Working with Terraform

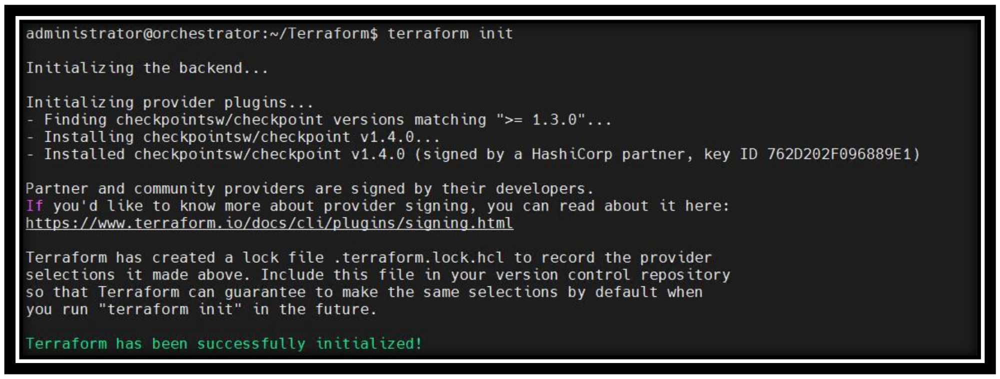
Task – 1: Simple Terraform Example

1. Change directory on the Ubuntu Orchestrator to the /home/administrator/Terraform directory.

2. Review the configurations files in the terraform directory.

3. Run the following command to initialize Terraform and the Check Point POC provider for Terraform. This will download the latest Check Point Terraform provider from Hashicorp.

```
terraform init
```



```
administrator@orchestrator:~/Terraform$ terraform init

Initializing the backend...

Initializing provider plugins...
- Finding checkpointsw/checkpoint versions matching ">= 1.3.0"...
- Installing checkpointsw/checkpoint v1.4.0...
- Installed checkpointsw/checkpoint v1.4.0 (signed by a HashiCorp partner, key ID 762D202F096889E1)

Partner and community providers are signed by their developers.
If you'd like to know more about provider signing, you can read about it here:
https://www.terraform.io/docs/cli/plugins/signing.html

Terraform has created a lock file .terraform.lock.hcl to record the provider
selections it made above. Include this file in your version control repository
so that Terraform can guarantee to make the same selections by default when
you run "terraform init" in the future.

Terraform has been successfully initialized!
```

4. Next, we will run a plan to see what would actually be done to build the objects that are described in the Simple Example directory.

```
terraform plan
```

5. How many objects does terraform need to add, how many need deleted?

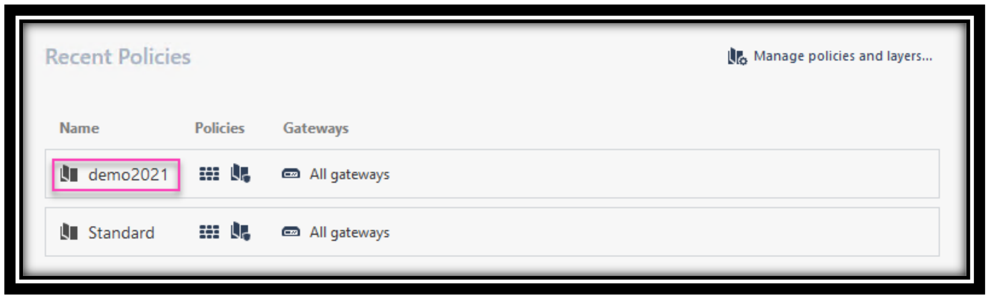
6. Now we can apply our configuration. Run the following

```
terraform apply
```

7. Once the apply is complete run the follow to publish the changes with the python script

```
python3 publish.py
```

8. Verify that the objects are in the .tf files were created in the R81 SmartConsole

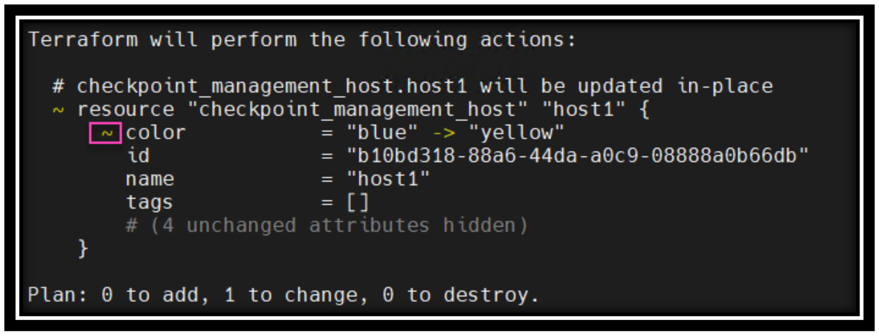


Name	Policies	Gateways
demo2021		All gateways
Standard		All gateways

Task – 2: Update the configuration

Now that we have built some elements of our infrastructure, we can see how easy it is to apply new changes to the configuration by simply changing the .tf files.

1. Change the color of a host in the hosts.tf file and run the terraform plan and terraform apply. What is going to be changed?

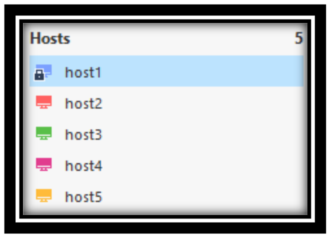


```
Terraform will perform the following actions:

  # checkpoint_management_host.host1 will be updated in-place
  ~ resource "checkpoint_management_host" "host1" {
      ~ color             = "blue" -> "yellow"
        id                = "b10bd318-88a6-44da-a0c9-08888a0b66db"
        name              = "host1"
        tags              = []
        # (4 unchanged attributes hidden)
    }

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

2. Check to see if that host is now locked in SmartConsole



3. Publish the changes and verify again if it is updated in SmartConsole.

4. Next add 2 new hosts in the hosts.tf file and add a new network

5. Now apply this new configuration again using what you have learned so far.

6. Add an additional rule with the new network and new hosts

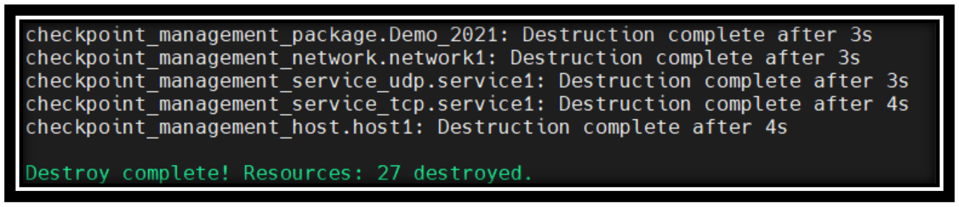
7. Now use SmartConsole to disable a rule, negate a source and make a rule ANY/ANY/ANY. Publish your changes

8. Run a terraform apply and verify what is going to be updated/fixed.

9. Did the policy go back to where it started based on what is in the terraform files?

10. Now delete the entire configuration using the following command

```
terraform destroy
```



```
checkpoint_management_package.Demo_2021: Destruction complete after 3s
checkpoint_management_network.network1: Destruction complete after 3s
checkpoint_management_service_udp.service1: Destruction complete after 3s
checkpoint_management_service_tcp.service1: Destruction complete after 4s
checkpoint_management_host.host1: Destruction complete after 4s

Destroy complete! Resources: 27 destroyed.
```

11. Once the destroy is complete run the follow to publish the changes with the python script

```
python3 publish.py
```