ICN

Overview

ICN BP family intends to address deployment of workloads in a large number of edges and also in public clouds using K8S as resource orchestrator in each site and ONAP-K8S as service level orchestrator (across sites). ICN also intends to integrate infrastructure orchestration which is needed to bring up a site using bare-metal servers. Infrastructure orchestration, which is the focus of this page, needs to ensure that the infrastructure software required on edge servers is installed on per-site basis, but controlled from a central dashboard. Infrastructure orchestration is expected to do the following:

- Installation : First-time installation of all infrastructure software.
  - Keep monitoring for new servers and install the software based on the role of the server machine.
- Patching: Continue to install the patches (mainly security related) if new patch release is made in any one of the infrastructure software packages.
  - May need to work with resource and service orchestrators to ensure that workload functionality does not get impacted.
- Software updates: Updating software due to new releases.

infra-global-controller: If infrastructure provisioning needs to controlled from a central location, this component is expected to be brought up in one location. This controller communicates with infra-local-controller (which is kind of agent) that perform the actual software installation/update/patch and provisions the software or BIOS etc...

infra-local-controller: Typically sits in each site in a bootstrap machine. Typically provided as bootable USB disk. It works in conjunction with the infra-global-controller. Note that, if there is no requirement to manage the software provisioning from a central location, then infra-global-controller is brought up along with the infra-local-controller.

User experience needs to be as simple as possible and even novice user shall be able to set up a site

- 1st-time installation:
  - User procures set of machines or racks.
  - User connects them together with switches.
  - User uses the USB or other mechanisms to boot a machine (call it as bootstrap machine) with infra-local-controller stack.
  - User provides infra-global-controller FQDN/IP address for local controller to reach to it (Note that if global controller is in the same machine as the local controller, loopback IP address can be used to communicate - this is default option).
  - User updates the inventory of machines, defines the role of each machine and inform the central ZTP system (infra-global-controller) or infra-local-controller (bootstrap) stack using its provided UI or via API.
  - System installs the software and verifies the installation of each software component by running tests that are part of ICN-infra.
  - User/Entity gets informed when all the machines are installed with the software.

- Addition of new server
    - User updates the site inventory with its role to the infra global controller using its UI.
    - System expected to install the software.
    - System expected to verify the installation of software using tests that are part of bootstrap stack.
    - User/entity gets informed when the machine is successfully brought online.
- Deletion of existing server:
    - User informs the infra-global-controller to bring down the server.
    - System removes the software and cleans up any local disk and other persistent systems
    - User/Entity gets informed that server can be taken off from the network and disposed of.
- Patching
    - User or external patch system informs the infra-global-controller that new patch(es) is available for a given software package (or packages).
    - User or external patch system also informs whether the patch or patches require a restart of the process or kernel.
    - System then takes care of patching every server that has these software packages.
        - If the software package impacts the workloads:
            - Informs the local workload (resource orchestrator) to not deploy new workloads or move existing workloads. Many resource level orchestrator provides a way to decommission the server on a temporary basis.
            - Ensures that there are no workloads on the system.
            - Installs the patch and do needed restarts.
            - Informs the local orchestrator to put the server back in the pool.
- Updating
    - User or external update system informs the infra-global-controller that a new software version is available for a current software package
    - User or external update system should inform the update is minor or major. A major update required to remove completely the old version, and then install the new version
    - Infra-global-controller based the update nature then takes care of update as follows
        - Reschedule the existing workload in the server that required an update to the other server in the cluster
        - Remove the server from the local orchestrator cluster, provide the update software and reconnects it back

Akraino's "Integrated Cloud Native NFV & App Stack" (ICN) Blueprint is a Cloud Native Compute and Network Framework(CN-CNF) to integrated NFV's application to the de-facto standard and setting a framework to address 5G, IOT and various Linux Foundation edge use case in Cloud Native.

ICN has ONAP as the Service Orchestration Engine(SOE) and the Cloud Native(CN) projects such as Kubernetes for Resource Orchestration Engine (ROE), Prometheus as the monitoring and alerting, OVN as the SDN controller, Container Network Interface(CNI) for Orchestration Networking, provides networking between the clusters, Envoy for Service proxy, Helm and Operators for package management and Rook for storage. The framework stack specifics the best configuration methodology, enables development projects, installation scripts, software package to bind CNCF and LF edge use cases together.

