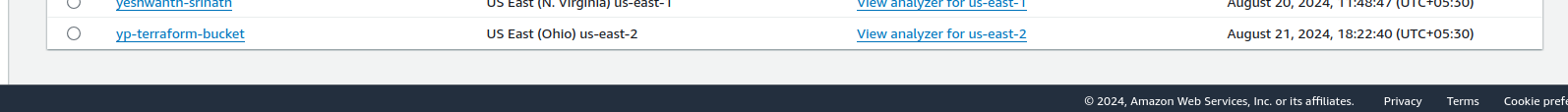
1. **Remote State Management:**
   * **S3 Bucket for State**:
     + Create an S3 bucket using Terraform (this can be separate from the custom module).
     + Configure Terraform to store the state file in the S3 bucket.
   * **State Locking with DynamoDB**:
     + Create a DynamoDB table using Terraform (or manually if required) to store the state lock information.
     + Configure Terraform to use this DynamoDB table for state locking.





1. **Terraform Module Creation:**
   * **Custom Module**:

Create a Terraform module to deploy the following AWS resources:

**EC2 instance**: Use an AMI for the region and allow SSH access using a security group.

**S3 bucket**: Create an S3 bucket for application data.

Use Terraform variables (txvars) to parameterize important aspects such as:

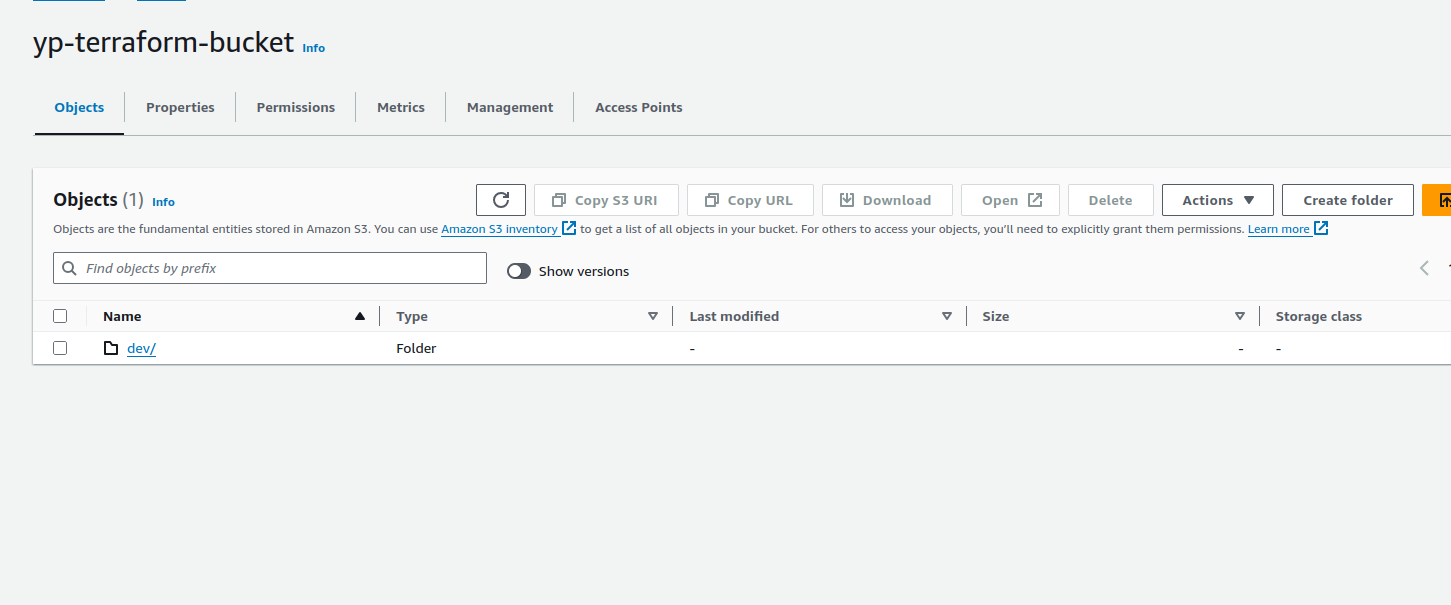
**Instance Type**: Allow the instance type to be configurable (e.g., t2.micro).

