You make **possible**

Deploying Kubernetes with Cisco ACI

Camillo Rossi
Technical Marketing Engineer

BRKACI-2505

Barcelona | January 27-31, 2020

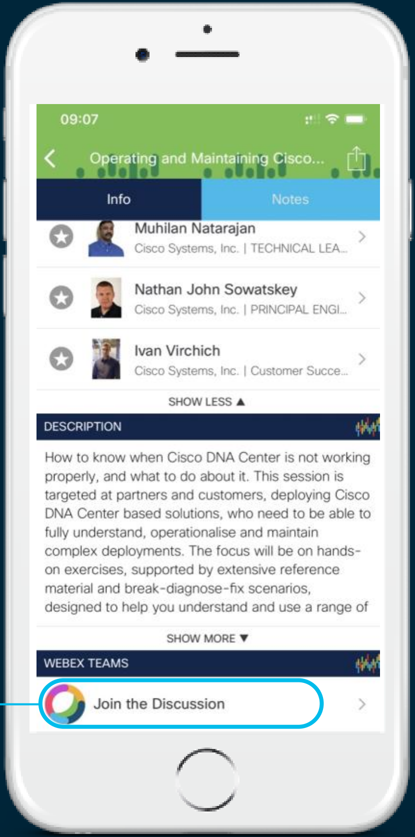
Cisco Webex Teams

Questions?

Use Cisco Webex Teams to chat
with the speaker after the session

How

1. Find this session in the Cisco Events Mobile App
2. Click “Join the Discussion”
3. Install Webex Teams or go directly to the team space
4. Enter messages/questions in the team space


09:07
Operating and Maintaining Cisco...
Info
Notes
Muhilan Natarajan
Cisco Systems, Inc. | TECHNICAL LEA...
Nathan John Sowatskey
Cisco Systems, Inc. | PRINCIPAL ENGI...
Ivan Virchich
Cisco Systems, Inc. | Customer Succe...
SHOW LESS ▲
DESCRIPTION
SHOW MORE ▼
WEBEX TEAMS
Join the Discussion

How to know when Cisco DNA Center is not working properly, and what to do about it. This session is targeted at partners and customers, deploying Cisco DNA Center based solutions, who need to be able to fully understand, operationalise and maintain complex deployments. The focus will be on hands-on exercises, supported by extensive reference material and break-diagnose-fix scenarios, designed to help you understand and use a range of

Agenda

- ACI-Kubernetes value proposition
- Kubernetes Recap
- ACI and Kubernetes Solution Overview
- Demos
- Q&A

Why ACI for Application Container Platforms

Turnkey solution for node and container connectivity

Flexible policy: Native platform policy API and ACI policies

Hardware-accelerated: Integrated load balancing and Source NAT

Visibility: Live statistics in APIC per container and health metrics

Enhanced Multitenancy and unified networking for containers, VMs, bare metal

Fast, easy, secure and scalable **networking** for your Application Container Platform

Agenda

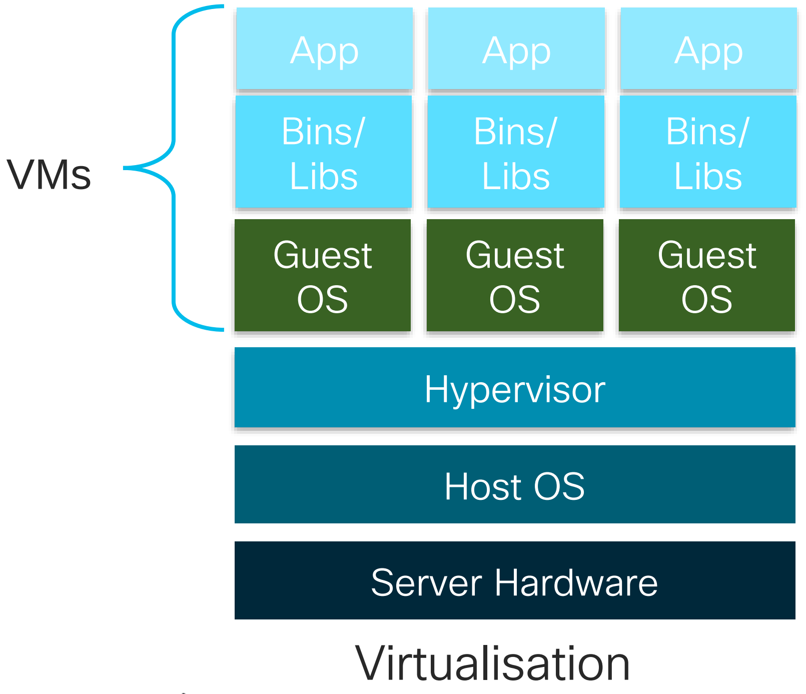
- ACI-Kubernetes value proposition
- Kubernetes Recap
- ACI and Kubernetes Solution Overview
- Demos
- Q&A

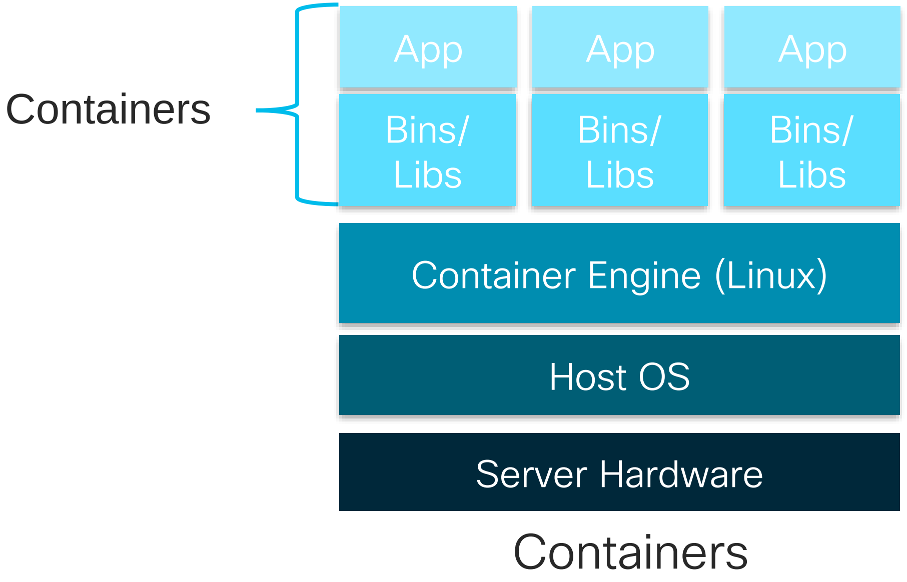
What are containers?

What is a container?

- A container is a binary executable, packaged with dependencies and intended for execution in a private namespace with optional resource constraints.
- This provides the containers multiple isolated operating system environments with their own file system, network, process and block I/O space on the same host

Compute Virtualisation and Containers



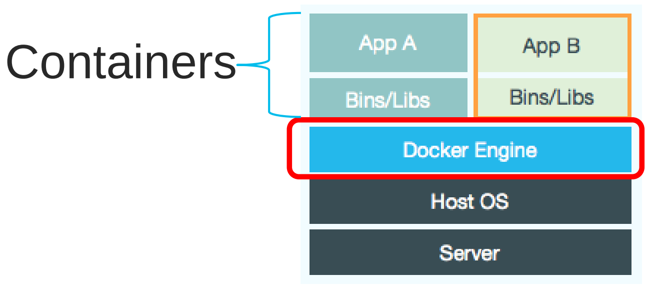


Docker

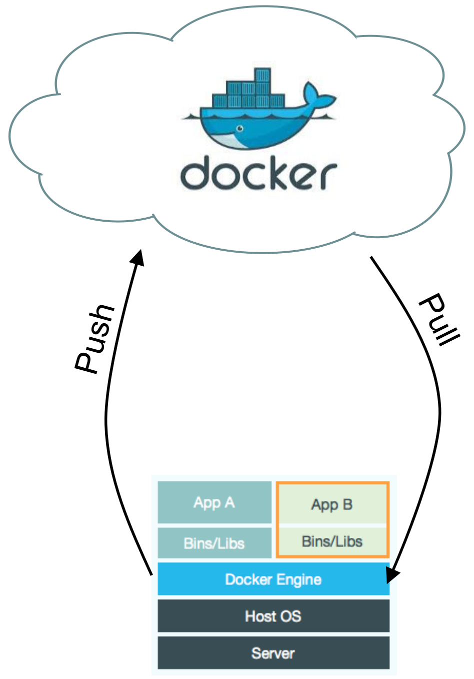
A Docker containers wrap a piece of software in a complete filesystem that contains *everything needed to run*: code, runtime, system tools, system libraries – anything that can be installed on a server.

This guarantees that the software will always run the same, regardless of its environment.

Docker consists of two main components:



Docker Engine
– the actual app
running on the host.



Docker Hub – SaaS component for managing and sharing containers.

Kubernetes Refresh

Kubernetes

- Kubernetes is an open source Container Orchestration system for automating deployment, scaling and management of containerised applications.
- It was inspired by the Google Borg System and with its v1.0 release in July 2015, Google donated it to the Cloud Native Computing Foundation (CNCF).
- Generally, Kubernetes has new releases every three months. The current stable version is 1.17 (as of Jan 2020).

Kubernetes and Docker

- Kubernetes uses Docker to execute/run the containers
- Kubernetes adds, on top of Docker, all the intelligence and features of an orchestrator

Kubernetes - pod

- A pod is the scheduling unit in Kubernetes. It is a logical collection of one or more containers which are always scheduled together.
- The set of containers composed together in a pod share an IP.

```
[root@k8s-01-p1 ~]# kubectl get pod  --namespace=kube-system
NAME                                      READY     STATUS    RESTARTS   AGE
aci-containers-controller-1201600828-qsw5g   1/1       Running   1          69d
aci-containers-host-lt9kl                 3/3       Running   0          72d
aci-containers-host-xnwkr                 3/3       Running   0          58d
aci-containers-openvswitch-0rjbw          1/1       Running   0          58d
aci-containers-openvswitch-7j1h5          1/1       Running   0          72d
```

Kubernetes – Deployment

- Deployments are a collection of pods providing the same service
- You describe the desired state in a Deployment object, and the Deployment controller will change the actual state to the desired state at a controlled rate for you
- For example you can create a deployment that declare you need to have 2 copies of your front-end pod.

```
[root@k8s-01-p1 ~]# kubectl get deployment --namespace=kube-system
NAME                         DESIRED   CURRENT   UP-TO-DATE    AVAILABLE   AGE
aci-containers-controller    1         1         1             1           72d
```

Kubernetes – Services

- A service tells the rest of the Kubernetes environment (including other pods and Deployments) what services your application provides.
- While pods come and go, the service IP addresses and ports remain the same.
- Kubernetes automatically load balance the load across the replicas in the deployment that you expose through a Service
- Other applications can find your service through Kubernetes service discovery.
    - Every time a service is create a DNS entry is added to kube-dns

```
[root@k8s-01-p1 ~]# kubectl get svc --namespace=kube-system
NAME        CLUSTER-IP    EXTERNAL-IP    PORT(S)          AGE
kube-dns    11.96.0.10    <none>         53/UDP,53/TCP    72d
```

Kubernetes – External Services

- If there are external IPs that route to one or more cluster nodes, Kubernetes services can be exposed on those external IPs.
- Traffic that ingresses into the cluster with the external IP (as destination IP), on the service port, will be routed to one of the service endpoints.
- External IPs are not managed by Kubernetes and are the responsibility of the cluster administrator.

```
[root@k8s-01-p1 ~]# kubectl get svc front-end --namespace=guest-book
NAME         CLUSTER-IP    EXTERNAL-IP    PORT(S)         AGE
front-end    11.96.0.33    11.3.0.2       80:30002/TCP    3m
```

Kubernetes – Ingress

- An Ingress is a collection of rules that allow inbound connections to reach the cluster services.
- It can be configured to give services externally-reachable URLs, load balance traffic, terminate SSL, offer name based virtual hosting, and more
- Think of NGINX

```
[root@k8s-01-p1 ~]# kubectl get ingress
NAME            HOSTS     ADDRESS    PORTS      AGE
test-ingress    *                    80         7s
```

Kubernetes - Labels

- Kubernetes uses labels as “nametags” to identify things.
- Can be used to indicate roles, stability, or other important attributes.
- You can query anything in Kubernetes via a label.
  - i.e. Return all the pod that are running “PreProduction” workload

```
[root@k8s-01-p1 ~]# kubectl get pod --namespace=kube-system -l component=kube-apiserver
NAME                           READY     STATUS    RESTARTS   AGE
kube-apiserver-k8s-01-p1       1/1       Running   0          72d
```

