Day 17 Internship Report

Terraform Infrastructure Deployment on Signiance AWS account & NAT gateway Debugging

Intern Name: Arsalan Sharief

Company: Signiance Technology

Role: DevOps Intern

Day: 17

Day 17 Terraform Infrastructure Deployment & NAT Debugging

Todays Task

Todays objective was to use my Terraform modules to provision complete infrastructure inside Signian's AWS account using standard Terraform workflows. The goal was to deploy a VPC, ECS Cluster, RDS Database, S3 storage, and all necessary networking components (NAT, route tables) to establish full frontend–backend connectivity.

1. Terraform Initialization Issue

While running `terraform init`, I encountered module loading errors preventing the initialization process.

Problem: Module source path could not be resolved.

Root Cause: I was using an incorrect relative path for the RDS module configuration.

```
# Incorrect Path
source = "../modules/rds"

# Corrected Path
source = "../../modules/rds"


Then Used:
```

terraform init

terraform plan

terraform apply

2. S3 Bucket Global Uniqueness Fix

During deployment, the creation of S3 buckets failed because the requested bucket names were already taken globally. S3 bucket names must be unique across all AWS accounts.

Solution: Implemented dynamic naming using the `random` provider.

```
resource "random_id" "bucket_suffix" {
  byte_length = 4
}

resource "aws_s3_bucket" "main" {
  bucket = "signian-app-${random_id.bucket_suffix.hex}"
}
```

This ensured every bucket name included a unique suffix, resolving the collision error.

3. Major Issue, NAT Gateway Failure

The most significant blocker of the day was the repeated failure of the AWS NAT Gateway resource.

Symptoms:

1. NAT Gateway entered "Pending" state.
2. Automatically transitioned to "Deleting" or "Failed".
3. Even manual creation via AWS Console failed immediately.

Investigation:

- Suspected Terraform state corruption; performed full cleanup (deleted `terraform.tfstate` and `.terraform/`).
- Verified Elastic IP allocation limits (Account was within limits).
- Checked tagging strategies. (User, Usage).

4. Temporary Workaround, NAT Instance

To bypass the NAT Gateway failure and restore internet access to the private subnets, I manually deployed a NAT Instance. This involves configuring a standard EC2 instance to route traffic.

Step-by-Step Implementation

1. Launch NAT EC2 Instance

- **AMI:** Amazon Linux 2 (Kernel 5.10)
- **Instance Type:** t3.micro
- **Subnet:** Public Subnet (Crucial for internet access)
- **Auto-assign Public IP:** Enabled
- **Security Group:**
  - Inbound: SSH (22) from My IP
  - Inbound: All Traffic from VPC CIDR (10.0.0.0/16)
  - Outbound: Allow All Traffic

2. Allocate & Attach Elastic IP

- Navigated to **EC2 → Elastic IPs**.
- Allocated a new EIP and associated it with the `nat-instance-dev`.

CRITICAL STEP: Disabled Source/Destination Check

3. Enable IP Forwarding

SSH into the NAT instance and configure the kernel to allow packet forwarding.

```
# Enable immediately
sudo sysctl -w net.ipv4.ip_forward=1

# Make permanent
echo "net.ipv4.ip_forward = 1" | sudo tee -a /etc/sysctl.conf
```

4. Configure IPTables (NAT Masquerade)

Configure `iptables` to mask the source IP of outgoing traffic, making it look like it came from the NAT instance.

```
sudo yum install -y iptables-services

# Add NAT rule
sudo iptables -t nat -A POSTROUTING -o eth0 -j MASQUERADE
```

```
# Save and enable
sudo service iptables save
sudo systemctl enable iptables
```

5. Updated Private Route Table

- **VPC → Route Tables**.
- Select the **Private Route Table**.
- Edit Routes: Add `0.0.0.0/0` targeting **Instance → nat-instance-dev**.

5. Testing NAT Connectivity

To verify the workaround, I launched a test EC2 instance in the private subnet (which has no public IP) and attempted to reach the internet.

```
# SSH to Private Instance
ssh -i key.pem ec2-user@10.0.2.50

# Test Connectivity
ping google.com
```

✔ Result: PING google.com (142.250.x.x) 56(84) bytes of data.

Internet connectivity successfully established via NAT Instance.

6. ECS Backend & Database Setup

With networking resolved, the ECS tasks could now pull images and connect to AWS services.

- **Database:** RDS instance provisioned and endpoint retrieved.
- **ECS Task Definition:** Updated environment variables with RDS host, user, and password.
- **Deployment:** Service started without crash loops.

