Terraform: Infrastructure as Code

Backends

terraform init

```
terraform init
...
Initializing the backend...
...
```

Backends

Backends are completely optional.

Here are some of the benefits of backends:

- Working in a team
- Keeping sensitive information off disk
- Remote operations

Backend Types:

- **Standard**: State management, functionality covered in State Storage & Locking
- **Enhanced**: Everything in standard plus remote operations.

Enhanced Backends

- local
- remote(Terraform Cloud)

Standard Backends

- artifactory(with no locking)
- azurerm(with state locking)
- consul(with locking)
- etcd(with no locking)
- etcdv3(with locking)
- gcs(with locking)
- http(with optional locking)
- pg(with locking)
- s3(with locking via DynamoDB)
- swift(with no locking)
- terraform enterprise(with no locking)

backend-config

`backend-config=path`

This can be either a path to an HCL file with key/value assignments (same format as terraform.tfvars) or a 'key=value' format.

```
$ terraform init -backend-config='bucket=mycompany-tfstate' -backend-config='key=prod.tfstate'\
-backend-config='region=eu-central-1'
```

Remote backends solve all three of the issues listed above:

1. **Manual error**: Once you configure a remote backend, Terraform will automatically load the state file from that backend every time you run `plan` or `apply` and it'll automatically store the state file in that backend after each `apply`, so there's no chance of manual error.
2. **Locking**: Most of the remote backends natively support locking. To run `terraform apply`, Terraform will automatically acquire a lock; if someone else is already running `apply`, they will already have the lock, and you will have to wait.
3. **Secrets**: Most of the remote backends natively support encryption in transit and encryption on disk of the state file. Moreover, those backends usually expose ways to configure access permissions (e.g., using IAM policies with an S3 bucket), so you can control who has access to your state files and the secrets the may contain.

src: How to manage Terraform state. A guide to file layout, isolation, and… | by

State in AWS: bootstrap

```
terraform {
  required_version = ">=0.12.24"
  required_providers {
    aws = ">= 2.58.0"
  }
}
provider "aws" {
  region = "eu-central-1"
}
locals {
  tags = {
    source = "terraform"
    env    = var.env
  }
}
resource "aws_s3_bucket" "backend" {
  bucket = "tf-..."
  acl    = "private"

  versioning {
    enabled = true
  }

  tags = local.tags
}
resource "aws_dynamodb_table" "backend_locks" {
  name         = "tf-..."
  billing_mode = "PAY_PER_REQUEST"
  hash_key     = "LockID"

  attribute {
    name = "LockID"
    type = "S"
  }

  tags = local.tags
}
```

State in AWS

```
terraform {
  backend "s3" {
    bucket         = "tf-..."
    encrypt        = true
    key            = "tf-..."
    region         = "eu-central-1"
    dynamodb_table = "tf-..."
  }
  required_version = ">=0.12.24"
  required_providers {
    aws = ">= 2.58.0"
  }
}
provider "aws" {
  region = "eu-central-1"
  assume_role {
    role_arn = var.admin_role
  }
}
```

S3 Remote State

```
data "terraform_remote_state" "prod" {
  backend = "s3"
  config = {
    bucket = "terraform-state-prod"
    key    = "terraform.tfstate"
    region = "eu-central-1"
  }
}
```

The terraform_remote_state data source will return all of the root module outputs defined in the referenced remote state.