Kubernetes - Annotations

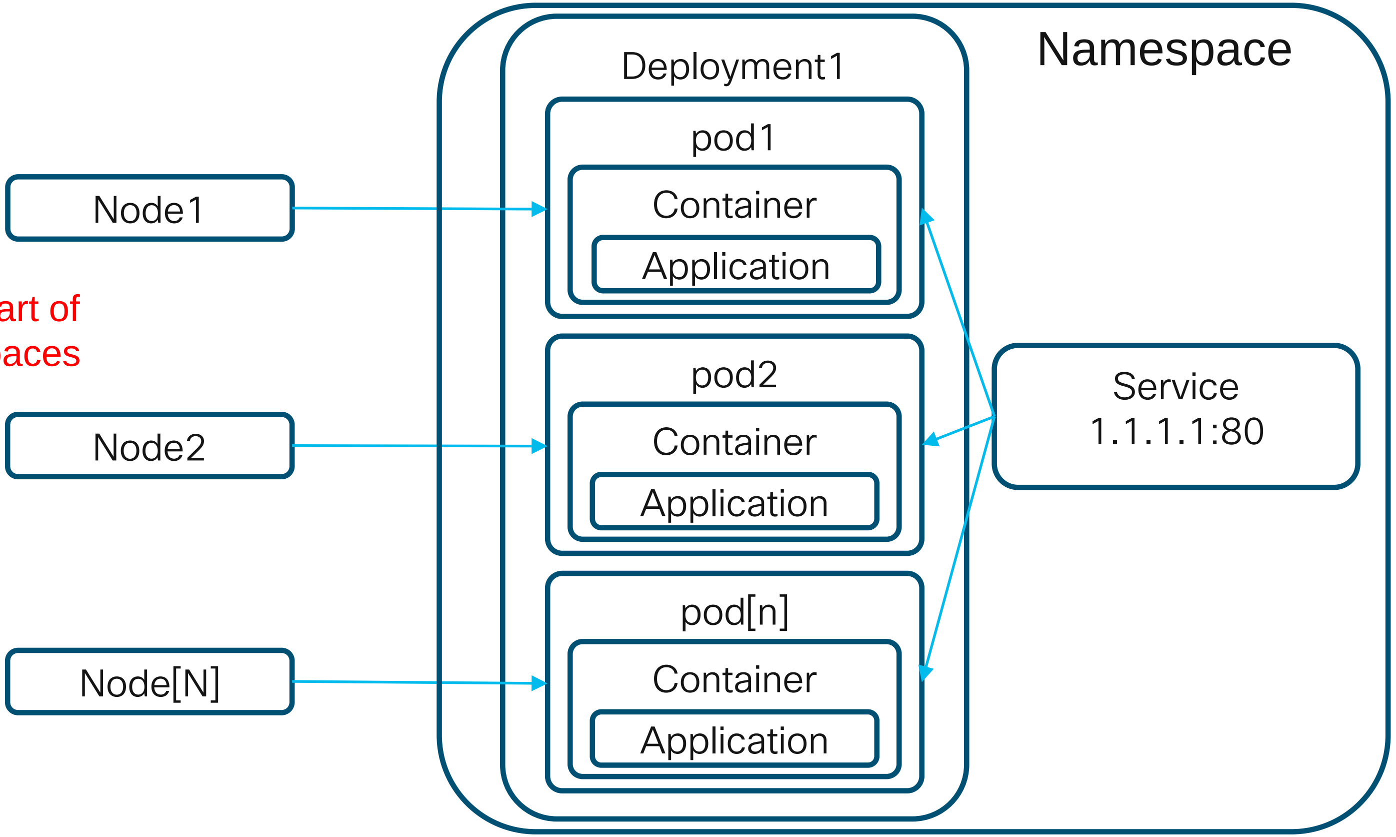
- Similar to labels but are NOT used to identify and select object
- Used in ACI, yes soon we will be speaking about ACI and Kubernetes ☺

```
[root@k8s-01-p1 ~]# kubectl describe node k8s-01-p1 | more
Name:                  k8s-01-p1
Role:
Labels:                      beta.kubernetes.io/arch=amd64
                       beta.kubernetes.io/os=linux
                       kubernetes.io/hostname=k8s-01-p1
                       node-role.kubernetes.io/master=
Annotations:                 node.alpha.kubernetes.io/ttl=0
                       opflex.cisco.com/pod-network-ranges={"V4":[{"start":"11.2.0.130","end":"11.2.1.1"}]}
                       opflex.cisco.com/service-endpoint={"mac":"66:85:9a:e9:ef:2f","ipv4":"11.5.0.3"}
                       volumes.kubernetes.io/controller-managed-attach-detach=true
```

Kubernetes – Namespace

- Groups everything together:
  - Pod
  - Deployment
  - Volumes
  - Services
  - Etc…

All Together: A K8S Cluster



A node can be part of
Several Namespaces

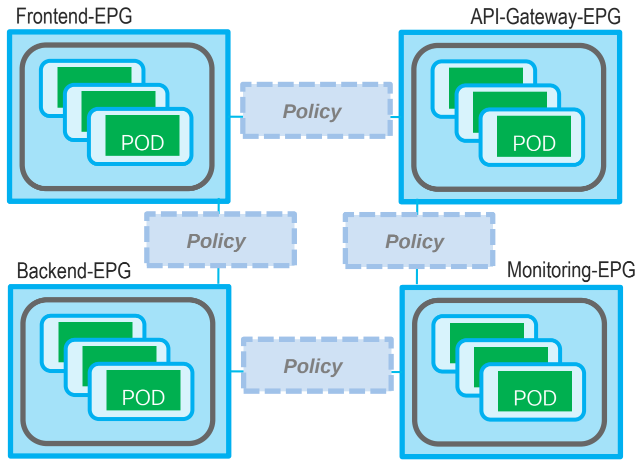
‘Native’ Network Challenges

Operations and Visibility

- Skills gap between network and Kubernetes admins
- Visibility and governance of network policies
- Simplified Network Operations

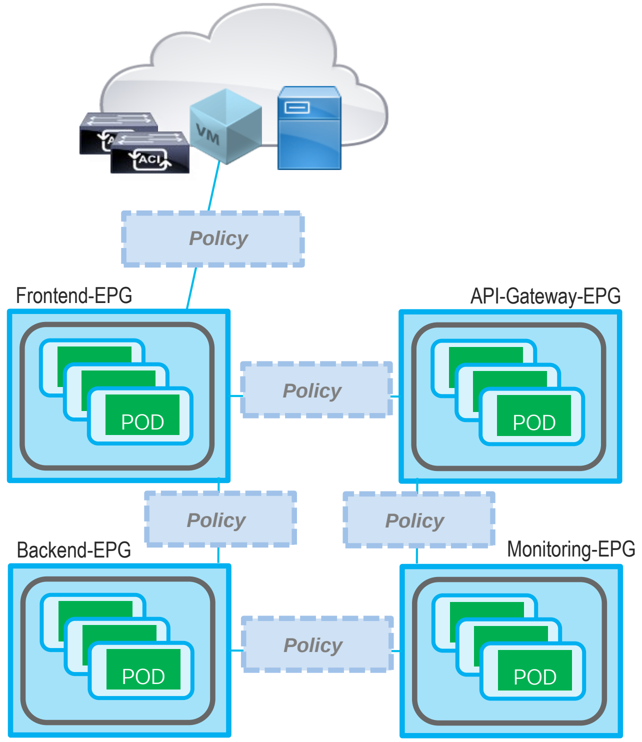
Segmentation

- Secure K8s **infrastructure**:
  - network isolation for kube-system and other infrastructure related objects (i.e. heapster, hawkular, etc.)
- Network isolation between **namespaces**

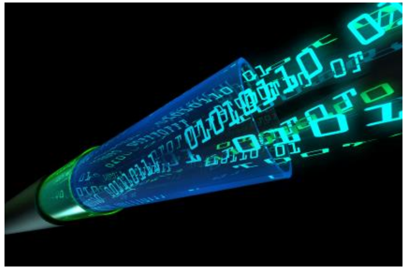


Communications outside of the Cluster

- Non-Cluster endpoints communicating with Cluster:
    - Exposing external services, how? NodePort? LoadBalancer?
    - Scaling-out ingress controllers, how can you scale?
- Cluster endpoints communicating with non-cluster endpoints:
    - POD access to external services and endpoints



Storage Access from Nodes



- Applications running in OpenShift Pods that need high-bandwidth, low-latency traffic to data external to the cluster suffer the bottleneck imposed by the egress router implementation. i.e. centralized storage from node or PODs:
  - iSCSI, NFS, GlusterFS, CEPH, etc.
  - HyperFlex

ACI and Kubernetes Solution Overview

What is Container Networking Interface

CNI

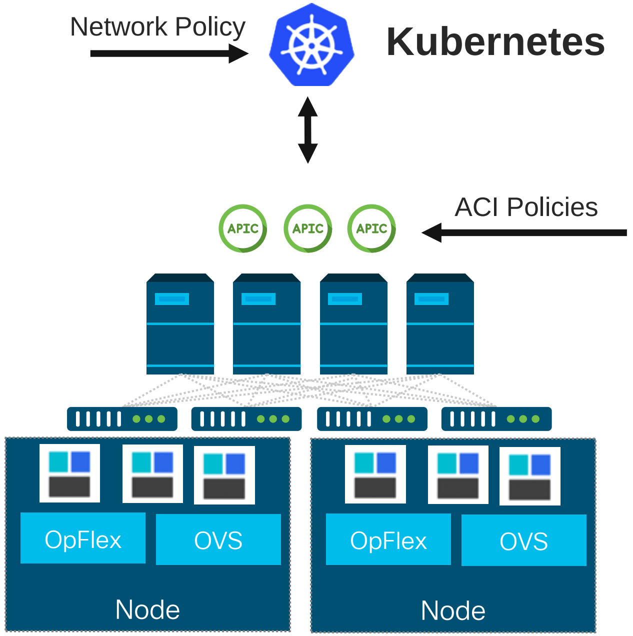
- A generic plugin-based networking solution for application containers on Linux
- The spec defines a container as being a Linux network namespace
- The plugin must connect containers to networks and is responsible for IPAM and DNS configurations.

Cisco ACI CNI Plugin Benefits

1. **Simplified Operations and Enhanced visibility**
2. **Granular security**: security can be implemented by using native NetworkPolicy or by using ACI EPGs and contracts, or both models complementing each other.
3. **Unified networking**: Pod and Service endpoints become first class citizens at the same level as Bare Metal or Virtual Machines.
4. **High performance**: low-latency secure connectivity without egress routers
5. **Hardware-assisted load balancing**: ingress connections to LoadBalancer-type services using ACI's Policy Based Redirect technology

Cisco ACI CNI plugin features

- IP Address Management for Pods and Services
- Distributed Routing and Switching with integrated VXLAN overlays implemented fabric wide and on Open vSwitch
- Distributed Firewall for implementing Network Policies
- EPG-level segmentation for OCP objects using annotations
- Consolidated visibility of OCP networking via VMM Integration



ACI Basic Configuration

Kubernetes Nodes will require the following interfaces

- InfraVLAN – sub-interface over which we build the opflex channel
- Node IP – sub-interface used for the Kubernetes API host IP address
- (Optional) OOB Management – sub-interface or physical interface used optionally for OOB access.

