
cisco Live!
Let's go
#CiscoLive

The bridge to possible

Kubernetes Infrastructure Connectivity for ACI

Network Designs for the Modern Data Centre

Camillo Rossi
Technical Marketing Engineer

BRKDCN-2663

cisco Live!

Cisco Webex App

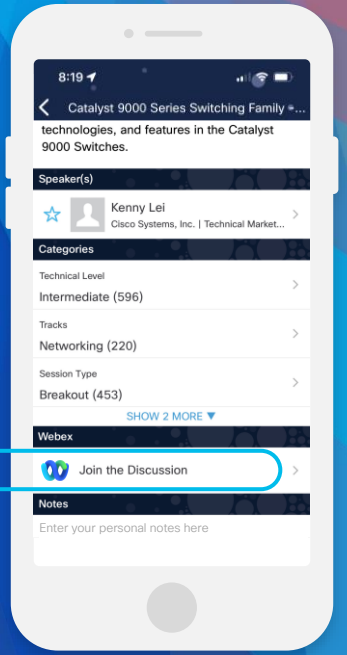
Questions?

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

How

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

Webex spaces will be moderated by the speaker until June 9, 2023.

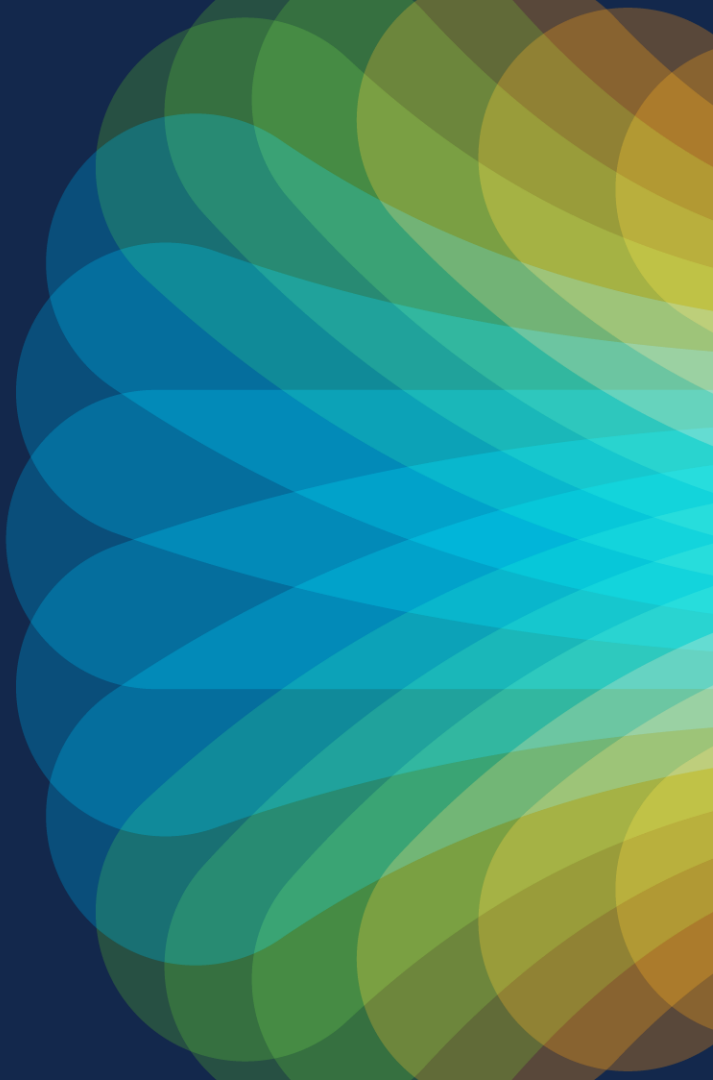


https://ciscolive.ciscoevents.com/ciscolivebot/#BRKDCN-2663

Agenda

- **Kubernetes Refresh**
- Kubernetes Network Challenges
- ACI-CNI
- BGP Based Architecture
  - Calico, Cilium and Kube-Router
  - Automation and Visibility
- Which solution is right for me?
- Q&A

Kubernetes Refresh



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 – Annotations

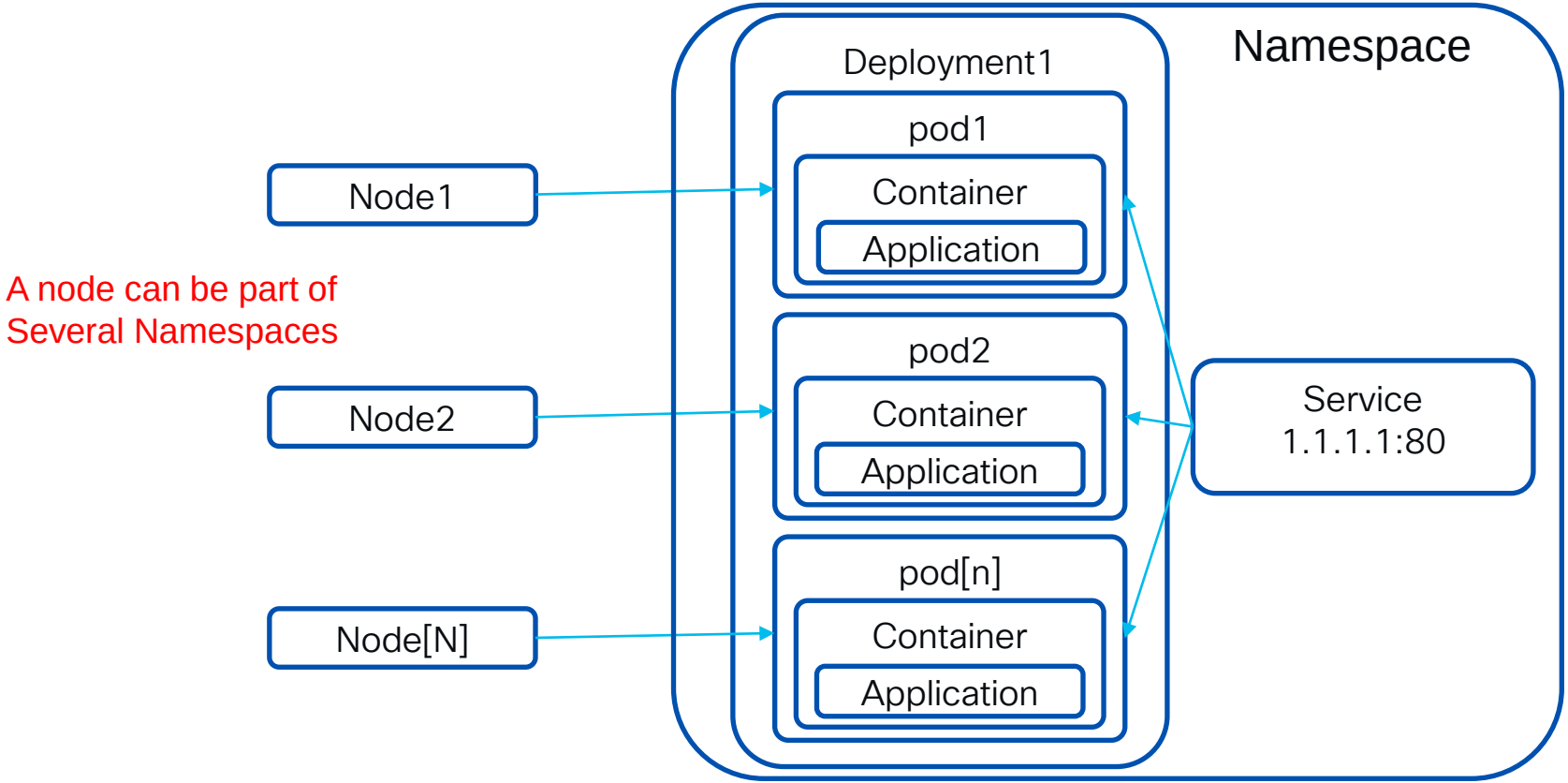
- Similar to labels but are NOT used to identify and select object

```
[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

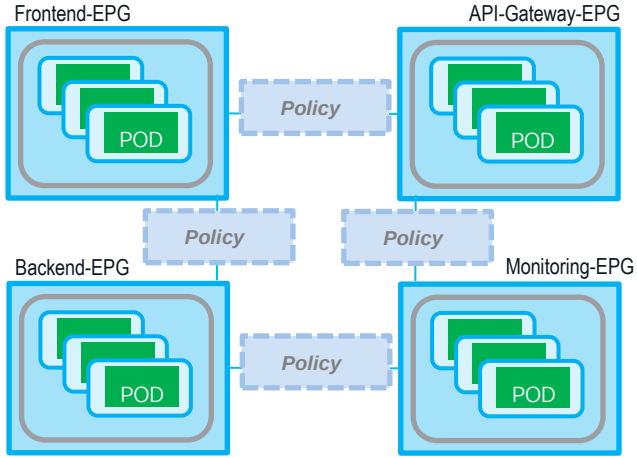


Agenda

- Kubernetes Refresh
- **Kubernetes Network Challenges**
- ACI-CNI
- BGP Based Architecture
  - Calico, Cilium and Kube-Router
  - Automation and Visibility
- Which solution is right for me?
- Q&A

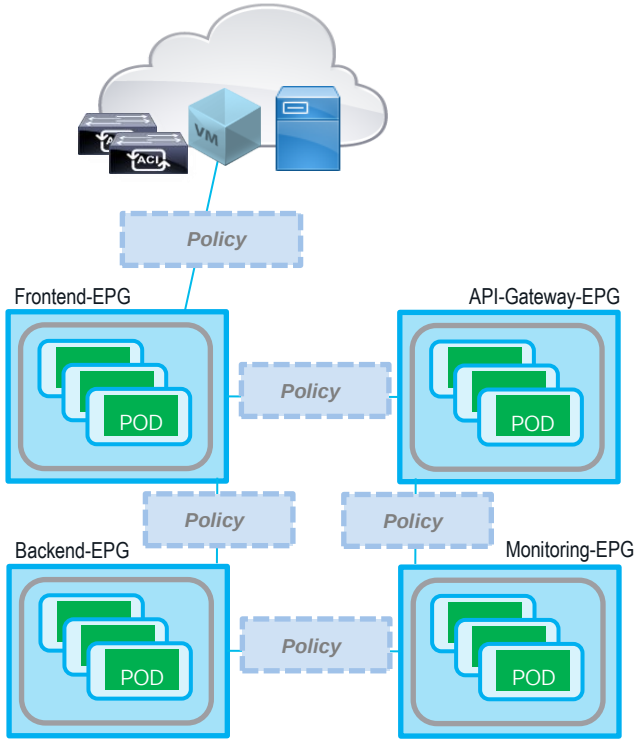
Segmentation

- 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

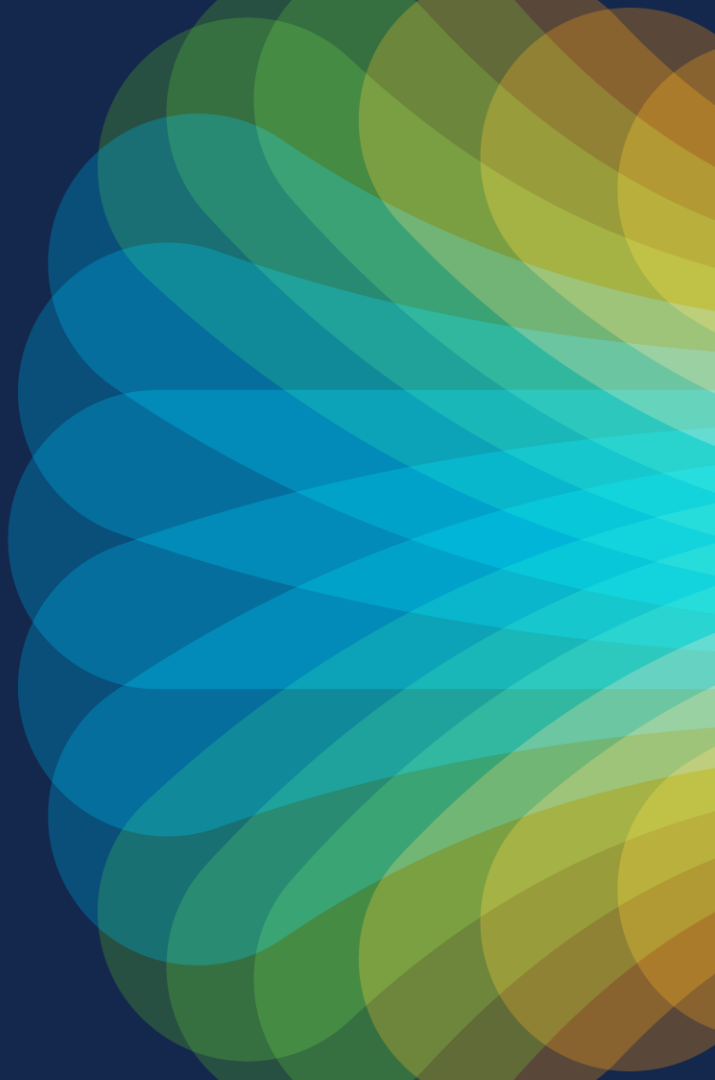
- Non-Cluster endpoints communicating with Cluster:
  - Exposing external services, how? NodePort? LoadBalancer?
  - Scaling-out ingress controllers?
- Cluster endpoints communicating with non-cluster endpoints:
  - POD access to external services and endpoints
- Cluster accessing shared resources like Storage



Agenda

- Kubernetes Refresh
- Kubernetes Network Challenges
- **ACI-CNI**
- BGP Based Architecture
  - Calico, Cilium and Kube-Router
  - Automation and Visibility
- Which solution is right for me?
- Q&A

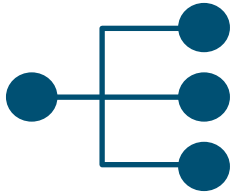
ACI-CNI
Solution Overview



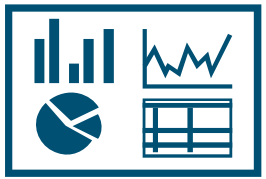
Why ACI-CNI for Application Container Platforms

Turnkey solution for node and container connectivity

Flexible policy: Native platform policy API and ACI policies



Hardware-accelerated: Integrated load balancing and Source NAT



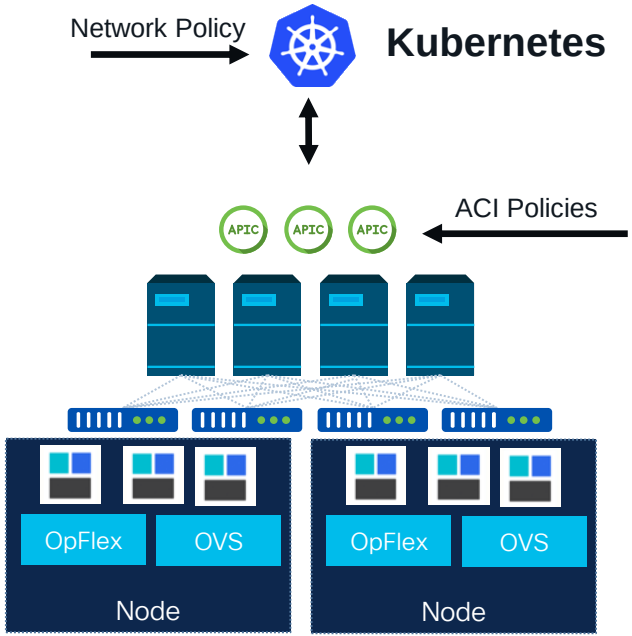
Visibility: Live statistics in APIC per container and health metrics

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

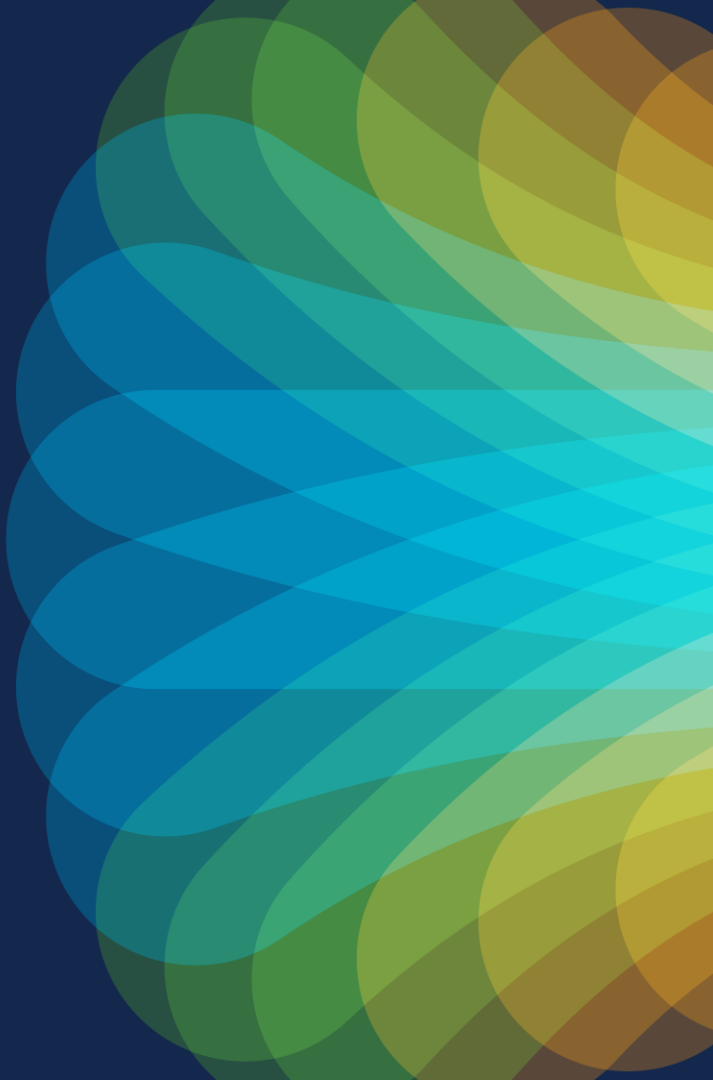
Fast, easy, secure and scalable **networking** for your Application Container Platform

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 K8s objects using annotations
- Consolidated visibility of K8s networking via VMM Integration



ACI-CNI 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.

acc-provision – configuration file (1)

