**Modules Directory**

To fully maximize the power of terraform, we must leverage modules. A *module* is a container for multiple resources that are used together. For this task, I created three 3 modules with the module directory:

1. aws-s3: Contains terraform config files for s3 bucket and related resources such as lifecycle rule, bucket policies, object versioning etc.
2. aws-ec2-instance: Contains terraform config files for instance and related resources such as role, instance profile and related resources
3. aws-vpc: Contains terraform config files for vpc, nat gateway, security groups and other relevant resources.
4. aws-sandbox: This module implements aws-s3, aws-ec2-instance and aws-vpc module to deploy all required sandbox resources in a single terraform module block.

Defining the s3, vpc, ec2 module differently will make them easily adoptable for other purposes beyond creating a sandbox. For instance, if an application required an s3 bucket to store client files, we can easily leverage this same s3 module.

**Environment**

For this task, we have one environment called **dev**, which among other possible use cases, this environment will be used for onboarding new developers as mentioned in the task. The environment contains the following configs.   
  
1. Provider.tf: Defined the terraform provider which in this case is AWS, this also include the default tags I want all resources created with this provider to have for easier resource management.

2. backend.tf: This defines the backend where the state files are stored, for this task a local backend was used.

3. sandbox.tf: This contains the actual implementation of the sandboxes for developers.

**Security Considerations**

The following are the security controls in place for the sandbox:

1. The security group only allow ssh/rdp inbound from specific allowed ips, which in this case is the home\_ip.
2. The instance was deployed in a private subnet with no direct internet access. Although a private subnet was also created incase there is a need for that.
3. Traffic from the instance in the private subnet travels through the nat gateway
4. The s3 objects are encrypted with AES256 server side encryption
5. Only SSL Requests are allowed to the bucket
6. Only the IAM user of the bucket owner has the permission to read and write from the bucket
7. All public access to the s3 bucket is blocked