IMPORTANT: the default route must be on the Node IP interface.

acc-provision

- ACI Container Controller Provision:
  - Takes a YAML file containing the parameters of your configuration
  - Generates and pushes most of the ACI config
  - Generates Kubernetes ACI CNI containers configuration

```
acc-provision --flavor=kubernetes-1.15 -a -u admin -p pass –c config.yml –o cni_conf.yml
```

Used to select if we are deploying kubernetes 1.x or OpenShift 3.x

APIC user and password

Configuration file

Output file for ACI CNI config

acc-provision – configuration file (1)

```
aci_config:
  system_id: KubeSpray          # Tenant Name and Controller Domain Name
  apic_hosts:                   # List of APIC hosts to connect for APIC API
  - 10.67.185.102
  vmm_domain:                   # Kubernetes VMM domain configuration
    encap_type: vxlan           # Encap mode: vxlan or vlan
    mcast_range:                # mcast range for BUM replication
      start: 225.22.1.1
      end: 225.22.255.255
    mcast_fabric: 225.1.2.4
    nested_inside:              # (OPTIONAL) If running k8s node as VMs specify the VMM Type and Name.
      type: vmware              # Only vmware for now, ports groups created automatically with system_id name
      name: ACI

  # The following resources must already exist on the APIC,
  # they are used, but not created by the provisioning tool.
  aep: ACI_AttEntityP           # The AEP for ports/VPCs used by this cluster
  vrf:                          # The VRF can be placed in the same Tenant or in Common.
    name: vrf1
    tenant: KubeSpray           # This can be the system-id or common
  l3out:
    name: l3out                 # Used to provision external IPs
    external_networks:
    - default_extepg            # Default Ext EPG, used for PBR redirection
```

acc-provision – configuration file (2)

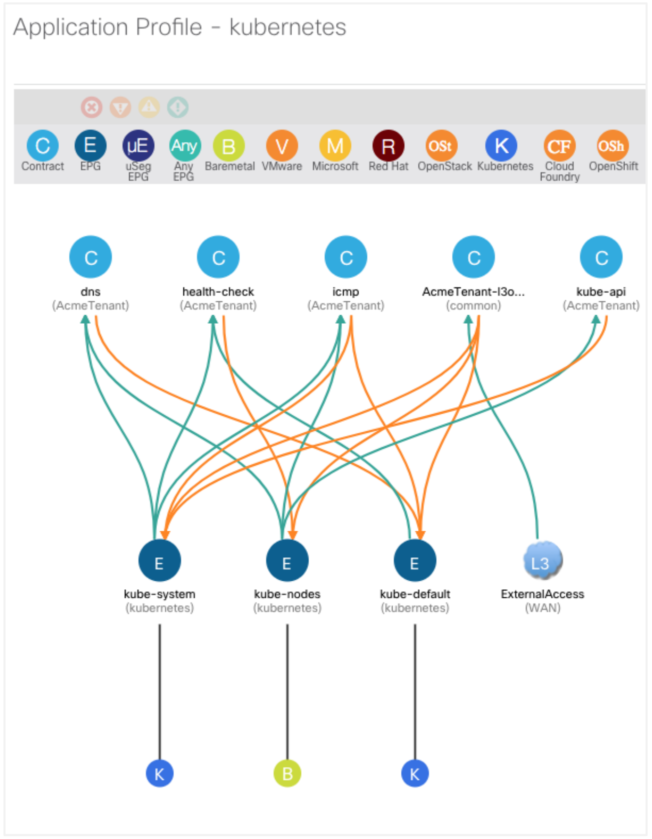
```
#
# Networks used by Kubernetes
#
net_config:
  node_subnet: 10.32.0.1/16      # Subnet to use for nodes
  pod_subnet: 10.33.0.1/16       # Subnet to use for Kubernetes Pods
  extern_dynamic: 10.34.0.1/24   # Subnet to use for dynamic external IPs
  extern_static: 10.35.0.1/24    # Subnet to use for static external IPs
  node_svc_subnet: 10.36.0.1/24  # Subnet to use for service graph
  kubeapi_vlan: 4011             # The VLAN used by for nodes to node API communications
  service_vlan: 4013             # The VLAN used by LoadBalancer services
  infra_vlan: 3456               # The ACI infra VLAN used to establish the OpFlex tunnel with the leaf

#
# Configuration for container registry
# Update if a custom container registry has been setup
#
registry:
  image_prefix: noiro                # DO NOT CHANGE
```

acc-provision will now create

tenant EPGs for nodes and Pods

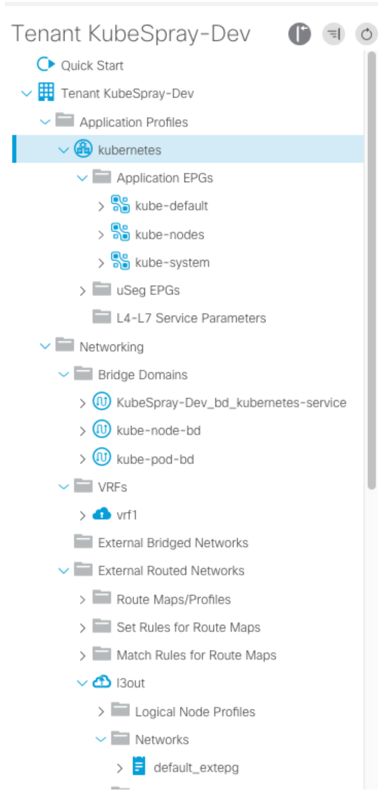
Within the tenant selected the provisioning tool creates a 'Kubernetes' Application Profile with three EPGs:

- 'kube-nodes': for node interfaces, mapped to PhysDom
- 'kube-system': for system PODs (i.e. kube-dns), mapped to VMMDom
- 'kube-default': base EPG for all containers on any namespace by default, mapped to VMMDom



tenant BDs for nodes and Pods

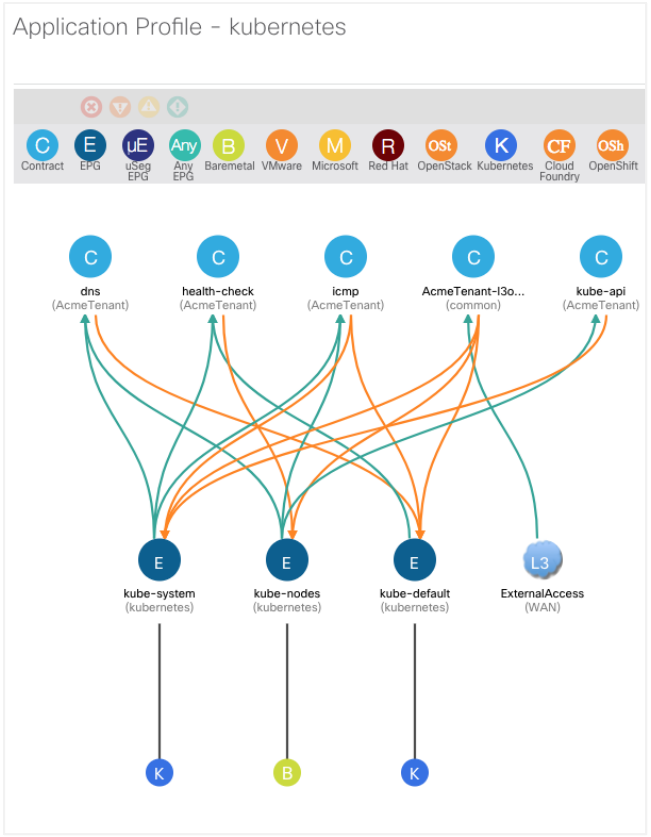
- kube-nodes-bd:
  - Only used for kube-node EPG
  - Maps to `node_subnet`
- kube-pod-bd:
  - Any pod will be assigned an IP from this BD Subnet
  - Used for kube-default, kube-system and any other user defined POD EPGs.
  - Maps to `pod_subnet`
- Cluster…-service:
  - BD for PBR/SG services
  - Created when ACI CNI plugin is deployed



Tenant KubeSpray-Dev
- Quick Start
- Tenant KubeSpray-Dev
  - Application Profiles
    - kubernetes
      - Application EPGs
        - kube-default
        - kube-nodes
        - kube-system
      - uSeg EPGs
      - L4-L7 Service Parameters
  - Networking
    - Bridge Domains
      - KubeSpray-Dev_bd_kubernetes-service
      - kube-node-bd
      - kube-pod-bd
    - VRFs
      - vrf1
    - External Bridged Networks
    - External Routed Networks
      - Route Maps/Profiles
      - Set Rules for Route Maps
      - Match Rules for Route Maps
      - l3out
        - Logical Node Profiles
        - Networks
          - default_extepg

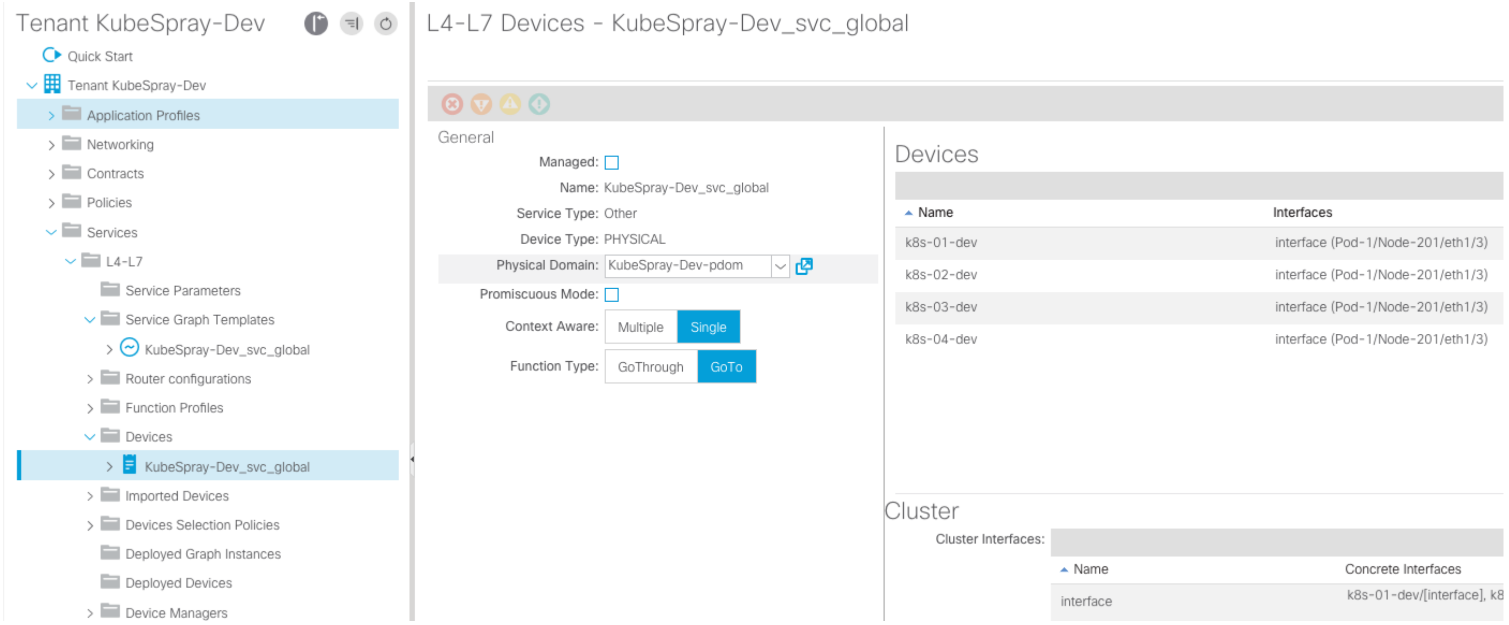
required tenant contracts

- The required minimum set of contracts are automatically configured to ensure basic cluster functionality
  - DNS
  - Health-check
  - ICMP
  - Kube-API
- Administrator can define additional contracts if/when required



ACI Fabric Configuration – L4-L7 Devices

- Created once the ACI CNI plugin is deployed
- Dynamically updated if nodes are added or removed from the k8s cluster
- Service Graph Template: Specify a template for PBR redirection


Tenant KubeSpray-Dev

- Quick Start
- Tenant KubeSpray-Dev
  - Application Profiles
  - Networking
  - Contracts
  - Policies
  - Services
    - L4-L7
      - Service Parameters
      - Service Graph Templates
        - KubeSpray-Dev_svc_global
      - Router configurations
      - Function Profiles
      - Devices
        - KubeSpray-Dev_svc_global
      - Imported Devices
      - Devices Selection Policies
      - Deployed Graph Instances
      - Deployed Devices
      - Device Managers

L4-L7 Devices - KubeSpray-Dev_svc_global

General

Managed:	☐
Name:	KubeSpray-Dev_svc_global
Service Type:	Other
Device Type:	PHYSICAL
Physical Domain:	KubeSpray-Dev-pdom
Promiscuous Mode:	☐
Context Aware:	Multiple [✓ Single]
Function Type:	GoThrough [✓ GoTo]

Devices

Name	Interfaces
k8s-01-dev	interface (Pod-1/Node-201/eth1/3)
k8s-02-dev	interface (Pod-1/Node-201/eth1/3)
k8s-03-dev	interface (Pod-1/Node-201/eth1/3)
k8s-04-dev	interface (Pod-1/Node-201/eth1/3)

