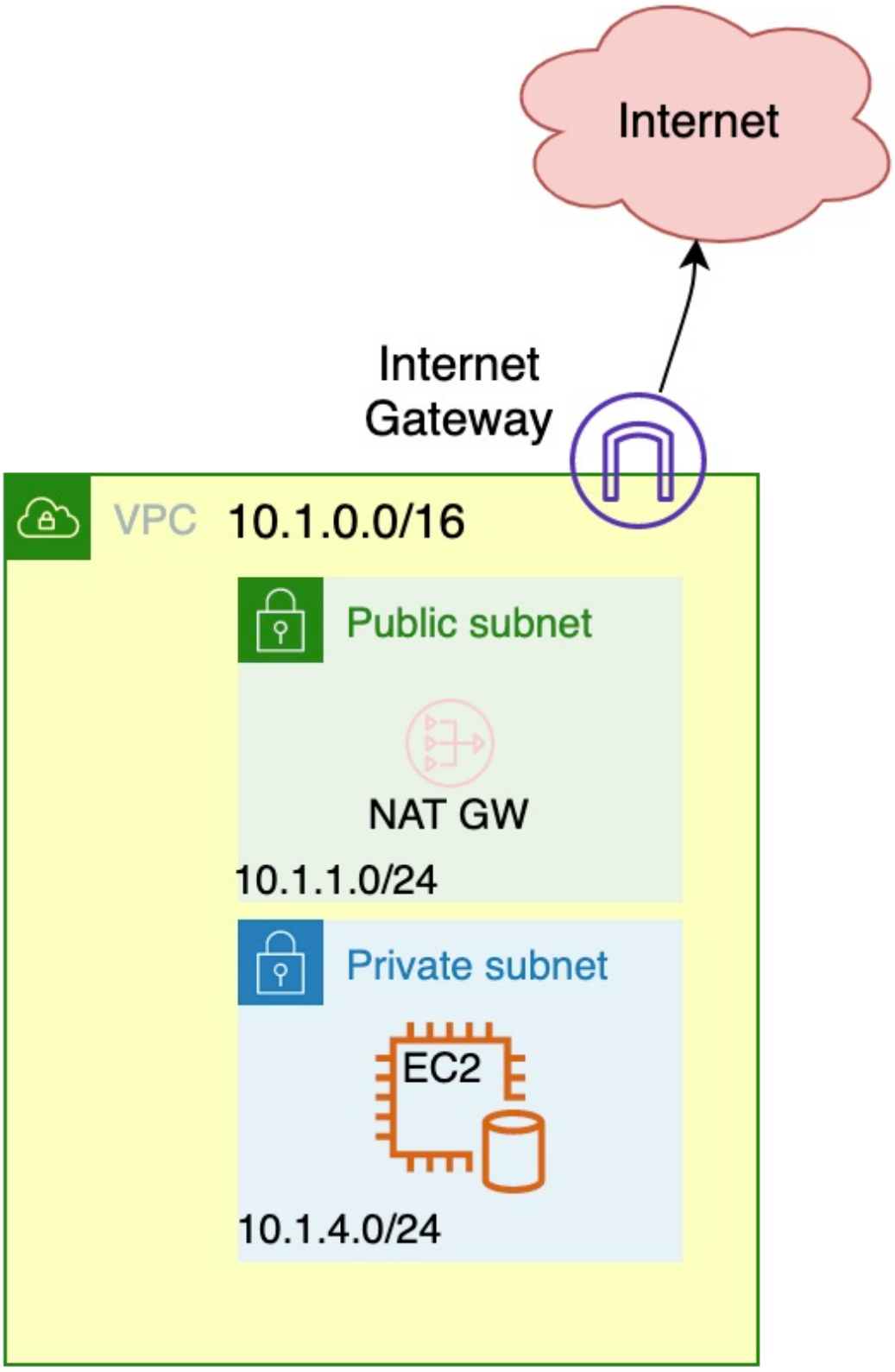
Terraform Networking - Part 1

Simplified architecture diagram for Part 1


Internet
Internet Gateway
VPC 10.1.0.0/16
Public subnet
NAT GW
10.1.1.0/24
Private subnet
EC2
10.1.4.0/24

Hands-on creating VPCs using Terraform

In your Cloud9 IDE terminal change directory to:

```
1
cd ~/environment/tfekscode/primer/tflab1
```

