Pegleg Documentation

Release 0.1.0

Pegleg Authors

Aug 10, 2021

CONTENTS

CHAPTER

ONE

OVERVIEW

1.1 Getting Started

1.1.1 What is Pegleg?

Pegleg is a document aggregator that aggregates all the documents in a repository and packs them into a single YAML file. This allows for operators to structure their site definitions in a maintainable directory layout, while providing them with the automation and tooling needed to aggregate, lint, and render those documents for deployment.

For more information on the documents that Pegleg works on see Document Fundamentals.

1.1.2 Basic Usage

Before using Pegleg, you must:

1. Clone the Pegleg repository:

```
git clone https://git.airshipit.org/airship-pegleg
```

2. Install Docker CE, as Pegleg relies on it for CLI execution.

3. Clone the repos containing your site definition libraries into the local filesystem where Pegleg is running, as Pegleg can only work with files available in the local directory.

You will then be able to use all of Pegleg's features through the CLI. See CLI for more information.

CHAPTER

TWO

DESIGN

2.1 Definition Artifact Layout

The definition artifacts are stored in the below directory structure. This structure is used only to assist humans in maintaining the data. When the documents are consumed by the Airship services, they are viewed as a flat set of all documents.:

```
deployment_files/deployment_files
 |- /global
 |    |- /common
 |    |    |- {definition library}
 |    |- /v1.0
 |         |- {definition library}
 |- /type
 |    |- /production
 |    |    |- /v1.0
 |    |         |- {definition library}
 |    |- /cicd
 |    |    |- /v1.0
 |    |         |- {definition library}
 |    |- /labs
 |         |- /v1.0
 |              |- {definition library}
 |- /site
      |- /{sitename}
           |- site_definition.yaml
           |- {definition library}
```

The root-level listings of `global`, `type` and `site` are the layers as listed in the Deckhand LayeringPolicy document. The process of choosing the definition libraries to compose the actual design for a site is described below.

2.1.1 site_definition.yaml

The site_definition.yaml file is what selects the definition libraries to use for a site. Additional metadata can be added to this file as needed to meet requirements.:

```
---
schema: pegleg/SiteDefinition/v1
metadata:
  layeringDefinition:
    abstract: false
    layer: 'site'
```

(continues on next page)

(continued from previous page)

```
  name: 'mtn13b.1'
  schema: metadata/Document/v1
  storagePolicy: cleartext
data:
  site_type: 'cicd'
  repositories:  # Optional field.
    global:
      revision: 47676764d3935e4934624bf9593e9115984fe668
      url: ssh://REPO_USERNAME@<git_url>:29418/global-manifests.git
    secrets:
      revision: master
      url: ssh://REPO_USERNAME@<git_url>:29418/security-manifests.git
```

The `repositories` field (optional) maps default authentication information for each of the manifests repositories supported, for example:

- global
- secrets
- site

Each of the above fields must have 2 pieces of information:

- `revision` (required) - specifies a valid *Git Reference*.
- `url` (required) - specifies the repository remote path. Consists of the following required segments:
  - protocol - http, https, or ssh.
  - REPO_USERNAME - must be included for ssh only. Can be overridden with the CLI via *Check PKI Certs*.
  - port - e.g. 29418 - must be included for ssh only.
  - repository name

Self-Contained Repository

Note that if the `repositories` field is omitted, then this implies that the repository contains **all** the manifests required for site deployment. One such example is Airship in a Bottle which hosts all the manifests required for deploying a minimum OpenStack development environment.

Please see the *related CLI documentation* for information on how to issue relevant commands.

2.2 Definition Library Layout

The definition library layout is replicated in each location that the site definition contains a set of documents.:

```
  {library root}
    |- /schemas
    |     |- /{namespace}
    |         |- /{kind}
    |             |- {version}.yaml
    |
    |- /profiles
    |     |- /hardware
```

(continues on next page)

(continued from previous page)

```
    |    |- /host
    |
    |- /pki
    |    |- kubernetes-nodes.yaml
    |
    |- /secrets
    |    |- /certifcate-authorities
    |    |- /certificates
    |    |- /keypairs
    |    |- /passphrases
    |
    |- /software
    |    |- /charts
    |    |    |- /{chart collection}
    |    |    |    |- dependencies.yaml
    |    |    |    |- /{chartgroup}
    |    |    |         |- chart-group.yaml
    |    |    |         |- {chart1}.yaml
    |    |    |         |- {chart2}.yaml
    |    |    |
    |    |    |- /{chart collection}
    |    |         |- dependencies.yaml
    |    |         |- /{chartgroup}
    |    |              |- chart-group.yaml
    |    |              |- {chart1}.yaml
    |    |              |- {chart2}.yaml
    |    |
    |    |- /config
    |    |    |- Docker.yaml
    |    |    |- Kubelet.yaml
    |    |    |- versions.yaml
    |    |
    |    |- /manifests
    |         |- bootstrap.yaml
    |         |- site.yaml
    |
    |- /networks
    |    |- /physical
    |    |    |- sitewide.yaml
    |    |    |- rack1.yaml
    |    |
    |    |- KubernetesNetwork.yaml
    |    |- common-addresses.yaml
    |
    |- /baremetal
         |- rack1.yaml
         |- rack2.yaml

* Schemas - The schemas should all be sourced from the Airship
  service repositories. Care should be taken that the schemas
  included in the site definition are taken from the version of
  the service being deployed in the site.
* Software
  * /config/versions.yaml will contain a manifest of all the
    chart, image and package versions. These should be substituted
    into all other documents that define version information.
  * dependencies.yaml - Contains Armada chart definitions that are
```

(continues on next page)

(continued from previous page)

```
    only utilized as dependencies for other charts (e.g. helm-toolkit)
  * Chart collection - Loose organization of chart groups
    such as 'kubernetes', 'ucp', 'osh'
* Physical networks and baremetal nodes can be split into files
  in whatever way makes sense. The best practice here to define
  them by racks is only a suggestion.
```

2.3 Document Fundamentals

The definition of a site consists of a set of small YAML documents that are managed by Deckhand. Each document is identified by a `schema` top-level key and the `metadata.name` value that uniquely identifies a particular document of the type `schema`. Deckhand provides functionality allowing documents to be authored such that data from multiple documents can be merged.

- Abstact vs Concrete - Documents define a value in `metadata.layeringDefinition.abstract` to determine if a document is abstract (a value of `true`) or concrete (a value of `false`). When calling the `/revisions/{id}/rendered-documents` API, only concrete documents are returned.
- Layering - Document layering is used for whole documents that have known defaults but may need to be transformed in specific instances.
- Substitution - Data substitution is used for extracting particular values from a document's data section (whole or in-part) and inserting that data into a destination document (at the root of the data section or deeper into a document).

2.4 Shared Documents

2.4.1 Secrets

Several generic document types exist to support encryption of sensitive data.

These must be utilized for all data considered sensitive.

2.4.2 Global Catalogue Documents

Deckhand's layering functionality can be utilized in several ways, i.e site definitions. At the `global` layer there will be several documents providing different configurations for an object or service. Each of these will be abstract documents. They can be incorporated into a particular site definition by creating a concrete child document in the `site` layer that selects the correct `global` parent. The child can then do further customization on the configuration if needed.

