ACI Tenancy Models

ACME Inc. will be using tenancy for a couple of use cases. They will be using tenant constructs for the application lifecycle of their current deployment, maintaining a separate tenant for the resources that developers will be using to build the application, a tenant that will be used for the automated testing, and finally a production tenant. Additionally, as mentioned in the introduction, they are also looking to build an infrastructure which can be leveraged for similar initiatives in the future. Tenants will be used to draw virtual boundaries for different lines of business. The information security team will be able to integrate this into the corporate LDAP system, and prevent changes which would impact other groups.

Cisco Application Centric Infrastructure (ACI) has been designed from the beginning to be "multi-tenant". This means different things to different people (much like the term Cloud) based on their perspective. In the case of a classic service provider, a tenant is a unique customer, while in a typical end-customer environment a tenant could be an operating group, business unit, application owner, and so on.

The decision on how to leverage tenancy models is driven by a number of factors:

1. Overall IT operations and support models in your organization to manage application, networking, servers, security, and so on.
2. Separation of environments from a software development lifecycle perspective: development, quality assurance, and production.
3. Separation of duties by domain owner, such as web, app, and database owners.
4. Fault domain size and scope to limit the impact of failures, such as different business units.

In traditional networking environments, making a routing protocol change on a router or Layer 3 switch could potentially affect hundreds of unique VLANs/subnets. This introduces a warranted level of caution around change control and application impact. Leveraging the ACI policy model, the physical hardware is abstracted from the logical constructs. The tenant object gives us the ability to draw a box around the logical and concrete objects that we use to provide a unified view of the configuration dependencies for underlay and overlay networks.

A tenant in the ACI object model represents the highest-level object. Inside, you can differentiate between the objects that define the tenant networking, such as private networks (VRFs), bridge domains and subnets; and the objects that define the tenant policies such as application profiles and endpoint groups.
In ACI, the tenant policies are where you define applications. An application could consist of a combination of physical servers or virtual machines that we will call servers from now on. For example, a website could use a 3-tier application model, comprised of web servers, application servers and database servers. When a user browses the web site, they might actually be communicating with a virtual IP address on a load balancer that in turn can distribute the web request to a number of different web servers. The web servers in turn communicate with core applications that can be divided amongst several application servers for load balancing or high availability purposes. Finally, the application servers communicate with the database which could also be a cluster of servers.

Each server is referred to as an endpoint in ACI. Endpoints are classified in ACI to apply policies. You create endpoint groups with endpoints that share the same type of policies, such as with whom are they going to communicate and what type of communication or restrictions are required. Therefore, an application can be formed by several endpoint groups and they are grouped in an application profile.

The tenant networking is used to define networking policies and will be applied to the underlying hardware in a transparent way thanks to the layer of abstraction provided by ACI using private networks, bridge domains and subnets. In the next sections of this chapter, these concepts will be covered in detail. Below you can find an illustration with the different objects that compound a tenant and how they are related.

Although the tenant networking and the tenant policies are defined separately, the networking policies used by an application are defined with a relationship between the endpoint groups and the bridge domain.

The following image shows all of the components that can be configured within a tenant. In the following sections each diagram shows the progress of how ACME Inc. adds each component.

**Figure 1: Tenant Logical Model**

There are 3 tenants that are preconfigured in the system by default:

1. **Common**—A special tenant with the purpose of providing "common" services to other tenants in the ACI fabric. Global reuse is a core principle in the common tenant. Some examples of common services are:
   
   1. Shared L3 out
   2. Shared private networks
   3. Shared bridge domains
   4. DNS
   5. DHCP
   6. Active directory
2. **Infra**—The Infrastructure tenant that is used for all internal fabric communications, such as tunnels and policy deployment. This includes switch to switch (leaf, spine, Application Virtual Switch (AVS)) and switch to Application Policy Infrastructure Controller (APIC). The Infra tenant does not get exposed to the user space (tenants) and it has its own private network space and bridge domains. Fabric discovery, image management, and DHCP for fabric functions are all handled within this tenant.

3. **Mgmt**—The management tenant provides convenient means to configure access policies for fabric nodes. While fabric nodes are accessible and configurable through the APIC, they can also be accessed directly using in-band and out-of-band connections. In-band and out-of-band policies are configured under the mgmt tenant:
   - In-Band Management Access
   - Out-of-Band Management Access

### Application Profile

An application profile is a convenient logical container for multiple hosts (physical or virtual). You can create application profile containers based on a variety of criteria, such as what function the application provides, how the application looks from the end-user perspective, where they are located within the context of the data center, or any other logical grouping relative to the implementation. Application profile servers are grouped in endpoint groups depending on the use of common policies.

Application profiles provide a mechanism to understand groups of servers as a single application. This approach makes an Cisco Application Centric Infrastructure (ACI) application aware and allows us to check the operational state for an application while monitoring all the servers that are part of an application as a whole. Furthermore, an administrator can become informed about relevant faults and health status for that particular application. Each application profile created can have a unique monitoring policy and QOS policy applied.

An application profile is a child object of the Tenant and a single Tenant can contain multiple application profiles.

*Figure 2: Adding components to a Tenant - 1. Application Profile*
Application Profile Configuration

- **Name** - The name of the application profile.
- **Tags** - A tag or metadata is a non-hierarchical keyword or term assigned to the fabric module.
- **Monitoring Policy** - The monitoring policy name for the EPG semantic scope (optional).

