Shipyard Documentation

Release 0.1.0

Shipyard Authors

Sep 15, 2023

Contents

Shipyard is a directed acyclic graph controller for Kubernetes and OpenStack control plane life-cycle management, and is part of the Airship platform.

CHAPTER 1

Sample Configuration File

The following is a sample Shipyard configuration for adaptation and use. It is auto-generated from Shipyard when this documentation is built, so if you are having issues with an option, please compare your version of Shipyard with the version of this documentation.

The sample configuration can also be viewed in file form.

```
[DEFAULT]


[airflow]

#
# From shipyard_api
#

# Airflow worker url scheme (string value)
#worker_endpoint_scheme = http

# Airflow worker port (integer value)
#worker_port = 8793


[armada]

#
# From shipyard_api
#

# The service type for the service playing the role of Armada. The specified
# type is used to perform the service lookup in the Keystone service catalog.
# (string value)
#service_type = armada
```

(continues on next page)

(continued from previous page)

```
[base]

#
# From shipyard_api
#

# The web server for Airflow (string value)
#web_server = http://localhost:8080/

# Seconds to wait to connect to the airflow api (integer value)
#airflow_api_connect_timeout = 5

# Seconds to wait for a response from the airflow api (integer value)
#airflow_api_read_timeout = 60

# The database for shipyard (string value)
#postgresql_db = postgresql+psycopg2://shipyard:changeme@postgresql.ucp:5432/shipyard

# The database for airflow (string value)
#postgresql_airflow_db = postgresql+psycopg2://shipyard:changeme@postgresql.ucp:5432/
↪airflow

# The SQLalchemy database connection pool size. (integer value)
#pool_size = 15

# Should DB connections be validated prior to use. (boolean value)
#pool_pre_ping = true

# How long a request for a connection should wait before one becomes available.
# (integer value)
#pool_timeout = 30

# How many connections above pool_size are allowed to be open during high
# usage. (integer value)
#pool_overflow = 10

# Time, in seconds, when a connection should be closed and re-established. -1
# for no recycling. (integer value)
#connection_recycle = -1

# The directory containing the alembic.ini file (string value)
#alembic_ini_path = /home/shipyard/shipyard

# Enable profiling of API requests. Do NOT use in production. (boolean value)
#profiler = false


[deckhand]

#
# From shipyard_api
#

# The service type for the service playing the role of Deckhand. The specified
# type is used to perform the service lookup in the Keystone service catalog.
# (string value)
#service_type = deckhand
```

(continues on next page)

(continued from previous page)

```
[deployment_status_configmap]

#
# From shipyard_api
#

# Name of the Deployment Status ConfigMap (string value)
#name = deployment-status

# Namespace of the Deployment Status ConfigMap (string value)
#namespace = ucp


[document_info]

#
# From shipyard_api
#

# The name of the deployment version document that Shipyard validates (string
# value)
#deployment_version_name = deployment-version

# The schema of the deployment version document that Shipyard validates (string
# value)
#deployment_version_schema = pegleg/DeploymentData/v1

# The name of the deployment-configuration document that Shipyard expects and
# validates (string value)
#deployment_configuration_name = deployment-configuration

# The schema of the deployment-configuration document that Shipyard expects and
# validates (string value)
#deployment_configuration_schema = shipyard/DeploymentConfiguration/v1

# The schema of the deployment strategy document that Shipyard expects and
# validates. Note that the name of this document is not configurable, because
# it is controlled by a field in the deployment configuration document. (string
# value)
#deployment_strategy_schema = shipyard/DeploymentStrategy/v1


[drydock]

#
# From shipyard_api
#

# The service type for the service playing the role of Drydock. The specified
# type is used to perform the service lookup in the Keystone service catalog.
# (string value)
#service_type = physicalprovisioner


[k8s_logs]
```

(continues on next page)

(continued from previous page)

```
#
# From shipyard_api
#

# Namespace of Airship Pods (string value)
#ucp_namespace = ucp


[keystone_authtoken]

#
# From keystonemiddleware.auth_token
#

# Complete "public" Identity API endpoint. This endpoint should not be an
# "admin" endpoint, as it should be accessible by all end users.
# Unauthenticated clients are redirected to this endpoint to authenticate.
# Although this endpoint should ideally be unversioned, client support in the
# wild varies. If you're using a versioned v2 endpoint here, then this should
# *not* be the same endpoint the service user utilizes for validating tokens,
# because normal end users may not be able to reach that endpoint. (string
# value)
# Deprecated group/name - [keystone_authtoken]/auth_uri
#www_authenticate_uri = <None>

# DEPRECATED: Complete "public" Identity API endpoint. This endpoint should not
# be an "admin" endpoint, as it should be accessible by all end users.
# Unauthenticated clients are redirected to this endpoint to authenticate.
# Although this endpoint should ideally be unversioned, client support in the
# wild varies. If you're using a versioned v2 endpoint here, then this should
# *not* be the same endpoint the service user utilizes for validating tokens,
# because normal end users may not be able to reach that endpoint. This option
# is deprecated in favor of www_authenticate_uri and will be removed in the S
# release. (string value)
# This option is deprecated for removal since Queens.
# Its value may be silently ignored in the future.
# Reason: The auth_uri option is deprecated in favor of www_authenticate_uri
# and will be removed in the S  release.
#auth_uri = <None>

# API version of the Identity API endpoint. (string value)
#auth_version = <None>

# Interface to use for the Identity API endpoint. Valid values are "public",
# "internal" (default) or "admin". (string value)
#interface = internal

# Do not handle authorization requests within the middleware, but delegate the
# authorization decision to downstream WSGI components. (boolean value)
#delay_auth_decision = false

# Request timeout value for communicating with Identity API server. (integer
# value)
#http_connect_timeout = <None>

# How many times are we trying to reconnect when communicating with Identity
```

(continues on next page)

(continued from previous page)

```
# API Server. (integer value)
#http_request_max_retries = 3

# Request environment key where the Swift cache object is stored. When
# auth_token middleware is deployed with a Swift cache, use this option to have
# the middleware share a caching backend with swift. Otherwise, use the
# ``memcached_servers`` option instead. (string value)
#cache = <None>

# Required if identity server requires client certificate (string value)
#certfile = <None>

# Required if identity server requires client certificate (string value)
#keyfile = <None>

# A PEM encoded Certificate Authority to use when verifying HTTPs connections.
# Defaults to system CAs. (string value)
#cafile = <None>

# Verify HTTPS connections. (boolean value)
#insecure = false

# The region in which the identity server can be found. (string value)
#region_name = <None>

# Optionally specify a list of memcached server(s) to use for caching. If left
# undefined, tokens will instead be cached in-process. (list value)
# Deprecated group/name - [keystone_authtoken]/memcache_servers
#memcached_servers = <None>

# In order to prevent excessive effort spent validating tokens, the middleware
# caches previously-seen tokens for a configurable duration (in seconds). Set
# to -1 to disable caching completely. (integer value)
#token_cache_time = 300

# (Optional) If defined, indicate whether token data should be authenticated or
# authenticated and encrypted. If MAC, token data is authenticated (with HMAC)
# in the cache. If ENCRYPT, token data is encrypted and authenticated in the
# cache. If the value is not one of these options or empty, auth_token will
# raise an exception on initialization. (string value)
# Possible values:
# None - <No description provided>
# MAC - <No description provided>
# ENCRYPT - <No description provided>
#memcache_security_strategy = None

# (Optional, mandatory if memcache_security_strategy is defined) This string is
# used for key derivation. (string value)
#memcache_secret_key = <None>

# (Optional) Number of seconds memcached server is considered dead before it is
# tried again. (integer value)
#memcache_pool_dead_retry = 300

# (Optional) Maximum total number of open connections to every memcached
# server. (integer value)
#memcache_pool_maxsize = 10
```

(continues on next page)

(continued from previous page)

```
# (Optional) Socket timeout in seconds for communicating with a memcached
# server. (integer value)
#memcache_pool_socket_timeout = 3

# (Optional) Number of seconds a connection to memcached is held unused in the
# pool before it is closed. (integer value)
#memcache_pool_unused_timeout = 60

# (Optional) Number of seconds that an operation will wait to get a memcached
# client connection from the pool. (integer value)
#memcache_pool_conn_get_timeout = 10

# (Optional) Use the advanced (eventlet safe) memcached client pool. (boolean
# value)
#memcache_use_advanced_pool = true

# (Optional) Indicate whether to set the X-Service-Catalog header. If False,
# middleware will not ask for service catalog on token validation and will not
# set the X-Service-Catalog header. (boolean value)
#include_service_catalog = true

# Used to control the use and type of token binding. Can be set to: "disabled"
# to not check token binding. "permissive" (default) to validate binding
# information if the bind type is of a form known to the server and ignore it
# if not. "strict" like "permissive" but if the bind type is unknown the token
# will be rejected. "required" any form of token binding is needed to be
# allowed. Finally the name of a binding method that must be present in tokens.
# (string value)
#enforce_token_bind = permissive

# A choice of roles that must be present in a service token. Service tokens are
# allowed to request that an expired token can be used and so this check should
# tightly control that only actual services should be sending this token. Roles
# here are applied as an ANY check so any role in this list must be present.
# For backwards compatibility reasons this currently only affects the
# allow_expired check. (list value)
#service_token_roles = service

# For backwards compatibility reasons we must let valid service tokens pass
# that don't pass the service_token_roles check as valid. Setting this true
# will become the default in a future release and should be enabled if
# possible. (boolean value)
#service_token_roles_required = false

# The name or type of the service as it appears in the service catalog. This is
# used to validate tokens that have restricted access rules. (string value)
#service_type = <None>

# Authentication type to load (string value)
# Deprecated group/name - [keystone_authtoken]/auth_plugin
#auth_type = <None>

# Config Section from which to load plugin specific options (string value)
#auth_section = <None>

#
```

(continues on next page)

(continued from previous page)

```
# From shipyard_api
#

# PEM encoded Certificate Authority to use when verifying HTTPs connections.
# (string value)
#cafile = <None>

# PEM encoded client certificate cert file (string value)
#certfile = <None>

# PEM encoded client certificate key file (string value)
#keyfile = <None>

# Verify HTTPS connections. (boolean value)
#insecure = false

# Timeout value for http requests (integer value)
#timeout = <None>

# Collect per-API call timing information. (boolean value)
#collect_timing = false

# Log requests to multiple loggers. (boolean value)
#split_loggers = false


[logging]

#
# From shipyard_api
#

# The default logging level for the root logger. ERROR=40, WARNING=30, INFO=20,
# DEBUG=10 (integer value)
#log_level = 10

# The logging levels for named loggers. Use standard representations for
# logging levels: ERROR. WARN, INFO, DEBUG. Configuration file format:
# named_log_levels = keystoneauth:INFO,othlgr:WARN (dict value)
#named_log_levels = keystoneauth:20,keystonemiddleware:20


[promenade]

#
# From shipyard_api
#

# The service type for the service playing the role of Promenade. The specified
# type is used to perform the service lookup in the Keystone service catalog.
# (string value)
#service_type = kubernetesprovisioner


[requests_config]

#
```

(continues on next page)

(continued from previous page)

```
# From shipyard_api
#

# Airflow logs retrieval connect timeout (in seconds) (integer value)
#airflow_log_connect_timeout = 5

# Airflow logs retrieval timeout (in seconds) (integer value)
#airflow_log_read_timeout = 300

# Airship component validation connect timeout (in seconds) (integer value)
#validation_connect_timeout = 5

# Airship component validation timeout (in seconds) (integer value)
#validation_read_timeout = 300

# Maximum time to wait to connect to a note source URL (in seconds) (integer
# value)
#notes_connect_timeout = 5

# Read timeout for a note source URL (in seconds) (integer value)
#notes_read_timeout = 10

# Deckhand client connect timeout (in seconds) (integer value)
#deckhand_client_connect_timeout = 5

# Deckhand client timeout (in seconds) for GET, PUT, POST and DELETE request
# (integer value)
#deckhand_client_read_timeout = 300

# Connect timeout used for connecting to Drydock using the Drydock client (in
# seconds) (integer value)
#drydock_client_connect_timeout = 20

# Read timeout used for responses from Drydock using the Drydock client (in
# seconds) (integer value)
#drydock_client_read_timeout = 300


[shipyard]

#
# From shipyard_api
#

# The service type for the service playing the role of Shipyard. The specified
# type is used to perform the service lookup in the Keystone service catalog.
# (string value)
#service_type = shipyard


[validations]

#
# From shipyard_api
#

# Control the severity of the deployment-version validation during create
```

(continues on next page)

(continued from previous page)

```
# configdocs. (string value)
# Possible values:
# Skip - Skip the validation altogether
# Info - Print an Info level message if the validation fails
# Warning - Print a Warning level message if the validation fails
# Error - Return an error when the validation fails and prevent the configdocs
# create from proceeding
#deployment_version_create = Skip

# Control the severity of the deployment-version validation validation during
# commit configdocs. (string value)
# Possible values:
# Skip - Skip the validation altogether
# Info - Print an Info level message if the validation fails
# Warning - Print a Warning level message if the validation fails
# Error - Return an error when the validation fails and prevent the commit from
# proceeding
#deployment_version_commit = Skip
```

CHAPTER 2

Shipyard API

Logically, the API has several parts, each to handle each area of Shipyard functionality:

1. Document Staging
2. Action Handling
3. Airflow Monitoring
4. Site Statuses
5. Logs Retrieval
6. Notes Handling

2.1 Standards used by the API

See API Conventions

2.1.1 Query Parameters

Query parameters are mostly specific to a Shipyard API resource, but the following are reused to provide a more consistent interface:

verbosity `?verbosity=1`

> Provides the user some control over the level of details provided in a response, with values ranging from 0 (none) to 5 (most). Only some resources are affected by setting verbosity, but all resources will accept the parameter. Setting the verbosity parameter to 0 will instruct the resource to turn off all optional data being returned. The default verbosity level is 1 (summary).

2.2 Notes on examples

Examples assume the following environment variables are set before issuing the curl commands shown:

```
$TOKEN={a valid keystone token}
$URL={the url and port of the shipyard api}
```

- Examples will use json formatted by the jq command for sake of presentation.
- Actual responses will not formatted.
- The use of ellipsis indicate a repeated structure in the case of lists, or prior/subsequent structure unimportant to the example (or considered understood).
- The content-length response headers have been removed so as to not cause confusion with the listed output.

2.2.1 Example response for an invalid token:

```
HTTP/1.1 401 Unauthorized
content-type: application/json
x-shipyard-req: a8194b97-8973-4b04-a3b3-2bd319024c5d
WWW-Authenticate: Keystone uri='http://keystone-api.ucp.svc.cluster.local:80/v3'

{
  "apiVersion": "v1.0",
  "status": "Failure",
  "metadata": {},
  "message": "Unauthenticated",
  "code": "401 Unauthorized",
  "details": {
    "errorList": [
      {
        "message": "Credentials are not established"
      }
    ],
    "errorCount": 1,
    "errorType": "ApiError"
  },
  "kind": "status",
  "reason": "Credentials are not established"
}
```

2.3 Document Staging API

Shipyard will serve as the entrypoint for documents (designs, secrets, configurations, etc...) into a site. Documents are posted to Shipyard in collections, rather than individually. At any point in time, there will be several versions of documents in a site that are accessible via this API:

- The "Committed Documents" version, which represents the last version of documents that were successfully committed with a commit_configdocs action.
- The "Shipyard Buffer" version, which represents the collection of documents that have been ingested by this API since the last committed version. Note that only one set of documents maybe posted to the buffer at a time by default. (This behavior can be overridden by query parameters issued by the user of Shipyard)

- The "Last Site Action" version represents the version of documents associated with the last successful or failed site action.
- The "Successful Site Action" version represents the version of documents associated with the last successful site action.
- Site actions include `deploy_site`, `update_site`, and `update_software`.

