Table of Contents

ACI sandbox

Overview:

The New ACI Sandbox version 4 release provides a developer with an environment to design, develop and test using the ACI RESTful APIs over http/https with XML and JSON encodings. The central SDN Controller of the ACI solution, the Application Policy Infrastructure Controller (APIC) manages and configures the policy on each of the switches in the ACI fabric. The primary function of Cisco APIC is to provide policy authority and policy resolution mechanisms for the Cisco ACI fabric and devices attached to the fabric. Automation is provided as a direct result of policy resolution and renders its effects on the Cisco ACI fabric, so that end users no longer have to touch each network element and manually make sure that all policies are configured appropriately.

Connecting to the ACI Server:

This ACI Sandbox Lab contains a shared ACI Server Instance.
The server can be connected using login credentials **admin** and password : **!v3G@!4@Y** and is located at

- https://sandboxapicdc.cisco.com

Lab Architecture: Diagram and network element information about the ACI Sandbox Topology

Good Citizen Code of Conduct:

This ACI Server resource is shared. This means that you can
see other developer's resources, and they can see yours. Please, do not erase resources you have not created yourself. Do not upload very large resources or you may drain capacity from other users. Also, do not perform performance testing against this shared instance of ACI.

Looking for tutorials to get started with ACI?

Check out the ACI Learning Labs! The Learning Labs contain helpful tutorials for basic coding, using Cisco APIs and more!

Lab 01: Understanding ACI

https://developer.cisco.com/learning/modules/intro-to-aci/sbx-intro-aci-01_understanding-aci/step/1

Step 1: Introduction

NX-OS Mode vs. ACI
The Nexus 9000 has two modes of operation to fit different operational models: NX-OS mode and Application Centric Infrastructure (ACI) mode.

In the NX-OS mode of operations, the Nexus 9000 platform utilizes a traditional operating system with enhancements to provide a platform for existing network deployments, while maintaining the capability to leverage next-generation data center protocols and technologies. In NX-OS mode, there is the ability for standalone or device by device level programmability using APIs, such Cisco's Nexus NX-API.

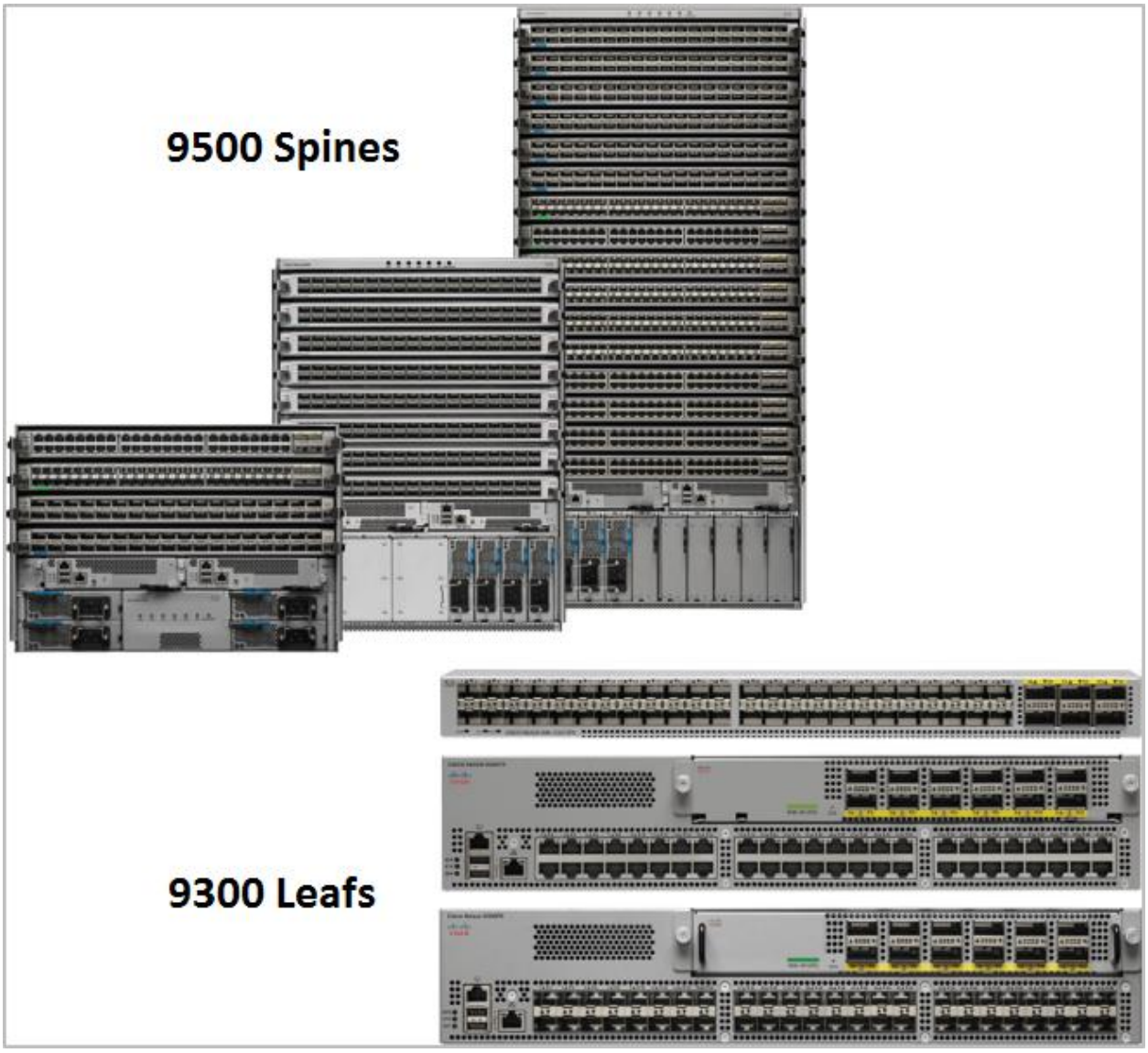
With Cisco Nexus 9000 in ACI mode, the infrastructure is centrally managed by a cluster of controllers, the Application Policy Infrastructure Controllers (APICs). ACI is Cisco’s core Software-Defined Networking (SDN) solution for the data center.

While NX-OS and ACI modes have independent road maps and feature sets, the common hardware platform provides flexibility, choice, and value. The common hardware being referred to is the Nexus platform, specifically the Nexus 9000 series.

ACI Hardware
On most Nexus 9000 hardware, you can perform a software upgrade to migrate hardware to ACI, a SDN-focused model of operations.

In ACI mode, switches operate as either a Spine or a Leaf.

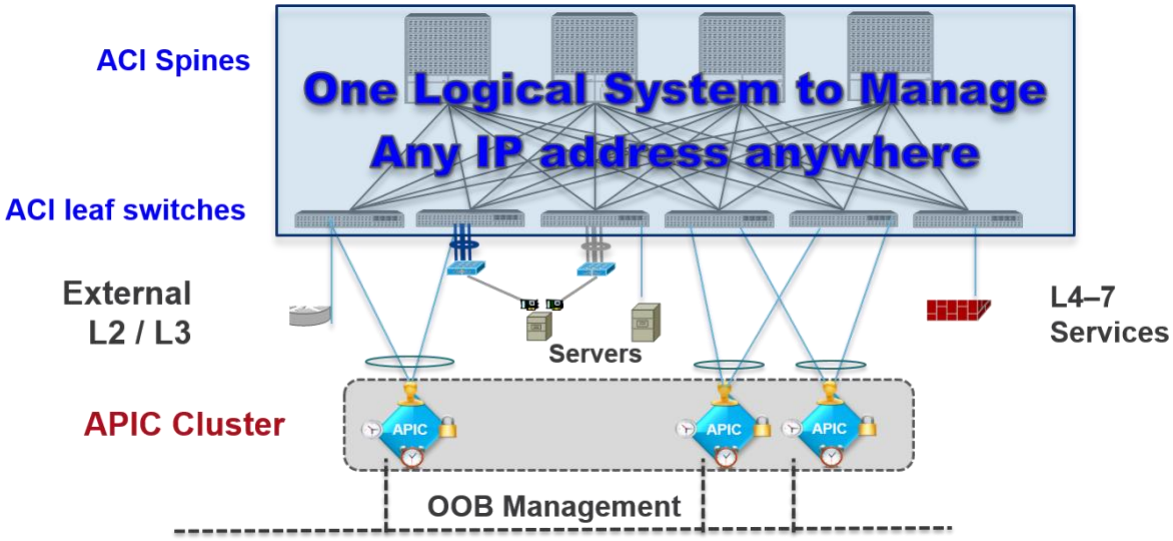
- Spine switches are used to aggregate leaf switches.
- Leaf switches are used as access devices.



Step 2: Dive deeper into ACI

The Nexus switches running in ACI mode are only programmable via an object-based Policy Engine operating in the APIC controller. The controller is integrated part of the network and holds profiles containing the policies for programming the switches centrally. The switches themselves do not maintain a CLI configuration file as previously used in NX-OS based systems.

The configuration is held on the APIC using an object-oriented schema. It is represented with either XML or JSON and stored in a profile to implement application-driven, network-driven, and security-driven policies.



ACI Components

The core components of an ACI deployment include the following:

- **Leaf Switches** provide connectivity into the Fabric at the Top-of-Rack (ToR) or End-of_Row (EoR). They serve as distributed layer 3 gateways, the policy enforcement points, and gateways into external networks.
- **Border Leaf Switches** are any leaf nodes that connect to a network device external to the ACI fabric, such as firewalls, load balancers, routers, or non-ACI switches; allowing a smooth migration to an ACI network.
- **Spine Switches** provide a non-blocking fabric with rapid failure detection and re-routing. These are used to forward traffic between two leaf switches. Beginning with Software version 2.0(2), ACI supports Layer 3 connections with Ethernet VPN (EVPN) to the spine switches.
- **APIC Controllers** provide the centralized point of management for fabric configuration and observing the summary operational state. From a policy perspective, the APIC is the primary point of contact for configuration and acts as the policy repository.

Next step: ACI terminology

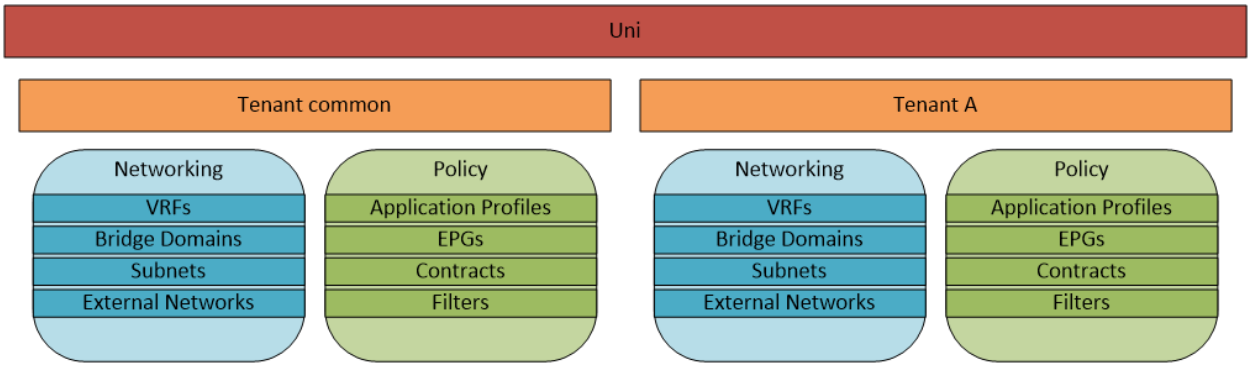
Step 3: ACI terminology

It was previously mentioned that the APICs operate using an object-based Policy Engine. This enables the network administrator to define desired states of the fabric, but leave the implementation up to the controller (APIC). As workloads move, the controller re-configures the underlying infrastructure to ensure the necessary policies are still in place for the end hosts. The

branch of the object-model these policies are defined under is the **Tenant** object. This is where most of the day-to-day operations exist.

