Lab: Terraform State Migration

As your maturity and use of Terraform develops there may come a time when you need change the backend type you are using. Perhaps you are onboarding new employees and now need to centralize state. You might be part of a merger/acquistion where you need to onboard another organization's Terraform code to your standard configuration. You may simply like to move from a standard backend to an enhanced backend to leverage some of those backend features. Luckily Terraform makes it relatively easy to change your state backend configuration and migrate the state between backends along with all of the data that the state file contains.

- Task 1: Use Terraform's default `local` backend
- Task 2: Migrate State to `s3` backend
- Task 3: Migrate State to `remote` backend
- Task 4: Migrate back to `local` backend

Task 1: Use Terraform's default `local` backend

Update the terraform configuration block within the `terraform.tf` and remove the `backend`. This will indicate to Terraform to use it's deafult `local` backend and store the contents of state inside a `terraform.tfstate` file locally inside the working directory.

```
terraform {
  required_version = ">= 1.0.0"
  required_providers {
    aws = {
      source  = "hashicorp/aws"
      version = "~> 3.0"
    }
    http = {
      source  = "hashicorp/http"
      version = "2.1.0"
    }
    random = {
      source  = "hashicorp/random"
      version = "3.1.0"
    }
    local = {
      source  = "hashicorp/local"
      version = "2.1.0"
    }
    tls = {
      source  = "hashicorp/tls"
```


```
      version = "3.1.0"
    }
  }
}
```

Validate your configuration and re-intialize to terraform's default `local` backend.

```
terraform validate
terraform init -migrate-state
```

```
Initializing the backend...
Terraform has detected you're unconfiguring your previously set "remote" backend.

Successfully unset the backend "remote". Terraform will now operate locally.

Initializing provider plugins...
- Reusing previous version of hashicorp/tls from the dependency lock file
- Reusing previous version of hashicorp/aws from the dependency lock file
- Reusing previous version of hashicorp/http from the dependency lock file
- Reusing previous version of hashicorp/random from the dependency lock file
- Reusing previous version of hashicorp/local from the dependency lock file
- Using previously-installed hashicorp/tls v3.1.0
- Using previously-installed hashicorp/aws v3.65.0
- Using previously-installed hashicorp/http v2.1.0
- Using previously-installed hashicorp/random v3.1.0
- Using previously-installed hashicorp/local v2.1.0

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.
```

Build the infrastructure and state using a `terraform apply`

```
terraform apply

Plan: 27 to add, 0 to change, 0 to destroy.

Do you want to perform these actions?
  Terraform will perform the actions described above.
  Only 'yes' will be accepted to approve.

  Enter a value: yes
```

Validate the buildout of infrastructure and state were successful by listing the items in state:

```
terraform state list
```

Task 2: Migrate State to s3 backend

We will now migrate existing state from the default `local` backend to our `s3` backend to store the state information centrally in S3. Update the terraform configuration block within the `terraform.tf` and add our `s3` backend configuration. Remember that a Terraform configuration can only specify a single backend for a given working directory. If a backend is already configured be sure to replace it.

`terraform.tf`

> Note: Don't forget to update the configuration block below to specify your bucket name, key and DynamoDB table name that were created in earlier labs.

Example:

```
terraform {
 backend "s3" {
    bucket = "my-terraform-state-ghm"
    key    = "prod/aws_infra"
    region = "us-east-1"

    dynamodb_table = "terraform-locks"
    encrypt        = true
  }
}
```

Validate your configuration and re-intialize to terraform's `s3` backend.

```
terraform validate
terraform init -migrate-state
```

```
Success! The configuration is valid.


Initializing the backend...
Do you want to copy existing state to the new backend?
  Pre-existing state was found while migrating the previous "local" backend to the
  newly configured "s3" backend. No existing state was found in the newly
  configured "s3" backend. Do you want to copy this state to the new "s3"
  backend? Enter "yes" to copy and "no" to start with an empty state.

  Enter a value: yes
```


```
Successfully configured the backend "s3"! Terraform will automatically
use this backend unless the backend configuration changes.
```

Validate that the migration was successful by listing the items in state using the new backend.

```
terraform state list
```

