing a node from the cluster
3. Adding and removing node labels.

Adding a node to the cluster

Adding a node to the cluster is done by running a shell script on the node that installs the `kubelet` and configures it to find and join the cluster. This script can either be generated up front via the CLI, or it can be obtained via the *join-scripts* endpoint of the API (development of this API is in-progress).

Nodes can only be joined assuming all the proper configuration documents are available, including required certificates for Kubelet.

Removing a node from the cluster

This is currently possible by leveraging the `promenade-teardown` script placed on each host. API support for this function is planned, but not yet implemented.

Adding and removing node labels

Promenade provides `node-labels` API for updating node labels. For more information about updating node labels, please reference the *Promenade API Documentation*.

It through relabeling nodes that key day 2 operations functionality like moving a master node are achieved.

Control-Plane Component Life-Cycle Management

With the exception of the Docker daemon and the `kubelet`, life-cycle management of control plane components is handled via Helm chart updates, which are orchestrated by Armada.

The Docker daemon is managed as an APT package, with configuration installed at the time the node is configured to join the cluster.

The `kubelet` is directly installed and configured at the time nodes join the cluster. Work is in progress to improve the upgradability of `kubelet` via either a system package or a chart.

Resiliency

The two primary failure scenarios Promenade is designed to be resilient against are node loss and full cluster restart.

Kubernetes has a well-defined High Availability pattern, which deals well with node loss.

However, this pattern requires an external load balancer for `apiserver` discovery. Since it is a goal of this project for the cluster to be able to operate without ongoing external dependencies, we must avoid that requirement.

Additionally, in the event of full cluster restart, we cannot rely on any response from the `apiserver` to give any `kubelet` direction on what processes to run. That means, each master node must be self-sufficient, so that once a quorum of Etcd members is achieved the cluster may resume normal operation.

The solution approach is two-pronged:

1. Deploy a local discovery mechanism for the `apiserver` processes on each node so that core components can always find the `apiservers` when their nodes reboot.
2. Apply the Anchor pattern described below to ensure that essential components on master nodes restart even when the `apiservers` are not available.

Currently, the discovery mechanism for the `apiserver` processes is provided by CoreDNS via a zone file written to disk on each node. This approach has some drawbacks, which might be addressed in future work by leveraging a HAProxy for discovery instead.

Anchor Pattern

The anchor pattern provides a way to manage process life-cycle using Helm charts in a way that allows them to be restarted immediately in the event of a node restart – even when the Kubernetes `apiserver` is unreachable.

In this pattern, a `DaemonSet` called the `anchor` that runs on selected nodes and is responsible for managing the life-cycle of assets deployed onto the node file system. In particular, these assets include a Kubernetes `Pod` manifest to be

consumed by `kubelet` and it manages the processes specified by the `Pod`. That management continues even when the node reboots, since static pods like this are run by the `kubelet` even when the `apiserver` is not available.

Cleanup of these resources is managed by the `anchor` pods' `preStop` life-cycle hooks. This is usually simply removing the files originally placed on the nodes' file systems, but, e.g. in the case of Etcd, can actually be used to manage more complex cleanup like removal from cluster membership.

Pod Checkpointer

Before moving to the Anchor pattern above, the pod-checkpointer approach pioneered by the Bootkube project was implemented. While this is an appealing approach, it unfortunately suffers from race conditions during full cluster reboot.

During cluster reboot, the checkpointer copies essential static manifests into place for the `kubelet` to run, which allows those components to start and become available. Once the `apiserver` and `etcd` cluster are functional, `kubelet` is able to register the failure of its workloads, and delete those pods via the API. This is where the race begins.

Once those pods are deleted from the `apiserver`, the pod checkpointer notices that the flagged pods are no longer scheduled to run on its node and then deletes the static manifests for those pods. Concurrently, the `controller-manager` and `scheduler` notice that new pods need to be created and scheduled (sequentially) and begin that work.

If the new pods are created, scheduled and started on the node before pod checkpointers on other nodes delete their critical services, then the cluster may remain healthy after the reboot. If enough nodes running the critical services fail to start the newly created pods before too many are removed, then the cluster does not recover from hard reboot.

The severity of this race is exacerbated by:

1. The sequence of events required to successfully replace these pods is long (`controller-manager` must create pods, then `scheduler` can schedule pods, then `kubelet` can start pods).
2. The `controller-manager` and `scheduler` may need to perform leader election during the race, because the leader might have been killed early.
3. The failure to recover any one of the core sets of processes can cause the entire cluster to fail. This is somewhat trajectory-dependent, e.g. if at least one `controller-manager` is scheduled before the `controller-manager` processes are all killed, then assuming the other processes are correctly restarted, then the `controller-manager` will also recover.
4. `etcd` is somewhat more sensitive to this race, because it requires two successfully restarted pods (assuming a 3 node cluster) rather than just one as the other components.

This race condition was the motivation for the construction and use of the Anchor pattern. In future versions of Kubernetes, it may be possible to use built-in checkpointing from the `kubelet`.

Alternatives

- Kubeadm
  - Does not yet support HA
  - Current approach to HA Etcd is to use the etcd opreator, which recovers from cluster reboot by loading from an external backup snapshot
  - Does not support chart-based management of components
- kops
  - Does not support bare metal

- Bootkube
  - Does not support automatic recovery from a full cluster reboot
  - Does not yet support full HA
  - Adheres to different design goals (minimal direct server contact), which makes some of these changes challenging, e.g. building a self-contained, multi-master cluster
  - Does not support chart-based management of components

1.1.3 Getting Started

Note: This document is meant to give a general understanding of how Promenade could be exercised in a development environment or for general learning and understanding. For holistic Airship deployment procedures, refer to Treasuremap

Basic Deployment

This approach is quick to get started, but generates the scripts used for joining up-front rather than generating them in the API as needed.