Creating a New Application Profile Using the GUI

Modifying an Application Profile Using the GUI

Removing Application Profile Using the GUI

Verify Application Profile

REST :: /api/node/class/fvAp.xml

CLI :: moquery -c fvAp

Endpoint Group

Endpoint groups (EPGs) are used to create logical groupings of hosts or servers that perform similar functions within the fabric and that will share similar policies. Each endpoint group created can have a unique monitoring policy or QoS policy and are associated with a bridge domain.

An endpoint group is a child object of the application profile and an application profile can contain multiple endpoint groups. Each endpoint within an endpoint group is susceptible to the same policy in the Fabric.

*Figure 3: Adding components to a tenant - 2. End Point Group in the Application Profile*

All of the endpoints inside an EPG can communicate with each other. Communications between EPGs is governed by contracts and not traditional Layer 2/Layer 3 forwarding constructs. For example, Host-A in EPG-A can have the IP address/mask of 10.1.1.10/24 and Host B in EPG B can have the IP address/mask 10.1.1.20/24 (note that both hosts believe they are "in the same subnet"). In this case they would not be allowed to communicate unless a contract that permitted connectivity existed between EPG-A and EPG-B. Contracts will be explained in greater detail in a following section.
There are some types of endpoint groups within the fabric that are not contained under application profiles such as, Application endpoint group, External Bridge Networks (aka Layer2 External), External Routed Networks (aka as Layer3 External) and Management endpoint groups. These endpoint groups might have special requirements, for example, in External Bridge Networks, MAC addresses of the endpoints are not learnt by the leaf switches.

Endpoint groups are linked to bridge domains but they will receive a VLAN ID different from the bridge domain, unless Bridge Domain legacy mode is used.

It is important to understand that a single subnet can be extended across several EPGs. Each EPG is identified by an encapsulation VLAN or VXLAN so that the same subnet will be using different encapsulation IDs across the fabric. This concept is different from traditional networking.

**Endpoint Group Subtypes**

In Application Policy Infrastructure Controller (APIC) software version 1.1, legacy endpoint group became split into two types:

**Application Endpoint Group**

This is the traditional endpoint group, which can be applied to Virtual or Physical endpoints using static path bindings, VMM integration, or Layer 2/Layer 3 domain binding.

**Microsegment (uSeg) endpoint group**

This classification of endpoint group allows various "attributes" to be matched against endpoints to assign them automatically into a uSeg endpoint group. This type of endpoint group is also referred to as attribute-based endpoint groups. Some of the attributes that you can match against include VM properties (such as VM Name, VM ID, and Hypervisor), MAC addresses, and IP sets.

Once a uSeg endpoint group has been created and eventually assigned to a VMM domain, it will auto-match on any endpoint within the VMM domain that exists within the tenant and move any endpoints from their assigned application endpoint group to the uSeg endpoint group. Once this occurs, any policies applied to the uSeg EPG (Contracts, QoS, Monitoring Policies, and so on) are now applied. Policies from their original application EPG are no longer applied to the endpoint.

---

**Note**

VM endpoints are assigned to their application endpoint group, but the fabric will automatically move them to their uSeg endpoint group if an attribute match exists. This means that within their Virtual Machine Manager (vCenter), the endpoint will still show as assigned to the application EPG/port group. However, if you examine the uSeg EPG > Operational > Client End Points, you should see the endpoint learned under its new uSeg EPG.

When adding attributes to a uSeg endpoint group, it must not be currently assigned to any VMM domains. This ensures the endpoint group is not currently assigned to any VM endpoints and therefore prevents accidentally moving of functional endpoints by mistakenly assigning an attribute to an always bound VM domain. For this reason, during the create uSeg endpoint group procedure, you must create the endpoint group first, then add the VMM domain afterwards. This ensures that a uSeg endpoint group's attributes are assigned before the VMM domain is added.
Create a New Endpoint Group

Modify Endpoint Group

Remove Endpoint Group

Verify Endpoint Group

REST :: /api/node/class/fvAEPg.xml
CLI :: moquery -c fvAEPg

Endpoint

Endpoints are devices that are connected to the network either directly or indirectly. Endpoints have an address (identity), a location, and attributes, and can be either virtual or physical. Each endpoint has a path, an encapsulation, and a deployment Immediacy mode associated with it.

An Endpoint is a child object of the Endpoint Group and an Endpoint Group construct can contain multiple Endpoints. The Endpoints referenced within the fabric can be either static (defined within the APIC) or dynamic (automated by vCenter/Openstack).

You can add Static Endpoints by creating Static Bindings within the Endpoint Group. Below is an example of a static binding. See the VVM section for an example of a dynamic binding.

In order to show the endpoints that are connected to the fabric under certain EPGs:

Verify Endpoint

REST :: /api/node/class/fvCEp.xml
CLI :: moquery -c fvCEp

Private Networks

A private network is also referred to as a Virtual Routing and Forwarding (VRF), private Layer 3 network, or context. It is a unique Layer 3 forwarding and application policy domain. Private networks are a child of the Tenant object. All of the endpoints within the private network must have unique IP addresses because it is possible to forward packets directly between these devices if the policy allows it. One or more bridge domains are associated with a private network.
The most common method to share private networks between tenants is through the common tenant. For more information about common tenants, see the overview section of this chapter. Private networks created in the common tenant are shared globally within the fabric. However, a private network that is intended to be used by multiple tenants and is not created in the common tenant requires explicit configuration to be shared. When there is a requirement to route traffic between separate private network instances, special consideration for subnet configuration is needed. This will be discussed in detail in the bridge domain and endpoint group configuration sections.

