Steps

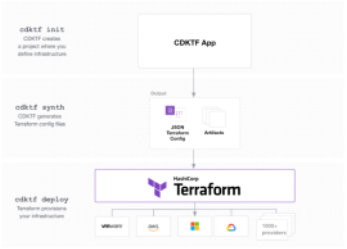
November 14, 2022 12:14 PM

Introduction

The goal of this demo is to provision a new tenant in ACI as well a VRF, BD, App profile and EPG using Terraform CDK (CDKTF) using a general programming language instead of HCL - in this case we will use Python.

This will be provisioned on the Cisco DevNet Sandbox ACI Simulator v5.2.

The workflow for CDKTF is as follow:



Prerequisites:

- Download and install Python 3
  - https://www.python.org/downloads/
  - Make sure python is added to your PATH environment variable.
    - You can test by checking if "python --help" is a valid command in your command prompt terminal:

```
PS C:\Users\alexp> python --help
usage: C:\Users\alexp\AppData\Local\Programs\Python\Python311\python.exe [option] ... [-c cmd | -m mod | file | -] [arg]
 ...
Options (and corresponding environment variables):
-b     : issue warnings about str(bytes_instance), str(bytearray_instance)
         and comparing bytes/bytearray with str. (-bb: issue errors)
-B     : don't write .pyc files on import; also PYTHONDONTWRITEBYTECODE=x
```

  - Install package "pipenv" to create virtual environments.
    - "pip install pipenv"
    - Note: If pip doesn't work, disconnect from VPN and try again without being connected on VPN.

- Download and install node.js
  - https://nodejs.org/en/
  - npm (node package manager) is bundled with nodejs installer.
  - Make sure node is added to your PATH environment variable.
    - You can test by checking if "node --help" is a valid command in your command prompt terminal:

```
PS C:\Users\alexp> node --help
Usage: node [options] [ script.js ] [arguments]
       node inspect [options] [ script.js | host:port ] [arguments]

Options:
  -                                 script read from stdin (default if no file name is
                                    provided, interactive mode if a tty)
  --                                indicate the end of node options
  --abort-on-uncaught-exception     aborting instead of exiting causes a core file to
```

  - Make sure NPM is also in your PATH.
    - You can test by checking if "npm" is a valid command in your command prompt terminal:

```
PS C:\Users\alexp> npm
npm <command>

Usage:

npm install        install all the dependencies in your project
npm install <foo>  add the <foo> dependency to your project
npm test           run this project's tests
npm run <foo>      run the script named <foo>
npm <command> -h   quick help on <command>
npm -l             display usage info for all commands
npm help <term>    search for help on <term> (in a browser)
```

- Download and install Terraform CLI
  - There is no installer - you only have to add the terraform.exe to your PATH in order to "install" Terraform.
    - https://developer.hashicorp.com/terraform/tutorials/aws-get-started/install-cli
  - Make sure terraform is added to your PATH environment variable.
    You can test by checking if "terraform" is a valid command in your command prompt terminal:

```
PS C:\Users\alexp> terraform
Usage: terraform [global options] <subcommand> [args]

The available commands for execution are listed below.
The primary workflow commands are given first, followed by
less common or more advanced commands.

Main commands:
  init          Prepare your working directory for other commands
  validate      Check whether the configuration is valid
  plan          Show changes required by the current configuration
  apply         Create or update infrastructure
  destroy       Destroy previously-created infrastructure
```

Procedures

Installing CDKTF

The installation of CDKTF is all through the node.js package manager (npm). Type the command below in a command prompt which will fetch the latest version of CDKTF from the NPM repository and install it automatically.

```
"npm install --global cdktf-cli@latest"
```

If you encounter any problems, make sure your network connection is ok (VPN may or may not block npm) and that all the prerequisites are met and all variable are added to PATH.

Creating a CDKTF Project

First, let's create a new folder name ACI_Terraform wherever you want the project to be located on your machine. In that folder, we're going to create a new Python virtual environment.

