Airship | A New Open Infrastructure Project for OpenStack

Declaratively define your OpenStack & Kubernetes Infrastructure

Airship | A New Open Infrastructure Project for OpenStack

Declaratively define your OpenStack & Kubernetes Infrastructure

Contents

Introduction

Airship is a new Open Infrastructure Project for OpenStack, intended to build on the foundation laid by the OpenStack-Helm project launched in 2017.

We have chosen the name Airship for the nautically themed collection of interoperable open-source tools/services that provide for automated cloud provisioning and management by establishing an under-cloud platform (UCP) leveraging Kubernetes.

The initial focus of this project is the implementation of a declarative platform to introduce OpenStack on Kubernetes (OOK), and the lifecycle management of the resulting cloud, with the scale, speed, resiliency, flexibility and operational predictability demanded of Network Clouds.

Airship – An Undercloud Platform Enabled Network Cloud

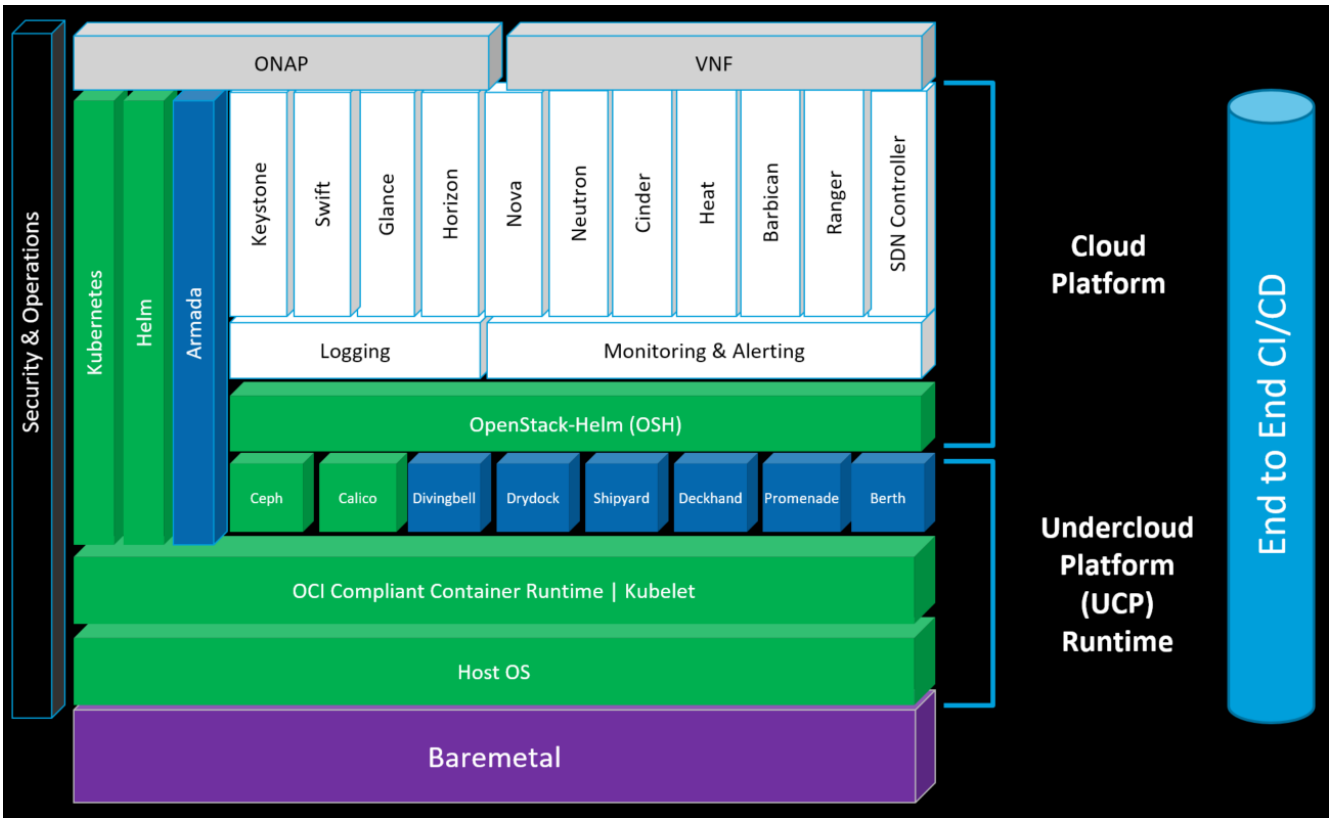

Figure 1 – Software Layers of the AT&T Network Cloud Reference Design

The best way to fully explain what this new Open Infrastructure Project is to explain the layers of AT&T's Network Cloud and the roles the Airship services are performing within the under cloud and cloud platform itself. There are many layers in this design (Figure 1), so we will start at the bottom and work our way up.