```
aci_config:
  system_id: Kubernetes          # 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
      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
```

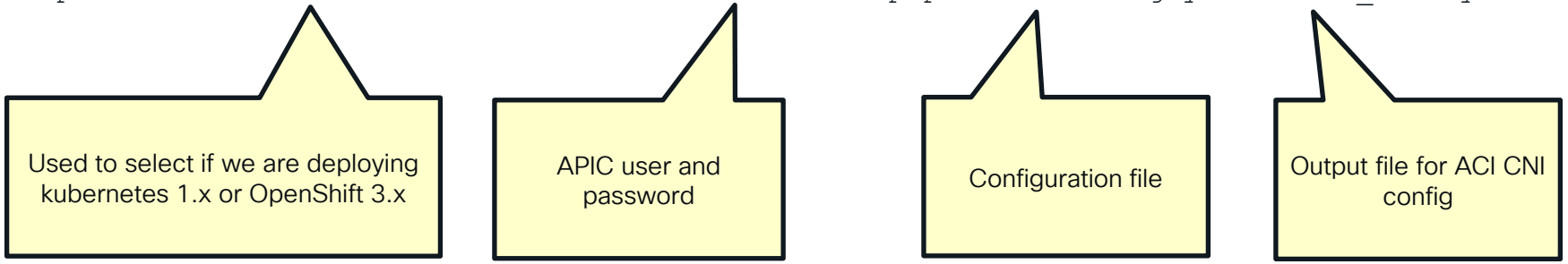
acc-provision – configuration file (2)

```
#
# 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
```

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-1.25 -a -u admin -p pass -c config.yml -o cni_conf.yml
```

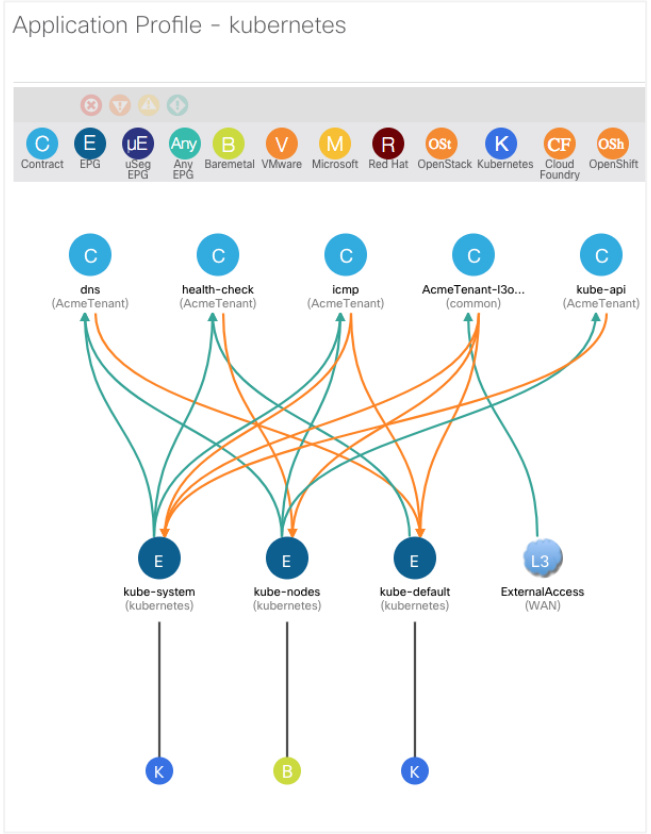


acc-provision will now create

EPGs for nodes and Pods

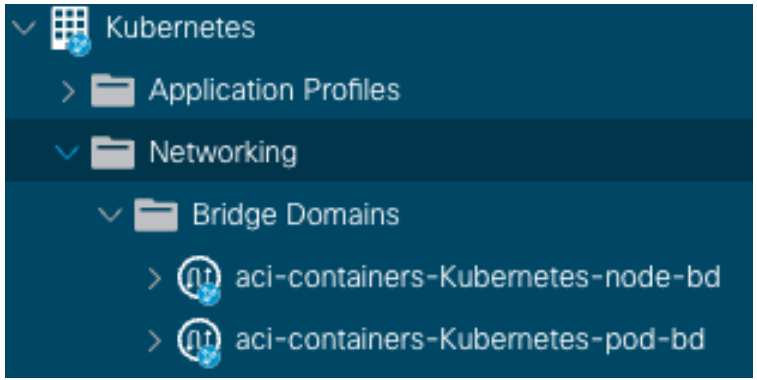
Within the tenant selected the provisioning tool creates a ‘Kubernetes’ Application Profile with three EPGs:

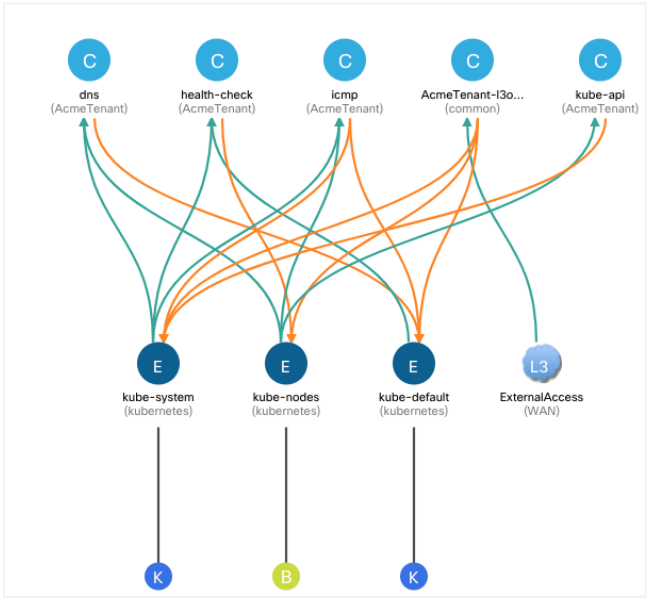
- for the node interfaces
- for the system PODs
- Default EPG for all containers on any namespace