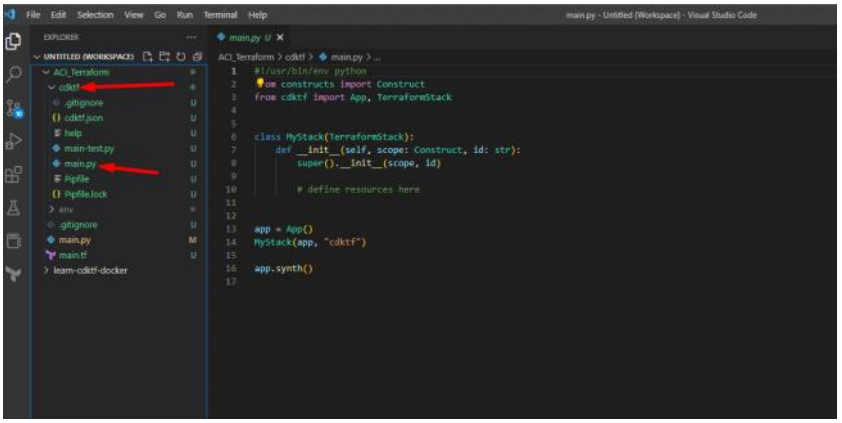
Virtual env named env and activate the virtual envrionment.

The next step will be to create the CDKTF project. This is done through "cdktf init", which will generate all the required configuration file. Since we can't cdktf init in an non -empty directory. Created a cdktk folder inside ACI_Terraform and cd into it.

Then, we need to run cdktf init --template=python --local in that directory, while making sure our Python virtual environment is still activated. The --template=python arguments tells CDKTF to prepare the main.py Python template file with the necessary class definition for the stack where we will define our resources.

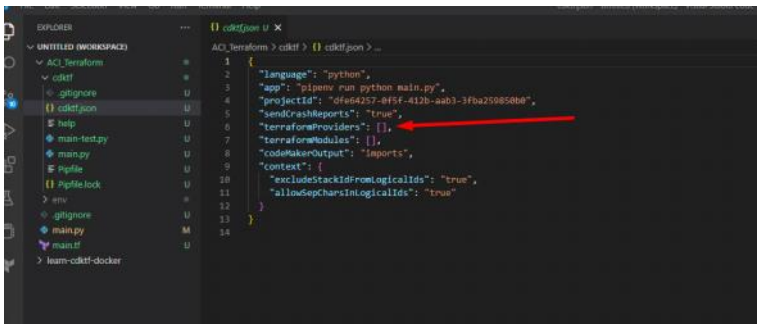
```
(env) PS C:\Users\alexp\Code\ACI_Terraform> cdktf init --template=python --local
ERROR: Cannot initialize a project in a non-empty directory
(env) PS C:\Users\alexp\Code\ACI_Terraform> cd .\cdktf\
(env) PS C:\Users\alexp\Code\ACI_Terraform\cdktf> cdktf init --template=python --local
Note: By supplying '--local' option you have chosen local storage mode for storing the state of your stack.
This means that your Terraform state file will be stored locally on disk in a file 'terraform.<STACK NAME>.tfstate' in the root of your project.
? Project Name ACI_cdktk
? Project Description Simple demo for ACI CDKTF
? Do you want to send crash reports to the CDKTF team? See https://www.terraform.io/cdktf/create-and-deploy/configuration-file#enable-crash-reporting-for-the-cli for more information (Y/n)
```

Notice how "cdktf init --template=python --local" has has generated multiple files in the new directory.

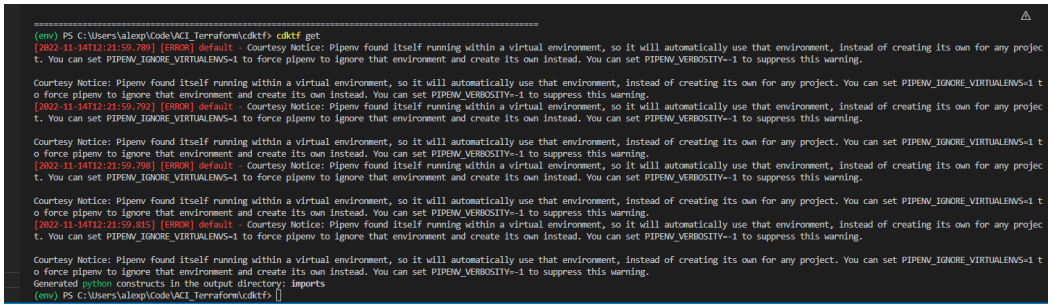


Importing a Terraform Provider in CDKTF

Now we need to tell CDKTF to use the Cisco ACI provider. In the last step, the CDKTF init generated cdktf.json for us, which contains certain property such as the programming language we're interacting with as well as the list of Terraform providers we are planning to use. Those are the exact same provider as with "regular" Terraform. We need to add our provider(s) to cdktf.json - inside the terraformProviders list, just as follow:

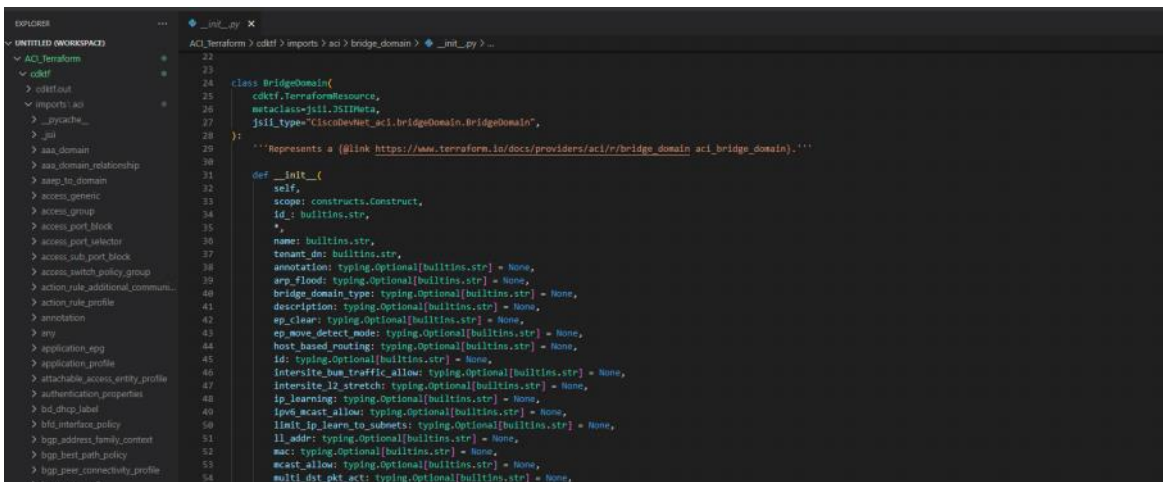


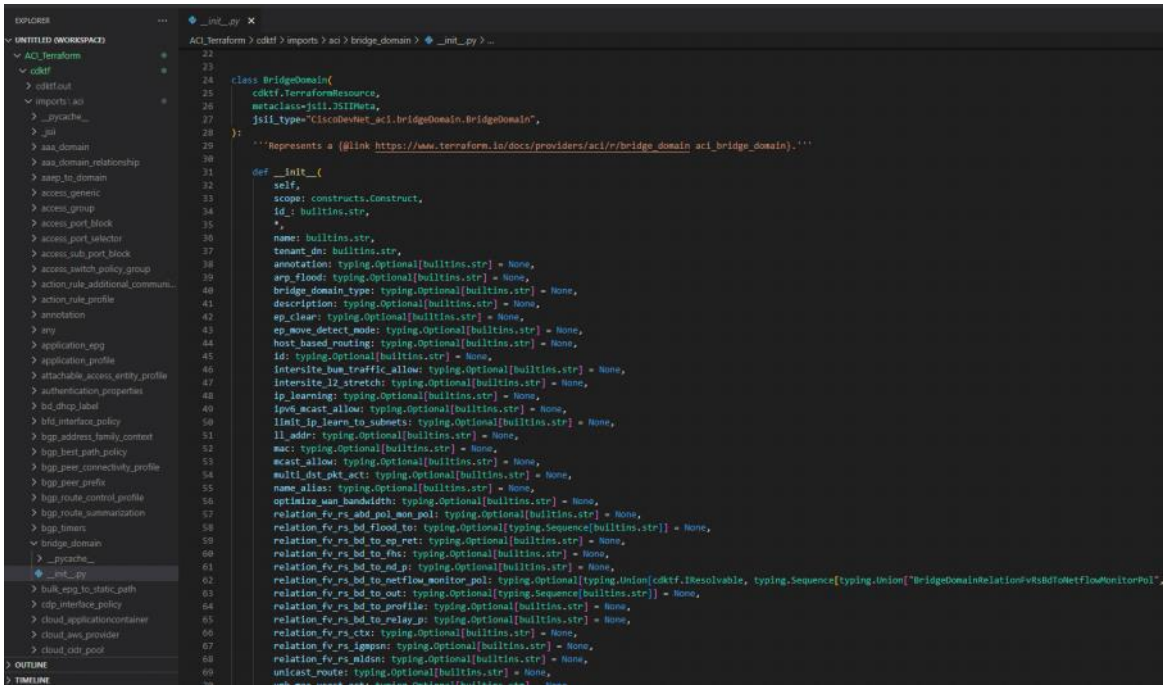
After which we need to do a cdktf get in order to generate the Python class automatically for all resources defined in the CiscoDevNet/aci provider.