All versions of documents rely upon Deckhand for storage. Shipyard uses the tagging features of Deckhand to find the appropriate Committed Documents, Last Site Action, Successful Site Action and Shipyard Buffer version.

2.3.1 /v1.0/configdocs

Represents the site configuration documents' current statuses

GET /v1.0/configdocs

Returns a list of collections including their base and new status.

Note: The output type for this request is 'Content-Type: application/json'

Query Parameters

- version=committed,buffer (default) Indicates which revisions tags to compare. Comparision can only be done between 2 different revision tags and the default behavior is to compare the revision with the 'committed' tag and the one with the 'buffer' tag. Valid revision tags that can be used for comparison using the API include 'buffer', 'committed', 'last_site_action' and 'successful_site_action'.

Responses

200 OK If documents can be retrieved

2.3.2 /v1.0/configdocs/{collection_id}

Represents the site configuration documents

Entity Structure

The documents as noted above (commonly yaml), in a format understood by Deckhand

POST /v1.0/configdocs/{collection_id}

Ingests a collection of documents. Synchronous. If a POST to the commitconfigdocs is already in progress, this POST should be rejected with a 409 error.

Note: The expected input type for this request is 'Content-Type: application/x-yaml'

Query Parameters

- buffermode=append|replace|**rejectOnContents** Indicates how the existing Shipyard Buffer should be handled. By default, Shipyard will reject the POST if contents already exist in the Shipyard Buffer.
  - append: Add the collection to the Shipyard Buffer, only if that collection doesn't already exist in the Shipyard Buffer. If the collection is already present, the request will be rejected and a 409 Conflict will be returned.
  - replace: Clear the Shipyard Buffer before adding the specified collection.
- empty-collection: Set to true to indicate that this collection should be made empty and effectively deleted when the Shipyard Buffer is committed. If this parameter is specified, the POST body will be ignored.

Responses

201 Created If the documents are successfully ingested, even with validation failures. Response message includes:

- a list of validation results
- The response headers will include a Location indicating the GET endpoint to retrieve the configDocs

400 Bad Request When:

- The request is missing a message body, attempting to create a collection with no contents.
- The request has no new/changed contents for the collection.
- The request is missing a Content-Length header.
- The provided document(s) fail Shipyard/Deckhand validations.

409 Conflict A condition in the system is blocking this document ingestion

- If a commitconfigdocs POST is in progress.
- If any collections exist in the Shipyard Buffer unless buffermode=replace or buffermode=append.
- If buffermode=append, and the collection being posted is already in the Shipyard Buffer

GET /v1.0/configdocs/{collection_id}

Returns the source documents for a collection of documents

Note: The output type for this request is 'Content-Type: application/x-yaml'

Query Parameters

- version=committed | last_site_action | successful_site_action | **buffer** Return the documents for the version specified - buffer by default.
- cleartext-secrets=true/**false** If true then returns cleartext secrets in encrypted documents, otherwise those values are redacted.

Responses

200 OK If documents can be retrieved.

- If the response is 200 with an empty response body, this indicates that the buffer version is attempting to 'delete' the collection when it is committed. An empty response body will only be possible for version=buffer.

404 Not Found If the collection is not represented

- When version=buffer, this indicates that no representations of this collection have been POSTed since the last committed version.
- When version=committed, this indicates that either the collection has never existed or has been deleted by a prior commit.

2.3.3 /v1.0/renderedconfigdocs

Represents the site configuration documents, as a whole set - does not consider collections in any way.

GET /v1.0/renderedconfigdocs

Returns the full set of configdocs in their rendered form.

Note: The output type for this request is 'Content-Type: application/x-yaml'

Query Parameters

- version=committed | last_site_action | successful_site_action | **buffer** Return the documents for the version specified - buffer by default.
- cleartext-secrets=true/**false** If true then returns cleartext secrets in encrypted documents, otherwise those values are redacted.

Responses

200 OK If documents can be retrieved.

2.3.4 /v1.0/commitconfigdocs

An RPC style command to trigger a commit of the configuration documents from the Shipyard Buffer to the Committed Documents. This resource will support POST only.

Entity Structure

The response will be the list of validations from all downstream systems that perform validation during the commit process. The structure will match the error response object described in the API Conventions and will be an aggregation of each validating component's responses.

POST /v1.0/commitconfigdocs

Synchronous. Performs the commit of the Shipyard Buffer to the Committed Documents. This invokes each of the validating components to examine the Shipyard Buffer version of the configuration documents and aggregate the responses. While performing this commit, further POSTing of configdocs, or other commits may not be invoked (Shipyard will block those requests with a 409 response). If there are any failures to validate, the Shipyard Buffer and Committed Documents will remain unchanged. If successful, the Shipyard Buffer will be cleared, and the Committed documents will be updated.

Note: If there are unhandled runtime errors during the commitconfigdocs POST, a deadlock situation may be possible. Future enhancements may improve this handling.

Query Parameters

force=true | false By default, false, if there are validation failures the POST will fail with a 400 response. With force=true, allows for the commit to succeed (with a 200 response) even if there are validation failures from downstream components. The aggregate response of validation failures will be returned in this case, but the invalid documents will still be moved from the Shipyard Buffer to the Committed Documents.

dryrun=true | false By default, false. With dryrun=true, the response will contain the validation status for the contents of the buffer. The Shipyard Buffer will not be committed.

Responses

200 OK If the validations are successful. Returns an "empty" structure as as response indicating no errors. A 200 may also be returned if there are validation failures, but the force=true query parameter was specified. In this case, the response will contain the list of validations.

400 Bad Request If the validations fail. Returns a populated response structure containing the aggregation of the failed validations.

409 Conflict If the there is a POST to commitconfigdocs in progress.

Example

```
{
    "apiVersion": "v1",
    "code": "400 Bad Request",
    "details": {
        "errorCount": 2,
        "messageList": [
            {
                "error": true,
                "message": "Error loading effective site: 'NoneType' object is not␣
↪iterable",
                "name": "Drydock"
            },
            {
                "error": true,
                "message": "Armada unable to validate configdocs",
                "name": "Armada"
```

(continues on next page)

(continued from previous page)

```
            }
        ]
    },
    "kind": "Status",
    "message": "Validations failed",
    "metadata": {},
    "reason": "Validation",
    "status": "Failure"
}
```

2.4 Action API

The Shipyard Action API is a resource that allows for creation, control and investigation of triggered workflows. These actions encapsulate a command interface for the Airship Undercloud Platform. See *Action Commands* for supported actions

2.4.1 /v1.0/actions

Entity Structure

A list of actions that have been executed through shipyard's action API.

```
[
  { Action objects summarized, See below},
  ...
]
```

GET /v1.0/actions

Returns the list of actions in the system that have been posted, and are accessible to the current user.

Responses

200 OK If the actions can be retrieved.

Example

```
$ curl -X GET $URL/api/v1.0/actions -H "X-Auth-Token:$TOKEN"

HTTP/1.1 200 OK
x-shipyard-req: 0804d13e-08fc-4e60-a819-3b7532cac4ec
content-type: application/json; charset=UTF-8

[
  {
    "dag_status": "failed",
    "parameters": {},
```

(continues on next page)

(continued from previous page)


```
    "steps": [
      {
        "id": "action_xcom",
        "url": "/actions/01BTP9T2WCE1PAJR2DWYXG805V/steps/action_xcom",
        "index": 1,
        "state": "success"
      },
      {
        "id": "dag_concurrency_check",
        "url": "/actions/01BTP9T2WCE1PAJR2DWYXG805V/steps/dag_concurrency_check",
        "index": 2,
        "state": "success"
      },
      {
        "id": "preflight",
        "url": "/actions/01BTP9T2WCE1PAJR2DWYXG805V/steps/preflight",
        "index": 3,
        "state": "failed"
      },
      ...
    ],
    "action_lifecycle": "Failed",
    "dag_execution_date": "2017-09-23T02:42:12",
    "id": "01BTP9T2WCE1PAJR2DWYXG805V",
    "dag_id": "deploy_site",
    "datetime": "2017-09-23 02:42:06.860597+00:00",
    "user": "shipyard",
    "context_marker": "416dec4b-82f9-4339-8886-3a0c4982aec3",
    "name": "deploy_site"
  },
  ...
]
```


POST /v1.0/actions

Creates an action in the system. This will cause some action to start. The input body to this post will represent an action object that has at least these fields:

name The name of the action to invoke, as noted in *Action Commands*

parameters A dictionary of parameters to use for the trigger invocation. The supported parameters will vary for the action invoked.

```
{
  "name" : "action name",
  "parameters" : { varies by action }
}
```

The POST will synchronously create the action (a shell object that represents a DAG invocation), perform any checks to validate the preconditions to run the DAG, and trigger the invocation of the DAG. The DAG will run asynchronously in airflow.

Query Parameters

allow-intermediate-commits=true | false By default, false. User will not be able to continue with a site action, e.g. update_site if the current committed revision of documents has other prior commits that have not been used as part of a site action. With allow-intermediate-commits=true, it allows user to override the default behavior and continue with the site action. This may be the case when the user is aware of the existence of such commits and/or when such commits are intended.

Responses

201 Created If the action is created successfully, and all preconditions to run the DAG are successful. The response body is the action entity created.

400 Bad Request If the action name doesn't exist, or the input entity is otherwise malformed.

409 Conflict For any failed pre-run validations. The response body is the action entity created, with the failed validations. The DAG will not begin execution in this case.

Example

```
$ curl -D - -d '{"name":"deploy_site"}' -X POST $URL/api/v1.0/actions \
  -H "X-Auth-Token:$TOKEN" -H "content-type:application/json"

HTTP/1.1 201 Created
location: {$URL}/api/v1.0/actions/01BTTMFVDKZFRJM80FGD7J1AKN
x-shipyard-req: 629f2ea2-c59d-46b9-8641-7367a91a7016
content-type: application/json; charset=UTF-8

{
  "dag_status": "SCHEDULED",
  "parameters": {},
  "dag_execution_date": "2017-09-24T19:05:49",
  "id": "01BTTMFVDKZFRJM80FGD7J1AKN",
  "dag_id": "deploy_site",
  "name": "deploy_site",
  "user": "shipyard",
  "context_marker": "629f2ea2-c59d-46b9-8641-7367a91a7016",
  "timestamp": "2017-09-24 19:05:43.603591"
}
```

2.4.2 /v1.0/actions/{action_id}

Each action will be assigned an unique id that can be used to get details for the action, including the execution status.

Entity Structure

All actions will include fields that indicate the following data:

action_lifecycle A summarized value indicating the status or lifecycle phase of the action.

- Pending - The action is scheduled or preparing for execution.
- Processing - The action is underway.

- Complete - The action has completed successfully.
- Failed - The action has encountered an error, and has failed.
- Paused - The action has been paused by a user.

command audit A list of commands that have been issued against the action. Initially, the action listed will be "invoke", but may include "pause", "unpause", or "stop" if those commands are issued.

context_marker The user supplied or system assigned context marker associated with the action

dag_execution_date The execution date assigned by the workflow system during action creation.

dag_status Represents the status that airflow provides for an executing DAG.

datetime The time at which the action was invoked.

id The identifier for the action, a 26 character ULID assigned during the creation of the action.

name The name of the action, e.g.: deploy_site.

parameters The parameters configuring the action that were supplied by the user during action creation.

steps The list of steps for the action, including the status for that step.

user The user who has invoked this action, as acquired from the authorization token.

validations A list of validations that have been done, including any status information for those validations. During the lifecycle of the action, this list of validations may continue to grow.

GET /v1.0/actions/{action_id}

Returns the action entity for the specified id.

Responses

200 OK

Example

```
$ curl -D - -X GET $URL/api/v1.0/actions/01BTTMFVDKZFRJM80FGD7J1AKN \
  -H "X-Auth-Token:$TOKEN"

HTTP/1.1 200 OK
x-shipyard-req: eb3eacb3-4206-40df-bd91-2a3a6d81cd02
content-type: application/json; charset=UTF-8

{
  "name": "deploy_site",
  "dag_execution_date": "2017-09-24T19:05:49",
  "validations": [],
  "id": "01BTTMFVDKZFRJM80FGD7J1AKN",
  "dag_id": "deploy_site",
  "command_audit": [
    {
      "id": "01BTTMG16R9H3Z4JVQNBMRV1MZ",
      "action_id": "01BTTMFVDKZFRJM80FGD7J1AKN",
      "datetime": "2017-09-24 19:05:49.530223+00:00",
```

(continues on next page)

(continued from previous page)


```
      "user": "shipyard",
      "command": "invoke"
    }
  ],
  "user": "shipyard",
  "context_marker": "629f2ea2-c59d-46b9-8641-7367a91a7016",
  "datetime": "2017-09-24 19:05:43.603591+00:00",
  "dag_status": "failed",
  "parameters": {},
  "steps": [
    {
      "id": "action_xcom",
      "url": "/actions/01BTTMFVDKZFRJM80FGD7J1AKN/steps/action_xcom",
      "index": 1,
      "state": "success"
    },
    {
      "id": "dag_concurrency_check",
      "url": "/actions/01BTTMFVDKZFRJM80FGD7J1AKN/steps/dag_concurrency_check",
      "index": 2,
      "state": "success"
    },
    {
      "id": "preflight",
      "url": "/actions/01BTTMFVDKZFRJM80FGD7J1AKN/steps/preflight",
      "index": 3,
      "state": "failed"
    },
    {
      "id": "deckhand_get_design_version",
      "url": "/actions/01BTTMFVDKZFRJM80FGD7J1AKN/steps/deckhand_get_design_version",
      "index": 4,
      "state": null
    },
    ...
  ],
  "action_lifecycle": "Failed"
}
```


2.4.3 /v1.0/actions/{action_id}/validations/{validation_id}

Allows for drilldown to validation detailed info.

Entity Structure

The detailed information for a validation

```
{ TBD }
```

GET /v1.0/actions/{action_id}/validations/{validation_id}

Returns the validation detail by Id for the supplied action Id.

Responses

200 OK

2.4.4 /v1.0/actions/{action_id}/steps/{step_id}

Allow for drilldown to step information. The step information includes details of the steps execution, successful or not, and enough to facilitate troubleshooting in as easy a fashion as possible.

Entity Structure

A step entity represents detailed information representing a single step of execution as part of an action. Not all fields are necessarily represented in every returned entity.

dag_id The name/id of the workflow DAG that contains this step.

duration The duration (seconds) for the step.

end_date The timestamp of the completion of the step.

execution_date The execution date of the workflow that contains this step.

index The numeric value representing the position of this step in the sequence of steps associated with this step.

operator The name of the processing facility used by the workflow system.

queued_dttm The timestamp when the step was enqueued by the workflow system.

start_date The timestamp for the beginning of execution for this step.

state The execution state of the step.

task_id The name of the task used by the workflow system (and also representing this step name queried in the request.

try_number A number of retries taken in the case of failure. Some workflow steps may be configured to retry before considering the step truly failed.