BDs and Contracts

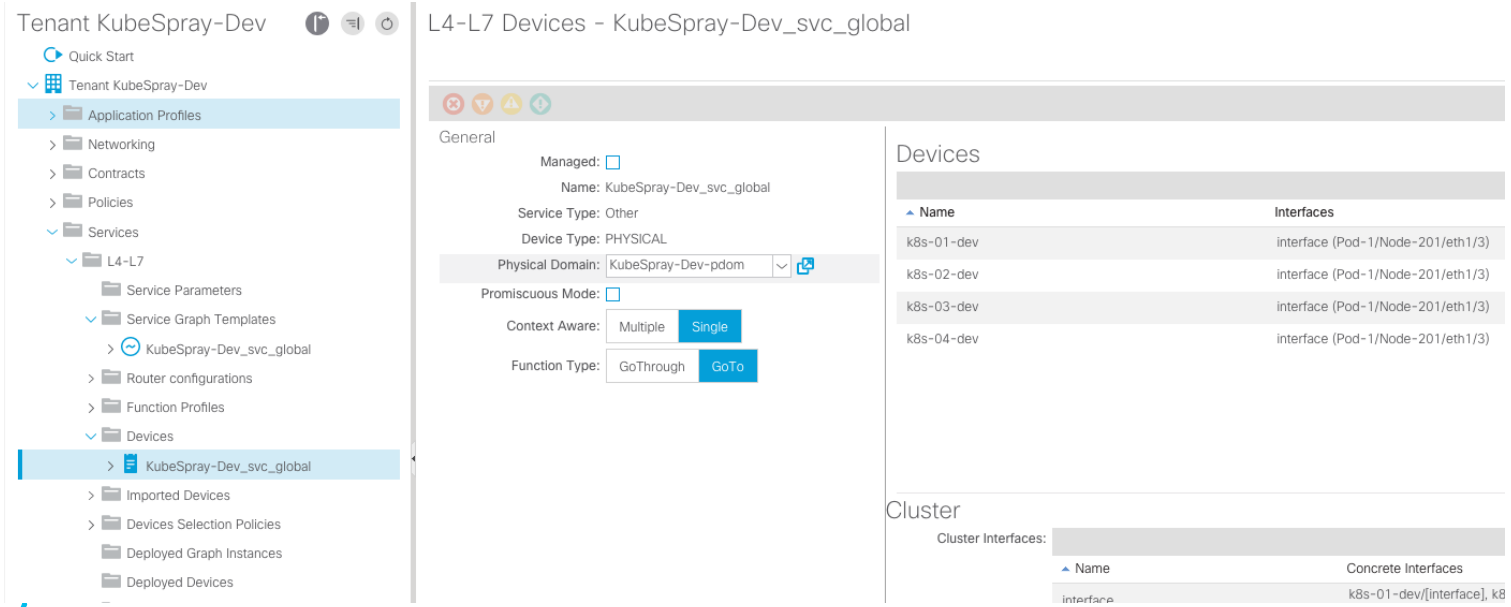
- minimum set of contracts to ensure basic cluster functionality and security





L4-L7 Devices

- Dynamically updated if nodes are added or removed from the k8s cluster
- Service Graph Template



For your reference

ACI CNI Plugin components

ACI CNI Plugin Components

For your reference

- 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

For your reference

- 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
```

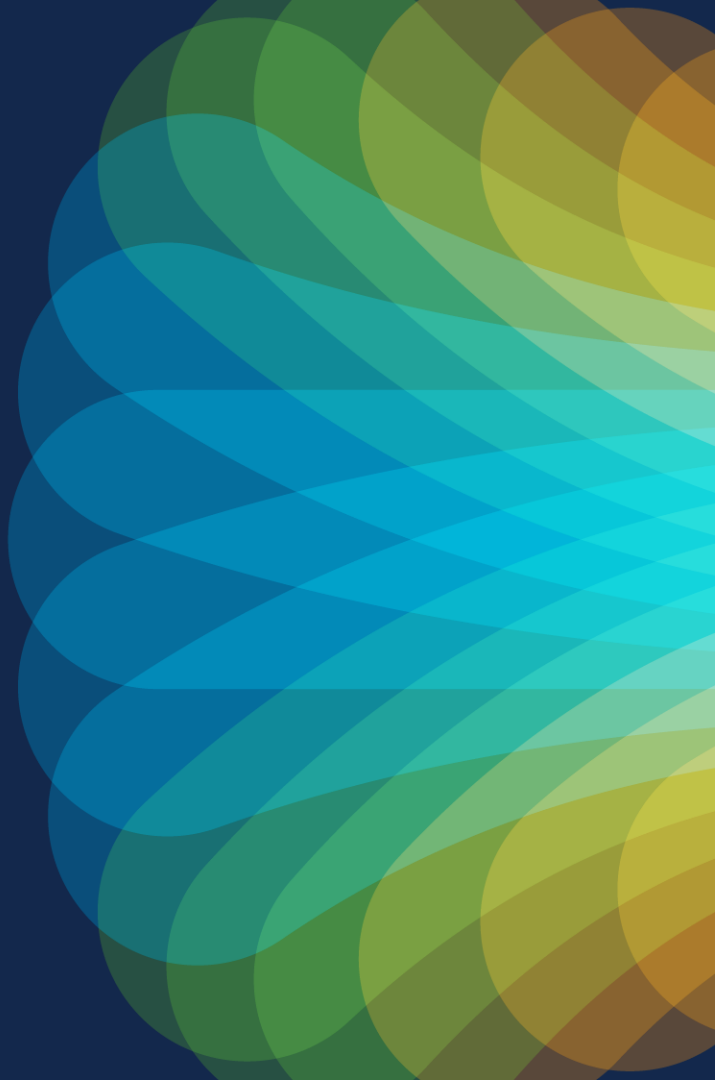
ACI CNI Plugin Components

For your reference

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

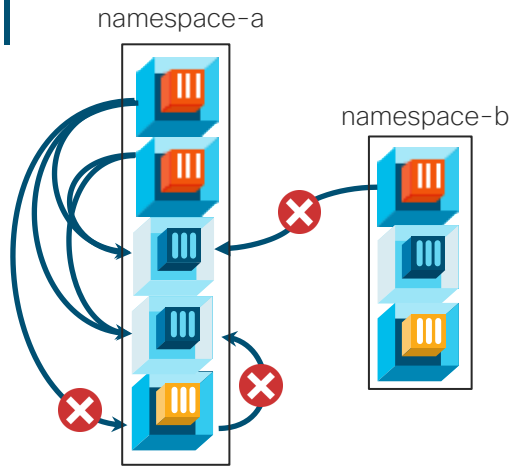
```
[root@dom-master1 ~]# oc get daemonsets --namespace=aci-containers-system |grep vswitch
NAME                          DESIRED   CURRENT   READY   UP-TO-DATE   AVAILABLE   NODE SELECTOR   AGE
aci-containers-openvswitch    2         2         2       2            2           <none>          223d
```

ACI-CNI:
K8s 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

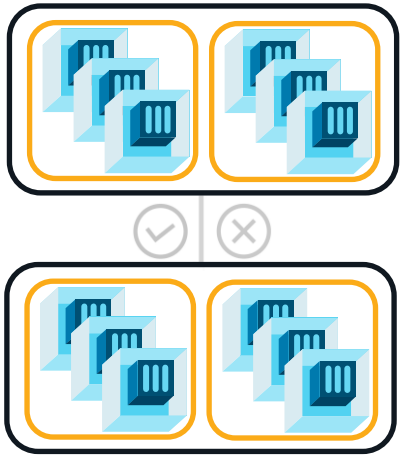
Cluster Isolation

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.

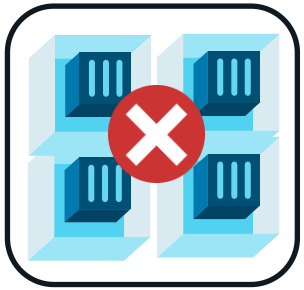
Deployment Isolation

Each deployment mapped to an EPG

Contracts tightly control service traffic

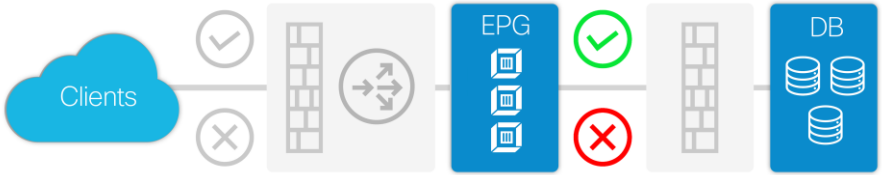
Dual level Policy Enforcement by ACI

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

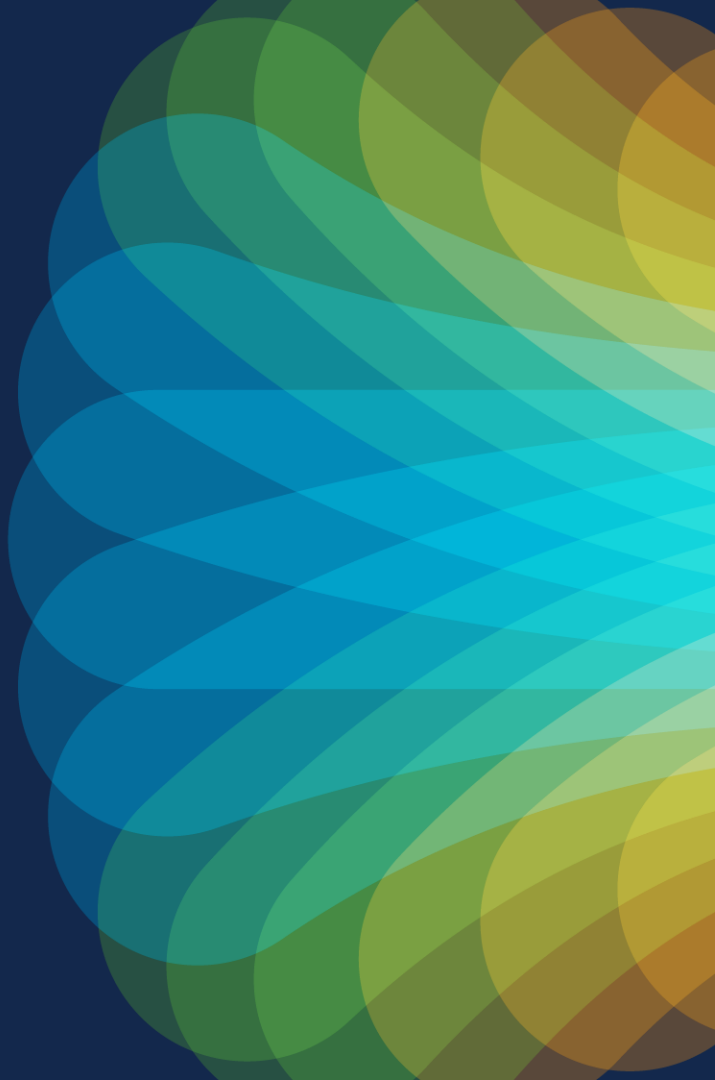
```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: default-deny
spec: podSelector: {}
policyTypes:
- Ingress
- Egress
```



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.

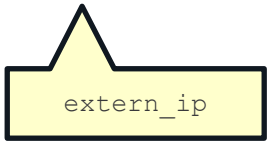
Exposing Services



Automated LoadBalancing

- Create a service of type “LoadBalancer” (as per K8s standard)
- ACI CNI will:
  - Allocate an external IP from a user-defined subnet
  - Deploy a Service Graph with PBR redirection to LoadBalance the traffic between any K8s Nodes that have PODs for the exposed service