**Private Network Configuration Parameters**

The following list describes the private network configuration parameters:

- **Name**—The name of the private network.
- **Policy Control Enforcement Preference**—The preferred policy control. The values can be **enforced** or **unenforced**. When enforced is chosen, contracts between endpoint groups are required to allow traffic. Unenforced allows all traffic within the Private Network. The default is **enforced**.
- **Policy Control Enforcement Direction**—The preferred policy control in relation to where the policy will be applied, that is to say in which direction. The default is ingress.
- **End Point Retention Policy**—The end point retention policy name (optional).
- **Monitoring Policy**—The monitoring policy name for the Tenant semantic scope (optional).
- **DNS Label**—The network domain name label. Labels enable classifying which objects can and cannot communicate with one another (optional).
- **BGP Timers**—Name of the BGP timers policy associated with this object.
- **OSPF Timers**—Name of the OSPF timers policy associated with this object.
- **OSPF Address Family Context**—The OSPF Address Family Context policy name.
- **EIGRP Address Family Context**—The EIGRP Address Family Context policy name.
Creating a New Private Network

Modifying a Private Network Using the GUI

Removing Private Network Using the GUI

Verify Private Network

REST :: /api/node/class/fvCtx.xml
CLI :: moquery -c fvCtx

Bridge Domain

A bridge domain is the logical representation of a Layer 2 forwarding domain within the fabric. A bridge domain is a child of the tenant object and must be linked to a private network.

The bridge domain defines the unique Layer 2 MAC address space and a Layer 2 flood domain if flooding is enabled. While a private network defines a unique IP address space, that address space can consist of multiple subnets. Those subnets will be spread across one or more bridge domains contained in the private network.

Bridge domains will span all switches in which associated endpoint groups are configured. A bridge domain can have multiple subnets. However, a subnet is contained within a single bridge domain.

*Figure 5: Adding components to a Tenant - 4. Bridge Domain as part of the Tenant Application Profile*

The following image provides an example of a tenant to show how bridge domains are contained inside of private networks and how they are linked to endpoint groups and the other elements.
A bridge domain is not a VLAN, although it can act similar to a VLAN. Think of a bridge domain as a distributed switch, which, on a leaf, can be translated locally as a VLAN with local significance.

From a practical perspective, each bridge domain will exist in a particular leaf if there is a connected endpoint that belongs to that endpoint group. Each bridge domain receives a VLAN ID in the leaf switches.

The VLAN ID used is also called the platform independent VLAN or PI VLAN. This VLAN concept is different from traditional networking and is not used to forward traffic, but as an identifier. Each PI VLAN is then linked to a VXLAN ID that will be used for forwarding purposes inside of the fabric.

In the following example, under the Tenant Acme, the bridge domain Acme-Applications-BD was assigned the PI VLAN ID 42 in the Leaf-1.

Endpoint groups are also assigned with a PI VLAN ID that is locally significant in each leaf. This VLAN ID is different from the bridge domain. Therefore in Cisco Application Centric Infrastructure (ACI), several VLANs will be used for endpoints inside on one bridge domain. For more details refer to the endpoint section in this chapter.

When a Subnet is defined in a bridge domain, the leaf switches will be the default gateway for the endpoint groups using that subnet. If the endpoint groups have endpoints on multiple leaves, each leaf will configure the default gateway. In that way, the default gateway for the endpoints will always be the first switch of the fabric that is reached, also know as a pervasive gateway. This means that an SVI will be configured under the
VRF that represents the private network that the bridge domain is linked to. If a bridge domain has several subnets, there will be only one SVI per bridge domain but it will use secondary IP addresses.