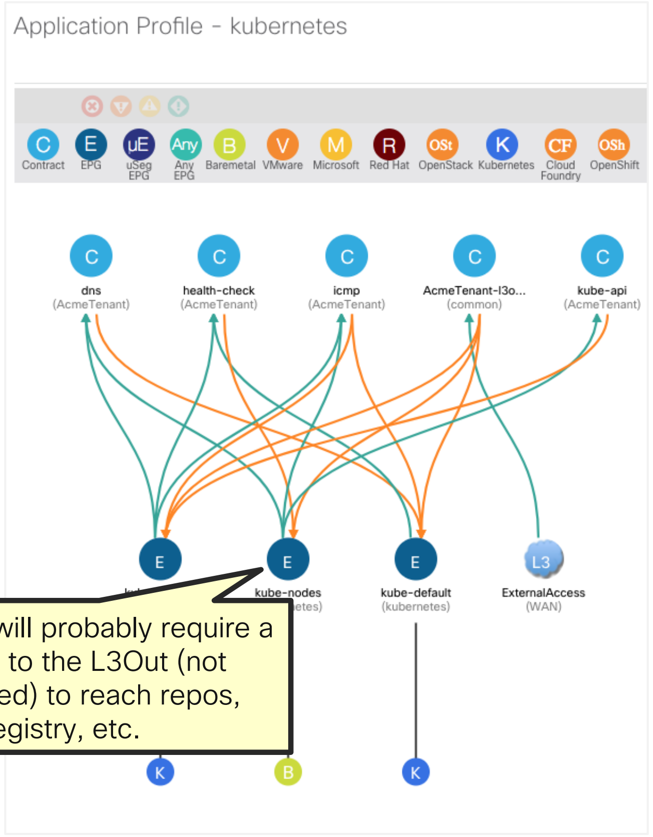
Cluster

Cluster Interfaces:

Name	Concrete Interfaces
interface	k8s-01-dev/[interface], k8

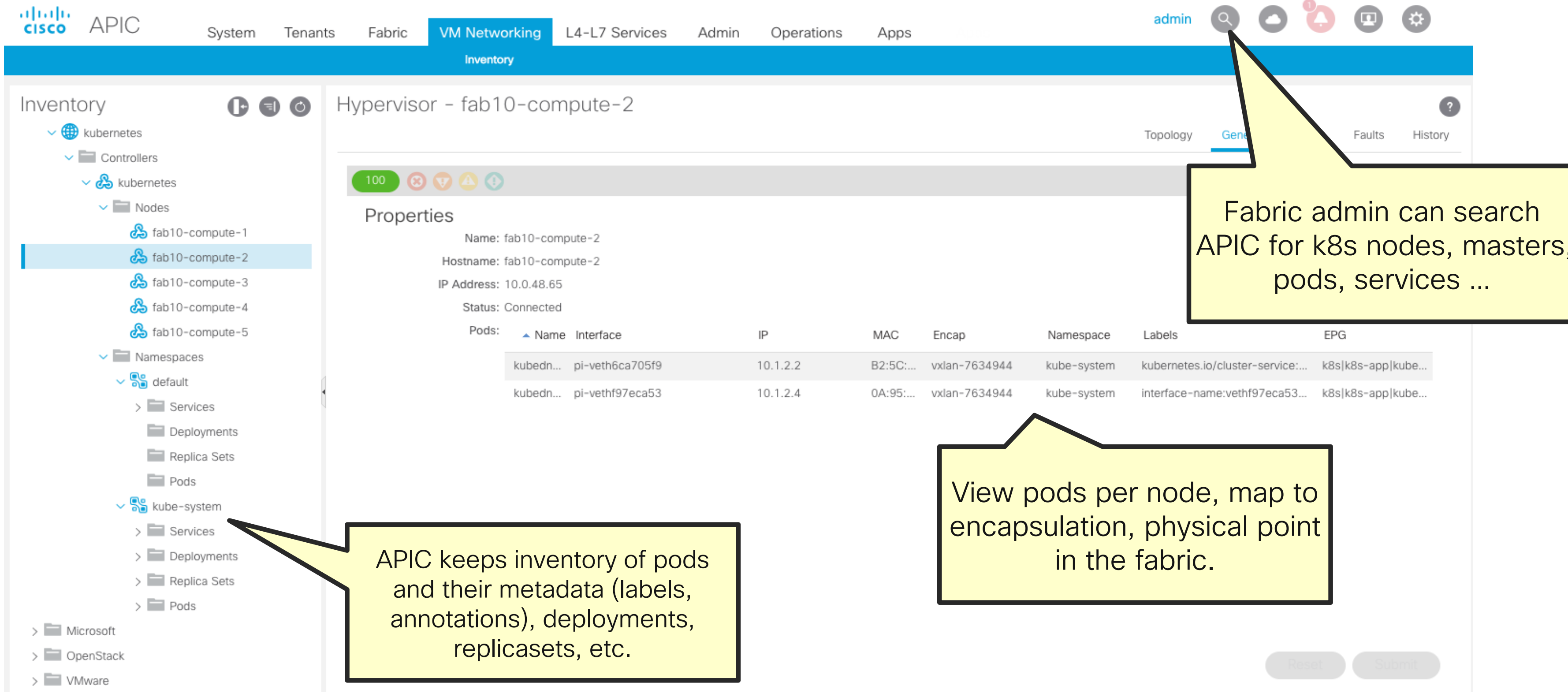
ACI Fabric Configuration – Required L3OUT

- Fabric Administrator must create and configure the L3OUT that will be used to expose external services
- The L3OUT name and the Default Networks name must match those in acc-provision configuration
- The External EPG used must be associated with the contract that allows nodes to reach repos, registry, etc.



The nodes will probably require a contract to the L3Out (not provisioned) to reach repos, registry, etc.

ACI Fabric Configuration – Container Domain



Inventory
- kubernetes
  - Controllers
    - kubernetes
      - Nodes
        - fab10-compute-1
        - fab10-compute-2
        - fab10-compute-3
        - fab10-compute-4
        - fab10-compute-5
      - Namespaces
        - default
          - Services
          - Deployments
          - Replica Sets
          - Pods
        - kube-system
          - Services
          - Deployments
          - Replica Sets
          - Pods
- Microsoft
- OpenStack
- VMware

Hypervisor - fab10-compute-2

Properties

Name:	fab10-compute-2
Hostname:	fab10-compute-2
IP Address:	10.0.48.65
Status:	Connected

Pods:

Name	Interface	IP	MAC	Encap	Namespace	Labels	EPG
kubedn...	pi-veth6ca705f9	10.1.2.2	B2:5C:...	vxlan-7634944	kube-system	kubernetes.io/cluster-service:...	k8s|k8s-app|kube...
kubedn...	pi-vethf97eca53	10.1.2.4	0A:95:...	vxlan-7634944	kube-system	interface-name:vethf97eca53...	k8s|k8s-app|kube...

Fabric admin can search APIC for k8s nodes, masters, pods, services ...

View pods per node, map to encapsulation, physical point in the fabric.

APIC keeps inventory of pods and their metadata (labels, annotations), deployments, replicasets, etc.

ACI CNI Plugin components

ACI CNI Plugin Components

- ACI Containers Controller (ACC, aci-containers-controller)
  - It is a Kubernetes **Deployment** running one POD instance.
  - Handles IPAM
  - Management of endpoint state
  - Management of SNAT Policies
  - Policy Mapping (annotations)
  - Controls Load Balancing
  - Pushes configurations into the APIC

```
[root@dom-master1 ~]# oc get deployments --namespace=aci-containers-system
NAME                       DESIRED   CURRENT   UP-TO-DATE   AVAILABLE   AGE
aci-containers-controller  1         1         1            1           223d
```

ACI CNI Plugin Components

- Aci Containers Host (ACH, aci-container-host):
  - is a DaemonSet composed of 3 containers running on every node
  - mcast-daemon:
    - Handles Broadcast, unknown unicast and multicast replication
  - aci-containers-host:
    - Endpoint metadata, Pod IPAM, Container Interface Configuration
  - opflex-agent:
    - Support for Stateful Security Groups, Manage configuration of OVS, Render policy to openflow rules to program OVS, handles loadbalancer services

```
[root@dom-master1 ~]# oc get daemonsets --namespace=aci-containers-system |grep host
NAME                       DESIRED  CURRENT  READY   UP-TO-DATE  AVAILABLE  NODE SELECTOR  AGE
aci-containers-host        2        2        2       2           2          <none>         223d
```

ACI CNI Plugin Components

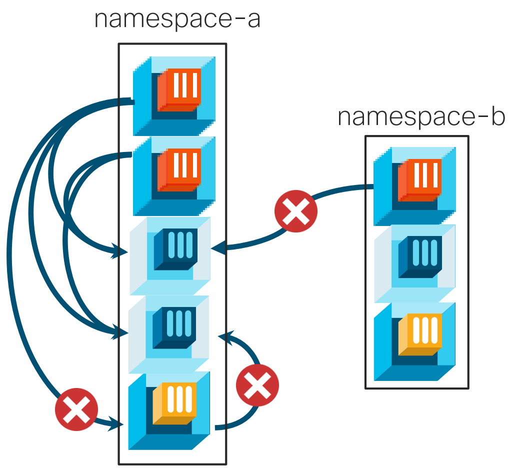
- Aci Containers Openvswitch (ACO, aci-container-openvswitch)
  - DaemonSet composed of 3 containers running on every node
  - It is the Open vSwitch enforcing the required networking and security policies provisioned through the OpFlex agent

```
[root@dom-master1 ~]# oc get daemonsets --namespace=aci-containers-system |grep vswitch
NAME                          DESIRED   CURRENT   READY   UP-TO-DATE   AVAILABLE   NODE SELECTOR   AGE
aci-containers-openvswitch    2         2         2       2            2           <none>          223d
```

ACI and Kubernetes Security Model

Support for Network Policy in ACI

- Specification of how selections of pods are allowed to communicate with each other and other network endpoints.
- Network namespace isolation using defined labels
  - directional: allowed ingress pod-to-pod traffic
  - filters traffic from pods in other projects
  - can specify protocol and ports (e.g. tcp/80)

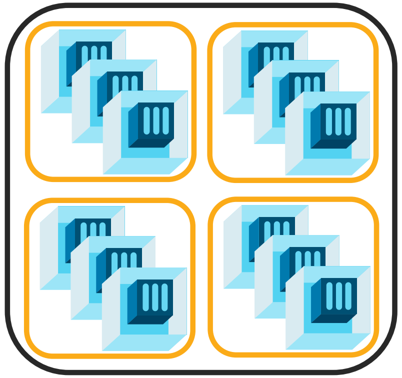


Policy applied to namespace: namespace-a

```
kind: NetworkPolicy
apiVersion: extensions/v1beta1
metadata:
  name: allow-red-to-blue-same-ns
spec:
  podSelector:
    matchLabels:
      type: blue
  ingress:
  - from:
    - podSelector:
        matchLabels:
          type: red
```

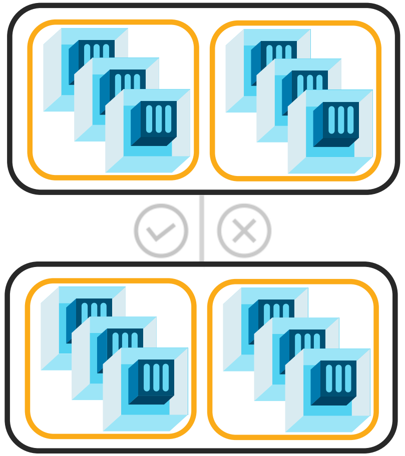
Mapping Network Policy and EPGs

Cluster Isolation



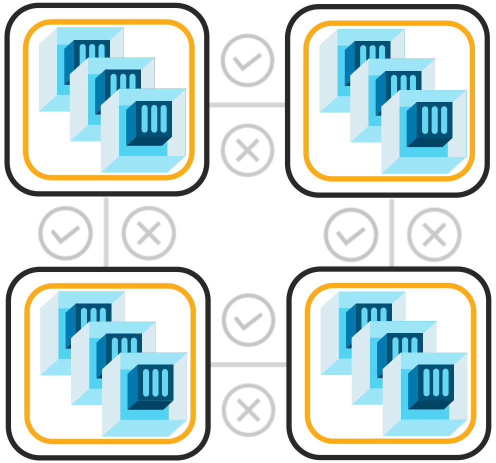
Single EPG for entire cluster.
(Default behavior)
No need for any internal contracts.

Namespace Isolation



Each namespace is mapped to its own EPG.
Contracts for inter-namespace traffic.