```
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
```

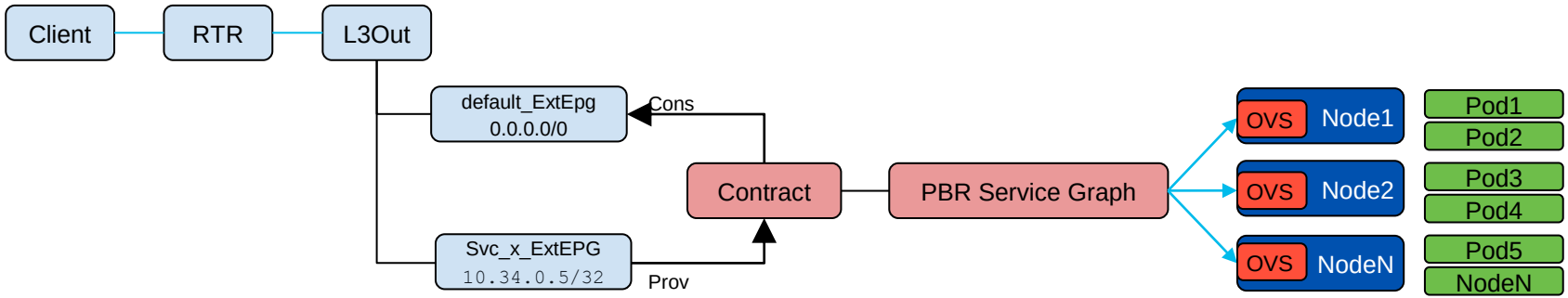


Service Graphs and PBR

For your reference

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



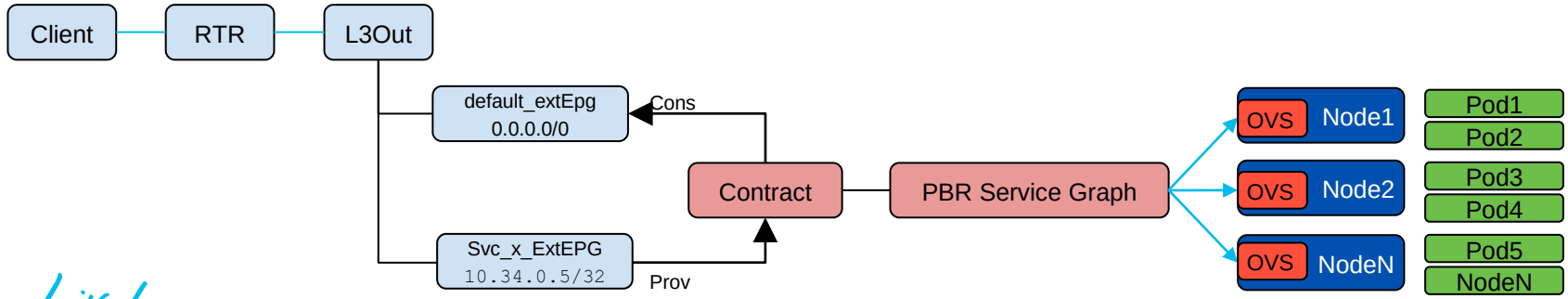
* defined in the acc-provision config file

Service Graphs and PBR – Packet walk

For your reference

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

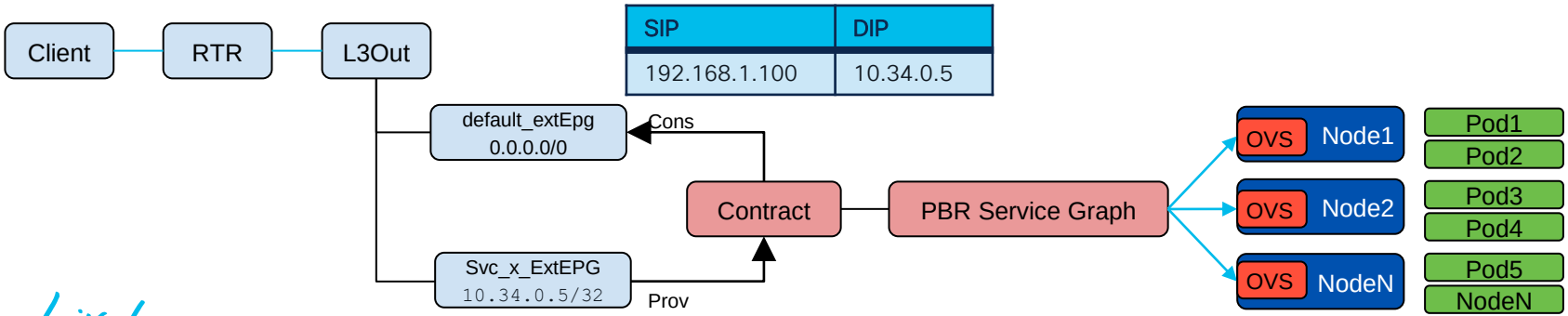
SIP	DIP
192.168.1.100	10.34.0.5



Service Graphs and PBR – Packet walk

For your reference

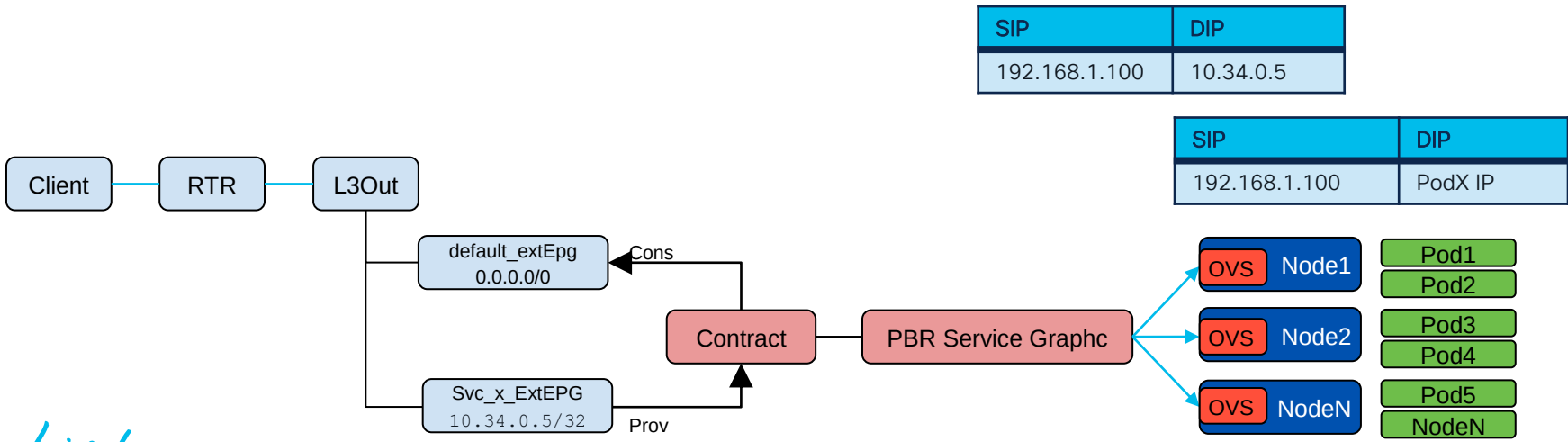
1. Client send a request to 10.34.0.5, ACI performs policy look up on the SIP and classify the traffic in the default_extEPG
2. ACI performs policy lookup for 10.34.0.5 and matches it in Svc_x_ExtEPG activating the contract between the two EPGs
3. PBR redirection is triggered and the traffic is LoadBalanced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



Service Graphs and PBR – Packet walk

For your reference

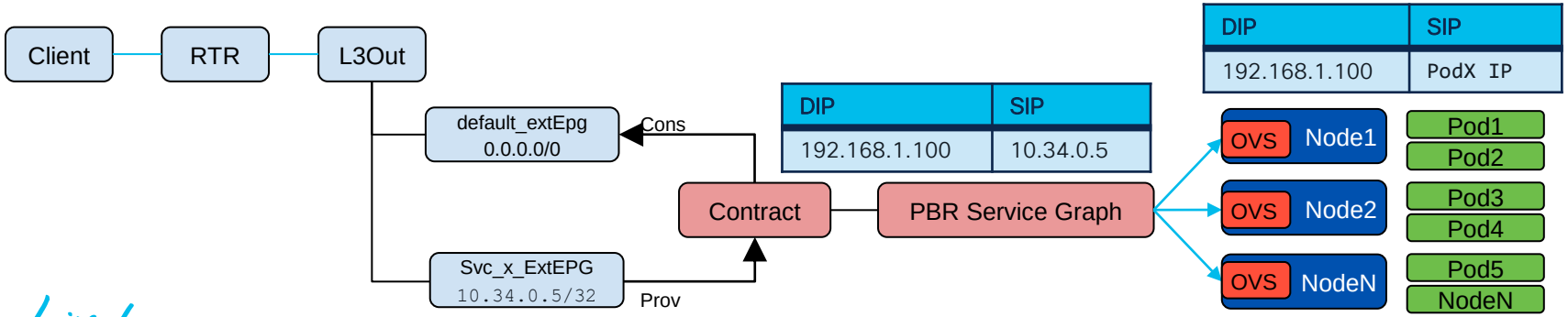
4. The K8S node is not expecting any traffic directed to the external service IP so OVS will perform NAT as required
5. If there are multiple POD on a single node OVS will perform a second stage LB to distribute the load between Pods running on the same node



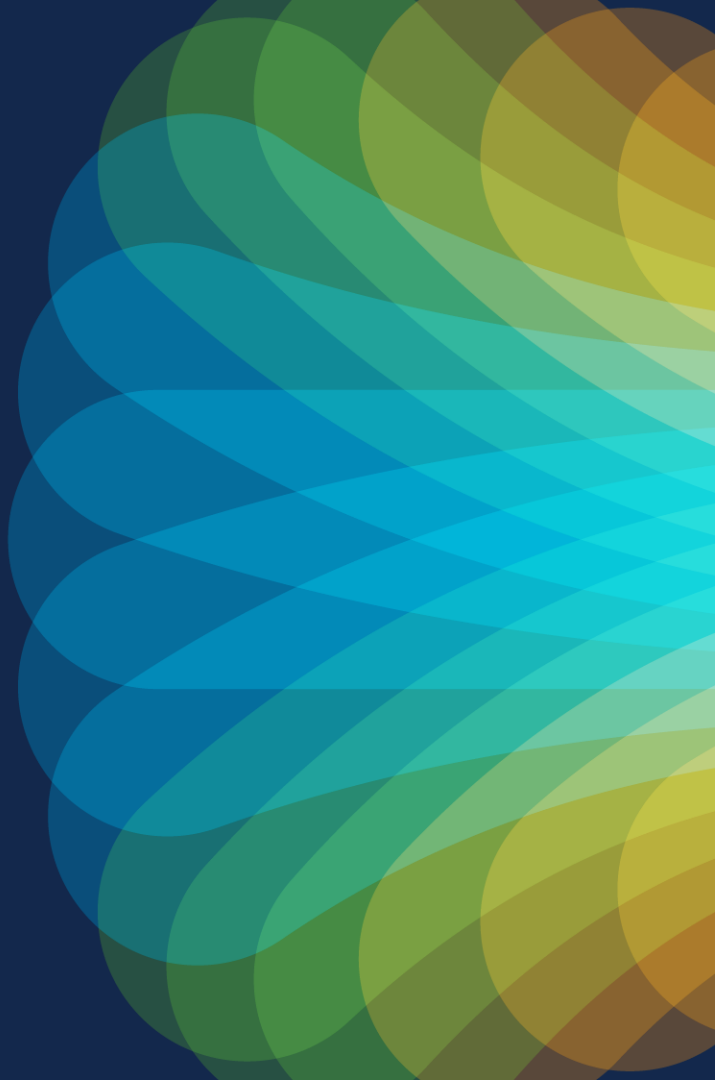
Service Graphs and PBR – Packet walk

For your reference

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)



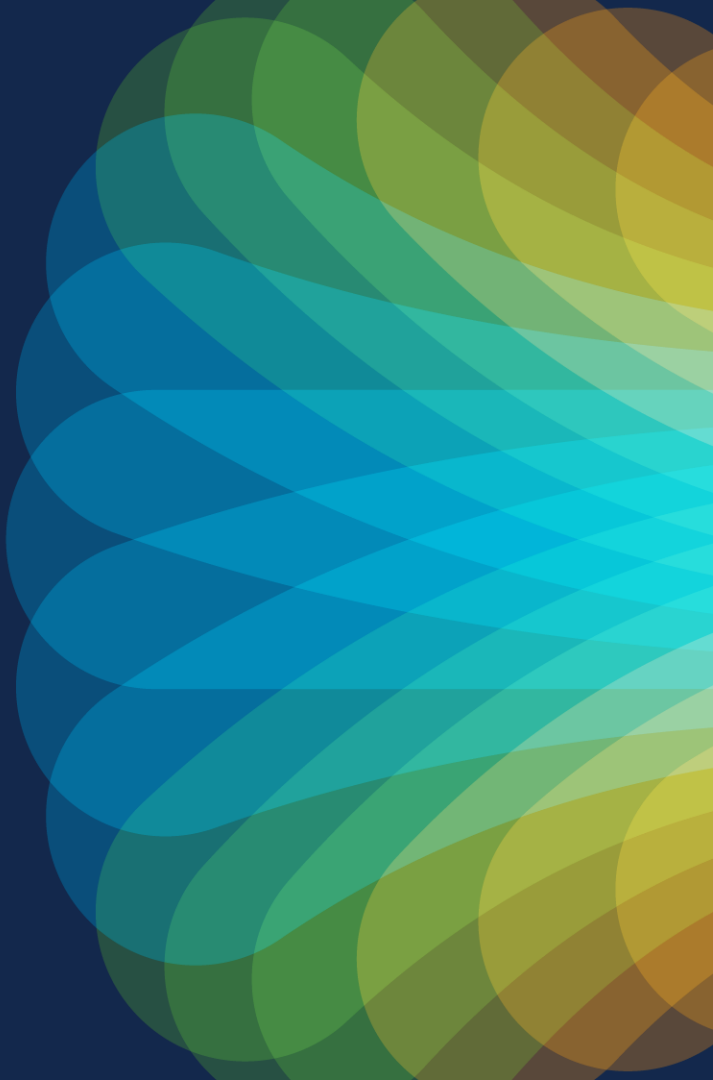
POD SNAT



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!

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!

- POD Initiated traffic can be natted to an IP address selected by the user
  - 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.

For your reference

ACI CNI SNAT Design Overview

ACI CNI SNAT Architecture Overview

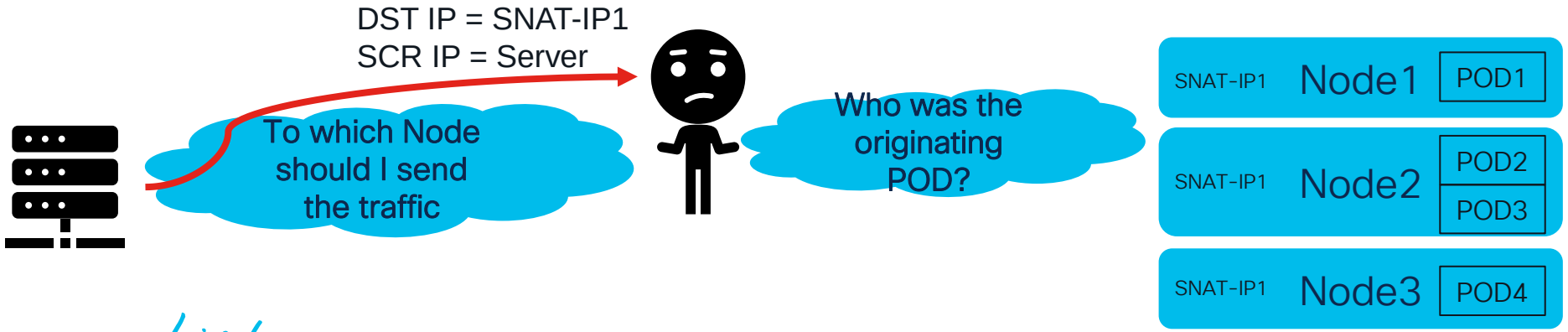
For your reference

- 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

For your reference

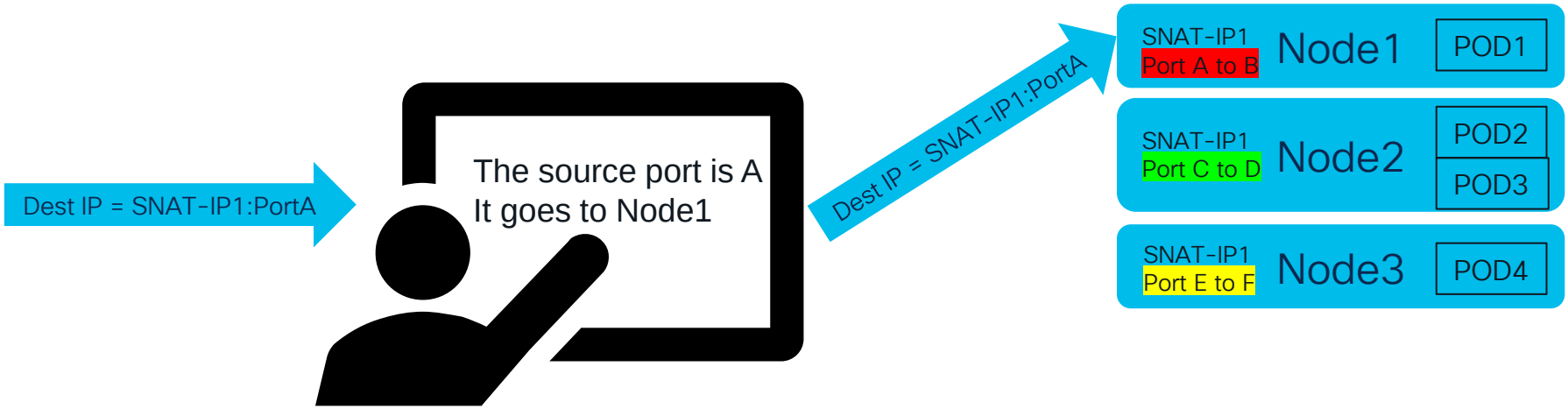
- SNAT IPs are allocated per scope:
  - Kubernetes cluster
  - Namespace
  - Deployment
  - Service (Re-Using the external service IP)
- A “Scope” can exist on multiple Kubernetes Nodes and the same SNAT IP can be used on multiple nodes and this requires some clever thinking for the return traffic because:



ACI CNI SNAT IP Allocation (cont.)

For your reference

- 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



ACI CNI SNAT IP Allocation (cont.)

For your reference

- The port-range is user configurable, by default:
  - A SNAT-IP will use ports in the range 5000 to 65000
  - The per-node SNAT-IP Port range size is 3000 ports
  - With 2500 port per node 1 SANT-IP can support up to 24 nodes in a cluster (65000 – 5000)/2500=20
  - If Multiple IPs are allocated to the SAME SNAT Policy once the port range of the SNAT-IP (5000 to 65000) is exhausted a new SNAT-IP is allocated
- It is important to size this correctly: if you expect that a single SnatPolicy can have more than 3000 POD initiated connections per Node you should increase this range and allocate a sufficient number of SNAT-IP in the SnatPolicy

Data Path – Egress traffic

For your reference

- When traffic is initiated from a POD with SNAT enabled, the source NATing happens on the node on which the pod resides. The following will happen:
  - Verify the ACI Contracts are allowing the egress traffic
  - If Contract allows, and if SNAT is configured for the pod, we determine if the destination is cluster local or not. If it belongs to the cluster, it follows current processing used for the east-west data path and no SNAT is applied.