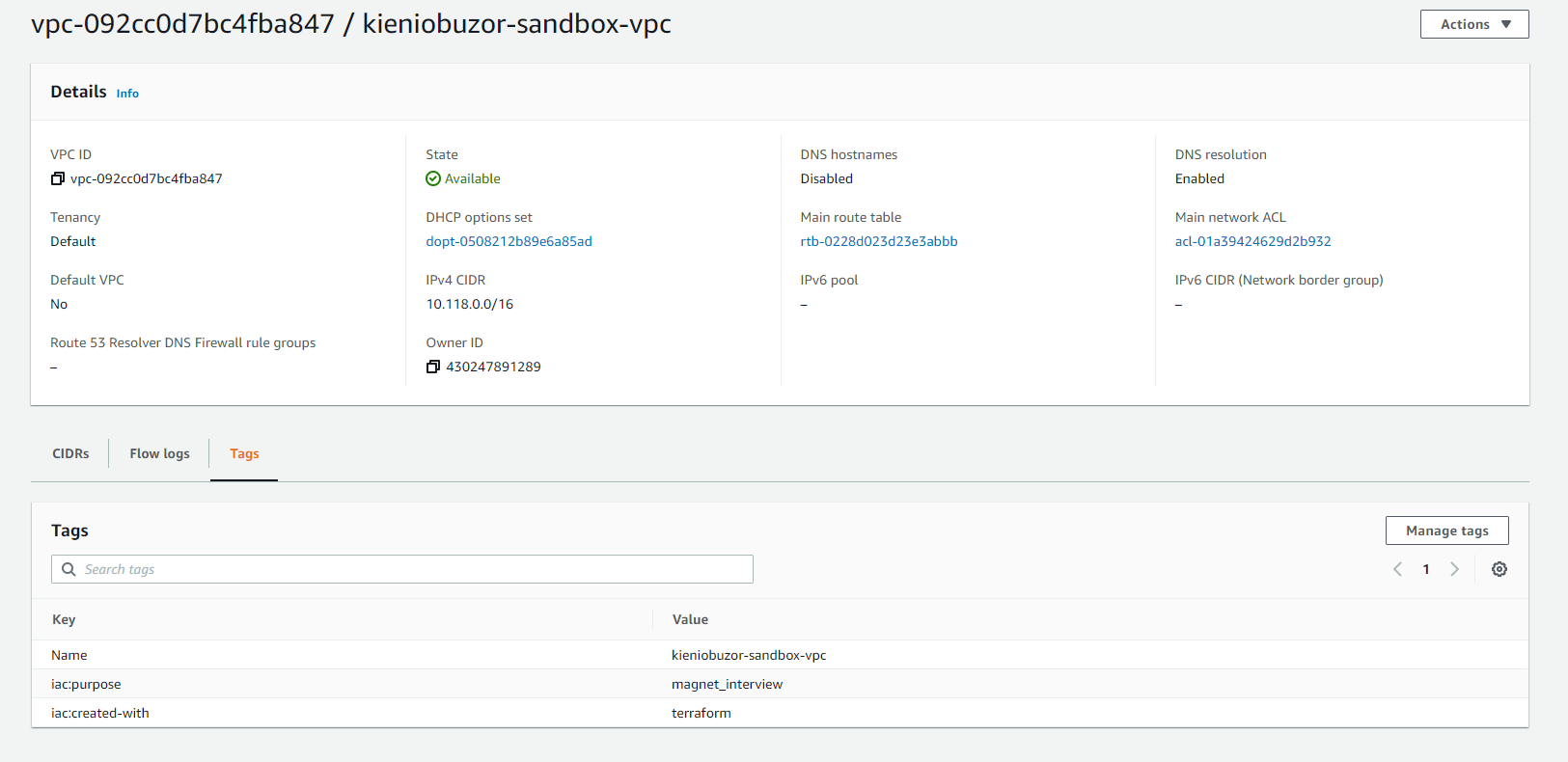
**How to deploy**

1. Create a programmatic user with the right permissions to deploy the required resources and download the access key and secret
2. Run **aws configure** to setup the access key and secret key for terraform
3. cd **environment/dev**
4. Run **terraform init** to initialize the terraform modules
5. Run **terraform plan** to see a speculative plan of the resources that would be deployed
6. Run **terraform apply** to deploy the resources in aws