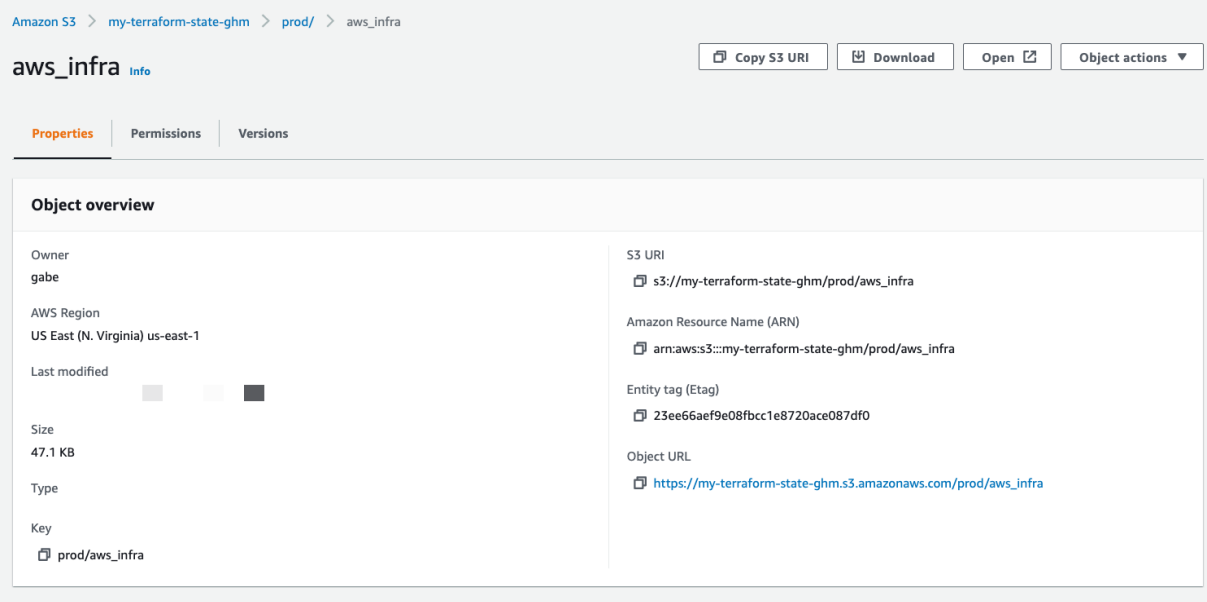

Figure 1: Remote State S3

Congratulations! You have succesfully migrated active state from the `local` to `s3` backend and are now storing state remotely. With the `s3` standard backend you are able to share your workspace with teammates, leverage state locking, versioning and the encryption features of the `s3` backend.

If you can read the state information then the migration was sucessful. You can now remove the local `terraform.tfstate` and `terraform.tfstate.backup` files locally in your working directory.

Task 3: Migrate State to `remote` backend

We will now migrate existing state from the `s3` standard backend to the `remote` enhanced backend to store the state information centrally in Terraform Cloud. Update the terraform configuration block within the `terraform.tf` to replace the `s3` backend configuration with the `remote` backend configuration. Remember that a Terraform configuration can only specify a single backend for a given working

directory.

`terraform.tf`

> Note: Don't forget to update the configuration block below to specify your Terraform Cloud organization and workspace name that were created in earlier labs.

Example:

```
terraform {
   backend "remote" {
    hostname = "app.terraform.io"
    organization = "Enterprise-Cloud"

    workspaces {
      name = "my-aws-app"
    }
  }
}
```

Validate your configuration and re-intialize to terraform's `remote` backend.

```
terraform validate
terraform init -migrate-state
```

```
Initializing the backend...
Backend configuration changed!

Terraform has detected that the configuration specified for the backend
has changed. Terraform will now check for existing state in the backends.


Terraform detected that the backend type changed from "s3" to "remote".
Do you want to copy existing state to the new backend?
  Pre-existing state was found while migrating the previous "s3" backend to the
  newly configured "remote" backend. No existing state was found in the newly
  configured "remote" backend. Do you want to copy this state to the new "remote"
  backend? Enter "yes" to copy and "no" to start with an empty state.

  Enter a value: yes

Successfully configured the backend "remote"! Terraform will automatically
use this backend unless the backend configuration changes.
```

Validate that the migration was successful by listing the items in state using the new backend.

```
terraform state list
```