As a best practice, `global` level documents using the catalog pattern should utilize the layering labels `component` and `configuration` to provide a consistent method for children documents select the correct parent. The below example shows a set of documents for two configuration options for OpenStack Keystone: one using local SQL-backed identity stores and one using an LDAP backend. A site definition can then select and customize the appropriate option.

When using a catalogue document, it is important to review that document to ensure you understand all the requirements for it.

- Abstract documents are not required to be fully formed, so selecting a catalogue document may require the child document to add data so the document passes validation. In the below example, the child document adds several required fields to the catalogue Chart: `chart_name`, `release`, and `namespace`.

- A catalogue document may define substitutions with the expectation that the substitution source documents are defined at a lower layer. In the example below, all of the required credentials in the chart are defined as substitutions in the `global` catalogue document, but the source documents for the substitutions are defined in the `site` layer.

This catalogue pattern can also be utilized for the `type` layer if needed.

2.4.3 Global Layer

```
---
schema: armada/Chart/v1
metadata:
  schema: metadata/Document/v1
  name: ldap-backed-keystone
  labels:
    component: keystone
    configuration: ldap-backed
  layeringDefinition:
    abstract: true
    layer: global
  storagePolicy: cleartext
  substitutions:
    - src:
        schema: deckhand/Passphrase/v1
        name: keystone_admin_password
        path: .
      dest:
        path: .values.endpoints.identity.auth.admin.password
    - src:
        schema: deckhand/Passphrase/v1
        name: mariadb_admin_password
        path: .
      dest:
        path: .values.endpoints.oslo_db.auth.admin.password
    - src:
        schema: deckhand/Passphrase/v1
        name: mariadb_keystone_password
        path: .
      dest:
        path: .values.endpoints.oslo_db.auth.user.password
    - src:
        schema: pegleg/SoftwareVersions/v1
        name: software-versions
        path: .charts.ucp.keystone
      dest:
        path: .source
    - src:
        schema: pegleg/StringValue/v1
        name: ldap_userid
        src: .
      dest:
        path: .values.conf.ks_domains.cicd.identity.ldap.user
        pattern: '(^USERID)'
    - src:
        schema: deckhand/Passphrase/v1
        name: ldap_userid_password
```

(continues on next page)

(continued from previous page)

```
        path: .
      dest:
        path: .values.conf.ks_domain.cicd.identity.ldap.password
data:
  install:
    no_hooks: false
  upgrade:
    no_hooks: false
  pre:
    delete:
      - type: job
        labels:
          job-name: keystone-db-sync
      - type: job
        labels:
          job-name: keystone-db-init
  post:
    delete: []
    create: []
  values:
    conf:
      keystone:
        identity:
          driver: sql
          default_domain_id: default
          domain_specific_drivers_enabled: True
          domain_configurations_from_database: True
          domain_config_dir: /etc/keystonedomains
      ks_domains:
        cicd:
          identity:
            driver: ldap
            ldap:
              url: "ldap://your-ldap-server.example.com"
              user: "USERID@example.com"
              password: USERID_PASSWORD_REPLACEME
              suffix: "dc=example,dc=com"
              query_scope: sub
              page_size: 1000
              user_tree_dn: "DC=example,DC=com"
              user_objectclass: user
              user_name_attribute: sAMAccountName
              user_mail_attribute: mail
              user_enabled_attribute: userAccountControl
              user_enabled_mask: 2
              user_enabled_default: 512
              user_attribute_ignore: "default_project_id,tenants,projects,password"
    replicas: 2
    labels:
      node_selector_key: ucp-control-plane
      node_selector_value: enabled
...
---
schema: armada/Chart/v1
metadata:
  schema: metadata/Document/v1
  name: sql-backed-keystone
```

(continues on next page)

(continued from previous page)

```
  labels:
    component: keystone
    configuration: sql-backed
  layeringDefinition:
    abstract: true
    layer: global
  substitutions:
    - src:
        schema: deckhand/Passphrase/v1
        name: keystone_admin_password
        path: .
      dest:
        path: .values.endpoints.identity.auth.admin.password
    - src:
        schema: deckhand/Passphrase/v1
        name: mariadb_admin_password
        path: .
      dest:
        path: .values.endpoints.oslo_db.auth.admin.password
    - src:
        schema: deckhand/Passphrase/v1
        name: mariadb_keystone_password
        path: .
      dest:
        path: .values.endpoints.oslo_db.auth.user.password
    - src:
        schema: pegleg/SoftwareVersions/v1
        name: software-versions
        path: .charts.ucp.keystone
      dest:
        path: .source
data:
  timeout: 300
  install:
    no_hooks: false
  upgrade:
    no_hooks: false
    pre:
      delete:
        - name: keystone-bootstrap
          type: job
          labels:
            application: keystone
            component: bootstrap
        - name: keystone-credential-setup
          type: job
          labels:
            application: keystone
            component: credential-setup
        - name: keystone-db-init
          type: job
          labels:
            application: keystone
            component: db-init
        - name: keystone-db-sync
          type: job
          labels:
```

(continues on next page)

(continued from previous page)

```
            application: keystone
            component: db-sync
        - name: keystone-fernet-setup
          type: job
          labels:
            application: keystone
            component: fernet-setup
  values: {}
  source: {}
...
```

2.4.4 Site Layer

```
---
schema: armada/Chart/v1
metadata:
  schema: metadata/Document/v1
  name: ucp-helm-toolkit
  layeringDefinition:
    abstract: false
    layer: site
  substitutions:
    - src:
        schema: pegleg/SoftwareVersions/v1
        name: software-versions
        path: .charts.ucp.helm-toolkit
      dest:
        path: .source
data:
  chart_name: ucp-helm-toolkit
  release: ucp-helm-toolkit
  namespace: ucp
  timeout: 100
  values: {}
  source: {}
  dependencies: []
...
---
schema: armada/Chart/v1
metadata:
  schema: metadata/Document/v1
  name: ucp-keystone
  layeringDefinition:
    abstract: false
    layer: site
    parentSelector:
      component: keystone
      configuration: ldap-backed
    actions:
      - method: merge
        path: .
data:
  chart_name: ucp-keystone
  release: ucp-keystone
  namespace: ucp
```

(continues on next page)

(continued from previous page)

```
  dependencies:
    - ucp-helm-toolkit
...
---
schema: deckhand/Passphrase/v1
metadata:
  schema: metadata/Document/v1
  name: ldap_userid_password
  storagePolicy: encrypted
data: a-secret-password
...
---
schema: deckhand/Passphrase/v1
metadata:
  schema: metadata/Document/v1
  name: keystone_admin_password
  storagePolicy: encrypted
data: a-secret-password
...
---
schema: deckhand/Passphrase/v1
metadata:
  schema: metadata/Document/v1
  name: mariadb_admin_password
  storagePolicy: encrypted
data: a-secret-password
...
---
schema: deckhand/Passphrase/v1
metadata:
  schema: metadata/Document/v1
  name: mariadb_keystone_password
  storagePolicy: encrypted
data: a-secret-password
...
---
schema: pegleg/StringValue/v1
metadata:
  schema: metadata/Document/v1
  name: keystone_ldap_userid
  storagePolicy: cleartext
data: myuser
...
```

2.5 Secrets Catalog

2.5.1 Artifacts

Pegleg Document Types

Overview

Pegleg is not only the custodian of deployment manifests that handles responsibilities such as aggregation and linting, but is also the author of certain Deckhand-formatted manifests. These manifests are generated via `Catalog` classes.