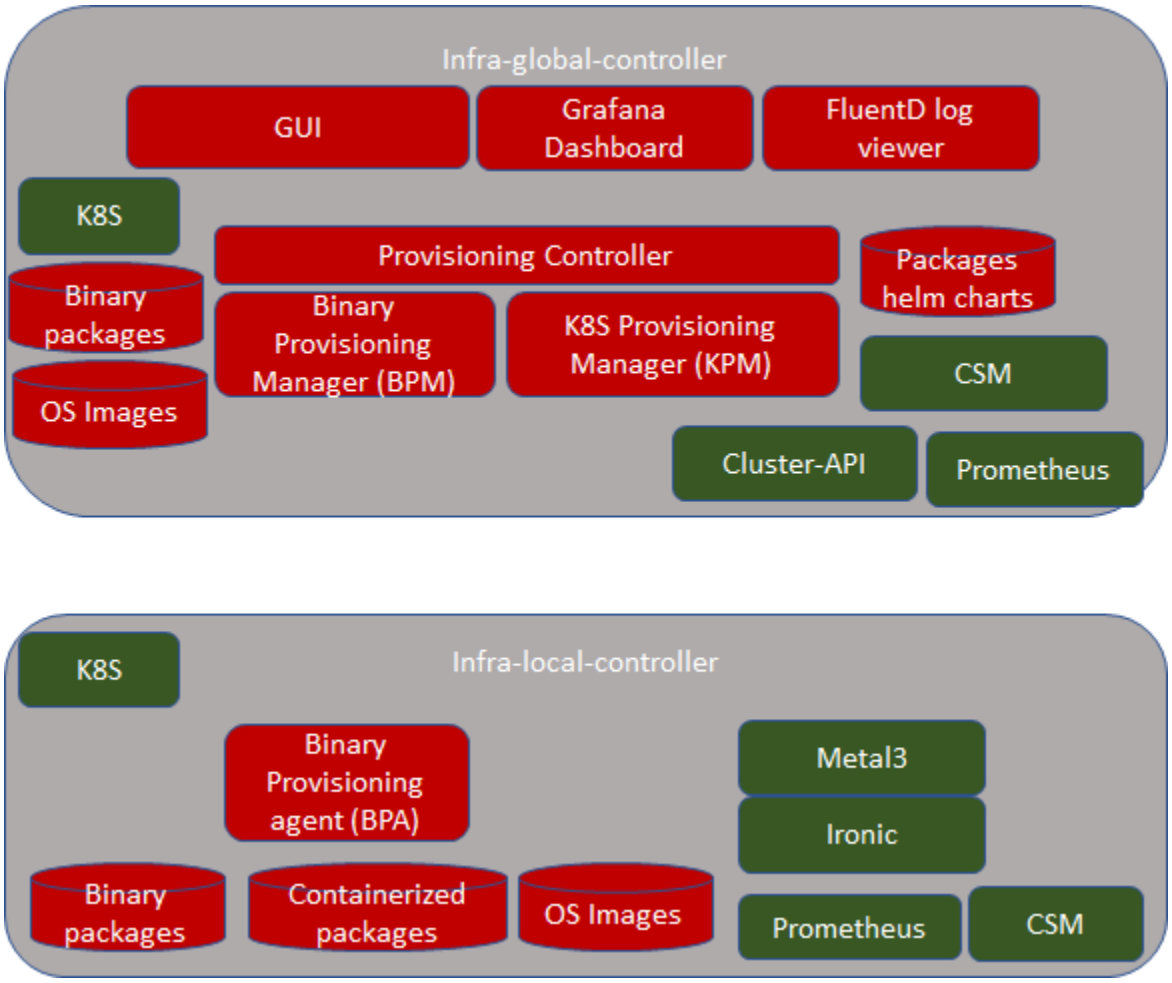
This document break downs the hardware requirements, software ingredient, Testing and benchmarking for the R2 and R3 release for and provides overall picture toward blue print effect in Edge use cases.

Goals

- Generic: Infrastructure Orchestration shall be as generic. Even though this work is being done on behalf of one BP (MICN), infrastructure orchestration shall be common across all BPs in the ICN family. Also, it shall be possible to use this component in other BPs outside of ICN family.
- Leverage open source projects:
    - Leverage cluster-API for infra-global-controller. Identify gaps and provide fixed and also provide UI/CLI for good user experience.
    - Leverage Ironic and metal3 for infra-local-controller to do bare-metal provisioning. Identify any gaps to make it work with Cluster-API.
    - Leverage KuD in infra-local-controller to do Kubernetes installation. Identify any gaps and fix them.
- Figure out ways to use the bootstrap machine also as workload machine (Not in scope for Akraino-R2)
- Flexible and Extensible :
    - Adding any new package in future shall be a simple addition.
    - Interaction with workload orchestrator shall not be limited to K8S. Shall be able to talk to any workload orchestrator.
- Data Model driven:
    - Follow Custom Resource Definition(CRD) models as much as possible.
- Security:
    - Infra-global and infra-local controller may have privileged access to secrets, keys etc.. Shall ensure to protect them by putting them in HW RoT or at least ensure that they are not visible in clear in HDD/SSDs.
- Redundancy: Infra-global controller shall be redundant, especially, if it used to manage multiple sites.
- Performance:
    - Shall be able to complete the first time installation or patching across multiple servers in a site in < 10 minutes for 10 server site. (May need to ensure that jobs are done in parallel - Multi-threading of infra-local-controller).
    - Shall be able to complete the patching across sites shall be done in <10 minutes for 100 sites.

Architecture:

Blocks and Modules



All the green items are existing open source projects. If they require any enhancements, it is best done in the upstream community.

All the red items are expected to be part of the Akraino BP. In some cases, the code in various upstream projects can be leveraged. But, we made them in red color as we don't know at this time to what extent we can use the upstream ASIS. Some guidance

- KPM can be borrowed from Multi-Cloud K8S plugin.
- Some part of provisioning controller (mainly location registration) can be borrowed from Multi-Cloud K8S plugin.
- Provisioning controller might use Tekton or Argo for workflow management.

Since, there are few K8S cluster, let us define them:

- infra-global-controller-K8S : This is the K8S cluster where infra-global-controller related containers are run.
- infra-local-controller-K8S: This is the K8S cluster where the infra-local-controller related containers are run.
- application-K8S : These are K8S clusters where application workloads are run.