ACI Tenants

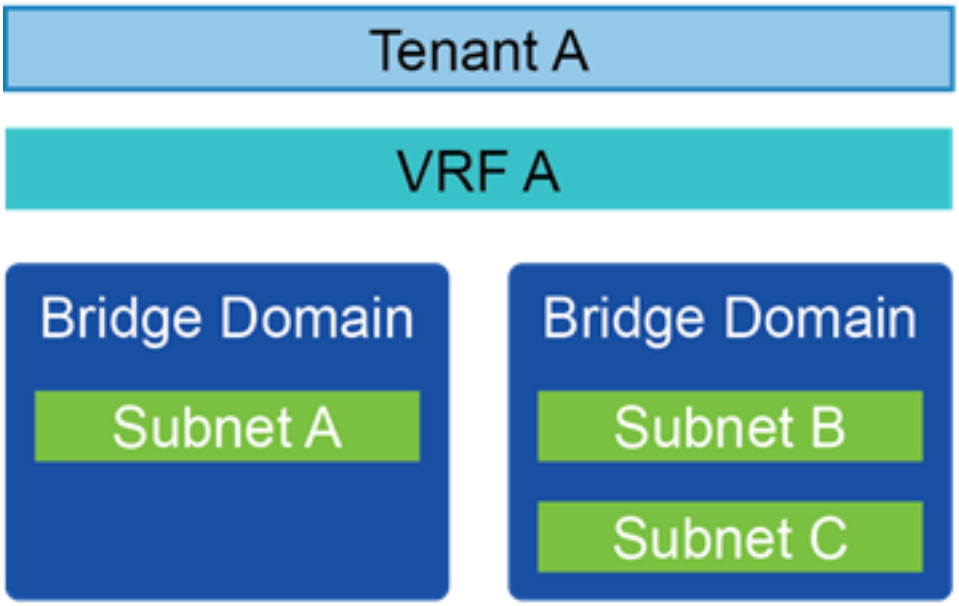
Tenants are a top level object that identify and separate administrative control, network/failure domains, and application policies. A Tenant's sub-level objects can be grouped into two basic categories: Tenant Networking and Tenant Policy. These two categories do have inherent relationships, but it can be helpful to discuss them separately.



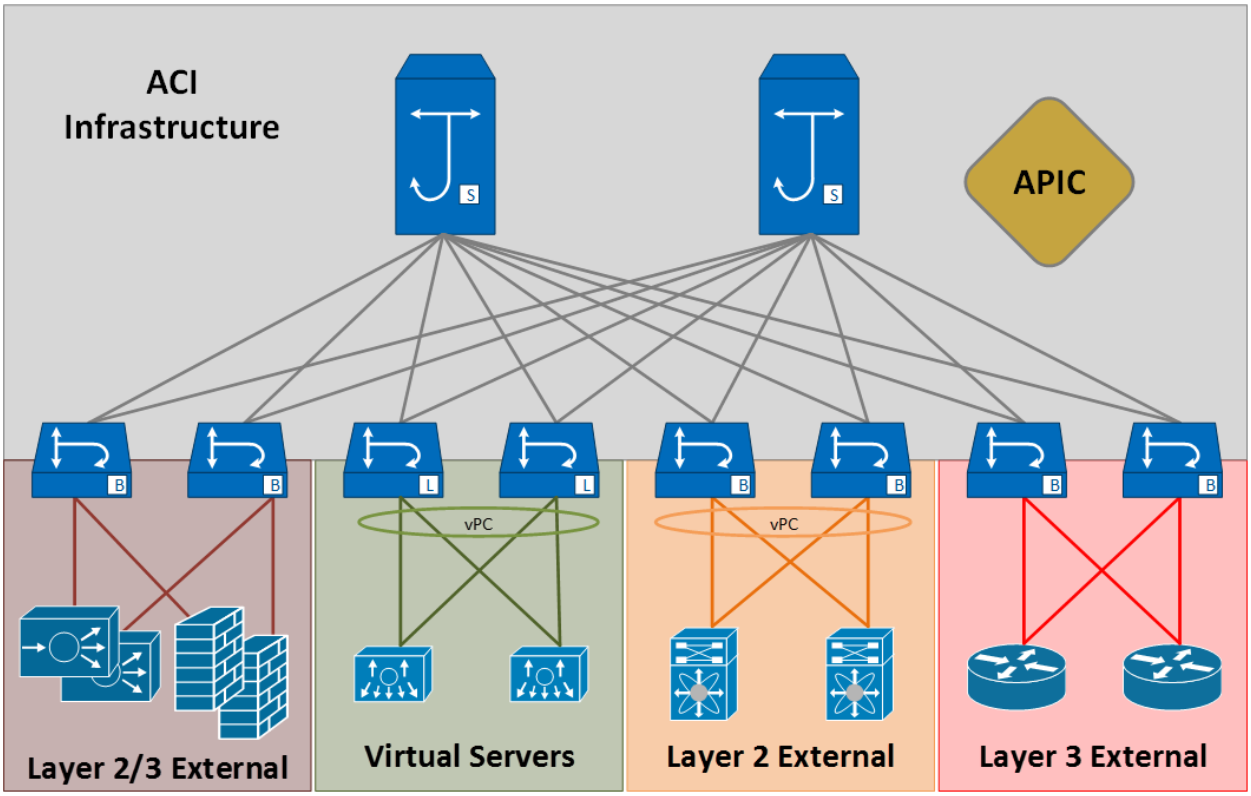
Tenant "common"
ACI has a special Tenant named "common" that has unique abilities to easily share its resources with other Tenants. The "common" Tenant is designed to provide shared resources to the entire ACI fabric, such as DNS, authentication services, security tools, and others. You can also define all of the Tenant Networking objects, Contracts, and Filters in the "common" Tenant and associate these with Applications in other Tenants.

Step 4: Tenant networking

The Tenant Networking objects are similar to network constructs engineers are already familiar with, and provide Layer 2 and Layer 3 connectivity between hosts. Tenant Networking consists of VRFs (Virtual Routing and Forwarding), Bridge Domains, Subnets, and External Networks.



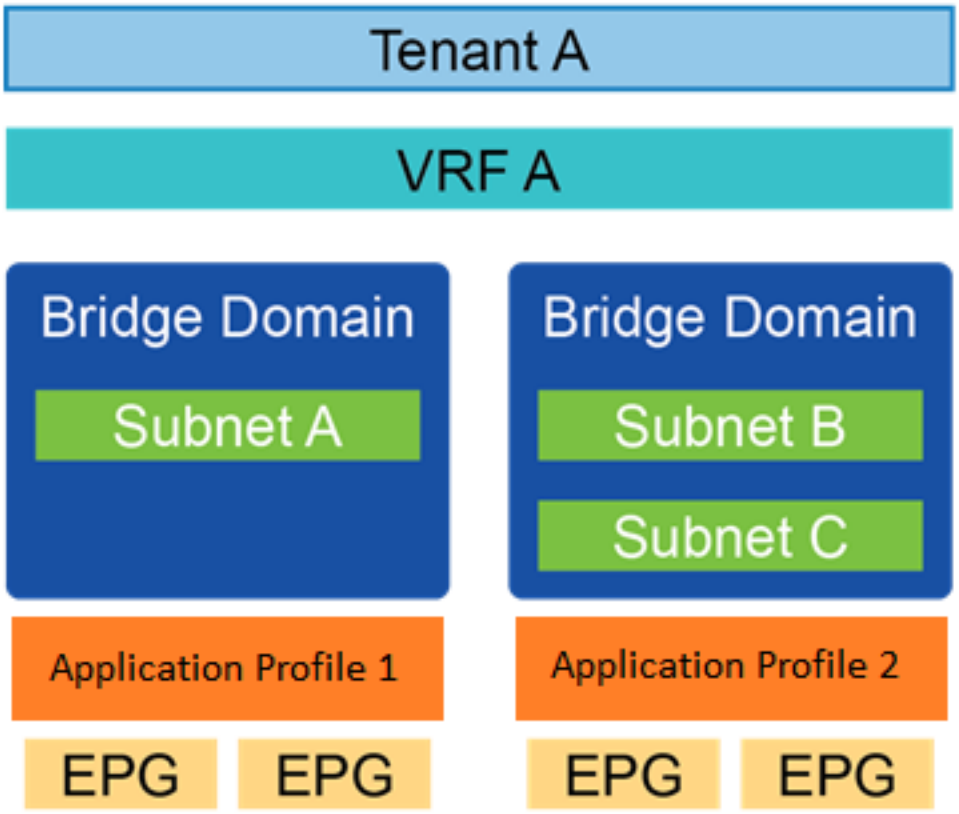
- **VRFs**, also named Contexts and Private Networks, are isolated routing tables for the Tenant. A Tenant can have one or many VRFs, or could use a VRF from the "common" Tenant. In the following diagram, the **infra** VRF is in the grey area. All additional VRFs exist on any leaf that has a host assigned to the VRF.
- **Bridge Domains** are the Layer 2 forwarding domains within the fabric, and define the unique MAC address space and flooding domain (Broadcast, Unknown Unicast, and Multicast). Each Bridge Domain is associated with only one VRF, however a VRF can be associated with many Bridge Domains. Unlike the way VLANs have been traditionally deployed, each Bridge Domain can contain multiple subnets.
- **Subnets** are the Layer 3 networks that provide IP space and gateway services for hosts to connect to the network. Each subnet is associated with only one Bridge Domain.
- **External Bridged Networks** connect a Layer 2/Spanning-Tree Network to the ACI fabric. This is commonly used in brownfield environments to have a smooth migration from a traditional network infrastructure to an ACI network. The following diagram also has Layer 2 External networks for L4-L7 devices.
- **External Routed Networks** create a Layer 3 adjacency with a network outside of the ACI fabric. Layer 3 External networks support adjacencies using static routes or BGP, OSPF, and EIGRP. Layer 3 connections also have networks defined, which provide and consume contracts (discussed more in Tenant Policy).