Setup Build Machine

On the machine you wish to use to generate deployment files, install docker:

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

This can be the same machine you intend to be the Genesis host, or it may be a separate build machine.

Generate Build files

To create the certificates and scripts needed to perform a basic deployment, you can use the following helper script on your build machine:

```
sudo ./tools/simple-deployment.sh examples/basic build
```

This will copy the configuration provided in the `examples/basic` directory into the `build` directory. Then, it will generate self-signed certificates for all the needed components in Deckhand-compatible format. Finally, it will render the provided configuration into directly-usable `genesis.sh` and `join-<NODE>.sh` scripts.

Genesis Host Provision

Install Ubuntu 16.04 on the machine intended to be the genesis host. Ensure the host has outbound internet access and DNS resolution. Ensure that the hostname matches the hostname specified in the Genesis.yaml file used to build the above configurations.

Execution

Perform the following steps to execute the deployment:

1. Copy the `genesis.sh` script to the genesis node and run it as sudo. In the event of runtime errors, refer to *Genesis Troubleshooting*
2. Validate the genesis node by running `validate-genesis.sh` on it.
3. Nodes for which `join-<NODE>.sh` scripts have been generated should be provisioned at this point, and need to have network connectivity to the genesis node. (This could be a manual Ubuntu provision, or a Drydock-initiated PXE boot in the case of a full fledged Airship deployment).
4. Join master nodes by copying their respective `join-<NODE>.sh` scripts to them and running them.
5. Validate the master nodes by copying and running their respective `validate-<NODE>.sh` scripts on each of them.
6. Re-provision the Genesis node
    a) Run the `/usr/local/bin/promenade-teardown` script on the Genesis node:
    b) Delete the node from the cluster via one of the other nodes `kubectl delete node <GENESIS>`.
    c) Power off and re-image the Genesis node.
    d) Join the genesis node as a normal node using its `join-<GENESIS>.sh` script.
    e) Validate the node using `validate-<GENSIS>.sh`.
7. Join and validate all remaining nodes using the `join-<NODE>.sh` and `validate-<NODE>.sh` scripts described above.

API-Driven Deployment

This approach leverages the Promenade API to fetch join scripts as needed. This is the approach used in the functional testing discussed below.

Setup

Follow the setup instructions above for *Basic Deployment*. Then, start a webserver to serve configuration to Promenade.

```
cat build/*.yaml > promenade.yaml
mv promenade.yaml build/promenade.yaml
docker rm -fv promenade-nginx
docker run -d \
    -p 7777:80 \
    --restart=always \
    --name promenade-nginx \
    -v build:/usr/share/nginx/html:ro \
        nginx:stable
export DESIGN_REF=http://192.168.77.1:7777/promenade.yaml
```

Execution

Perform the following steps to execute the deployment:

1. Copy the `genesis.sh` script to the genesis node and run it.
2. Validate the genesis node by running `validate-genesis.sh` on it.
3. Generate join script for a node using:

```
URL=http://promenade-api.ucp.svc.cluster.local/api/v1.0/join-scripts?
URL="${URL}design_ref=${DESIGN_REF}"
URL="${URL}&hostname=<HOSTNAME>&ip=<IP>"
URL="${URL}&labels.dynamic=calico-etcd=enabled"
URL="${URL}&labels.dynamic=kubernetes-apiserver=enabled"
URL="${URL}&labels.dynamic=kubernetes-controller-manager=enabled"
URL="${URL}&labels.dynamic=kubernetes-etcd=enabled"
URL="${URL}&labels.dynamic=kubernetes-scheduler=enabled"
URL="${URL}&labels.dynamic=ucp-control-plane=enabled"
curl -Lo join-<NODE>.sh "${URL}"
```

4. Copy the join script to the node, and run it via `bash join-<NODE>.sh`.
5. Repeat 3 and 4 until all nodes are joined.
6. Reprovision the Genesis node by tearing it down as above in *Basic Deployment*, then generating and using a join script for it as done in 3 and 4.

Running Tests

Initial Setup of Virsh Environment

To setup a local functional testing environment on your Ubuntu 16.04 machine, run:

```
./tools/setup_gate.sh
```

Running Functional Tests

To run complete functional tests locally:

```
./tools/gate.sh
```

For more verbose output, try:

```
PROMENADE_DEBUG=1 ./tools/gate.sh
```

For extremely verbose output, try:

```
GATE_DEBUG=1 PROMENADE_DEBUG=1 ./tools/gate.sh
```

The gate leaves its test VMs running for convenience. To shut everything down:

```
./tools/stop_gate.sh
```

To run a particular set of functional tests, you can specify the set on the command line:

```
./tools/gate.sh <SUITE>
```

Valid functional test suites are defined by JSON files that live in `tools/g2/manifests`.

Utilities

There are a couple of helper utilities available for interacting with gate VMs. These can be found in `tools/g2/bin`. The most important is certainly `ssh.sh`, which allows you to connect easily to test VMs:

```
./tools/g2/bin/ssh.sh n0
```

Development

Using a Local Registry

Repeatedly downloading multiple copies images during development can be quite slow. To avoid this issue, you can run a docker registry on the development host:

```
./tools/registry/start.sh
./tools/registry/update_cache.sh
```

Then, the images used by the basic example can be updated using:

```
./tools/registry/update_example.sh
```

That change can be undone via:

```
./tools/registry/revert_example.sh
```

The registry can be stopped with:

```
./tools/registry/stop.sh
```

Building the image

To build the image directly, you can use the standard Docker build command:

```
docker build -t promenade:local .
```

To build the image from behind a proxy, you can:

```
export http_proxy=...
export no_proxy=...
docker build --build-arg http_proxy=$http_proxy --build-arg https_proxy=$http_proxy --
↪build-arg no_proxy=$no_proxy -t promenade:local .
```

For convenience, there is a script which builds an image from the current code, then uses it to generate certificates and construct scripts:

```
./tools/dev-build.sh examples/basic build
```

Using Promenade Behind a Proxy

To use Promenade from behind a proxy, use the proxy settings see *Kubernetes Network*.