Look at the contents of the file aws.tf - this file is specifying to Terraform:

- Which version of Terraform should be used **required_version = "~> 0.15.3"**
- Where the AWS "provider" comes from and it's version
- And for the AWS provider itself which region to use and where to get the AWS login credentials

```
1
more aws.tf
terraform {
 required_version = "> 1.5.0"
 required_providers {
  aws = {
   source = "hashicorp/aws"
   #  Fix version version of the AWS provider
   version = "= 5.3.0"
  }
 }
}

provider "aws" {
 region                = var.region
 shared_credentials_files = ["~/.aws/credentials"]
 profile               = "default"
}

variable "region" {
 description = "The name of the AWS Region"
```

```
  type        = string
  default     = "eu-west-1"
}
```

Initialize Terraform

In the first step we initialize Terraform, most Terraform commands will give output telling you what they have done. In this case we will have some local files placed in a hidden directory `.terraform` which Terraform uses for your subsequent Terraform operations. This step is always required.

```
1
terraform init
```

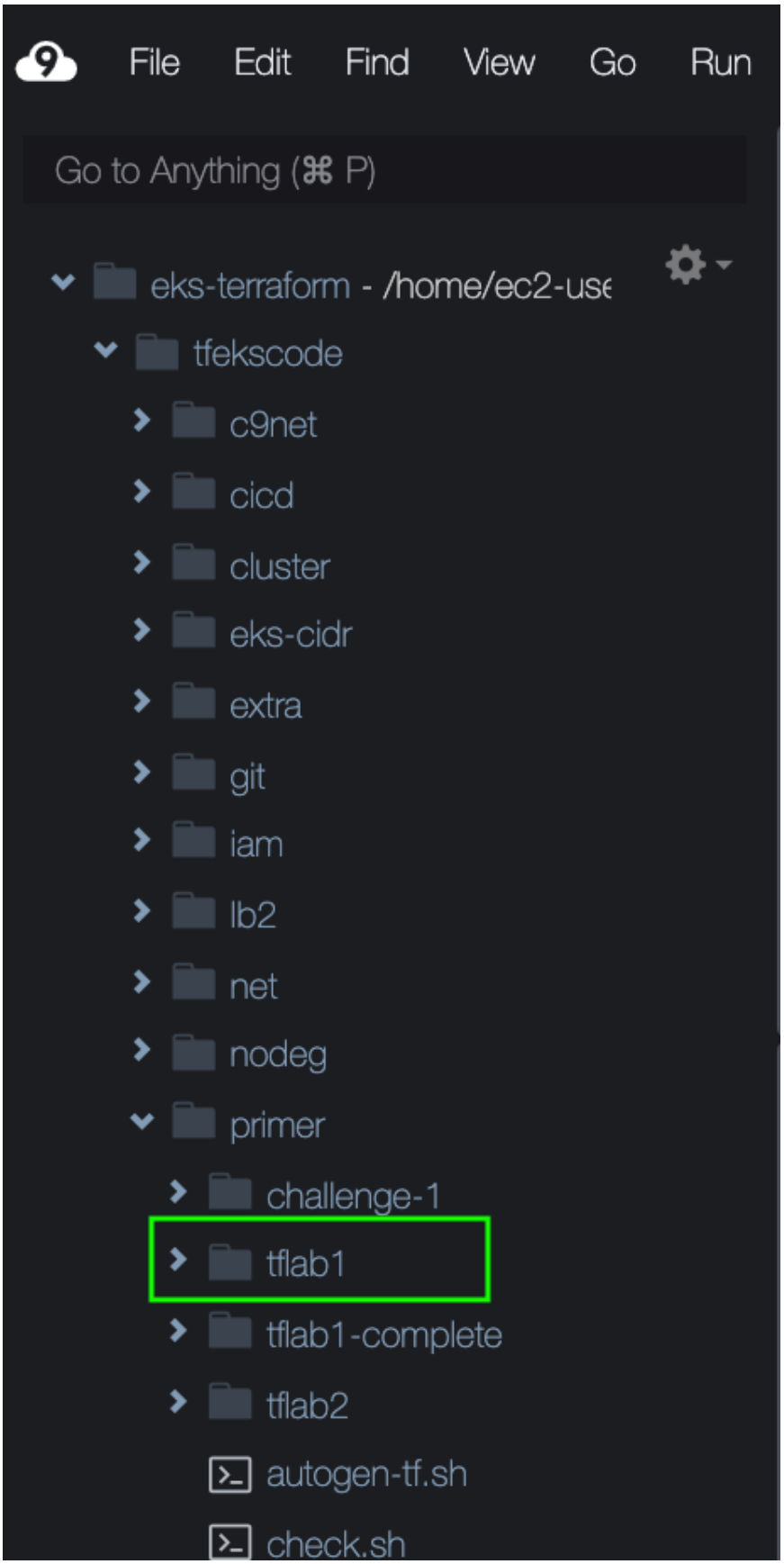
You should see some output that looks like:

```
Initializing the backend...

Initializing provider plugins...
- Finding hashicorp/aws versions matching "5.3.0"...
- Installing hashicorp/aws v5.3.0...
- Installed hashicorp/aws v5.3.0 (signed by HashiCorp)

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!

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.

Create a Terraform file to create a new VPC:

- Make sure you have navigated and clicked the "tflab1" directory in the Cloud9 file browser:

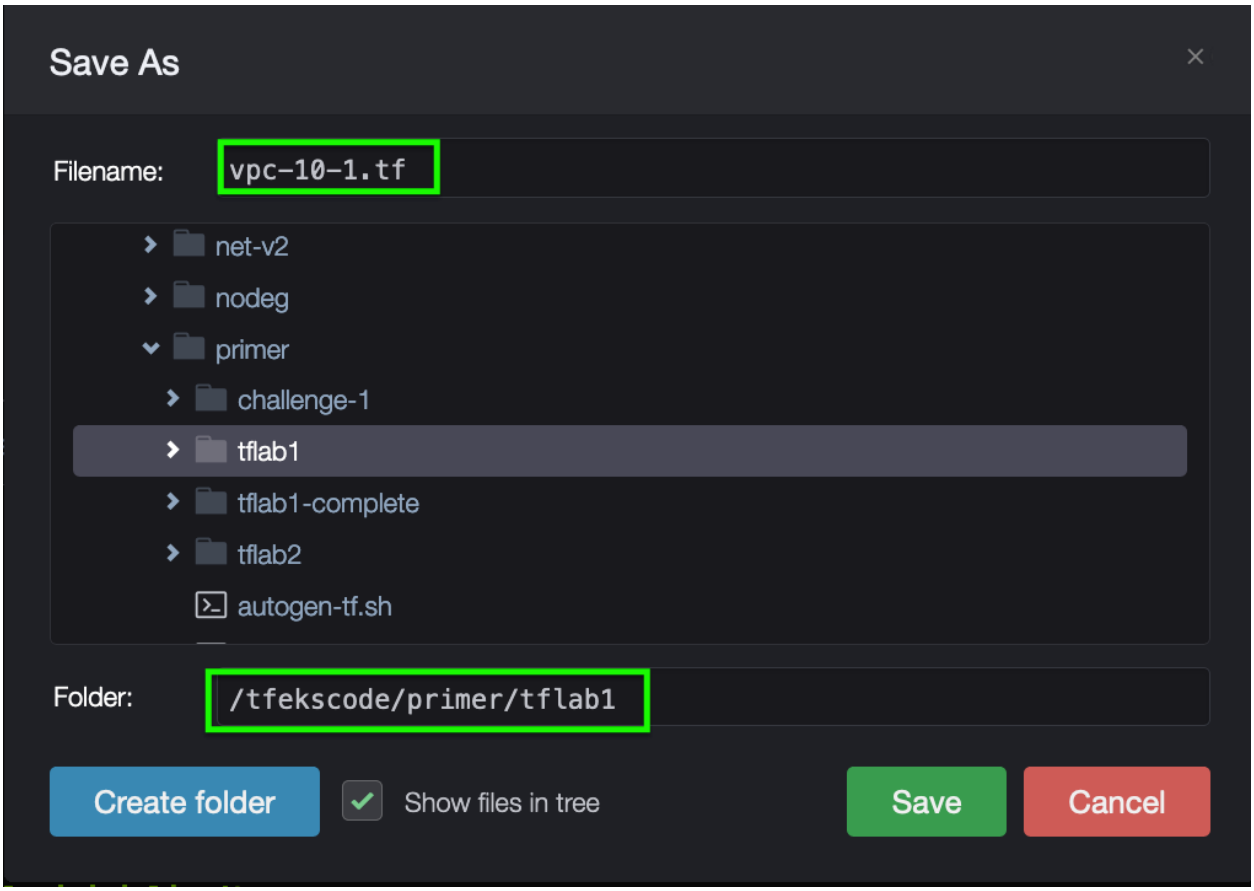

File
Edit
Find
View
Go
Run
Go to Anything (⌘ P)
eks-terraform - /home/ec2-use
tfekscode
c9net
cicd
cluster
eks-cidr
extra
git
iam
lb2
net
nodeg
primer
challenge-1
tflab1
tflab1-complete
tflab2
autogen-tf.sh
check.sh

- File -> New File

Cut and paste these contents into the new file (use the copy icon top right):

