Terraform Remote Backend on Azure (Storage Container)

Overview

- Terraform allows you to store your state file remotely using an Azure Storage Container.
- This ensures better collaboration and state consistency across teams.
- Additionally, it provides state locking which prevents simultaneous operations that could corrupt the state.

Prerequisites

Before setting up the remote backend, ensure you have:

- An **Azure Account** with necessary permissions.
- **Azure CLI** installed and configured.
- **Terraform** installed on your system.

Setting Up Storage Account Container for Remote Backend

Steps:

1. Create the project directory: **azure-remote-terraform**.
2. Define providers:
   - Create a *providers.tf* file in the *azure-remote-terraform* directory.
   - Define:
     - terraform
       - required_providers
     - provider
       - azure
   - Reference: providers.tf.
3. Define infrastructure:
   - Create *main.tf* file.
   - Use predefined modules:
     - module.resource-group
     - module.storage
   - Reference: main.tf.
4. Define local variables:
   - Create *locals.tf* file.
   - Define variables:
     - local.subscription-id
     - local.resource-group-properties
     - local.vnet-public-subnet-id
     - local.storage-properties

  ○ Reference: locals.tf.

> Ensure you give the appropriate values to the variables defined in *locals.tf* file.

Provisioning the Infrastructure

Steps:

1. Open PowerShell.
2. Navigate to `azure-remote-terraform` directory.
3. Run:
   ○ **`terraform fmt -recursive`** → Format Terraform files.
   ○ **`terraform init`** → Initialize Terraform.
   ○ **`terraform validate`** → Validate configuration.
   ○ **`terraform plan`** → Plan resource creation.
   ○ **`terraform apply`** → Apply configuration (type `yes` when prompted).
4. Verify the created resources in Azure Console.

Configuring a Sample Project for Remote Backend

Steps:

1. Create the project directory: **sample-terraform**.
2. Define providers:
   ○ Create *providers.tf* file.
   ○ Define:
     ▪ terraform
       ▪ required_providers
         ▪ backend
     ▪ provider
       ▪ azure
   ○ Reference: providers.tf.
3. Define infrastructure:
   ○ Create *main.tf* file.
   ○ Use predefined modules, e.g.,

```
module "resource-group" {
    source = "github.com/inflection-templates/devops-
templates/terraform/modules/azure/resource-group"

    resource-group-properties = local.resource-group-properties
}
```

4. Define local variables:
   ○ Create *locals.tf* file.
   ○ Define

- local.subscription-id
      - local.resource-group-properties
    - Reference: locals.tf.

> Ensure you give the appropriate values to the variables defined in *locals.tf* file.

Provisioning the Sample Infrastructure

Steps:

1. Open PowerShell.
2. Navigate to `azure-remote-terraform`.
3. Run:
    - **`terraform fmt -recursive`** → Format files.
    - **`terraform init`** → Initialize Terraform.
    - **`terraform validate`** → Validate configuration.
    - **`terraform plan`** → Plan resource creation.
    - **`terraform apply`** → Apply configuration (type `yes` when prompted).
4. Verify resources in Azure Console.

Migrating an Existing Terraform State to Remote Backend

Steps:

1. Run **`terraform init -migrate-state`** to migrate local state to Storage Container.
2. Run **`terraform state list`** to verify the migrated resources.
3. Run **`terraform show`** to confirm the remote state.
4. Run **`terraform plan`** and **`terraform apply`** to reapply infrastructure if needed.

Destroying the Infrastructure

Steps:

1. Open PowerShell.
2. Navigate to `azure-remote-terraform`.
3. Run **`terraform destroy`** (type `yes` when prompted).
4. Resources will be deleted.

Conclusion

- By following this guide, you have successfully set up a Terraform remote backend using Azure Storage Container for state storage & locking.
- This ensures secure, scalable, and team-friendly infrastructure management.