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

The **green highlights** the OpenSource projects that Airship leverages/ integrates with to deploy the undercloud platform.

Foundational Software

1. **Host OS / OCI Compliant Container Runtime**

   The Operating System (Linux), and the container runtime, form the foundational bedrocks of enabling containerization in the platform.

2. **Kubernetes**

   Kubernetes serves as our container orchestrator. This brings us a tremendous number of out of the box platform primitives to tap into at the layers above, including rolling updates and resiliency. It is really the combination of both containers themselves and an orchestrator like Kubernetes that makes containerization so powerful.

3. **Helm**

   Helm serves as our way to package, release, and interact with sets of Kubernetes resources we want to install. Probably most simply put, helm is a package manager for Kubernetes. It helps you define, install, and upgrade even the most complex Kubernetes applications using helm charts.

   A chart is a collection of files that describe a related set of Kubernetes resources. Helm wraps up each charts deployment into a concrete release, a tidy little box that is a collection of all the Kubernetes resources that compose that service. So, you can interact with a collection of Kubernetes resources that compose a release as a single unit, either to install, upgrade, or remove.

   At its core, the value that helm brings to the table at least for us, is allowing us to templatize our experience with Kubernetes resources, providing a standard interface for

operators or high-level software orchestrators to control the installation and life cycle of Kubernetes applications.

4. **Calico**

   Project Calico provides a pure Layer3 software-defined-networking framework for the control plane, which can evolve completely independently from our tenant networking. This allows us for the first time to integrate routing announcements for the control plane itself directly into the physical fabric, truly enabling any workload anywhere for the new control plane.

5. **Ceph**

   Ceph is our software-defined-storage (SDS) backend for the control plane. Like Calico, this allows us to separate the control plane storage requirements from the cloud tenants and evolve that independently.

   Ceph also allows the control plane to remain completely self-contained, scales linearly as we add additional control plane hosts, and removes external dependencies such as storage arrays.

   Finally, An SDS solution for the control plane allows Kubernetes to programmatically allocate and decommission persistent storage to containers and move those gracefully as those workloads move between physical hosts.

Airship – The Remaining Layers of the Under Cloud

On top of these foundational open source container and orchestration layers we've collaborated with other OpenStack contributing companies to build an open source software stack to help operators manage their cloud delivery. Each of these software components is designed to perform a particular function well and do no more or less, to keep them well scoped.

We've also designed them where possible to function stand-alone, allowing them to be useful by themselves without having to adopt the entire Airship project ecosystem and architecture (Figure 2). We believe this will help foster external community adoption and participation in these open source projects anywhere there is interest.