```
resource "aws_vpc" "vpc-10-1" {
 assign_generated_ipv6_cidr_block = false
 cidr_block                       = "10.1.0.0/16"
 enable_dns_hostnames             = true
 enable_dns_support               = true
 instance_tenancy                 = "default"
 tags = {
  "Name" = "vpc-10-1"
 }
}
```

File - Save As.... Use the file name **vpc-10-1.tf** Be sure the filename and the path are specified as follows:



Look at the terraform documentation for examples of how to write infrastructure as code https://www.terraform.io/docs/providers/aws/r/vpc.html

---

Next run these commands which will format the terraform code, validate it's syntax:

```
1
terraform fmt
```

Note this command only produces output if it finds a Terraform file that needs to be reformatted.

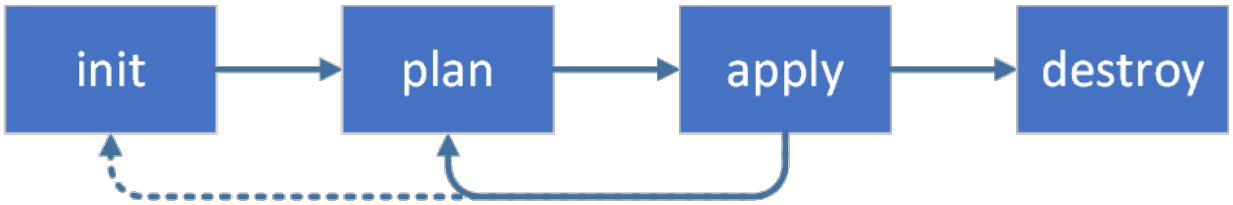
Next validate the file contents. The validate step runs checks that verify whether a configuration is syntactically valid and internally consistent.

```
1
terraform validate
```

Success! The configuration is valid.

Next lets "plan" our terraform infrastructure and get Terraform to tell us what it is going to do

Terraform is used in several stages, you have already completed the initialize stage.



1

terraform plan -out tfplan

Terraform give us a "plan" of what it's actions will be:

Refreshing Terraform state in-memory prior to plan...

The refreshed state will be used to calculate this plan, but will not be

persisted to local or remote state storage.

An execution plan has been generated and is shown below.

Resource actions are indicated with the following symbols:

+ create

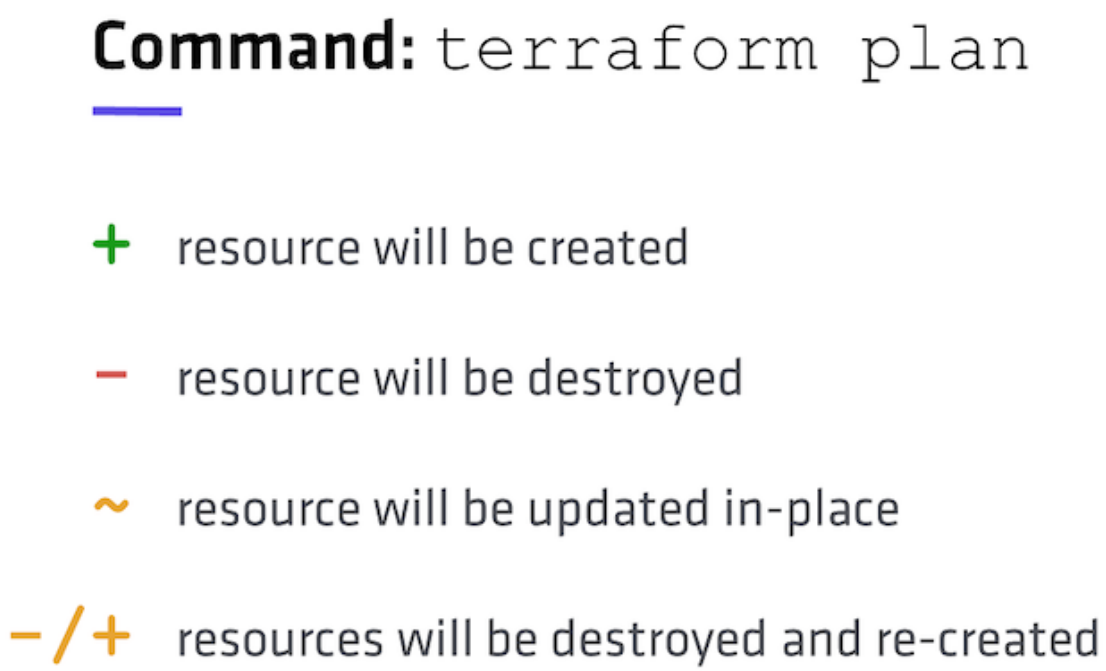
Terraform will perform the following actions:

```
 # aws_vpc.vpc-10-1 will be created
 + resource "aws_vpc" "vpc-10-1" {
    + arn                              = (known after apply)
    + assign_generated_ipv6_cidr_block = false
    + cidr_block                       = "10.1.0.0/16"
    + default_network_acl_id           = (known after apply)
    + default_route_table_id           = (known after apply)
    + default_security_group_id        = (known after apply)
    + dhcp_options_id                  = (known after apply)
```

```
  + enable_classiclink             = (known after apply)
  + enable_classiclink_dns_support   = (known after apply)
  + enable_dns_hostnames           = false
  + enable_dns_support             = true
  + id                             = (known after apply)
  + instance_tenancy               = "default"
  + ipv6_association_id            = (known after apply)
  + ipv6_cidr_block                = (known after apply)
  + main_route_table_id            = (known after apply)
  + owner_id                       = (known after apply)
  + tags                           = {
      + "Name" = "vpc-10-1"
    }
}

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

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

Note: You didn't specify an "-out" parameter to save this plan, so Terraform
can't guarantee that exactly these actions will be performed if
"terraform apply" is subsequently run.
```

This is a "dry run" and shows which actions will be made. This allows manual verification of the changes before running the apply step.

Terraform uses these symbols to indicate the type of changes it will make:



Next lets build the VPC !

1
terraform apply tfplan

This command produces output similar to:

aws_vpc.vpc-10-1: Creating...
aws_vpc.vpc-10-1: Creation complete after 1s [id=vpc-0817cafa92c07b435]

Apply complete! Resources: 1 added, 0 changed, 0 destroyed.

Return to the console - look to see if you now have a new VPC ?

Terraform State

Terraform has now also created a local "state" file with it's own understanding of your provisioned resources

look in the directory tflab1 for this file

```
1
ls terraform.tfstate
```

You can list the contents of the local state. Try using these commands:

```
1
terraform state list
```

aws_vpc.vpc-10-1

```
1
terraform state show aws_vpc.vpc-10-1
```

Notice how similar this output is to our original vpc-10-1.tf file

---

Terraform will undertake the task of keeping it's local state file in sync with the AWS resources and understanding when the Local Terraform files *.tf no longer match the terraform state (and therefore the AWS state)

You can see this in action straight away:

re-run the plan command

```
1
terraform plan -out tfplan
```

This outputs:

```
aws_vpc.vpc-10-1: Refreshing state... [id=vpc-0c5fe24b92b09dbfd]


No changes. Your infrastructure matches the configuration.


Terraform has compared your real infrastructure against your configuration and found no differences, so no changes are needed.
```

Note what this does:

- It updates it's local state first
- Then it re-reads your local .tf files
- Then it figures out if any changes need to be made, new infrastructure deployed or even destroyed.

Try making some changes and see what happens

In the file vpc-10-1.tf

change enable_dns_hostname to be false

Remember to Save the file

re-run:

```
1
terraform plan -out tfplan
```

Note the output says "1 to change" - and it also highlights the parameter that changed

Next apply the change

```
1
terraform apply tfplan
```

Now change it back to true, re-save and plan and apply again

Now proceed to part 2