**Bridge Domain Configuration Parameters**

- **Name**—The name of the bridge domain.
- **Network**—The associated Layer 3 context.
- **Forwarding**—Optimize/Custom.
- **L2 Unknown Unicast**—The forwarding method for unknown Layer 2 destinations. Default method is **Proxy**, which means that the leaf will forward to the spine for a lookup using a global database. If the destination is not found, it is dropped. The second method is **Flood**, which utilizes a multicast tree rooted in the spine for a specific bridge domain.
- **L3 Unknown Multicast Flooding**—The node forwarding parameter for unknown multicast destinations. You can choose **Flood** or **Optimized Flood**.
- **Multidestination Flood**—This parameter configures the flooding behavior for Layer 2 multidestination flood, such as multicast, broadcast, and link local-specific traffic. You can choose to flood in the bridge domain (default), drop all of the traffic, or flood in an encapsulation or VLAN.
- **ARP Flooding**—A property to specify whether ARP flooding is enabled. If flooding is disabled, unicast routing will be performed on the target IP address. ARP flooding is disabled by default.
- **Unicast Routing**—The forwarding method based on predefined forwarding criteria (IP or MAC address). Unicast routing is enabled by default. Unicast routing uses a destination IP address to forward traffic by creating specific /32 routes in hardware.
- **Config BD MAC Address**—The MAC address of the bridge domain or switched virtual interface (SVI). Every bridge domain by default takes the fabric-wide default MAC address.
- **IGMP Snoop Policy**—The IGMP Snooping policy name. By examining (snooping) IGMP membership report messages from interested hosts, multicast traffic is limited to the subset of VLAN interfaces on which the hosts reside.
- **Associated L3 Outs**—The name of the Layer 3 outside interface associated with this object.
- **L3 Out for Route Profile**—The Layer 3 outside interface identifier controlling connectivity to outside networks.
- **Route Profile**—The associated route profile name.
- **Monitoring Policy**—The monitoring policy name for the tenant semantic scope (optional).
- **Subnets**—The network visibility of the subnet. The subnet is a portion of a network sharing a particular subnet address. The scope can be:
  - **Shared Between VRFs**—Defines subnets under an endpoint group, with the Shared option configured, to route leak to other tenants within the Fabric.
  - **Advertise Externally**—Defines subnets under a bridge domain, with the Public option configured, to share with Layer 3 outbound.
  - **Private to VRF**—Defines subnets under a bridge domain, with the Private option configured, to only be used in that tenant (will not be leaked). The default is **Private**.
• **Subnet Control**—The control can be specific protocols applied to the subnet such as IGMP Snooping. The control can be:
  • **Querier IP**—Enables IGMP snooping on the subnet.
  • **DHCP Labels**—The network domain name label.

Creating a New Bridge Domain Using the GUI

Creating a New Bridge Domain Using the NX-OS-Style CLI

You can use the NX-OS-Style CLI to create a bridge domain.

**Procedure**

Step 1  SSH to an APIC in the fabric.
  
  ```
  # ssh admin@node_name
  ```

Step 2  Enter the configure mode:
  
  ```
  apic1# configure
  ```

Step 3  Enter the configure mode for a tenant:
  
  ```
  apic1(config)# tenant tenant1
  ```

Step 4  Enter the configure mode for a bridge domain:
  
  ```
  apic1(config-tenant)# bridge-domain bd1
  ```

Step 5  Enter the configure mode for a VRF:
  
  ```
  apic1(config-bd)# vrf vrf1
  ```

Step 6  Enter ? for a list of commands.

Modifying a Bridge Domain Using the GUI

Modifying a Bridge Domain Using NX-OS-Style CLI

You can use the NX-OS-style CLI to modify a bridge domain.

**Procedure**

Step 1  SSH to an APIC in the fabric.
  
  ```
  # ssh admin@node_name
  ```

Step 2  Enter the configure mode:
  
  ```
  apic1# configure
  ```
Removing a Bridge Domain Using the GUI

Removing a Bridge Domain Using NX-OS-Style CLI

You can use the NX-OS-style CLI to remove a bridge domain.

Procedure

Step 1 SSH to an APIC in the fabric.
  # ssh admin@node_name

Step 2 Enter the configure mode:
apic1# configure

Step 3 Enter the configure mode for a tenant:
apic1(config)# tenant tenant1

Step 4 Remove the bridge domain:
apic1(config-tenant)# no bridge-domain bd1

Verifying a Bridge Domain Using the Object Model CLI

You can use the object model CLI to verify the bridge domains.

Procedure

Step 1 SSH to an APIC in the fabric.
  # ssh admin@node_name

Step 2 Switch to the object model CLI:
apic1# bash
admin@apic1:~>

Step 3 Verify a concrete bridge domain:
admin@apic1:~> moquery -c l2BD
Step 4  Verify a resolved bridge domain:

admin@apic1:~> moquery -c fvBDDef

Step 5  Verify a local bridge domain:

admin@apic1:~> moquery -c fvBD

Tenant Networking Use Cases

Common Private Network for All Tenants

This use case may be typical for environments where an ACI administrator wishes to create multiple tenants, but place all within a single private network in the fabric.

This method has the following advantages and disadvantages:

Advantages:

• Ability to use a single private network for all internal and external fabric connectivity
• No route leaking needed between EPGs in different VRFs
• Single Layer 3 Outside can be used by all tenants

Disadvantages:

• Changes to routing will impact all tenants

From a containment and relationship perspective, this topology looks as follows:

*Figure 7: Common Private Network for all Tenants*

To Configure the common Tenant private network:

The Tenant has been created. Now the network administrator will have to associate the common private network to the Tenant by first creating a bridge domain.

The configuration for this use case can be applied via the following CLI configuration:

**CLI : Tenant Cisco**

```bash
# tenant
cd '/aci/tenants'
mocreate 'Cisco'
moconfig commit
# bridge-domain
cd '/aci/tenants/Cisco/networking/bridge-domains'
mocreate 'Cisco'
cd 'Cisco'
```
moset network 'default'
moconfig commit
# subnet
cd '/aci/tenants/Cisco/networking/bridge-domains/Cisco/subnets'
mocreate '172.16.0.1/24'
moconfig commit
# application-profile
cd '/aci/tenants/Cisco/application-profiles'
mocreate 'App1'
moconfig commit
# application-epg
cd '/aci/tenants/Cisco/application-profiles/App1/application-epgs'
mocreate 'EPG1'
cd 'EPG1'
omoset bridge-domain 'Cisco'
moconfig commit
# criterion
cd '/aci/tenants/Cisco/application-profiles/App1/application-epgs/EPG1/vm-attributescriteria'
mocreate 'default'
moconfig commit