Documents

Pegleg generates or ingests each of the documents below, each identified by its schema.

`pegleg/PeglegManagedDocument/v1`

Pegleg both generates and ingests this type of document. A `PeglegManagedDocument` can have one or both of the following data elements:

- `generated`
- `encrypted`

A `PeglegManagedDocument` serves as a wrapper around other documents, and the wrapping serves to capture additional metadata that is necessary, but separate from the managed document proper.

The managed document data lives in the `data.managedDocument` portion of a `PeglegManagedDocument`.

Generated

If a `PeglegManagedDocument` is `generated`, then its contents have been created by Pegleg, and it thus includes provenance information per this example:

```
schema: pegleg/PeglegManagedDocument/v1
metadata:
  name: matches-document-name
  schema: deckhand/Document/v1
  labels:
    matching: wrapped-doc
  layeringDefinition:
    abstract: true
    # Pegleg will initially support generation at site level only
    layer: site
  storagePolicy: encrypted
data:
  generated:
    at: <timestamp>
    by: <author>
    specifiedBy:
      repo: <...>
      reference: <git ref-head or similar>
      path: <PKICatalog/PassphraseCatalog details>
  managedDocument:
    schema: <as appropriate for wrapped document>
    metadata:
      storagePolicy: encrypted
      schema: <as appropriate for wrapped document>
      <metadata from parent PeglegManagedDocument>
      <any other metadata as appropriate>
    data: <generated data>
```

Encrypted

If a `PeglegManagedDocument` is `encrypted`, then its contents have been encrypted by Pegleg, and it thus includes provenance information per this example:

```
schema: pegleg/PeglegManagedDocument/v1
metadata:
  name: matches-document-name
  schema: deckhand/Document/v1
  labels:
    matching: wrapped-doc
  layeringDefinition:
    abstract: false
    layer: matching-wrapped-doc
  storagePolicy: encrypted
data:
  encrypted:
    at: <timestamp>
    by: <author>
  managedDocument:
    schema: <as appropriate for wrapped document>
    metadata:
      storagePolicy: encrypted
      schema: <as appropriate for wrapped document>
      <metadata from parent PeglegManagedDocument>
      <any other metadata as appropriate>
    data: <encrypted string blob>
```

Note that this `encrypted` has a different purpose than the Deckhand `storagePolicy: encrypted` metadata, which indicates an *intent* for Deckhand to store a document encrypted at rest in the cluster. The two can be used together to ensure security. If a document is marked as `storagePolicy: encrypted`, then automation may validate that it is only persisted (e.g. to a Git repository) if it is in fact encrypted within a `PeglegManagedDocument`.

Generated & Encrypted

A `PeglegManagedDocument` that is both generated via a `Catalog`, and encrypted (as specified by the `Catalog`) will contain both `generated` and `encrypted` stanzas.

Supported Managed Documents

Supported managed document schemas include one of the following Deckhand schemas:

- Certificates:
  - `deckhand/Certificate/v1`
  - `deckhand/CertificateKey/v1`
- Certificate Authorities:
  - `deckhand/CertificateAuthority/v1`
  - `deckhand/CertificateAuthorityKey/v1`
- Keypairs:
  - `deckhand/PrivateKey/v1`

- deckhand/PublicKey/v1

2.5.2 Public Key Infrastructure (PKI)

Public Key Infrastructure (PKI) Catalog

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

Dependencies

Pegleg's PKI Catalog depends on CloudFlare's PKI/TLS toolkit, which is installed as a part of Pegleg's Dockerfile.

Sample Document

Here is a sample document:

```yaml
# Basic example of pki-catalog.yaml for k8s.
---
schema: promenade/PKICatalog/v1
metadata:
  schema: metadata/Document/v1
  name: cluster-certificates-addition
  layeringDefinition:
    abstract: false
    layer: site
  storagePolicy: cleartext
data:
  certificate_authorities:
    kubernetes:
      description: CA for Kubernetes components
      certificates:
        - document_name: kubelet-n3
          common_name: system:node:n3
          hosts:
            - n3
            - 192.168.77.13
          groups:
            - system:nodes
...
```

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.

CHAPTER

THREE

DEVELOPER'S GUIDE

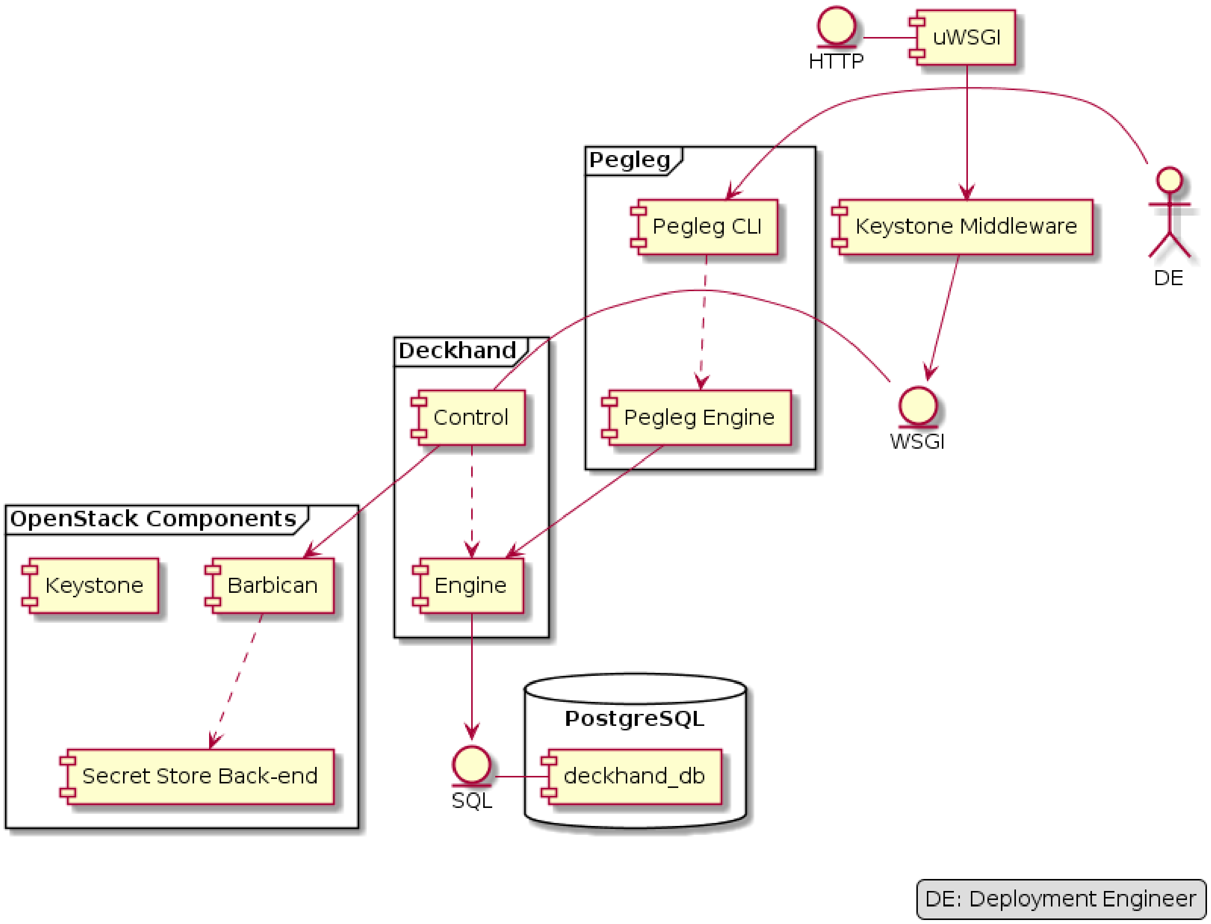
3.1 Developer Overview of Pegleg

Pegleg's core mission is to, alongside Deckhand, facilitate document authoring strategies within Airship, by:

- aggregating documents across multiple revisioned repositories, each of which contains multiple documents defining sites' software and hardware stacks
- providing early linting of documents prior to their collection and eventual deployment
- including utility functions enabling operators and developers alike to list available sites, render individual manifests via Deckhand, bootstrap repositories with Pegleg-compliant directory layouts, to name a few

3.1.1 Architecture

Pegleg, as a CLI, has a rather simplistic architecture. It is meaningful to visualize Pegleg alongside Deckhand:



3.1.2 Components

cli

The Pegleg `cli` module implements the user-facing CLI. For more information about this module, reference the *CLI documentation*.

engine

The `engine` module implements the following functionality:

- document linting
- document rendering via Deckhand
- document aggregation
- additional document utility functions

3.1.3 Developer Workflow

Because Airship is a container-centric platform, the developer workflow heavily utilizes containers for testing and publishing. It also requires Pegleg to produce multiple artifacts that are related, but separate: the Python package, the Docker image and the Helm chart. The code is published via the Docker image artifact.

Pegleg strives to conform to the Airship coding conventions.

Python

The Pegleg code base lives under `pegleg`. Pegleg supports py36 interpreter.

Docker

Pegleg Dockerfiles for supported distributions are located in `/images/pegleg` along with any artifacts built specifically to enable the container image. Make targets are used for generating and testing the artifacts.

- `make images` - Build the Pegleg Docker image.

Pegleg, as a containerized CLI, uses Docker via `tools/pegleg.sh` to execute CLI commands. Commands can also be executed using the `Makefile` target: `run_pegleg`.

Virtual Environment

Rather than, after each local code change, rebuilding the Pegleg image and overriding the `IMAGE` environment variable so `tools/pegleg.sh` uses the latest code changes, it is possible to use a virtual environment for much faster development.

This can achieved by issuing the following commands (from the root Pegleg directory):

```
# Quick way of building a virtualenv and installing all required
# dependencies into it.
tox -e py36 --notest
source .tox/py36/bin/activate
pip install -e .

# Now is it possible to run the Pegleg CLI to test local changes:
pegleg <command> <options>

# Or run unit tests:
pytest -k <regex>
```

Note that after setting up the virtual environment, one only needs to source it in order to re-run unit tests or Pegleg CLI commands, to test local changes.

3.1.4 Testing

All Pegleg tests are nested under `tests`.

Pegleg comes equipped with a number of tox targets for running unit tests, as well as pep8 and Bandit checks.

Unit Tests

To run all unit tests, execute:

```
$ tox -e py36
```

To run unit tests using a regex, execute:

```
$ tox -e py36 -- <regex>
```

CHAPTER

FOUR

OPERATOR’S GUIDE

4.1 Pegleg CLI

The Pegleg CLI is used in conjunction with the script located in pegleg/tools called `pegleg.sh`.

Note: The default workspace for the `pegleg.sh` script is `/workspace`. The examples below require that this workspace be used.

Note: Pegleg collect and render commands generate a deployment-version document containing information gathered from the site-definition, which includes the specific commit for each repository used and whether that repository was clean or dirty.

4.1.1 Environment Variables

```
$WORKSPACE = Location of the folder that holds the repositories containing
the site definition libraries. Pegleg makes no assumptions about the root
directory. $WORKSPACE is /workspace in the container context.

    Example: $WORKSPACE=/home/ubuntu/all_repos

$IMAGE = Location of pegleg docker image.

    Example: $IMAGE=quay.io/airshipit/pegleg:latest-ubuntu_xenial
```

4.1.2 Usage

To run:

```
export WORKSPACE=<repo_location>
export IMAGE=<docker_image>
./pegleg.sh <command> <options>
```

For example:

```
cd /opt/airship-pegleg
export WORKSPACE=/opt/airship/treasuremap
./tools/pegleg.sh site -r /workspace --help
```

Note: If `sudo` permissions are required to execute `pegleg.sh`, then it is necessary to use the `-E` flag with `sudo` in order for the current environment to be used. For example:

```
cd /opt/airship-pegleg
export WORKSPACE=/opt/airship/treasuremap
sudo -E ./tools/pegleg.sh site -r /workspace --help
```

4.1.3 CLI Options

-v / \-\-verbose (Optional, Default=False).

Enable debug logging.

-l / \-\-logging-level (Optional, Default=40).

Specifies the logging level, as a number, with which to run pegleg. The available levels are as follows:

- 10 (DEBUG)
- 20 (INFO)
- 30 (WARNING)
- 40 (ERROR)
- 50 (CRITICAL)

The *-v* option will override any logging level specified in favor of DEBUG.

4.1.4 Repo Group

Allows you to perform repository-level operations.

Options

-r / \-\-site-repository (Required).

Path to the root of the site repository (containing site_definition.yaml) repo.

For example: /opt/airship/treasuremap

The revision can also be specified via (for example):

```
-r /opt/airship/treasuremap@revision
```

-p / \-\-clone-path (Optional, Default=/tmp/).

The path where the repo will be cloned. By default the repo will be cloned to the /tmp path. If this option is included and the repo already exists, then the repo will not be cloned again and the user must specify a new clone path or pass in the local copy of the repository as the site repository. Suppose the repo name is `treasuremap` and the clone path is `/tmp/mypath` then the following directory is created `/tmp/mypath/treasuremap` which will contain the contents of the repo. Example of using clone path:

```
-p /tmp/mypath
```

Lint

Sanity checks for repository content (all sites in the repository). To lint a specific site, see *site-level linting*.

See *Linting* for more information.

4.1.5 Site Group

Allows you to perform site-level operations.

```
./pegleg.sh site -r <site_repo> -e <extra_repo> <command> <options>
```

Options

-r / \-\-site-repository (Required).

Path to the root of the site repository (containing site_definition.yaml) repo.

For example: /opt/airship/treasuremap

The revision can also be specified via (for example):

```
-r /opt/airship/treasuremap@revision
```

-e / \-\-extra-repository (Optional).

Path to the root of extra repositories used for overriding those specified under the `repositories` field in a given `site-definition.yaml`.

These should be named per the site-definition file, e.g.:

```
-e global=/opt/global -e secrets=/opt/secrets
```

-p / \-\-clone-path (Optional, Default=/tmp/).