Now if you take a look, there's a new folder called imports/aci in the current directory. You will find bunch of Python modules which include all the Terraform resources that are defined in the Terraform Cisco ACI provider.

For example, below is the modules aci_bridge_domain, which has the class definition for the BridgeDomain resource:

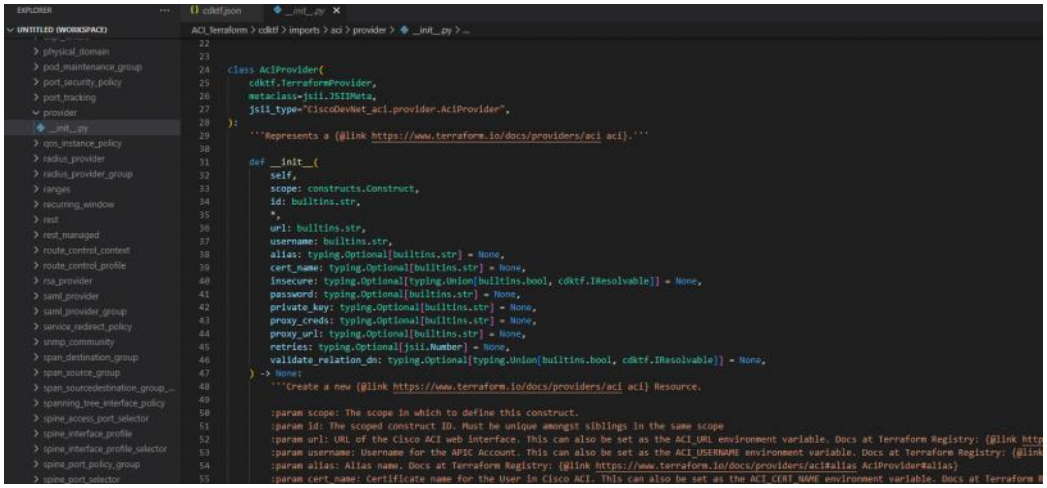




Provider Declaration

Before creating any ACI objects such as EPGs or BDs, we need to configure how Terraform CDK will connect to the ACI. This works very similar to Terraform CLI with the Terraform provider declaration in HCL, but now we'll use Python instead.

The class definition can be found here for the provider can also be found in imports/aci.



We need to first import the ACI Provider module into the main.py file that cdktf init has created for us. To do that, add :

from imports.aci import provider

At the top of the main.py file.