1.1.4 Configuration

Promenade is configured using a set of Deckhand compatible configuration documents and a bootstrapping Armada manifest that is responsible for deploying core components into the cluster.

Details about Promenade-specific documents can be found here:

Docker

Configuration for the docker daemon. This document contains a single *config* key that directly translates into the contents of the *daemon.json* file described in Docker's configuration.

Sample Document

Here is a sample document:

```
schema: promenade/Docker/v1
metadata:
  schema: metadata/Document/v1
  name: docker
  layeringDefinition:
    abstract: false
    layer: site
data:
  config:
    live-restore: true
    storage-driver: overlay2
```

EncryptionPolicy

Encryption policy defines how encryption should be applied via Promenade, either directly or via charts maintained in the Promenade project.

Encrypting script in-line data

The primary use-case for this is to encrypt `genesis.sh` or `join.sh` scripts.

```
---
schema: promenade/EncryptionPolicy/v1
metadata:
  schema: metadata/Document/v1
  name: encryption-policy
  layeringDefinition:
    abstract: false
    layer: site
  storagePolicy: cleartext
data:
  scripts:
    genesis:
      gpg: {}
...
```

Scripts

The genesis and join scripts can be built with sensitive content encrypted. Currently the only encryption method available is `gpg`, which can be enabled by setting that key to an empty dictionary.

Kubernetes apiserver persistence encryption

Kubernetes supports encrypting data it writes to etcd. This is defined by an encryption policy document enabled using a CLI option for the apiserver binary. Separating out the policy into the EncryptionPolicy document is needed as there must be guaranteed consistency between the policy put in place for bootstrapping the cluster and apiservers put in place via Helm chart.

Neither Promenade, nor the apiserver chart, do anything to ensure you do not lock yourself out of your data. When rotating encryption keys, you will need to always leave all keys that reflect data currently encrypted in the profile. Note the instructions on how to rotate keys in the linked Kubernetes documentation.

To make this encryption configuration effective, you must substitute into two other documents

- Substitute `.etcd` into `.apiserver.encryption` of your Genesis profile document.
- Substitute `.etcd` into `.values.conf.encryption_provider.content.resources` of your Armada chart definition for the apiserver chart. See the Promenade `basic` examples for reference.

```
---
schema: promenade/EncryptionPolicy/v1
metadata:
  schema: metadata/Document/v1
  name: encryption-policy
  layeringDefinition:
    abstract: false
    layer: site
  storagePolicy: cleartext
data:
  etcd:
    - resources:
        - 'secrets'
      providers:
        - secretbox:
            keys:
              - name: key1
                secret: blzKzBp6wkjU/2xzBqzgJV9FrVkkjBTT43mbctIhdPQ=
...
```

Genesis

Specific configuration for the genesis process. This document is a strict superset of the combination of *Kubernetes Node* and *HostSystem*, so only differences are discussed here.

Sample Document

Here is a complete sample document:

```
schema: promenade/Genesis/v1
metadata:
  schema: metadata/Document/v1
  name: genesis
  layeringDefinition:
    abstract: false
    layer: site
data:
  hostname: n0
  ip: 192.168.77.10
  armada:
    target_manifest: cluster-bootstrap
    metrics:
      output_dir: /var/log/armada/metrics
      max_attempts: 5
  tiller:
    listen: 24134
    probe_listen: 24135
    storage: secret
  labels:
    static:
      - calico-etcd=enabled
      - node-role.kubernetes.io/master=
    dynamic:
      - kubernetes-apiserver=enabled
      - kubernetes-controller-manager=enabled
      - kubernetes-etcd=enabled
      - kubernetes-scheduler=enabled
      - promenade-genesis=enabled
      - ucp-control-plane=enabled
  images:
    armada: quay.io/airshipit/armada:latest
    helm:
      tiller: ghcr.io/helm/tiller:v2.17.0
    kubernetes:
      apiserver: k8s.gcr.io/kube-apiserver-amd64:v1.20.5
      controller-manager: k8s.gcr.io/kube-controller-manager-amd64:v1.20.5
      etcd: quay.io/coreos/etcd:v3.4.13
      scheduler: k8s.gcr.io/kube-scheduler-amd64:v1.20.5
  files:
    - path: /var/lib/anchor/calico-etcd-bootstrap
      content: ""
      mode: 0644
```

Armada

Configuration options for bootstrapping with Armada.

keyword	type	action
target_manifest	string	Specifies the `armada/Manifest/v1` to use during Genesis.
metrics	object	See *Metrics*.

Metrics

Configuration for Armada bootstrap metric collection.

keyword	type	action
output_dir	string	(optional, default */var/log/node-exporter-textfiles*) The directory path in which to output Armada metric data.
max_attempts	integer	(optional, default 10) The maximum Armada attempts to collect metrics for. Can be set to 0 to disable metrics collection.

Tiller

Configuration options for bootstrapping with Tiller.

keyword	type	action
storage	string	(optional, not passed by default) The tiller storage arg to use. ‘
listen	integer	(optional, default *24134*) The tiller *listen* arg to use. See *Ports*.
probe_listen	integer	(optional, default *24135*) The tiller *probe_listen* arg to use. See *Ports*.

Ports

By default, promenade uses tiller ports outside of *net.ipv4.ip_local_port_range* to avoid conflicts with apiserver connections to etcd, see example.

The *listen* and *probe_listen* parameters allow setting these back to the upstream tiller defaults (or any other value) if desired.