Infra-local-controller:

"infra-local-controller" is expected to run in bootstrap machine of each location. Bootstrap is the one which installs the required software in compute nodes used for future workloads. Just an example, say a location has 10 servers. 1 server can be used as bootstrap machine and all other 9 servers can be used compute nodes for running workloads. Bootstrap machine is not only installs all required software in the compute nodes, but also is expected to patch and update compute nodes with newer patched versions of the software.

As you see above in the picture, bootstrap machine itself is based on K8S. Note that this K8S is different from the K8S that gets installed in compute nodes. That is, these are two are different K8S clusters. In case of bootstrap machine, it itself is complete K8S cluster with one node that has both master and minion software combined. All the components of infra-local-controller (such as BPA, Metal3 and Ironic) themselves are containers.

Since we expect infra-local-controller is reachable from outside we expect it to be secured using

- ISTIO and Envoy (for internal communication as well as for external communication)

Infra-local-controller is expected to be brought in two ways:

- As a USB bootable disk: One should be able to get any bare-metal server machine, insert USB and restart the server. What it means is that USB bootable disk shall have basic Linux, K8S and all containers coming up without any user actions. It is also expected to have packages and OS images that are required to provision actual compute nodes. As in above example, these binary, OS and packages are installed on 9 compute nodes.
- As individual entities : As developers, one shall be able to use any machine without inserting USB disk. In this case, developer can choose a machine as bootstrap machine, install Linux OS, Install K8S using Kubeadm and then bring up BPA, Metal3 and Ironic. Then upload packages via RESTAPIs provided by BPA to the system.
- As a KVM/QMEU Virtual machine image: One shall be able to use any VM as bootstrap machine using this image.

Note that infra-local-controller can be run without infra-global-controller. In interim release, we expect that only infra-local-controller is supported. infra-global-controller is targeted for final Akraino R2 release. It is the goal that any operations done in interim release on infra-local-controller manually are automated by infra-global-controller. And hence the interface provided by infra-local-controller is flexible to support both manual actions as well as automated actions.

As indicated above, infra-local-controller is expected to bring K8S cluster on the compute nodes used for workloads. Bringing up workload K8S cluster normally requires the following steps

1. Bring up Linux operating system.
2. Provision the software with right configuration
3. Bring up basic Kubernetes components (such as Kubelet, Docker, kubectl, kubeadm etc..
4. Bring up components that can be installed using kubectl.

Step 1 and 2 are expected to be taken care using Metal3 and Ironic. Step 3 is expected to be taken care by BPA and Step 4 is expected to be taken care by talking to application-K8S

User experience for infrastructure administrators:

When using USB bootable disk

1. Select a machine in the location for bootstrapping.
2. Boot up a bootstrap machine using USB bootable disk.
3. Via Kubectl to infra-local-controller via Metal3 CRs, make ironic ready for compute nodes to do PXEBOOT and install Linux.
4. Upload site specific information via BPA CR - Compute nodes, their roles etc...
5. Once Linux get installed, Via Kuberctl to BPA (via CR), make BPA install the binary packages (such as Kubelet, docker, kubectl, kubenetes API server for application-K8S)
6. Via Kuberctl to BPA, get hold of kubeconfig of application-K8S
7. Using this kubeconfig, via kubectl to application-K8S, install the packages that can be done via kubectl (such as Multus, OVN Controllers, Virtlet etc...)

As a developer:

1. Select a machine in the location for bootstrapping.
2. Install Linux OS
3. Install Kubernetes on this machine using Kubeadm or any of your favorite tool
4. Upload all binary packages, Linux OSes to be installed in compute nodes using for applications.
5. Upload site specific information - Compute nodes, their roles etc...
6. Once Linux get installed, Via Kuberctl to BPA (via CR), make BPA install the binary packages (such as Kubelet, docker, kubectl, kubenetes API server for application-K8S)
7. Via Kuberctl to BPA, get hold of kubeconfig of application-K8S
8. Using this kubeconfig, via kubectl to application-K8S, install the packages that can be done via kubectl (such as Multus, OVN Controllers, Virtlet etc...)
9. Make a USB bootable disk for administrators to use in real deployments.
10. Make a VM image for administrators to use in real deployments.

Binary Provisioning Agent (BPA)

BPA job is to install all packages that can't be installed using kubectl to application-K8S. Hence, BPA is normally used right after compute nodes get installed with Linux operating system, before installing kubernetes based packages. BPA is also an implementation of CRD controller of infra-local-controller-k8s. We expect to have following CRs:

- To upload site specific information - compute nodes and their roles
- To instantiate the binary package installation.
- To get hold of application-K8S kubeconfig file.
- Get status of the installation

BPA also provides some RESTful API for doing the following:

- To upload binary images that are used to install the stuff in compute nodes.
- To upload Linux Operating system that are needed in compute nodes.
- Get status of installation of all packages as prescribed before.

Since compute nodes may not have Internet connectivity

- Ensure that BPA also acts as local Docker Hub repository and BPA needs to ensure that all K8S container packages (that need to be installed on the application-K8S) are served locally here.
- BPA also need to ensure that it configures the docker to get hold of packages from this local repository.