✔ Backend API Service: **RUNNING**

✔ Database Connection: **ESTABLISHED**

7. Frontend ↔ Backend Integration

The final step was ensuring the frontend application could communicate with the backend API.

- Updated frontend configuration to point to the ECS Load Balancer / API domain.
- Verified CloudFront distribution settings for HTTPS.
- inspected browser Network tab to confirm 200 OK responses from API calls.

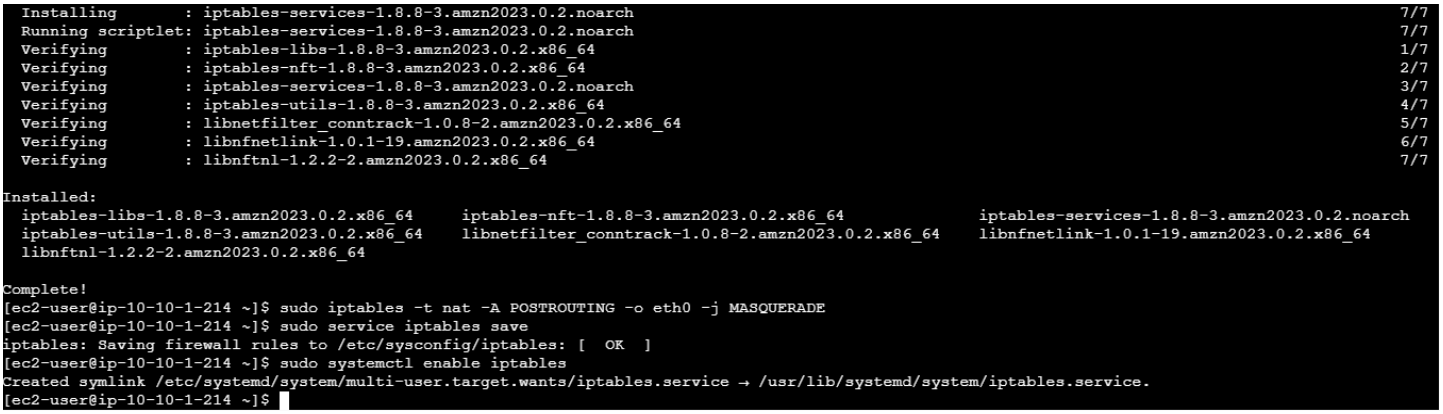
8. Key Learnings

Terraform State: State files referencing resources from previous deployments or different accounts can cause confusing failures. Always ensure a clean state when switching contexts.

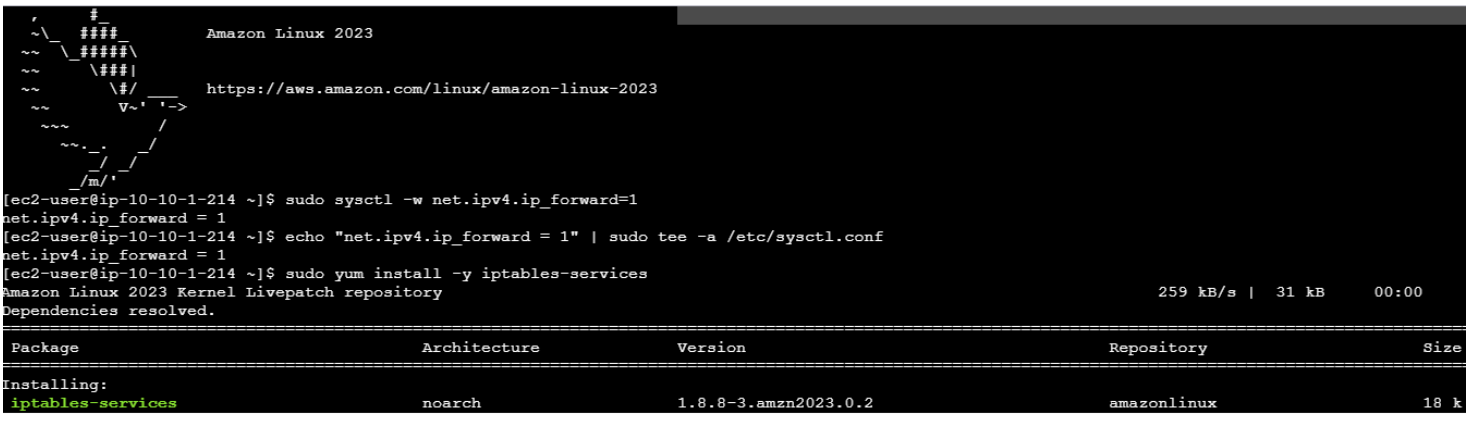
NAT Gateway vs. NAT Instance: While NAT Gateways are managed and preferred, they can fail due to account limits. Knowing how to manually build a NAT Instance using Linux primitives (`iptables`, `sysctl`) is a critical fallback skill.