Bootstrapping Images

Bootstrapping images are specified in the top level key `images`:

```
armada: <Armada image for bootstrapping>
helm:
  tiller: <Tiller image for bootstrapping>
kubernetes:
  apiserver: <API server image for bootstrapping>
  controller-manager: <Controller Manager image for bootstrapping>
  etcd: <etcd image for bootstrapping>
  scheduler: <Scheduler image for bootstrapping>
```

HostSystem

Sample Document to run containers in Docker runtime

```
schema: promenade/HostSystem/v1
metadata:
  schema: metadata/Document/v1
  name: host-system
```

(continues on next page)

(continued from previous page)

```
  layeringDefinition:
    abstract: false
    layer: site
data:
  files:
    - path: /opt/kubernetes/bin/kubelet
      tar_url: https://dl.k8s.io/v1.20.5/kubernetes-node-linux-amd64.tar.gz
      tar_path: kubernetes/node/bin/kubelet
      mode: 0555
  images:
    haproxy: haproxy:1.8.3
    helm:
      helm: lachlanevenson/k8s-helm:v2.14.0
    monitoring_image: busybox:1.28.3
  packages:
    repositories:
      - deb [arch=amd64] https://download.docker.com/linux/ubuntu bionic stable
    keys:
      - |-
        -----BEGIN PGP PUBLIC KEY BLOCK-----

        mQINBFit2ioBEADhWpZ8/wvZ6hUTiXOwQHXMAlaFHcPH9hAtr4F1y2+OYdbtMuth
        lqqwp028AqyY+PRfVMtSYMbjuQuu5byyKR01BbqYhuS3jtqQmljZ/bJvXqnmiVXh
        38UuLa+z077PxyxQhu5BbqntTPQMfiyqEiU+BKbq2WmANUKQf+1AmZY/IruOXbnq
        L4C1+gJ8vfmXQt99npCaxEjaNRVYfOS8QcixNzHUYnb6emjlANyEVlZzeqo7XKl7
        UrwV5inawTSzWNvtjEjj4nJL8NsLwscpLPQUhTQ+7BbQXAwAmeHCUTQIvvWXqw0N
        cmhh4HgeQscQHYgOJjjDVfoY5MucvglbIgCqfzAHW9jxmRL4qbMZj+b1XoePEtht
        ku4bIQN1X5P07fNWzlgaRL5Z4POXDDZTlIQ/El58j9kp4bnWRCJW0lya+f8ocodo
        vZZ+Doi+fy4D5ZGrL4XEcIQP/Lv5uFyf+kQtl/94VFYVJOleAv8W92KdgDkhTcTD
        G7c0tIkVEKNUq48b3aQ64NOZQW7fVjfoKwEZdOqPE72Pa45jrZzvUFxSpdiNk2tZ
        XYukHjlxxEgBdC/J3cMMNRE1F4NCA3ApfV1Y7/hTeOnmDuDYwr9/obA8t016Yljj
        q5rdkywPf4JF8mXUW5eCN1vAFHxeg9ZWemhBtQmGxXnw9M+z6hWwc6ahmwARAQAB
        tCtEb2NrZXIgUmVsZWFzZSAoQ0UgZGViKSA8ZG9ja2VyQGRvY2tlci5jb20+iQI3
        BBMBCgAhBQJYrefAAhsvBQsJCAcDBRUKCQgLBRYCAwEAAh4BAheAAAoJEI2BgDwO
        v82IsskP/iQZo68flDQmNvn8X5XTd6RRaUH33kXYXquT6NkHJciS7E2gTJmqvMqd
        tI4mNYHCSEYxI5qrcYV5YqX9P6+Ko+vozo4nseUQLPH/ATQ4qL0Zok+1jkag3Lgk
        jonyUf9bwtWxFp05HC3GMHPhhcUSexCxQLQvnFWXD2sWLKivHp2fT8QbRGeZ+d3m
        6fqcd5Fu7pxsqm0EUDK5NL+nPIgYhN+auTrhgzhK1CShfGccM/wfRlei9Utz6p9P
        XRKIlWnXtT4qNGZNTN0tR+NLG/6Bqd8OYBaFAUcue/w1VW6JQ2VGYZHnZu9S8LMc
        FYBa5Ig9PxwGQOgq6RDKDbV+PqTQT5EFMeR1mrjckk4DQJjbxeMZbiNMG5kGECA8
        g383P3elhn03WGbEEa4MNc3Z4+7c236QI3xWJfNPdUbXRaAwhy/6rTSFbzwKB0Jm
        ebwzQfwjQY6f55MiI/RqDCyuPj3r3jyVRkK86pQKBAJwFHyqj9KaKXMZjfVnowLh
        9svIGfNbGHpucATqREvUHuQbNnqkCx8VVhtYkhDb9fEP2xBu5VvHbR+3nfVhMut5
        G34Ct5RS7Jt6LIfFdtcn8CaSas/l1HbiGeRgc70X/9aYx/V/CEJv0lIe8gP6uDoW
        FPIZ7d6vH+Vro6xuWEGiuMaiznap2KhZmpkgfupyFmplh0s6knymuQINBFit2ioB
        EADneL9S9m4vhU3blaRjVUUyJ7b/qTjcSylvCH5XUE6R2k+ckEZjfAMZPLpO+/tF
        M2JIJMD4SifKuS3xck9KtZGCufGmcwiLQRzeHF7vJUKrLD5RTkNi23ydvWZgPjtx
        Q+DTT1Zcn7BrQFY6FgnRoUVIxwtdw1bMY/89rsFgS5wwuMESd3Q2RYgb7EOFOpnu
        w6da7WakWf4IhnF5nsNYGDVaIHzpiqCl+uTbf1epCjrOlIzkZ3Z3Yk5CM/TiFzPk
        z2lLz89cpD8U+NtCsfagWWfjd2U3jDapgH+7nQnCEWpROtzaKHG6lA3pXdix5zG8
        eRc6/0IbUSWvfjKxLLPfNeCS2pCL3IeEI5nothEEYdQH6szpLog79xB9dVnJyKJb
        VfxXnseoYqVrRz2VVbUI5Blwm6B40E3eGVfUQWiux54DspyVMMk41Mx7QJ3iynIa
        1N4ZAqVMAEruyXTRTxc9XW0tYhDMA/1GYvz0EmFpm8LzTHA6sFVtPm/ZlNCX6P1X
        zJwrv7DSQKD6GGlBQUX+OeEJ8tTkkf8QTJSPUdh8P8YxDFS5EOGAvhhpMBYD42kQ
        pqXjEC+XcycTvGI7impgv9PDY1RCC1zkBjKPa120rNhv/hkVk/YhuGoajoHyy4h7
        ZQopdcMtpN2dgmhEegny9JCSwxfQmQ0zK0g7m6SHiKMwjwARAQABiQQ+BBgBCAAJ
        BQJYrdoqAhsCAikJEI2BgDwOv82IwV0gBBkBCAAGBQJYrdoqAAoJEH6gqcPyc/zY
```