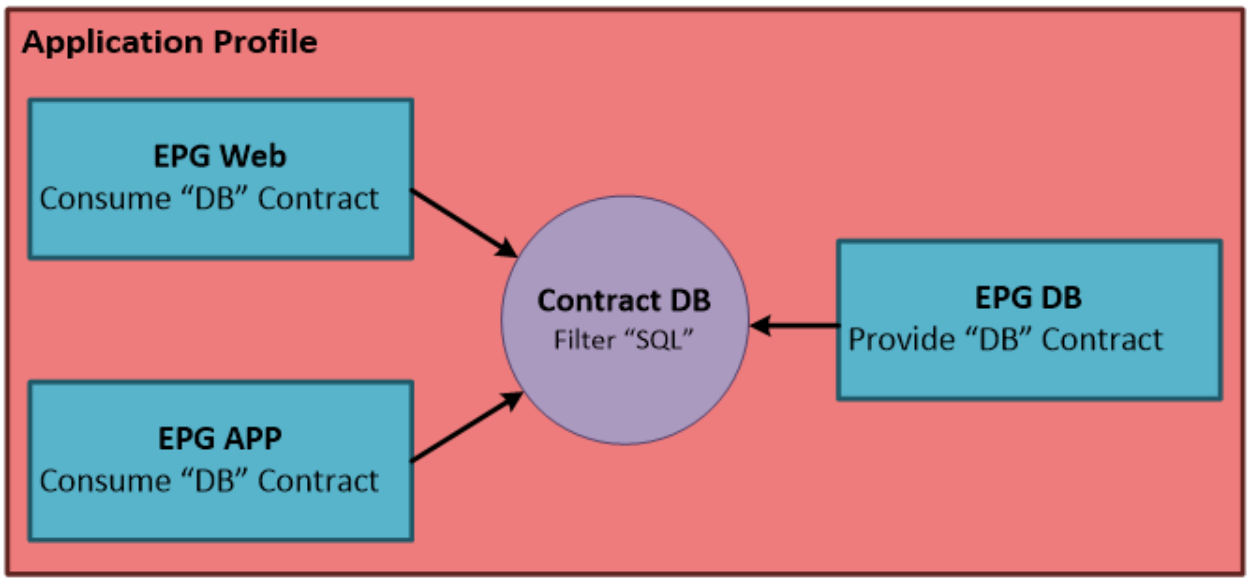
Next step: Tenant policy

Step 5: Tenant policy

The Tenant Policy objects are related to the network objects, but the Tenant Policy objects are more focused on the policies and services the endpoints receive. The Tenant Policy consists of Application Profiles, End Point Groups, Contracts, and Filters. These are all new and ACI specific terms.



- An **Application Profile** is a container for End Point Groups (EPGs). The Application Profile is used as an identifier for Applications, and allows for separation of administrative privileges.
- An **End Point Group (EPG)** is a collection of endpoints that have the same services and policies applied. EPGs define the switchports, virtual switches, and Layer 2 encapsulations associated with an application service. Each EPG can only be related to one Bridge Domain.
- **Contracts** define the services and policies applied to the end points in an EPG. Contracts can be used for service redirection to an L4-L7 device, assigning QoS values, and applying ACL rules. The EPG that offers the services **provides** the contract, and the EPGs that need to use the service **consume** the contract.
- **Filters** are the objects that define protocols (tcp, udp, icmp, and so on) and ports. Filter objects can contain multiple protocols and ports, and Contracts can consume multiple Filters.



Now that you have an understanding of the most common Tenant concepts, let's configure an Application in the APIC GUI.

Next step: ACI GUI Walkthrough

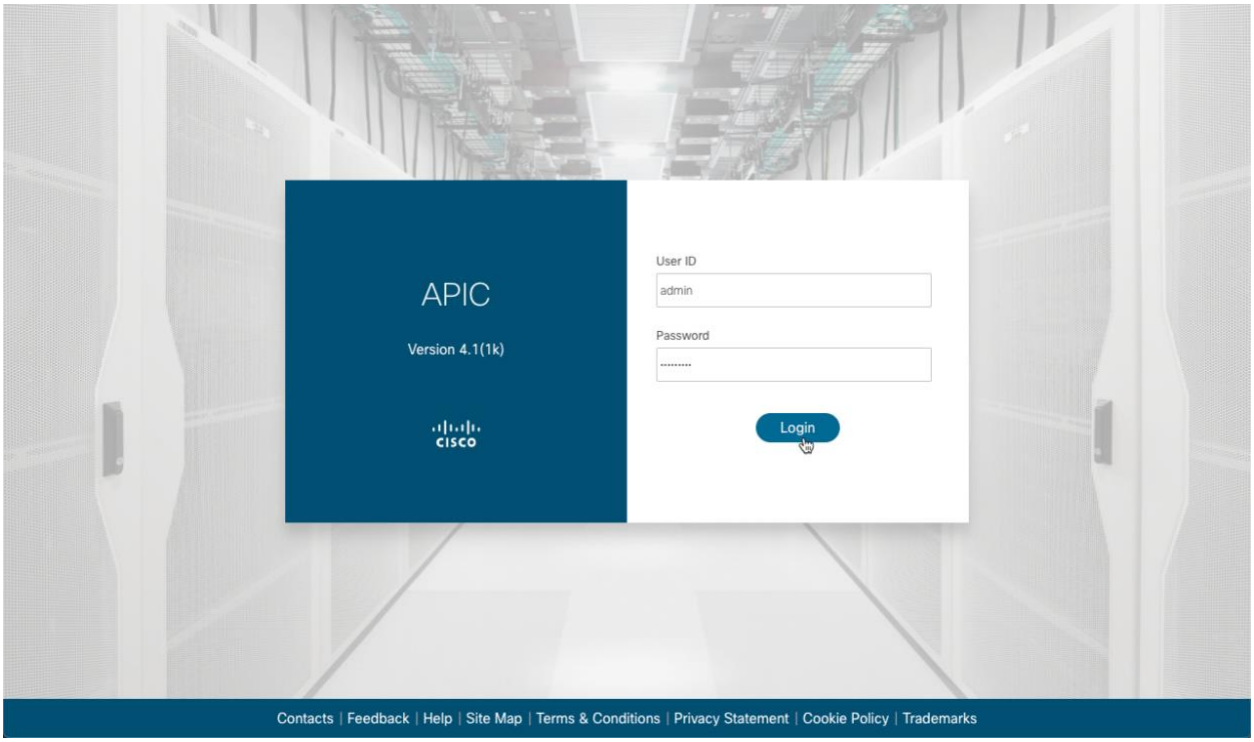
Exercise 1 (optional): ACI GUI Walkthrough

A Tenant named "Heroes" has been configured in the APIC GUI. To begin the walk-through, browse to the APIC and login.

URL: https://sandboxapicdc.cisco.com

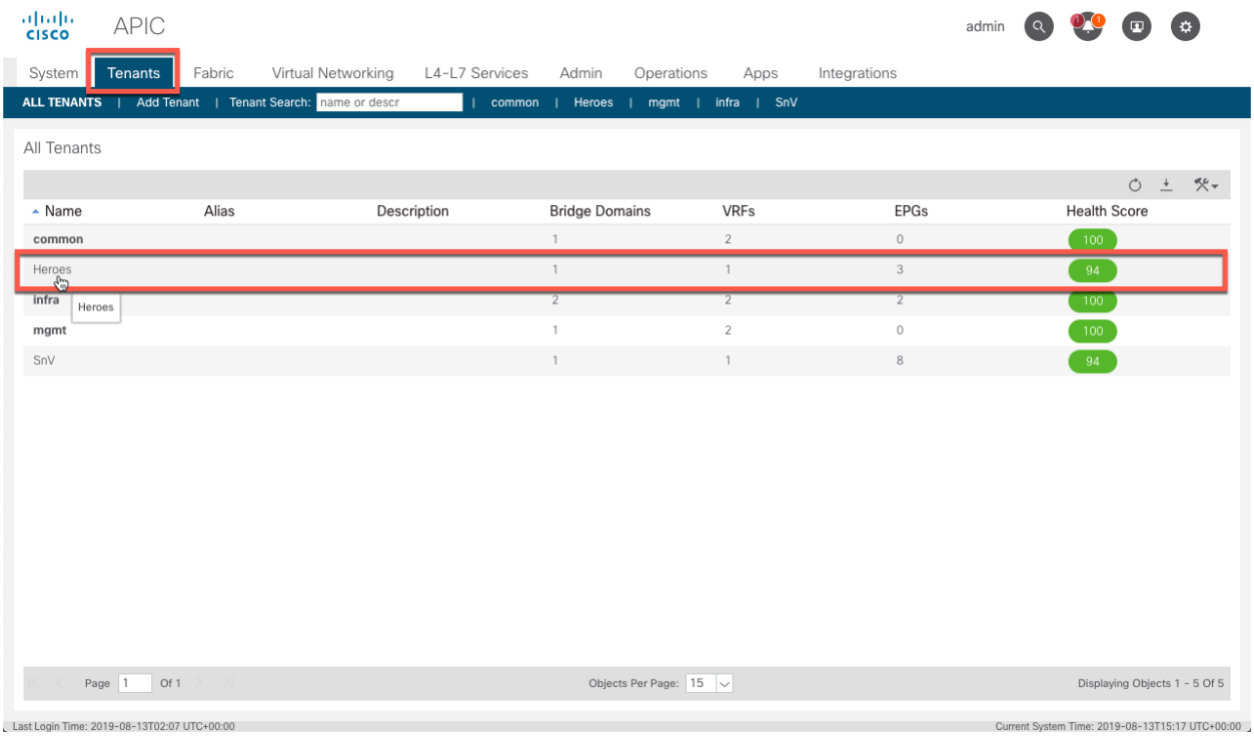
User: admin

Password: : !v3G@!4@Y



Browse to the "Heroes" Tenant:

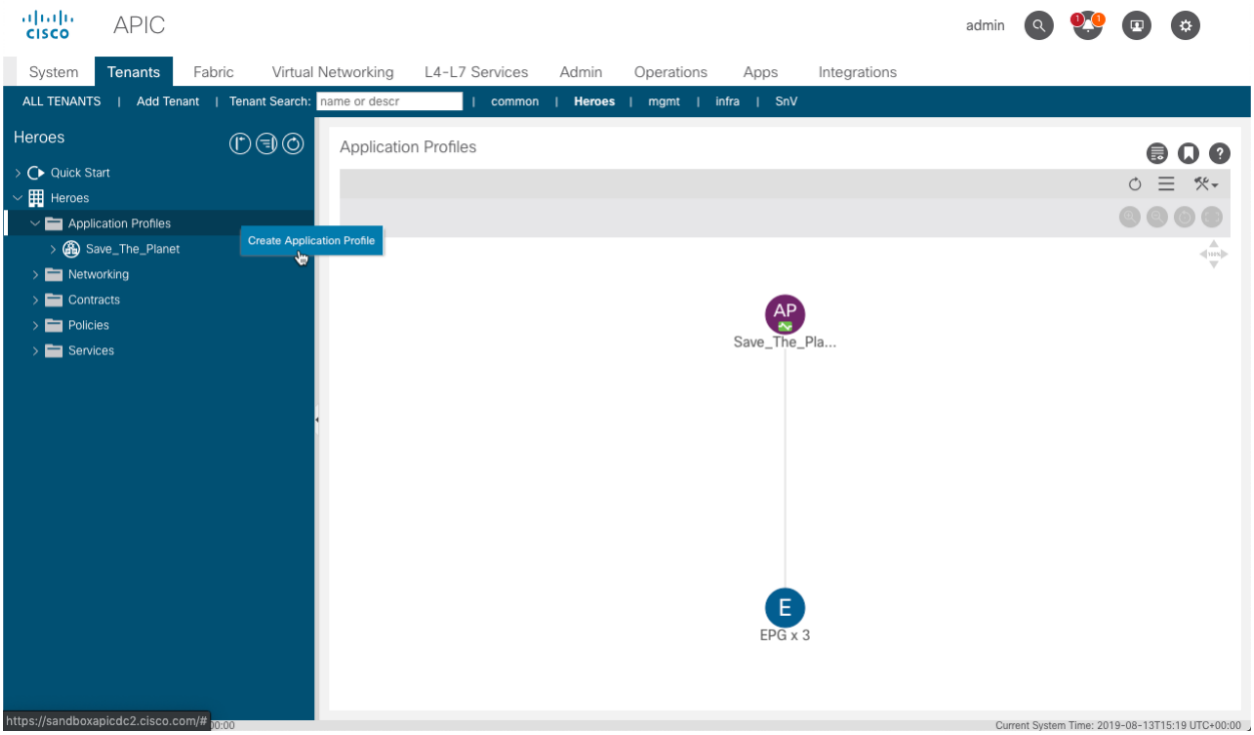
1. Click **Tenants**.
2. Double-click **Heroes**.



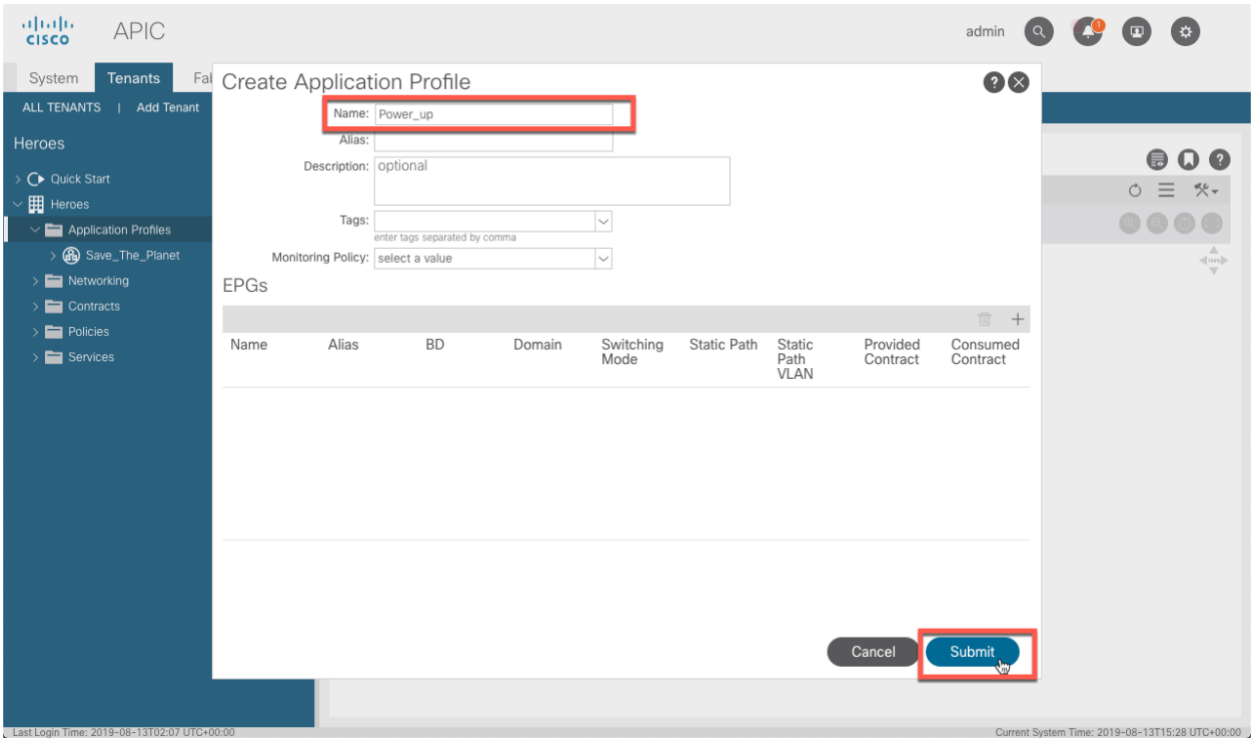
Create an Application Profile

You will create a new 2-tier application, and use existing contracts to provide and consume services. In the GUI:

1. Right click **Application Profiles**.
2. Choose **Create Application Profile**.



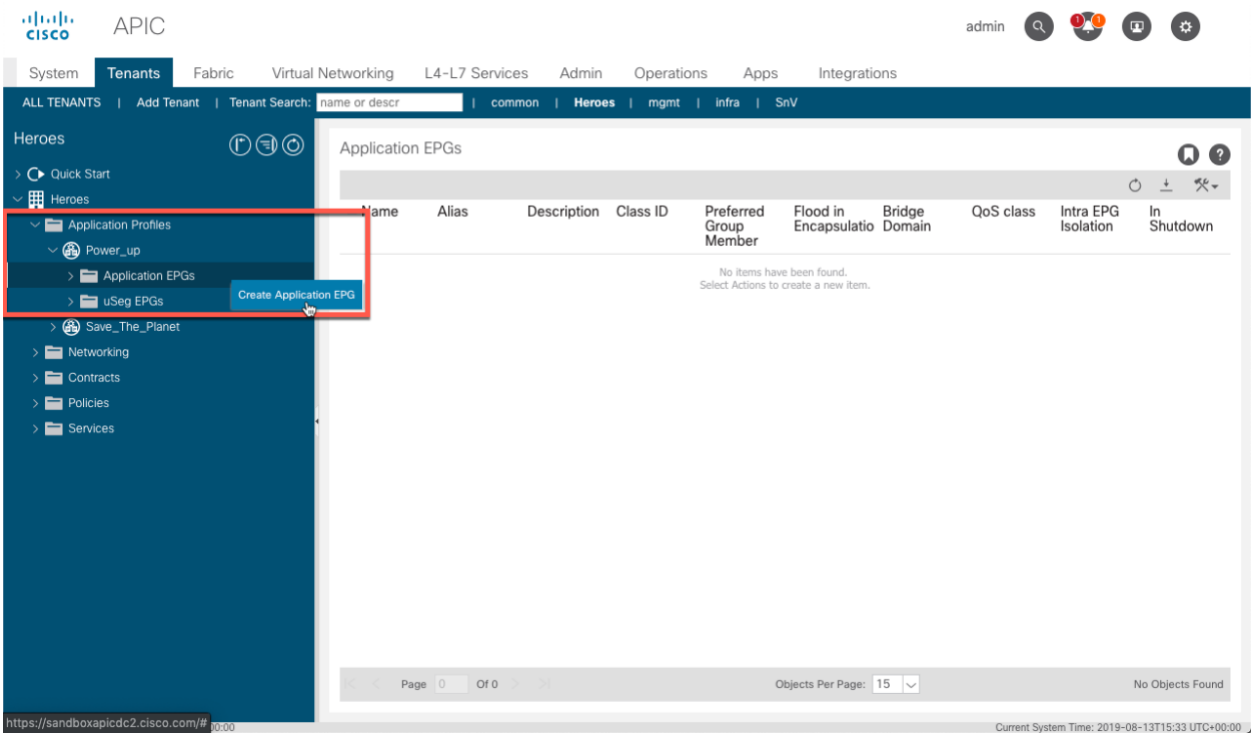
1. Name the Application "**Power_Up**".
2. Click **SUBMIT**.



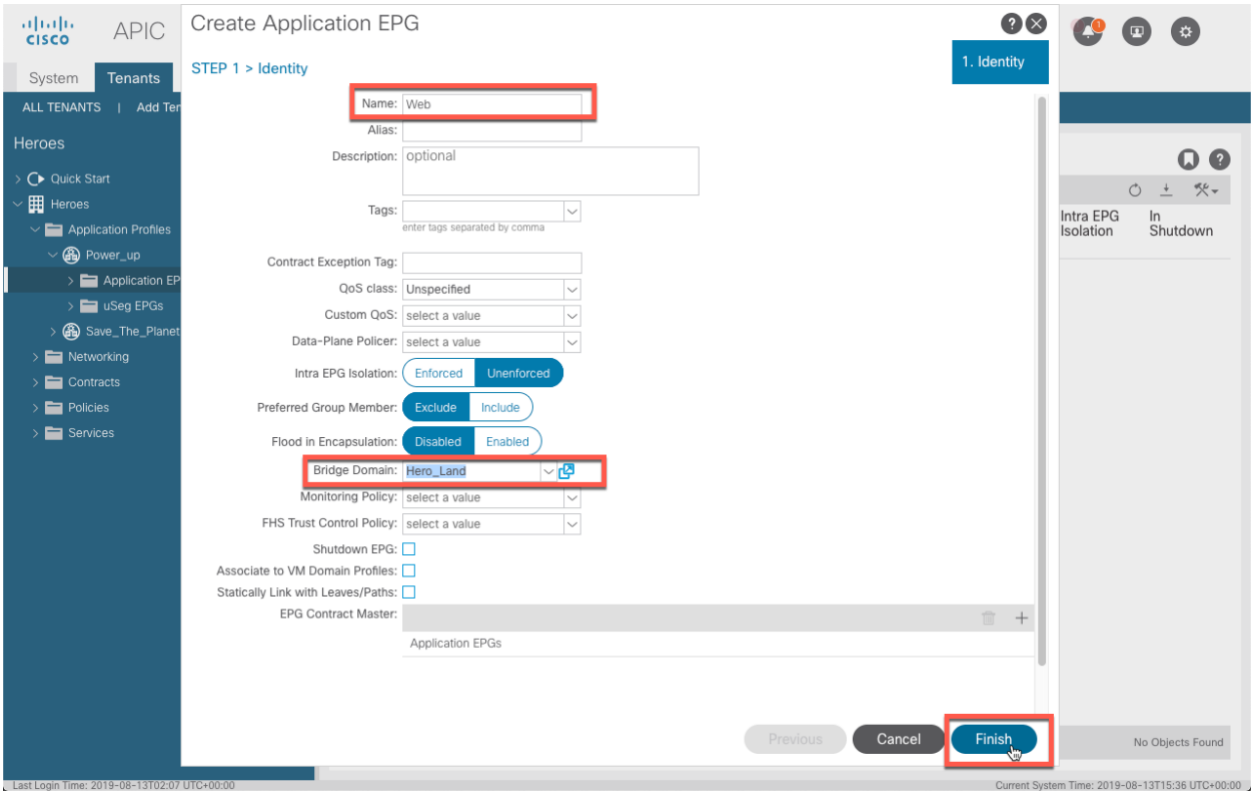
Create Web EPG

Now that the Application is created, configure the "Web" and "DB" EPGs that form this application.