GET /v1.0/actions/{action_id}/steps/{step_id}

Returns the details for a step by id for the given action by Id. #####

Responses

200 OK

Example

```
$ curl -D - \
  -X GET $URL/api/v1.0/actions/01BTTMFVDKZFRJM80FGD7J1AKN/steps/action_xcom \
  -H "X-Auth-Token:$TOKEN"

HTTP/1.1 200 OK
x-shipyard-req: 72daca4d-1f79-4e08-825f-2ad181912a47
```

(continues on next page)

(continued from previous page)

```
content-type: application/json; charset=UTF-8

{
  "end_date": "2017-09-24 19:05:59.446213",
  "duration": 0.165181,
  "queued_dttm": "2017-09-24 19:05:52.993983",
  "operator": "PythonOperator",
  "try_number": 1,
  "task_id": "action_xcom",
  "state": "success",
  "execution_date": "2017-09-24 19:05:49",
  "dag_id": "deploy_site",
  "index": 1,
  "start_date": "2017-09-24 19:05:59.281032"
}
```

2.4.5 /v1.0/actions/{action_id}/control/{control_verb}

Allows for issuing DAG controls against an action.

Entity Structure

None, there is no associated response entity for this resource

POST /v1.0/actions/{action_id}/control/{control_verb}

Trigger a control action against an activity.- this includes: pause, unpause

Responses

202 Accepted

Example

Failure case - command is invalid for the execution state of the action.

```
$ curl -D - \
  -X POST $URL/api/v1.0/actions/01BTTMFVDKZFRJM80FGD7J1AKN/control/pause \
  -H "X-Auth-Token:$TOKEN"

HTTP/1.1 409 Conflict
content-type: application/json
x-shipyard-req: 9c9551e0-335c-4297-af93-8440cc6b324f

{
  "apiVersion": "v1.0",
  "status": "Failure",
  "metadata": {},
  "message": "Unable to pause action",
  "code": "409 Conflict",
```

(continues on next page)

(continued from previous page)

```
  "details": {
    "errorList": [
      {
        "message": "dag_run state must be running, but is failed"
      }
    ],
    "errorCount": 1,
    "errorType": "ApiError"
  },
  "kind": "status",
  "reason": "dag_run state must be running, but is failed"
}
```

Success case

```
$ curl -D - \
  -X POST $URL/api/v1.0/actions/01BTTMFVDKZFRJM80FGD7J1AKN/control/pause \
  -H "X-Auth-Token:$TOKEN"

HTTP/1.1 202 Accepted
content-length: 0
x-shipyard-req: 019fae1c-03b0-4af1-b57d-451ae6ddac77
content-type: application/json; charset=UTF-8
```

2.5 Airflow Monitoring API

Airflow has a primary function of scheduling DAGs, as opposed to Shipyard's primary case of triggering DAGs. Shipyard provides functionality to allow for an operator to monitor and review these scheduled workflows (DAGs) in addition to the ones triggered by Shipyard. This API will allow for accessing Airflow DAGs of any type – providing a peek into the totality of what is happening in Airflow.

2.5.1 /v1.0/workflows

The resource that represents DAGs (workflows) in airflow

Entity Structure

A list of objects representing the DAGs that have run in airflow.

GET /v1.0/workflows

Queries airflow for DAGs that are running or have run (successfully or unsuccessfully) and provides a summary of those things.

Query parameters

since={iso8601 date (past) or duration} optional, a boundary in the past within which to retrieve results. Default is 30 days in the past.

Responses

200 OK

Example

Notice the workflow_id values, these can be used for drilldown.

```
curl -D - -X GET $URL/api/v1.0/workflows -H "X-Auth-Token:$TOKEN"

HTTP/1.1 200 OK
content-type: application/json; charset=UTF-8
x-shipyard-req: 3ab4ccc6-b956-4c7a-9ae6-183c562d8297

[
  {
    "execution_date": "2017-10-09 21:18:56",
    "end_date": null,
    "workflow_id": "deploy_site__2017-10-09T21:18:56.000000",
    "start_date": "2017-10-09 21:18:56.685999",
    "external_trigger": true,
    "dag_id": "deploy_site",
    "state": "failed",
    "run_id": "manual__2017-10-09T21:18:56"
  },
  {
    "execution_date": "2017-10-09 21:19:03",
    "end_date": null,
    "workflow_id": "deploy_site__2017-10-09T21:19:03.000000",
    "start_date": "2017-10-09 21:19:03.361522",
    "external_trigger": true,
    "dag_id": "deploy_site",
    "state": "failed",
    "run_id": "manual__2017-10-09T21:19:03"
  }
  ...
]
```

2.5.2 /v1.0/workflows/{workflow_id}

Entity Structure

An object representing the information available from airflow regarding a DAG's execution

GET /v1.0/workflows/{id}

Further details of a particular workflow's steps. All steps of all sub-dags will be included in the list of steps, as well as section indicating the sub-dags for this parent workflow.

Responses

200 OK

Example

Note: Sub_dags can be queried to restrict to only that sub-dag's steps. e.g. using this as {workflow_id}: deploy_site.preflight.armada_preflight_check__2017-10-09T21:19:03.000000

```
curl -D - \
    -X GET $URL/api/v1.0/workflows/deploy_site__2017-10-09T21:19:03.000000 \
    -H "X-Auth-Token:$TOKEN"

HTTP/1.1 200 OK
content-type: application/json; charset=UTF-8
x-shipyard-req: 98d71530-816a-4692-9df2-68f22c057467

{
  "execution_date": "2017-10-09 21:19:03",
  "end_date": null,
  "workflow_id": "deploy_site__2017-10-09T21:19:03.000000",
  "start_date": "2017-10-09 21:19:03.361522",
  "external_trigger": true,
  "steps": [
    {
      "end_date": "2017-10-09 21:19:14.916220",
      "task_id": "action_xcom",
      "start_date": "2017-10-09 21:19:14.798053",
      "duration": 0.118167,
      "queued_dttm": "2017-10-09 21:19:08.432582",
      "try_number": 1,
      "state": "success",
      "operator": "PythonOperator",
      "dag_id": "deploy_site",
      "execution_date": "2017-10-09 21:19:03"
    },
    {
      "end_date": "2017-10-09 21:19:25.283785",
      "task_id": "dag_concurrency_check",
      "start_date": "2017-10-09 21:19:25.181492",
      "duration": 0.102293,
      "queued_dttm": "2017-10-09 21:19:19.283132",
      "try_number": 1,
      "state": "success",
      "operator": "ConcurrencyCheckOperator",
      "dag_id": "deploy_site",
      "execution_date": "2017-10-09 21:19:03"
    },
    {
      "end_date": "2017-10-09 21:20:05.394677",
      "task_id": "preflight",
      "start_date": "2017-10-09 21:19:34.994775",
      "duration": 30.399902,
      "queued_dttm": "2017-10-09 21:19:28.449848",
      "try_number": 1,
      "state": "failed",
      "operator": "SubDagOperator",
      "dag_id": "deploy_site",
      "execution_date": "2017-10-09 21:19:03"
```

(continues on next page)

(continued from previous page)


```
    },
    ...
  ],
  "dag_id": "deploy_site",
  "state": "failed",
  "run_id": "manual__2017-10-09T21:19:03",
  "sub_dags": [
    {
      "execution_date": "2017-10-09 21:19:03",
      "end_date": null,
      "workflow_id": "deploy_site.preflight__2017-10-09T21:19:03.000000",
      "start_date": "2017-10-09 21:19:35.082479",
      "external_trigger": false,
      "dag_id": "deploy_site.preflight",
      "state": "failed",
      "run_id": "backfill_2017-10-09T21:19:03"
    },
    ...,
    {
      "execution_date": "2017-10-09 21:19:03",
      "end_date": null,
      "workflow_id": "deploy_site.preflight.armada_preflight_check__2017-10-
↪09T21:19:03.000000",
      "start_date": "2017-10-09 21:19:48.265023",
      "external_trigger": false,
      "dag_id": "deploy_site.preflight.armada_preflight_check",
      "state": "failed",
      "run_id": "backfill_2017-10-09T21:19:03"
    }
  ]
}
```


2.6 Site Statuses API

Site Statuses API retrieves node provision status and/or node power state for all nodes in the site.

2.6.1 /v1.0/site_statuses

GET /v1.0/site_statuses

Returns the dictionary with nodes provision status and nodes power state status

Query Parameters

- filters=nodes-provision-status,machines-power-state filters query parameter allows to specify one or more status types to return statuses of those types. The filter value `nodes-provision-status` will fetch provisioning statuses of all nodes in the site. The filter value `machines-power-state` will fetch power states of all baremetal machines in the site. By omitting the filters query parameter, statuses of all status types will be returned. To specify multiple items explicitly, separate items with the URL encoded version of a comma: %2C. e.g.:

```
&filters=nodes-provision-status%2Cmachines-power-state
```

Responses

200 OK If statuses are retrieved successfully.

400 Bad Request If invalid filters option is given.

Example

```
 $ curl -X GET $URL/api/v1.0/site_statuses -H "X-Auth-Token:$TOKEN"

 HTTP/1.1 200 OK
 x-shipyard-req: 0804d13e-08fc-4e60-a819-3b7532cac4ec
 content-type: application/json; charset=UTF-8

{
  {
        "nodes-provision-status": [
          {
            "hostname": "abc.xyz.com",
            "status": "Ready"
          },
          {
            "hostname": "def.xyz.com",
            "status": "Ready"
          }
        ],
        "machines-power-state": [
          {
            "hostname": "abc.xyz.com",
            "power_state": "On",
          },
          {
            "hostname": "def.xyz.com",
            "power_state": "On",
          }
        ]
      }
}
```

```
 $ curl -X GET $URL/api/v1.0/site_statuses?filters=nodes-provision-status \
        -H "X-Auth-Token:$TOKEN"

 HTTP/1.1 200 OK
 x-shipyard-req: 0804d13e-08fc-4e60-a819-3b7532cac4ec
 content-type: application/json; charset=UTF-8

{
  {
        "nodes-provision-status": [
          {
            "hostname": "abc.xyz.com",
            "status": "Ready"
```

(continues on next page)

(continued from previous page)

```
          },
          {
            "hostname": "def.xyz.com",
            "status": "Ready"
          }
        ]
      }
}
```

```
   $ curl -X GET $URL/api/v1.0/site_statuses?filters=machines-power-state \
          -H "X-Auth-Token:$TOKEN"

   HTTP/1.1 200 OK
   x-shipyard-req: 0804d13e-08fc-4e60-a819-3b7532cac4ec
   content-type: application/json; charset=UTF-8

  {
    {
          "machines-power-state": [
            {
              "hostname": "abc.xyz.com",
              "power_state": "On",
            },
            {
              "hostname": "def.xyz.com",
              "power_state": "On",
            }
          ]
        }
  }

::

   $ curl -X GET $URL/api/v1.0/site_statuses?filters=nodes-provision-status
↪%2Cmachines-power-state \
              -H "X-Auth-Token:$TOKEN"

   HTTP/1.1 200 OK
   x-shipyard-req: 0804d13e-08fc-4e60-a819-3b7532cac4ec
   content-type: application/json; charset=UTF-8

  {
    {
          "nodes-provision-status": [
            {
              "hostname": "abc.xyz.com",
              "status": "Ready"
            },
            {
              "hostname": "def.xyz.com",
              "status": "Ready"
            }
          ],
          "machines-power-state": [
            {
              "hostname": "abc.xyz.com",
```

(continues on next page)

(continued from previous page)

```
            "power_state": "On",
          },
          {
            "hostname": "def.xyz.com",
            "power_state": "On",
          }
        ]
      }
}
```

2.7 Logs Retrieval API

This API allows users to query and view logs. Its usuage is currently limited to Airflow logs retrieval but it can be extended in the future to retrieve other logs. For instance, a possible use case might be to retrieve or `tail` the Kubernetes logs.

2.7.1 /v1.0/actions/{action_id}/steps/{step_id}/logs

This API allows users to query and view the logs for a particular workflow step in Airflow. By default, it will retrieve the logs from the last attempt. Note that a workflow step can retry multiple times with the names of the logs as 1.log, 2.log, 3.log, etc. A user can specify the try number to view the logs for a particular failed attempt, which will be useful during a troubleshooting session.

Entity Structure

Raw text of the logs retrieved from Airflow for that particular workflow step.

GET /v1.0/actions/{action_id}/steps/{step_id}/logs

Queries Airflow and retrieves logs for a particular workflow step.

Query parameters

try={int try_number} optional, represents a particular attempt of the workflow step. Default value is set to None.

Responses

200 OK 4xx or 5xx

A 4xx or 5xx code will be returned if some error happens during Airflow HTTP request or Airflow responds with a status code of 400 or greater.

Example

```
curl -D - \
    -X GET $URL/api/v1.0/actions/01CASSSZT7CP1F0NKHCAJBCJGR/steps/action_xcom/logs?
↪try=2 \
    -H "X-Auth-Token:$TOKEN"

HTTP/1.1 200 OK
content-type: application/json; charset=UTF-8
x-shipyard-req: 49f74418-22b3-4629-8ddb-259bdfccf2fd

[2018-04-11 07:30:41,945] {{cli.py:374}} INFO - Running on host airflow-worker-0.
↪airflow-worker-discovery.ucp.svc.cluster.local
[2018-04-11 07:30:41,991] {{models.py:1197}} INFO - Dependencies all met for
↪<TaskInstance: deploy_site.action_xcom 2018-04-11 07:30:37 [queued]>
[2018-04-11 07:30:42,001] {{models.py:1197}} INFO - Dependencies all met for
↪<TaskInstance: deploy_site.action_xcom 2018-04-11 07:30:37 [queued]>
[2018-04-11 07:30:42,001] {{models.py:1407}} INFO -
--------------------------------------------------------------------------------
Starting attempt 2 of 2
--------------------------------------------------------------------------------

[2018-04-11 07:30:42,022] {{models.py:1428}} INFO - Executing <Task(PythonOperator):␣
↪action_xcom> on 2018-04-11 07:30:37
[2018-04-11 07:30:42,023] {{base_task_runner.py:115}} INFO - Running: ['bash', '-c',
↪'airflow run deploy_site action_xcom 2018-04-11T07:30:37 --job_id 2 --raw -sd DAGS_
↪FOLDER/deploy_site.py']
[2018-04-11 07:30:42,606] {{base_task_runner.py:98}} INFO - Subtask: [2018-04-11␣
↪07:30:42,606] {{driver.py:120}} INFO - Generating grammar tables from /usr/lib/
↪python3.5/lib2to3/Grammar.txt
[2018-04-11 07:30:42,635] {{base_task_runner.py:98}} INFO - Subtask: [2018-04-11␣
↪07:30:42,634] {{driver.py:120}} INFO - Generating grammar tables from /usr/lib/
↪python3.5/lib2to3/PatternGrammar.txt
[2018-04-11 07:30:43,515] {{base_task_runner.py:98}} INFO - Subtask: [2018-04-11␣
↪07:30:43,515] {{configuration.py:206}} WARNING - section/key [celery/celery_ssl_
↪active] not found in config
[2018-04-11 07:30:43,516] {{base_task_runner.py:98}} INFO - Subtask: [2018-04-11␣
↪07:30:43,515] {{default_celery.py:41}} WARNING - Celery Executor will run without␣
↪SSL
[2018-04-11 07:30:43,517] {{base_task_runner.py:98}} INFO - Subtask: [2018-04-11␣
↪07:30:43,516] {{__init__.py:45}} INFO - Using executor CeleryExecutor
[2018-04-11 07:30:43,822] {{base_task_runner.py:98}} INFO - Subtask: [2018-04-11␣
↪07:30:43,821] {{models.py:189}} INFO - Filling up the DagBag from /usr/local/
↪airflow/dags/deploy_site.py
[2018-04-11 07:30:43,892] {{cli.py:374}} INFO - Running on host airflow-worker-0.
↪airflow-worker-discovery.ucp.svc.cluster.local
[2018-04-11 07:30:43,945] {{base_task_runner.py:98}} INFO - Subtask: [2018-04-11␣
↪07:30:43,944] {{python_operator.py:90}} INFO - Done. Returned value was: None
[2018-04-11 07:30:43,992] {{base_task_runner.py:98}} INFO - Subtask:   """)
```

2.8 Notes Handling API

The notes facilities of Shipyard are primarily interwoven in other APIs. This endpoint adds the ability to retrieve additional information associated with a note. The first use case for this API is the retrieval of builddata from Drydock, which can be many hundreds of kilobytes of text.