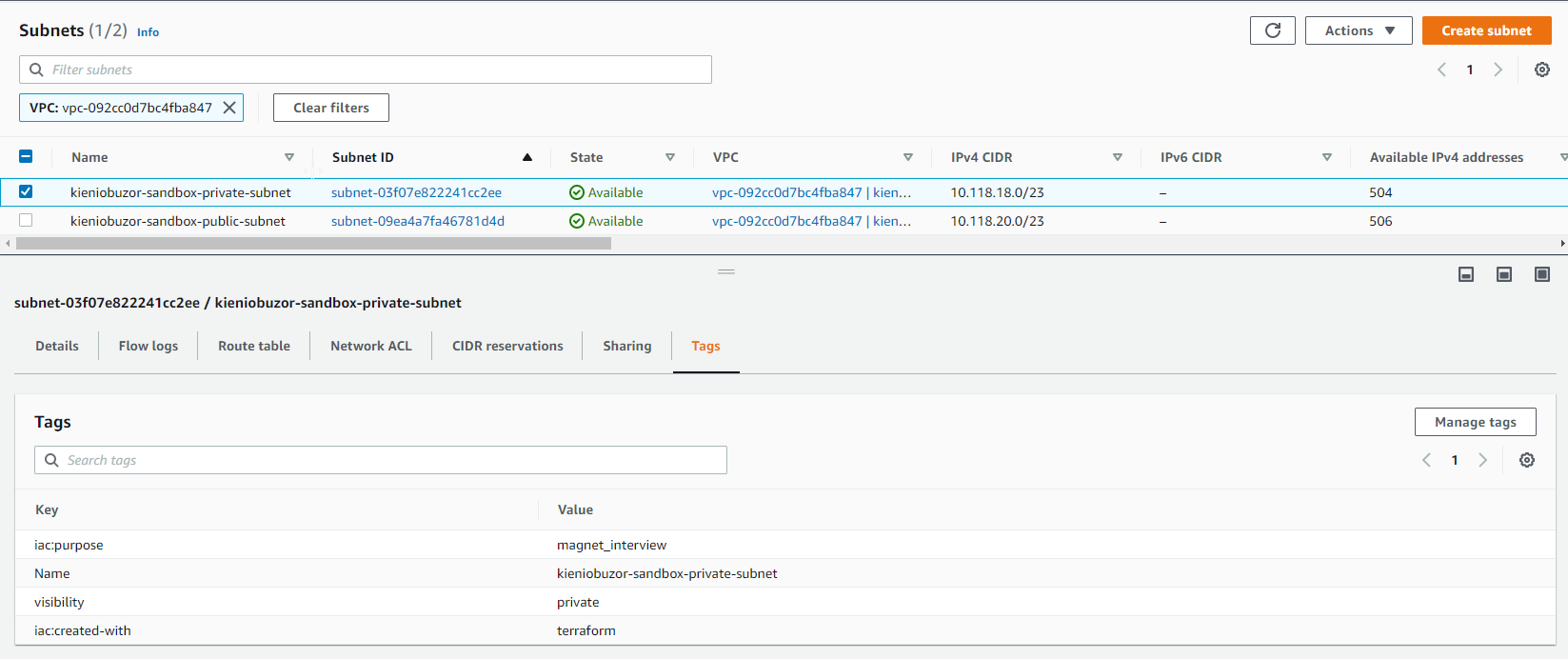
**Screenshots**

Below are screenshots of resources deployed within AWS:

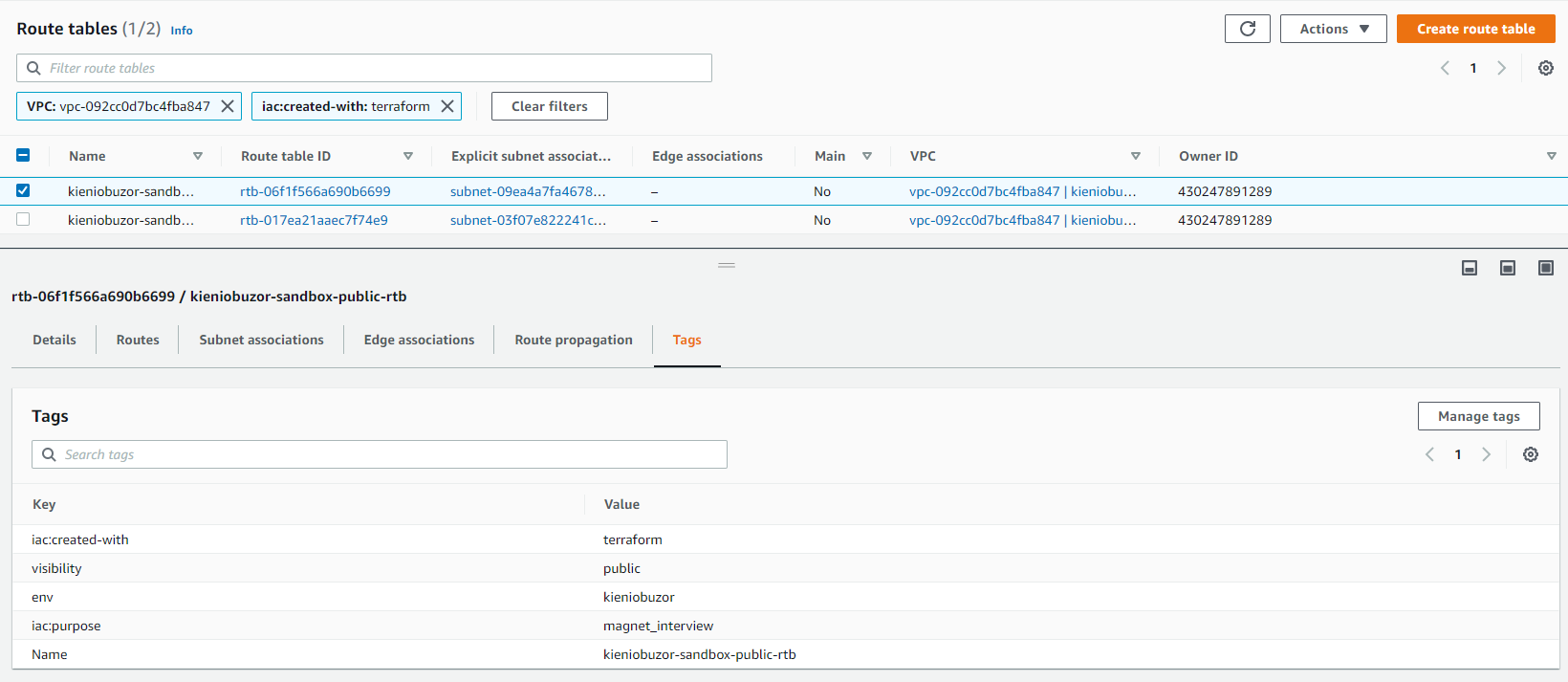
1. VPC



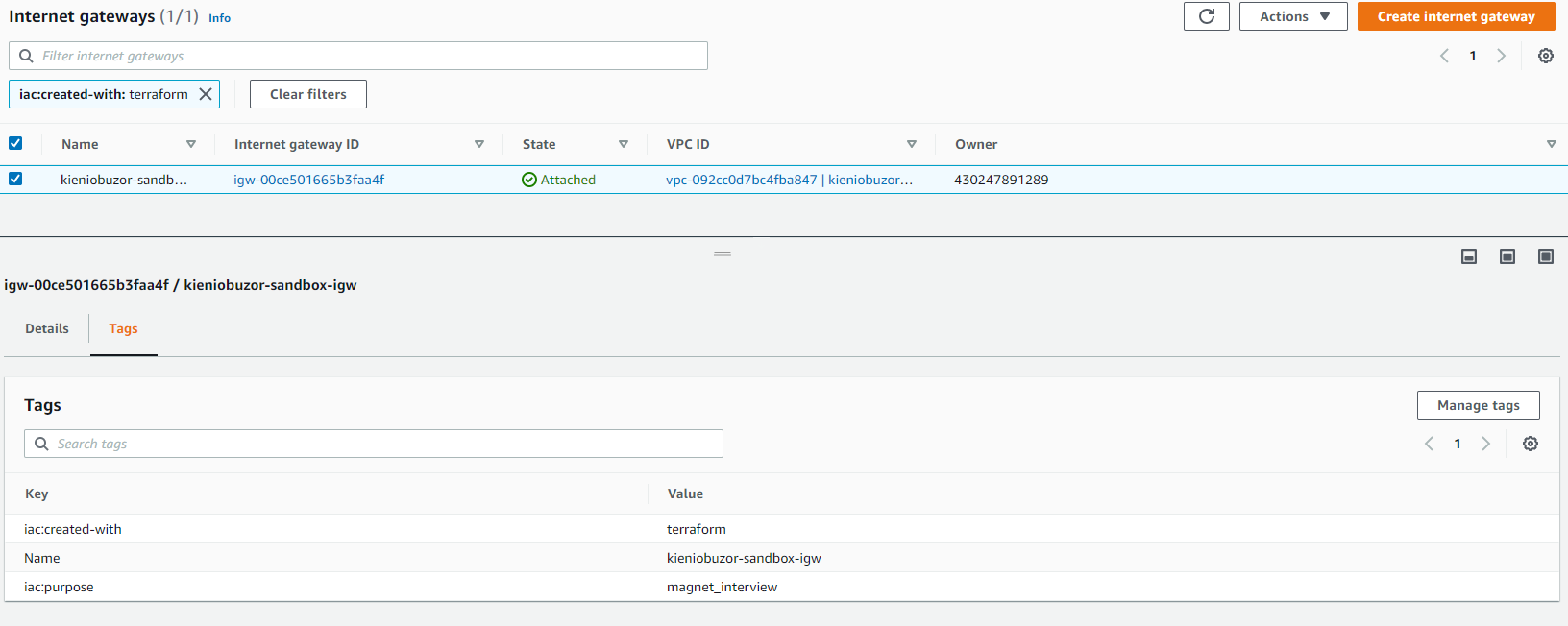
