ICN R4 Architecture Document

Introduction

The ICN blueprint 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 EMCO 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 a 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.

The user experience needs to be as simple as possible and even a novice user should be able to set up a site.

Use Cases

1. SDEWAN Controller with Open source based SDWAN CNF and SDEWAN HUB to establish IPSEC tunneling between Edge Distributions with Service Function Chaining(SFC)
2. Composite vFirewall to show case Telco, Cable Use cases using EMCO(Edge Multi-Cluster Orchestration)

Where on the Edge

Nowadays best efforts are put to keep the Cloud native control plane close to workload to reduce latency, increase performance, and fault tolerance. A single orchestration engine to be lightweight and maintain the resources in a cluster of compute node, Where the customer can deploy multiple Network Functions, such as VNF, CNF, Micro service, Function as a service (FaaS), and also scale the orchestration infrastructure depending upon the customer demand.

ICN target on-prem edge, 5G, IoT, SDWAN, Video streaming, Edge Gaming Cloud. A single deployment model to target multiple edge use case.

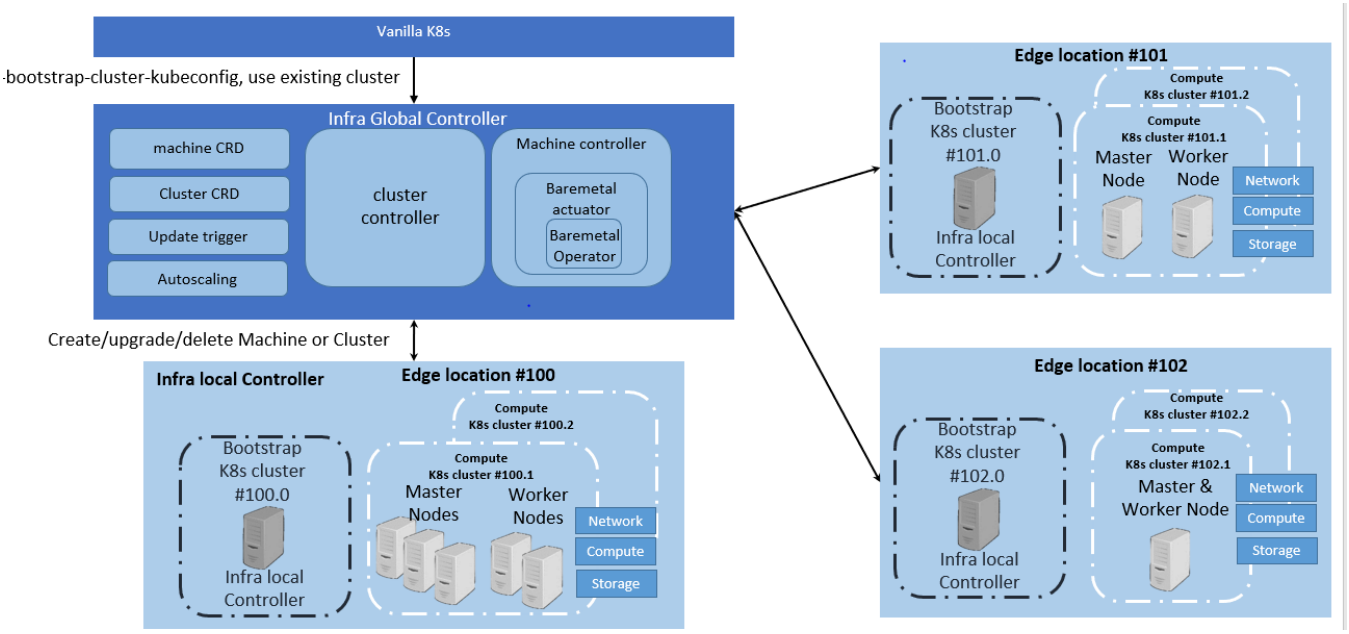
Overall Architecture

On an edge deployment, there may be multiple edges that need to be brought up. The Administrator going to each location, using the infra-local-controller to bring up application-K8S clusters in compute nodes of each location, is not scalable. Therefore, we have an "**infra-global-controller**" to control multiple "**infra-local-controllers**" which are controlling the worker nodes. The "infra-global-controller" is expected to provide a centralized software provisioning and configuration system. It provides one single-pane-of-glass for administrating the edge locations with respect to infrastructure. The worker nodes may be baremetal servers, or they may be virtual machines resident on the infra-local-controller. So the minimum platform configuration is one global controller and one local controller (although the local controller can be run without a global controller).