  - If the destination does not belong to the cluster, we leverage the OVS SNAT flow rules to map the pod's Source-IP:Port to a SNAT-IP:SNAT-Port on the host on which that pod runs.
  - The SNAT-Port is chosen dynamically by OVS from the allocated port-range for the SNAT-IP address for that node.
  - The packet is forwarded from the Node on the service VLAN and will make its way through the ACI fabric to the external network.

Data Path – Return traffic

For your reference

- The return traffic uses as destination IP the SNAT-IP
- Since multiple nodes could map to the same SNAT-IP, we leverage ACI service graph with PBR redirection to send the return packet to the cluster
  - A new externalEPG called <clusterName>_ snat_svcgraph is created and configured with the subnets used for SNAT
    - The only exception is if we are using the External Service IP as SANT-IP. In such case the same PBR Policy/ExtEPG/Contract of the exposed service will be also used for the return SNAT traffic
  - A new contract called <clusterName>_snat_svcgraph is created and configured as a provider of your default external EPG and as a consumer of the <clusterName>_ snat_svcgraph externalEPG

Note: The contract Scope is “Global” and the extEPG subnet is configured as “Shared Security Import Subnet” to allow for route leaking across VRFs. This is non-configurable. A future ACI CNI release will allow the user to override this behavior

Data Path – Return traffic (cont.)

For your reference

- The ACI PBR redirection feature uses, as load balancing mechanism, an hash based on Source IP, Destination IP and Protocol Type. Thus, the return traffic may or may not land on the node on which source pod resides to handle this case:
  - If the destination pod resides on the node, OVS connection tracking rules translate the SNAT-IP:SNAT-Port to the pod's Source-IP:Port.
  - If the destination pod reside on a different node, OVS rules bounce the traffic to the actual destination node by replacing the destination MAC address with the one of the correct node.

Note: ICMP is not supported

For your reference

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

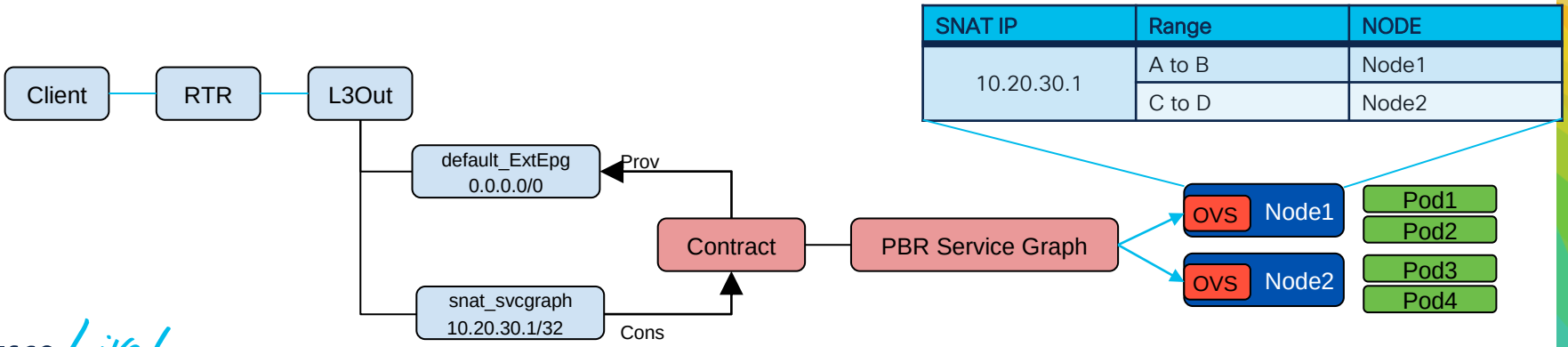
For your reference

ACI CNI SNAT Packet Walk

Service Graphs for SNAT

For your reference

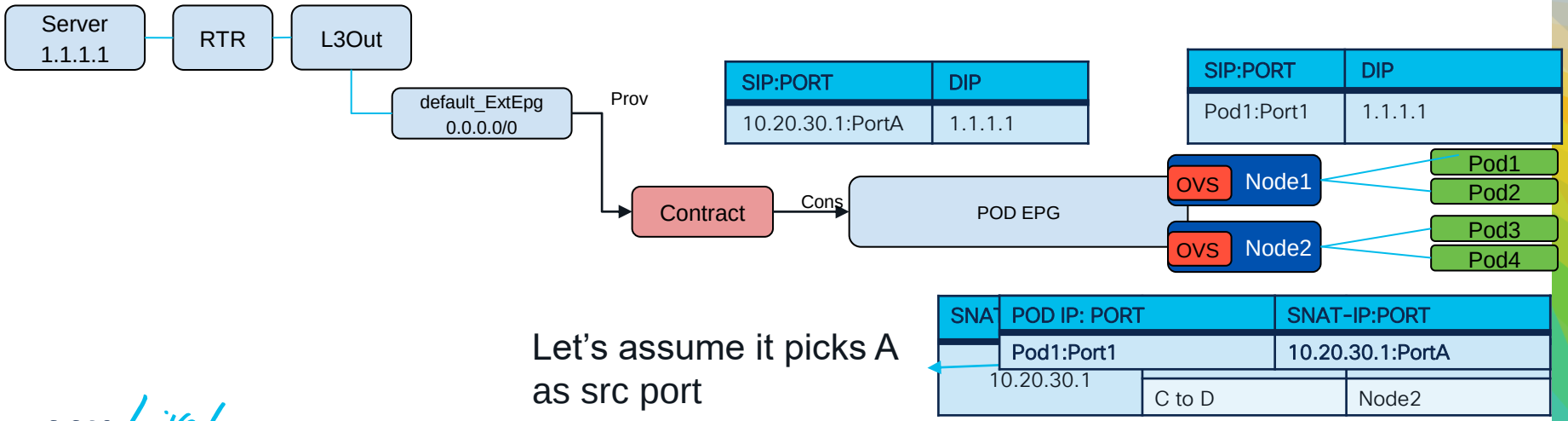
- The first time a SNAT Policy is configured the aci-container-controller will:
  - Create a snat_svcgraph externalEPG[1]
  - Add the SNAT IP under the snat_svcgraph
  - Create a new snat_svcgraph contract between the snat_svcgraph ExtEPG and the default_ExtEPG
  - Create a Service Graph with PBR redirection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



Service Graphs for SNAT – Packet walk egress

For your reference

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



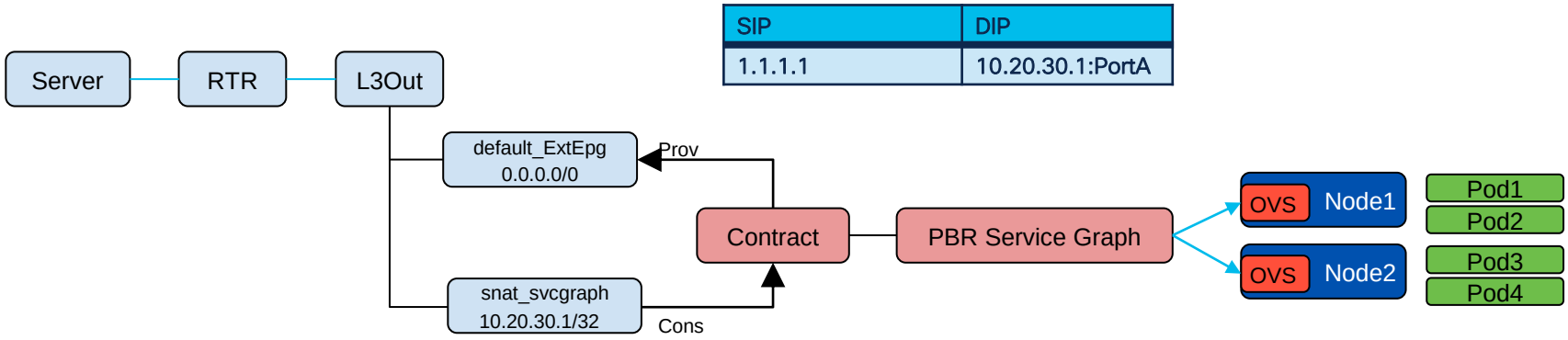
Let's assume it picks A as src port

For your reference

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

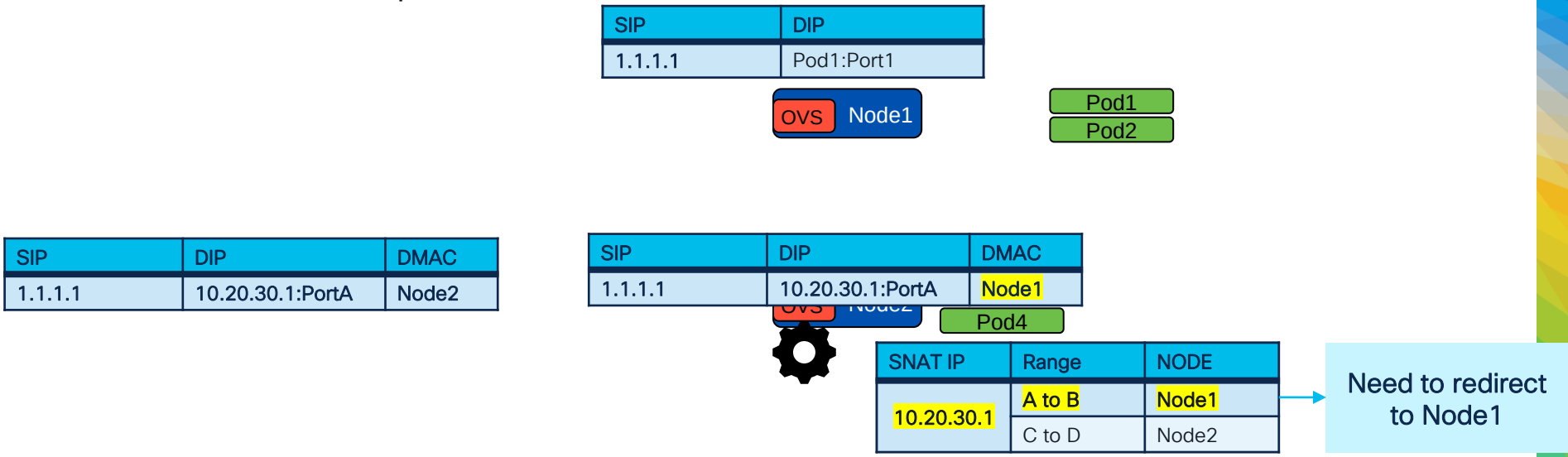
SIP	DIP
1.1.1.1	10.20.30.1:PortA



Service Graphs for SNAT – Packet walk ingress

For your reference

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.

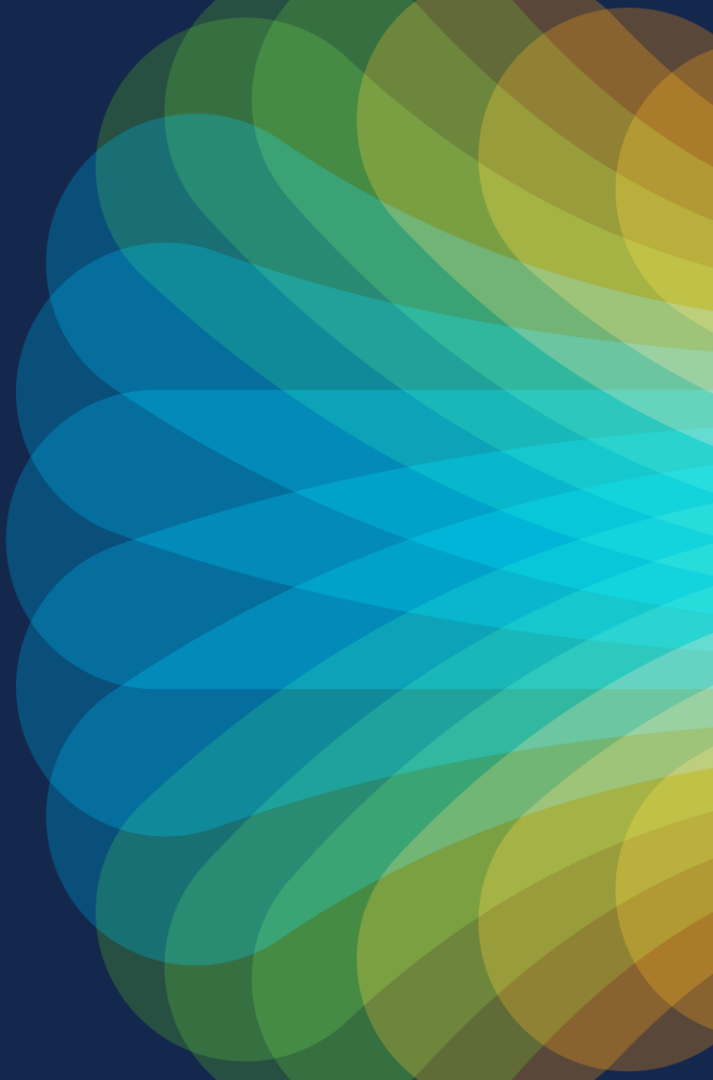


ACI CNI Demo

Agenda

- Kubernetes Refresh
- Kubernetes Network Challenges
- ACI-CNI
- **BGP Based Architecture**
  - Calico, Cilium and Kube-Router
  - Automation and Visibility
- Which solution is right for me?
- Q&A

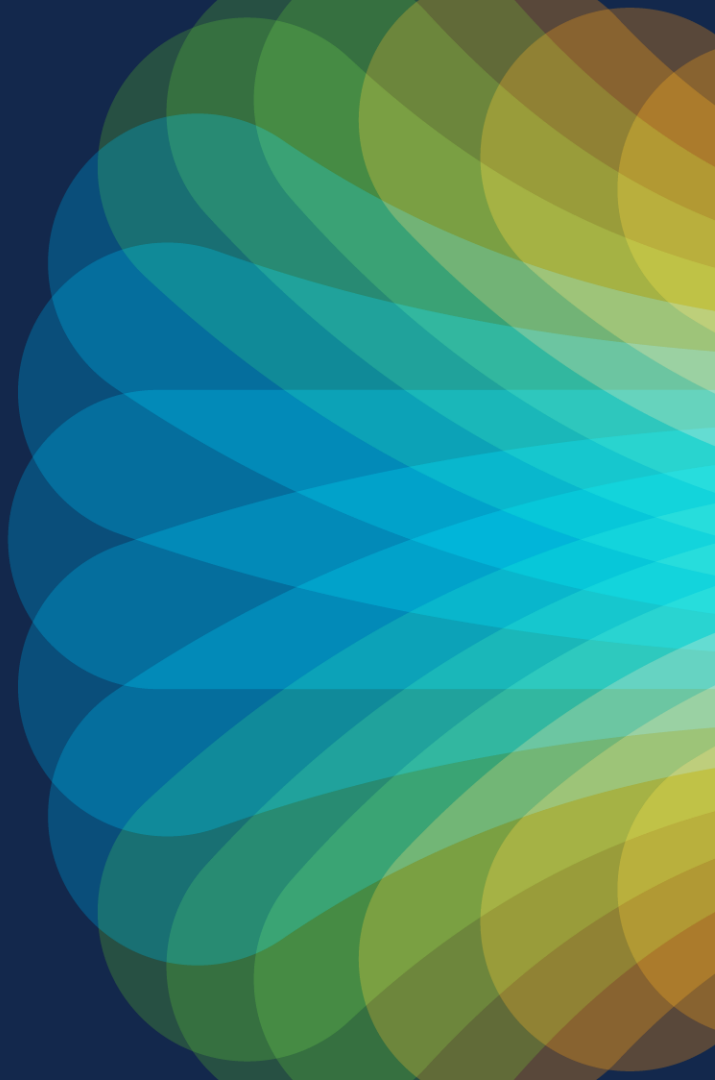
BGP Based Architecture



BGP Based Integration Benefits – why?

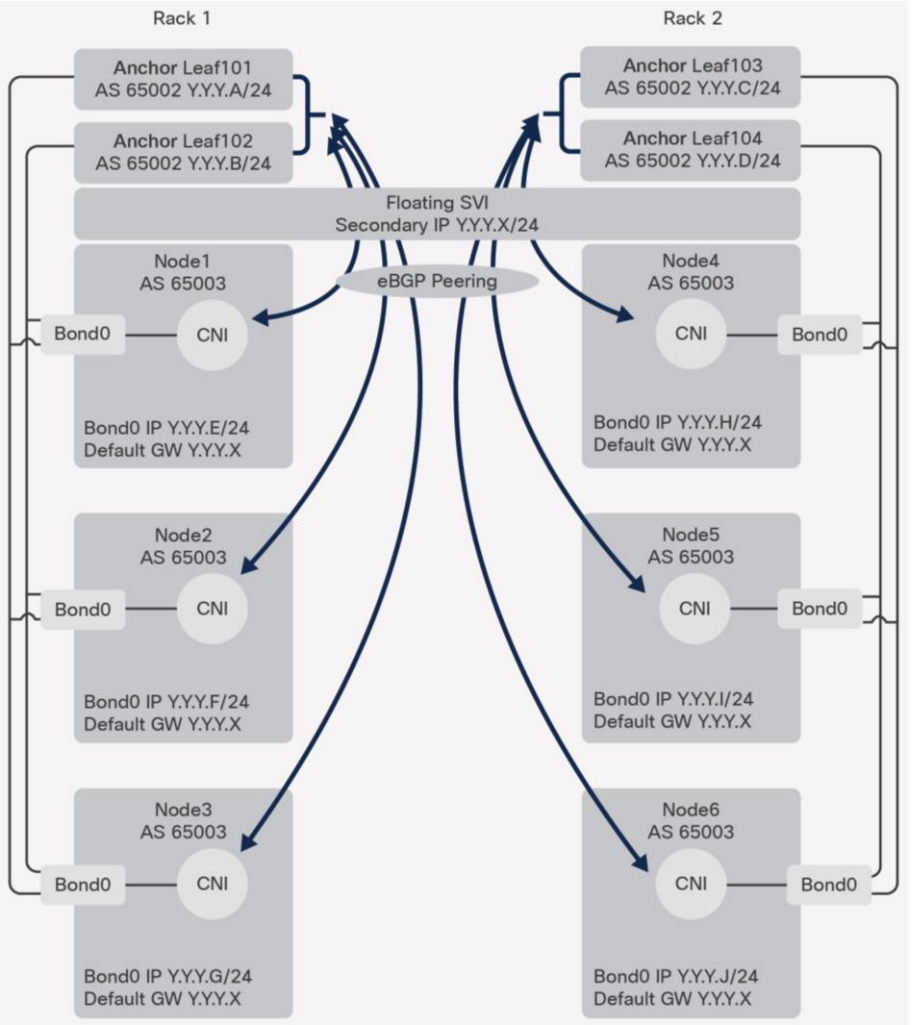
1. **Relies on a well-established protocol (BGP)**
2. **Unified networking**: Node, Pod and Service endpoints are accessible from an L3OUT providing easy connectivity across and outside the fabric