The path where the repo will be cloned. By default the repo will be cloned to the /tmp path. If this option is included and the repo already exists, then the repo will not be cloned again and the user must specify a new clone path or pass in the local copy of the repository as the site repository. Suppose the repo name is `treasuremap` and the clone path is `/tmp/mypath` then the following directory is created `/tmp/mypath/treasuremap` which will contain the contents of the repo. Example of using clone path:

```
-p /tmp/mypath
```

Repository Overrides

By default, the revision specified in the `site-definition.yaml` for the site will be leveraged but can be *overridden* using:

```
-e global=/opt/global@revision
```

-k / \-\-repo-key (Optional, SSH only).

The SSH public key to use when cloning remote authenticated repositories.

Required for cloning repositories via SSH protocol.

-u / \-\-repo-username (Optional, unless required by repo URL).

The SSH username to use when cloning remote authenticated repositories specified in the site-definition file. Any occurrences of `REPO_USERNAME` in an entry under the `repositories` field in a given `site-definition.yaml` will be replaced with this value.

Required for cloning repositories via SSH protocol. This argument will generate an exception if no repo URL uses `REPO_USERNAME`.

Examples

Example usage:

```
./pegleg.sh site -r /opt/site-manifests/ \
  -u <AUTH_USER> \
  -k /opt/.ssh/git.pub \
  -e global=ssh://REPO_USERNAME@<GIT URL>:29418/global-manifests.git@master \
  <command> <options>
```

Collect

Output complete config for one site.

site_name (Required).

Name of the site.

-s / \-\-save-location (Optional).

Where to output collected documents. If omitted, the results will be dumped to `stdout`.

-x (Optional, validation only).

Will exclude the specified lint option. -w takes priority over -x.

-w (Optional, validation only).

Will warn of lint failures from the specified lint options.

\-\-validate (Optional, validation only, Default=False).

Perform validation of documents prior to collection. See *Lint* for additional information on document linting. It is recommended that document linting be executed prior to document collection. However, `--validate` is False by default for backwards compatibility concerns.

Usage:

```
./pegleg.sh <command> <options> collect <site_name> -s <save_location> \
  -x P001 -w P002 --validate
```

Examples

Example without validation:

```
./pegleg.sh site -r /opt/site-manifests \
  -e global=/opt/manifests \
  collect <site_name> -s /workspace
```

Example with validation:

```
./pegleg.sh site -r /opt/site-manifests \
  -e global=/opt/manifests \
  collect <site_name> -s /workspace -x P004 --validate
```

List

List known sites.

-s / \-\-save-location (Optional, Default=stdout).

Location where the output is saved.

-o / \-\-output (Optional).

Same as -s (–save-location). Deprecated.

```
./pegleg <command> <options> list
```

Examples

Example:

```
./pegleg site -r /opt/site-manifests list -s /workspace
```

Show

Show details for one site.

site_name (Required).

Name of site.

-s / \-\-save-location (Optional, Default=stdout).

Location where the output is saved.

-o / \-\-output (Optional).

Same as -s (–save-location). Deprecated.

```
./pegleg <command> <options> show site_name
```

Examples

Example:

```
./pegleg site -r /opt/site-manifests show site_name -s /workspace
```

Render

Render documents via Deckhand for one site.

site_name (Required).

Name of site.

-s / \-\-save-location (Optional, Default=stdout).

Location where the output is saved.

-o / \-\-output (Optional).

Same as -s (–save-location). Deprecated.

-v / \-\-validate (Optional, Default=True).

Whether to pre-validate documents using built-in schema validation. Skips over externally registered DataSchema documents to avoid false positives.

```
./pegleg <command> <options> render site_name
```

Examples

Example:

```
./pegleg site -r /opt/site-manifests render site_name -s save_location
```

Lint

Sanity checks for repository content (for a specific site in the repository). Validations for linting are done utilizing Deckhand Validations.

To lint all sites in the repository, see *repository-level linting*.

See *Linting* for more information.

Examples

Generic example:

```
./pegleg.sh site -r <site_repo> -e <extra_repo> \
  lint <site_name> \
  -f -x <lint_code> -w <lint_code>
```

The most basic way to lint a document set is as follows:

```
./pegleg.sh site -r <site_repo> -e <extra_repo> lint <site_name>
```

A more complex example involves excluding certain linting checks:

```
./pegleg.sh site -r /opt/site-manifests \
  -e global=/opt/manifests \
  lint <site_name> \
  -x P001 -x P002 -w P003
```

Upload

Uploads documents to Shipyard.

site_name (Required).

Name of the site. The `site_name` must match a `site` name in the site repository folder structure

\-\-os-<various> (Required).

Shipyard needs these options for authenticating with OpenStack Keystone. This option can be set as environment variables or it can be passed via the command line.

Please reference Shipyard's CLI documentation for information related to these options.

\-\-context-marker (Optional).

Specifies a UUID (8-4-4-4-12 format) that will be used to correlate logs, transactions, etc. in downstream activities triggered by this interaction.

-b / \-\-buffer-mode (Optional, Default=auto).

Set the buffer mode when uploading documents. Supported buffer modes include append, replace, auto.

append: Add the collection to the Shipyard Buffer, only if that collection does not already exist in the Shipyard buffer.

replace: Clear the Shipyard Buffer before adding the specified collection.

\-\-collection (Required, Default=<site_name>).

Specifies the name of the compiled collection of documents that will be uploaded to Shipyard.

Usage:

```
./pegleg.sh site <options> upload <site_name> --context-marker=<uuid> \
                                              --buffer-mode=<buffer> \
                                              --collection=<collection>
```

Site Secrets Group

Subgroup of *Site Group*. The commands below create *PeglegManagedDocument manifests* in the local repository.

A sub-group of site command group, which allows you to perform secrets level operations for secrets documents of a site.

Note: For the CLI commands `encrypt`, `decrypt`, `generate certificates` and `wrap` in the `secrets` command group, which encrypt or decrypt site secrets, two environment variables, `PEGLEG_PASSPHRASE`, and `PEGLEG_SALT`, are used to capture the master passphrase, and the salt needed for encryption and decryption of the site secrets. The contents of `PEGLEG_PASSPHRASE`, and `PEGLEG_SALT` are not generated by Pegleg, but are created externally, and set by deployment engineers or tooling.

A minimum length of 24 for master passphrases will be checked by all CLI commands, which use the `PEGLEG_PASSPHRASE` and `PEGLEG_SALT`. All other criteria around master passphrase strength are assumed to be enforced elsewhere.

```
./pegleg.sh site -r <site_repo> -e <extra_repo> secrets <command> <options>
```

Generate PKI (deprecated)

Generate certificates and keys according to all PKICatalog documents in the site using the *Public Key Infrastructure (PKI) Catalog* module. The default behavior is to generate all certificates that are not yet present. For example, the first time generate PKI is run or when new entries are added to the PKICatalogue, only those new entries will be generated on subsequent runs.

Pegleg also supports a full regeneration of all certificates at any time, by using the –regenerate-all flag.

Pegleg places generated document files in `<site>/secrets/passphrases`, `<site>/secrets/certificates`, or `<site>/secrets/keypairs` as appropriate:

- The generated filenames for passphrases will follow the pattern `<passphrase-doc-name>.yaml`.
- The generated filenames for certificate authorities will follow the pattern `<ca-name>_ca.yaml`.
- The generated filenames for certificates will follow the pattern `<ca-name>_<certificate-doc-name>_certificate.yaml`.
- The generated filenames for certificate keys will follow the pattern `<ca-name>_<certificate-doc-name>_key.yaml`.
- The generated filenames for keypairs will follow the pattern `<keypair-doc-name>.yaml`.