BPA also takes care of: (After interim release)

- When new compute node is added, once the administrator adds new compute node in the site list, it shall take care of installing the packages.
- If a new binary package version is uploaded, it shall take care of figuring out the compute nodes that require this new version and update that compute node with the new version.

BPA is expected to store any private key and secret information in CSM.

- SSH passwords used to authenticate with the compute nodes is expected to be stored in SMS of CSM
- Kuberconfig used to authenticate with application-K8S.

BPA and Ironic related integration:

Ironic is expected to bring up Linux on compute nodes. It is also expected to create SSH keys automatically for each compute node. In addition, it is also expected to create SSH user for each compute node. Usernames and password are expected to be stored in SMS for security reasons in infra-local-controller. BPA is expected to leverage these authentication credentials when it installs the software packages.

CSM is expected to be used not only for storing secrets, but also securely store and perform crypto operations using CSM.

- Use PKCS11
- If TPM is present, Citadel keys are expected to be distributed to TPM and also use TPM for signing operations.

Implementation suggestions:

KuD needs to be broken into pieces

- KuD that installs basic packages via Kubespray and packages that are not containerized. BPA can inherit this code.
- KuD that acts as private docker hub repository. BPA can inherit this code.
- KuD that builds the packages from the source code - this needs to be done outside of BPA and binary packages and container packages that result from these are expected to be part of USB bootable disk.
- KuD that brings containerized packages : This needs to be taken care as a script on top of infra-local-controller.
- CSM (Certificate and Secret management) can be used ASIS. Integration with CSM can be for Akraino-R2 and not for interim release

Infra-global-controller:

There could be multiple edges that need to be brought up. Administrator going to each location, using infra-local-controller to bring up application-K8S cluster in compute nodes of location is not scalable. "infra-global-controller" is expected to provide centralized software provisioning and configuration system. It provides one single-pane-of-glass for administrating the edge locations with respect to infrastructure. Administration involves

- First time bring up.
- Addition of new compute nodes in locations.
- Removal of compute nodes from locations
- Software patching
- Software upgrading

It is expected that infra-local-controller is brought up in each location. infra-local-controller kubeconfig is something that is expected to be made known to the infra-global-controller. Beyond that everything else is taken care by infra-global-controller. infra-global-controller communicates with various infra-local-controllers to do the job of software installation and provisioning.

Infra-global-controller runs in its own K8S cluster. All the components of infra-global-controllers are containers. Following components are part of the infra-global-controller.

- Provisioning controller (PC) Micro Services
- Binary Provisioning Manager (BPM) Micro services
- K8S Provisioning Manager (KPM) Micro-services
- Certificate and Secret management related Micro-services
- Cluster-API related Micro-services
- MongoDB for storing packages and OS images.

Since we expect infra-global-controller is reachable from the Internet, we expect it to be secured using

- ISTIO and Envoy (for internal communication as well as for external communication)
- Store Citadel private keys using CSM.
- Store secrets using SMS of CSM.

Admin user experience:

Assuming that infra-global-controller is brought up with all its micro-services, following steps are expected to be taken up to provision sites/edges.

1. Register infra-local-controllers using infra-local-k8s kubeconfig information and BPA rechability information of each infra-local-controller : This step is required for each infra-local-controller i.e for each location. This will provide information to infra-global-controller to reach K8S API server of infra-local-controller of each location.
2. Upload binary packages (which are binary packages installed in compute nodes by infra-local-controller), helm charts and corresponding container images : This step occurs normally once or when the new versions of the packages.
3. For every site, upload information about compute nodes and their roles.
4. Trigger installation on each site.
5. Monitor the progress
6. Take corrective actions if necessary.
7. Monitor the status of each site on continuous basis. (Expectation is that all K8S clusters - global, local, application - would be installed with Prometheus, Node-exporter and cAdvisor microservices. It is an assumption is that all logs are generated and put in fluentd)
    a. Monitor the status of infra-local-K8S and its node.

b. Monitor the status of application-K8S and compute nodes.
c. Monitor itself
d. Log viewer

Infra-global-controller uses Cluster-API to provision OS related installation in the locations via infra-local-controller.

Following sections describe the components of infra-global-controller.

Provisioning Controller:

It has following functions

Site information :

- It maintains site reachability information - infra-local-controller and BPS reachability information. It provides CRD operator to allow registration of sites.

Inventory information:

- It maintains inventory of compute nodes for each site and the roles of each compute node - It gets this information via CRs.
- It maintains the inventory of software packages and container packages - It gets this information via RESTful API (CRs are not good here as the software packages and container packages are very big).

Application-K8S reachability information:

- It gets this information from BPM and stores it locally. This is used by PC and KPM at later time to install containerized packages in application-K8S.

Software and configuration of site:

- Upon invocation of install, patch or update, it goes through series of steps
  - It uploads the OS images, binary packages to the site (whatever is needed and if the versions are different from the site)
  - It communicates with Cluster-API to talk to infra-local-controller Metal3 to prepare it for installation of basic operating system and utilities using Linux image. As part of this, it converts its local inventory information to CRs expected by Cluster-API.
  - Keeps monitoring to ensure the compute nodes are installed with the OS and utilities.
  - Then it communicates with BPM to talk to BPA to install binary packages.
  - Once BPM provides status that all packages are installed successfully, it gets the kubeconfig of application-K8S.
  - Then it invokes KPM to instantiate containerized packages.
  - Once KPM indicates that all containerized packages are ready/complete, then it provides appropriate status to the user by updating the CR status.