(continues on next page)

(continued from previous page)

```
        1WAP/2wJ+R0gE6qsce3rjaIz58PJmc8goKrir5hnElWhPgbq7cYIsW5qiFyLhkdp
        YcMmhD9mRiPpQn6Ya2w3e3B8zfIVKipbMBnke/ytZ9M7qHmDCcjoiSmwEXN3wKYI
        mD9VHONsl/CG1rU9Isw1jtB5g1YxuBA7M/m36XN6x2u+NtNMDB9P56yc4gfsZVES
        KA9v+yY2/l45L8d/WUkUi0YXomn6hyBGI7JrBLq0CX37GEYP6O9rrKipfz73XfO7
        JIGzOKZlljb/D9RX/g7nRbCn+3EtH7xnk+TK/50euEKw8SMUg147sJTcpQmv6UzZ
        cM4JgL0HbHVCojV4C/plELwMddALOFeYQzTif6sMRPf+3DSj8frbInjChC3yOLy0
        6br92KFom17EIj2CAcoeq7UPhi2oouYBwPxh5ytdehJkoo+sN7RIWua6P2WSmon5
        U888cSylXC0+ADFdgLX9K2zrDVYUG1vo8CX0vzxFBaHwN6Px26fhIT1/hYUHQR1z
        VfNDcyQmXqkOnZvvoMfz/Q0s9BhFJ/zU6AgQbIZE/hm1spsfgvtsD1frZfygXJ9f
        irP+MSAI80xHSf91qSRZOj4Pl3ZJNbq4yYxv0b1pkMqeGdjdCYhLU+LZ4wbQmpCk
        SVe2prlLureigXtmZfkqevRz7FrIZiu9ky8wnCAPwC7/zmS18rgP/17bOtL4/iIz
        QhxAAoAMWVrGyJivSkjhSGx1uCojsWfsTAm11P7jsruIL61ZzMUVE2aM3Pmj5G+W
        9AcZ58Em+1WsVnAXdUR//bMmhyr8wL/G1YO1V3JEJTRdxsSxdYa4deGBBY/Adpsw
        24jxhOJR+lsJpqIUeb999+R8euDhRHG9eFO7DRu6weatUJ6suupoDTRWtr/4yGqe
        dKxV3qQhNLSnaAzqW/1nA3iUB4k7kCaKZxhdhDbClf9P37qaRW467BLCVO/coL3y
        Vm50dwdrNtKpMBh3ZpbB1uJvgi9mXtyBOMJ3v8RZeDzFiG8HdCtg9RvIt/AIFoHR
        H3S+U79NT6i0KPzLImDfs8T7RlpyuMc4Ufs8ggyg9v3Ae6cN3eQyxcK3w0cbBwsh
        /nQNfsA6uu+9H7NhbehBMhYnpNZyrHzCmzyXkauwRAqoCbGCNykTRwsur9gS41TQ
        M8ssD1jFheOJf3hODnkKU+HKjvMROl1DK7zdmLdNzA1cvtZH/nCC9KPj1z8QC47S
        xx+dTZSx4ONAhwbS/LN3PoKtn8LPjY9NP9uDWI+TWYquS2U+KHDrBDlsgozDbs/O
        jCxcpDzNmXpWQHEtHU7649OXHP7UeNST1mCUCH5qdank0V1iejF6/CfTFU4MfcrG
        YT90qFF93M3v01BbxP+EIY2/9tiIPbrd
        =0YYh
        -----END PGP PUBLIC KEY BLOCK-----
    additional:
      - curl
      - jq
      - chrony
    required:
      runtime: docker-ce=5:19.03.8~3-0~ubuntu-bionic
      socat: socat=1.7.3.1-1
```

Sample Document to run containers in Containerd runtime

```
schema: promenade/HostSystem/v1
metadata:
  schema: metadata/Document/v1
  name: host-system
  layeringDefinition:
    abstract: false
    layer: site
data:
  files:
    - path: /opt/kubernetes/bin/kubelet
      tar_url: https://dl.k8s.io/v1.20.5/kubernetes-node-linux-amd64.tar.gz
      tar_path: kubernetes/node/bin/kubelet
      mode: 0555
  images:
    haproxy: haproxy:1.8.3
    helm:
      helm: lachlanevenson/k8s-helm:v2.14.0
    monitoring_image: busybox:1.28.3
  packages:
    additional:
```

(continues on next page)

(continued from previous page)

```
      - curl
      - jq
      - chrony
    required:
      runtime: containerd
      socat: socat=1.7.3.1-1
```

Files

A list of files to be written to the host. Files can be given as precise content, extracted from a tarball specified by url, or downloaded from a url:

```
- path: /etc/direct-content
  content: |-
    This
    exact
    text
- path: /etc/from-tar
  tar_url: http://example.com/file.tgz
  tar_source: dir/file.txt
- path: /etc/from-url
  url: http://example.com/file
```

Images

Core Images

These images are used for essential functionality:

haproxy HAProxy is configured and used for Kubernetes API discovery during bootstrapping.

kubectl Used for label application and validation tasks during bootstrapping.