1. Expand the "**Power_Up**" folder to get access to the EPGs.
2. Right-click **Application EPGs**.
3. Choose **Create Application EPG**.

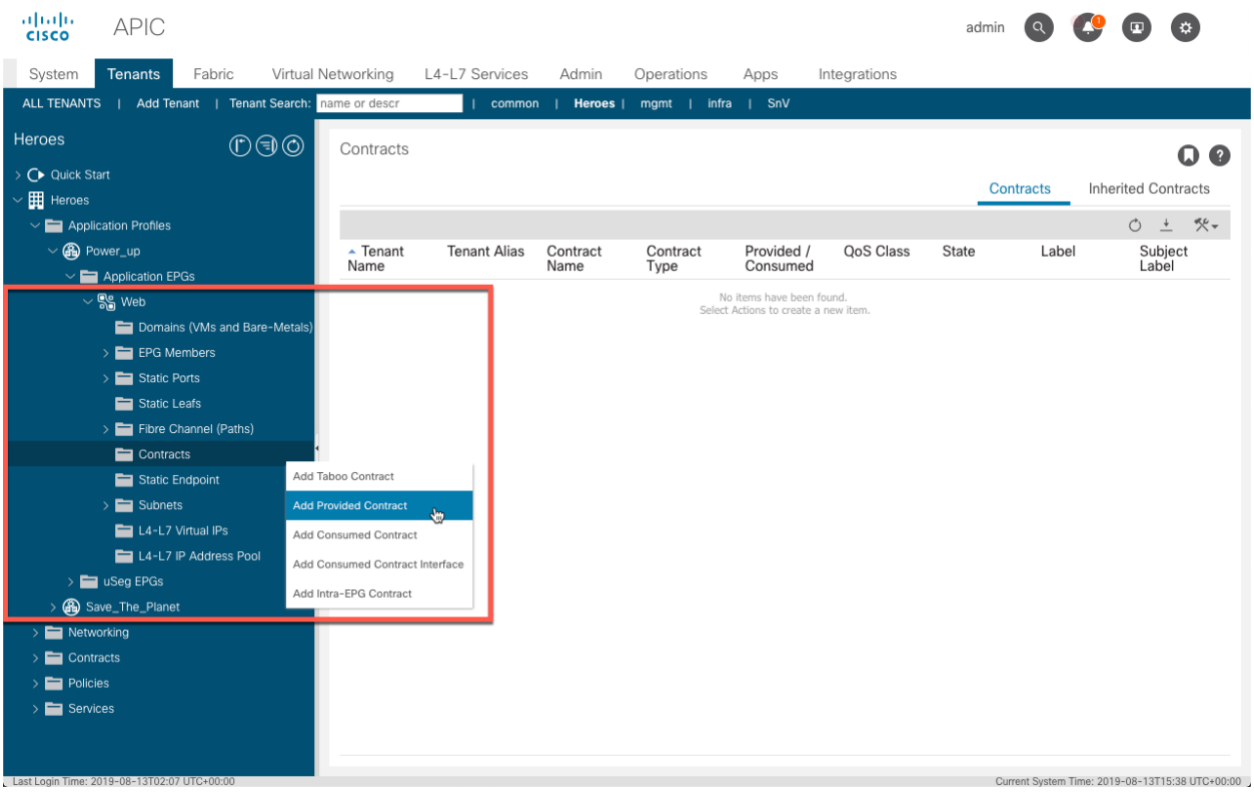


4. Name the EPG "**Web**".
5. Choose **Heroes/Hero_Land** from the Bridge Domain drop-down.
6. Click **FINISH**.



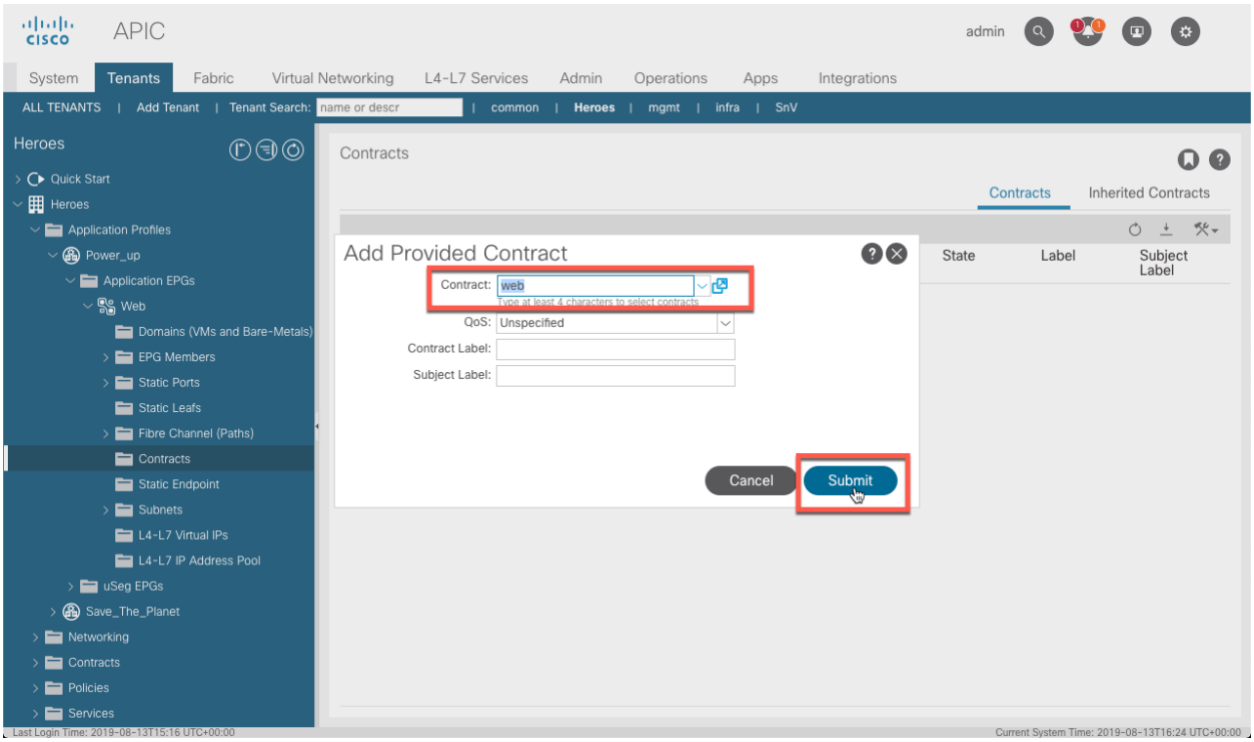
In order to have the Web tier provide a contract:

1. Expand the **Web** EPG.
2. Right-click **Contracts**.
3. Select **Add Provided Contract**.



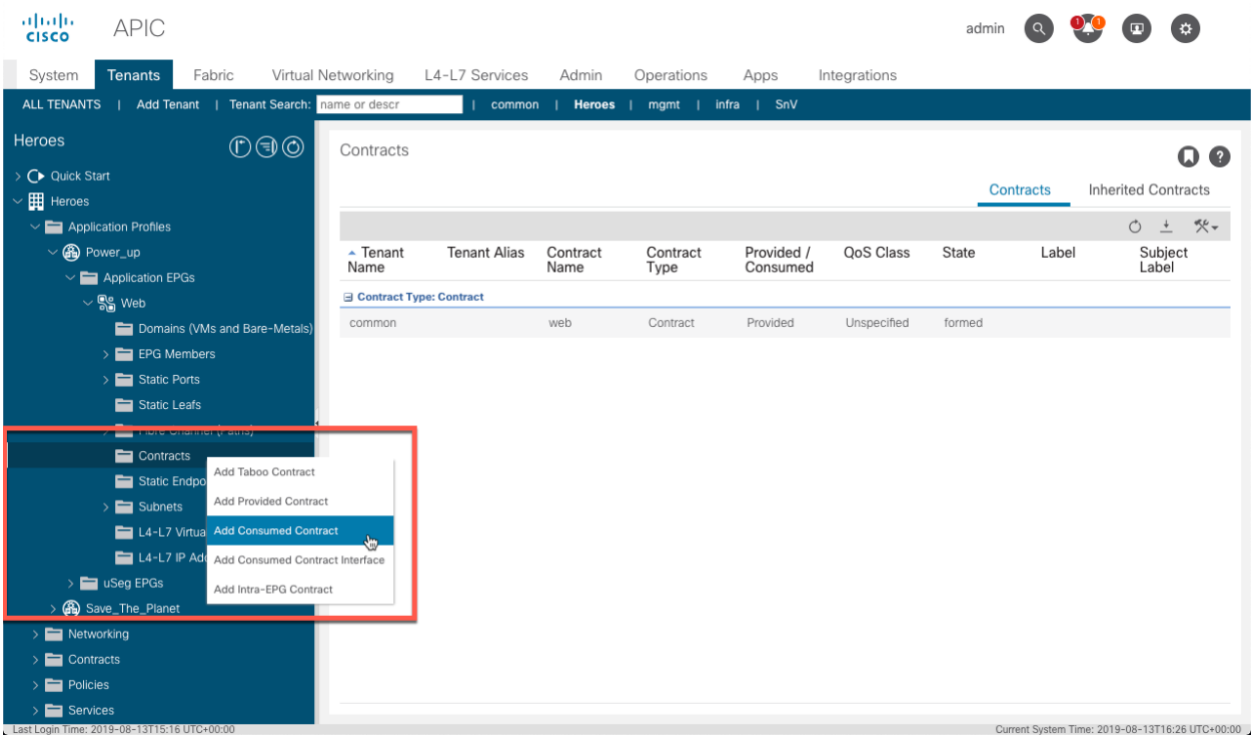
From the **Add Provided Contract** window:

1. Choose **web/common** from the **Contract** drop-down menu.
2. Click **SUBMIT**.



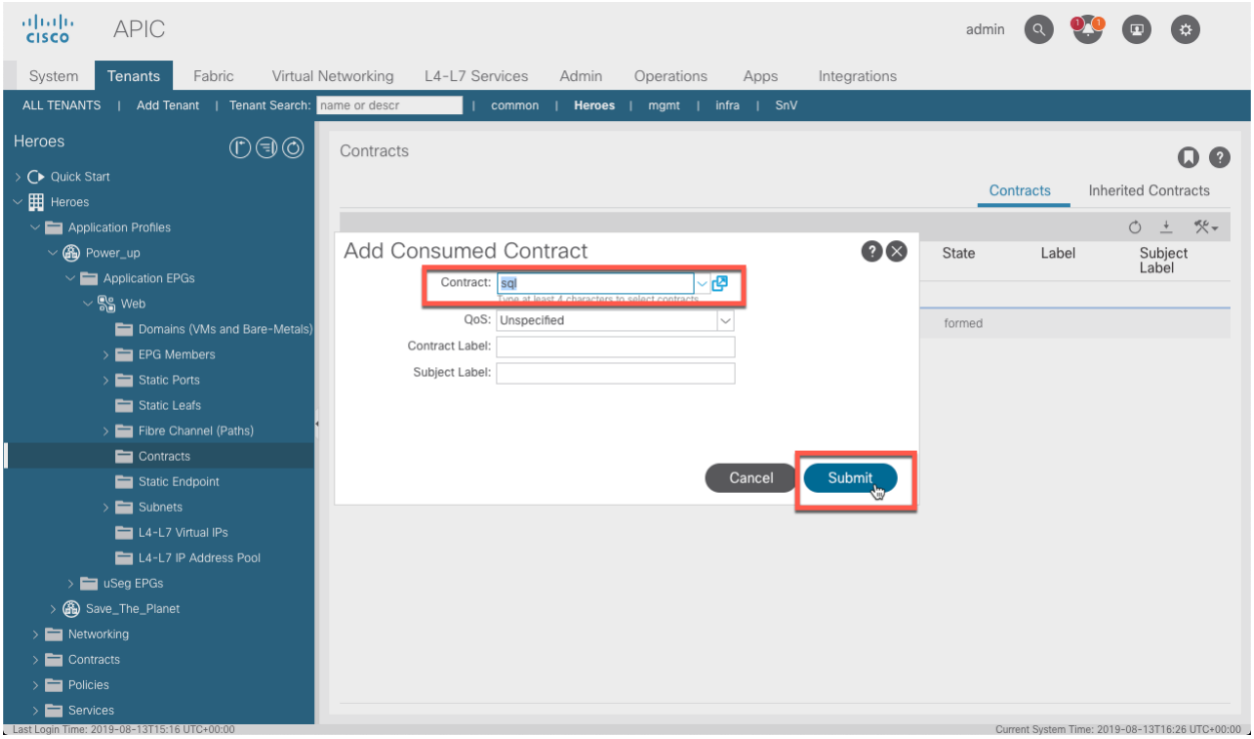
Consuming a contract follows the same steps

1. choose **Add Consumed Contract** this time.



1. Choose **common/sql** from the **Contract** drop-down menu.

2. Click **SUBMIT**.

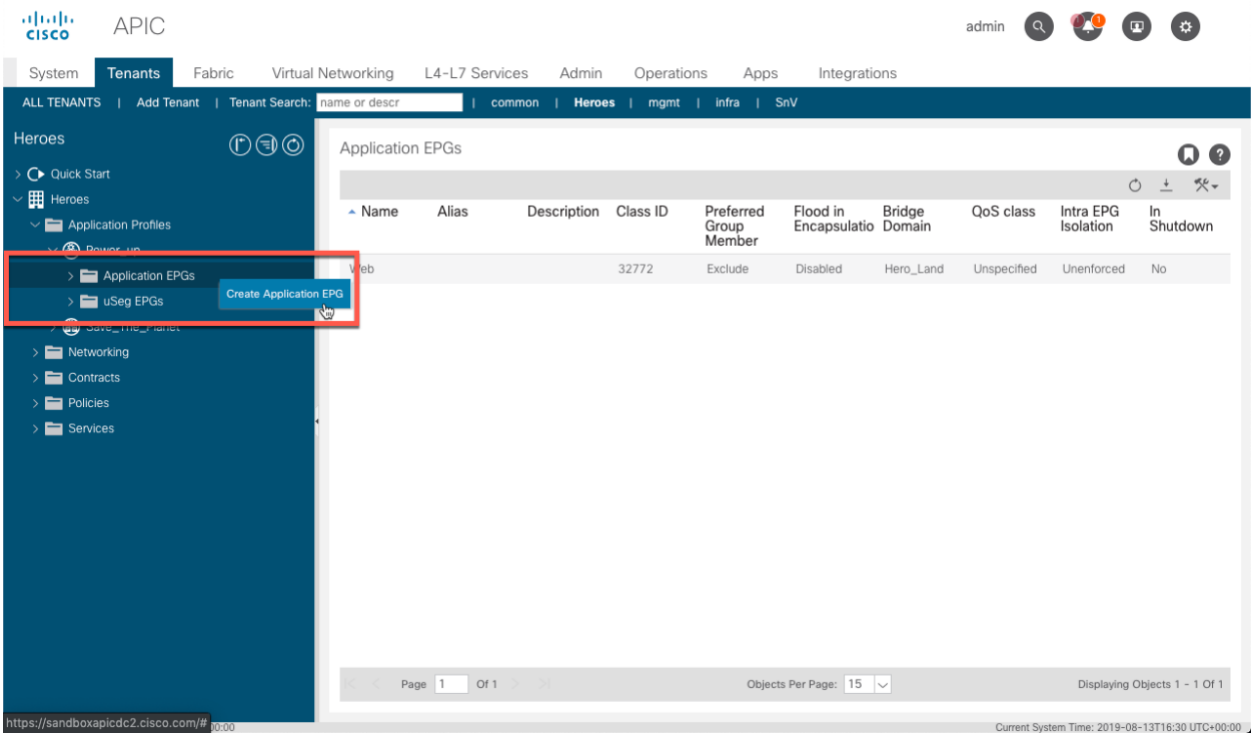


The Web EPG is created. Follow the same steps to create the "DB" EGP.

Create DB EPG

The following steps create the "DB" EGP:

1. Right-click **Application EPGs**.
2. Choose **Create Application EPG**.

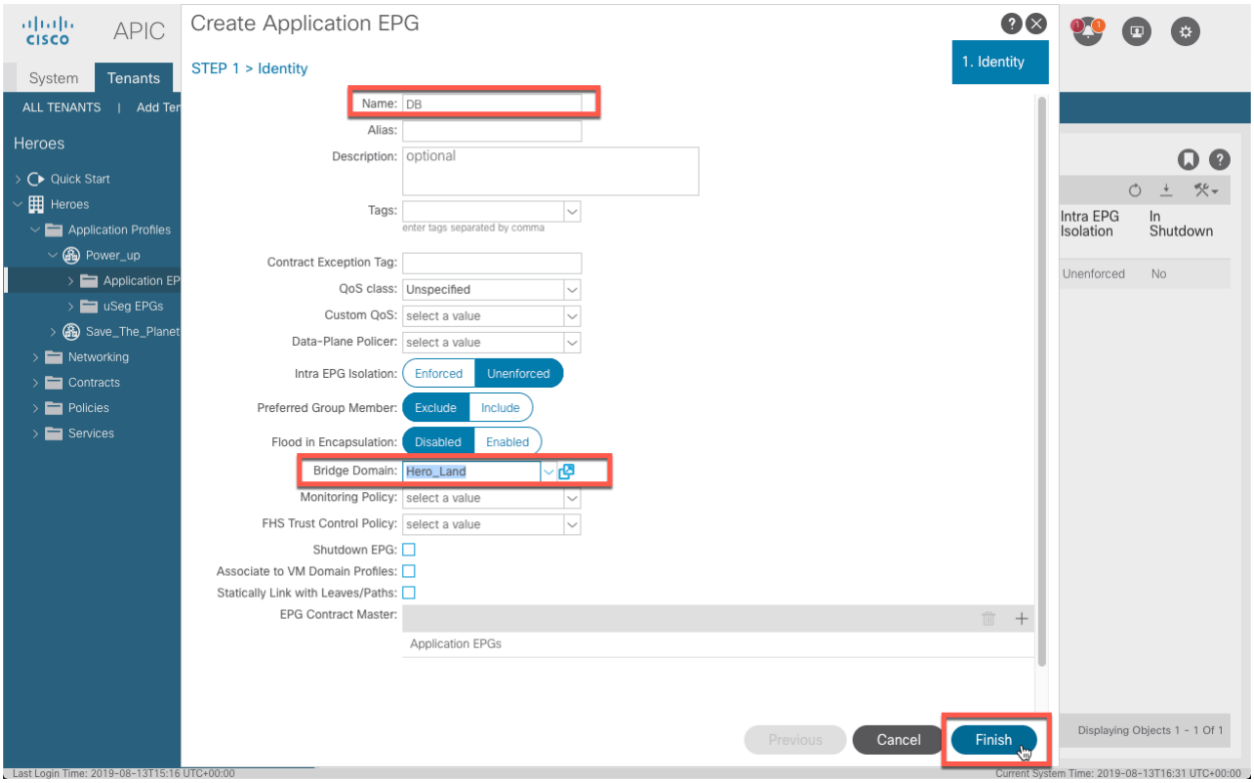


In the **Create Application EPG** window:

1. Name the EPG "**DB**".
2. Choose **Heroes/Hero_Land** from the **Bridge Domain** drop-down.
3. Click **FINISH**.

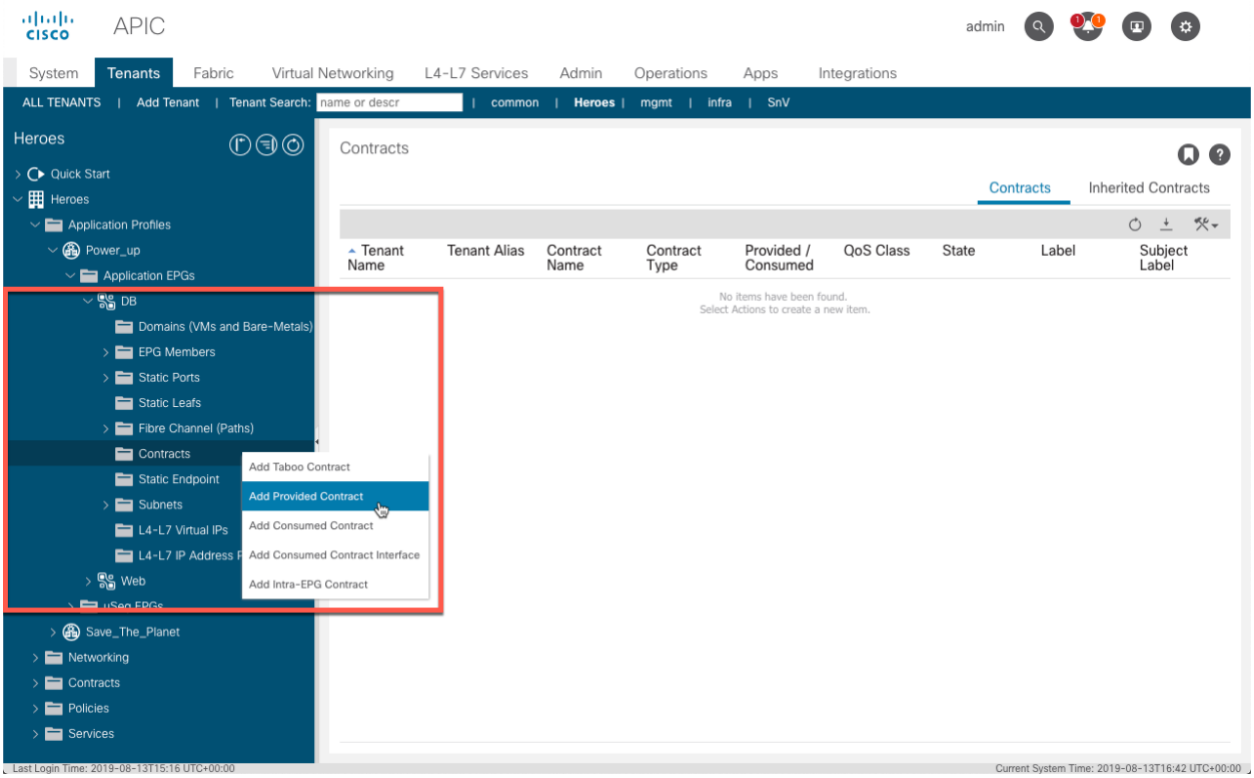
In the **Create Application EPG** window:

1. Name the EPG "**DB**".
2. Select **Heroes/Hero_Land** from the **Bridge Domain** drop-down.
3. Click **FINISH**.



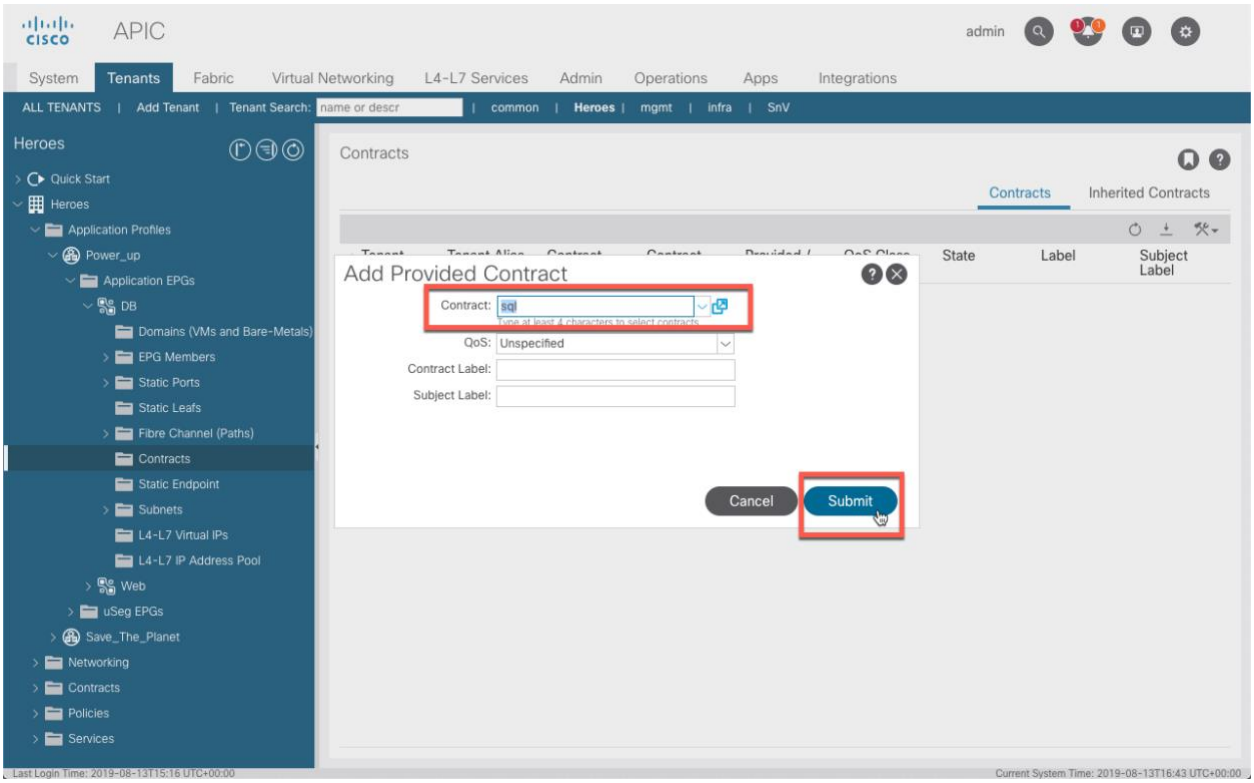
Have the DB tier provide the "SQL" contract:

1. Expand **EPG DB**.
2. Right-click **Contracts**.
3. Select **Add Provided Contract**.



In the **Add Provided Contract** window:

1. Choose **common/sql** from the **Contract** drop-down menu.
2. Click **SUBMIT**.



The DB tier does not consume any services. The application is now complete with the Web servers able to access the DBs for SQL services. Since existing contracts from the "common" Tenant were used, the necessary resources are also able to consume the Web services from the Web EPG.

It is extremely important to understand Tenant Networking and Tenant Policy before you start programming ACI.

Great Job!

Since the basics are covered, let's now move to looking at the different programmability options ACI has to offer.

Lab 02: APIC REST API Using Postman

Interacting with the APIC REST API Using Postman

This introductory lab exercise will get you started in interacting with the ACI REST API. POSTMAN, a graphical tool for API development and testing, will be used to allow you to focus on the outcome and not the process.

Objective

The objective of this lab is to show how to:

- Show how to authenticate against the APIC to obtain the APIC-Cookie
- Perform several calls against an ACI sandbox to retrieve data from the fabric controller

Prerequisites

- Option 1: Set up my own machine
  - Select the *Prerequisites* menu to the left of this frame. Ensure that you have all tools installed, including POSTMAN
- Option 2: I'm at a DevNet event and using a DevNet provisioned machine
  - The workstation you're using is either preconfigured with all necessary toolsets or the instructor will guide you through setup

Rest API Basics is recommended to understand REST API fundamentals.
Understanding ACI is highly recommended to understand the ACI Management Information Tree (MIT).

Step 1: Authenticating Against ACI

ACI was built with programmability in mind and designed to be configured and maintained through a central controller via a REST API. This API is how admins interact with the object-model allowing them to create, make changes, gather stats, and troubleshoot the ACI fabric.

The REST API uses HTTP and supports all four CRUD methods:

- (C)reate new objects
- (R)ead objects to view configuration and statistical data
- (U)pdate existing objects
- (D)elete objects that are no longer needed

POSTMAN can be used to perform Create, Read, Update and Delete methods using the various managed objects and classes contained within ACI

Before any interaction can occur via the API, an authentication cookie must be obtained. This cookie will allow us to access the API for subsequent calls.

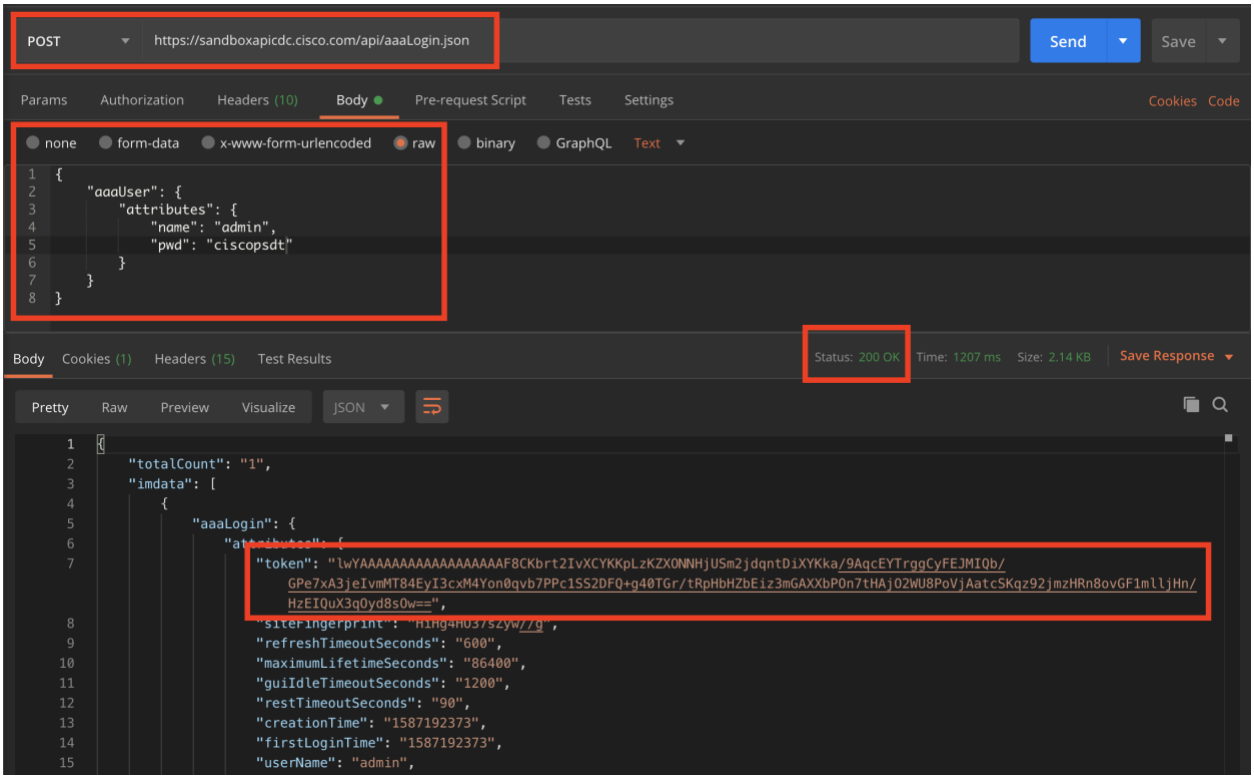
For these exercises, it is sufficient to know that login and configuration changes use HTTP POST and reading data uses HTTP GET.

To use POSTMAN to log in to the APIC

1. Set the method to POST
2. Enter the URL as `https://sandboxapicdc.cisco.com/api/aaaLogin.json`
3. Enter the JSON below into the Body as `raw`
4. Click SEND

```
{
   "aaaUser": {
      "attributes": {
         "name": "admin",
         "pwd": "!v3G@!4@Y"
      }
   }
}
```

Make sure 200 OK is returned.



The value seen in the Token field is the APIC-Cookie returned from the APIC. This token is required for all subsequent authentication to the controller; it will not accept a username/password combination. POSTMAN will automatically store this cookie so that we don't have to, however, if we don't use the cookie before it's timeout value, we'll need to reauthenticate before our API calls will work again.