Since, there are a few K8S clusters, let us define them:

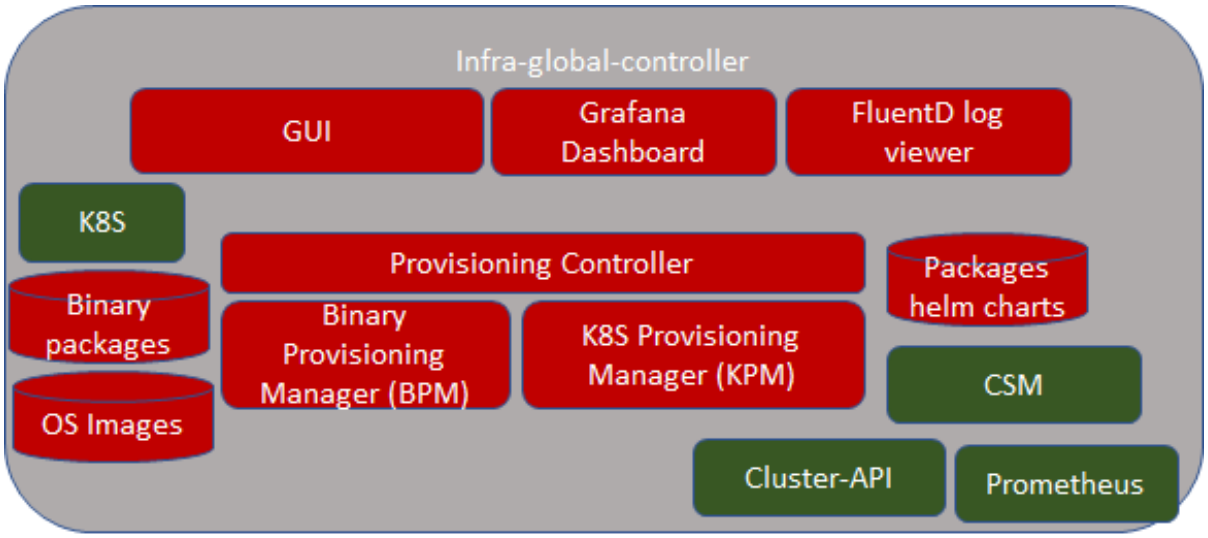
- infra-global-controller-K8S : This is the K8S cluster where infra-global-controller related containers are run.
- infra-local-controller-K8S: This is the K8S cluster where the infra-local-controller related containers are run, which bring up compute nodes.
- application-K8S : These are K8S clusters on compute nodes, where application workloads are run.

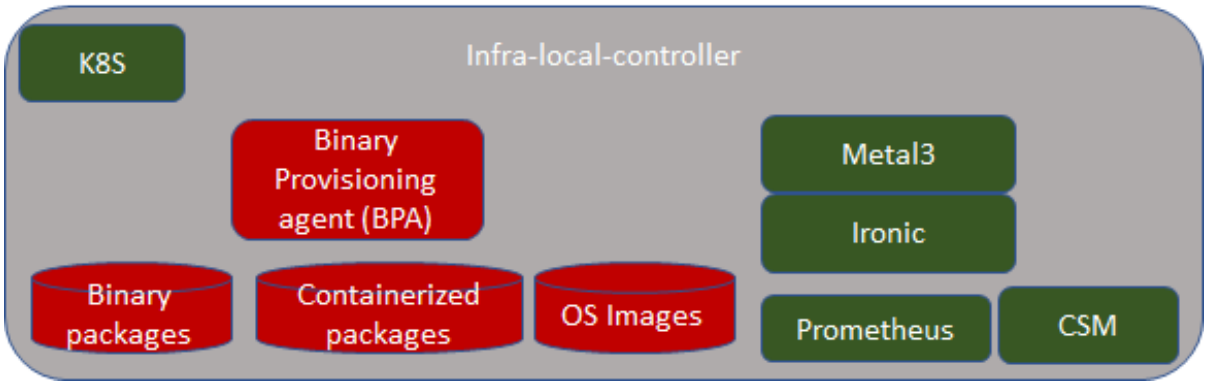
Flows & Sequence Diagrams



Each edge location has infra local controller, which has a bootstrap cluster, which has all the components required to boot up the compute cluster.

Platform Architecture





Infra-global-controller:

Administration involves

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

The infra-local-controller will be brought up in each location. The infra-local-controller kubeconfig will be made known to the infra-global-controller. Beyond that, everything else is taken care by the infra-global-controller. The 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. The 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
- CSM: Certificate and Secret management related Micro-services
- Cluster-API related Micro-services
- MongoDB for storing packages and OS images.
- Prometheus: Monitoring and alerting

Since we expect the infra-global-controller to be reachable from the Internet, we should 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.

Infra-local-controller:

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

As you see above in the picture, the 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 different K8S clusters. In case of the bootstrap machine, it itself is a complete K8S cluster with one node that has both master and minion software combined. All the components of the infra-local-controller (such as BPA, Metal3 and Ironic) 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 up 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. This means that the USB bootable disk shall have basic Linux, K8S and all containers coming up without any user actions. It must also 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 a USB disk. In this case, the developer can choose a machine as a 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 a bootstrap machine using this image.

Note that the infra-local-controller can be run without the infra-global-controller. In the interim release, we expect that only the infra-local-controller is supported. The infra-global-controller is targeted for the final Akraino R4 release. It is the goal that any operations done in the interim release on infra-local-controller manually are automated by infra-global-controller. And hence the interface provided by infra-local-controller is flexible enough to support both manual actions as well as automated actions.

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

1. Bring up a Linux operating system.
2. Provision the software with the 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 performed by Metal3 and Ironic. Step 3 is performed by BPA and Step 4 is done by talking to application-K8S

Metal3 Baremetal Operator & Ironic

The Baremetal Operator provides provisioning of compute nodes (either bare-metal or VM) by using the Kubernetes API. The Baremetal Operator defines a CRD BaremetalHost Object representing 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.

Binary Provisioning Agent (BPA)

The job of the BPA is to install all packages to the application-K8S that can't be installed using kubectl. Hence, the BPA is used right after the compute nodes get installed with the 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 the 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

The BPA also provides some RESTful APIs for doing the following:

- To upload binary images that are used to install the stuff in compute nodes.
- To upload a 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

- The BPA also acts as a local Docker Hub repository and ensures that all K8S container packages (that need to be installed on the application-K8S) are served locally here.
- The BPA also configures docker to access packages from this local repository.

BPA also takes care of: (After interim release)

- When a new compute node is added, once the administrator adds the 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:

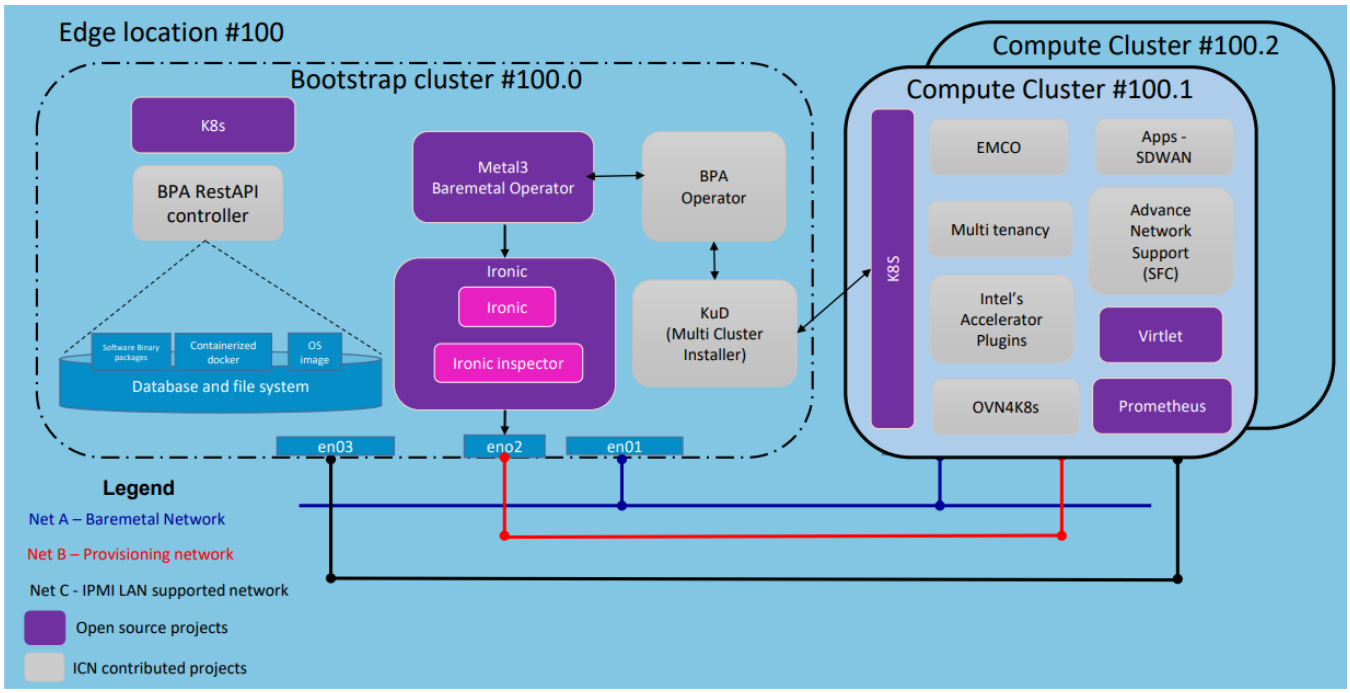
Ironic is expected to bring up Linux on compute nodes. It is also expected to create SSH keys automatically for each compute node. In addition, it is also expected to create SSH user for each compute node. Usernames and password are expected to be stored in SMS for security reasons in infra-local-controller.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

Software Platform Architecture

Local Controller: Kubeadm, Metal3, Baremetal Operator, Ironic, Prometheus, EMCO

Global Controller: Kubeadm, KuD, K8S Provisioning Manager, Binary Provisioning Manager, Prometheus, CSM

R4 Release cover only Infra local controller:



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, Baremetal operator from metal3 project is used as bare metal provider. 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.

KuD

Kubernetes deployment (KUD) is a project that uses Kubespray to bring up a Kubernetes deployment and some addons on a provisioned machine. As it already part of EMCO it can be effectively reused to deploy the K8s App components(as shown in fig. II), NFV Specific components and NFVi SDN controller in the edge cluster. In R4 release KuD will be used to deploy the K8s addon such as Virlet, OVN, NFD, and Intel device plugins such as SRIOV in the edge location(as shown in figure I).

One of the Kubernetes clusters with high availability, which is provisioned and configured by KUD will be used to deploy EMCO on K8s. ICN family uses Edge Multi-Cluster Orchestration for service orchestration. EMCO provides a set of helm chart to be used to run the workloads on a Multi - cluster.

EMCO Block and Modules:

EMCO 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 EMCO 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 and docker as a de-facto container runtime. 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 SRIOV

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.

Modules Design & Architecture:

Please explain each component & their design/architecture, Please keep maximum 2 paragraph, if possible link your project wiki link for more information

Metal3:

ICN uses Metal3 project for provisioning server in the edge locations, ICN project uses IPMI protocol to identify the servers in the edge locations, and use Ironic & Ironic - Inspector to provision the OS in the edge location. For R4 release, ICN project provision Ubuntu 18.04.5 in each server, and uses the distinguished network such provisioning network and bare-metal network for inspection and ipmi provisioning

ICN project injects the user data in each server regarding network configuration, grub update to enable IOMMU, remote command execution using ssh and maintain a common secure mechanism for all provisioning the servers. Each local controller maintains IP address management for that edge location. For more information refer - Metal3 Baremetal Operator in ICN stack

BPA Operator:

ICN uses the BPA operator to install KUD. It can install KUD either on Baremetal hosts or on Virtual Machines. The BPA operator is also used to install software on the machines after KUD has been installed successfully

KUD Installation

Baremetal Hosts: When a new provisioning CR is created, the BPA operator function is triggered, it then uses a dynamic client to get a list of all Baremetal hosts that were provisioned using Metal3. It reads the MAC addresses from the provisioning CR and compares with the baremetal hosts list to confirm that a host with that MAC address exists. If it exists, it then searches the DHCP lease file for corresponding IP address of the host, using the IP addresses of all the hosts in the provisioning CR, it then creates a host.ini file and triggers a job that installs KUD on the machines using the hosts.ini file. When the job is completed successfully, a k8s cluster is running in the Baremetal hosts. The bpa operator then creates a configmap using the hosts name as keys and their corresponding IP addresses as values. If a host containing a specified MAC address does not exist, the BPA operator throws an error.

Virtual Machines : ICN project uses Virtlet for provisioning virtual machines in the edge locations. For this release, it involves a nested Kubernetes implementation. Kubernetes is first installed with Virtlet. Pod spec files are created with cloud init user data, network annotation with mac address, CPU and Memory requests. Virtlet VMs are created as per cluster spec or requirement. Corresponding provisioning custom resources are created to match the mac addresses of the Virtlet VMs.

BPA operator checks the provisioning custom resource and maps the mac address(es) to the running Virtlet VM(s). BPA operator gets the IP addresses of those VMs and initiates an installer job which runs KuD scripts in those VMs. Upon completion, the K8s cluster is ready running in the Virtlet VMs.

Software Installation

When a new software CR is created, the reconcile loop is triggered, on seeing that it is a software CR, the bpa operator checks for a configmap with a cluster label corresponding to that in the software CR, if it finds one, it gets the IP addresses of all the master and worker nodes, ssh's into the hosts and installs the required software. If no corresponding config map is found, it throws an error.

Refer

- Binary Provisioning Agent (BPA) Operator Specs
- BPA Software CR Specs

BPA Rest Agent:

Provides a straightforward RESTful API that exposes resources: Binary Images, Container Images, and OS Images. This is accomplished by using MinIO for object storage and Mongodb for metadata.

POST - Creates a new image resource using a JSON file.

GET - Lists available image resources.

PATCH - Uploads images to the MinIO backend and updates Mongodb.

DELETE - Removes the image from MinIO and Mongodb.

More on BPA Restful API can be found at ICN Rest API.

KuD

Kubernetes deployment (KUD) is a project that uses Kubespray to bring up a Kubernetes deployment and some addons on a provisioned machine. As it already part of EMCO it can be effectively reused to deploy the K8s App components(as shown in fig. II), NFV Specific components and NFVi SDN controller in the edge cluster. In R4 release KuD will be used to deploy the K8s addon such as Virlet, OVN, NFD, CMK CPU Manager for Kubernetes and Intel device plugins such as SRIOV and QAT in the edge location(as shown in figure I).

EMCO:

EMCO is used as Service orchestration in ICN BP. EMCO is developed as part of Multicloud-k8s project in ONAP community. ICN BP developed containerized KUD multi-cluster to install the EMCO as a plugin in any cluster provisioned by BPA operator. EMCO installed Composite vFW application to install in any edge location.

SDEWAN:

SDEWAN CNF module is worked as a software-defined router located in each edge location and central hub k8s cluster to manage central-edge and edge-edge communication. It's functionality is realized via CNF (Containerized Network Function) and deployed by K8s, it is based on OpenWRT (an open-source project based on Linux, and used on embedded devices to route network traffic) and leverages Linux kernel functionality for packet processings to support network functionalities such as multiple wan link support (mwan3), firewall/SNAT/DNAT (fw3) , IPSec (strongswan) etc. It exposes Restful APIs for configuration, detail information can be found at: SDEWAN CNF

SDEWAN Configure Agent(also named SDEWAN Controller) module is worked as K8s controller located in each edge location and central hub k8s cluster to support configuration of SDEWAN CNF functionalities (e.g. mwan3, firwall, SNAT, DNAT, IPSec etc.) and monitor SDEWAN CNF status. It exposes CRDs to support configuration via K8s API server for unified authentication and authorization, detail information can be found at: Sdewan CRD Controller

Cloud Storage:

Cloud Storage (Cloud Storage Design) act as storage service and plugins, currently can divide into two parts:

1. Storage Service for Local controller: which used by BPA Rest Agent to provide storage service for image objects with binary, container and operating system. There are 2 solutions, MinIO and GridFS, with the consideration of Cloud native and Data reliability, we propose to use MinIO, which is CNCF project for object storage and compatible with Amazon S3 API, and provide language plugins for client application, it is also easy to deploy in Kubernetes and flexible scale-out. MinIO also provide storage service for HTTP Server. Since MinIO need export volume in bootstrap, local-storage is a simple solution but lack of reliability for the data safety, we will switch to reliability volume provided by Ceph CSI RBD in next release.
2. Optane Persistent Memory plugin in KUD, which can provide LVM and direct volumes on Optane PM namespaces, since the Optane PM has high performance and low latency compared with normal SSD storage device, it can be used as cache, metadata volume or other high throughput and low latency scenarios.

Software components:

Components	Link	License	Akraino Release target
ICN	https://github.com/akraino-edge-stack/icn - v0.4.0	Apache License 2.0	R4
Provision stack - Metal3	https://github.com/akraino-icn/baremetal-operator - v1.1-icn	Apache License 2.0	R4
Ironic - Ironic IPA downloader	https://github.com/akraino-icn/ironic-ipa-downloader - v1.0-icn	Apache License 2.0	R4
Ironic - Ironic image	https://github.com/akraino-icn/ironic-image - v1.0-icn	Apache License 2.0	R4
Ironic - Ironic Inspector Image	https://github.com/akraino-icn/ironic-inspector-image - v1.0-icn	Apache License 2.0	R4
Host Operating system	Ubuntu 18.04.5	GNU General Public License	R4
NIC drivers	XL710 - https://www.intel.com/content/dam/www/public/us/en/documents/datasheets/xl710-10-40-controller-datasheet.pdf	GNU General Public License Version 2	R4
QAT drivers	Intel® C627 Chipset - https://ark.intel.com/content/www/us/en/ark/products/97343/intel-c627-chipset.html	GNU General Public License Version 2	R4
Intel® Optane™ DC Persistent Memory	Intel® Optane™ DC 256GB Persistent Memory Module - https://www.intel.com/content/www/us/en/products/memory-storage/optane-dc-persistent-memory/optane-dc-256gb-persistent-memory-module.html PMDK: Persistent Memory Development Kit - https://github.com/pmem/pmdk/	SPDX-License-Identifier - BSD-3-Clause	R4