Implementation notes:

- Site registration code can be borrowed from the ONAP K8S plugin service.
- New CRD controller is expected to be created with following CRs
  - Site registration related CRs.
  - Compute inventory related CRs.
  - Site install trigger related CRs.
- Expected to provide APIs
  - For uploading binary packages
  - For uploading containerized packages
  - For uploading OS images
  - Each package, OS image or containerized package is supposed to have right meta data information for identification at later time.

Binary Provisioning Manager (BPM)

It has following functions

Binary package installation and configuration management : This functionality via RESTful API is called by PC to trigger the installation. It then internally calls BPA of infra-local-controller to initiate the installation and configuration process. It triggers BPA via infra-local-K8S BPA CRs.

Binary package distribution : This functionality via RESTful API is called by PC. It figures out the differences between the binary packages & container packages it has locally for this location with the packages that are already in the BPA. Any differences are uploaded to BPA via BPA provided RESTful API.

Collection of KubeConfig of application-K8S : This functionality gets the KubeConfig of application-K8S from BPA. This gets stored in the database table that is specific to site.

K8S Provisioning Manager (KPM)

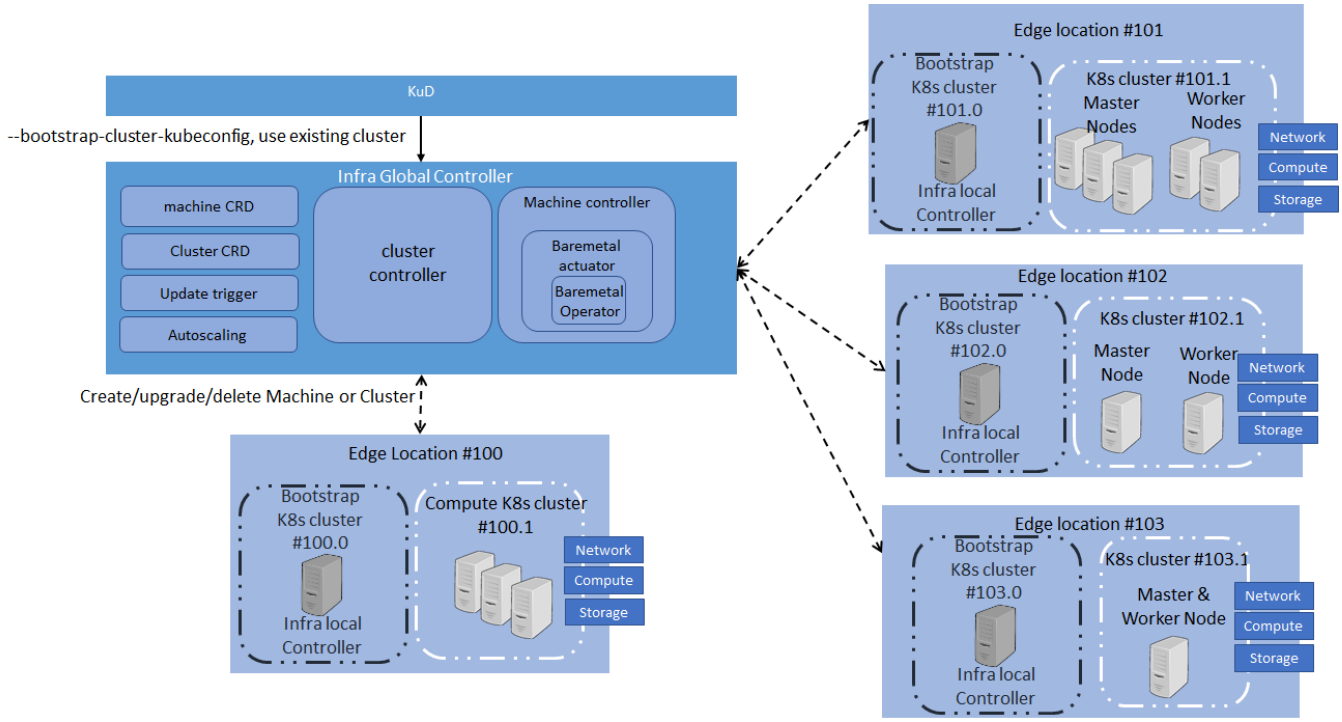
KPM is used to install containerized packages on application-K8S. KPM looks at all the relevant helm charts and instantiates them by talking to application-K8S.

Implementation details:

Code can be borrowed from the ONAP Multi-Cloud K8S plugin service which does similar functionality.

Design Details(WIP)

Note : ZTP (Zero Touch Provisioning) term is used in the BP presentation. This represents both infra-local-controller and infra-global-controller.



infra-local-controller

As shown in the above figure, the infra local controller is itself a Bootstrap K8s cluster, that brings up the compute k8s cluster in the edge location. Infra-local controller has BPA, Metal3, Baremetal operator(Ironic). This section explains the details of it.