3. **(Limited) ACI Security:** ability to use external EPG classification to secure communications to Node/Pod/Service Subnets (no /32 granularity)
4. **High performance**: low-latency connectivity without egress routers if no Overlay are used
5. **Hardware-assisted load balancing**: ECMP up to 64 paths/Nodes
6. **Any Hypervisor/Bare Metal:** allows to mix form factors together

Architecture and Configuration



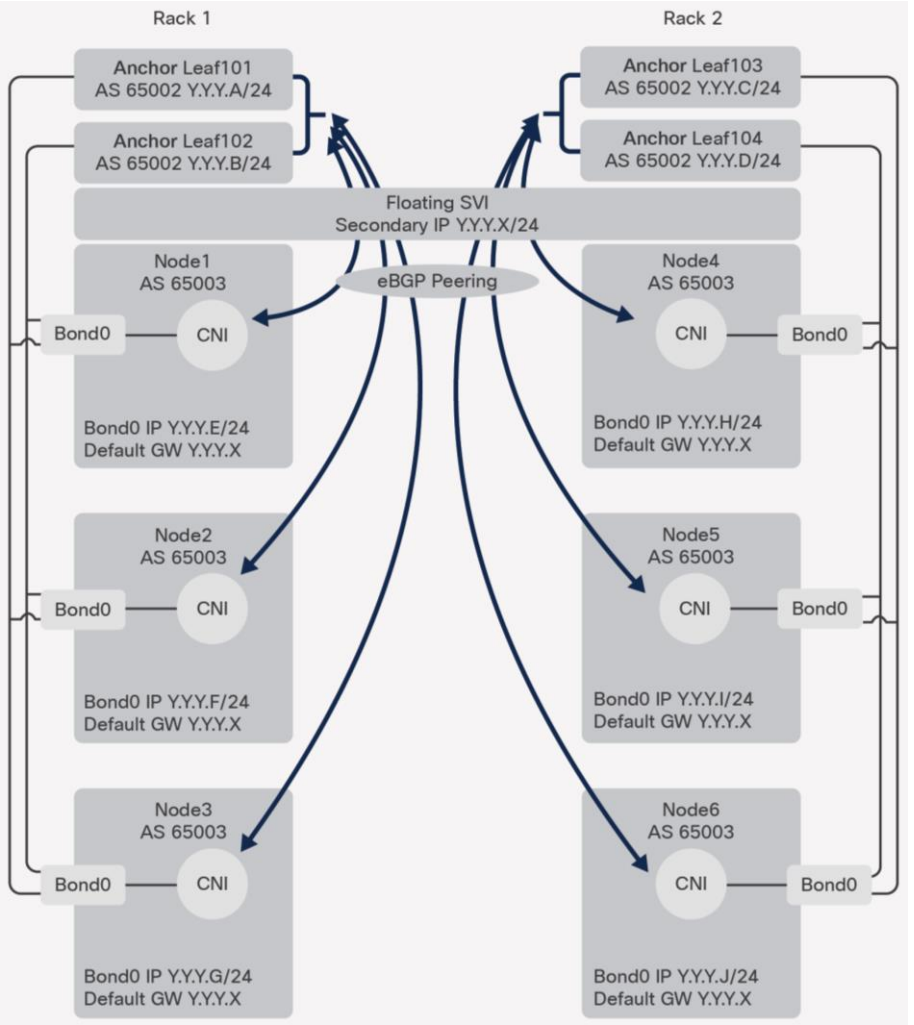
Architecture

- Each K8s Node will peer with a pair of border leaves
- Single AS for the whole cluster
  - Simpler ACI config (can use a subnet for passive peering)
- CNI Advertise all the K8s subnets to ACI as well as host-routes for exposed services leveraging ECMP for LoadBalancing



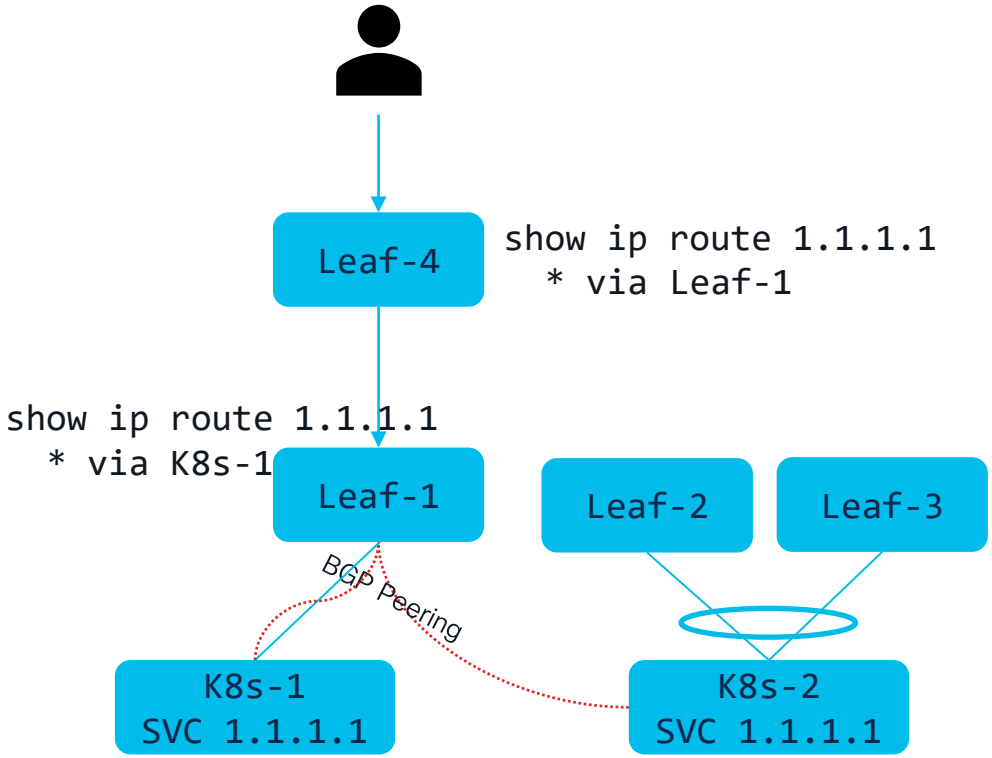
L3OUT Design

- K8s Nodes are connected to an L3OUT via vPC
  - External EPGs can be used to classify the traffic coming from the cluster
- Floating L3OUT
  - VM Mobility
  - Ability to mix BareMetal and VMs running on any hypervisor



ACI Best Practice: Peer to local ToR

- If some K8s nodes are connected to **Anchor** and some to **Non-**Anchor Leaves and are advertising the same Service IP only the one connected to the **Anchor** Leaves are selected as valid next hop.
- This happens because the Next-Hop cost is higher for to Non-Anchor Leaves connected K8s Nodes.
- We are working to address this in an upcoming ACI release



ACI BGP Tuning

- AS override and Disable Peer AS Check: To support having a single AS per cluster without the presence of Route Reflectors or Full Mesh inside the cluster
- BGP Graceful Restart
- BGP timers tuned to 1s/3s for quick eBGP node down detection
- Relax AS path policy to allow installing more than one ECMP path for the same route
- Increase Max BGP ECMP path to 64 for better load balancing

ACI BGP Hardening (Optional)

- Enabled BGP password authentication
- Set the maximum AS limit to one
- Configure BGP import route control to accept only the expected subnets from the Kubernetes cluster:
  - Pod subnet(s)
  - Node subnet(s)
  - Service subnet(s)
- Set a limit on the number of received prefixes from the nodes.

Expected Routing Behaviour

- Nodes, pods and service IPs Subnets will be advertised to the ACI fabric
- Every K8s nodes is allocated one or more subnets from the POD Supernet. Each subnets is advertised to ACI as well
- Exposed Services will be advertised to ACI as host routes from every nodes that has a running POD associated to the service.

Agenda

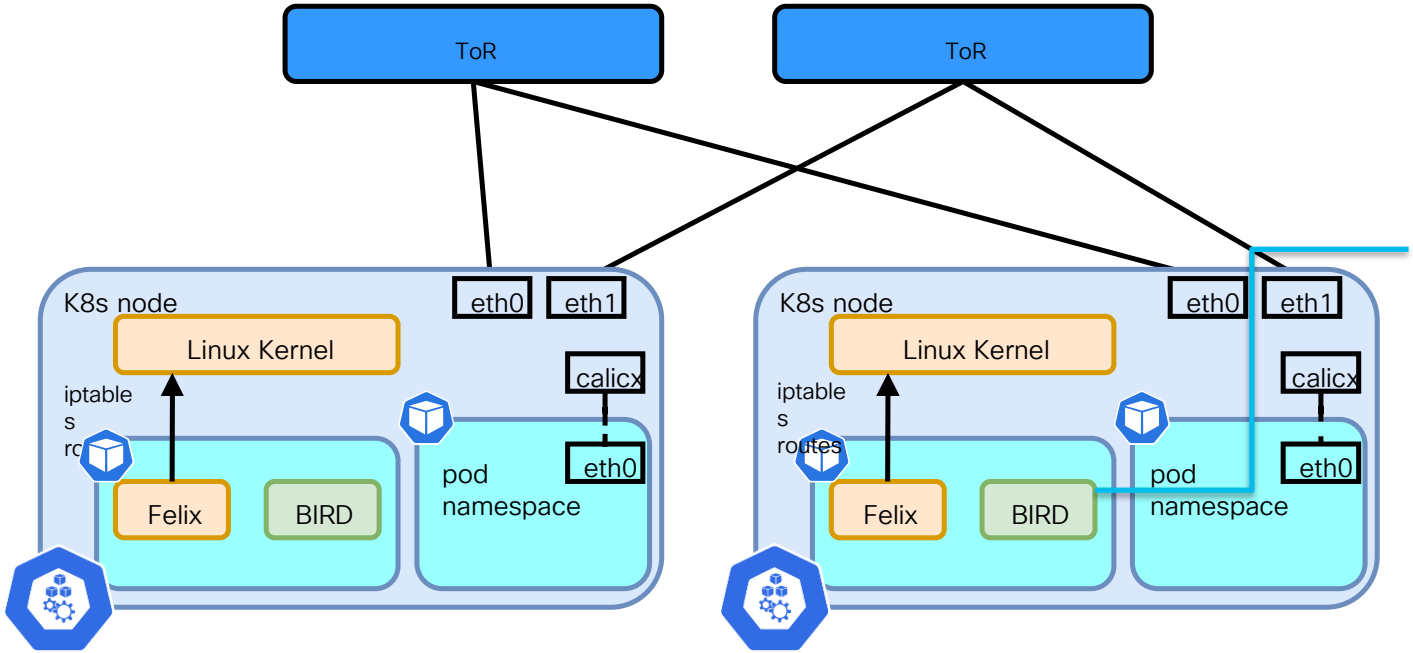
- Kubernetes Refresh
- Kubernetes Network Challenges
- ACI-CNI
- BGP Based Architecture
  - **Calico, Cilium and Kube-Router**
  - Automation and Visibility
- Which solution is right for me?
- Q&A

What is Calico

A Kubernetes CNI plugin

Calico

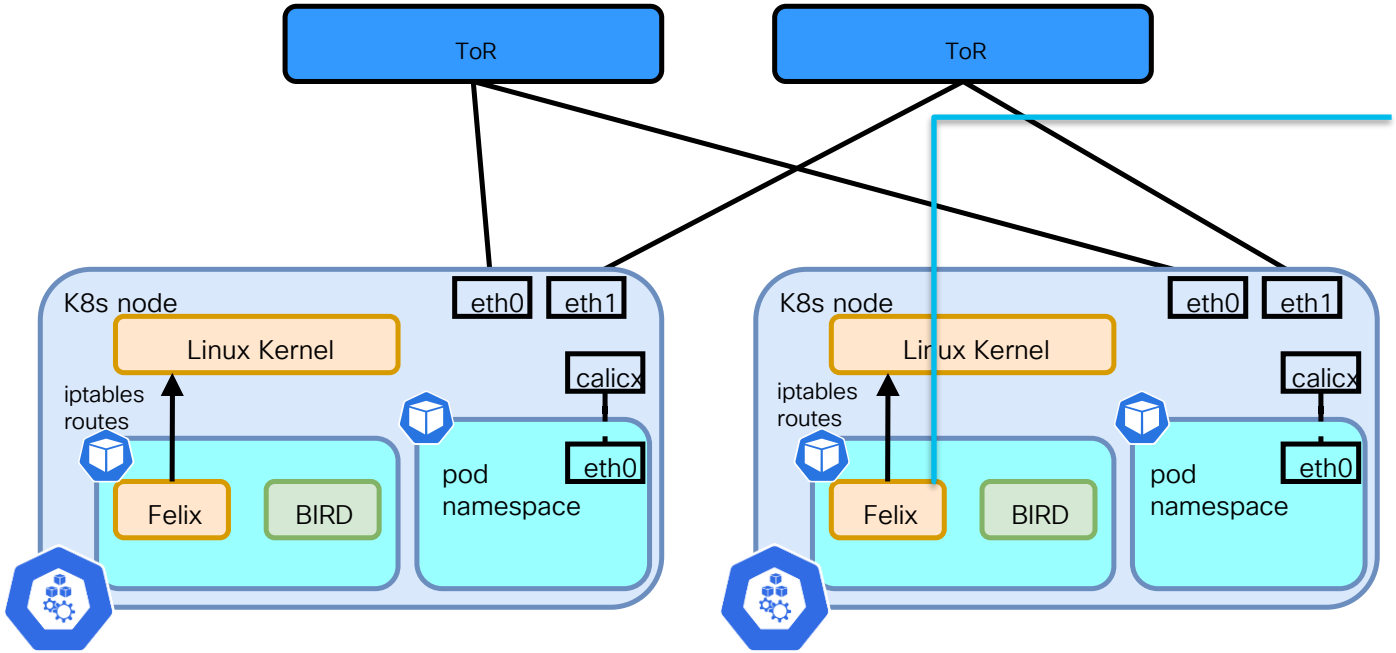
A CNI plugin of Kubernetes



BIRD: It is a routing daemon responsible for peering with other K8s nodes and exchanging routes of pod network and service network for inter-node communication.

Calico

A CNI plugin of Kubernetes



Felix: Running in same pod as BIRD, programs routes and ACLs (iptables) and anything required on Calico node to provide connectivity for the pods scheduled on that node

Calico BGP Config

- One or more IPPool with all overlays disabled
- BGPConfiguration with:
  - nodeToNodeMeshEnabled set to “false”
  - List of serviceClusterIPs and serviceExternalIPs subnets to enabled host routes advertisement for those subnets
- BGPPeer to define the BGP Peer the K8s nodes connects to
- A Secret, Role and RoleBinding to pass the BGP Password to the Calico BGP Process

Calico IPPool Config – Cont.