Convenience Images

The `helm` image is available for convenience.

Packages

Repository Configuration

Additional APT repositories can be configured using the `repositories` and `keys` fields of the `SystemPackages` document:

repositories A list of APT source lines to be configured during genesis or join.

keys A list of public PGP keys that can be used to verify installed packages.

Package Configuration

The `required` key specifies packages that are required for all deployments, and the `additional` key allows arbitrary additional system packages to be installed. The `additional` key is particularly useful for installing packages such as *ceph-common*.

Kubelet

Configuration for the Kubernetes worker daemon (the Kubelet). This document contains three keys: `arguments`, `images`, and `config_file_overrides`. The `arguments` are appended directly to the `kubelet` command line, along with arguments that are controlled by Promenade more directly. The `config_file_overrides` are appended directly to the static kubelet configuration file and only consists of a subset of kubelet arguments. More information regarding the format for this key can be found here.

The only image that is configurable is for the `pause` container.

Sample Document

Here is a sample document:

```
schema: promenade/Kubelet/v1
metadata:
  schema: metadata/Document/v1
  name: kubelet
  layeringDefinition:
    abstract: false
    layer: site
data:
  arguments:
    - --cni-bin-dir=/opt/cni/bin
    - --cni-conf-dir=/etc/cni/net.d
    - --network-plugin=cni
    - --v=5
  images:
    pause: k8s.gcr.io/pause-amd64:3.1
  config_file_overrides:
    evictionMaxPodGracePeriod: -1
    nodeStatusUpdateFrequency: "5s"
```

Kubernetes Network

Configuration for Kubernetes networking during bootstrapping and for the `kubelet`.

Sample Document

```
schema: promenade/KubernetesNetwork/v1
metadata:
  schema: metadata/Document/v1
  name: kubernetes-network
  layeringDefinition:
    abstract: false
```

(continues on next page)

(continued from previous page)

```
    layer: site
data:
  dns:
    cluster_domain: cluster.local
    service_ip: 10.96.0.10
    bootstrap_validation_checks:
      - calico-etcd.kube-system.svc.cluster.local
      - kubernetes-etcd.kube-system.svc.cluster.local
      - kubernetes.default.svc.cluster.local
    upstream_servers:
      - 8.8.8.8
      - 8.8.4.4

  kubernetes:
    apiserver_port: 6443
    haproxy_port: 6553
    pod_cidr: 10.97.0.0/16
    service_cidr: 10.96.0.0/16
    service_ip: 10.96.0.1

  etcd:
    container_port: 2379
    haproxy_port: 2378

  hosts_entries:
    - ip: 192.168.77.1
      names:
        - registry

  ntp:
    servers:
      - 0.us.pool.ntp.org
      - 1.us.pool.ntp.org
      - 2.us.pool.ntp.org
      - 3.us.pool.ntp.org

  proxy:
    url: http://proxy.example.com:8080
    additional_no_proxy:
      - 192.168.77.1
```

DNS

The data in the `dns` key is used for bootstrapping and `kubelet` configuration of cluster and host-level DNS, which is provided by coredns.

`bootstrap_validation_checks` Domain names to resolve during the genesis and join processes for validation.

`cluster_domain` The Kubernetes cluster domain. Used by the `kubelet`.

`service_ip` The IP to use for cluster DNS. Used by the `kubelet`.

`upstream_servers` Upstream DNS servers to be configured in */etc/resolv.conf*.

Kubernetes

The `kubernetes` key contains:

`apiserver_port` The port that the Kubernetes API server process will listen on hosts where it runs.

`haproxy_port` The port that HAProxy will listen on each host. This port will be used by the `kubelet` and `kube-proxy` to find API servers in the cluster.

`pod_cidr` The CIDR from which the Kubernetes Controller Manager assigns pod IPs.

`service_cidr` The CIDR from which the Kubernetes Controller Manager assigns service IPs.

`service_ip` The in-cluster Kubernetes service IP.

NTP

The `ntp` key contains:

`servers` The list of ntp server FQDN or ip addresses used for time synchronization.

Kubernetes Node

Configuration for a basic node in the cluster.

Sample Document

Here is a sample document:

```
schema: promenade/KubernetesNode/v1
metadata:
  schema: metadata/Document/v1
  name: n1
  layeringDefinition:
    abstract: false
    layer: site
data:
  hostname: n1
  ip: 192.168.77.11
  join_ip: 192.168.77.10
  labels:
    static:
      - node-role.kubernetes.io/master=
    dynamic:
      - calico-etcd=enabled
      - kubernetes-apiserver=enabled
      - kubernetes-controller-manager=enabled
      - kubernetes-etcd=enabled
      - kubernetes-scheduler=enabled
      - ucp-control-plane=enabled
```

Host Information

Essential host-specific information is specified in this document, including the `hostname`, `ip`, and `join_ip`.

The `join_ip` is used to specify which host should be used when adding a node to the cluster.

Labels

Kubernetes labels can be specified under the `labels` key in two ways:

1. Via the `static` key, which is a list of labels to be applied immediately when the `kubelet` process starts.
2. Via the `dynamic` key, which is a list of labels to be applied after the node is marked as *Ready* by Kubernetes.