**Region**: Parameterize the AWS region so that the module can be reused across regions.

**Bucket Name**: Use a variable to set the S3 bucket name.

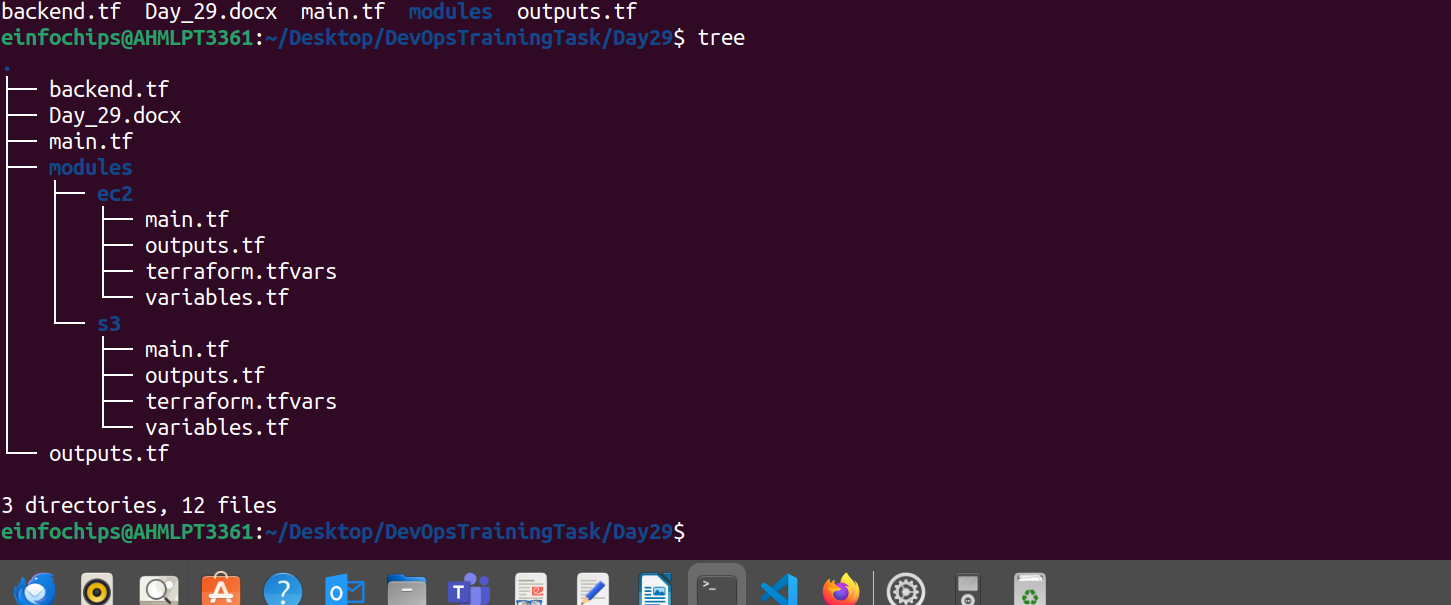
**Terraform Functions**:

* + - Use Terraform functions in your module to manipulate and process the variables. For example:
      * Use join to combine strings for resource names.
      * Use lookup to set default values if a variable is not provided.
      * Use length to count the number of instances or resources.