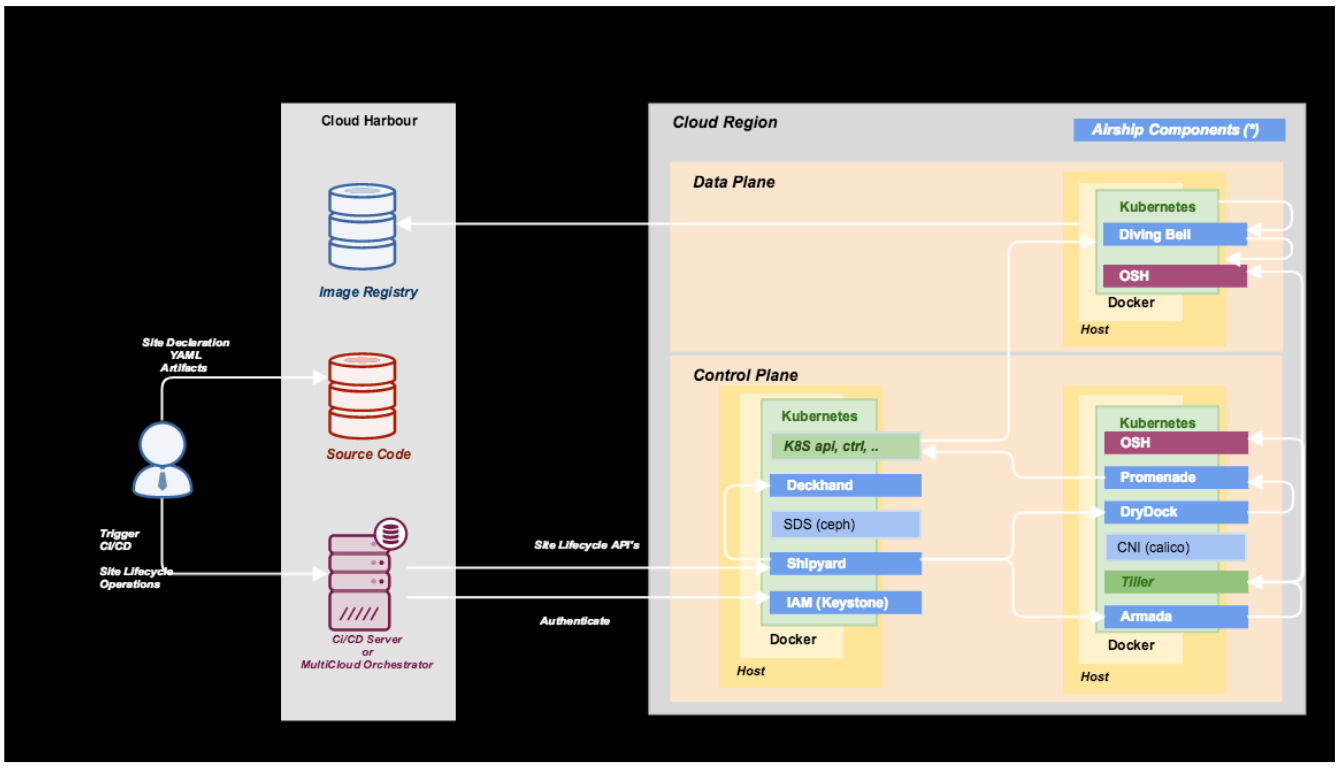

Figure 2 – Airship Software Architecture

1. **Promenade**

   Promenade has been engineered to allow us to deploy a resilient production grade Kubernetes cluster. It supports a desire to manage installs and updates in exactly the same way for every piece of software deployed, including the Airship components themselves.

   This process is called Genesis, during which Promenade uses just a single physical host to manufacture a Kubernetes cluster and all of the required under cloud components to start provisioning the rest of the physical environment ensuring we work with all software components in the cloud exactly the same way during a site buildout, as we will during changes and future augments.

2. **Shipyard**

   Shipyard has been built as the designated REST front door for the platform, allowing CI/CD to deliver YAML based site-designs and updates, facilitating audit requests, and operational actions. Shipyard acts as the site orchestrator, leaning on the other undercloud components to implement the actual request.

3. **Armada**

   Armada is our enterprise answer to orchestrating multiple helm charts using a declarative YAML definition, allowing us to articulate every piece of software we will run in a site in a set of documents we can life cycle.

4. **Drydock**

   Drydock focuses solely on allowing us to provision additional hosts once genesis completes on the first host. Drydock handles both control plane hosts and tenant computes, their operating system, BIOS and firmware updates, RAID configurations, and

complex production networking configurations. It works in conjunction with Promenade to label and join hosts to the Kubernetes cluster, as we are careful to keep this tool focused on what it does, and unaware of Kubernetes, OpenStack or any other service layer.

Today, it focuses on baremetal provisioning, but it was designed with a pluggable backend, so it can evolve to support provisioning infrastructure as virtual machines on top of the cloud itself or even third-party clouds.

5. **Deckhand**

   Deckhand acts as a centralized site-design document storage backend. It keeps revision history of all site-designs and changes, performs secret substitution so we can separate sensitive data such as certificates and passwords from traditional deployment data and allows us to layer documents so that we can re-use common site-design settings across similar site-types.

