# Master SOP: Terraform Backend & Environment Management

## 1. Infrastructure Strategy & Logic

### 1.1 Managing Backend as a Module

Yes, you absolutely can (and should) manage these "backend resources" (the S3 bucket and DynamoDB table) with Terraform itself, rather than using a shell script. Using a module within your root project is a professional way to keep your code DRY (Don't Repeat Yourself) and reproducible. This is often called a **"Bootstrap" process**.

### 1.2 The "Chicken and Egg" Logic is the Two-Phase Workflow

Because your project depends on the S3 bucket it creates, you cannot run a standard terraform apply from the start. This is the "Chicken and Egg" problem: you need the bucket to store the state, but you need the state to manage the bucket.

* **Why a standard run fails:** If you set create\_backend\_resources = false or enable the S3 backend immediately, the code will crash. The module tries to run a data "aws\_s3\_bucket" block to check for the bucket. Since it doesn't exist yet, AWS returns a "Resource not found" error.
* **The Logic:** You must create the storage resources while the state is **"Local"** (on your laptop), then you **"Pivot"** the state to the Cloud.

## 2. The Master Strategy Table

*Reference this table to ensure your variables and files are in the correct state at each step.*

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Step** | **.terraform/ Folder** | **backend.tf Code** | **create\_backend\_resources** | **create\_buckets** | **enable\_cert\_upload** | **Primary Command** |
| **1. Local Reset** | **DELETE** | Commented | **true** | false | false | rm -rf .terraform/ then terraform init -reconfigure |
| **2. Build Storage** | Leave alone | Commented | **true** | false | false | terraform apply -target=module.backend\_infra -var-file="prod.tfvars" |
| **3. Cloud Migration** | Leave alone | **UNCOMMENT** | **true** | false | false | terraform init -migrate-state |
| **4. Final Deploy** | Leave alone | **UNCOMMENT** | **true** | **true** | **true** | terraform apply -var-file="prod.tfvars" |

## 3. Detailed Step-by-Step Execution

### Step 1: The Local Reset

We must force Terraform to forget the broken S3 link and use your laptop's memory.

1. **Edit backend.tf:** Wrap the terraform { backend "s3" {...} } block in /\* ... \*/ comments.
2. **Commands:**  
   Bash  
   rm -rf .terraform/  
   terraform init -reconfigure

* **Result:** Terraform is now in "Local Mode."

### Step 2: Provision the State Storage

We create the S3 bucket that will eventually hold your state.

1. **Edit prod.tfvars:** Set create\_backend\_resources = true, create\_buckets = false, and enable\_cert\_upload = false.
2. **Command:**  
   Bash  
   terraform apply -target=module.backend\_infra -var-file="prod.tfvars"

* **Result:** The bucket grp1-ce11-prod-iot-tfstate now exists in your AWS Console.

### Step 3: Enable Remote State (The Migration)

Now we move your local memory (including the resources already in state) up to the cloud.

1. **Edit backend.tf:** Remove the /\* and \*/ comments to uncomment the backend block.
2. **Command:**  
   Bash  
   terraform init -migrate-state
3. **Action:** Type **yes** when asked to copy the local state to S3.

* **Result:** Your .tfstate file is now safely stored in the S3 bucket.

### Step 4: Final Infrastructure Build

Now we build the rest of the stack (VPC, ECS, ALB) and connect your certificates.

1. **Edit prod.tfvars:** Set create\_buckets = true and enable\_cert\_upload = true.
2. **Command:**  
   Bash  
   terraform apply -var-file="prod.tfvars"

* **Result:** Everything is deployed. The app\_url will be visible in the outputs.

## 4. Advanced Technical Details & Warnings

### 4.1 The kms\_key\_id Dependency

If your Step 2 creates an S3 bucket with a custom **KMS key** for encryption, your Step 3 (backend.tf) must have the correct permissions to use that key. If you are using default AES256 (as in your code), you are safe. If you ever switch to a custom KMS key, you must ensure your IAM user has kms:Decrypt and kms:GenerateDataKey permissions, or the migration will fail.

### 4.2 The .terraform.lock.hcl File

When you do the Local Reset (Step 1), you delete the .terraform folder, but you should also be aware of the .terraform.lock.hcl file.

* **Recommendation:** When you switch between Local and S3 backends, if you see "Provider requirement" errors, delete the .terraform.lock.hcl file along with the .terraform folder. This forces Terraform to re-calculate the "hashes" for the providers in the new environment.

### 4.3 State file "Lineage"

When you migrate from Local to S3, Terraform checks the "Lineage" (a unique ID in the JSON state file).

* **Warning:** If you ever try to migrate a state file that was created in a different project into this bucket, Terraform will warn you that the "Lineage does not match." **Never force this** unless you are 100% sure, as it can overwrite your existing cloud infrastructure records.

## 5. Maintenance & Environment Teardown (Destroy)

To delete everything safely without "orphaning" resources, you must perform a **Reverse Pivot**:

1. **Phase 1 (Destroy App):** With backend.tf active (uncommented), run terraform destroy -var-file="prod.tfvars".1
2. **Phase 2 (Move to Local):** Comment out the backend.tf code and run terraform init -migrate-state. Type **yes** to pull the state back from S3 to your laptop.
3. **Phase 3 (Destroy Backend):** Run terraform destroy -target=module.backend\_infra -var-file="prod.tfvars". This deletes the S3 bucket and DynamoDB table.

## 6. Handling Interruptions & State Locks

If your session crashes, the DynamoDB lock may stay active, preventing further changes.

1. **Identify Lock ID:** Find the ID in the error message (e.g., b921-f123-4567-890a).
2. **Unlock Command:**  
   Bash  
   terraform force-unlock b921-f123-4567-890a

## 7. Final "Peace of Mind" Checklist

* **Region:** Is the region in provider "aws" the same as the region in backend "s3"? (Must be us-east-1 in both).
* **Bucket Name:** Does the bucket string in backend.tf exactly match the bucket name defined in your module?
* **LockID:** In your DynamoDB resource, the hash\_key **must** be exactly named LockID (case-sensitive) for Terraform to recognize it.
* **VPC Cleanup:** Ensure the old VPC (vpc-090b...) is fully deleted in the AWS Console before starting Step 4.
* **IoT Certs:** Ensure your physical .pem and .key files are in the certs/ folder before Step 4.