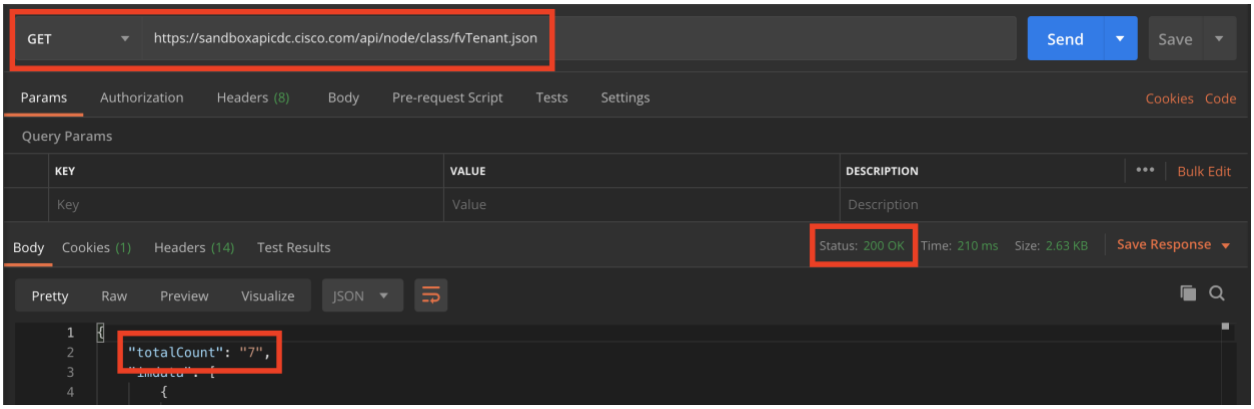
Step 2: Querying Tenant Information

Now that we've obtained our APIC-Cookie, we can start to do "interesting things" against the APIC. Lets start by looking at the tenants configured. Lets create a new call with the following parameters:

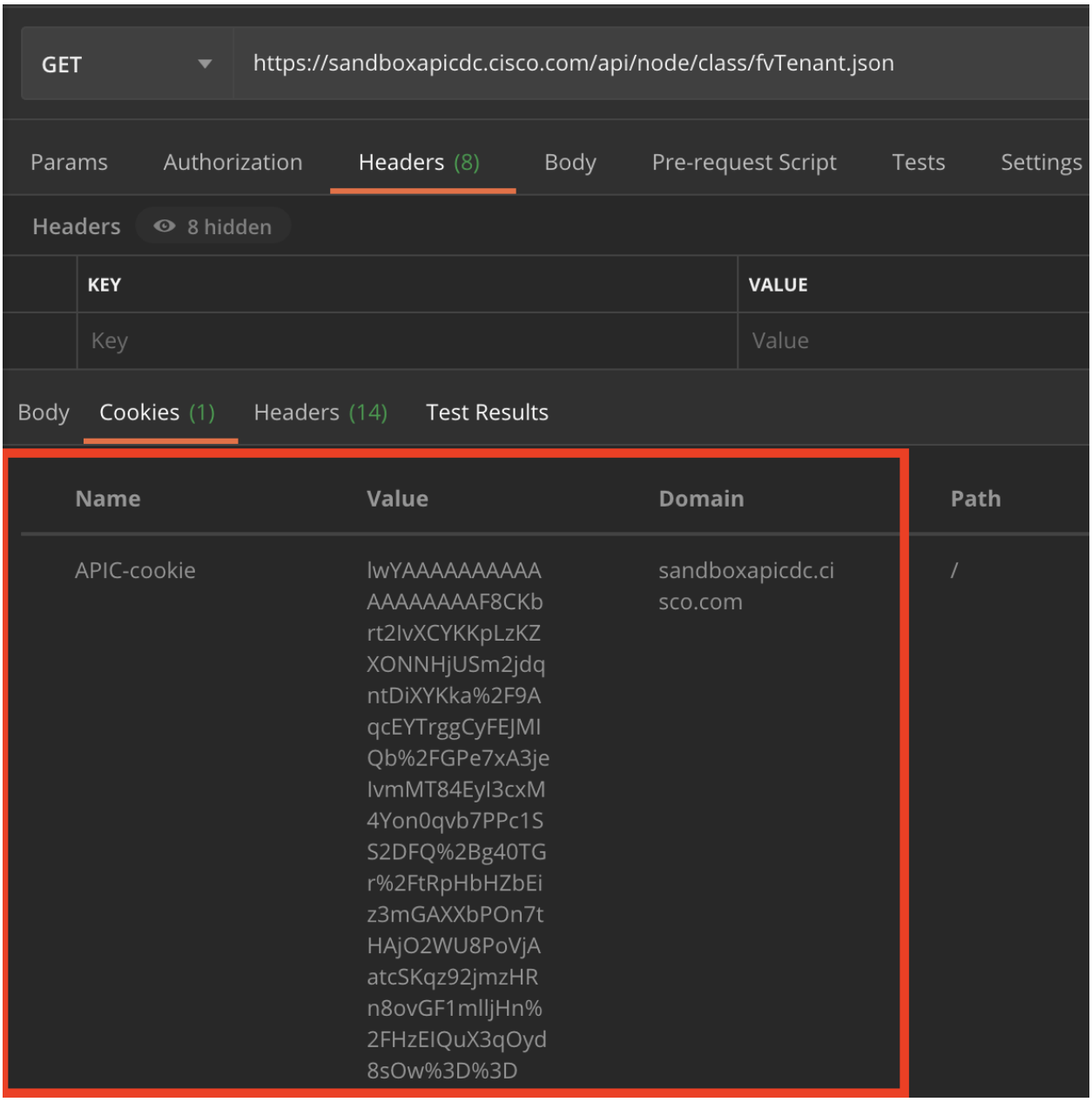
1. Set the method to GET
2. Enter the URL as https://sandboxapicdc.cisco.com/api/node/class/fvTenant.json
3. Click SEND

Make sure 200 OK is returned.

You will receive back a JSON-encoded list of all tenants configured within the ACI fabric, with a total count at the top. Something neat about the ACI API, if we were to switch the `.json` in the URL to `.xml`, we'd receive an XML-encoded response. Try it by switching the URL in POSTMAN!



But what about the cookie we obtained? After the response is obtained, if we click Cookies above the response window, we can see the stored cookie value for `APIC-Cookie`. This should match the returned value from the authentication exercise.



Step 3: Create a Tenant

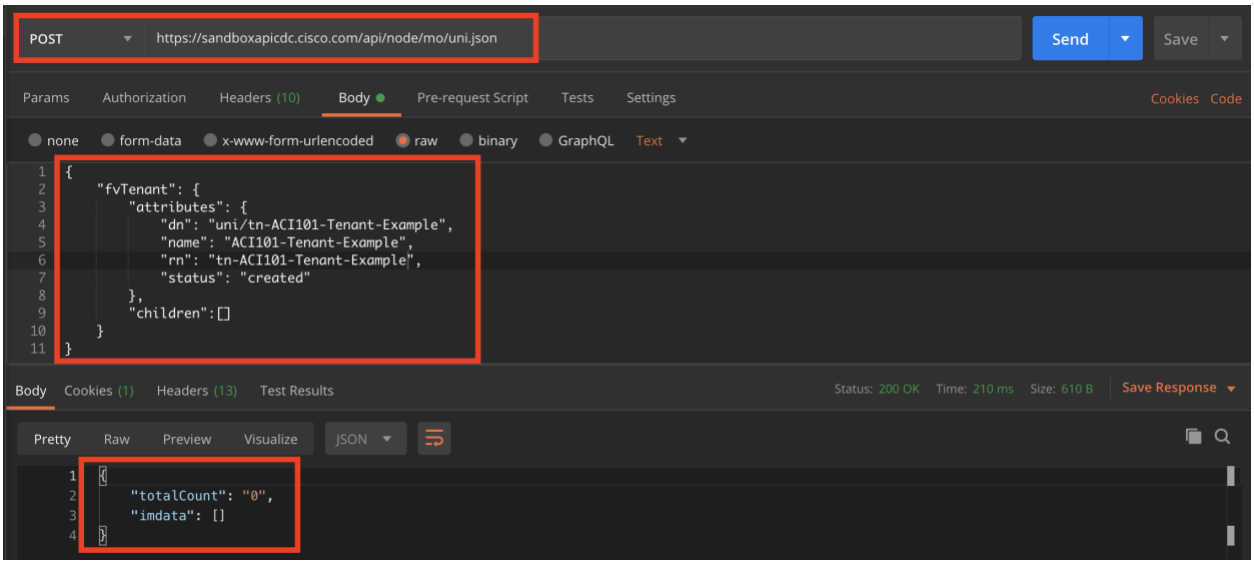
Now that we've gathered a list of current tenants, lets create one of our own.

1. Set the method to POST
2. Enter the URL as `https://sandboxapicdc.cisco.com/api/node/mo/uni.json`

3. Enter the following JSON body as `raw` (you may want to edit the tenant `dn`, `name`, and `rn` to ensure its unique)
4. Click SEND

```
{
   "fvTenant": {
      "attributes": {
         "dn": "uni/tn-ACI101-Tenant-Example",
         "name": "ACI101-Tenant-Example",
         "rn": "tn-ACI101-Tenant-Example",
         "status": "created"
      },
      "children":[]
   }
}
```

You'll receive a pretty empty response, but you've created a tenant!

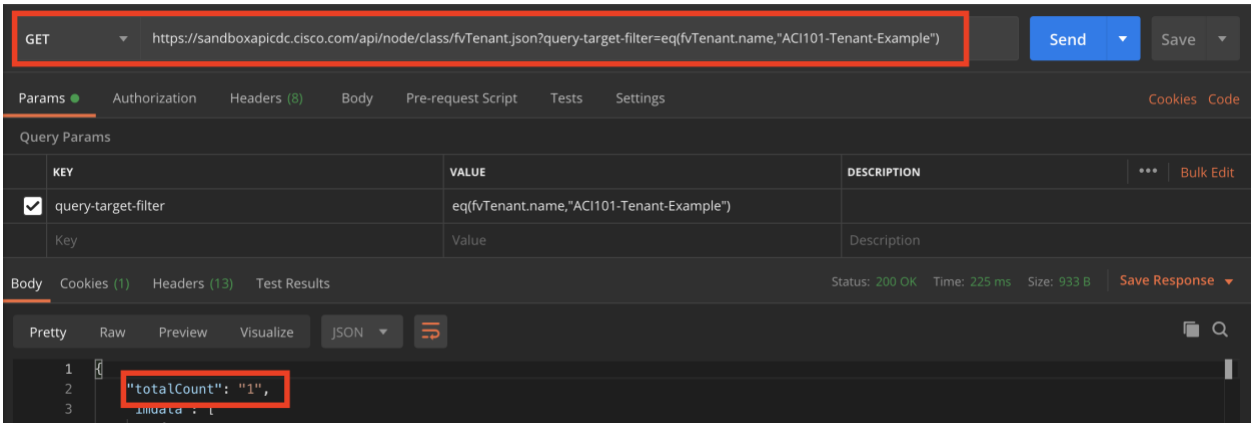


Step 4: Verifying Tenant Creation

Lets just make sure that everything worked and the tenant was created. We'll use a query-filter applied to the end of the ACI URL to ensure we find our newly created tenant.

1. Set the method to GET
2. Enter the URL as `https://sandboxapicdc.cisco.com/api/node/class/fvTenant.json?query-target-filter=eq(fvTenant.name,"ACI101-Tenant-Example")`
3. Click SEND

This will create a query-filter against your API call; rather than searching for all tenants as before, the results are filtered to only return the tenant name that is queried. If the call returns a count of 1, everything is successful.



Continue your journey by finishing this activity in the Learning Labs