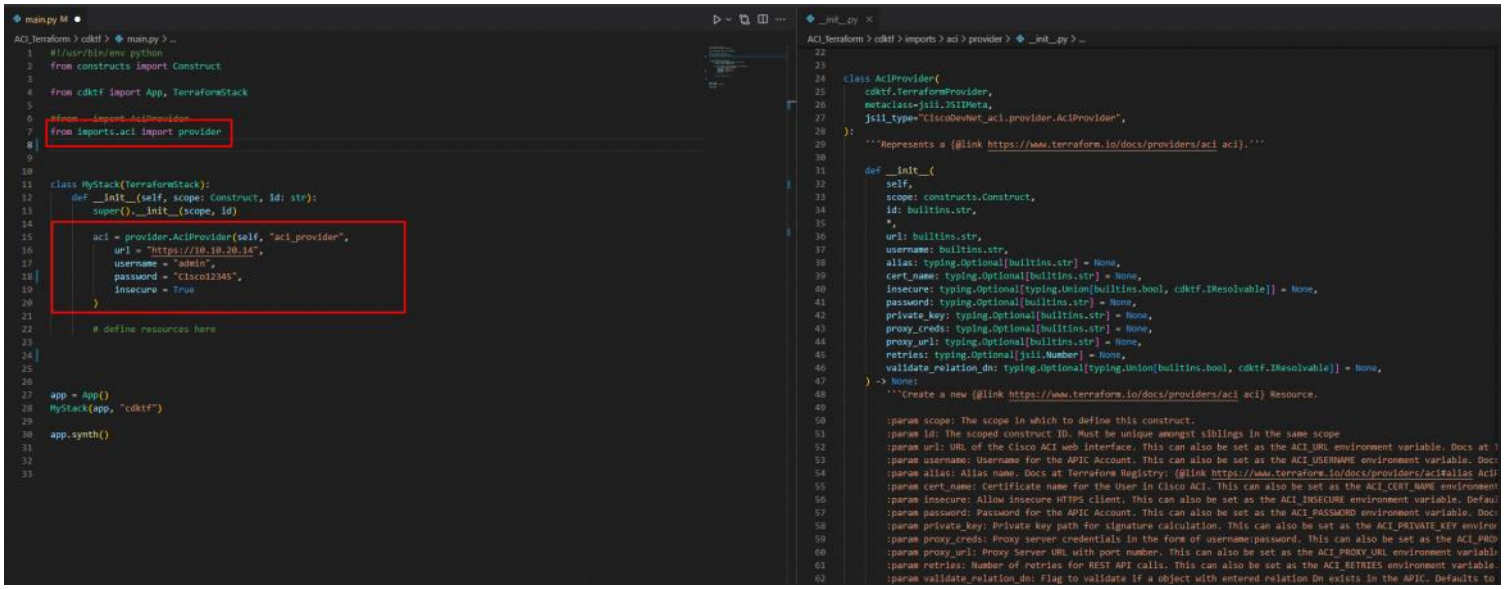
Then, within the MyStack class definition, after the constructer method, we have to create a new object of type AciProvider and initialize the object with the attributes we need. In this case, Terraform need at least the following parameters to be able to connect to the APIC:

- Url : URL of the APIC
- Username : username to log in the APIC
- Password : password to log in the APIC
- Insecure : Use HTTP or HTTPs

This is the code I used to create an object name "aci" of type AciProvider. Notice, after passing argument self, I am passing "aci_provider". This is an arbitrary name, it can be anything you want - it's just an ID, similar to what you'd do in HCL.

```
aci = provider.AciProvider(self, "aci_provider",
            url = "https://10.10.20.14",
            username = "admin",
            password = "C1sco12345",
            insecure = True
        )
```

Note: it's always a good idea to look into all the available options/attributes in the constructor method within the AciProvider or any resources class in the respective python module for more info on what is configurable.



Configuring ACI objects

Now that we have our provider declaration completed, we can build our ACI objects.

Tenant

Here I defined a new tenant based on the property/attributes of the Tenant class which can be found in the tenant module in ./imports/aci/tenant

```
#!/usr/bin/env python
from constructs import Construct
from cdktf import App, TerraformStack
#from . import AciProvider
from imports.aci import provider
from imports.aci import tenant

class MyStack(TerraformStack):
    def __init__(self, scope: Construct, id: str):
        super().__init__(scope, id)

        aci = provider.AciProvider(self, "aci_provider",
            url = "https://10.10.20.14",
            username = "admin",
            password = "C1sco12345",
            insecure = True
        )
        # define resources here
        terraform_cdktf_tenant =
tenant.Tenant(self,"terraform_cdktf_tenant",
            name = "terraform_cdktf_tenant",
            description = "This tenant was built with CDKTF using Pyhton"
            )

app = App()
MyStack(app, "cdktf")
app.synth()
```

As you can see at the end of the main.py, there is a function call for app.synth(), which is the same as the cdktf synth command. Hence, running the main.py file will automatically call cdktf synth.