Metal3 & Ironic:

This subsection is referred from https://github.com/metal3-io/metal3-docs/blob/master/design/nodes-machines-and-hosts.md

Baremetal operator provides hardware provisioning of compute nodes by using the kubernetes API. The Baremetal operator defines a CRD BaremetalHost Object represents a physical server, it represents several hardware inventories. Ironic is responsible for provisioning the physical servers, and the Baremetal Operator is for responsible for wrapping the Ironic and represents them as CRD object.

BPA (Define CRD, example CRs, RESTful API)

KuD Changes (Describe how KuD works today and what specific changes would be required)

Metal3 & Ironic

Sequence Diagrams involving all of above + CSM + Logging + Monitoring stuff

Infra-global-controller

PC (Define CRD, Restful API and the example CRs and example API requests)

BPM

KPM

Cluster-API

Global ZTP:

Global ZTP system is used for Infrastructure provisioning and configuration in ICN family. It is subdivided into 3 deployments Cluster-API, KuD and ONAP on K8s.

Cluster-API & Baremetal Operator

One of the major challenges to cloud admin managing multiple clusters in different edge location is coordinate control plane of each cluster configuration remotely, managing patches and updates/upgrades across multiple machines. Cluster-API provides declarative APIs to represent clusters and machines inside a cluster. Cluster-API provides the abstraction for various common logic that can be seen in various cluster provider such as GKE, AWS, Vsphere. Cluster-API consolidated all those logic provide abstractions for all those logic functions such as grouping machines for the upgrade, autoscaling mechanism.

In ICN family stack, Cluster-API Baremetal provider is metal3 Baremetal Operator, it is used as a machine actuator that uses Ironic to provide k8s API to manage the physical servers that also run Kubernetes clusters on bare metal host. Cluster-API manages the kubernetes control plane through cluster CRD, and Kubernetes node(host machine) through machine CRDs, Machineset CRDs and MachineDeployment CRDS. It also has an autoscaler mechanism that checks the Machineset CRD that is similar to the analogy of K8s replica set and MachineDeployment CRD similar to the analogy of K8s Deployment. MachineDeployment CRDs are used to update/upgrade of software drivers in

Cluster-API provider with Baremetal operator is used to provision physical server, and initiate the kubernetes cluster with user configuration

KuD

Kubernetes deployer(KUD) in ONAP can be reused to deploy the K8s App components(as shown in fig. II), NFV Specific components and NFVi SDN controller in the edge cluster. In R2 release KuD will be used to deploy the K8s addon such as Prometheus, Rook, Virlet, OVN, NFD, and Intel device plugins in the edge location(as shown in figure I). In R3 release, KuD will be evolved as "ICN Operator" to install all K8s addons.

ONAP on K8s

One of the Kubernetes clusters with high availability, which is provisioned and configured by Cluster-API will be used to deploy ONAP on K8s. ICN family uses ONAP Operations Manager(OOM) to deploy ONAP installation. OOM provides a set of helm chart to be used to install ONAP on a K8s cluster. ICN family will create OOM installation and automate the ONAP installation once a kubernetes cluster is configured by cluster-API

ONAP Block and Modules:

ONAP will be the Service Orchestration Engine in ICN family and is responsible for the VNF life cycle management, tenant management and Tenant resource quota allocation and managing Resource Orchestration engine(ROE) to schedule VNF workloads with Multi-site scheduler awareness and Hardware Platform abstraction(HPA). Required an Akraino dashboard that sits on the top of ONAP to deploy the VNFs

Kubernetes Block and Modules:

Kubernetes will be the Resource Orchestration Engine in ICN family to manage Network, Storage and Compute resource for the VNF application. ICN family will be using multiple container runtimes as Virtlet, Kata container, Kubevirt and gVisor. Each release supports different container runtimes that are focused on use cases.

Kubernetes module is divided into 3 groups - K8s App components, NFV specific components and NFVi SDN controller components, all these components will be installed using KuD addons

K8s App components: This block has k8s storage plugins, container runtime, OVN for networking, Service proxy and Prometheus for monitoring, and responsible application management

NFV Specific components: This block is responsible for k8s compute management to support both software and hardware acceleration(include network acceleration) with CPU pinning and Device plugins such as QAT, FPGA, SRIOV & GPU.