PKI Catalog

Configuration for certificate and keypair generation in the cluster. The `promenade generate-certs` command will read all `PKICatalog` documents and either find pre-existing certificates/keys, or generate new ones based on the given definition.

Sample Document

Here is a sample document:

```
schema: promenade/PKICatalog/v1
metadata:
  schema: metadata/Document/v1
  name: cluster-certificates
  layeringDefinition:
    abstract: false
    layer: site
data:
  certificate_authorities:
    kubernetes:
      description: CA for Kubernetes components
      certificates:
        - document_name: apiserver
          description: Service certificate for Kubernetes apiserver
          common_name: apiserver
          hosts:
            - localhost
            - 127.0.0.1
            - 10.96.0.1
          kubernetes_service_names:
            - kubernetes.default.svc.cluster.local
        - document_name: kubelet-genesis
          common_name: system:node:n0
          hosts:
            - n0
            - 192.168.77.10
          groups:
            - system:nodes
        - document_name: kubelet-n0
          common_name: system:node:n0
          hosts:
            - n0
            - 192.168.77.10
          groups:
            - system:nodes
```

(continues on next page)

(continued from previous page)

```
      - document_name: kubelet-n1
        common_name: system:node:n1
        hosts:
          - n1
          - 192.168.77.11
        groups:
          - system:nodes
      - document_name: kubelet-n2
        common_name: system:node:n2
        hosts:
          - n2
          - 192.168.77.12
        groups:
          - system:nodes
      - document_name: kubelet-n3
        common_name: system:node:n3
        hosts:
          - n3
          - 192.168.77.13
        groups:
          - system:nodes
      - document_name: scheduler
        description: Service certificate for Kubernetes scheduler
        common_name: system:kube-scheduler
      - document_name: controller-manager
        description: certificate for controller-manager
        common_name: system:kube-controller-manager
      - document_name: admin
        common_name: admin
        groups:
          - system:masters
      - document_name: armada
        common_name: armada
        groups:
          - system:masters
  kubernetes-etcd:
    description: Certificates for Kubernetes's etcd servers
    certificates:
      - document_name: apiserver-etcd
        description: etcd client certificate for use by Kubernetes apiserver
        common_name: apiserver
      - document_name: kubernetes-etcd-anchor
        description: anchor
        common_name: anchor
      - document_name: kubernetes-etcd-genesis
        common_name: kubernetes-etcd-genesis
        hosts:
          - n0
          - 192.168.77.10
          - 127.0.0.1
          - localhost
          - kubernetes-etcd.kube-system.svc.cluster.local
      - document_name: kubernetes-etcd-n0
        common_name: kubernetes-etcd-n0
        hosts:
          - n0
          - 192.168.77.10
```

(continues on next page)

(continued from previous page)

```
          - 127.0.0.1
          - localhost
          - kubernetes-etcd.kube-system.svc.cluster.local
      - document_name: kubernetes-etcd-n1
        common_name: kubernetes-etcd-n1
        hosts:
          - n1
          - 192.168.77.11
          - 127.0.0.1
          - localhost
          - kubernetes-etcd.kube-system.svc.cluster.local
      - document_name: kubernetes-etcd-n2
        common_name: kubernetes-etcd-n2
        hosts:
          - n2
          - 192.168.77.12
          - 127.0.0.1
          - localhost
          - kubernetes-etcd.kube-system.svc.cluster.local
      - document_name: kubernetes-etcd-n3
        common_name: kubernetes-etcd-n3
        hosts:
          - n3
          - 192.168.77.13
          - 127.0.0.1
          - localhost
          - kubernetes-etcd.kube-system.svc.cluster.local
  kubernetes-etcd-peer:
    certificates:
      - document_name: kubernetes-etcd-genesis-peer
        common_name: kubernetes-etcd-genesis-peer
        hosts:
          - n0
          - 192.168.77.10
          - 127.0.0.1
          - localhost
          - kubernetes-etcd.kube-system.svc.cluster.local
      - document_name: kubernetes-etcd-n0-peer
        common_name: kubernetes-etcd-n0-peer
        hosts:
          - n0
          - 192.168.77.10
          - 127.0.0.1
          - localhost
          - kubernetes-etcd.kube-system.svc.cluster.local
      - document_name: kubernetes-etcd-n1-peer
        common_name: kubernetes-etcd-n1-peer
        hosts:
          - n1
          - 192.168.77.11
          - 127.0.0.1
          - localhost
          - kubernetes-etcd.kube-system.svc.cluster.local
      - document_name: kubernetes-etcd-n2-peer
        common_name: kubernetes-etcd-n2-peer
        hosts:
          - n2
```

(continues on next page)

(continued from previous page)

```
            - 192.168.77.12
            - 127.0.0.1
            - localhost
            - kubernetes-etcd.kube-system.svc.cluster.local
        - document_name: kubernetes-etcd-n3-peer
          common_name: kubernetes-etcd-n3-peer
          hosts:
            - n3
            - 192.168.77.13
            - 127.0.0.1
            - localhost
            - kubernetes-etcd.kube-system.svc.cluster.local
    calico-etcd:
      description: Certificates for Calico etcd client traffic
      certificates:
        - document_name: calico-etcd-anchor
          description: anchor
          common_name: anchor
        - document_name: calico-etcd-n0
          common_name: calico-etcd-n0
          hosts:
            - n0
            - 192.168.77.10
            - 127.0.0.1
            - localhost
            - 10.96.232.136
        - document_name: calico-etcd-n1
          common_name: calico-etcd-n1
          hosts:
            - n1
            - 192.168.77.11
            - 127.0.0.1
            - localhost
            - 10.96.232.136
        - document_name: calico-etcd-n2
          common_name: calico-etcd-n2
          hosts:
            - n2
            - 192.168.77.12
            - 127.0.0.1
            - localhost
            - 10.96.232.136
        - document_name: calico-etcd-n3
          common_name: calico-etcd-n3
          hosts:
            - n3
            - 192.168.77.13
            - 127.0.0.1
            - localhost
            - 10.96.232.136
        - document_name: calico-node
          common_name: calcico-node
    calico-etcd-peer:
      description: Certificates for Calico etcd clients
      certificates:
        - document_name: calico-etcd-n0-peer
          common_name: calico-etcd-n0-peer
```