6. **Divingbell**

   Divingbell is a lightweight solution for 1) Bare metal configuration management for a few very targeted use cases and 2) Bare metal package manager orchestration.

7. **Berth**

   Berth is s a deliberately minimalist VM runner for Kubernetes.

8. **Pegleg**

   Pegleg is a document aggregator that provides early linting and validations via Deckhand for documents that can be consumed by Airship.

9. **Treasure Map:**

   And finally, Treasure Map is the documentation project that outlines a reference architecture for automated cloud provisioning and management, leveraging the Airship interoperable open-source tools.

All of these software components (excluding Treasure Map) are managed exactly the same way we do OpenStack - through the use of Containers, Kubernetes and Helm. Again, effectively, we describe this undercloud as the ideal way to manage and release software quickly, with minimal disruption to workloads, and with absolute predictability.

Other OpenSource Projects Airship Leverages or Integrates with

1. **OpenStack-Helm**

   OpenStack-Helm is an OpenStack project to provide a collection of Helm charts that simply, resiliently, and flexibly deploy OpenStack and related services on Kubernetes.

2. **Barbican**

   Barbican is an OpenStack REST API designed for the secure storage, provisioning and management of secrets.

3. **Keystone**

   Keystone is an OpenStack project that provides authentication, authorization and service discovery mechanisms via HTTP primarily for use by projects in the OpenStack family.

4. **Ironic (future integration)**

   Ironic consists of an API and plug-ins for managing and provisioning physical machines in a security-aware and fault-tolerant manner.

5. **Mogan (future integration)**

   Mogan is an OpenStack project which offers bare metals as first class resources to users, supporting variety of bare metal provisioning drivers including Ironic.

Operations Layer

At the operations layer, the platform bundles many infrastructure needs that are required regardless of the service layer on top, including DNS, NTP, log collection, centralization, and search capabilities, monitoring, metrics, graphing, as well as security policies for both baremetal hosts themselves and the containerized applications.

Service Layer

Finally, at the service layer, services like OpenStack and the local SDN controller, as well as the infrastructure services they depend on such as databases and message queues run on top of the containerized control plane as containerized applications themselves. The undercloud itself remains agnostic of their specific presence, so as we introduce other cloud orchestration technologies as the platform evolves, we can begin supporting those easily with the existing undercloud.

Conclusion - Why Airship?

To evolve how we deliver our cloud platform as well as manage the lifecycle of the software running there – including OpenStack -- we created Airship.

1. **Declarative**

   Sites are declared using YAML. This includes both hard assets such as network configuration and bare metal hosts as well as soft assets like helm charts, their overrides, and container images. You manage the document and Airship implements it.

2. **One Workflow for Life Cycle Management**

   We needed a system that was predictable with life cycle management at its core. This meant ensuring we had one workflow that handled both initial deployments as well as future site updates. In other words, there should be virtually nothing different when interacting with a new deployment or providing an update to an existing site.

3. **Containers as the New and Only Unit of Software Delivery**

   Containers are the unit of software delivery for Airship. Everything is a container. This allows us to progress environments from development, to testing, and finally to production with confidence.

4. **Flexible for Different Architectures and Software**

   Airship is delivering environments both very small and large with a wide range of configurations. We use Airship to manage our entire cloud platform, not just OpenStack.

This whitepaper presents information about a New Open Infrastructure Project for OpenStack that AT&T has been collaborating with other OpenStack contributing companies to create. The code for Airship is available under the Apache 2 license. This information is subject to change without notice to you. No information contained in this whitepaper is or should be interpreted by you as a legal representation, express or implied warranty, agreement, or commitment by AT&T or any of the authors concerning (1) any information or subjects contained in or referenced by this whitepaper, or (2) the furnishing of any products or services by AT&T to you, or (3) the purchase of any products or services by AT&T from you, or (4) any other topic or subject. AT&T may own intellectual property that relates to the information contained in this whitepaper. Notwithstanding anything in this whitepaper to the contrary, no rights or licenses in or to this intellectual property are granted, either expressly or impliedly to you. Rights to AT&T intellectual property may be obtained only by express written agreement with AT&T, signed by AT&T's Chief Technology Officer (CTO) or the CTO's authorized designate.