2.8.1 /v1.0/notedetails/{note_id}

Retrieves the note details that are associated via URL with a note at the time of note creation. Unlike some responses from Shipyard, this API returns the remote information as-is, as the response body, without any further wrapping in a JSON structure.

Entity Structure

Raw text of the note's associated information.

GET /v1.0/notedetails/{node_id}

Looks up the specified note and follows the associated URL to retrieve information related to the note.

Query parameters

N/A

Responses

200 OK

Accompanied by the text looked up from the note's associated URL

400 Bad Request

When the note_id is not a valid ULID value.

404 Not Found

When the note does not exist, or the note does not have a URL associated.

500 Internal Server Error

When the remote source of the information cannot be accessed, or if there is a misconfiguration of the type of note preventing appropriate authorization checking.

Example

```
curl -D - \
    -X GET $URL/api/v1.0/notedetails/01CASSSZT7CP1F0NKHCAJBCJGR \
    -H "X-Auth-Token:$TOKEN"

HTTP/1.1 200 OK
x-shipyard-req: 49f74418-22b3-4629-8ddb-259bdfccf2fd

Potentially a lot of information here
```

CHAPTER 3

Action Commands

3.1 Example invocation

API input to create an action follows this pattern, varying the name field:

Without Parmeters:

```
POST /v1.0/actions

{"name" : "update_site"}
```

With Parameters:

```
POST /v1.0/actions

{
  "name": "redeploy_server",
  "parameters": {
    "target_nodes": ["node1", "node2"]
  }
}

POST /v1.0/actions

{
  "name": "update_site",
  "parameters": {
    "continue-on-fail": "true"
  }
}
```

Analogous CLI commands:

```
shipyard create action update_site
shipyard create action redeploy_server --param="target_nodes=node1,node2"
shipyard create action update_site --param="continue-on-fail=true"
```

3.2 Supported actions

These actions are currently supported using the Action API and CLI

3.2.1 deploy_site

Triggers the initial deployment of a site, using the latest committed configuration documents. Steps, conceptually:

1. **Concurrency check** Prevents concurrent site modifications by conflicting actions/workflows.
2. **Preflight checks** Ensures all Airship components are in a responsive state.
3. **Validate design** Asks each involved Airship component to validate the design. This ensures that the previously committed design is valid at the present time.
4. **Drydock build** Orchestrates the Drydock component to configure hardware and the Kubernetes environment (Drydock -> Promenade)
5. **Armada build** Orchestrates Armada to configure software on the nodes as designed.

3.2.2 update_site

Applies a new committed configuration to the environment. The steps of update_site mirror those of *deploy_site*.

3.2.3 update_software

Triggers an update of the software in a site, using the latest committed configuration documents. Steps, conceptually:

1. **Concurrency check** Prevents concurrent site modifications by conflicting actions/workflows.
2. **Validate design** Asks each involved Airship component to validate the design. This ensures that the previously committed design is valid at the present time.
3. **Armada build** Orchestrates Armada to configure software on the nodes as designed.

3.2.4 redeploy_server

Using parameters to indicate which server(s) triggers a teardown and subsequent deployment of those servers to restore them to the current committed design.

This action is a *target action*, and does not apply the *site action* labels to the revision of documents in Deckhand. Application of site action labels is reserved for site actions such as *deploy_site* and *update_site*.

Like other *target actions* that will use a baremetal or Kubernetes node as a target, the *target_nodes* parameter will be used to list the names of the nodes that will be acted upon.

Using redeploy_server

Danger: At this time, there are no safeguards with regard to the running workload in place before tearing down a server and the result may be *very* disruptive to a working site. Users are cautioned to ensure the server being torn down is not running a critical workload. To support controlling this, the Shipyard service allows actions to be associated with RBAC rules. A deployment of Shipyard can restrict access to this action to help prevent unexpected disaster.

Redeploying a server can have consequences to the running workload as noted above. There are actions that can be taken by a deployment engineer or system administrator before performing a redeploy_server to mitigate the risks and impact.

There are three broad categories of nodes that can be considered in regard to redeploy_server. It is possible that a node is both a Worker and a Control node depending on the deployment of Airship:

1. Broken Node:

   A non-functional node, e.g. a host that has been corrupted to the point of being unable to participate in the Kubernetes cluster.

2. Worker Node:

   A node that is participating in the Kubernetes cluster not running control plane software, but providing capacity for workloads running in the environment.

3. Control Node:

   A node that is participating in the Kubernetes cluster and is hosting control plane software. E.g. Airship or other components that serve as controllers for the rest of the cluster in some way. These nodes may run software such as etcd or databases that contribute to the health of the overall Kubernetes cluster.

   Note that there is also the Genesis host, used to bootstrap the Airship platform. This node currently runs the Airship containers, including some that are not yet able to be migrated to other nodes, e.g. the MAAS rack controller, and disruptions arising from moving PostgreSQL.

Important: Use of redeploy_server on the Airship Genesis host/node is not supported, and will result in serious disruption.

Yes Recommended step for this node type

No Generally not necessary for this node type

N/A Not applicable for this node type

Action	Broken	Worker	Control
Coordinate workload impacts with users*[0]	Yes	Yes	No
Clear Kubernetes labels from the node (for each label)	N/A	Yes	Yes
`$ kubectl label nodes <node> <label>-`			
Etcd - check for cluster health	N/A	N/A	Yes
`$ kubectl -n kube-system exec kubernetes-etcd-<hostname> etcdctl member list`			
Drain Kubernetes node	N/A	Yes	Yes
`$ kubectl drain <node>`			
Disable the kubelet service	N/A	Yes	Yes
`$ systemctl stop kubelet` `$ systemctl disable kubelet`			
Remove node from Kubernetes	Yes	Yes	Yes
`$ kubectl delete node <node>`			
Backup Disks (processes vary)†[0]	Yes	Yes	Yes

3.2.5 relabel_nodes

Using parameters to indicate which server(s), triggers an update to the Kubernetes node labels for those servers.

3.2.6 test_site

Triggers the execution of the Helm tests corresponding to all deployed releases in all namespaces. Steps, conceptually:

1. **Preflight checks** Ensures all Airship components are in a responsive state.
2. **Armada test** Invokes Armada to execute Helm tests for all releases.

Using test_site

The `test_site` action accepts one optional parameter:

1. release: The name of a release to test. When provided, tests are only executed for the specified release.

An example of invoking Helm tests for a single release:

```
shipyard create action test_site --param="namespace=openstack" --param=
↪"release=keystone"
```

3.2.7 update labels

Triggers an update to the Kubernetes node labels for specified server(s)

[0] Of course it is up to the infrastructure operator if they wish to coordinate with their users. This guide assumes client or user communication as a common courtesy.

[0] Server redeployment will (quick) erase all disks during the process, but desired enhancements to redeploy_server may include options for disk handling. Situationally, it may not be necessary to backup disks if the underlying implementation already provides the needed resiliency and redundancy.

CHAPTER 4

Shipyard CLI

4.1 Environment Variables

All commands will utilize the following environment variables to determine necessary information for execution, unless otherwise noted.

4.1.1 OpenStack Keystone Authorization environment variables

The Shipyard CLI/API Client will check for the presence of appropriate environment setup to do authentication on behalf of the user. The openrc variables that will be used are as follows:

- OS_PROJECT_DOMAIN_NAME ("default" if not specified)
- OS_USER_DOMAIN_NAME ("default" if not specified)
- OS_DOMAIN_NAME
- OS_AUTH_TOKEN
- OS_PROJECT_NAME
- OS_USERNAME
- OS_PASSWORD
- OS_AUTH_URL The fully qualified identity endpoint. E.g. http://keystone.ucp.fully.qualified.name:80/v3

4.1.2 OpenStack Keystone Authorization environment variables *not* used

These OpenStack identity variables are not supported by shipyard.

- **OS_IDENTITY_API_VERSION** This value will be ignored as Shipyard only supports version 3 at this time

4.2 Shipyard command options

The base shipyard command supports options that determine cross-CLI behaviors. These options are positioned immediately following the shipyard command as shown here:

```
shipyard <--these options> subcommands...

shipyard
  [--context-marker=<uuid>]
  [--debug/--no-debug]
  [--os-{various}=<value>]
  [--output-format=[format | raw | cli]]  (default = cli)
  [--verbosity=[0-5]  (default = 1)
  <subcommands, as noted in this document>
```

–context-marker=<uuid> 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. If not specified, Shipyard will supply a new UUID to serve as this marker. (optional)

–debug | –no-debug Enable/disable debugging of this CLI and API client. Defaults to no debug

–os-<various>=<value> See supported OpenStack Keystone Authorization Environment variables above for the list of supported names, converting to a downcase version of the environment variable. E.g.: –os-auth-url=http://keystone.ucp:80/v3 If not specified, the environment variables matching these options will be used instead. The Keystone os-auth-url should reference the exposed keystone:port for the target Shipyard environment, as this Keystone will be used to discover the instance of Shipyard. For most invocations other than help, a valid combination of values must be resolved to authenticate and authorize the user's invocation.

–output-format=<format | raw | cli> Specifies the desired output formatting such that:

- **format** Display the raw output from the invoked Shipyard API in a column restricted mode.
- **raw** Display the result from the invoked Shipyard API as-is, without modification.
- **cli (default)** Display results in a plain text interpretation of the response from the invoked Shipyard API.

–verbosity=<0-5> Integer value specifying the level of verbosity for the response information gathered from the API server. Setting a verbosity of `0` will remove all additional information from the response, a verbosity setting of `1` will include summary level notes and information, and `5` will include all available information. This setting does not necessarily effect all of the CLI commands, but may be set on all invocations. A default value of `1` is used if not specified.

4.3 Commit Commands

4.3.1 commit configdocs

Attempts to commit the Shipyard Buffer documents, first invoking validation by downstream components.

```
shipyard commit configdocs
    [--force]
    [--dryrun]

Example:
    shipyard commit configdocs
```

–force Force the commit to occur, even if validations fail.

–dryrun Retrieve validation status for the contents of the buffer without committing.

Sample

```
$ shipyard commit configdocs
Configuration documents committed.
Status: Validations succeeded
Reason: Validation
- Info: DD1001
        Message: Rational Boot Storage: Validation successful.
        Source: Drydock
- Info: DD2002
        Message: IP Locality Check: Validation successful.
        Source: Drydock
- Info: DD2003
        Message: MTU Rationality: Validation successful.
        Source: Drydock
- Info: DD2004
        Message: Network Trunking Rationalty: Validation successful.
        Source: Drydock
- Info: DD2005
        Message: Duplicated IP Check: Validation successful.
        Source: Drydock
- Info: DD3001
        Message: Platform Selection: Validation successful.
        Source: Drydock
- Info: DD1006
        Message: Network Bond Rationality: Validation successful.
        Source: Drydock
- Info: DD2002
        Message: Storage Partitioning: Validation successful.
        Source: Drydock
- Info: DD2003
        Message: Storage Sizing: Validation successful.
        Source: Drydock
- Info: DD1007
        Message: Allowed Network Check: Validation successful.
        Source: Drydock

####  Errors: 0, Warnings: 0, Infos: 10, Other: 0  ####
```

4.4 Control commands

4.4.1 pause, unpause, stop

Three separate commands with a common format that allow for controlling the processing of actions created in Shipyard.

pause pause something in progress e.g. an executing action

unpause unpause something paused e.g. a paused action

stop stops an executing or paused item e.g. an action

```
shipyard pause
    <type>
    <id>

shipyard unpause
    <type>
    <id>

shipyard stop
    <type>
    <id>

shipyard
    pause|unpause|stop
    <qualified name>

Example:

    shipyard pause action 01BTG32JW87G0YKA1K29TKNAFX

    shipyard unpause action 01BTG32JW87G0YKA1K29TKNAFX

    shipyard stop action 01BTG32JW87G0YKA1K29TKNAFX

    shipyard pause action/01BTG32JW87G0YKA1K29TKNAFX
```

<type> The type of entity to take action upon. Currently supports: action

<id> The id of the entity to take action upon.

<qualified name> The qualified name of the item to take the specified action upon

Sample

```
$ shipyard pause action/01BZZMEXAVYGG7BT0BMA3RHYY7
pause successfully submitted for action 01BZZMEXAVYGG7BT0BMA3RHYY7
```

A failed command:

```
$ shipyard pause action/01BZZK07NF04XPC5F4SCTHNPKN
Error: Unable to pause action
Reason: dag_run state must be running, but is failed
- Error: dag_run state must be running, but is failed
```

4.5 Create Commands

4.5.1 create action

Invokes the specified workflow through Shipyard. Returns the id of the action invoked so that it can be queried subsequently.

```
shipyard create action
    <action_command>
```

(continues on next page)

(continued from previous page)

```
    --param=<parameter>    (repeatable)
    [--allow-intermediate-commits]

Example:
    shipyard create action redeploy_server --param="target_nodes=mcp"
    shipyard create action update_site --param="continue-on-fail=true"
```

<action_command> The action to invoke.

–param=<parameter> A parameter to be provided to the action being invoked. (repeatable) Note that we can pass in different information to the create action workflow, i.e. name of server to be redeployed, whether to continue the workflow if there are failures in Drydock, e.g. failed health checks.

–allow-intermediate-commits Allows continuation of a site action, e.g. update_site even when the current committed revision of documents has other prior commits that have not been used as part of a site action.

Sample

```
$ shipyard create action deploy_site
Name                Action                                  Lifecycle
deploy_site         action/01BZZK07NF04XPC5F4SCTHNPKN       None
```

4.5.2 create configdocs

Load documents into the Shipyard Buffer. The use of one or more filenames or one or more directory options must be specified.

```
shipyard create configdocs
    <collection>
    [--append | --replace] [--empty-collection]
    --filename=<filename>    (repeatable)
        |
    --directory=<directory>  (repeatable)

Example:
    shipyard create configdocs design --append --filename=site_design.yaml
```

Note: If neither append nor replace are specified, the Shipyard API default value of rejectoncontents will be used.

