Terraform Remote Backend on AWS (S3 & DynamoDB)

Overview

- Terraform allows you to store your state file remotely using an AWS S3 bucket.
- This ensures better collaboration and state consistency across teams.
- Additionally, using AWS DynamoDB for state locking prevents simultaneous operations that could corrupt the state.

Prerequisites

Before setting up the remote backend, ensure you have:

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

Setting Up S3 Bucket and DynamoDB for Remote Backend

Steps:

1. Create the project directory: **aws-remote-terraform**.
2. Define providers:
    - Create a *providers.tf* file in the *aws-remote-terraform* directory.
    - Define:
        - terraform
            - required_providers
        - provider
            - aws
    - Reference: providers.tf.
3. Define infrastructure:
    - Create *main.tf* file.
    - Use predefined modules:
        - module.s3-bucket
        - module.dynamodb
    - Reference: main.tf.
4. Define local variables:
    - Create *locals.tf* file.
    - Define variables:
        - local.s3-bucket-arn
        - local.s3-bucket-properties
        - local.s3-bucket-policy
        - local.dynamodb-table-arn
        - local.dynamodb-properties
        - local.dynamodb-resource-policy

  - 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 `aws-remote-terraform`.
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 AWS 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
            - aws
    - Reference: providers.tf.
3. **Define infrastructure**:
    - Create *main.tf* file.
    - Use predefined modules, e.g.,

```
module "s3-bucket" {
  source = "github.com/inflection-templates/devops-
templates/terraform/modules/aws/s3-bucket"

  s3-bucket-properties = local.s3-bucket-properties
  s3-bucket-policy     = local.s3-bucket-policy
}
```

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

- Define
  - local.s3-bucket-arn
  - local.s3-bucket-properties
  - local.s3-bucket-policy
- 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 `aws-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 AWS Console.

Migrating an Existing Terraform State to Remote Backend

Steps:

1. Run **`terraform init -migrate-state`** to migrate local state to S3.
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 `aws-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 AWS S3 for state storage and DynamoDB for state locking.
- This ensures secure, scalable, and team-friendly infrastructure management.