Cdktf synth takes care of generating Terraform JSON compliant configuration file based on what you have configured in your Python script.

I then ran the main.py within the current virtual environment. then I did, from the CLI, the cdktf deploy and approved the change.

```
(env) PS C:\Users\alexp\Code\ACI_Terraform\cdktf> & c:/Users/alexp/Code/ACI_Terraform/env/Scripts/python.exe c:/Users/alexp/Code/ACI_Terraform/cdktf/main.py
(env) PS C:\Users\alexp\Code\ACI_Terraform\cdktf> cdktf deploy

!  Synthesizing
[2022-11-15T08:31:37.090] [ERROR] default - Courtesy Notice: Pipenv found itself running within a virtual environment, so it will automatically use that environment, instead of creating its own for any projec
t. You can set PIPENV_IGNORE_VIRTUALENVS=1 to force pipenv to ignore that environment and create its own instead. You can set PIPENV_VERBOSITY=-1 to suppress this warning.

Courtesy Notice: Pipenv found itself running within a virtual environment, so it will automatically use that environment, instead of creating its own for any project. You can set PIPENV_IGNORE_VIRTUALENVS=1 t
o force pipenv to ignore that environment and create its own instead. You can set PIPENV_VERBOSITY=-1 to suppress this warning.
[2022-11-15T08:31:37.098] [ERROR] default - Courtesy Notice: Pipenv found itself running within a virtual environment, so it will automatically use that environment, instead of creating its own for any projec
t. You can set PIPENV_IGNORE_VIRTUALENVS=1 to force pipenv to ignore that environment and create its own instead. You can set PIPENV_VERBOSITY=-1 to suppress this warning.

Courtesy Notice: Pipenv found itself running within a virtual environment, so it will automatically use that environment, instead of creating its own for any project. You can set PIPENV_IGNORE_VIRTUALENVS=1 t
o force pipenv to ignore that environment and create its own instead. You can set PIPENV_VERBOSITY=-1 to suppress this warning.
[2022-11-15T08:31:37.110] [ERROR] default - Courtesy Notice: Pipenv found itself running within a virtual environment, so it will automatically use that environment, instead of creating its own for any projec
t. You can set PIPENV_IGNORE_VIRTUALENVS=1 to force pipenv to ignore that environment and create its own instead. You can set PIPENV_VERBOSITY=-1 to suppress this warning.
      Saved the plan to: plan

      To perform exactly these actions, run the following command to apply:
          terraform apply "plan"

Please review the diff output above for cdktf
> Approve  Applies the changes outlined in the plan.
  Dismiss
  Stop
```

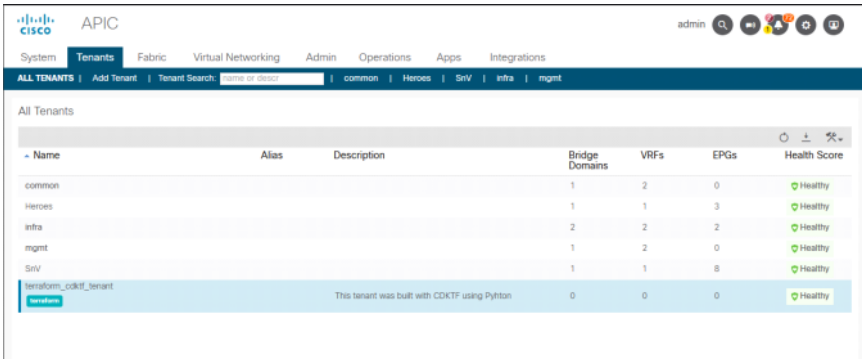
Then CDKTF let me know it is complete.

```
          terraform apply "plan"
cdktf  aci_tenant.terraform_cdktf_tenant (terraform_cdktf_tenant): Creating...
cdktf  aci_tenant.terraform_cdktf_tenant (terraform_cdktf_tenant): Creation complete after 1s [id=uni/tn-terraform_cdktf_tenant]
cdktf
      Apply complete! Resources: 1 added, 0 changed, 0 destroyed.


No outputs found.
(env) PS C:\Users\alexp\Code\ACI_Terraform\cdktf>
```

Note:

Within the Pyhton code we can only do the cdktf synth at the moment. We cannot call cdktf deploy from the Python code directly, unless I invoke a system call.
https://github.com/hashicorp/terraform-cdk/issues/1624
We'd have to call the system to call cdktf if we wanted to launch it from the python code

Checking on the APIC in Tenants, we can see that our new tenant tarraform_cdktf_tenant is now listed and has the terraform annotation.