This configuration can also be applied using the following XML posted to the APIC REST API

**XML : Tenant Cisco**

```xml
<fvTenant name="Cisco">
  <fvBD arpFlood="no" multiDstPktAct="bd-flood" name="Cisco" unicastRoute="yes"
    unkMacUcastAct="proxy" unkMcastAct="flood">
    <fvRstCtx tnFvCtxName="default"/>
    <fvSubnet ctrl="nd" descr="" ip="172.16.0.1/24" preferred="no"
      scope="private"/>
  </fvBD>
  <fvAp name="App1">
    <fvAEPg matchT="AtleastOne" name="EPG1">
      <fvRsBD tnFvBDName="Cisco"/>
    </fvAEPg>
  </fvAp>
  <fvRsTenantMonPol tnMonEPGPOLName=""/>
</fvTenant>
```

For many multi-tenant environments it is desirable to allow each tenant to manage and own their own address space and not be concerned with overlaps between other tenants. This particular use case demonstrates how a private network can be associated with each tenant. One Private Network per Tenant with Intra-EPG communications

**Advantages:**

- Allow for maximum isolation between tenants
- Ability to address hosts in tenants with overlapping IP addresses

**Disadvantages:**

- Increased complexity when needing to allow EPG communication between different tenants with dedicated VRF

The object containment for this particular setup can be depicted as shown below:
To create the tenant:

The configuration for this use case can be applied via the following CLI configuration:

**CLI : Tenant Cisco**

```
# tenant
cd '/aci/tenants'
mocreate 'Cisco'
moconfig commit

# bridge-domain
cd '/aci/tenants/Cisco/networking/bridge-domains'
mocreate 'Cisco'
cd 'Cisco'
omset network 'Cisco'
moconfig commit

# subnet
cd '/aci/tenants/Cisco/networking/bridge-domains/Cisco/subnets'
mocreate '172.16.0.1/24'
moconfig commit

# private-network
cd '/aci/tenants/Cisco/networking/private-networks'
mocreate 'Cisco'
moconfig commit

# application-profile
cd '/aci/tenants/Cisco/application-profiles'
mocreate 'Appl'
moconfig commit

# application-epg
cd '/aci/tenants/Cisco/application-profiles/Appl/application-epgs'
mocreate 'EPG1'
cd 'EPG1'
omset bridge-domain 'Cisco'
moconfig commit
```

This configuration can also be applied using the following XML posted to the APIC REST API:

**XML : Tenant Cisco**

```xml
<fvTenant name="Cisco">
  <fVBD arpFlood="no" multiDstPktAct="bd-flood" name="Cisco" unicastRoute="yes"
       unKMacUcastAct="proxy" unKMcactAct="flood">
    <fvRsCtx tnFvCtxName="Cisco"/>
  </fVBD>
</fvTenant>
```
Multiple Private Networks with Intra-Tenant Communication

Another use case that may be desirable to support is the option to have a single tenant with multiple private networks. This may be a result of needing to provide multi-tenancy at a network level, but not at a management level. It may also be caused by needing to support overlapping subnets within a single tenant, due to mergers and acquisitions or other business changes.

This method has the following advantages and disadvantages:

**Advantages:**
- Ability to have overlapping subnets within a single tenant

**Disadvantages:**
- EPGs residing in overlapping subnets cannot have policy applied between one another

The object containment for this particular setup can be depicted as shown below:

*Figure 9: Multiple Private Networks with Intra-Tenant Communication*

To create the tenant:

The configuration for this use case can be applied via the following CLI configuration:

**CLI : Tenant Cisco**

```bash
# tenant
cd '/aci/tenants'
```
mocreate 'Cisco'
moconfig commit
# bridge-domain
cd '/aci/tenants/Cisco/networking/bridge-domains'
mocreate 'Cisco'
cd 'Cisco'
moset network 'Cisco'
moconfig commit
# bridge-domain
cd '/aci/tenants/Cisco/networking/bridge-domains'
mocreate 'Cisco1'
cd 'Cisco1'
moset network 'Cisco1'
moconfig commit
# private-network
cd '/aci/tenants/Cisco/networking/private-networks'
mocreate 'Cisco'
moconfig commit
# private-network
cd '/aci/tenants/Cisco/networking/private-networks'
mocreate 'Cisco1'
moconfig commit
# application-profile
cd '/aci/tenants/Cisco/application-profiles'
mocreate 'App1'
moconfig commit
# application-epg
cd '/aci/tenants/Cisco/application-profiles/App1/application-epgs'
mocreate 'EPG1'
cd 'EPG1'
moset bridge-domain 'Cisco'
moconfig commit
# fv-rscon
cd '/aci/tenants/Cisco/application-profiles/App1/application-epgs/EPG1/contracts/consumed-contracts'
mocreate 'ICMP'
moconfig commit
# fv-subnet
cd '/aci/tenants/Cisco/application-profiles/App1/application-epgs/EPG1/subnets'
mocreate '172.16.1.1/24'
cd '172.16.1.1:24'
moset scope 'private,shared'
moconfig commit
# application-epg
cd '/aci/tenants/Cisco/application-profiles/App1/application-epgs'
mocreate 'EPG2'
cd 'EPG2'
moset bridge-domain 'Cisco1'
moconfig commit
# fv-rsprov
cd '/aci/tenants/Cisco/application-profiles/App1/application-epgs/EPG2/contracts/provided-contracts'
mocreate 'ICMP'
moconfig commit
# fv-subnet
cd '/aci/tenants/Cisco/application-profiles/App1/application-epgs/EPG2/subnets'
mocreate '172.16.2.1/24'
cd '172.16.2.1:24'
moset scope 'private,shared'
moconfig commit