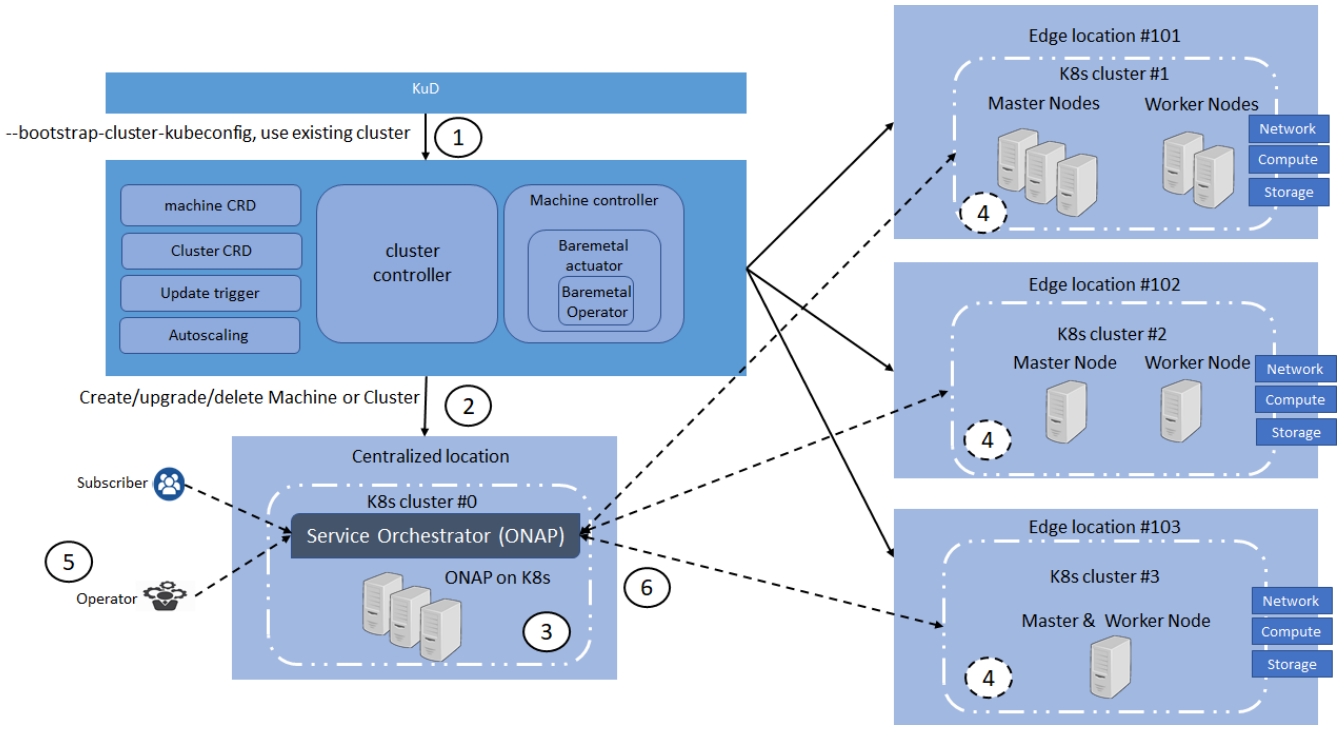
SDN Controller components: This block is responsible for managing SDN controller and to provide additional features such as Service Function chaining (SFC) and Network Route manager.

Apps/ Use cases:

- **SDWAN usecase**
- **Distributed Analytics as a Service**
- **EdgeXFoundry use case**
- **VR 360 streaming**

ICN Infrastructure layout

Flows & Sequence Diagrams



1. **Use Clusterctl command to create the cluster for the cluster-api-provider-baremetal provider. For this step, we required KuD to provide a cluster and run the machine controller and cluster controller**
2. **Users Machine CRD and Cluster CRD in configured to instated 4 clusters as #0, #1, #2, #3**
3. **Automation script for OOM deployment is trigged to deploy ONAP on cluster #0**
4. **KuD addons script in trigger in all edge location to deploy K8s App components, NFV Specific and NFVi SDN controller**
5. **Subscriber or Operator requires to deploy the VNF workload such as SDWAN in Service Orchestration**
6. **ONAP should place the workload in the edge location based on Multi-site scheduling and K8s HPA**

Installation demonstration

ICN EMCO vFW Demo.webm

Software components

fComponents	Link	Akraino Release target
Cluster-API	https://github.com/kubernetes-sigs/cluster-api - 0.1.0	R2
Cluster-API-Provider-bare metal	https://github.com/metal3-io/cluster-api-provider-baremetal	R2

Provision stack - Metal3	https://github.com/metal3-io/baremetal-operator/	R2
Host Operating system	Ubuntu 18.04	R2
Quick Access Technology (QAT) drivers	Intel® C627 Chipset - https://ark.intel.com/content/www/us/en/ark/products/97343/intel-c627-chipset.html	R2
NIC drivers	XL710 - https://www.intel.com/content/dam/www/public/us/en/documents/datasheets/xl710-10-40-controller-datasheet.pdf	R2
ONAP	Latest release 3.0.1-ONAP - https://github.com/onap/integration/	R2
Workloads	• OpenWRT SDWAN - https://openwrt.org/ • Distributed Analytics as a Service • EdgeXFoundry use case • VR 360 streaming	**R3**
KUD	https://git.onap.org/multicloud/k8s/	R2
Kubespray	https://github.com/kubernetes-sigs/kubespray	R2
K8s	https://github.com/kubernetes/kubeadm - v1.15	R2
Docker	https://github.com/docker - 18.09	R2
Virtlet	https://github.com/Mirantis/virtlet -1.4.4	R2
SDN - OVN	https://github.com/ovn-org/ovn-kubernetes - 0.3.0	R2
OpenvSwitch	https://github.com/openvswitch/ovs - 2.10.1	R2
Ansible	https://github.com/ansible/ansible - 2.7.10	R2
Helm	https://github.com/helm/helm - 2.9.1	R2
Istio	https://github.com/istio/istio - 1.0.3	R2
Kata container	https://github.com/kata-containers/runtime/releases - 1.4.0	**R3**
Kubevirt	https://github.com/kubevirt/kubevirt/ - v0.18.0	R3
Collectd	https://github.com/collectd/collectd	R2
Rook/Ceph	https://rook.io/docs/rook/v1.0/helm-operator.html v1.0	R2
MetalLB	https://github.com/danderson/metallb/releases - v0.7.3	**R3**
Kube - Prometheus	https://github.com/coreos/kube-prometheus - v0.1.0	R2
OpenNESS	Will be updated soon	R3
Multi-tenancy	https://github.com/kubernetes-sigs/multi-tenancy	R2
Knative	https://github.com/knative	**R3**
Device Plugins	https://github.com/intel/intel-device-plugins-for-kubernetes - QAT, SRIOV	R2
	https://github.com/intel/intel-device-plugins-for-kubernetes - FPGA, GPU	**R3**
Node Feature Discovery	https://github.com/kubernetes-sigs/node-feature-discovery -	R2
CNI	https://github.com/coreos/flannel/ - release tag v0.11.0 https://github.com/containernetworking/cni - release tag v0.7.0 https://github.com/containernetworking/plugins - release tag v0.8.1 https://github.com/containernetworking/cni#3rd-party-plugins - Multus v3.3tp, SRIOV CNI v2.0(withSRIOV Network Device plugin)	R2
Conformance Test for K8s	https://github.com/heptio/sonobuoy	R2