1. Subnets



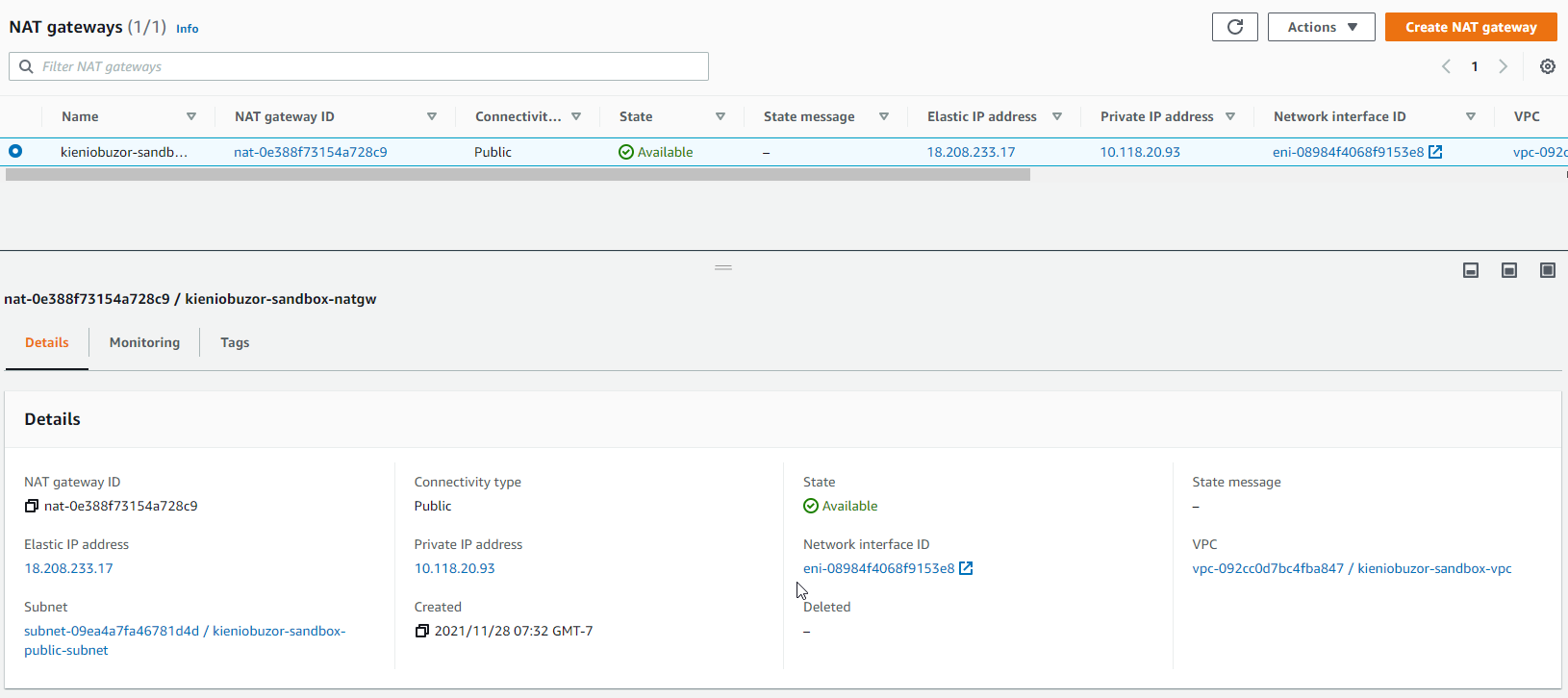
1. Route Table