Networking Debugging: Troubleshooting connectivity requires a checklist approach: Security Groups → Network ACLs → Route Tables → Source/Dest Checks → OS-level Firewalls.

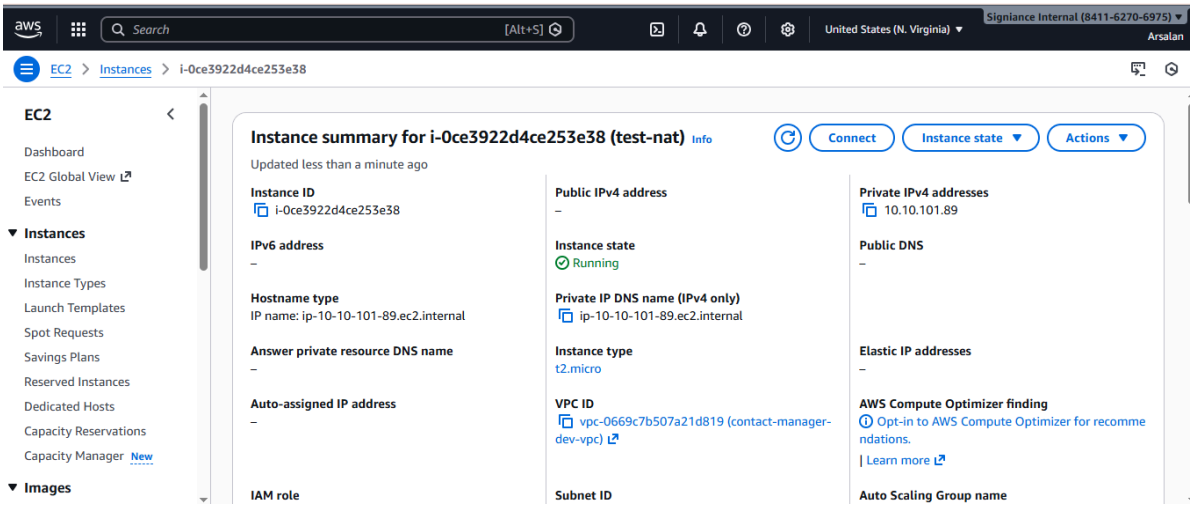
8. Screenshots



Terminal output showing successful installation and enabling of `iptables-services`



terminal where IP forwarding is enabled (`net.ipv4.ip_forward=1`) and `iptables-services` package installation is initiated via yum.



instance (`test-nat`) that i used to test NAT instance.

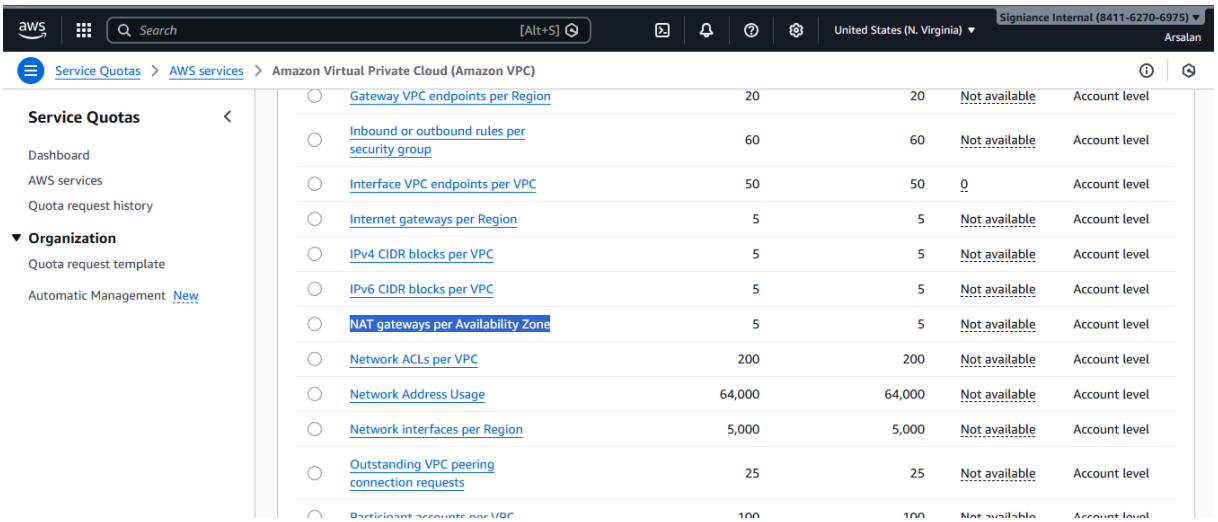
```
PING 8.8.8.8 (8.8.8.8) 56(84) bytes of data.
^C
--- 8.8.8.8 ping statistics ---
14 packets transmitted, 0 received, 100% packet loss, time 13518ms

[ec2-user@ip-10-10-101-89 ~]$ curl https://google.com
^C
[ec2-user@ip-10-10-101-89 ~]$ ping google.com
PING google.com (142.251.111.102) 56(84) bytes of data.
64 bytes from bk-in-f102.1e100.net (142.251.111.102): icmp_seq=1 ttl=108 time=3.03 ms
64 bytes from bk-in-f102.1e100.net (142.251.111.102): icmp_seq=2 ttl=108 time=3.72 ms
64 bytes from bk-in-f102.1e100.net (142.251.111.102): icmp_seq=3 ttl=108 time=3.08 ms
64 bytes from bk-in-f102.1e100.net (142.251.111.102): icmp_seq=4 ttl=108 time=3.07 ms
64 bytes from bk-in-f102.1e100.net (142.251.111.102): icmp_seq=5 ttl=108 time=3.36 ms
64 bytes from bk-in-f102.1e100.net (142.251.111.102): icmp_seq=6 ttl=108 time=3.39 ms
64 bytes from bk-in-f102.1e100.net (142.251.111.102): icmp_seq=7 ttl=108 time=2.95 ms
64 bytes from bk-in-f102.1e100.net (142.251.111.102): icmp_seq=8 ttl=108 time=3.21 ms
^C
--- google.com ping statistics ---
8 packets transmitted, 8 received, 0% packet loss, time 7012ms
rtt min/avg/max/mdev = 2.947/3.226/3.719/0.237 ms
[ec2-user@ip-10-10-101-89 ~]$ curl https://google.com
```