VRF

Adding a VRF is very similar to adding a tenant. However, since the VRF is a child of a Tenant in ACI MO Tree, we need to specify the dn of the Tenant within the VRF declaration.

This can easily be achieved by leveraging the ID attributes available in any resource class in Terraform. In our case, we want the DN of the tenant we just created. To do that, we call the terraform_cdktf_tenant object and we take it's id attribute with the terraform_cdktf_tenant.id statement.

The steps to add a new VRF in our tenant is as follow:

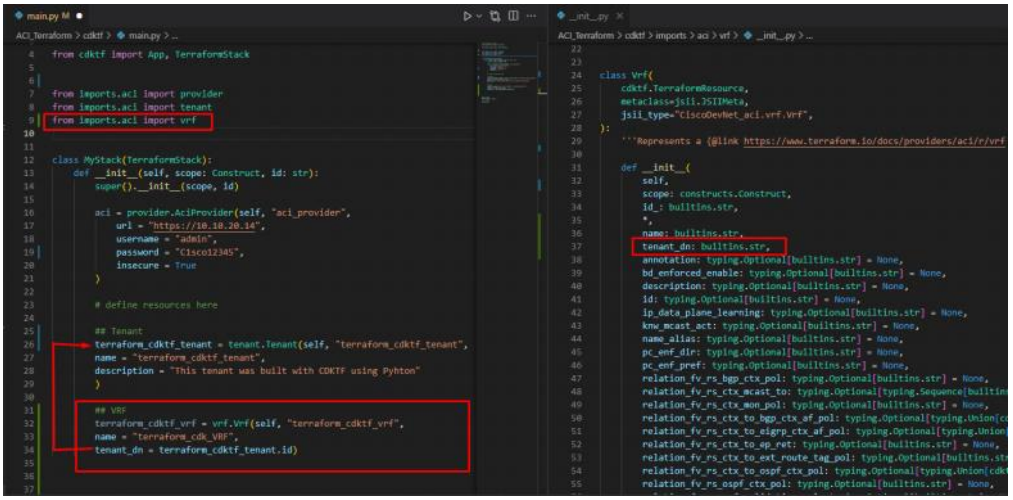
First, import the vrf module.

```
from imports.aci import vrf
```

With the rest of the import statements at the top of the main.py file.

Then create a new object of type VRF

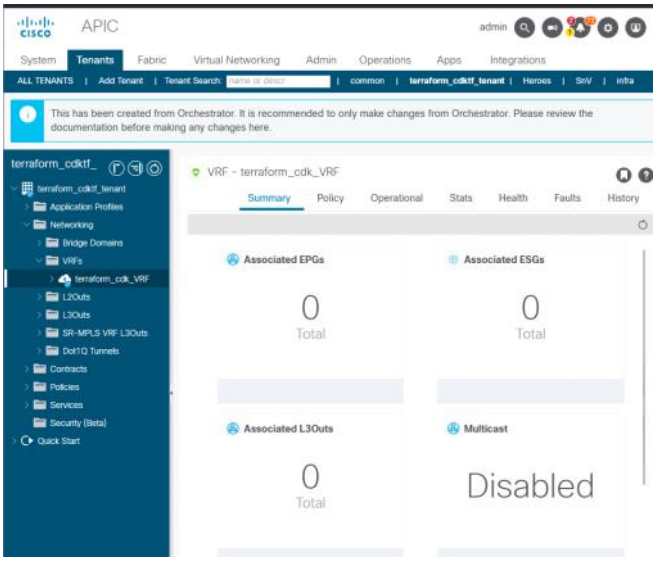
```
        ## VRF
        terraform_cdktf_vrf = vrf.Vrf(self, "terraform_cdktf_vrf",
        name = "terraform_cdk_VRF",
        tenant_dn = terraform_cdktf_tenant.id)
```