Dashes in the document names will be converted to underscores for consistency.

site_name (Required).

Name of site.

-a / \-\-author (Optional).

Identifying name of the author generating new certificates. Used for tracking provenance information in the PeglegManagedDocuments. An attempt is made to automatically determine this value, but should be provided.

-d / \-\-days (Optional, Default=365).

Duration (in days) certificates should be valid. Minimum=0, no maximum. Values less than 0 will raise an exception.

NOTE: A generated certificate where days = 0 should only be used for testing. A certificate generated in such a way will be valid for 0 seconds.

\-\-regenerate-all (Optional, Default=False).

Force Pegleg to regenerate all PKI items.

Examples

```
./pegleg.sh site -r <site_repo> -e <extra_repo> \
  secrets generate-pki \
  <site_name> \
  -a <author> \
  -d <days> \
  --regenerate-all
```

Check PKI Certs

Determine if any PKI certificates from a site are expired, or will be expired within `days` days. If any are found, print the cert names and expiration dates to `stdout`.

-d / \-\-days (Optional, Default=60).

Duration (in days) to check certificate validity from today. Minimum=0, no maximum. Values less than 0 will raise an exception.

NOTE: Checking PKI certs where days = 0 will check for certs that are expired at the time the command is run.

site_name (Required).

Name of the `site`. The `site_name` must match a `site` name in the site repository folder structure.

Usage:

```
./pegleg.sh site -r <site_repo> \
  secrets check-pki-certs <site_name> <options>
```

Examples

Example without days specified:

```
./pegleg.sh site -r <site_repo> secrets check-pki-certs <site_name>
```

Example with days specified:

```
./pegleg.sh site -r <site_repo> secrets check-pki-certs <site_name> -d <days>
```

Secrets

A sub-group of site command group, which allows you to perform secrets level operations for secrets documents of a site.

Note: For the CLI commands `encrypt` and `decrypt` in the `secrets` command group, which encrypt or decrypt site secrets, two environment variables, `PEGLEG_PASSPHRASE`, and `PEGLEG_SALT`, are used to capture the master passphrase, and the salt needed for encryption and decryption of the site secrets. The contents of `PEGLEG_PASSPHRASE`, and `PEGLEG_SALT` are not generated by Pegleg, but are created externally, and set by a deployment engineers or tooling.

A minimum length of 24 for master passphrases will be checked by all CLI commands, which use the `PEGLEG_PASSPHRASE`. All other criteria around master passphrase strength are assumed to be enforced elsewhere.

```
./pegleg.sh site -r <site_repo> -e <extra_repo> secrets <command> <options>
```

Encrypt

Encrypt one site's secrets documents, which have the `metadata.storagePolicy` set to encrypted, and wrap them in Pegleg Managed Documents

Note: The encrypt command is idempotent. If the command is executed more than once for a given site, it will skip the files, which are already encrypted and wrapped in a pegleg managed document, and will only encrypt the documents not encrypted before.

site_name (Required).

Name of the `site`. The `site_name` must match a `site` name in the site repository folder structure. The `encrypt` command looks up the `site-name` in the site repository, and searches recursively the `site_name` folder structure for secrets files (i.e. files with documents, whose `encryptionPolicy` is set to `encrypted`), and encrypts the documents in those files.

-p / \-\-path (Optional).

The file or directory path to encrypt. If a path is not provided, all applicable files discovered in the user specified repositories for `site_name` will be encrypted.

-a / \-\-author (Required).

Author is the identifier for the program or the person, who is encrypting the secrets documents. Author is intended to document the entity or the individual, who encrypts the site secrets documents, mostly for tracking purposes, and is expected to be leveraged in an operator-specific manner. For instance the `author` can be the "userid" of the person running the command, or the "application-id" of the application executing the command.

-s / \-\-save-location (Optional).

Where to output the encrypted and wrapped documents.

Warning: If the `save-location` parameter is not provided, the encrypted result documents will overwrite the original `cleartext` documents for the site. The reason for this default behavior, is to ensure that site secrets are only stored on disk or in any version control system as encrypted.

If the user for any reason wants to avoid overwriting the original cleartext files, the `save-location` parameter will provide the option to override this default behavior, and forces the encrypt command to write the encrypted documents in a different location than the original unencrypted files.

Usage:

```
./pegleg.sh site <options> secrets encrypt <site_name> -a <author_id> -s <save_
↪location>
```

Examples

Example with optional save location:

```
./pegleg.sh site -r /opt/site-manifests \
  -e global=/opt/manifests \
  -e secrets=/opt/security-manifests \
  secrets encrypt <site_name> -a <author_id> -s /workspace
```

Example without optional save location:

```
./pegleg.sh site -r /opt/site-manifests \
  -e global=/opt/manifests \
  -e secrets=/opt/security-manifests \
  secrets encrypt <site_name> -a <author_id>
```

Decrypt

Unwrap one or more encrypted secrets document from Pegleg Managed Documents, decrypt the encrypted secrets, and dump the cleartext to stdout or a specified location.

site_name (Required).

Name of the `site`. The `site_name` must match a `site` name in the site repository folder structure. This is used to ensure the correct revision of the site and global repositories are used, as specified in the site's `site-definition.yaml`.

\-\-path (Required). Multiple entries allowed.

Path to pegleg managed encrypted secrets file or directory of files.

-s / \-\-save-location (Optional).

The desired output path for the decrypted file. If not specified, decrypted data will output to stdout.

-o / \-\-overwrite (Optional). False by default.

When set, encrypted file(s) at the specified path will be overwritten with the decrypted data. Overrides `--save-location` option.

Usage:

```
./pegleg.sh site <options> secrets decrypt <site_name> --path <path>
  [-s <output_path>]
```

Examples

Example:

```
./pegleg.sh site -r /opt/site-manifests \
  -e global=/opt/manifests \
  -e secrets=/opt/security-manifests \
  secrets decrypt site1 \
  --path /opt/security-manifests/site/site1/passwords/password1.yaml \
  --path /opt/security-manifests/site/site1/passwords/password2.yaml \
  --path /opt/security-manifests/site/site1/passwords/passwordN.yaml \
  --path /opt/security-manifests/site/site1/certificates
```

Wrap

Wrap bare files (e.g. pem or crt) in a PeglegManagedDocument and optionally encrypt them.

site_name (Required).

Name of site.

-a / \-\-author

Identifying name of the author generating new certificates. Used for tracking provenance information in the PeglegManagedDocuments. An attempt is made to automatically determine this value, but should be provided.

\-\-filename

The relative path to the file to be wrapped.

\-\-save-location

The output path where the wrapped file is saved. (default: input path with the extension replaced with .yaml)

-o / \-\-output-path

Same as –save-location. Deprecated.

-s / \-\-schema

The schema for the document to be wrapped, e.g. deckhand/Certificate/v1

-n / \-\-name

The name for the document to be wrapped, e.g. new-cert.

-l / \-\-layer

The layer for the document to be wrapped, e.g. site.

\-\-encrypt / \-\-no-encrypt (Default=True).

A flag specifying whether to encrypt the output file.