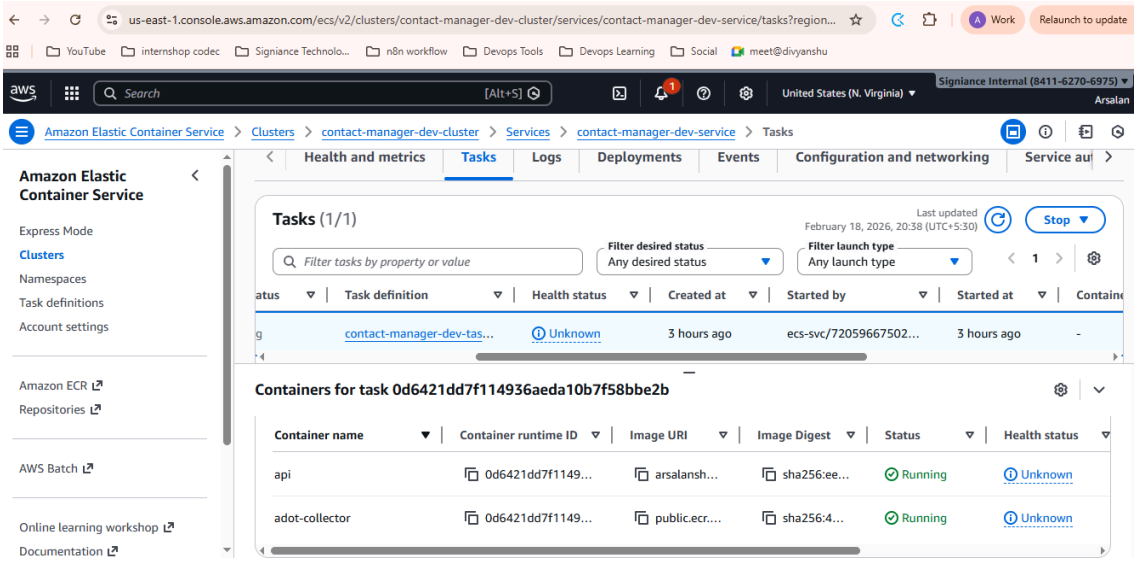
i-0ce3922d4ce253e38 (test-nat)

PrivateIPs: 10.10.101.89

Terminal showing initial failed ping to 8.8.8.8 followed by successful connectivity test to google.com, confirming outbound internet access after the outbound rule changed in sg of nat instance



I checked AWS Service Quotas page for Amazon VPC showing NAT gateway limits



AWS ECS console showing a running task in `contact-manager-dev-cluster` with containers (`api` and `adot-collector`) in running state

Prepared by: ArsalanSharief
DevOps Intern – Signiance Technology