Note: –filename and/or –directory must be specified unless –empty-collection is used.

<collection> The collection to load.

–append Add the collection to the Shipyard Buffer. This will fail if the collection already exists.

–replace Clear the shipyard buffer and replace it with the specified contents.

–empty-collection Indicate to Shipyard that the named collection should be made empty (contain no documents). If –empty-collection is specified, the files named by –filename or –directory will be ignored.

–filename=<filename> The file name to use as the contents of the collection. (repeatable) If any documents specified fail basic validation, all of the documents will be rejected. Use of `filename` parameters may not be used in conjunction with the directory parameter.

–directory=<directory> A directory containing documents that will be joined and loaded as a collection. (Repeatable) Any documents that fail basic validation will reject the whole set. Use of the `directory` parameter may not be used with the `filename` parameter.

–recurse Recursively search through all directories for sub-directories that contain yaml files.

Sample

```
$ shipyard create configdocs coll1 --filename=/home/ubuntu/yaml/coll1.yaml
Configuration documents added.
Status: Validations succeeded
Reason: Validation
```

Attempting to load the same collection into the uncommitted buffer.

```
$ shipyard create configdocs coll1 --filename=/home/ubuntu/yaml/coll1.yaml
Error: Invalid collection specified for buffer
Reason: Buffermode : rejectoncontents
- Error: Buffer is either not empty or the collection already exists in buffer.␣
↪Setting a different buffermode may provide the desired functionality
```

Replace the buffer with –replace

```
$ shipyard create configdocs coll1 --replace --filename=/home/ubuntu/yaml/coll1.yaml
Configuration documents added.
Status: Validations succeeded
Reason: Validation
```

4.6 Describe Commands

4.6.1 describe

Retrieves the detailed information about the supplied namespaced item

```
shipyard describe
    <namespaced_item>

Example:
    shipyard describe action/01BTG32JW87G0YKA1K29TKNAFX
      Equivalent to:
    shipyard describe action 01BTG32JW87G0YKA1K29TKNAFX

    shipyard describe notedetails/01BTG32JW87G0YKA1KA9TBNA32
      Equivalent to:
    shipyard describe notedetails 01BTG32JW87G0YKA1KA9TBNA32

    shipyard describe step/01BTG32JW87G0YKA1K29TKNAFX/preflight
      Equivalent to:
    shipyard describe step preflight --action=01BTG32JW87G0YKA1K29TKNAFX
```

(continues on next page)

(continued from previous page)

```
    shipyard describe validation/01BTG32JW87G0YKA1K29TKNAFX/01BTG3PKBS15KCKFZ56XXXBGF2
      Equivalent to:
    shipyard describe validation 01BTG3PKBS15KCKFZ56XXXBGF2 \
        --action=01BTG32JW87G0YKA1K29TKNAFX

    shipyard describe workflow/deploy_site__2017-01-01T12:34:56.123456
      Equivalent to:
    shipyard describe workflow deploy_site__2017-01-01T12:34:56.123456
```

4.6.2 describe action

Retrieves the detailed information about the supplied action id.

```
shipyard describe action
    <action_id>

Example:
    shipyard describe action 01BTG32JW87G0YKA1K29TKNAFX
```

Sample

```
$ shipyard describe action/01BZZK07NF04XPC5F4SCTHNPKN
Name:                   deploy_site
Action:                 action/01BZZK07NF04XPC5F4SCTHNPKN
Lifecycle:              Failed
Parameters:             {}
Datetime:               2017-11-27 20:34:24.610604+00:00
Dag Status:             failed
Context Marker:         71d4112e-8b6d-44e8-9617-d9587231ffba
User:                   shipyard

Steps                                                           Index         State␣
↪    Footnotes
step/01BZZK07NF04XPC5F4SCTHNPKN/action_xcom                     1             ␣
↪success
step/01BZZK07NF04XPC5F4SCTHNPKN/dag_concurrency_check           2             ␣
↪success
step/01BZZK07NF04XPC5F4SCTHNPKN/deckhand_get_design_version     3             ␣
↪failed    (1)
step/01BZZK07NF04XPC5F4SCTHNPKN/validate_site_design            4              None
step/01BZZK07NF04XPC5F4SCTHNPKN/deckhand_get_design_version     5              failed
step/01BZZK07NF04XPC5F4SCTHNPKN/deckhand_get_design_version     6              failed
step/01BZZK07NF04XPC5F4SCTHNPKN/drydock_build                   7              None

Step Footnotes        Note
(1)                   > step metadata: deckhand_get_design_version(2017-11-27␣
↪20:34:34.443053): Unable to determine version
                        - Info available with 'describe notedetails/
↪098765432109876543210987654'

Commands        User            Datetime
invoke          shipyard        2017-11-27 20:34:34.443053+00:00
```

(continues on next page)

(continued from previous page)

```
Validations: None

Action Notes:
> action metadata: 01BZZK07NF04XPC5F4SCTHNPKN(2017-11-27 20:34:24.610604): Invoked␣
↪using revision 3
```

4.6.3 describe notedetails

Retrieves extended information related to a note.

```
shipyard describe notedetails <note_id>

Example:
    shipyard describe notedetails/01BTG32JW87G0YKA1KA9TBNA32
```

<note_id> The id of the note referenced as having more details in a separate response

Sample

```
$ shipyard describe notedetails/01BTG32JW87G0YKA1KA9TBNA32
<a potentially large amount of data as returned by the source of info>
```

4.6.4 describe step

Retrieves the step details associated with an action and step.

```
shipyard describe step
    <step_id>
    --action=<action id>

Example:
    shipyard describe step preflight --action=01BTG32JW87G0YKA1K29TKNAFX
```

<step id> The id of the step found in the describe action response.

–action=<action id> The action id that provides the context for this step.

Sample

```
$ shipyard describe step/01BZZK07NF04XPC5F4SCTHNPKN/action_xcom
Name:              action_xcom
Task ID:           step/01BZZK07NF04XPC5F4SCTHNPKN/action_xcom
Index:             1
State:             success
Start Date:        2017-11-27 20:34:45.604109
End Date:          2017-11-27 20:34:45.818946
Duration:          0.214837
Try Number:        1
Operator:          PythonOperator
```

(continues on next page)

(continued from previous page)

```
Step Notes:
> step metadata: deckhand_get_design_version(2017-11-27 20:34:34.443053): Unable to␣
↪determine version
  - Info available with 'describe notedetails/09876543210987654321098765'
```

4.6.5 describe validation

Retrieves the validation details associated with an action and validation id

```
shipyard describe validation
    <validation_id>
    --action=<action_id>

Example:
    shipyard describe validation 01BTG3PKBS15KCKFZ56XXXBGF2 \
        --action=01BTG32JW87G0YKA1K29TKNAFX
```

<validation_id> The id of the validation found in the describe action response.

–action=<action_id> The action id that provides the context for this validation.

Sample

```
TBD
```

4.6.6 describe workflow

Retrieves the details for a workflow that is running or has run in the workflow engine.

```
shipyard describe workflow
    <workflow_id>

Example:
    shipyard describe workflow deploy_site__2017-01-01T12:34:56.123456
```

<workflow_id> The id of the workflow found in the get workflows response.

Sample

```
$ shipyard describe workflow deploy_site__2017-11-27T20:34:33.000000
Workflow:               deploy_site__2017-11-27T20:34:33.000000
State:                  failed
Dag ID:                 deploy_site
Execution Date:         2017-11-27 20:34:33
Start Date:             2017-11-27 20:34:33.979594
End Date:               None
External Trigger:       True

Steps                           State
action_xcom                     success
```

(continues on next page)

(continued from previous page)

```
dag_concurrency_check              success
deckhand_get_design_version        failed
validate_site_design               None
deckhand_get_design_version        failed
deckhand_get_design_version        failed
drydock_build                      None

Subworkflows:
Workflow:                deploy_site.deckhand_get_design_version__2017-11-27T20:34:33.
↪000000
State:                   failed
Dag ID:                  deploy_site.deckhand_get_design_version
Execution Date:          2017-11-27 20:34:33
Start Date:              2017-11-27 20:35:06.281825
End Date:                None
External Trigger:        False

Workflow:                deploy_site.deckhand_get_design_version.deckhand_get_design_
↪version__2017-11-27T20:34:33.000000
State:                   failed
Dag ID:                  deploy_site.deckhand_get_design_version.deckhand_get_design_
↪version
Execution Date:          2017-11-27 20:34:33
Start Date:              2017-11-27 20:35:20.725506
End Date:                None
External Trigger:        False
```

4.7 Get Commands

4.7.1 get actions

Lists the actions that have been invoked.

```
shipyard get actions
```

Sample

```
$ shipyard get actions
Name               Action                                  Lifecycle       ␣
↪Execution Time             Step Succ/Fail/Oth        Footnotes
deploy_site        action/01BTP9T2WCE1PAJR2DWYXG805V       Failed          2017-09-
↪23T02:42:12        12/1/3                    (1)
update_site        action/01BZZKMW60DV2CJZ858QZ93HRS       Processing      2017-09-
↪23T04:12:21        6/0/10                    (2)

Action Footnotes        Note
(1)                     > action metadata:01BTP9T2WCE1PAJR2DWYXG805V(2017-09-23␣
↪02:42:23.346534): Invoked with revision 3
(2)                     > action metadata:01BZZKMW60DV2CJZ858QZ93HRS(2017-09-23␣
↪04:12:31.465342): Invoked with revision 4
```

4.7.2 get configdocs

Retrieve documents loaded into Shipyard. The possible options include last committed, last site action, last successful site action and retrieval from the Shipyard Buffer. Site actions include `deploy_site`, `update_site` and `update_software`. Note that only one option may be selected when retrieving the documents for a particular collection.

The command will compare the differences between the revisions specified if the collection option is not specified. Note that we can only compare between 2 revisions. The relevant Deckhand revision id will be shown in the output as well.

If both collection and revisions are not specified, the output will show the differences between the 'committed' and 'buffer' revision (default behavior).

```
shipyard get configdocs
    [--collection=<collection>]
    [--committed | --last-site-action | --successful-site-action | --buffer]
    [--cleartext-secrets]

Example:
    shipyard get configdocs --collection=design
    shipyard get configdocs --collection=design --last-site-action
    shipyard get configdocs
    shipyard get configdocs --committed --last-site-action
```

–collection=<collection> The collection to retrieve for viewing. If no collection is entered, the status of the collections in the buffer and those that are committed will be displayed.

–committed Retrieve the documents that have last been committed for this collection

–last-site-action Retrieve the documents associated with the last successful or failed site action for this collection

–successful-site-action Retrieve the documents associated with the last successful site action for this collection

–buffer Retrive the documents that have been loaded into Shipyard since the prior commit. If no documents have been loaded into the buffer for this collection, this will return an empty response (default)

–cleartext-secrets Returns secrets as cleartext for encrypted documents if the user has the appropriate permissions in the target environment. If the user does not have the appropriate permissions and sets this flag to true an error is returned. Only impacts returned documents, not lists of documents.

Sample

```
$ shipyard get configdocs
 Comparing Base: committed (Deckhand revision 2)
         to New: buffer (Deckhand revision 3)
Collection                  Base                  New
coll1                       present               unmodified
coll2                       not present           created
```

```
$ shipyard get configdocs --committed --last-site-action
 Comparing Base: last_site_action (Deckhand revision 2)
         to New: committed (Deckhand revision 2)
Collection                    Base                    New
secrets                       present                 unmodified
design                        present                 unmodified
```

```
$ shipyard get configdocs --collection=coll1
data:
  chart_groups: [kubernetes-proxy, container-networking, dns, kubernetes, kubernetes-
↪rbac]
  release_prefix: ucp
id: 1
metadata:
  layeringDefinition: {abstract: false, layer: site}
  name: cluster-bootstrap-1
  schema: metadata/Document/v1.0
  storagePolicy: cleartext
schema: armada/Manifest/v1.0
status: {bucket: coll1, revision: 1}
```

4.7.3 get renderedconfigdocs

Retrieve the rendered version of documents loaded into Shipyard. Rendered documents are the "final" version of the documents after applying Deckhand layering and substitution.

```
shipyard get renderedconfigdocs
    [--committed | --last-site-action | --successful-site-action | --buffer]
    [--cleartext-secrets]

Example:
    shipyard get renderedconfigdocs
```

–committed Retrieve the documents that have last been committed.

–last-site-action Retrieve the documents associated with the last successful or failed site action.

–successful-site-action Retrieve the documents associated with the last successful site action.

–buffer Retrieve the documents that have been loaded into Shipyard since the prior commit. (default)

–cleartext-secrets Returns secrets as cleartext for encrypted documents if the user has the appropriate permissions in the target environment. If the user does not have the appropriate permissions and sets this flag to true an error is returned.

Sample

```
$ shipyard get renderedconfigdocs
data:
  chart_groups: [kubernetes-proxy, container-networking, dns, kubernetes, kubernetes-
↪rbac]
  release_prefix: ucp
id: 1
metadata:
  layeringDefinition: {abstract: false, layer: site}
  name: cluster-bootstrap-1
  schema: metadata/Document/v1.0
  storagePolicy: cleartext
schema: armada/Manifest/v1.0
status: {bucket: coll1, revision: 1}
```

4.7.4 get workflows

Retrieve workflows that are running or have run in the workflow engine. This includes processes that may not have been started as an action (e.g. scheduled tasks).

```
shipyard get workflows
  [--since=<date>]

Example:
    shipyard get workflows

    shipyard get workflows --since=2017-01-01T12:34:56.123456
```

–since=<date> The historical cutoff date to limit the results of this response.

Sample

```
$ shipyard get workflows
Workflows                                       State
deploy_site__2017-11-27T20:34:33.000000         failed
update_site__2017-11-27T20:45:47.000000         running
```

4.7.5 get site-statuses

Retrieve the provisioning status of nodes and/or power states of the baremetal machines in the site. If no option provided, retrieve records for both status types.

```
shipyard get site-statuses
    [--status-type=<status-type>] (repeatable)
                |

Example:
    shipyard get site-statuses
    shipyard get site-statuses --status-type=nodes-provision-status
    shipyard get site-statuses --status-type=machines-power-state
    shipyard get site-statuses --status-type=nodes-provision-status --status-
↪type=machines-power-state
```

–status-type=<status-type> Retrieve provisioning statuses of all nodes for status-type "nodes-provision-status" and retrieve power states of all baremetal machines in the site for status-type "machines-power-state".