Gaps(WIP)

Release	Block	Components	Identified Gaps	Initial thought

R2	ZTP	Cluster-API	The cluster upgrade yet to be support	The definition of "cluster upgrade" and expected behaviour should be documented here. For example cluster upgrade could be kubelet version upgrade.
			No node repair mechanism	Node logs such kubelet logs should be enable in the automation script
			No Multi-Master support	Required to confirm from engineers
	KuD	Virtlet , Multus, NFD & Istio	Installation script are in ansible and static. Required to be in daemonset	
		Virtlet & Intel Device plugin	Have to check with Virtlet support with device plugin framework	
	ONAP	OOM automation	Portal chart is deployed with loadbalancer with floating IP address	
		Dashboard	Monitoring tool to check the deployment across the multi site and show the metrics/statistics details to the operator	
R3	APP use cases	SDWAN	OpenWRT is potential candidate to configured SDWAN use case. Required more information on it	

Roadmap

August Intermediate release

Timeline	Release	required state of implementation	Expected Result
Aug 2nd	ICN-v0.1.0	• ICN Bootloader • Metal3 Baremetal operator • KUD provisioning	• ISO bootloader script • Installation script for the bootstrap cluster • Installation Script for compute cluster with KuD • Will be in dev branch • If the deadline is missed, Aug7th is the extended last deadline

Aug 9th	ICN-v0.1.1	• KUD addon plugin integration
		  ◦ Multus
		  ◦ NFD
		  ◦ SRIOV
		  ◦ Virtlet
		  ◦ QAT
		  ◦ ROOK
		• Initial integration of BPA controller and BPA RestAPI
			• Testing and integration of KUD plugins
			• Bug fix from the previous release
			• Will be in dev branch
			• If the deadline is missed, Aug 14th is the extended last deadline

Aug 16th	ICN-v0.2.0	• ICN Bootloader • Metal3 Baremetal operator • KUD provisioning • KUD add-plugins • BPA controller • BPA RestAPI	• Merged with Master • All Integration must be completed by Aug 16th.

Akraino R2 release

Components	required state of implementation	Expected Result
ZTP	• Cluster-API is integrated with Baremetal operator to instantiate 2 cluster #0, #1, #2 & #3 • Cluster #0 should have at least 3 machine for ONAP • Cluster #1 should have at least 5 machine for Edge location • Cluster #2 should have at least 2 machine for Edge location • Cluster #3 should have at least 1 machine for Edge location	All-in-one ZTP script with cluster-API and Baremetal operator
ONAP	• Consolidated OOM Helm chart and script to install ONAP in the cluster#0 • With ONAP Dashboard	Should be integrated with the above script

KuD addons	• Rook, Prometheus, SRIOV, QAT, Collectd, OVN, SFC Manager	Daemonset yaml should be integrated with the above script
Tenant Manager	• Create Tenant and resource quota	should be deployed as part of KuD addons
Dashboard	• Akraino Dashboard integrated with ONAP monitoring agent	Dashboard run as deployment in ONAP cluster
App	• Dummy 3 ubuntu instances to recreate SDWAN use case with Multiple networks, QAT, SRIOV and static SFC	Instantiate 3 workloads from ONAP to show the SFC functionality in Dashboard
CI	• Integrated and tested with conformance testing	End-to-End testing script

Akraino R3 release

Components	required state of implementation	Expected Result
ZTP	• Integrated with openNESS components and EdgeXFoundry	All-in-one ZTP script with cluster-API and Baremetal operator
ONAP	• Consolidated OOM Helm chart and script to install ONAP in the cluster#0 • R5/R6 ONAP release will have k8s HPA & Multi-site scheduler	Should be integrated with the above script
KuD addons	• Kata container, KubeVirt, MetallB	Daemonset yaml should be integrated with the above script
Dashboard	• Akraino Dashboard integrated with ONAP monitoring agent	Dashboard run as deployment in ONAP cluster
App	• Run vFW, vIPS, vSDWAN with Multiple networks, QAT, SRIOV, IPsec tunnel and dynamic SFC • Run the Distributed Analytics as a Service	Instantiate 3 workloads from ONAP to show the SFC functionality in Dashboard
CI	• Integrated and tested with conformance testing	End-to-End testing script

Future releases

Yet to discuss