```
apiVersion: crd.projectcalico.org/v1
kind: IPPool
metadata:
name: default-ipv4-ippool
spec:
  blockSize: 26
  cidr: 192.168.3.0/24
  ipipMode: Never
  nodeSelector: all()
  vxlanMode: Never
```

- `blockSize: 26` → How to split the POD subnet between nodes
- `cidr: 192.168.3.0/24` → POD Subnet
- `ipipMode: Never` → Disable IP in IP
- `nodeSelector: all()` → Allocate this Subnet to all the nodes
- `vxlanMode: Never` → Disable VXLAN Overlay

Calico BGPConfiguration – Cont.

```
apiVersion:
crd.projectcalico.org/v1
kind: BGPConfiguration
metadata:
  name: default
spec:
  asNumber: 65003
  listenPort: 179
  logSeverityScreen: Info
  nodeToNodeMeshEnabled: false
  serviceClusterIPs:
  - cidr: 192.168.4.0/24
  serviceExternalIPs:
  - cidr: 192.168.5.0/24
```

K8s Cluster BGP AS

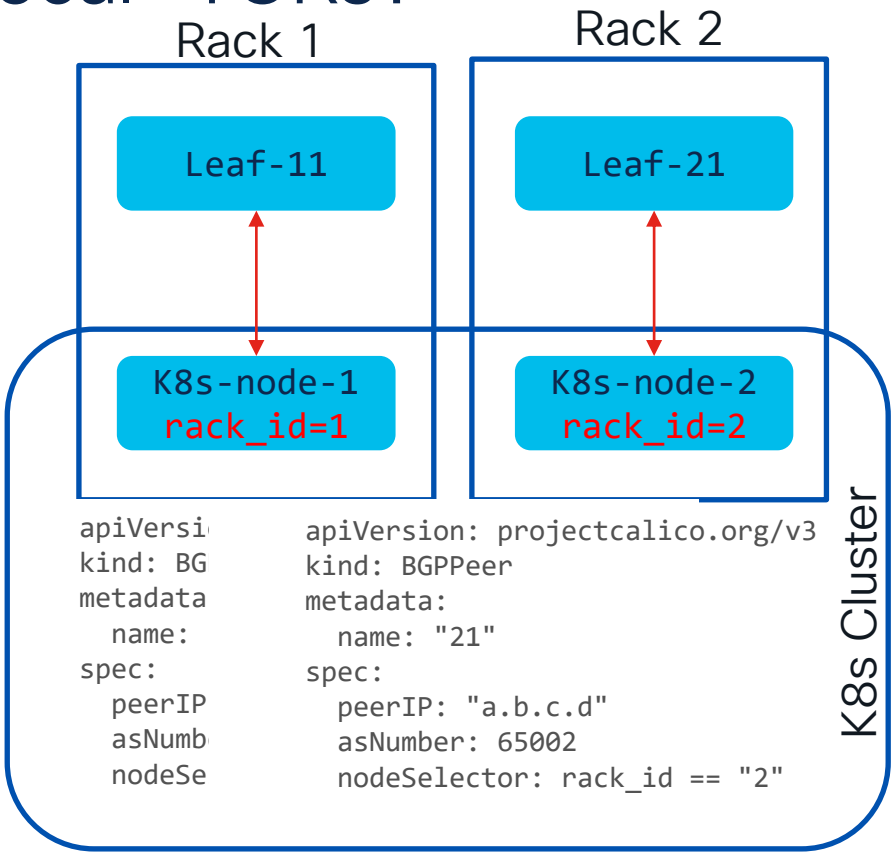
BGP Port

Disable iBGP Full Mesh Peering

Allow Calico to Advertise the Cluster and External service Subnets

How do I peer with the “local” TORs?

- Use a Node label to identify the location of the K8s Node, for example the rack id
- Configure the BGPPeer resource with a *nodeSelector* matching the label of the K8s Nodes
- The result will be that the peering is happening only between K8s Nodes and leaves with a matching rack id



Agenda

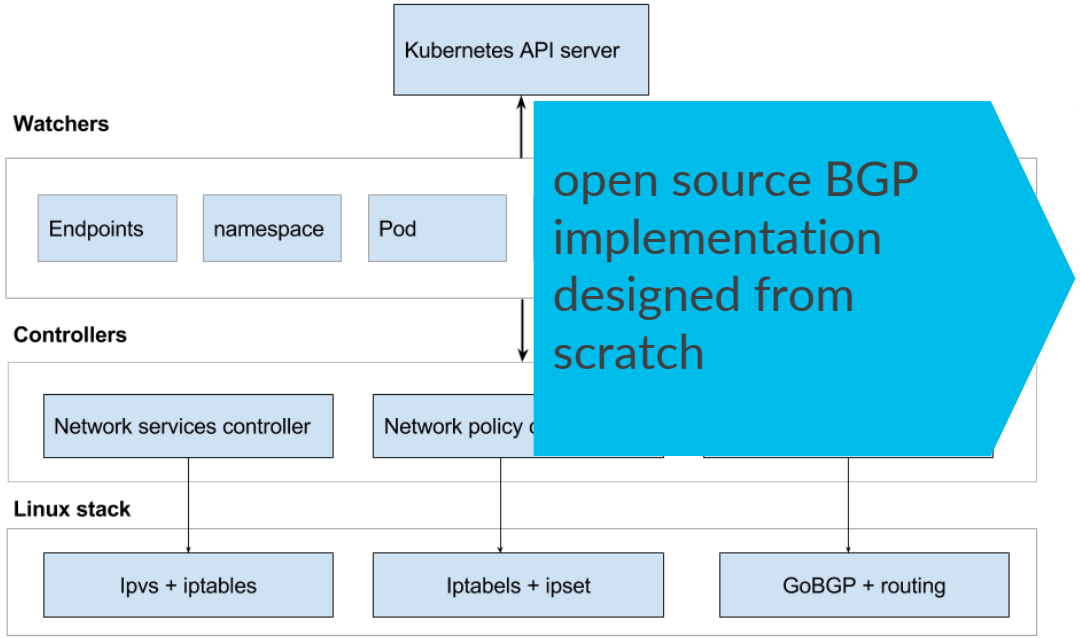
- Kubernetes Refresh
- Kubernetes Network Challenges
- ACI-CNI
- BGP Based Architecture
  - Calico, Cilium and Kube-Router
  - Automation and Visibility
- Which solution is right for me?
- Q&A

What is Kube-Router

A Kubernetes CNI plugin

Kube Router

A CNI plugin of Kubernetes



- Kube-router is built around concept of watchers and controllers.
- GoBGP Runs inside the KubeRouter POD
  - Injects learned routes into local Node routing table

Kube-Router eBGP Config

- Most of the configuration is applied at the Kube-Router DaemonSet level, for our design we need the following options