Sample

```
$ shipyard get site-statuses

 Nodes Provision Status:
Hostname            Status
abc.xyz.com         Ready
def.xyz.com         Deploying

 Machines Power State:
Hostname            Power State
```

(continues on next page)

(continued from previous page)

```
abc.xyz.com        On
def.xyz.com        On
```

```
$ shipyard get site-statuses --status-type=nodes-provision-status

 Nodes Provision Status:
Hostname           Status
abc.xyz.com        Ready
def.xyz.com        Deploying
```

```
$ shipyard get site-statuses --status-type=nodes-power-state

 Machines Power State:
Hostname           Power State
abc.xyz.com        On
def.xyz.com        On
```

```
$ shipyard get site-statuses --status-type=nodes-provision-status --status-type=nodes-
↪power-state

 Nodes Provision Status:
Hostname           Status
abc.xyz.com        Ready
def.xyz.com        Deploying

 Machines Power State:
Hostname           Power State
abc.xyz.com        On
def.xyz.com        On
```

4.8 Logs Commands

4.8.1 logs

Retrieves the logs of the supplied namespaced item

```
shipyard logs
    <namespaced_item>

Example:
    shipyard logs step/01BTG32JW87G0YKA1K29TKNAFX/drydock_validate_site_design
      Equivalent to:
    shipyard logs step drydock_validate_site_design --
↪action=01BTG32JW87G0YKA1K29TKNAFX

    shipyard logs step/01BTG32JW87G0YKA1K29TKNAFX/drydock_validate_site_design/2
      Equivalent to:
    shipyard logs step drydock_validate_site_design --
↪action=01BTG32JW87G0YKA1K29TKNAFX --try=2
```

4.8.2 logs step

Retrieves the logs for a particular workflow step. Note that 'try' is an optional parameter.

```
shipyard logs step
    <step_id> --action=<action_name> [--try=<try>]

Example:
    shipyard logs step drydock_validate_site_design --
↪action=01BTG32JW87G0YKA1K29TKNAFX

    shipyard logs step drydock_validate_site_design --
↪action=01BTG32JW87G0YKA1K29TKNAFX --try=2
```

Sample

```
$ shipyard logs step/01C9VVQSCFS7V9QB5GBS3WFVSE/action_xcom
[2018-04-11 07:30:41,945] {{cli.py:374}} INFO - Running on host airflow-worker-0.
↪airflow-worker-discovery.ucp.svc.cluster.local
[2018-04-11 07:30:41,991] {{models.py:1197}} INFO - Dependencies all met for
↪<TaskInstance: deploy_site.action_xcom 2018-04-11 07:30:37 [queued]>
[2018-04-11 07:30:42,001] {{models.py:1197}} INFO - Dependencies all met for
↪<TaskInstance: deploy_site.action_xcom 2018-04-11 07:30:37 [queued]>
[2018-04-11 07:30:42,001] {{models.py:1407}} INFO -
--------------------------------------------------------------------------------
Starting attempt 1 of 1
--------------------------------------------------------------------------------

[2018-04-11 07:30:42,022] {{models.py:1428}} INFO - Executing <Task(PythonOperator):␣
↪action_xcom> on 2018-04-11 07:30:37
[2018-04-11 07:30:42,023] {{base_task_runner.py:115}} INFO - Running: ['bash', '-c',
↪'airflow run deploy_site action_xcom 2018-04-11T07:30:37 --job_id 2 --raw -sd DAGS_
↪FOLDER/deploy_site.py']
[2018-04-11 07:30:42,606] {{base_task_runner.py:98}} INFO - Subtask: [2018-04-11␣
↪07:30:42,606] {{driver.py:120}} INFO - Generating grammar tables from /usr/lib/
↪python3.5/lib2to3/Grammar.txt
[2018-04-11 07:30:42,635] {{base_task_runner.py:98}} INFO - Subtask: [2018-04-11␣
↪07:30:42,634] {{driver.py:120}} INFO - Generating grammar tables from /usr/lib/
↪python3.5/lib2to3/PatternGrammar.txt
[2018-04-11 07:30:43,515] {{base_task_runner.py:98}} INFO - Subtask: [2018-04-11␣
↪07:30:43,515] {{configuration.py:206}} WARNING - section/key [celery/celery_ssl_
↪active] not found in config
[2018-04-11 07:30:43,516] {{base_task_runner.py:98}} INFO - Subtask: [2018-04-11␣
↪07:30:43,515] {{default_celery.py:41}} WARNING - Celery Executor will run without␣
↪SSL
[2018-04-11 07:30:43,517] {{base_task_runner.py:98}} INFO - Subtask: [2018-04-11␣
↪07:30:43,516] {{__init__.py:45}} INFO - Using executor CeleryExecutor
[2018-04-11 07:30:43,822] {{base_task_runner.py:98}} INFO - Subtask: [2018-04-11␣
↪07:30:43,821] {{models.py:189}} INFO - Filling up the DagBag from /usr/local/
↪airflow/dags/deploy_site.py
[2018-04-11 07:30:43,892] {{cli.py:374}} INFO - Running on host airflow-worker-0.
↪airflow-worker-discovery.ucp.svc.cluster.local
[2018-04-11 07:30:43,945] {{base_task_runner.py:98}} INFO - Subtask: [2018-04-11␣
↪07:30:43,944] {{python_operator.py:90}} INFO - Done. Returned value was: None
[2018-04-11 07:30:43,992] {{base_task_runner.py:98}} INFO - Subtask:   """)
```

4.9 Help Commands

4.9.1 help

Provides topical help for shipyard.

Note: –help will provide more specific command help.

```
shipyard help
    [<topic>]

Example:
    shipyard help configdocs
```

<topic> The topic of the help to be displayed. If this parameter is not specified the list of available topics will be displayed.

Sample

```
$ shipyard help
THE SHIPYARD COMMAND
The base shipyard command supports options that determine cross-CLI behaviors.

FORMAT
shipyard [--context-marker=<uuid>] [--os_{various}=<value>]
    [--debug/--no-debug] [--output-format] <subcommands>

Please Note: --os_auth_url is required for every command except shipyard help
     <topic>.

TOPICS
For information of the following topics, run shipyard help <topic>
    actions
    configdocs
```

CHAPTER 5

Site Definition Documents

Shipyard requires some documents to be loaded as part of the site definition for the *deploy_site* and *update_site* as well as other workflows that directly deal with site deployments.

5.1 Schemas

- DeploymentConfiguration schema
- DeploymentStrategy schema

5.2 Deployment Configuration

Allows for specification of configurable options used by the site deployment related workflows, including the timeouts used for various steps, and the name of the Armada manifest that will be used during the deployment/update.

A sample deployment-configuration shows a completely specified example.

Note that the name and schema Shipyard expects the deployment configuration document to have is configurable via the document_info section in the *Shipyard configuration*, but should be left defaulted in most cases.

Default configuration values are provided for most values.

5.2.1 Supported values

- Section: *physical_provisioner*:

  Values in the physical_provisioner section apply to the interactions with Drydock in the various steps taken to deploy or update bare-metal servers and networking.

  deployment_strategy The name of the deployment strategy document to be used. There is a default deployment strategy that is used if this field is not present.

deploy_interval The seconds delayed between checks for progress of the step that performs deployment of servers.

deploy_timeout The maximum seconds allowed for the step that performs deployment of all servers.

destroy_interval The seconds delayed between checks for progress of destroying hardware nodes.

destroy_timeout The maximum seconds allowed for destroying hardware nodes.

join_wait The number of seconds allowed for a node to join the Kubernetes cluster.

prepare_node_interval The seconds delayed between checks for progress of preparing nodes.

prepare_node_timeout The maximum seconds allowed for preparing nodes.

prepare_site_interval The seconds delayed between checks for progress of preparing the site.

prepare_site_timeout The maximum seconds allowed for preparing the site.

verify_interval The seconds delayed between checks for progress of verification.

verify_timeout The maximum seconds allowed for verification by Drydock.

- Section: *kubernetes_provisioner*:

  Values in the kubernetes_provisioner section apply to interactions with Promenade in the various steps of redeploying servers.

  drain_timeout The maximum seconds allowed for draining a node.

  drain_grace_period The seconds provided to Promenade as a grace period for pods to cease.

  clear_labels_timeout The maximum seconds provided to Promenade to clear labels on a node.

  remove_etcd_timeout The maximum seconds provided to Promenade to allow for removing etcd from a node.

  etcd_ready_timeout The maximum seconds allowed for etcd to reach a healthy state after a node is removed.

- Section: *armada*:

  The Armada section provides configuration for the workflow interactions with Armada.

  manifest The name of the Armada manifest document that the workflow will use during site deployment activities. e.g.:'full-site'

5.3 Deployment Strategy

The deployment strategy document is optionally specified in the *Deployment Configuration* and provides a way to group, sequence, and test the deployments of groups of hosts deployed using Drydock. A sample deployment-strategy shows one possible strategy, in the context of the Shipyard unit testing.

5.4 Using A Deployment Strategy

Defining a deployment strategy involves understanding the design of a site, and the desired criticality of the nodes that make up the site.

A typical site may include a handful or many servers that participate in a Kubernetes cluster. Several of the servers may serve as control nodes, while others will handle the workload of the site. During the deployment of a site, it may

be critically important that some servers are operational, while others may have a higher tolerance for misconfigured or failed nodes.

The deployment strategy provides a mechanism to handle defining groups of nodes such that the criticality is reflected in the success criteria.

The name of the DeploymentStrategy document to use is defined in the *Deployment Configuration*, in the `physical_provisioner.deployment_strategy` field. The most simple deployment strategy is used if one is not specified in the *Deployment Configuration* document for the site. Example:

```
schema: shipyard/DeploymentStrategy/v1
metadata:
  schema: metadata/Document/v1
  name: deployment-strategy
  layeringDefinition:
      abstract: false
      layer: global
  storagePolicy: cleartext
data:
  groups: [
    - name: default
      critical: true
      depends_on: []
      selectors: [
        - node_names: []
          node_labels: []
          node_tags: []
          rack_names: []
      ]
      success_criteria:
        percent_successful_nodes: 100
  ]
```

- This default configuration indicates that there are no selectors, meaning that all nodes in the design are included.
- The criticality is set to `true` meaning that the workflow will halt if the success criteria are not met.
- The success criteria indicates that all nodes must be succssful to consider the group a success.

Note that the schema Shipyard expects the deployment strategy document to have is configurable via the document_info section in the *Shipyard configuration*, but should be left defaulted in most cases.

In short, the default behavior is to deploy everything all at once, and halt if there are any failures.

In a large deployment, this could be a problematic strategy as the chance of success in one try goes down as complexity rises. A deployment strategy provides a means to mitigate the unforeseen.

To define a deployment strategy, an example may be helpful, but first definition of the fields follow:

5.4.1 Groups

Groups are named sets of nodes that will be deployed together. The fields of a group are:

name Required. The identifying name of the group.

critical Required. Indicates if this group is required to continue to additional phases of deployment.

depends_on Required, may be an empty list. Group names that must be successful before this group can be processed.

selectors Required, may be an empty list. A list of identifying information to indicate the nodes that are members of this group.

success_criteria Optional. Criteria that must evaluate to be true before a group is considered successfully complete with a phase of deployment.

Criticality

- Field: critical
- Valid values: true | false

Each group is required to indicate true or false for the *critical* field. This drives the behavior after the deployment of baremetal nodes. If any groups that are marked as *critical: true* fail to meet that group's success criteria, the workflow will halt after the deployment of baremetal nodes. A group that cannot be processed due to a parent dependency failing will be considered failed, regardless of the success criteria.

Dependencies

- Field: depends_on
- Valid values: [] or a list of group names

Each group specifies a list of depends_on groups, or an empty list. All identified groups must complete successfully for the phase of deployment before the current group is allowed to be processed by the current phase.

- A failure (based on success criteria) of a group prevents any groups dependent upon the failed group from being attempted.
- Circular dependencies will be rejected as invalid during document validation.
- There is no guarantee of ordering among groups that have their dependencies met. Any group that is ready for deployment based on declared dependencies will execute, however execution of groups is serialized - two groups will not deploy at the same time.

Selectors

- Field: selectors
- Valid values: [] or a list of selectors

The list of selectors indicate the nodes that will be included in a group. Each selector has four available filtering values: node_names, node_tags, node_labels, and rack_names. Each selector is an intersection of this critera, while the list of selectors is a union of the individual selectors.

- Omitting a criterion from a selector, or using empty list means that criterion is ignored.
- Having a completely empty list of selectors, or a selector that has no criteria specified indicates ALL nodes.
- A collection of selectors that results in no nodes being identified will be processed as if 100% of nodes successfully deployed (avoiding division by zero), but would fail the minimum or maximum nodes criteria (still counts as 0 nodes)
- There is no validation against the same node being in multiple groups, however the workflow will not resubmit nodes that have already completed or failed in this deployment to Drydock twice, since it keeps track of each node uniquely. The success or failure of those nodes excluded from submission to Drydock will still be used for the success criteria calculation.

E.g.:

```
selectors:
  - node_names:
      - node01
      - node02
    rack_names:
      - rack01
    node_tags:
      - control
  - node_names:
      - node04
    node_labels:
      - ucp_control_plane: enabled
```

Will indicate (not really SQL, just for illustration):

```
SELECT nodes
WHERE node_name in ('node01', 'node02')
      AND rack_name in ('rack01')
      AND node_tags in ('control')
UNION
SELECT nodes
WHERE node_name in ('node04')
      AND node_label in ('ucp_control_plane: enabled')
```

Success Criteria

- Field: success_criteria
- Valid values: for possible values, see below

Each group optionally contains success criteria which is used to indicate if the deployment of that group is successful. The values that may be specified:

percent_successful_nodes The calculated success rate of nodes completing the deployment phase.

E.g.: 75 would mean that 3 of 4 nodes must complete the phase successfully.

This is useful for groups that have larger numbers of nodes, and do not have critical minimums or are not sensitive to an arbitrary number of nodes not working.

minimum_successful_nodes An integer indicating how many nodes must complete the phase to be considered successful.

maximum_failed_nodes An integer indicating a number of nodes that are allowed to have failed the deployment phase and still consider that group successful.

When no criteria are specified, it means that no checks are done - processing continues as if nothing is wrong.

When more than one criterion is specified, each is evaluated separately - if any fail, the group is considered failed.

5.4.2 Example Deployment Strategy Document

This example shows a contrived deployment strategy with 5 groups: control-nodes, compute-nodes-1, compute-nodes-2, monitoring-nodes, and ntp-node.

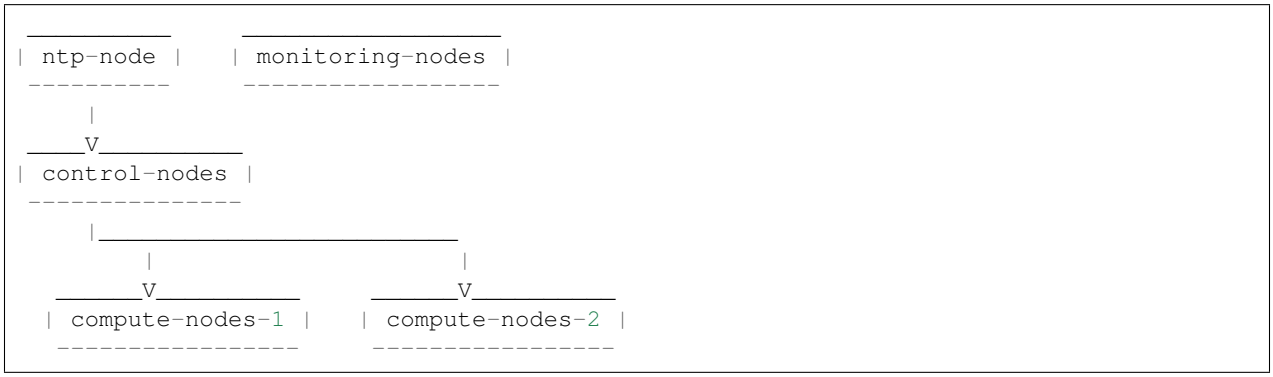
```
---
schema: shipyard/DeploymentStrategy/v1
metadata:
  schema: metadata/Document/v1
  name: deployment-strategy
  layeringDefinition:
      abstract: false
      layer: global
  storagePolicy: cleartext
data:
  groups:
    - name: control-nodes
      critical: true
      depends_on:
        - ntp-node
      selectors:
        - node_names: []
          node_labels: []
          node_tags:
            - control
          rack_names:
            - rack03
      success_criteria:
        percent_successful_nodes: 90
        minimum_successful_nodes: 3
        maximum_failed_nodes: 1
    - name: compute-nodes-1
      critical: false
      depends_on:
        - control-nodes
      selectors:
        - node_names: []
          node_labels: []
          rack_names:
            - rack01
          node_tags:
            - compute
      success_criteria:
        percent_successful_nodes: 50
    - name: compute-nodes-2
      critical: false
      depends_on:
        - control-nodes
      selectors:
        - node_names: []
          node_labels: []
          rack_names:
            - rack02
          node_tags:
            - compute
      success_criteria:
        percent_successful_nodes: 50
    - name: monitoring-nodes
      critical: false
      depends_on: []
      selectors:
        - node_names: []
```

(continues on next page)

(continued from previous page)

```
          node_labels: []
          node_tags:
            - monitoring
          rack_names:
            - rack03
            - rack02
            - rack01
    - name: ntp-node
      critical: true
      depends_on: []
      selectors:
        - node_names:
            - ntp01
          node_labels: []
          node_tags: []
          rack_names: []
      success_criteria:
        minimum_successful_nodes: 1
```

The ordering of groups, as defined by the dependencies (`depends-on` fields):



Given this, the order of execution could be any of the following:

- ntp-node > monitoring-nodes > control-nodes > compute-nodes-1 > compute-nodes-2
- ntp-node > control-nodes > compute-nodes-2 > compute-nodes-1 > monitoring-nodes
- monitoring-nodes > ntp-node > control-nodes > compute-nodes-1 > compute-nodes-2
- and many more ... the only guarantee is that ntp-node will run some time before control-nodes, which will run sometime before both of the compute-nodes. Monitoring-nodes can run at any time.

Also of note are the various combinations of selectors and the varied use of success criteria.

Example Processing

Using the defined deployment strategy in the above example, the following is an example of how it may process:

```
Start
|
| prepare ntp-node           <SUCCESS>
| deploy ntp-node            <SUCCESS>
V
| prepare control-nodes      <SUCCESS>
```

(continues on next page)

(continued from previous page)

```
| deploy control-nodes       <SUCCESS>
V
| prepare monitoring-nodes   <SUCCESS>
| deploy monitoring-nodes    <SUCCESS>
V
| prepare compute-nodes-2    <SUCCESS>
| deploy compute-nodes-2     <SUCCESS>
V
| prepare compute-nodes-1    <SUCCESS>
| deploy compute-nodes-1     <SUCCESS>
|
Finish (success)
```

If there were a failure in preparing the ntp-node, the following would be the result:

```
Start
|
| prepare ntp-node           <FAILED>
| deploy ntp-node            <FAILED, due to prepare failure>
V
| prepare control-nodes      <FAILED, due to dependency>
| deploy control-nodes       <FAILED, due to dependency>
V
| prepare monitoring-nodes   <SUCCESS>
| deploy monitoring-nodes    <SUCCESS>
V
| prepare compute-nodes-2    <FAILED, due to dependency>
| deploy compute-nodes-2     <FAILED, due to dependency>
V
| prepare compute-nodes-1    <FAILED, due to dependency>
| deploy compute-nodes-1     <FAILED, due to dependency>
|
Finish (failed due to critical group failed)
```

If a failure occurred during the deploy of compute-nodes-2, the following would result:

```
Start
|
| prepare ntp-node           <SUCCESS>
| deploy ntp-node            <SUCCESS>
V
| prepare control-nodes      <SUCCESS>
| deploy control-nodes       <SUCCESS>
V
| prepare monitoring-nodes   <SUCCESS>
| deploy monitoring-nodes    <SUCCESS>
V
| prepare compute-nodes-2    <SUCCESS>
| deploy compute-nodes-2     <FAILED, non critical group>
V
| prepare compute-nodes-1    <SUCCESS>
| deploy compute-nodes-1     <SUCCESS>
|
Finish (success with some nodes/groups failed)
```

5.4.3 Important Points

- By default, the deployment strategy is all-at-once, requiring total success.
- Critical group failures halt the deployment activity AFTER processing all nodes, but before proceeding to deployment of the software using Armada.
- Success Criteria are evaluated at the end of processing of each of two phases for each group. A failure in a parent group indicates a failure for child groups - those children will not be processed.
- Group processing is serial.

5.4.4 Interactions

During the processing of nodes, the workflow interacts with Drydock using the node filter mechanism provided in the Drydock API. When formulating the nodes to process in a group, Shipyard will make an inquiry of Drydock's /nodefilter endpoint to get the list of nodes that match the selectors for the group.

Shipyard will keep track of nodes that are actionable for each group using the response from Drydock, as well as prior group inquiries. This means that any nodes processed in a group will not be reprocessed in a later group, but will still count toward that group's success criteria.

Two actions (prepare, deploy) will be invoked against Drydock during the actual node preparation and deployment. The workflow will monitor the tasks created by Drydock and keep track of the successes and failures.

At the end of processing, the workflow step will report the success status for each group and each node. Processing will either stop or continue depending on the success of critical groups.

Example beginning of group processing output from a workflow step:

```
INFO     Setting group control-nodes with None -> Stage.NOT_STARTED
INFO     Group control-nodes selectors have resolved to nodes: node2, node1
INFO     Setting group compute-nodes-1 with None -> Stage.NOT_STARTED
INFO     Group compute-nodes-1 selectors have resolved to nodes: node5, node4
INFO     Setting group compute-nodes-2 with None -> Stage.NOT_STARTED
INFO     Group compute-nodes-2 selectors have resolved to nodes: node7, node8
INFO     Setting group spare-compute-nodes with None -> Stage.NOT_STARTED
INFO     Group spare-compute-nodes selectors have resolved to nodes: node11, node10
INFO     Setting group all-compute-nodes with None -> Stage.NOT_STARTED
INFO     Group all-compute-nodes selectors have resolved to nodes: node11, node7,␣
↪node4, node8, node10, node5
INFO     Setting group monitoring-nodes with None -> Stage.NOT_STARTED
INFO     Group monitoring-nodes selectors have resolved to nodes: node12, node6, node9
INFO     Setting group ntp-node with None -> Stage.NOT_STARTED
INFO     Group ntp-node selectors have resolved to nodes: node3
INFO     There are no cycles detected in the graph
```

Of note is the resolution of groups to a list of nodes. Notice that the nodes in all-compute-nodes node11 overlap the nodes listed as part of other groups. When processing, if all the groups were to be processed before all-compute-nodes, there would be no remaining nodes that are actionable when the workflow tries to process all-compute-nodes. The all-compute-nodes groups would then be evaluated for success criteria immediately against those nodes processed prior. E.g.:

```
INFO     There were no actionable nodes for group all-compute-nodes. It is possible␣
↪that all nodes: [node11, node7, node4, node8, node10, node5] have previously been␣
↪deployed. Group will be immediately checked against its success criteria
INFO     Assessing success criteria for group all-compute-nodes
INFO     Group all-compute-nodes success criteria passed
```

(continues on next page)

(continued from previous page)

```
INFO     Setting group all-compute-nodes with Stage.NOT_STARTED -> Stage.PREPARED
INFO     Group all-compute-nodes has met its success criteria and is now set to stage␣
↪Stage.PREPARED
INFO     Assessing success criteria for group all-compute-nodes
INFO     Group all-compute-nodes success criteria passed
INFO     Setting group all-compute-nodes with Stage.PREPARED -> Stage.DEPLOYED
INFO     Group all-compute-nodes has met its success criteria and is successfully␣
↪deployed (Stage.DEPLOYED)
```

Example summary output from workflow step doing node processing:

```
INFO     =====   Group Summary   =====
INFO       Group monitoring-nodes ended with stage: Stage.DEPLOYED
INFO       Group ntp-node [Critical] ended with stage: Stage.DEPLOYED
INFO       Group control-nodes [Critical] ended with stage: Stage.DEPLOYED
INFO       Group compute-nodes-1 ended with stage: Stage.DEPLOYED
INFO       Group compute-nodes-2 ended with stage: Stage.DEPLOYED
INFO       Group spare-compute-nodes ended with stage: Stage.DEPLOYED
INFO       Group all-compute-nodes ended with stage: Stage.DEPLOYED
INFO     ===== End Group Summary =====
INFO     =====   Node Summary   =====
INFO       Nodes Stage.NOT_STARTED:
INFO       Nodes Stage.PREPARED:
INFO       Nodes Stage.DEPLOYED: node11, node7, node3, node4, node2, node1, node12,␣
↪node8, node9, node6, node10, node5
INFO       Nodes Stage.FAILED:
INFO     ===== End Node Summary =====
INFO     All critical groups have met their success criteria
```

Overall success or failure of workflow step processing based on critical groups meeting or failing their success criteria will be reflected in the same fashion as any other workflow step output from Shipyard.

An Example of CLI *describe action* command output, with failed processing:

```
$ shipyard describe action/01BZZK07NF04XPC5F4SCTHNPKN
Name:                deploy_site
Action:              action/01BZZK07NF04XPC5F4SCTHNPKN
Lifecycle:           Failed
Parameters:          {}
Datetime:            2017-11-27 20:34:24.610604+00:00
Dag Status:          failed
Context Marker:      71d4112e-8b6d-44e8-9617-d9587231ffba
User:                shipyard

Steps                                                    Index        State
step/01BZZK07NF04XPC5F4SCTHNPKN/dag_concurrency_check    1            ␣
↪success
step/01BZZK07NF04XPC5F4SCTHNPKN/validate_site_design     2            ␣
↪success
step/01BZZK07NF04XPC5F4SCTHNPKN/drydock_build            3            failed
step/01BZZK07NF04XPC5F4SCTHNPKN/armada_build             4            None
step/01BZZK07NF04XPC5F4SCTHNPKN/drydock_prepare_site     5            ␣
↪success
step/01BZZK07NF04XPC5F4SCTHNPKN/drydock_nodes            6            failed
```

5.5 Deployment Version

A deployment version document is a Pegleg-generated document that captures information about the repositories used to generate the site defintion. The presence of this document is optional by default, but Shipyard can be *configured* to ensure this document exists, and issue a warning or error if it is absent from a configdocs collection. Document example:

```
---
schema: pegleg/DeploymentData/v1
metadata:
schema: metadata/Document/v1
name: deployment-version
layeringDefinition:
  abstract: false
  layer: global
storagePolicy: cleartext
data:
  documents:
    site-repository:
      commit: 37260deff6a213e30897fc284a993c791336a99d
      tag: master
      dirty: false
    repository-of-secrets:
      commit: 23e7265aee4843301807d649036f8e860fda0cda
      tag: master
      dirty: false
```

Currently, Shipyard does not use this document for anything. Use of this document's data will be added to a future version of Shipyard/Airship.

Note, the name and schema Shipyard expects this document to have can be configured via the document_info section in the *Shipyard configuration*.

CHAPTER 6

Shipyard Client User's Guide

Shipyard provides three methods of interaction:

1. An API - *Shipyard API*
2. An API Client - api-client
3. A Command Line Interface (CLI) - *Shipyard CLI*

Each of these components utilizes the prior.

```
 __________Client__________             __Server__
|                          |           |          |
| CLI -uses-> API Client | -calls-> |   API    |
|__________________________|           |__________|
```

This guide focuses on interaction with Shipyard via the CLI.

6.1 CLI Invocation Flow

It is useful to understand the flow of a request made using the Shipyard CLI first. There are several steps that occur with each invocation. This example will demonstrate the flow of an invocation of the Shipyard CLI.

`Step 1: Invocation:`

```
User --> CLI
e.g.:
  $ shipyard get actions
```

As noted in CLI documentation, Shipyard handles authentication by leveraging OpenStack's Keystone identity service. The CLI provides command line options to specify credentials, or extracts them from the environment. For the example started above, since the credentials are not specified, they would need to be set in the environment prior to invocation. The credentials, regardless of source, are passed from the CLI software to the API Client software.

`Step 2: API Client secures an authentication token:`

```
API Client --> Keystone authentication
                        /
  (Auth Token) <---------
```

Shipyard API Client calls Keystone to acquire an authentication token.

Step 3: API Client discovers Shipyard:

```
API Client --> Keystone service discovery
                        /
  (Shipyard URL) <-------
```

Shipyard API Client calls Keystone to find the URL for the Shipyard API.

Step 4: API Client invokes the appropriate Shipyard API:

```
API Client --> Shipyard API <--> Database, Airflow, etc...
                        /
  (JSON response) <------
```

As noted in the CLI documentation, some responses are YAML instead of JSON.

Step 5: CLI formats response:

```
User <-- (Formatted Response) <-- CLI <-- (JSON response)
e.g.:
  Name               Action                                  Lifecycle
  deploy_site        action/01BZZK07NF04XPC5F4SCTHNPKN       Failed
  update_site        action/01BZZKMW60DV2CJZ858QZ93HRS       Processing
```

The CLI maps the JSON response from the Shipyard API into a more tabular format and presents it to the user.

6.2 Setup

6.2.1 Server Components

Use of the Shipyard client requires that a working installation of the Shipyard API is available. See *Deployment Guide*

6.2.2 Local Environment

Several setup items may be required to allow for an operational Shipyard CLI, including several work-arounds depending on how the Shipyard API is deployed.

Prerequisites:

- Python 3.5+
- git

Note: It is recommended that a virtual environment setup with Python 3.5 is used to contain the dependencies and installation of the Shipyard client.

Retrieve Shipyard client software from git:

```
git clone --depth=1 https://github.com/openstack/airship-shipyard.git
```

Install requirements:

```
sudo apt install python3-pip -y
sudo pip3 install --upgrade pip
cd shipyard
pip3 install -r requirements.txt
```

Build/install Shipyard:

```
python3 setup.py install
```

At this point, invoking shipyard as a command should result in a basic help response:

```
$ shipyard
Usage: shipyard [OPTIONS] COMMAND [ARGS]...

  COMMAND: shipyard

  DESCRIPTION: The base shipyard command supports options that determine
  ...
```

Setup environment variables:

```
export OS_USER_DOMAIN_NAME=default
export OS_PROJECT_DOMAIN_NAME=default
export OS_PROJECT_NAME=service
export OS_USERNAME=shipyard
export OS_PASSWORD=password
export OS_AUTH_URL=http://keystone.ucp:80/v3
```

- The values of these variables should match the credentials and endpoint of the target Shipyard API/Keystone environment.
- The `shipyard` and `password` values are the insecure values used by default if not overridden by the installation of Shipyard.
- The dev_minimal manifest deployment from Airship-in-a-bottle referenced in the deployment guide provides a set of credentials that can be used.