We run the python script again to synthetize, and then we run cdktf deploy.

```
cdktf  Initializing provider plugins...
       - Reusing previous version of ciscodevnet/aci from the dependency lock file
cdktf  - Using previously-installed ciscodevnet/aci v2.5.2
cdktf  Terraform has been successfully initialized!

       You may now begin working with Terraform. Try running "terraform plan" to see
       any changes that are required for your infrastructure. All Terraform commands
       should now work.

       If you ever set or change modules or backend configuration for Terraform,
       rerun this command to reinitialize your working directory. If you forget, other
       commands will detect it and remind you to do so if necessary.
            + knw_mcast_act                       = (known after apply)
            + name                                = "terraform_cdk_VRF"
            + name_alias                          = (known after apply)
            + pc_enf_dir                          = (known after apply)
            + pc_enf_pref                         = (known after apply)
            + relation_fv_rs_bgp_ctx_pol          = (known after apply)
            + relation_fv_rs_ctx_to_ep_ret        = (known after apply)
            + relation_fv_rs_ctx_to_ext_route_tag_pol = (known after apply)
            + relation_fv_rs_ospf_ctx_pol         = (known after apply)
            + relation_fv_rs_vrf_validation_pol   = (known after apply)
            + tenant_dn                           = "uni/tn-terraform_cdktf_tenant"
          }

       Plan: 1 to add, 0 to change, 0 to destroy.

       ------------------------------------------------------------------------

       Saved the plan to: plan

       To perform exactly these actions, run the following command to apply:
           terraform apply "plan"

Please review the diff output above for cdktf
> Approve  Applies the changes outlined in the plan.
  Dismiss
  Stop
```

There we have it - now looking at the APIC GUI, we can find our new VRF in our terraform_cdktf_tenant.



App, BD, EPG

Here's the code to create an app profile, an EPG and a BD.

```
#!/usr/bin/env python
from constructs import Construct
from cdktf import App, TerraformStack

from imports.aci import provider
from imports.aci import tenant
from imports.aci import vrf
from imports.aci import bridge_domain
from imports.aci import subnet
from imports.aci import application_profile
from imports.aci import application_epg

class MyStack(TerraformStack):
    def __init__(self, scope: Construct, id: str):
        super().__init__(scope, id)

        aci = provider.AciProvider(self, "aci_provider",
            url = "https://10.10.20.14",
            username = "admin",
            password = "C1sco12345",
            insecure = True
        )
        # define resources here
        ## Tenant
        terraform_cdktf_tenant = tenant.Tenant(self, "terraform_cdktf_tenant",
        name = "terraform_cdktf_tenant",
        description = "This tenant was built with CDKTF using Pyhton"
        )
        ## VRF
        terraform_cdktf_vrf = vrf.Vrf(self, "terraform_cdktf_vrf",
        name = "terraform_cdk_VRF",
        tenant_dn = terraform_cdktf_tenant.id)
        ## BD
        terraform_cdktf_bd_1 = bridge_domain.BridgeDomain(self,
"terraform_cdktf_bd_1",
        name = "terraform_cdktf_bd_1",
        tenant_dn = terraform_cdktf_tenant.id,
        relation_fv_rs_ctx = terraform_cdktf_vrf.id)
        ## Subnet
        terraform_cdktf_subnet_1 = subnet.Subnet(self,
"terraform_cdktf_subnet_1",
        parent_dn = terraform_cdktf_bd_1.id,
        ip = "10.0.0.1/24",
        scope = [ "public" ]
        )
        ## Application Profile
        terraform_cdktf_app = application_profile.ApplicationProfile(self,
"terraform_cdktf_app",
        name = "terraform_cdktf_app",
        tenant_dn = terraform_cdktf_tenant.id)

        ## EPG
        terraform_cdktf_epg_1 = application_epg.ApplicationEpg(self,
"terraform_cdktf_epg_1",
        name = "terraform_cdktf_epg_1",
        application_profile_dn= terraform_cdktf_app.id,
        relation_fv_rs_bd = terraform_cdktf_bd_1.id)

app = App()
MyStack(app, "cdktf")
app.synth()
```

After synth and deploy, all the objects are created in ACI.