This configuration can also be applied using the following XML posted to the APIC REST API:

XML : Tenant Cisco
Multiple Private Networks with Inter-Tenant Communication

This use case may be typical for environments where an ACI administrator wishes to create multiple tenants with the ability to support inter-tenant communications.

This method has the following advantages and disadvantages:

Advantages

- Each tenant container can be managed separately
- Allows for maximum isolation between tenants

Disadvantages

- Tenant address space must be unique

From a containment and relationship perspective, this topology looks as follows:
To create the tenant:

The configuration for this use case can be applied via the following CLI configuration:

**CLI : TENANT Cisco1**

```bash
# tenant
cd '/aci/tenants'
mocreate 'Cisco1'
moconfig commit
# bridge-domain
cd '/aci/tenants/Cisco1/networking/bridge-domains'
mocreate 'Cisco1'
cd 'Cisco1'
moset network 'Cisco1'
moconfig commit
# private-network
cd '/aci/tenants/Cisco1/networking/private-networks'
mocreate 'Cisco1'
moconfig commit
# application-profile
cd '/aci/tenants/Cisco1/application-profiles'
mocreate 'App1'
moconfig commit
# application-epg
cd '/aci/tenants/Cisco1/application-profiles/App1/application-epgs'
mocreate 'EPG1'
cd 'EPG1'
moset bridge-domain 'Cisco1'
moconfig commit
# fv-rsprov
cd '/aci/tenants/Cisco/application-profiles/CCO/applicationepgs/App/contracts/provided-contracts'
mocreate 'ICMP'
moconfig commit
# fv-subnet
cd '/aci/tenants/Cisco/application-profiles/CCO/application-epgs/App/subnets'
mocreate '172.16.1.1/24'
```
cd '172.16.1.1:24'
moset scope 'private,shared'
moconfig commit
# contract
cd '/aci/tenants/Cisco/security-policies/contracts'
mocreate 'ICMP'
cd 'ICMP'
moset scope 'global'
moconfig commit
# contract-subject
cd '/aci/tenants/Cisco/security-policies/contracts/ICMP/subjects'
mocreate 'icmp'
moconfig commit
# vz-rssubjfilatt
cd '/aci/tenants/Cisco/security-policies/contracts/ICMP/subjects/icmp/common-filters'
mocreate 'icmp'
moconfig commit

CLI : TENANT Cisco2

# tenant
cd '/aci/tenants'
mocreate 'Cisco'
moconfig commit
# bridge-domain
cd '/aci/tenants/Cisco/networking/bridge-domains'
mocreate 'Cisco'
cd 'Cisco'
moset network 'Cisco'
moconfig commit
# private-network
cd '/aci/tenants/Cisco/networking/private-networks'
mocreate 'Cisco'
moconfig commit
# application-profile
cd '/aci/tenants/Cisco/application-profiles'
mocreate 'App1'
moconfig commit
# application-epg
cd '/aci/tenants/Cisco2/application-profiles/App1/application-epgs'
mocreate 'EPG1'
cd 'EPG1'
moset bridge-domain 'Cisco'
moconfig commit
# fv-rsconsif
cd '/aci/tenants/Cisco1/application-profiles/CCO/applicationepgs/
Web/contracts/conaued-contract-interfaces'
mocreate 'CiscoInterTenantICMP'
moconfig commit
# fv-subnet
cd '/aci/tenants/Cisco1/application-profiles/CCO/application-epgs/Web/subnets'
mocreate '172.16.2.1/24'
cd '172.16.2.1:24'
moset scope 'shared-subnet'
moconfig commit
# imported-contract
cd '/aci/tenants/Cisco1/security-policies/imported-contracts'
mocreate 'CiscoInterTenantICMP'
cd 'CiscoInterTenantICMP'
moset contract 'tenants/Cisco/security-policies/contracts/ICMP'
moconfig commit

This configuration can also be applied using the following XML posted to the APIC REST API:
Multiple Private Networks with Inter-Tenant Communication