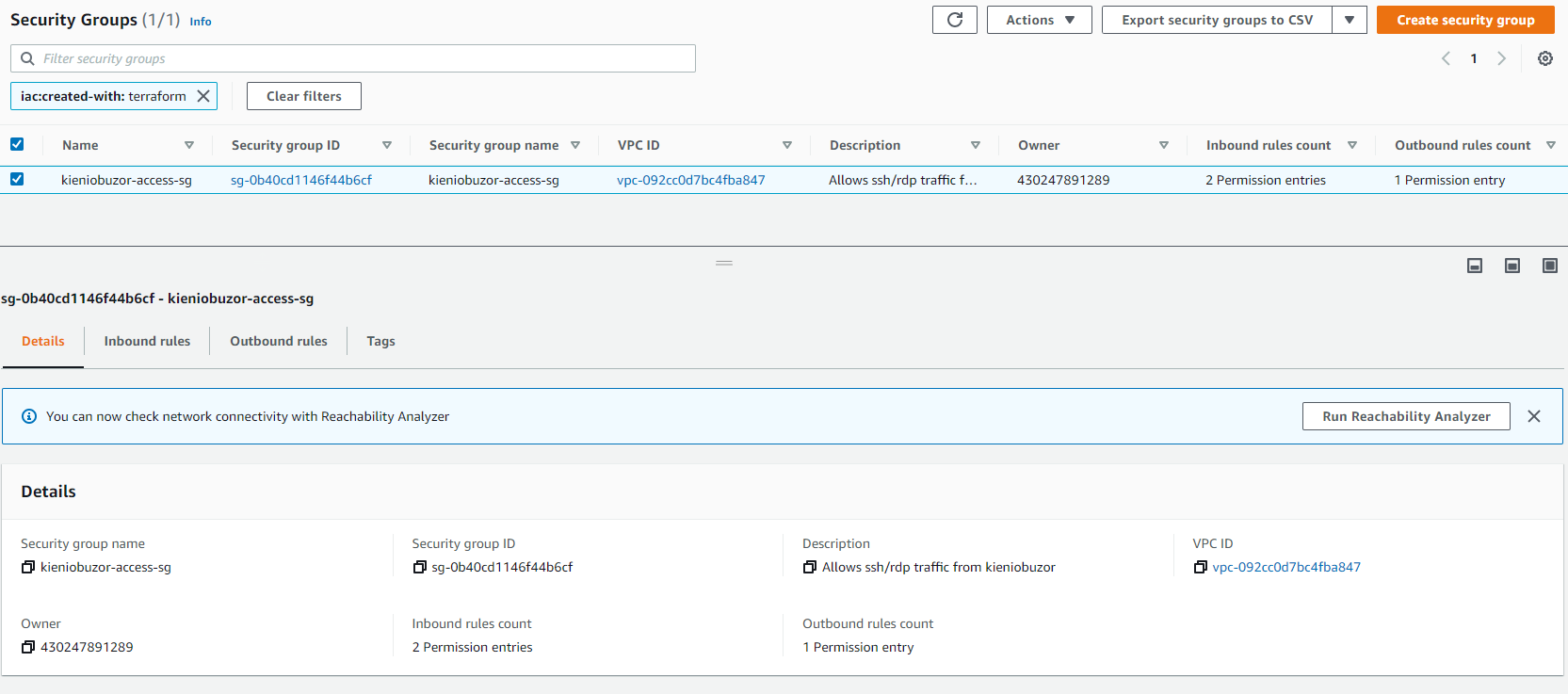
1. Internet Gateway



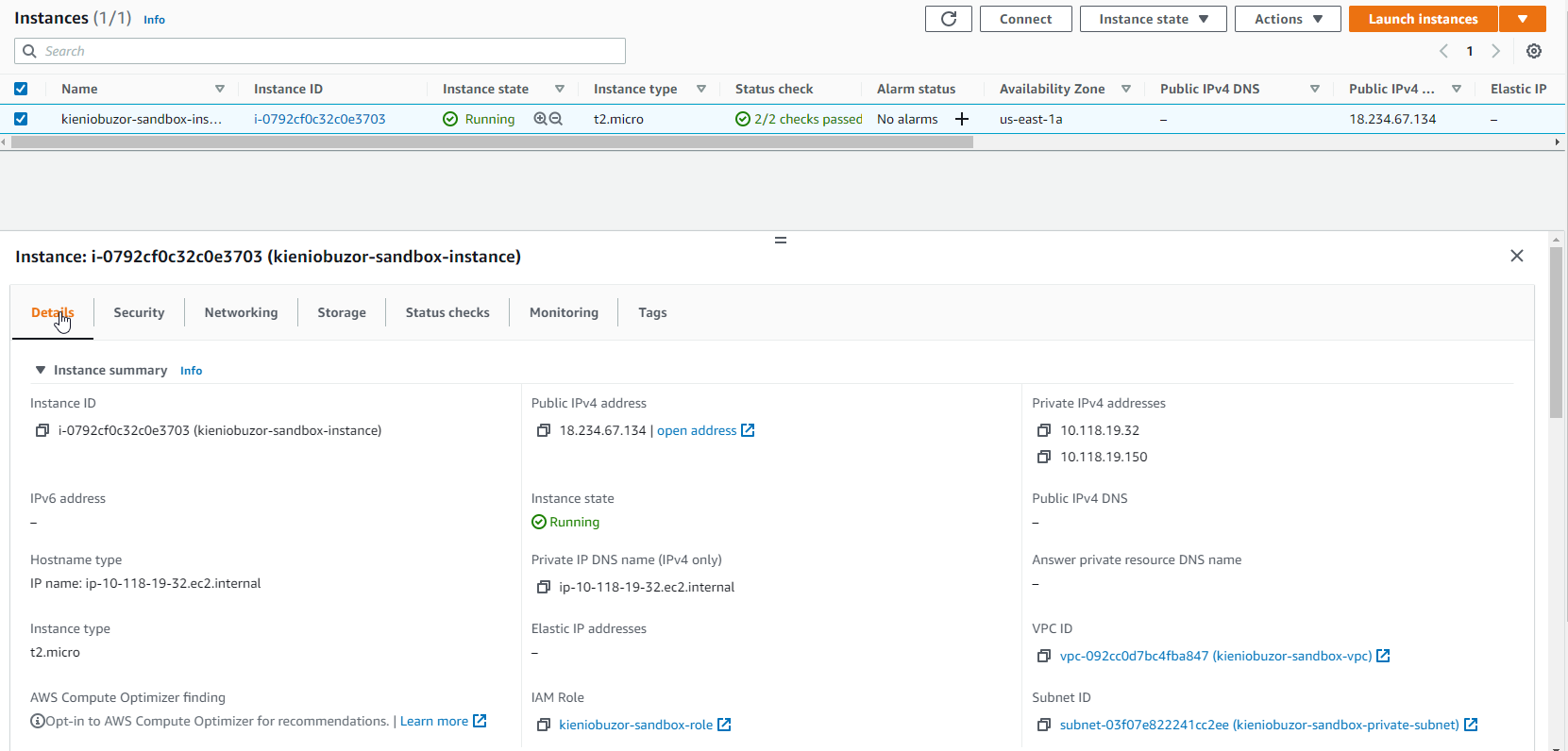
1. Nat gateway



1. Security Group



1. EC2



1. S3 Bucket