(continues on next page)

(continued from previous page)

```
          hosts:
            - n0
            - 192.168.77.10
            - 127.0.0.1
            - localhost
            - 10.96.232.136
        - document_name: calico-etcd-n1-peer
          common_name: calico-etcd-n1-peer
          hosts:
            - n1
            - 192.168.77.11
            - 127.0.0.1
            - localhost
            - 10.96.232.136
        - document_name: calico-etcd-n2-peer
          common_name: calico-etcd-n2-peer
          hosts:
            - n2
            - 192.168.77.12
            - 127.0.0.1
            - localhost
            - 10.96.232.136
        - document_name: calico-etcd-n3-peer
          common_name: calico-etcd-n3-peer
          hosts:
            - n3
            - 192.168.77.13
            - 127.0.0.1
            - localhost
            - 10.96.232.136
        - document_name: calico-node-peer
          common_name: calcico-node-peer
keypairs:
  - name: service-account
    description: Service account signing key for use by Kubernetes controller-manager.
```

Certificate Authorities

The data in the `certificate-authorities` key is used to generate certificates for each authority and node.

Each certificate authority requires essential host-specific information for each node, including the `hostname` and `ip` as listed in each *Kubernetes Node* document.

The provided Armada manifest and will be applied on the genesis node as soon as it is healthy.

1.1.5 Troubleshooting

Genesis Troubleshooting

genesis.sh

Kubernetes services failures

Before the Armada manifests are applied, the genesis.sh script will bring basic kubernetes services online by starting docker containers for these services.

One of the first services to be brought up is the kubernetes API. If it fails to come up, you may see a repeated error as follows from the genesis.sh script:

```
.The connection to the server apiserver.kubernetes.promenade:6443 was
refused - did you specify the right host or port?
```

Check that the hostname in your Genesis.yaml matches the hostname of the machine you are trying to install onto. If they do not match, change one to match the other. If you change Genesis.yaml, then re-generate the Promenade payloads.

If the hostnames match, check the container logs under /var/log/pods to see the reason for the provisioning failure. (`kubectl logs` function will not be available if the API container is not running).

Armada failures

When executing genesis.sh, you may encounter failures from Armada in the provisioning of other containers. For example:

```
CRITICAL armada [-] Unhandled error: armada.exceptions.tiller_exceptions.
↪ReleaseException: Failed to Install release: barbican
```

Use `kubectl logs` on the failed pod to determine the reason for the failure. E.g.:

```
sudo kubectl logs barbican-api-5b8bccdf8f-x7sld --namespace=ucp
```

Other errors may point to configuration errors. For example:

```
CRITICAL armada [-] Unhandled error: armada.exceptions.source_exceptions.
↪GitLocationException: master is not a valid git repository.
```

In this case, the git branch name was inadvertently substituted for the git URL in one of the chart definitions in `bootstrap-armada.yaml`.

Post-run failures

At its conclusion, the genesis script will output the list of containers provisioned and their status, as reported by kubernetes. It is possible that some containers may not be in a Running state. E.g.:

```
ucp   promenade-api-6696769cd-qwpzf   0/1   ImagePullBackOff   0   10h
```

For general failures, `kubectl logs` may be used as in the previous section. In this case, it was necessary to run `kubectl describe` on the pod to get the details of the image pull failure. E.g.:

```
kubectl describe pod promenade-api-7dc54d47c-qw27m --namespace=ucp
```

In this particular incident report, the problem was a missing certificate on the bare metal node which caused the image download to fail. Installing the certificate, restarting the docker service, and then waiting for the container to retry resolved this particular issue.

1.1.6 Promenade API Documentation

/v1.0/health

Allows other components to validate Promenade's health status.

GET /v1.0/health

Returns the health status.

Responses:

- 204 No Content

/v1.0/join-scripts

Generates join scripts and for Drydock.

GET /v1.0/join-scripts

Generates script to be consumed by Drydock.

Query parameters

hostname Name of the node

ip IP address of the node

design_ref Endpoint containing configuration documents

dynamic.labels Used to set configuration options in the generated script

static.labels Used to set configuration options in the generated script

Responses:

- 200 OK: Script returned as response body
- 400 Bad Request: One or more query parameters is missing or misspelled

/v1.0/validatedesign

Performs validations against specified documents.

POST /v1.0/validatedesign

Performs validation against specified documents.

Message Body

href Location of the document to be validated

Responses:

- 200 OK: Documents were successfully validated
- 400 Bad Request: Documents were not successfully validated

/v1.0/node-labels/<node_name>

Update node labels

PUT /v1.0/node-labels/<node_name>

Updates node labels eg: adding new labels, overriding existing labels and deleting labels from a node.

Message Body:

dict of labels

```
{"label-a": "value1", "label-b": "value2", "label-c": "value3"}
```

Responses:

- 200 OK: Labels successfully updated
- 400 Bad Request: Bad input format
- 401 Unauthorized: Unauthenticated access
- 403 Forbidden: Unauthorized access
- 404 Not Found: Bad URL or Node not found
- 500 Internal Server Error: Server error encountered
- 502 Bad Gateway: Kubernetes Config Error
- 503 Service Unavailable: Failed to interact with Kubernetes API

1.1.7 Promenade Exceptions