XML : TENANT Cisco1

<vfTenant dn="uni/tn-Cisco1" name="Cisco1">
  <vzBrCP name="ICMP" scope="global">
    <vzSubj consMatchT="AtleastOne" name="icmp" provMatchT="AtleastOne" revFltPorts="yes">
      <vzRsSubjFiltAtt tnVzFilterName="icmp"/>
    </vzSubj>
  </vzBrCP>
  <vzCPIf dn="uni/tn-Cisco1/cif-ICMP" name="ICMP">
    <vzRsIf consMatchT="AtleastOne" name="icmp" provMatchT="AtleastOne" revFltPorts="yes">
      <vzRsSubjFiltAtt tDn="uni/tn-Cisco2/brc-default"/>
    </vzRsIf>
  </vzCPIf>
  <fvCtx knwMcastAct="permit" name="CiscoCtx" pcEnfPref="enforced"/>
  <fvBD arpFlood="yes" mac="00:22:BD:F8:19:FF" name="CiscoBD2" unicastRoute="yes" unkMacUcastAct="flood" unkMcastAct="flood">
    <fvRsCtx tnFvCtxName="CiscoCtx2"/>
  </fvBD>
  <fvBD arpFlood="yes" name="CiscoBD" unicastRoute="yes" unkMacUcastAct="flood" unkMcastAct="flood">
    <fvRsCtx tnFvCtxName="CiscoCtx"/>
  </fvBD>
  <fvAp name="CCO">
    <fvAEPg matchT="AtleastOne" name="EPG1">
      <fvRsPathAtt encap="vlan-1202" instrImedcy="immediate" mode="native" tDn="topology/pod-1/paths-202/pathep-[eth1/2]"/>
      <fvSubnet ip="172.16.1.1/24" scope="private,shared"/>
    </fvAEPg>
    <fvRsDomAtt instrImedcy="lazy" resImedcy="lazy" tDn="uni/phys-PhysDomainforCisco"/>
    <fvRsBd tnFvBDName="CiscoBD2"/>
    <fvRsProv matchT="AtleastOne" tnVzBrCPName="ICMP"/>
  </fvAp>
</vfTenant>

XML : TENANT Cisco2

<vfTenant dn="uni/tn-Cisco2" name="Cisco2">
  <fvCtx knwMcastAct="permit" name="CiscoCtx" pcEnfPref="enforced"/>
  <fvBD arpFlood="yes" mac="00:22:BD:F8:19:FF" name="CiscoBD2" unicastRoute="yes" unkMacUcastAct="flood" unkMcastAct="flood">
    <fvRsCtx tnFvCtxName="CiscoCtx2"/>
  </fvBD>
  <fvBD arpFlood="yes" name="CiscoBD" unicastRoute="yes" unkMacUcastAct="flood" unkMcastAct="flood">
    <fvRsCtx tnFvCtxName="CiscoCtx"/>
  </fvBD>
  <fvAp name="CCO">
    <fvAEPg matchT="AtleastOne" name="EPG2">
      <fvRsPathAtt encap="vlan-1202" instrImedcy="immediate" mode="native" tDn="topology/pod-1/paths-202/pathep-[eth1/2]"/>
      <fvSubnet ip="172.16.1.1/24" scope="private,shared"/>
    </fvAEPg>
  </fvAp>
</vfTenant>
Dual Fabrics with a Common Pervasive Gateway

This use case might be typical for environments where an Cisco Application Centric Infrastructure (ACI) administrator wishes to manage dual fabrics and be able to seamlessly move workload between the fabrics.

Advantages

• Virtual machines can be migrated between hypervisor hosts that are connected to different fabrics
• Allows for seamless workload migration

Disadvantages

• Coordination between the configuration of the dual fabrics is required

The objective of the "Routing over Layer 2 out/Common pervasive gateway" feature is to allow an endpoint to seamlessly move from one fabric to another.

The following information applies to dual fabrics with a common pervasive gateway:

• Assumes dual fabrics are in use.
• "Routed" in this case means traffic between endpoints in different bridge domains (subnets).
• Bridge domains are mirrored between fabrics (with some exceptions).
• You may hear the term 'stretched bridge domain', when in fact it means a bridge domain that is mirrored between fabrics. It is still 2 separate bridge domains, one in each of the independent fabrics.
• There are no endpoints on the Layer 2 network using the bridge domain IP address as their gateway.
• The user can manually configure each fabric with regards to the bridge domains, or use some orchestration tool to distribute the synced configuration to each fabric.
• The ACI tool kit has an application that can help with this.

Using traffic between VM16 and VM17, we will explain why you need this feature.
If you look at the figure above, you see that VM17, when sending packets to VM16, will send to its default gateway. The packet is then 'routed' from BD17 to BD16. BD16 then forwards the packet from the BD16 SMAC (source MAC), onto VM16 as a Layer 2 packet. When VM16 responds, the same occurs in reverse order. The result is that the Layer 2 network, here depicted as a Nexus 5K, will continually learn the BD16/BD17 SMAC from a different port.

You must maintain the same bridge domain IP address and MAC address between the fabrics so that when the VM migrates across the fabrics, it will not have to ARP for a new MAC for its default gateway. As you can see, using the same SMAC from both fabrics poses a problem.

The solution introduces some additional configuration on the bridge domain. In the new release (1.2), you can now configure the bridge domain subnet IP address as 'virtual'. This is the address that will function as the VMs default gateway. You also can configure a VMAC (virtual MAC). This is the MAC that the VMs will resolve to when they issue ARP for their default gateway. These must match between fabrics for the bridge domain.

You also now must configure a CMAC (configured MAC) for the bridge domain, and second IP address in the same subnet as the virtual IP. These must be unique between the fabrics for the bridge domain.

You use the CMAC as the SMAC for routed packets, and you use the second IP for ARP gleaning and endpoint tracking. You will see this explained later.
In the above diagram, you see that BD16 and BD17 are replicated on both the top and bottom fabrics. You see the addition of the PIP and CMAC in the bridge domain configurations, and note that they are different between fabrics, while the VIP and VMAC are the same between fabrics.