EMCO (formerly known as ONAP4K8s)	https://git.onap.org/multicloud	Apache License 2.0	R4
SDEWAN CNFs	https://github.com/akraino-edge-stack/icn-sdwan - v1.0 https://hub.docker.com/repository/docker/integratedcloudnative/openwrt - 0.3.1	GNU General Public License Version 2	R4
KUD	https://git.onap.org/multicloud/k8s/	Apache License 2.0	R4
Kubespray	https://github.com/kubernetes-sigs/kubespray v2.14.1	Apache License 2.0	R4
K8s	https://github.com/kubernetes/kubeadm - v1.18.9	Apache License 2.0	R4
Docker	https://github.com/docker - 19.03.13	Apache License 2.0	R4
Virtlet	https://github.com/Mirantis/virtlet -1.4.4	Apache License 2.0	R4
SDN - OVN	https://github.com/akraino-icn/ovn/ - v20.06.0 (mirror repo - https://github.com/ovn-org/ovn)	Apache License 2.0	R4
vSwitch - OVS	https://github.com/akraino-icn/ovs - v2.14.0 (mirror repo - https://github.com/openvswitch/ovs)	Apache License 2.0	R4
Ansible	https://github.com/ansible/ansible - 2.9.7	Apache License 2.0	R4
Helm	https://github.com/helm/helm - 3.2.4	Apache License 2.0	R4
Istio	https://github.com/istio/istio - 1.0.3	Apache License 2.0	R4
Rook/Ceph	https://rook.io/docs/rook/v1.0/helm-operator.html v1.0	Apache License 2.0	R4
MetalLB	https://github.com/danderson/metallb/releases - v0.7.3	Apache License 2.0	R4
OVN4NFV-K8Ss-Plugin	https://github.com/opnfv/ovn4nfv-k8s-plugin - v0.9.0	Apache License 2.0	R4
SDEWAN controller	https://github.com/akraino-edge-stack/icn-sdwan - v1.0 https://hub.docker.com/repository/docker/integratedcloudnative/sdewan-controller - 0.3.0	Apache License 2.0	R4
Device Plugins	https://github.com/intel/intel-device-plugins-for-kubernetes - SRIOV	Apache License 2.0	R4
Node Feature Discovery	https://github.com/kubernetes-sigs/node-feature-discovery - 0.4.0	Apache License 2.0	R4
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/akraino-icn/multus-cni - Multus v3.4.1 tp, https://github.com/k8snetworkplumbingwg/sriov-cni	Apache License 2.0	R4

Hardware and Software Management

Software Management

ICN R4 Timelines

Hardware Management

Hostname	CPU Model	Memory	Storage	1GbE: NIC#, VLAN, (Connected extreme 480 switch)	10GbE: NIC# VLAN, Network (Connected with IZ1 switch)
Jump	2xE5-2699	64GB	3TB (Sata) 180 (SSD)	IF0: VLAN 110 (DMZ) IF1: VLAN 111 (Admin)	IF2: VLAN 112 (Private) **VLAN 114** (Management) IF3: VLAN 113 (Storage) **VLAN 1115** (Public)
node1	2xE5-2699	64GB	3TB (Sata) 180 (SSD)	IF0: VLAN 110 (DMZ) IF1: VLAN 111 (Admin)	IF2: VLAN 112 (Private) **VLAN 114** (Management) IF3: VLAN 113 (Storage) **VLAN 1115** (Public)
node2	2xE5-2699	64GB	3TB (Sata) 180 (SSD)	IF0: VLAN 110 (DMZ) IF1: VLAN 111 (Admin)	IF2: VLAN 112 (Private) **VLAN 114** (Management) IF3: VLAN 113 (Storage) **VLAN 1115** (Public)

node3	2xE5-2699	64GB	3TB (Sata) 180 (SSD)	IF0: VLAN 110 (DMZ) IF1: VLAN 111 (Admin)	IF2: VLAN 112 (Private) **VLAN 114** (Management) IF3: VLAN 113 (Storage) **VLAN 1115** (Public)
node4	2xE5-2699	64GB	3TB (Sata) 180 (SSD)	IF0: VLAN 110 (DMZ) IF1: VLAN 111 (Admin)	IF2: VLAN 112 (Private) **VLAN 114** (Management) IF3: VLAN 113 (Storage) **VLAN 1115** (Public)
node5	2xE5-2699	64GB	3TB (Sata) 180 (SSD)	IF0: VLAN 110 (DMZ) IF1: VLAN 111 (Admin)	IF2: VLAN 112 (Private) **VLAN 114** (Management) IF3: VLAN 113 (Storage) **VLAN 1115** (Public)

Licensing

Refer Software Components list