Examples

```
./pegleg.sh site -r /home/myuser/myrepo \
  secrets wrap -a myuser --filename secrets/certificates/new_cert.crt \
  --save-location secrets/certificates/new_cert.yaml \
  -s "deckhand/Certificate/v1" -n "new-cert" -l site mysite
```

genesis_bundle

Constructs genesis bundle based on a site configuration.

Note: This command requires the environment variable PEGLEG_PASSPHRASE to be set and at least 24 characters long, to be used for encrypting genesis bundle data. PEGLEG_SALT must be set as well. There are no constraints on its length, but at least 24 characters is recommended.

PROMENADE_ENCRYPTION_KEY environment variable is required only if manifests specify an encryption policy. There are no constraints on its length or character pools.

-b / \-\-build-dir (Required).

Destination directory for the genesis bundle.

\-\-include-validators (Optional, Default=False).

A flag to request build genesis validation scripts as well.

Usage:

::

> **./pegleg.sh site <options> genesis_bundle <site_name>** -b <build_locaton> -k <encryption_passphrase/key> –include-validators

Examples

```
./pegleg.sh site -r  ./site-manifests \
  genesis_bundle site1 \
  -b ../../site1_build \
  -k yourEncryptionPassphrase \
  --include-validators
```

generate

A sub-group of secrets command group, which allows you to auto-generate secrets documents of a site.

Note: The types of documents that pegleg cli generates are passphrases, certificate authorities, certificates and keys. Passphrases are declared in a new `pegleg/PassphraseCatalog/v1` document, while CAs, certificates, and keys are declared in the `pegleg/PKICatalog/v1`.

The `pegleg/PKICatalog/v1` schema is identical with the existing `promenade/PKICatalog/v1`, promenade currently uses to generate the site CAs, certificates, and keys.

The `pegleg/PassphraseCatalog/v1` schema is specified in Pegleg Passphrase Catalog

```
./pegleg.sh site -r <site_repo> -e <extra_repo> secrets generate <command> <options>
```

certificates

Generate certificates and keys according to all PKICatalog documents in the site using the *Public Key Infrastructure (PKI) Catalog* module. The default behavior is to generate all certificates that are not yet present. For example, the first time generate PKI is run or when new entries are added to the PKICatalogue, only those new entries will be generated on subsequent runs.

Pegleg also supports a full regeneration of all certificates at any time, by using the \-\-regenerate-all flag.

Pegleg places generated document files in `<site>/secrets/passphrases`, `<site>/secrets/certificates`, or `<site>/secrets/keypairs` as appropriate:

- The generated filenames for passphrases will follow the pattern `<passphrase-doc-name>.yaml`.
- The generated filenames for certificate authorities will follow the pattern `<ca-name>_ca.yaml`.
- The generated filenames for certificates will follow the pattern `<ca-name>_<certificate-doc-name>_certificate.yaml`.

- The generated filenames for certificate keys will follow the pattern `<ca-name>_<certificate-doc-name>_key.yaml`.
- The generated filenames for keypairs will follow the pattern `<keypair-doc-name>.yaml`.

Dashes in the document names will be converted to underscores for consistency.

site_name (Required).

Name of site.

-a / \-\-author (Optional).

Identifying name of the author generating new certificates. Used for tracking provenance information in the Pegleg-ManagedDocuments. An attempt is made to automatically determine this value, but should be provided.

-d / \-\-days (Optional, Default=365).

Duration (in days) certificates should be valid. Minimum=0, no maximum. Values less than 0 will raise an exception.

NOTE: A generated certificate where days = 0 should only be used for testing. A certificate generated in such a way will be valid for 0 seconds.

-s / \-\-save-location

Directory to store the generated site certificates in. It will be created automatically, if it does not already exist. The generated, wrapped, and encrypted passphrases files will be saved in: <save_location>/site/<site_name>/secrets/certificates/ directory. Defaults to site repository path if no value given.’

\-\-regenerate-all (Optional, Default=False).

Force Pegleg to regenerate all PKI items.

Examples

```
./pegleg.sh site -r <site_repo> -e <extra_repo> \
  secrets generate certificates \
  <site_name> \
  -a <author> \
  -d <days> \
  -s <save_location>
  --regenerate-all
```

passphrases

Generates, wraps and encrypts passphrase documents specified in the `pegleg/PassphraseCatalog/v1` document for a site. The site name, and the directory to store the generated documents are provided by the `site_name`, and the `save_location` command line parameters respectively. The generated passphrases are stored in:

```
<save_location>/site/<site_name>/passphrases/<passphrase_name.yaml>
```

The schema for the generated passphrases is defined in Pegleg Managed Documents

site_name (Required).

Name of the `site`. The `site_name` must match a `site` name in the site repository folder structure. The `generate` command looks up the `site-name`, and searches recursively the `site_name` folder structure in the site repository for pegleg/PassphraseCatalog/v1 documents. Then it parses the passphrase catalog documents it found, and generates one passphrase document for each passphrase document_name declared in the site passphrase catalog.

-a / \-\-author (Required)

Author is intended to document the application or the individual, who generates the site passphrase documents, mostly for tracking purposes. It is expected to be leveraged in an operator-specific manner. For instance the author can be the "userid" of the person running the command, or the "application-id" of the application executing the command.

-s / \-\-save-location (Required).

Where to output generated passphrase documents. The passphrase documents are placed in the following folder structure under save_location:

```
<save_location>/site/<site_name>/secrets/passphrases/<passphrase_name.yaml>
```

-c / \-\-passphrase-catalog (Optional).

Specifies a path for a passphrase catalog file to use instead of the catalogs found in the repositories specified by the user. The specified catalog will be used when this option is specified and all other discovered catalogs will be disregarded. This can be used to specify a subset of passphrases to generate instead of the whole catalog or for testing new passphrases before merging them into production.

-i / \-\-interactive (Optional). False by default.

Enables input prompts for "prompt: true" passphrases. Input prompts are otherwise disabled by default and prompted passphrases will be skipped.

\-\-force-cleartext (Optional). False by default.

Force cleartext generation of passphrases. This is not recommended.

Usage:

```
./pegleg.sh site <options> secrets generate passphrases <site_name> -a
<author_id> -s <save_location>
```

Example

```
./pegleg.sh site -r /opt/site-manifests \
  -e global=/opt/manifests \
  -e secrets=/opt/security-manifests \
  secrets generate passphrases <site_name> -a <author_id> -s /workspace
```

4.1.6 CLI Repository Overrides

Repository overrides should only be used for entries included underneath the repositories field for a given site-definition.yaml.

Overrides are specified via the -e flag for all *Site Group* commands. They have the following format:

```
-e <REPO_NAME>=<REPO_PATH_OR_URL>@<REVISION>
```

Where:

- REPO_NAME is one of: global, secrets or site.