Deployment Isolation

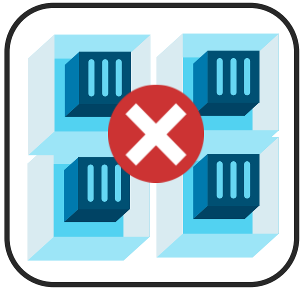


Each deployment mapped to an EPG
Contracts tightly control service traffic

Key Map | EPG | NetworkPolicy | ✓ | ✗ Contract

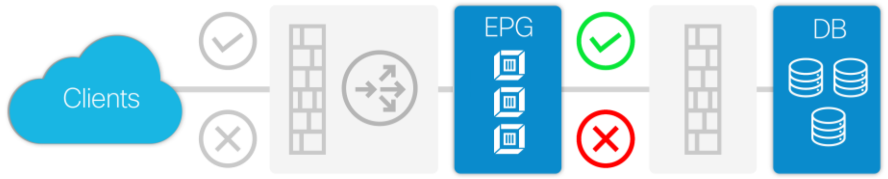
Dual level Policy Enforcement by ACI

Both Kubernetes Network Policy and ACI Contracts are enforced in the Linux kernel of every server node that containers run on.



Native API Default deny all traffic

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: default-deny
spec: podSelector: {}
policyTypes:
- Ingress
- Egress
```



Containers are mapped to EPGs and contracts between EPGs are also enforced on all switches in the fabric where applicable.

Both policy mechanisms can be used in conjunction.

Demo 1
Deploying an Application

Demo 2
Placing PODs/Namespaces into EPGs

Exposing Services

The extern_dynamic subnet

- Defined in acc-provision configuration file
- An IP address will be automatically selected from this subnet to expose your service outside of the k8s cluster/fabric
- Expose the service as “LoadBalancer” (as per kubernetes standard)
- The extern_dynamic subnet is not associated to a BD: You need to configure your external router with static routes toward your L3OUT for this subnet

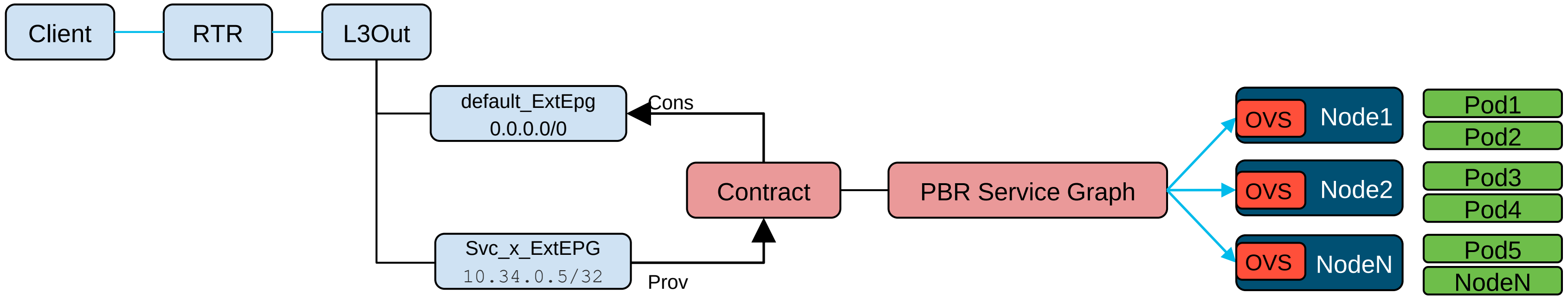
```
cisco@k8s-01:~/demo/guestbook1$ kubectl --namespace=guestbook get svc frontend
NAME       CLUSTER-IP    EXTERNAL-IP   PORT(S)        AGE
frontend   10.37.0.124   10.34.0.5     80:32677/TCP   5h
```

extern_dynamic

Service Graphs and PBR

Every time a service is exposed the ACI CNI controller will deploy:

- An External EPG with a /32 match for the Service IP
- A new contract between the svc_ExtEPG and the default_ExtEPG*
- A Service Graph with PBR redirection containing every node where an exposed POD is running

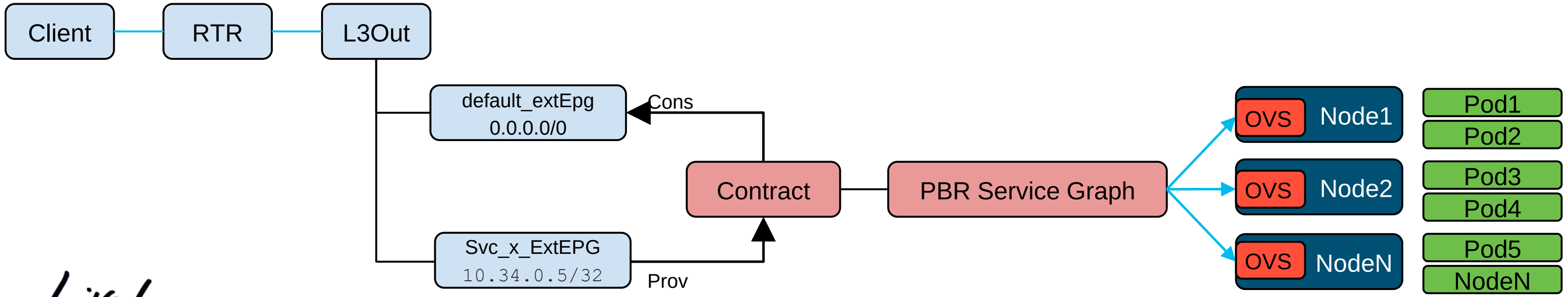


* defined in the acc-provision config file

Service Graphs and PBR – Packet walk

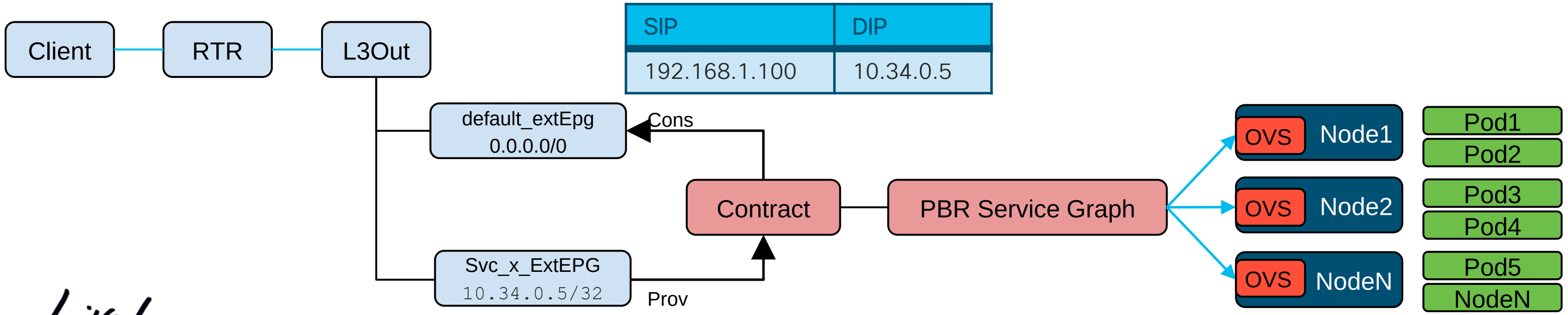
1. Client send a request to 10.34.0.5, ACI performs Longest Prefix Match (LPM) on the SIP and classify the traffic in the default_extEPG
2. ACI does a routing lookup for 10.34.0.5, IP does not exist in the fabric, we should route it out however LPM places it in the Svc_x_ExtEPG
3. PBR redirection is triggered and the traffic is LoadBalanced by the fabric to one of the nodes

SIP	DIP
192.168.1.100	10.34.0.5



Service Graphs and PBR – Packet walk

1. Client send a request to 10.34.0.5, ACI performs policy look up on the SIP and classify the traffic in the default_extEPG
2. ACI performs policy lookup for 10.34.0.5 and matches it in Svc_x_ExtEPG activating the contract between the two EPGs
3. PBR redirection is triggered and the traffic is LoadBalanced by the fabric to one of the nodes



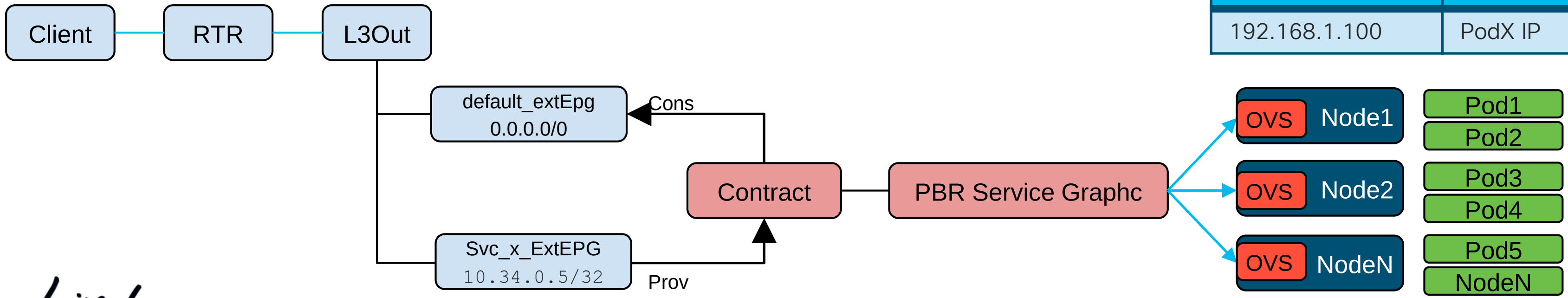
SIP	DIP
192.168.1.100	10.34.0.5

Service Graphs and PBR – Packet walk

4. The K8S node is not expecting any traffic directed to the external service IP so OVS will perform NAT as required
5. If there are multiple POD on a single node OVS will perform a second stage LB to distribute the load between Pods running on the same node

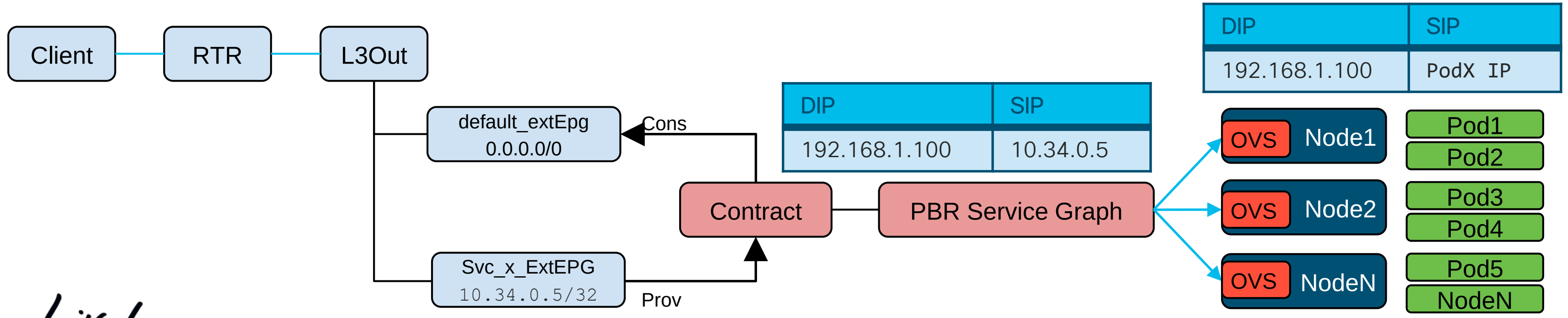
SIP	DIP
192.168.1.100	10.34.0.5

SIP	DIP
192.168.1.100	PodX IP



Service Graphs and PBR – Packet walk

4. PodX replies to the client
5. OVS restore the original external Service IP
6. PBR redirection is not triggered since the source EPG is the Shadow EPG of the PBR node
7. Traffic is routed back to the client (and is permitted by the contract)



DIP	SIP
192.168.1.100	10.34.0.5

DIP	SIP
192.168.1.100	PodX IP

Demo 3
Exposing Services

Exposing a service

- Simply choose the LoadBalancer ”type” in the service definition
- The ACI CNI plug in will:
  - Automatically pick a free IP from the extern_dynamic subnet
  - Create the ExtEPG
  - Create contracts
  - Create PBR redirection rules
  - Deploy the service graph

Scalability

- Currently the scalability of exposing external service with PBR is limited by the number of external EPGs per L3OUT.
- As of ACI 4.2.2 we supports 500 external EPGs per L3 OUT per leaf *
- This is a soft limit and will increase with time
- But we want more! So?

