Gerrit Code Repository Overview

The code for Radio Edge Cloud is primarily derived from the Telco Appliance Blueprint Family and is stored in Gerrit under the "ta" prefix. As with most Linux Foundation projects TA and REC follow the typical usage patterns of the Gerrit code review tool for cloning repos and pushing changes with the "git review" command. The information below provides a detailed overview of the structure of the repositories. The function of the Continuous Integration (CI) system (the LF's Jenkins server) is to use the contents of these repos along with a large amount of upstream binaries (including CentOS, Kubernetes, Docker, Ansible, Disk Image Builder, and others) to build an ISO image of Radio Edge Cloud which will be deployed by Continuous Deployment systems to install and test on bare metal test systems. After successful testing, the exact same ISO can be used to deploy a fully tested and fully reproducible image onto production servers. Much of the code in the repos relates to how to build and configure all the intermediate building blocks.

Overview

This document provides a high-level overview of the initial content of the Akraino REC repositories, at the time of their open sourcing.

The project initially contains around 50 repositories, each repository either holding a self-contained functional component, or a set of components belonging to the same sub-system. Every component belongs to one of these higher-level blocks:

- SCM: code in these repos is used to build a compact AKREC install media from the content of all the repositories
- CaaS: content of these repositories form the Containers-as-a-Service layer, based on Docker and Kubernetes
- L3 Deployer: an Ironic-based hardware manager framework, used to automatically deploy a whole AKREC cluster on a set of hardware
- Middleware services: various value-added platform services to provide cloud infrastructure management services, such as:
    - North-bound REST API
    - CLI interface
    - AAA server to manage cloud infrastructure users, and their roles
    - Configuration manager: a pluggable configuration management framework to centrally manage the configuration of the infrastructure both during initial deployment, and in run-time

Generic repository structure

Even though it can wildly vary what is stored inside the different repositories, there is convention most of the repositories follow inside the project.

SPECS: the output of every repository is one, or more RPM packages, which can be installed on the AKREC CentOS 7.6 operating system. If the repository is packaged into one RPM, its spec file is usually located in the root of the repo. In case of multiple spec files, they are located under SPECS

docker-build: in case a component is deployed as a Docker container, its Docker image is built as part of building the RPM. This directory holds the Dockerfile describing the process of building the specific image

systemd: components deployed directly to the host are deployed as standard, systemd

(ansible/)playbooks, (ansible/)roles: most of the host configuration tasks executed during cloud infrastructure deployment are automated via Ansible playbooks. These directories host the playbooks, and roles related to deploying the specific service inside the repository. Playbooks are invoked by the Deployer component in their appropriate phase: setup, bootstrapping, provisioning, post-configuration, or cleanup.

src: if the component is not 100% coming from another upstream project, but contains its own code; then it is stored under this directory

test(s): contain the unit test cases for the code of the component

Repository content

SCM related repos

ta/build-tools (tree view): build-tools contain the code which is the backbone of the current SCM system. The description of the whole pipeline can be found here, together with the configuration of the image builder tool (DIB: https://github.com/openstack/diskimage-builder)

ta/cloudtaf (tree view): cloudtaf repo contains the ROBOT test case library used to verify the whole AKREC installation

ta/manifest (tree view): manifest repo is the “super repo” of the AKREC project. It contains two major configuration files:

- **yaml** denotes which exact upstream RPM packages need to be installed during disk image creation procedure
- **xml** file collects the AKREC source repositories which need to be installed, and that from which branch, or exact revision of the repo the delivery package needs to be built

ta/rpmbuilder: (tree view) the tool used to build the delivery packages of every repository, that is, an RPM

CaaS (Container as a Service) related repos

ta/caas-cpupooler (tree view): packages, configures, and integrates the components of the upstream CPU-Pooler project (https://github.com/nokia/CPU-Pooler). This component is responsible for providing advanced CPU management policies to both containerized CaaS, and application components

ta/caas-danm (tree view): packages, configures, and integrates the components of all upstream projects related to the network management service. Components are coming from DANM (https://github.com/nokia/danm), Flannel (https://github.com/coreos/flannel), and SR-IOV Device Plugin (https://github.com/intel/sriov-network-device-plugin) repositories.

ta/caas-etcd (tree view): packages, configures, and integrates the components of the upstream Etcd database project (https://github.com/etcd-io/etcd). This is the data backend of the Kubernetes management plane.

ta/caas-helm (tree view): packages, configures, and integrates the components of the upstream Helm project (https://github.com/helm/helm), to provide package management capabilities for containerized applications. Also contains the source code of the AKREC Helm chart repository component.

ta/caas-install (tree view): contains the generic deployment playbooks installing the whole CaaS sub-system during the post-configuration phase. Contains AKREC utility scripts installed to the target operating system under “utils”, and the Helm Chart of the CaaS layer under “infra-charts”. **Note: not all CaaS components are installed via Helm.**

ta/caas-kubedns (tree view): packages, configures, and integrates the components of the upstream Kubernetes DNS project (https://github.com/kubernetes/dns). Kube-DNS backs-up the CaaS in-built service discovery feature.

ta/caas-kubernetes (tree view): packages, configures, and integrates the major components of the Kubernetes management plane (https://github.com/kubernetes/kubernetes). Includes the code related to deploying the API server, scheduler, controller-manager, kube-proxy, and kubelet components.