- REPO_PATH_OR_URL is one of:
  - path (relative or absolute) - /opt/global or ../global though absolute is recommended
  - url (fully qualified) - must have following formats:
    - ssh - <PROTOCOL>://<REPO_USERNAME>@<GIT URL>:<PORT>/<REPO_NAME>.git
    - http|https - <PROTOCOL>://<GIT URL>/<REPO_NAME>.git

  Where:

  - <PROTOCOL> must be a valid authentication protocol: ssh, https, or http
  - <REPO_USERNAME> must be a user with access rights to the repository. This value will replace the literal string REPO_USERNAME in the corresponding entry under the `repositories` field in the relevant `site-definition.yaml` using `-u` CLI flag
  - <GIT_URL> must be a valid Git URL
  - <PORT> must be a valid authentication port for SSH
  - <REVISION> must be a valid *Git Reference*
  - <REPO_NAME> must be a valid Git repository name, e.g. site-manifests

Self-Contained Repository

For self-contained repositories, specification of extra repositories is unnecessary. The following command can be used to deploy the manifests in the example repository `/opt/airship-in-a-bottle` for the *currently checked out revision*:

```
pegleg site -r /opt/airship-in-a-bottle/deployment_files <command> <options>
```

To specify a specific revision at run time, execute:

```
pegleg site -r /opt/airship-in-a-bottle/deployment_files@<REVISION> \
  <command> <options>
```

Where `<REVISION>` must be a valid *Git Reference*.

Git Reference

Valid Git references for checking out repositories include:

- 47676764d3935e4934624bf9593e9115984fe668 (commit ID)
- refs/changes/79/47079/12 (ref)
- master (branch name)

4.1.7 Linting

-f / \-\-fail-on-missing-sub-src (Optional, Default=True).

Raise Deckhand exception on missing substitution sources.

-x (Optional).

Will exclude the specified lint option. -w takes priority over -x.

-w (Optional).

Will warn of lint failures from the specified lint options.

If you expect certain lint failures, then those lint options can be excluded or you can choose to be warned about those failures using the codes below.

P001 - Document has storagePolicy cleartext (expected is encrypted) yet its schema is a mandatory encrypted type.

Where mandatory encrypted schema type is one of:

- deckhand/CertificateAuthorityKey/v1
- deckhand/CertificateKey/v1
- deckhand/Passphrase/v1
- deckhand/PrivateKey/v1

P002 - Deckhand rendering is expected to complete without errors. P003 - All repos contain expected directories.

4.1.8 Generate

Allows you to perform generate operations.

Passphrase

Generate a passphrase and print to `stdout`.

-l / \-\-length (Optional, Default=24).

Length of passphrase to generate. Minimum length is 24. Lengths less than minimum will default to 24. No maximum length.

Usage:

```
./pegleg.sh generate passphrase -l <length>
```

Examples

Example without length specified:

```
./pegleg.sh generate passphrase
```

Example with length specified:

```
./pegleg.sh generate passphrase -l <length>
```

Salt

Generate a salt and print to `stdout`.

-l / \-\-length (Optional, Default=24).

Length of salt to generate. Minimum length is 24. Lengths less than minimum will default to 24. No maximum length.

Usage:

```
./pegleg.sh generate salt -l <length>
```

Examples

Example without length specified:

```
./pegleg.sh generate salt
```

Example with length specified:

```
./pegleg.sh generate salt -l <length>
```

4.2 Lint Codes

4.2.1 Overview

Below are the lint codes that are used by the *lint* Pegleg CLI command.

4.2.2 Codes

- P001 - Document has storagePolicy cleartext (expected is encrypted) yet its schema is a mandatory encrypted type.

  Where mandatory encrypted schema type is one of:

  - `deckhand/CertificateAuthorityKey/v1`
  - `deckhand/CertificateKey/v1`
  - `deckhand/Passphrase/v1`
  - `deckhand/PrivateKey/v1`

  See the Deckhand Utility Document Kinds documentation for more information.

- P003 - All repos contain expected directories.
- P004 - Duplicate Deckhand DataSchema document detected.
- P005 - Deckhand rendering exception.
- P006 - YAML file missing document header (`---`).
- P007 - YAML file is not valid YAML.
- P008 - Document `metadata.layeringDefinition.layer` does not match its location in the site manifests tree (e.g. document with `site` layer should be found in folder named `site`).

- P009 - Document found in `secrets` folder in site manifests repository but doesn't have `storagePolicy: encrypted` set.
- P010 - Site folder in manifests repository is missing `site-definition.yaml`
- P011 - `site-definition.yaml` failed Pegleg schema validation.

4.3 Pegleg Exceptions

4.3.1 Base Exceptions

exception `pegleg.engine.exceptions.`**PeglegBaseException**(*message=None*, ***kwargs*)

The base Pegleg exception for everything.

message = 'Base Pegleg exception'

4.3.2 Git Exceptions

exception `pegleg.engine.exceptions.`**GitConfigException**(*message=None*, ***kwargs*)

Bases: *pegleg.engine.exceptions.PeglegBaseException*

Exception that occurs when reading Git repo config fails.

message = 'Failed to read Git config file for repo path: %(repo_url)s'

exception `pegleg.engine.exceptions.`**GitException**(*message=None*, ***kwargs*)

Bases: *pegleg.engine.exceptions.PeglegBaseException*

Exception when an error occurs cloning a Git repository.

message = 'Git exception occurred: [%(location)s] may not be a valid git repository.

exception `pegleg.engine.exceptions.`**GitAuthException**(*message=None*, ***kwargs*)

Bases: *pegleg.engine.exceptions.PeglegBaseException*

Exception that occurs when authentication fails for cloning a repo.

message = 'Failed to authenticate for repo %(repo_url)s with ssh-key at path %(ssh_key_

exception `pegleg.engine.exceptions.`**GitProxyException**(*message=None*, ***kwargs*)

Bases: *pegleg.engine.exceptions.PeglegBaseException*

Exception when cloning through proxy.

message = 'Could not resolve proxy [%(location)s]'

exception `pegleg.engine.exceptions.`**GitSSHException**(*message=None*, ***kwargs*)

Bases: *pegleg.engine.exceptions.PeglegBaseException*

Exception that occurs when an SSH key could not be found.

message = 'Failed to find specified SSH key: %(ssh_key_path)s'

exception `pegleg.engine.exceptions.`**GitInvalidRepoException**(*message=None*, ***kwargs*)

Bases: *pegleg.engine.exceptions.PeglegBaseException*

Exception raised when an invalid repository is detected.

message = 'The repository path or URL is invalid: %(repo_url)s'

4.3.3 Authentication Exceptions

exception pegleg.engine.util.shipyard_helper.**AuthValuesError**(*, *diagnostic*)

Shipyard authentication failed.

4.3.4 PKI Exceptions

exception pegleg.engine.exceptions.**IncompletePKIPairError**(*message=None*, ***kwargs*)

Exception for incomplete private/public keypair.

4.3.5 Genesis Bundle Exceptions

exception pegleg.engine.exceptions.**GenesisBundleEncryptionException**(*message=None*, ***kwargs*)

Bases: *pegleg.engine.exceptions.PeglegBaseException*

Exception raised when encryption of the genesis bundle fails.

message = 'Encryption is required for genesis bundle, but no encryption policy or key

exception pegleg.engine.exceptions.**GenesisBundleGenerateException**(*message=None*, ***kwargs*)

Bases: *pegleg.engine.exceptions.PeglegBaseException*

Exception raised when pormenade engine fails to build the genesis bundle.

4.3.6 Passphrase Exceptions

exception pegleg.engine.exceptions.**PassphraseCatalogNotFoundException**(*message=None*, ***kwargs*)

Bases: *pegleg.engine.exceptions.PeglegBaseException*

Failed to find Catalog for Passphrases generation.

message = 'Could not find the Passphrase Catalog to generate the site Passphrases!'

INDEX