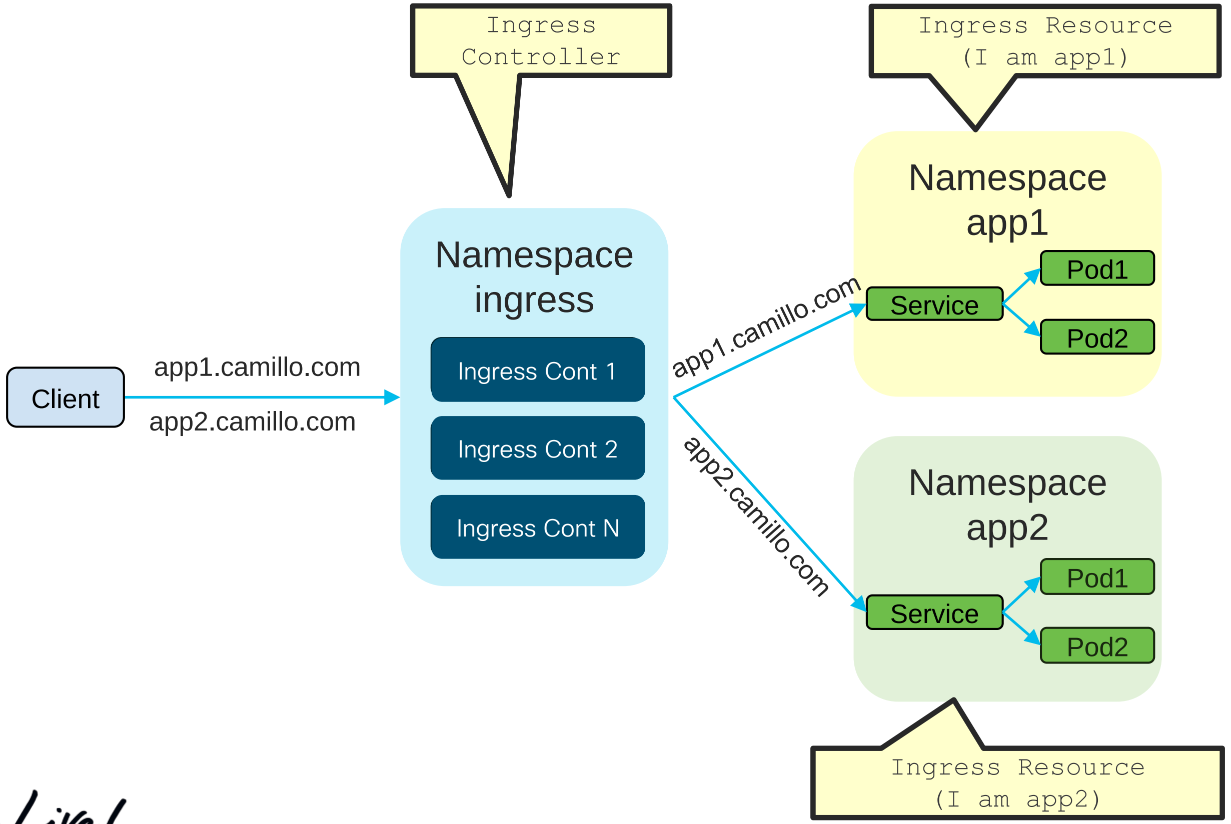
*For details check:
https://www.cisco.com/c/en/us/td/docs/switches/datacenter/aci/apic/sw/4-x/verified-scalability/Cisco-ACI-Verified-Scalability-Guide-422.html

Scaling External Services with Ingress

Kubernetes – Ingress

- Composed of two parts:
  - Ingress Resources: collection of rules that defines how inbound connections can reach the internal cluster services.
  - Ingress controller: responsible for fulfilling the Ingress, usually with a virtual loadbalancer (nginx, ha-proxy)
- Ingress controller can be shared between multiple namespaces
- It can be configured to give services externally-reachable URLs, load balance traffic, terminate SSL, offer name based virtual hosting etc…
- Easy integration with DNS: configure a wildcard DNS record (*.camillo.com) pointing to the IP of the ingress controller

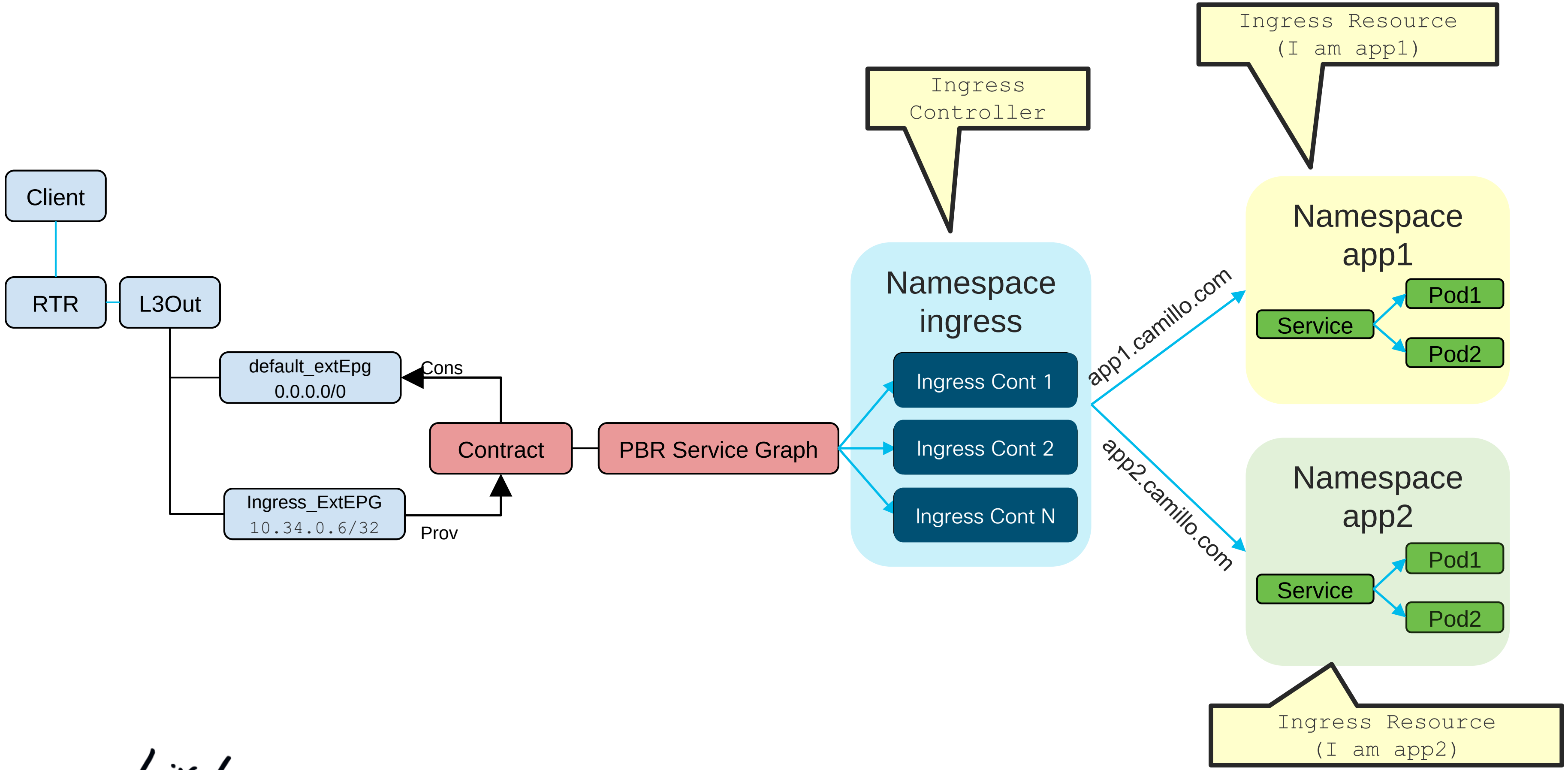
Kubernetes – Ingress



ACI and Kubernetes Ingress

- Expose the Ingress Controller via Service Graph with PBR
- A single Service Graph/ExtEPG can now host as many services as we want
- Ingress Controller can be scaled as needed
- If you create a dedicated EPG for ingress you need the following contracts:
  - All the contracts used in kube-default (remember contract inheritance)
  - Consume: Kube-API, Ingress need to be able to speak with the Kube API server
  - Consume: any required ports between Ingress Controller and the service you wan to expose

ACI and Kubernetes Ingress



ACI and Kubernetes External Services - Summary

- Two options (can be used at the same time even for the same service)
  - Exposing services via ingress
  - Exposing up to 500 services directly with Service Graph with PBR

POD Source NAT

POD Networking - Recap

- During the ACI CNI installation a Bridge Domain with a dedicated subnet is created for your POD Networking.
- Every POD that is created in the Kubernetes cluster will be assigned an IP address from the POD Subnet.
- In most cases this is an advantage to other CNI implementation:
  - the POD IP/Subnet is equivalent to any other IP/Subnet in ACI
  - The POD/IP Subnet can communicate directly with any other IP/Subnet directly connect to ACI
  - The POD/IP Subnet can communicate directly with any other IP/Subnet outside of ACI via a standard L3OUT
  - Your PODs are first class citizen in ACI!

However there are some challenges

Challenge 1: External Firewall Configuration

- The POD IP is ephemeral:
  - It is not possible to predict what IP address a POD will be assigned.
  - It is not possible to manually assign an IP to a POD
- This standard Kubernetes behavior but it does not work well with firewalls:
  - Is not possible to configure the firewalls ACLs based on the POD IPs as the POD IP can change at any time.
- The same Kubernetes cluster can host multiple applications:
  - It is not possible to use the POD subnet as security boundary

Challenge 2: POD Subnet Routing

- The POD Subnet is, most likely, a private subnet
- In certain scenarios a POD might need communicated with an external environment (i.e. internet) and the POD IP address needs to be natted

ACI CNI SNAT to the rescue!

- ACI CNI now support distributed SNAT
  - POD Initiated traffic can be natted to an IP address selected by the user
- Kubernetes user can decide:
  - SNAT IP: Single IP or Range
  - Ability to apply SNAT Policy at different levels:
    - Cluster Level: connection initiated by any POD in any Namespaces is natted to the selected SNAT IP
    - Namespace: connection initiated by any POD in the selected Namespaces is natted to the selected SNAT IP
    - Deployment: connection initiated by any POD in the selected Deployment is natted to the selected SNAT IP
    - LoadBalanced Service: connection initiated by any POD mapped to an external Service of Type LoadBalance are natted to the external Service IP.

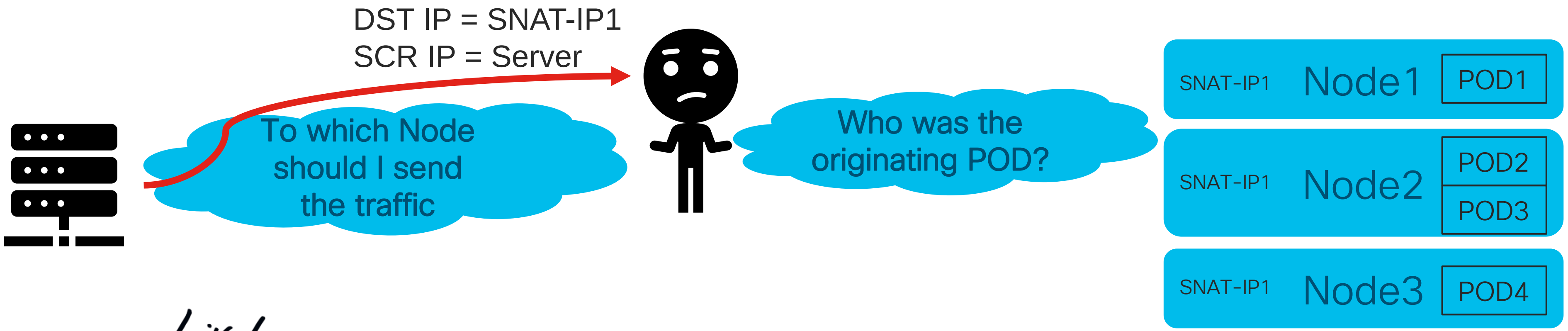
ACI CNI SNAT Design Overview

ACI CNI SNAT Architecture Overview

- Kubernetes SNAT configuration is done via Custom Resource Definitions (CRD)
- Connection Tracking and Source NAT in performed by OpenVSwitch (OVS), distributed across all the cluster nodes