For the packet flow, VM17 sends a packet to VM16. BD17 in the bottom fabric routes the packet to BD16 in the bottom fabric. BD16 in the bottom fabric uses the configured CMAC as the SMAC when forwarding the frame to VM16. The N5K learns MAC 0022.1616.0002 from the bottom fabric. When VM16 in the top fabric responds, the same occurs in the top fabric. BD17 in the top fabric will use its configured CMAC as the SMAC for the packet as it exits the Layer 2 out. The N5K now learns that MAC 0022.1717.0001 from the top fabric. The Nexus 5K will no longer see a MAC flap when packets routed between bridge domains pass though it.

Key to note here is that even if VM17 moves to the top fabric, and VM16 remains on the top fabric, the bridge domains will still use their configured CMAC as the SMAC when they forward routed packets to endpoints.
Using VM17 as a LINUX host, you can use TCPDUMP to confirm that the MAC address for packets to VM16 go to the VMAC, and that return traffic is received from the CMAC.

```
[root@localhost ~]# ping 16.1.1.2
PING 16.1.1.2 (16.1.1.2) 56(84) bytes of data.
64 bytes from 16.1.1.2: icmp_seq=1 ttl=64 time=0.332 ms
64 bytes from 16.1.1.2: icmp_seq=2 ttl=64 time=0.376 ms
64 bytes from 16.1.1.2: icmp_seq=3 ttl=64 time=0.383 ms
--- 16.1.1.2 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2001ms
rtt min/avg/max/mdev = 0.332/0.363/0.383/0.031 ms
```

```
[root@localhost ~]# tcpdump -e -i eth1 -n
```

Traffic from BD17 to BD16 is routed, and SMAC and SRC_IP are changed in local fabric.
Using the `tcpdump` command with the `-e` switch, we can see that a PING from 17.1.1.2 to 16.1.1.2 does use the VMAC as the SMAC for the PING request, and that the PING response is seen from the CMAC configured on the BD that the endpoint is in.

The first set of highlighted text shows the ICMP echo request (PING), which is sent to the configured VMAC for BD17. The second set of highlighted text shows the ICMP echo response, which is received from the BD17 configured CMAC, and not the VMAC.

The each fabric will send a GARP sourced from the VIP and the VMAC every 30 seconds (by default). This should not cause any problems for the Layer 2 network, as these GARPs are at a low rate. Using the Nexus 5000 `debug ip arp packet` command, you can see the frequency.

We use the second IP configured on the bridge domain in use. In our example above, BD17 learned EP 17.1.1.230 via the L2 out.

The following example output is from the `debug ip arp packet` command on the Nexus 5000 in the topology. You can see the BD17 PIP and CMAC as they perform endpoint tracking. The frames originated from the ACI leaf using the configured BD17 PIP and CMAC to confirm that endpoint 17.1.1.1 is still alive on the Layer 2 endpoint group.

We use the second IP configured on the bridge domain in use. In our example above, BD17 learned EP 17.1.1.230 via the L2 out.
### Configuring Dual Fabrics with a Common Pervasive Gateway Using the GUI

The following procedure describes how to configure dual fabrics with a common pervasive gateway using the GUI for this use case.

#### Procedure

<table>
<thead>
<tr>
<th>Command or Action</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Step 1</strong></td>
<td>On the menu bar, choose <strong>Tenant &gt; All TENANTS</strong>.</td>
</tr>
<tr>
<td><strong>Step 2</strong></td>
<td>In the Work pane, double click the <strong>Tenant_Name</strong>.</td>
</tr>
<tr>
<td><strong>Step 3</strong></td>
<td>In the Navigation Pane, choose <strong>Tenant_Name &gt; Networking &gt; Bridge Domains &gt; BD_name</strong>.</td>
</tr>
<tr>
<td><strong>Step 4</strong></td>
<td>In the <strong>Work</strong> pane, set the following values for the properties:</td>
</tr>
<tr>
<td><strong>Step 5</strong></td>
<td>Choose the <strong>L3 Configurations</strong> tab.</td>
</tr>
<tr>
<td><strong>Step 6</strong></td>
<td>In the <strong>Work</strong> pane, set the following values for the properties:</td>
</tr>
<tr>
<td><strong>Step 7</strong></td>
<td>Click <strong>Submit</strong>.</td>
</tr>
<tr>
<td><strong>Step 8</strong></td>
<td>In the Navigation Pane, choose <strong>Tenant_Name &gt; Networking &gt; Bridge Domains &gt; BD_name &gt; Subnets &gt; Subnet_IP</strong>.</td>
</tr>
<tr>
<td><strong>Step 9</strong></td>
<td>In the <strong>Work</strong> pane, put a check in the <strong>Treat as virtual IP address</strong> box. <strong>This configures the bridge domain gateway IP address as a VIP (virtual IP address).</strong> Also under the <strong>Subnets</strong> folder is an 16.1.1.103 address that is used for ARP gleaning and endpoint tracking. This address must be unique for the bridge domain across the fabrics.</td>
</tr>
<tr>
<td><strong>Step 10</strong></td>
<td>Repeat steps 3 to 9 for each bridge domain that is mirrored between the fabrics.</td>
</tr>
</tbody>
</table>
Configuring Dual Fabrics with a Common Pervasive Gateway Using the GUI