Configure hosts file, if necessary:

```
Add to /etc/hosts:

10.96.0.44   keystone.ucp
10.96.0.44   shipyard-api.ucp.svc.cluster.local
```

- These values would need to be set in the case where DNS resolution of the Keystone and Shipyard URLs is not available.
- The IP addresses should be set to resolve to the IP address of the ingress controller for the target Shipyard API/Keystone environment.
- The value listed as `keystone.ucp` needs to match the value set for OS_AUTH_URL.
- The value listed as `shipyard-api.ucp.svc.cluster.local` needs to match the value that Keystone returns when service lookup is done for the public URL for Shipyard.

6.3 Running Shipyard CLI with Docker Container

It is also possible to execute Shipyard CLI using a docker container.

Note that we will need to pass the relevant environment information as well as the Shipyard command that we wish to execute as part of the `docker run` command. In this example we will execute the `get actions` command:

```
sudo docker run -e 'OS_AUTH_URL=http://keystone-api.ucp.svc.cluster.local:80/v3' \
-e 'OS_PASSWORD=password' -e 'OS_PROJECT_DOMAIN_NAME=default' \
-e 'OS_PROJECT_NAME=service' -e 'OS_USERNAME=shipyard' \
-e 'OS_USER_DOMAIN_NAME=default' -e 'OS_IDENTITY_API_VERSION=3' \
--rm --net=host airshipit/shipyard:latest-ubuntu_focal get actions
```

The output will resemble the following:

```
+ CMD=shipyard
+ PORT=9000
+ '[' get = server ']'
+ exec shipyard get actions
Name               Action                                  Lifecycle
deploy_site        action/01C1Z4HQM8RFG823EQT3EAYE4X       Processing
```

6.4 Use Case: Ingest Site Design

Shipyard serves as the entry point for a deployment of Airship. One can imagine the following activities representing part of the lifecycle of a group of servers for which Airship would serve as the control plane:

Definition A group of servers making up a `site` has been identified. Designs covering the hardware, network, and software are assembled.

Preparation The site is assembled, racking, and wiring is completed, and the hardware is readied for operation. The `Genesis Node` is preinstalled with an (Ubuntu 18.04) image. Airship is deployed; See *Deployment Guide*

At this point, Airship is ready for use. This is when the Shipyard API is available for use.

Load Configuration Documents A user, deployment engineer, or automation – i.e. the operator interacts with Shipyard, perhaps by using the CLI. The operator loads `configdocs` which are a product of the definition step. These `configdocs` are declarative set of YAML documents using a format compatible with Deckhand and containing information usable by the other Airship components.

The interaction with Shipyard could happen as follows:

```
$ git clone --depth=1 https://gitrepo.with.designs/site1.git
```

Note: Assume: /home/user/site1 now contains .yaml files with Drydock, Promenade, Armada, and Divingbell configurations, as well as secrets such as certificates, CAs, and passwords.

Note: Assume: the appropriate credentials are set in the environment

```
$ shipyard create configdocs site1 --directory=/home/user/site1
Configuration documents added.
```

(continues on next page)

(continued from previous page)

```
Status: Validations succeeded
Reason: Validation
```

This loads the documents as a named collection "site1" into Deckhand as a bucket in a revision.

Note: Alternatively, the command could have loaded a single file using –filename=<file>.yaml instead of the –directory option

---

Following the creation of a configdocs collection in the Shipyard buffer, the configdocs must be committed before Shipyard will use those documents as part of an action:

```
$ shipyard commit configdocs
```

During this command, the other Airship components are contacted to validate the designs in Deckhand. If the validations are not successful, Shipyard will not mark the revision as committed.

Important: It is not necessary to load all configuration documents in one step, but each named collection may only exist as a complete set of documents (i.e. must be loaded together).

Important: Shipyard will prevent the loading of two collections into the buffer at the same time unless –append is utilized. This option allows for the loading of multiple collections into the buffer to be later committed together.

An example of this is a base collection that defines some common design elements, a secrets collection that contains certificates, and a site-specific collection that combines with the other two collections to fully define the site.

6.5 Use Case: Deploy Site

Continuing the lifecycle steps from the Ingest Site Design use case, the `operator` proceeds with the deployment of the site.

Deployment The operator creates a deploy_site action and monitors its progress

Maintenance The operator loads new or changed configuration documents (as above), commits them, and creates an `update_site` action

The deployment interactions with Shipyard could happen as follows:

```
$ shipyard create action deploy_site
Name               Action                                  Lifecycle
deploy_site        action/01BZZK07NF04XPC5F4SCTHNPKN       None
```

The deploy_site action is issued to Shipyard which relays a command to the Airflow driven workflow processor. During and following execution of the action, the operator can query the status and results of the action:

```
$ shipyard get actions
Name               Action                                  Lifecycle
deploy_site        action/01BZZK07NF04XPC5F4SCTHNPKN       Processing

$ shipyard describe action/01BZZK07NF04XPC5F4SCTHNPKN
Name:                  deploy_site
```

(continues on next page)

(continued from previous page)

```
Action:                 action/01BZZK07NF04XPC5F4SCTHNPKN
Lifecycle:              Processing
Parameters:             {}
Datetime:               2017-11-27 20:34:24.610604+00:00
Dag Status:             running
Context Marker:         71d4112e-8b6d-44e8-9617-d9587231ffba
User:                   shipyard

Steps                                                                   Index         State
step/01BZZK07NF04XPC5F4SCTHNPKN/action_xcom                             1             ␣
↪success
step/01BZZK07NF04XPC5F4SCTHNPKN/dag_concurrency_check                   2             ␣
↪success
...
```

More information is returned than shown here - for sake of abbreviation. The process of maintenance (update_site) is very similar to the process of deploying a site.

CHAPTER 7

Deployment Guide

Note: Shipyard is still under active development and this guide will evolve along the way

7.1 Deployment

The current deployment makes use of the airship-in-a-bottle project to set up the underlaying Kubernetes infrastructure, container networking (Calico), disk provisioner (Ceph or NFS), and Airship components that are used by Shipyard.

The dev_minimal manifest is the recommended manifest. Please see the README.txt that exists in that manifest's directory.

This approach sets up an 'All-In-One' Airship environment that allows developers to bring up Shipyard and the rest of the Airship components on a single Ubuntu Virtual Machine.

The deployment is fully automated and can take a while to complete. It can take 30 minutes to an hour or more for a full deployment to complete.

7.2 Post Deployment

1. The environment should include the following after executing the required steps:

```
# sudo kubectl get pods -n ucp | grep -v Completed
NAME                                    READY     STATUS    RESTARTS   AGE
airflow-scheduler-79754bfdd5-2wpxn      1/1       Running   0          4m
airflow-web-7679866685-g99qm            1/1       Running   0          4m
airflow-worker-0                        3/3       Running   0          4m
airship-ucp-keystone-memcached-mem...   1/1       Running   0          31m
airship-ucp-rabbitmq-rabbitmq-0         1/1       Running   0          35m
armada-api-5488cbdb99-zjb8n             1/1       Running   0          12m
```

(continues on next page)

(continued from previous page)

```
barbican-api-5fc8f7d6f-s7h7j            1/1       Running   0          11m
deckhand-api-7b476d6c46-qlvtm           1/1       Running   0          8m
drydock-api-5f9fdc858d-lnxvj            1/1       Running   0          1m
ingress-6cd5b89d5d-hzfzj                1/1       Running   0          35m
ingress-error-pages-5c97bb46bb-zqqbx    1/1       Running   0          35m
keystone-api-7657986b8c-6bf92           1/1       Running   0          31m
maas-ingress-66447d7445-mgklj           2/2       Running   0          27m
maas-ingress-errors-8686d56d98-vrjzg    1/1       Running   0          27m
maas-rack-0                             1/1       Running   0          27m
maas-region-0                           2/2       Running   0          27m
mariadb-ingress-6c4f9c76f-lk9ff         1/1       Running   0          35m
mariadb-ingress-6c4f9c76f-ns5kj         1/1       Running   0          35m
mariadb-ingress-error-pages-5dd6fb...   1/1       Running   0          35m
mariadb-server-0                        1/1       Running   0          35m
postgresql-0                            1/1       Running   0          32m
promenade-api-764b765d77-ffhv4          1/1       Running   0          7m
shipyard-api-69888d9f68-8ljfk           1/1       Running   0          4m
```

CHAPTER 8

Sample Policy File

The following is a sample Shipyard policy file for adaptation and use. It is auto-generated from Shipyard when this documentation is built, so if you are having issues with an option, please compare your version of Shipyard with the version of this documentation.

The sample policy file can also be viewed in file form.

```
# Actions requiring admin authority
#"admin_required": "role:admin"

# Rule to deny all access. Used for default denial
#"deny_all": "!"

# List workflow actions invoked by users
# GET  /api/v1.0/actions
#"workflow_orchestrator:list_actions": "rule:admin_required"

# Create a workflow action
# POST  /api/v1.0/actions
#"workflow_orchestrator:create_action": "rule:admin_required"

# Retrieve an action by its id
# GET  /api/v1.0/actions/{action_id}
#"workflow_orchestrator:get_action": "rule:admin_required"

# Retrieve an action step by its id
# GET  /api/v1.0/actions/{action_id}/steps/{step_id}
#"workflow_orchestrator:get_action_step": "rule:admin_required"

# Retrieve logs of an action step by its id
# GET  /api/v1.0/actions/{action_id}/steps/{step_id}/logs
#"workflow_orchestrator:get_action_step_logs": "rule:admin_required"

# Retrieve an action validation by its id
# GET  /api/v1.0/actions/{action_id}/validations/{validation_id}
```

(continues on next page)

(continued from previous page)

```
#"workflow_orchestrator:get_action_validation": "rule:admin_required"

# Send a control to an action
# POST  /api/v1.0/actions/{action_id}/control/{control_verb}
#"workflow_orchestrator:invoke_action_control": "rule:admin_required"

# Retrieve the status of the configdocs
# GET  /api/v1.0/configdocs
#"workflow_orchestrator:get_configdocs_status": "rule:admin_required"

# Ingest configuration documents for the site design
# POST  /api/v1.0/configdocs/{collection_id}
#"workflow_orchestrator:create_configdocs": "rule:admin_required"

# Retrieve a collection of configuration documents with redacted
# secrets
# GET  /api/v1.0/configdocs/{collection_id}
#"workflow_orchestrator:get_configdocs": "rule:admin_required"

# Retrieve a collection of configuration documents with cleartext
# secrets.
# GET  /api/v1.0/configdocs/{collection_id}
#"workflow_orchestrator:get_configdocs_cleartext": "rule:admin_required"

# Move documents from the Shipyard buffer to the committed documents
# POST  /api/v1.0/commitconfigdocs
#"workflow_orchestrator:commit_configdocs": "rule:admin_required"

# Retrieve the configuration documents rendered by Deckhand into a
# complete design
# GET  /api/v1.0/renderedconfigdocs
#"workflow_orchestrator:get_renderedconfigdocs": "rule:admin_required"

# Retrieve the configuration documents with cleartext secrets rendered
# by Deckhand into a complete design
# GET  /api/v1.0/renderedconfigdocs
#"workflow_orchestrator:get_renderedconfigdocs_cleartext": "rule:admin_required"

# Retrieve the list of workflows (DAGs) that have been invoked in
# Airflow, whether via Shipyard or scheduled
# GET  /api/v1.0/workflows
#"workflow_orchestrator:list_workflows": "rule:admin_required"

# Retrieve the detailed information for a workflow (DAG) from Airflow
# GET  /api/v1.0/workflows/{id}
#"workflow_orchestrator:get_workflow": "rule:admin_required"

# Retrieve the details for a note. Further authorization is required
# depending on the topic of the note
# GET  /api/v1.0/notedetails/{note_id}
#"workflow_orchestrator:get_notedetails": "rule:admin_required"

# Retrieve the statuses for the site
# GET  /api/v1.0/site_statuses
#"workflow_orchestrator:get_site_statuses": "rule:admin_required"

# Create a workflow action to deploy the site
```

(continues on next page)

(continued from previous page)

```
# POST  /api/v1.0/actions
#"workflow_orchestrator:action_deploy_site": "rule:admin_required"

# Create a workflow action to update the site
# POST  /api/v1.0/actions
#"workflow_orchestrator:action_update_site": "rule:admin_required"

# Create a workflow action to update the site software
# POST  /api/v1.0/actions
#"workflow_orchestrator:action_update_software": "rule:admin_required"

# Create a workflow action to redeploy target servers
# POST  /api/v1.0/actions
#"workflow_orchestrator:action_redeploy_server": "rule:admin_required"

# Create a workflow action to relabel target nodes
# POST  /api/v1.0/actions
#"workflow_orchestrator:action_relabel_nodes": "rule:admin_required"

# Create a workflow action to invoke Helm tests on all releases or a
# targeted release
# POST  /api/v1.0/actions
#"workflow_orchestrator:action_test_site": "rule:admin_required"
```

CHAPTER 9

Multiple Distro Support

This project builds images for Shipyard and Airflow components. Currently, it supports building images for ubuntu and opensuse (leap 15.1 as base image).

By default, Ubuntu images are built and are published to public registry server. Recently support for publishing opensuse image has also been added.

If you need to build opensuse images locally, the following parameters can be passed to the *make* command in shipyard repository's root directory with *images* as target:

```
DISTRO: opensuse_15
DISTRO_BASE_IMAGE: "opensuse/leap:15.1"
DOCKER_REGISTRY: { your_docker_registry }
IMAGE_TAG: latest
IMAGE_NAME: airflow
PUSH_IMAGE: false
```

Following is an example in command format to build and publish images locally. Command is run in shipyard repository's root directory.

> DISTRO=opensuse_15 DOCKER_REGISTRY={ your_docker_registry } IMAGE_NAME=airflow IMAGE_TAG=latest PUSH_IMAGE=true make images

Following parameters need to be passed as environment/shell variable to make command:

DISTRO parameter to identify distro specific Dockerfile, ubuntu_focal (Default)

DISTRO_BASE_IMAGE parameter to use different base image other than what's used in DISTRO specific Dockerfile (optional)

DOCKER_REGISTRY parameter to specify local/internal docker registry if need to publish image (optional), quay.io (Default)

IMAGE_TAG tag to be used for image built, untagged (Default)

PUSH_IMAGE flag to indicate if images needs to be pushed to a docker registry, false (Default)

This work is done as per approved spec multi_distro_support. Currently only image building logic is enhanced to support multiple distro.

9.1 Adding New Distro Support

To add support for building images for a new distro, following steps can be followed.

1. Shipyard uses images for shipyard and airflow. So to build images for those components, two Dockerfiles are required, one for each component.
2. Add distro specific Dockerfile for each component which will have steps to include necessary packages and run environment configuration. Use existing Dockerfile as sample to identify needed packages and environment information.
3. New dockerfile can be named as Dockefile.{DISTRO} where DISTRO is expected to be distro identifier which is passed to makefile.
4. Respective dockerfile should be placed in {shipyard_root}/images/airflow and {shipyard_root}/images/shipyard
5. Add check, gate, and post jobs for building, testing and publishing images. These entries need to be added in {shipyard_root}/.zuul.yaml file. Please refer to existing zuul file for better existing opensuse support.
6. Add any relevant information to this document.

CHAPTER 10

Building this Documentation

Use `make docs` or `tox -e docs` to generate these docs. This will and build an html version of this documentation that can be viewed using a browser at doc/build/index.html on the local filesystem.