- ACI CNI's Opflex agent programs OVS flows (SNAT Rules) on each host
- ACI CNI Controller programs ACI Service Graph to send return traffic back to the cluster (more on this later)
- SNAT is applied only to traffic exiting the cluster via an L3OUT
  - Not supported inside the same

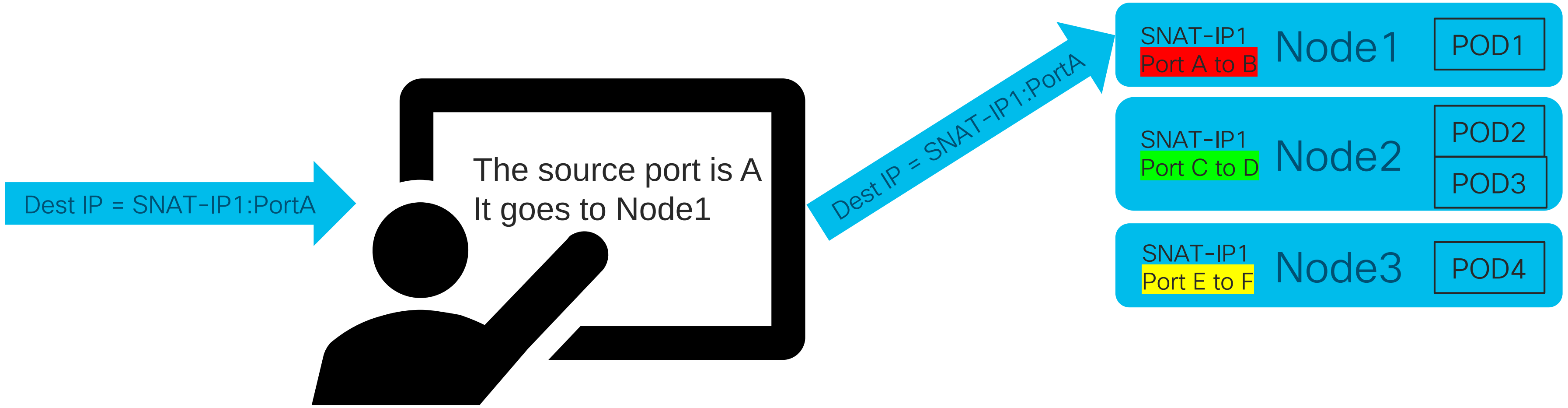
ACI CNI SNAT IP Allocation

- SNAT IPs are allocated per scope:
  - Kubernetes cluster
  - Namespace
  - Deployment
  - Service (Re-Using the external service IP)
- A "Scope" can exist on multiple Kubernetes Nodes and the same SNAT IP can be used on multiple nodes and this requires some clever thinking for the return traffic because:



ACI CNI SNAT IP Allocation (cont.)

- To solve this issue a unique range of TCP and UDP ports for the SNAT IP is allocated to each node: this will allow ACI CNI to know which node is hosting the POD that initiated the connection



Note: ICMP is not supported

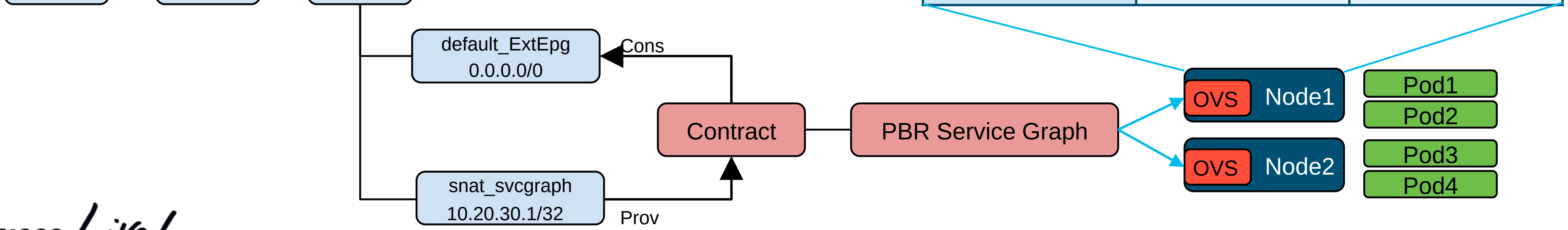
- Since ICMP has no port is not possible to distinguish between different PODs.
- ICMP (Ping) is hence not supported

ACI CNI SNAT Packet Walk

Service Graphs for SNAT

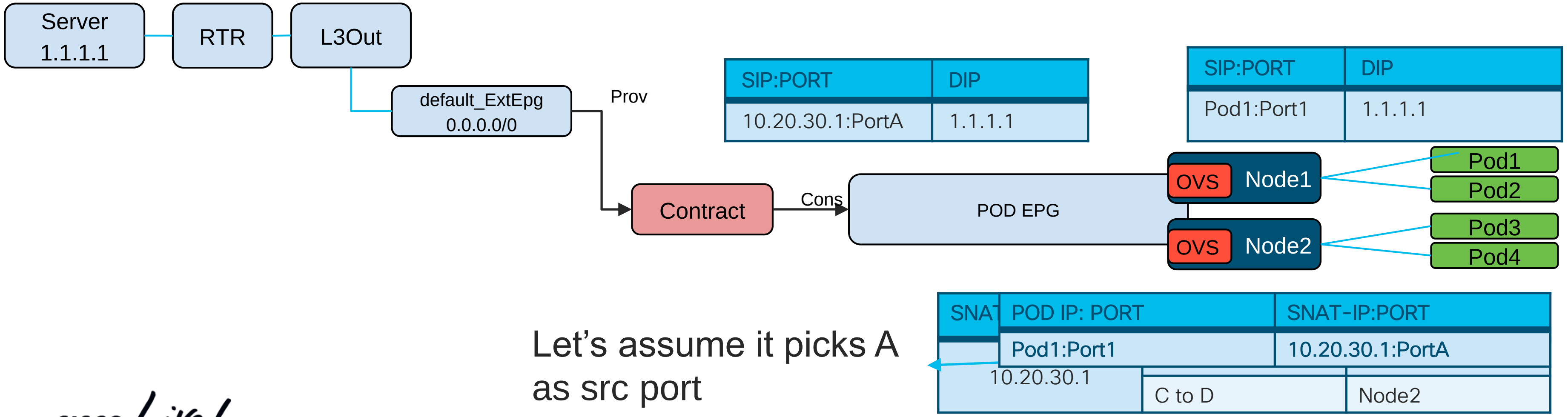
- The first time a SNAT Policy is configured the aci-container-controller will:
  - Create a snat_svcgraph externalEPG[1]
  - Add the SNAT IP under the snat_svcgraph
  - Create a new snat_svcgraph contract between the snat_svcgraph ExtEPG and the default_ExtEPG
  - Create a Service Graph with PBR redirection containing every node of the Cluster
  - Allocate a port range to each node for the SNAT IP

SNAT IP	Range	NODE
10.20.30.1	A to B	Node1
	C to D	Node2



Service Graphs for SNAT – Packet walk egress

1. POD1 initiate a flow to 1.1.1.1
2. OVS checks ACI contract/K8S Policies are allowing the communication
3. OVS allocate a SNAT-IP:PORT from the node range
4. OVS NATs the POD IP to the SNAT-IP and track the mapping between the POD-IP:Port and the SNAT-IP:SNAT-Port
5. Normal ACI routing takes place



SIP:PORT	DIP
10.20.30.1:PortA	1.1.1.1

SIP:PORT	DIP
Pod1:Port1	1.1.1.1

POD IP: PORT	SNAT-IP:PORT
Pod1:Port1	10.20.30.1:PortA

SNAT		
10.20.30.1	C to D	Node2

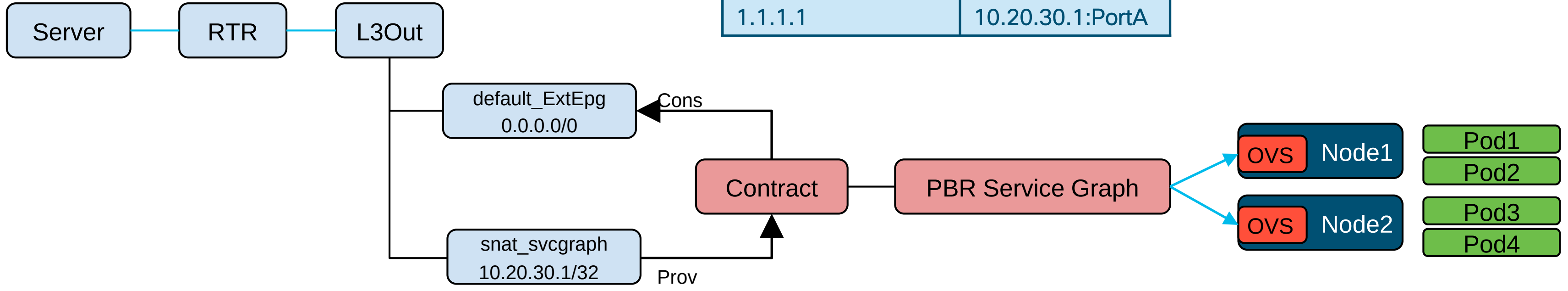
Let’s assume it picks A as src port

Service Graphs for SNAT – Packet walk ingress

1. Server send a reply to 10.20.30.1, policy lookup on the SIP (1.1.1.1) and classify the traffic in the default_extEPG
2. The destination IP 10.20.30.1 matches the snat_svcgraph resulting in the contract between these groups to be applied
3. PBR redirection is triggered and the traffic is LoadBalanced by the fabric to one of the nodes

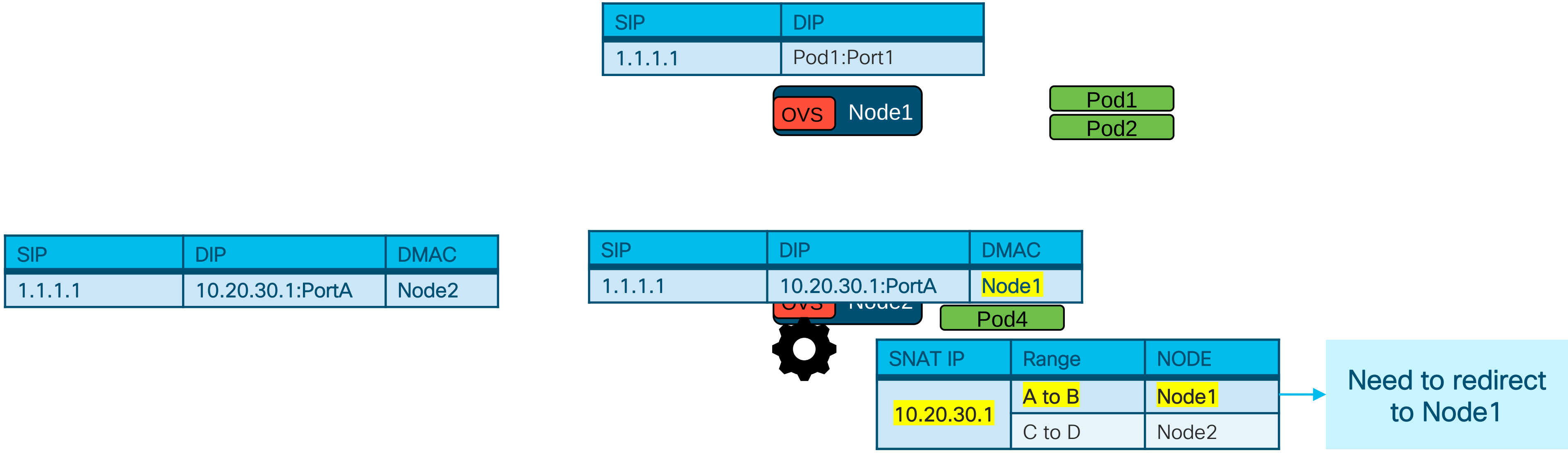
SIP	DIP
1.1.1.1	10.20.30.1:PortA

SIP	DIP
1.1.1.1	10.20.30.1:PortA



Service Graphs for SNAT – Packet walk ingress

4. The destination POD might not be on the node selected by the PBR hashing algorithm. If the pod does not reside on the node, an OVS rules bounce the traffic to the actual destination node by replacing the destination MAC address to point to the pod's node MAC address.
5. Once the traffic reaches the destination Node, OVS connection tracking rules translate the SNAT-IP:SNAT-Port to the pod's Source-IP:Port.



SIP	DIP
1.1.1.1	Pod1:Port1

SIP	DIP	DMAC
1.1.1.1	10.20.30.1:PortA	Node2

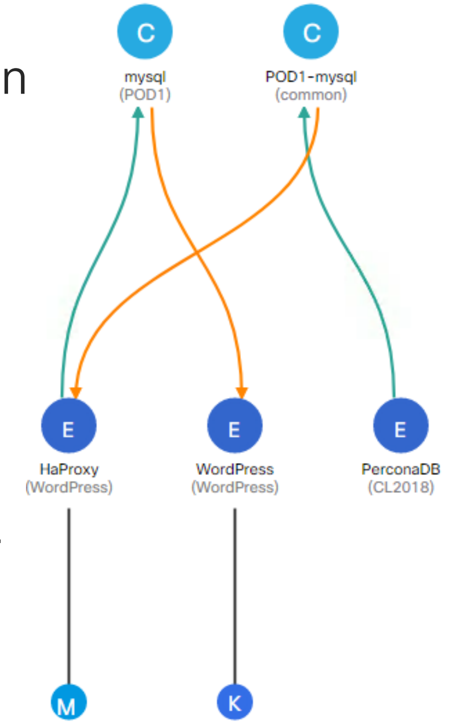
SIP	DIP	DMAC
1.1.1.1	10.20.30.1:PortA	Node1

SNAT IP	Range	NODE
10.20.30.1	A to B	Node1
	C to D	Node2

Container to Non-Container Communications

Container to Non-Container Communications

- In production environments is preferred, for example, to run services like high performance databases as VMs or Bare Metal Servers
- This calls for the ability to easily provide communication between K8S POD and VMs/Bare Metal
- Simply deploy a contract between your EPGs, ACI will do the rest!
- This works for any VMM domain and Physical Domains, for example you can have a Container Domain using VXLAN speaking with a Microsoft SCVMM Domain using VLAN.

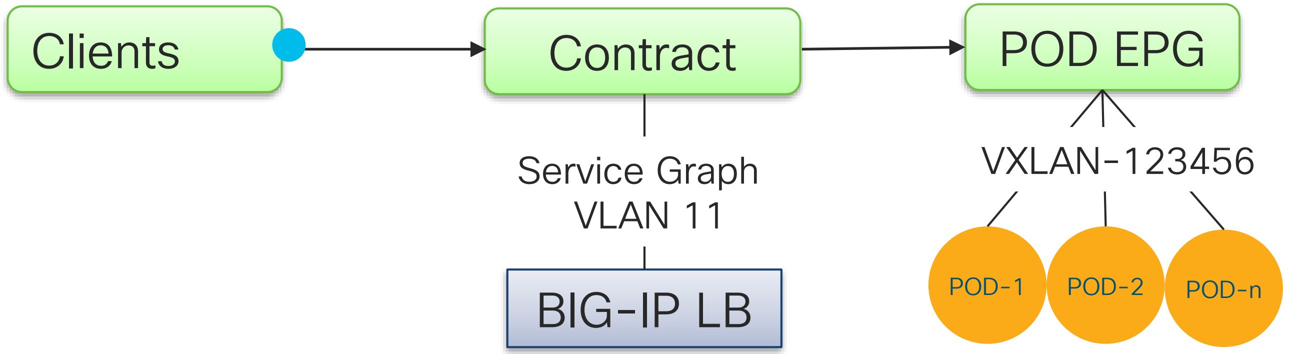


Container to Non-Container Communications: F5 Integration with ACI CNI

F5 As an Ingress LoadBalancer

- It is possible to use a Physical or Virtual BIG-IP to expose your services