ta/caas-lcm (tree view): Contains the implementation of the higher-level life-cycle management workflows. The workflows are written in Ansible and are not yet exposed on the AKREC remote API.

ta/caas-logging (tree view): packages, configures, and integrates the major components making up the CaaS log management pipeline. Fluentd (https://github.com/fluent/fluentd) is used to gather and forward the standard output channels of the infrastructure components, and Elasticsearch (https://github.com/elastic/elasticsearch) is the central log store collocating them.

ta/caas-metrics (tree view): packages, configures, and integrates the major components making up the CaaS performance management pipeline. Metrics server (https://github.com/kubernetes-incubator/metrics-server) integrates the container’s core, while Prometheus (https://github.com/prometheus) and custom metrics adapter (https://github.com/kubernetes-incubator/custom-metrics-apiserver) integrates their custom metrics to the Horizontal Pod Autoscaler API.

ta/caas-registry (tree view): packages, configures, and integrates the components responsible for managing container images. Docker Registry (https://github.com/docker/distribution) is used as the front-end, and Swift object store (https://github.com/openstack/swift) is used as the backend component.

ta/caas-security (tree view): contains all the code related to managing users, certificates for TLS, authentication, authorization, and hardening of the CaaS sub-system.

ta/caas-storage (tree view): packages, configures, and integrates the components responsible for configuring the availability of persistent storage for K8s Pods

L3 Deployer related repos

ta/ansible-role-ntp (tree view): configures NTP for the whole cluster during deployment

ta/hw-detector (tree view): responsible for recognizing the specific hardware type the deployment is executed on. Can be used both as a library, or through CLI. Uses IPMI. Contains the hardware specific configuration templates

ta/image-provision (tree view): This project contains dracut modules. They are used for provisioning image to the installation controller’s hard disk from the AKREC boot CD.

ta/infra-ansible (tree view): This repository contains all the generic deployment playbooks, which configure services running directly on the host. Includes playbooks for disk partitioning, Ceph configuration, hardening, security, SSH, operating system level user management etc.

ta/ipa-deployer (tree view): some scripts responsible for bootstrapping the Ironic Python agent on all hosts

ta/ironic-virtmedia-driver (tree view): this project contains Ironic drivers for baremetal provisioning using Virtual media for Quanta Hardware and Virtual environment. The main motivation for writing own drivers is to avoid L2 Network dependency and to support L3 based deployment.

ta/openstack-ansible-XYZ: these projects are re-used from the upstream Openstack-Ansible project (https://github.com/openstack/openstack-ansible) for the purpose of deploying Galera, Keystone, RabbitMQ, and Ironic. These services are used by various middleware, and deployer components.

ta/os-net-config (tree view): contains a fork of the Openstack os-net-config tool (https://github.com/openstack/os-net-config). Used to configure the host network interfaces based on the deployment configuration.

ta/python-ilorest-library (tree view): forked from https://github.com/HewlettPackard/python-ilorest-library. Used to remotely manage the iLO and iLO Chassis Manager based HPE servers.

ta/start-menu (tree view): The installation menu which is used to configure the external IP of the installation controller and starting the installation after the user-config is copied to the installation controller

ta/storage (tree view): this project stores the static disk partitioning configuration for the root disk of all the supported deployment variants'

ta/ironic (tree view): Patched version of openstack ironic that supports setting boot media to floppy. Used for L3 provisioning on certain hardware. (https://github.com/openstack/ironic, https://github.com/rdo-packages/ironic-distgit)

ta/ironicclient (tree view): Patched version of openstack ironicclient that adds the support for floppy.(https://github.com/openstack/python-ironicclient, https://github.com/rdo-packages/ironicclient-distgit)

Middleware services

AAA

ta/access-management (tree view): code of the AAA middleware service lives here

ta/python-peewee (tree view): A small, expressive ORM written in python (https://github.com/coleifer/peewee) - this repository just packages the upstream code into CentOS RPM

CLI

ta/hostcli (tree view): this repo holds the code for the pluggable AKREC CLI framework. The CLI connects to the infra REST API

ta/lockcli (tree view): CLI for global system locks

Config Manager

ta/cm-plugins (tree view): cm plugins store all the existing AKREC plugins slotted into the configuration manager framework. There are four distinct plugin types:

- **validators** are responsible to ensure only semantically correct configuration changes are admitted into the configuration manager server's backend (that is, Redis)
- **userconfighandlers** can mutate the content of a user's configuration change based on domain-specific policies
- **inventoryhandlers** are responsible to create Ansible inventories from the configuration data. Ansible inventories are consumed by the Deployer playbooks
- **activators** are invoked when something was changed in the data of their respective domain. These plugins are responsible for executing run-time changes in the system, based on the submitted config data changes

ta/distributed-state-server (tree view): a service for persistent state management. It is used to store/share the state information between multiple nodes. It uses either etcd or file-based backend. CM uses it to store e.g. the configuration activation state.

ta/config-manager (tree view): the framework of the configuration manager is located in this repo

ta/monitoring (tree view): A set of components related to virtual IP management, database bootstrapping

REST framework

ta/yarf (tree view): "Yet-Another-REST-Framework" is -as the name suggests- the pluggable framework implementing the AKREC north-bound management API

Installation

ta/remote-installer (tree view): install REC from an ISO image