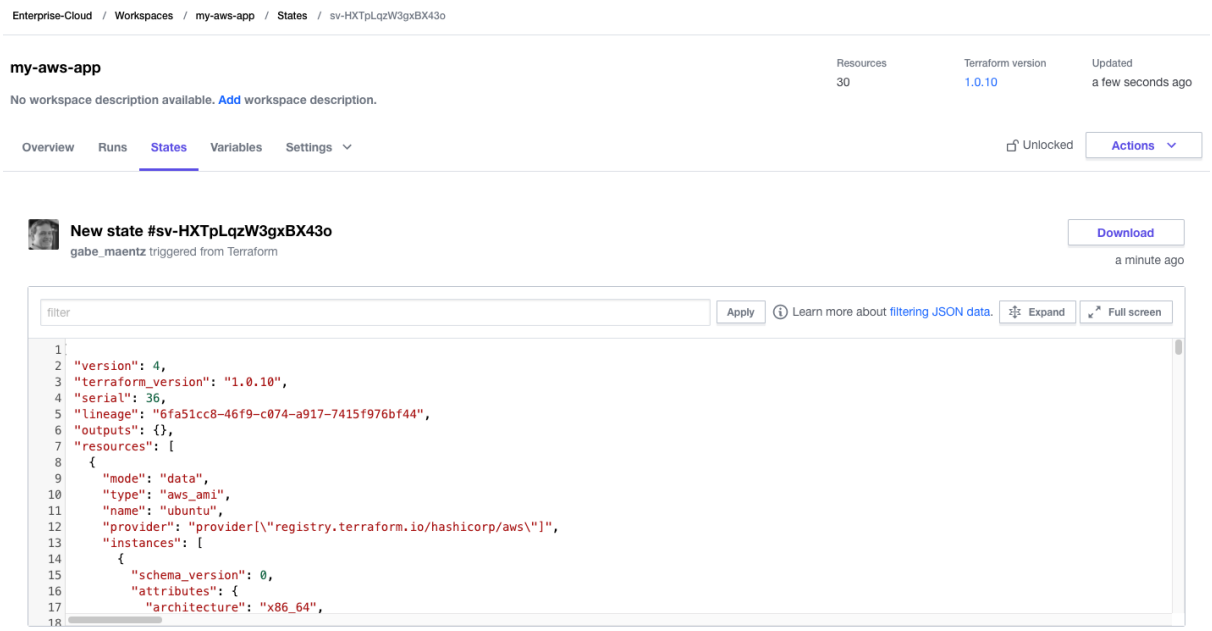

Figure 2: Remote State TFC

Congratulations! You have succesfully migrated active state from the `s3` to `remote` backend and are now storing state remotely in Terraform Cloud. With the `remote` enhanced backend you are able to share your workspace with teammates, leverage state locking, versioning, encryption and centerally perform operations using the features of Terraform Cloud.

Task 4: Migrate back to `local` backend

Now that we have migrated our state to several different backend types, let's show how to restore the Terraform state back to it's default `local` backend.

Update the terraform configuration block within the `terraform.tf` and remove the `backend`. This will indicate to Terraform to use it's deafult `local` backend and store the contents of state inside a `terraform.tfstate` file locally inside the working directory.

```
terraform {
  required_version = ">= 1.0.0"
  required_providers {
    aws = {
      source  = "hashicorp/aws"
      version = "~> 3.0"
    }
```


```
    http = {
      source  = "hashicorp/http"
      version = "2.1.0"
    }
    random = {
      source  = "hashicorp/random"
      version = "3.1.0"
    }
    local = {
      source  = "hashicorp/local"
      version = "2.1.0"
    }
    tls = {
      source  = "hashicorp/tls"
      version = "3.1.0"
    }
  }
}
```

Validate your configuration and re-intialize to terraform's `local` backend.

```
terraform validate
terraform init -migrate-state
```

```
Initializing the backend...
Terraform has detected you're unconfiguring your previously set "remote" backend.
Do you want to copy existing state to the new backend?
  Pre-existing state was found while migrating the previous "remote" backend to the
  newly configured "local" backend. No existing state was found in the newly
  configured "local" backend. Do you want to copy this state to the new "local"
  backend? Enter "yes" to copy and "no" to start with an empty state.

  Enter a value: yes

Successfully unset the backend "remote". Terraform will now operate locally.
```

Validate that the migration was successful by listing the items in state using the new backend.

```
terraform state list
```

As part of this migration back to the `local` backend you will see that terraform created a `terraform.tfstate` file in which the state is now being managed.

Congratulations! You have succesfully migrated active state from `local` to `s3` to `remote` and back again. You are once again storing state locally within a `terraform.tfstate` file of your working directory. As you can see, Terraform makes it relatively easy to change your state backend configuration and migrate to the backend that is appropriate for your team.