- Runs a k8s-bigip-ctlr to send config to the BIG-IP
- Two mode of operation:
  - LoadBalance the traffic directly between the PODs: the BIG-IP needs direct connectivity to the POD subnet.
  - LoadBalance the traffic to the NodePort of the service that you are exposing, this is suboptimal as all the nodes are added in the LB pool so you can send traffic to nodes where no PODs are present and you have to trombone the traffic around the cluster

ACI – F5 Design

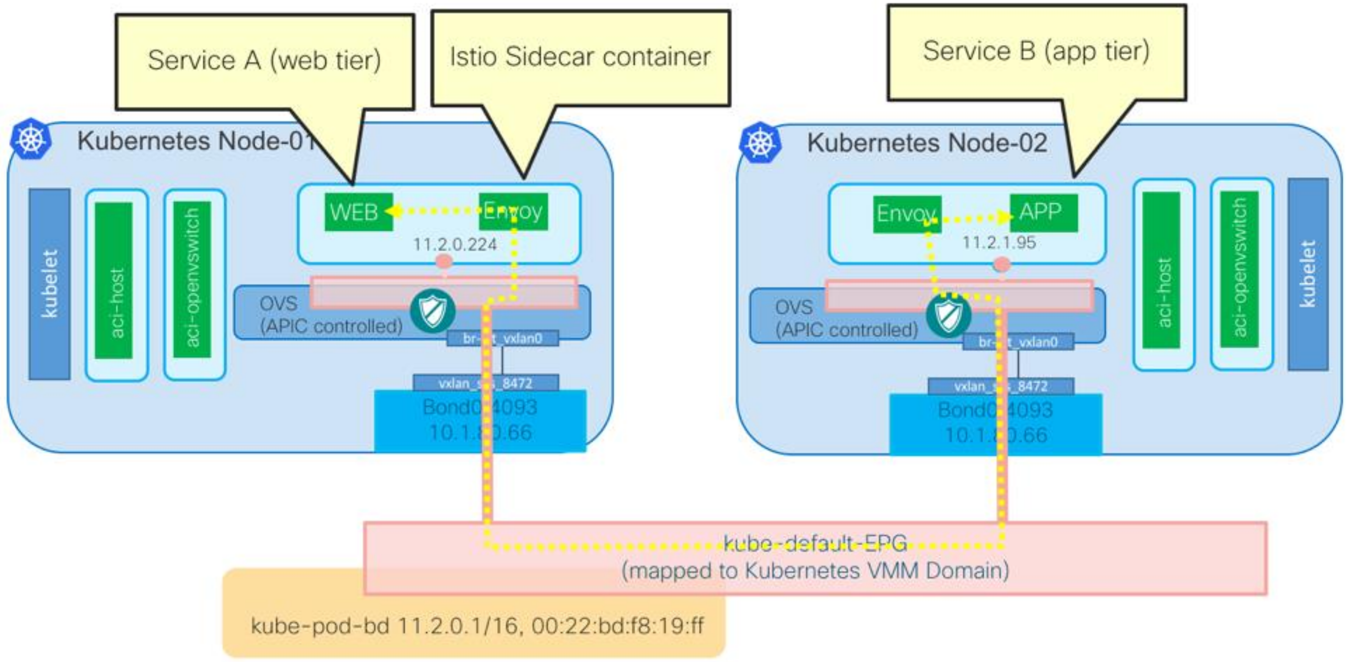


Demo 5
F5 and ACI CNI

What about Service Meshes? (ISTIO)

ISTO is Transparent

- The sidecar proxy sits inside the original application POD and is completely transparent to our CNI plugin as per the picture below.



Kubernetes Cluster Node Failure

Kubernetes Cluster Node Failure Detection

- Kubernetes Monitors by default all the node in the clusters
- Depending on the configuration, node failure detection and container restart can take from ~50s to 5min. This will depend on your specific configuration.
- Once a node is detected as NotReady (failed) the aci-container-controller will update the ACI configuration as required i.e. a failed node will be removed from the PBR redirection policy

ACI CNI redundancy during node failure

aci-containers-host and aci-containers-openvswitch

- DataPlane of the CNI Plugin
- Start and Stop with the Node
- If isolated from the network they will try to reconnect to the leaf

ACI CNI redundancy during node failure

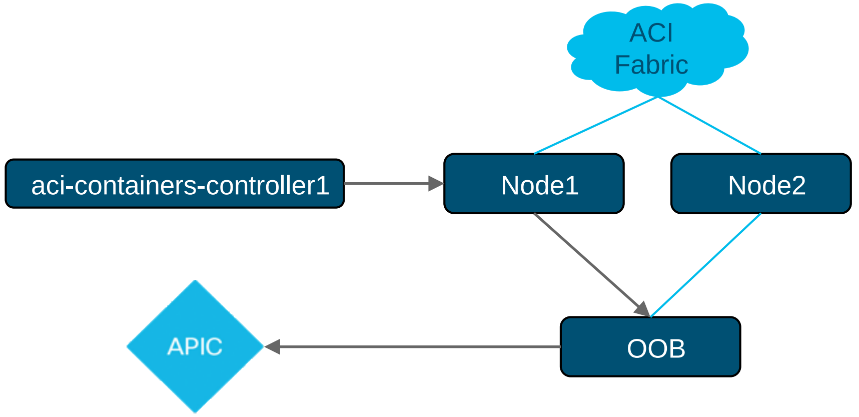
aci-containers-controller (acc)

- Stateless
- Does not sit in the data-path
- In case of failure k8s will restart it on a different node

ACI CNI redundancy common configuration mistake

aci-containers-controller

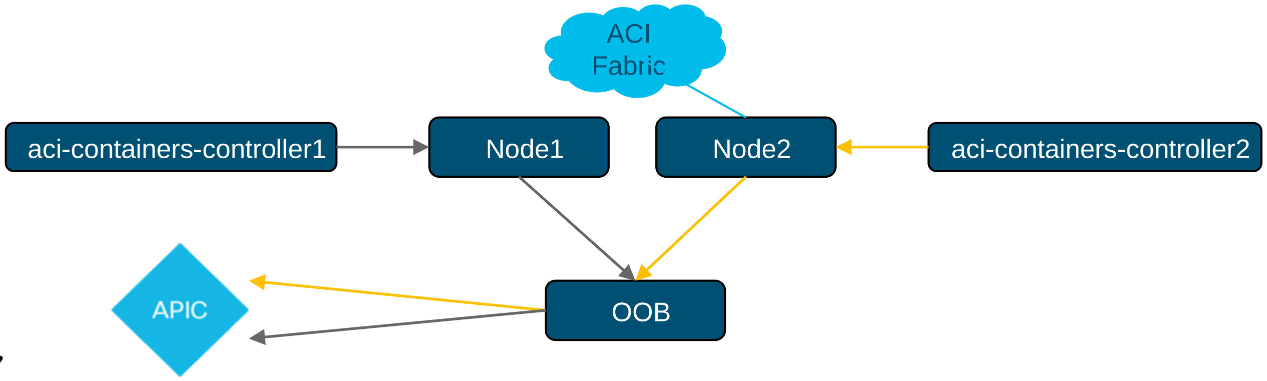
- Node connects to OOB and ACI Fabric
- K8S Cluster communications are happening over the ACI Fabric
- aci-containers-controller communicates with APIC via OOB



ACI CNI redundancy common configuration mistake

aci-containers-controller

- Node 1 losses connectivity with the ACI Fabric (interface down)
- K8S master will detect node1 as lost and restart acc on Node2
- The old instance of acc1 is still running and will keep injecting the old config, overwriting the configuration changes pushed by acc2
- When designing your network ensure that acc communication with the APIC goes trough the fabric



What is coming

ACI CNI Upcoming Features

- Open Shift 4.2 support
- Mixed form factor (VMs and Bare Metals in the same cluster)
- POD and Node BD in common tenant support
  - Support multiple cluster in the same tenant
- ACI CNI in Public Cloud
  - AWS with OpenShift 4.2

How can I build my own lab?

Cisco Container Platform

certified kubernetes

Turnkey Solution
For Production-Grade Container Environments

Native Kubernetes (100% Upstream)
Direct updates and best practices from open source community

Hybrid Cloud Optimized
E.g.: Google, AWS, ...

Integrated
Networking | Management | Security | Analytics

Flexible Deployment Model
VM | Bare metal <-> HX, UCS, ACI | Public cloud

Easy to acquire, deploy and manage | Open and consistent | Extensible platform | World-class advisory and support

aci_kubeadm

Not officially supported

- Set of ansible scrips to deploy a single master cluster using ACI CNI plugin
- Open Source (not supported by TAC/Cisco etc…)
- Optionally can clone VM templates and configure everything providing a 1-Click deployment solution for your lab
- https://github.com/camrossi/aci_kubeadm

Yes it is me… Did I mentioned is not officially supported? ☺

Complete your online session survey

- Please complete your session survey after each session. Your feedback is very important.
- Complete a minimum of 4 session surveys and the Overall Conference survey (starting on Thursday) to receive your Cisco Live t-shirt.
- All surveys can be taken in the Cisco Events Mobile App or by logging in to the Content Catalog on ciscolive.com/emea.

Cisco Live sessions will be available for viewing on demand after the event at ciscolive.com.

Continue your education

Demos in the Cisco campus

Walk-in labs

Meet the engineer 1:1 meetings

Related sessions

Thank you

You make **possible**