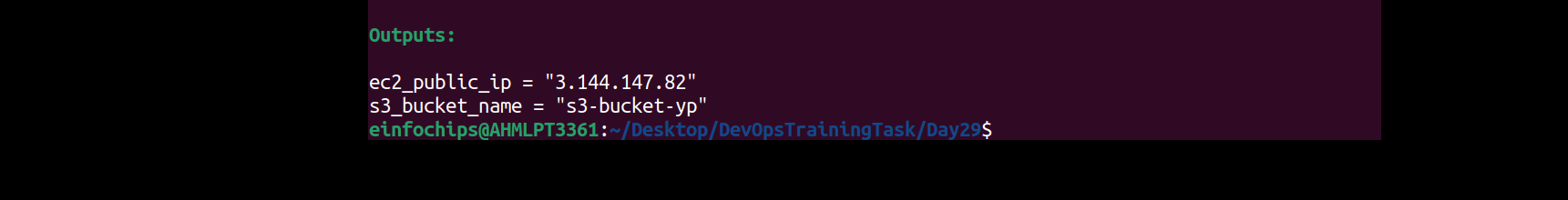




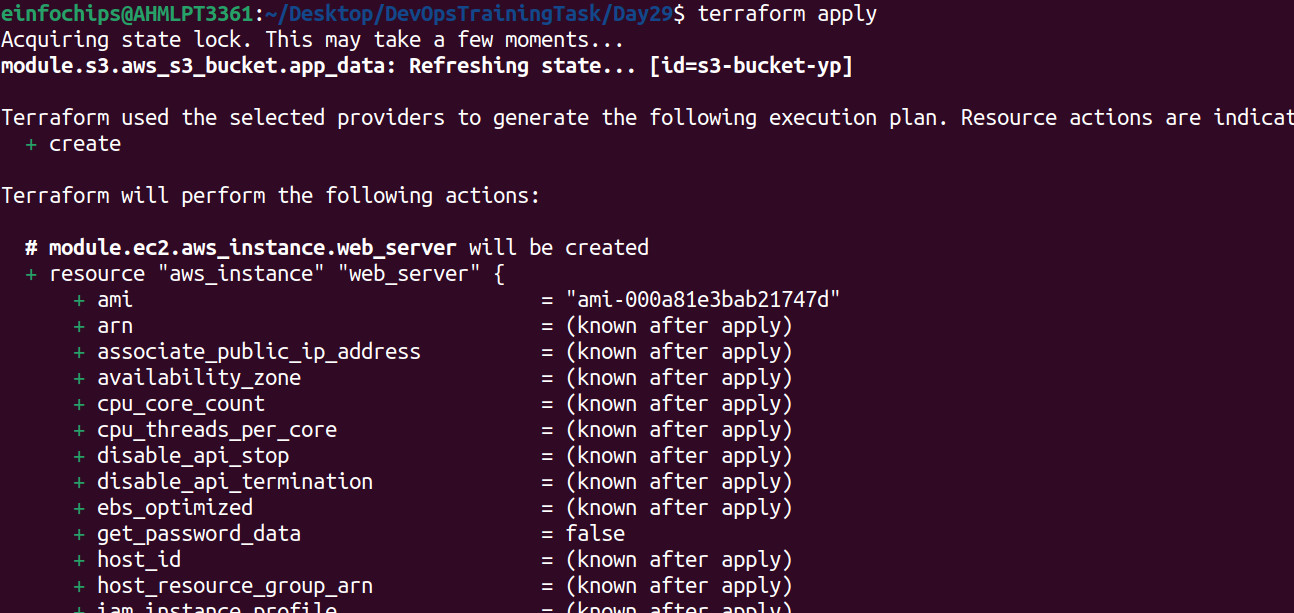
1. **Input Variables and Configuration (txvars):**
   * Define input variables to make the infrastructure flexible:
     + EC2 instance type.
     + S3 bucket name.
     + AWS region.
     + Any other variable relevant to the infrastructure.
   * Use the default argument for variables where appropriate.



1. **Output Configuration:**
   * Set up Terraform outputs to display key information after the infrastructure is created:
     + **EC2 Public IP**: Output the public IP of the EC2 instance.
     + **S3 Bucket Name**: Output the name of the S3 bucket created.
     + **Region**: Output the region where the resources were deployed.



1. **Testing State Locking and Remote State:**
   * **State Locking**:
     + Attempt to run terraform apply from two different terminals simultaneously to test state locking.
     + Confirm that DynamoDB properly handles the state lock, preventing concurrent updates.
   * **Remote State Management**:
     + Verify that Terraform state is being stored in the S3 bucket and that updates are reflected in the remote state file.
2. **Apply and Modify Infrastructure:**
   * **Initial Deployment**:
     + Use terraform plan and terraform apply to deploy the infrastructure.
     + Verify that the EC2 instance, S3 bucket, and all configurations are properly set up.
   * **Infrastructure Changes**:
     + Modify one of the variables (e.g., change the instance type or add tags) and re-run terraform apply.
     + Observe how Terraform plans and applies only the necessary changes, with state locking in effect.



1. **Resource Termination:**
   * Once the deployment is complete and tested, use terraform destroy to tear down all the resources created by Terraform.
   * Ensure that the S3 bucket, EC2 instance, and DynamoDB table (if not reused) are deleted.