```
--run-router=true
--run-firewall=true
--run-service-proxy=true
--bgp-graceful-restart=true
--bgp-holdtime=3s
--kubeconfig=/var/lib/kube-router/kubeconfig
--cluster-asn=<BGP AS>
```

```
--advertise-external-ip
--advertise-loadbalancer-ip
--advertise-pod-cidr=true
--enable-ibgp=false
--enable-overlay=false
--enable-pod-egress=false
--override-nexthop=true
```

How do I peer with the “local” TORs?

- Kube-Router expects the K8s nodes to be annotated with the peer (leaf) IP, AS and password so we simply need to annotate the K8s nodes accordingly for example:

```
kube-router.io/peer.ips=<leaf-1_IP>,<leaf-2_IP>
kube-router.io/peer.asns=<ACI AS>,<ACI AS>
kube-router.io/peer.passwords=<MD5 Pass>,<MD5 Pass>
```

- Annotating any node with the above config will result in such node to peer with “leaf-1” and ”leaf-2”

BGP CNI Demo

Agenda

- Kubernetes Refresh
- Kubernetes Network Challenges
- ACI-CNI
- BGP Based Architecture
  - Calico, Cilium and Kube-Router
  - **Automation and Visibility**
- Which solution is right for me?
- Q&A

Automation and Visibility Demo

Agenda

- Kubernetes Refresh
- Kubernetes Network Challenges
- ACI-CNI
- BGP Based Architecture
  - Calico, Cilium and Kube-Router
  - Automation and Visibility
- **Which solution is right for me?**
- Q&A

Which solution is right for me?

Which solution is right for me?

- This is an hard question, all the supported CNI plugins are enterprise grade and provide a rich feature set
- Some question you might ask yourself:
  - Is running the same CNI plugin on ANY network infrastructure important?
  - Is the K8s team already using a specific CNI ?
  - Are the advanced ACI CNI feature important?
    - Ability to place PODs into EPG, SNAT, PBR based load balancing
  - Is having a single vendor for the networking stack support important ?

CNI Comparison

	ACI CNI	Calico	Kuber-Router	Cilium
Support	TAC	OpenSource/Pay	OpenSource	OpenSource/Pay
Network Infra	ACI Only	Any	Any	Any
Network Config	Automated	Automated	Automated	Automated
Linux OS Support	Ubuntu/CoreOS	Any	Any	Any
Service LoadBalancing	ACI PBR	BGP ECMP	BGP ECMP	BGP ECMP
POD SNAT	NAT Pools	NAT To Node IP	NAT To Node IP	NAT To Node IP
Security Model	ACI Policy Model + K8s Network Policies	Calico Network Policies	K8s Network Policies	Extended K8s Network Policies
End To End Visibility	Yes	Via Opensource Tools	Via Opensource Tools	Hubble
Data Plane	OVS	Linux or eBPF	Linux + IPVS for LoadBalancing	eBPF

Fill out your session surveys!

Attendees who fill out a minimum of four session surveys and the overall event survey will get **Cisco Live-branded socks** (while supplies last)!

Attendees will also earn 100 points in the **Cisco Live Game** for every survey completed.

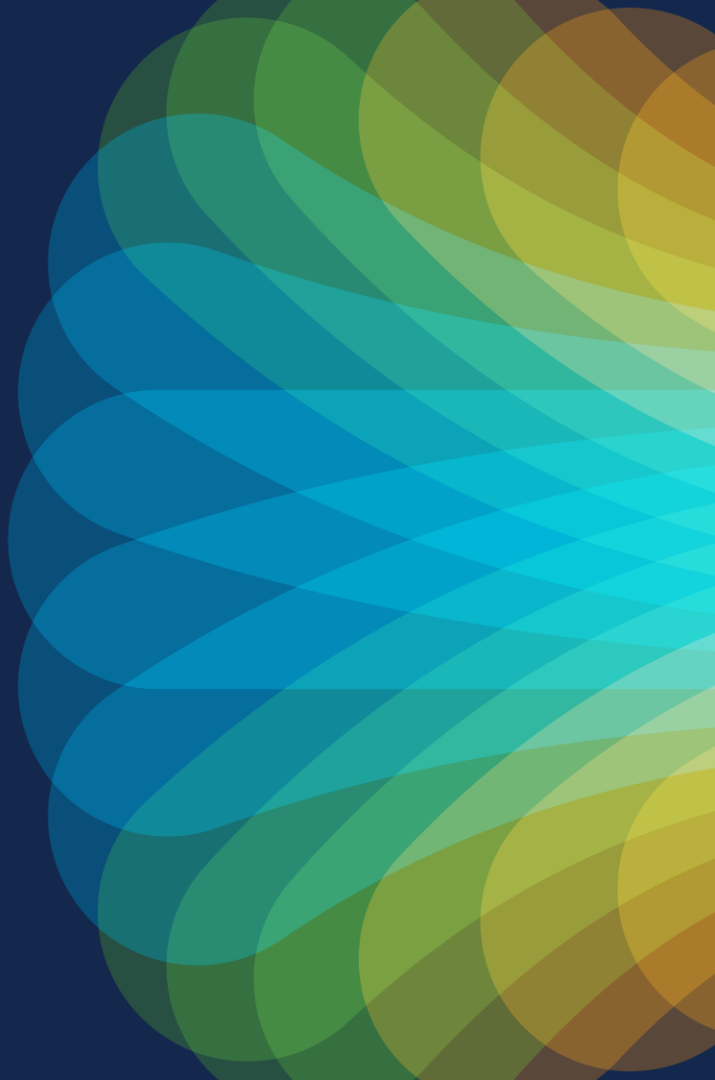
These points help you get on the leaderboard and increase your chances of winning daily and grand prizes

Continue your education

- Visit the Cisco Showcase for related demos
- Book your one-on-one Meet the Engineer meeting
- Attend the interactive education with DevNet, Capture the Flag, and Walk-in Labs
- Visit the On-Demand Library for more sessions at www.CiscoLive.com/on-demand

The bridge to possible

Thank you



Cisco Live
Challenge

Gamify your Cisco Live experience!

Get points for attending this session!

How:

1. Open the Cisco Events App.
2. Click on 'Cisco Live Challenge' in the side menu.
3. Click on View Your Badges at the top.
4. Click the + at the bottom of the screen and scan the QR code: