

Final Project

Muhammad Asad ur Rehman Nadeem - 29456

This project demonstrates the design and deployment of a scalable, secure, and containerized infrastructure on AWS using Terraform. The objective was to automate the provisioning of cloud resources and orchestrate the deployment of a full-stack Dockerized web application, complete with database integration, load balancing, custom domain configuration, SSL encryption, and real-time business intelligence (BI) visualization.

By leveraging Infrastructure as Code (IaC) with Terraform, modular and reusable templates were developed to launch and manage critical AWS services, including EC2 instances (via Auto Scaling Groups), RDS (PostgreSQL and MySQL in private subnets), and Application Load Balancers with HTTPS support.

The frontend and backend applications were containerized using multi-stage Docker builds and deployed across EC2 instances behind an ALB. A separate EC2 instance was provisioned for the BI tool (Metabase), which was containerized using Docker and connected securely to the PostgreSQL RDS database. The BI dashboard was configured to reflect live updates, providing real-time insights into application data.

This document outlines the architecture, Terraform implementation, deployment flow, and key configurations that were used to meet the project requirements.

OBJECTIVE

- Build a production-ready infrastructure that includes:
- Auto Scaling EC2 instances with Nginx, Docker, and Node.js 20
- RDS databases (MySQL and PostgreSQL) in private subnets
- Load Balancer with HTTPS support
- Multi-stage Dockerized web applications (Frontend + Backend)
- BI Tool deployment (Metabase)
- Secure domain and SSL setup (ACM)
- SSH tunneling for database access
- Live dashboard visualization using Metabase

LINKS

React-App: <https://github.com/MARN121/reactapp-devops>

Terraform (IaC): https://github.com/MARN121/DevOps_Project

PROJECT STRUCTURE (MODULAR)

Project/

- main.tf
- outputs.tf
- providers.tf
- terraform.tfvars
- variables.tf
- README.md
- modules/
 - network/
 - main.tf
 - outputs.tf
 - variables.tf
 - security_groups/
 - main.tf
 - outputs.tf
 - variables.tf
 - target_group/
 - main.tf
 - outputs.tf
 - variables.tf
 - ec2/
 - main.tf
 - outputs.tf
 - variables.tf
 - ec2-bi/
 - main.tf
 - outputs.tf
 - variables.tf
 - rds/
 - main.tf
 - outputs.tf
 - variables.tf
 - alb/
 - main.tf
 - outputs.tf
 - variables.tf
 - alb-bi/
 - main.tf
 - outputs.tf
 - variables.tf
 - route53/
 - main.tf
 - outputs.tf
 - variables.tf
- userdata/
 - userdata-app.sh
 - userdata-bi.sh
- docker/
 - Dockerfile
- snapshot-and-destroy.sh

PROJECT WALKTHROUGH

1. EC2 Auto Scaling Group

- 3 EC2 instances were launched using a Launch Template.
- User data scripts (*userdata-app.sh* and *userdata-bi.sh*) installed:
 - Nginx
 - Docker
 - Node.js 20

Three instances served frontend and backend application. And one instance hosted Metabase for BI visualization.

Screenshots:

Code/Script:

```
main.tf X
modules > ec2 > main.tf > resource "aws_autoscaling_group" "app_asg" > launch_template > version
1 resource "aws_launch_template" "app_lt" {
2   name_prefix = "${var.project_name}-lt-"
3   image_id    = data.aws_ami.amazon_linux.id
4   instance_type = "t2.micro"
5   key_name    = var.key_name
6
7   user_data = base64encode(file("${path.module}/../userdata/userdata-app.sh"))
8
9   vpc_security_group_ids = [var.sg_id]
10
11   tag_specifications {
12     resource_type = "instance"
13     tags = {
14       Name = "${var.project_name}-ec2"
15     }
16   }
17 }
18
19 resource "aws_autoscaling_group" "app_asg" {
20   name = "${var.project_name}-asg"
21   max_size = 3
22   min_size = 3
23   desired_capacity = 3
24   vpc_zone_identifier = var.subnet_ids
25
26   launch_template {
27     id = aws_launch_template.app_lt.id
28     version = "Latest"
29   }
30
31   target_group_arns = [var.target_group_arn]
32
33   tag {
34     key = "Name"
35     value = "${var.project_name}-asg-instance"
36     propagate_at_launch = true
37   }
38
39   lifecycle {
40     create_before_destroy = true
41   }
42 }
43
44 data "aws_ami" "amazon_linux" {
45   most_recent = true
46   owners      = ["amazon"]
47
48   filter {
49     name = "Name"
50     values = ["amzn2-ami-hvm-*x86_64-gp2"]
51   }
52
53   filter {
54     name = "virtualization-type"
55     values = ["hvm"]
56   }
57 }
58
```

EC2 Main.tf (Application file)

```
main.tf X
modules > ec2-bi > main.tf > resource "aws_instance" "metabase_instance" > tags
1 resource "aws_instance" "metabase_instance" {
2   ami = data.aws_ami.amazon_linux.id
3   instance_type = "t2.micro"
4   subnet_id = var.subnet_id
5   vpc_security_group_ids = [var.sg_id]
6   key_name = var.key_name
7
8   user_data = base64encode(file("${path.module}/../userdata/userdata-bi.sh"))
9
10  tags = {
11    Name = "${var.project_name}-bi-instance"
12  }
13 }
14
15 data "aws_ami" "amazon_linux" {
16   most_recent = true
17   owners      = ["amazon"]
18
19   filter {
20     name = "Name"
21     values = ["amzn2-ami-hvm-*x86_64-gp2"]
22   }
23
24   filter {
25     name = "virtualization-type"
26     values = ["hvm"]
27   }
28 }
29
```

EC2-BI Main.tf (BI file)

```

$ userdata-app.sh X
userdata > $ userdata-app.sh
1 #!/bin/bash
2
3 # Update system
4 yum update -y
5
6 # Enable Amazon Linux extras for nginx and install required packages
7 amazon-linux-extras enable nginx1 -y
8 yum install -y nginx docker git
9
10 # Start and enable Docker service
11 systemctl enable docker --now
12 usermod -sG docker ec2-user
13
14 # Clone your monorepo (frontend + backend)
15 cd /home/ec2-user
16 git clone https://github.com/MA00121/reactapp-devops.git
17 chown -R ec2-user:ec2-user reactapp-devops
18 cd reactapp-devops
19
20 # --- FRONTEND SETUP ---
21 cd frontend
22 docker build -t frontend-app .
23 docker run -d -p 3000:80 --name frontend-app frontend-app
24 cd ..
25
26 # --- BACKEND SETUP ---
27 cd backend
28 docker build -t backend-app .
29 docker run -d -p 5000:5000 --name backend-app backend-app
30 cd ..
31
32 # --- NGINX CONFIGURATION ---
33 cat <<EOF > /etc/nginx/conf.d/frontend.conf
34 server {
35     listen 80;
36     server_name app.mendo.fun;
37
38     location / {
39         proxy_pass http://127.0.0.1:3000;
40         proxy_http_version 1.1;
41         proxy_set_header Upgrade $http_upgrade;
42         proxy_set_header Connection 'upgrade';
43         proxy_set_header Host $host;
44         proxy_cache_bypass $http_upgrade;
45     }
46 }
47 EOF
48
49 cat <<EOF > /etc/nginx/conf.d/backend.conf
50 server {
51     listen 80;
52     server_name appback.mendo.fun;
53
54     location / {
55         proxy_pass http://127.0.0.1:5000;
56         proxy_http_version 1.1;
57         proxy_set_header Upgrade $http_upgrade;
58         proxy_set_header Connection 'upgrade';
59         proxy_set_header Host $host;
60         proxy_cache_bypass $http_upgrade;
61     }
62 }
63 EOF
64
65 # Validate and restart NGINX
66 nginx -t && systemctl enable nginx --now && systemctl restart nginx
67

```

Userdata-app.sh (file)

```

$ userdata-bi.sh X
userdata > $ userdata-bi.sh
1 #!/bin/bash
2
3 yum update -y
4 amazon-linux-extras install docker -y
5 service docker start
6 usermod -sG docker ec2-user
7 docker run -d -p 3000:3000 --name metabase metabase/metabase
8

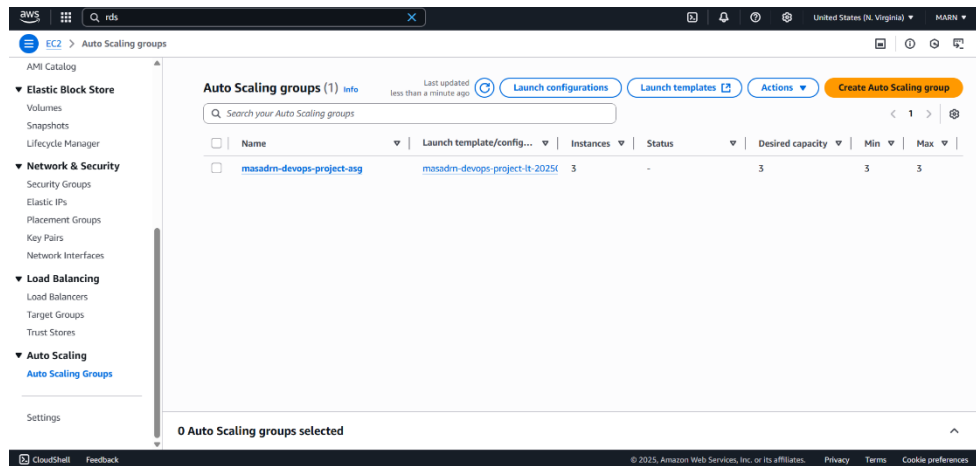
```

Userdata-bi.sh (file)

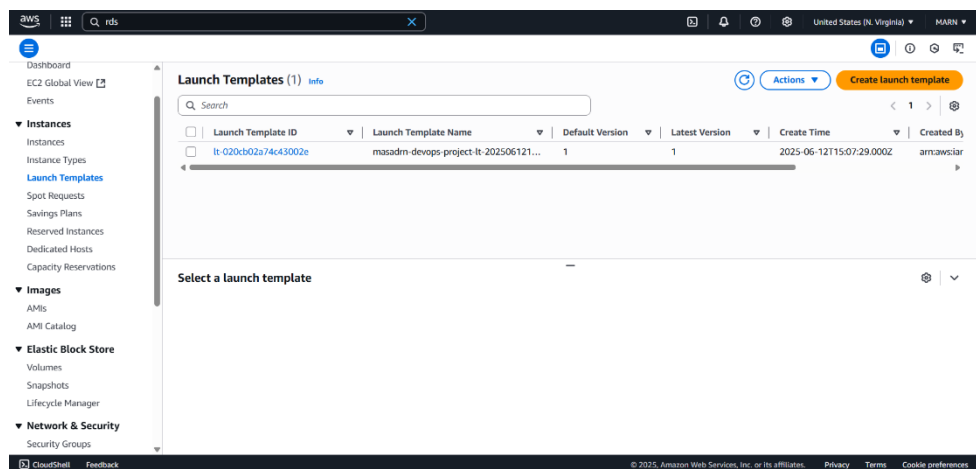
AWS:

Name	Instance ID	Instance state	Instance type	Status check	Alarm status
masadrm-devops-project-asg-instance	i-0ca1eb531d37db5e	Running	t2.micro	2/2 checks passed	View alarms
masadrm-devops-project-asg-instance	i-07824f7a2f9a79c	Running	t2.micro	2/2 checks passed	View alarms
masadrm-devops-project-asg-instance	i-01b823b7f491c23e9	Running	t2.micro	2/2 checks passed	View alarms
masadrm-devops-project-bi-instance	i-0bcad41b61797ef9a	Running	t2.micro	2/2 checks passed	View alarms
masadrm-devops-project-nat-instance	i-04e11fa1413b1ed9c	Running	t2.micro	2/2 checks passed	View alarms

Instances (AWS Console)



Auto Scaling Group (AWS Console)



Launch Template (AWS Console)

2. RDS Instances

- 1 MySQL and 1 PostgreSQL RDS instance created.
- Both RDS instances were launched in private subnets.
- Connected securely via SSH tunneling through EC2.
- Subnet groups and parameter groups configured via Terraform.

Screenshot:

Code/Script:

```
main.tf
modules > rds > main.tf > resource "aws_db_instance" "postgres" > publicly_accessible

1 # Subnet group for RDS
2 resource "aws_db_subnet_group" "rds_subnet_group" {
3   name = "${var.project_name}-rds-subnet-group"
4   subnet_ids = var.subnet_ids
5   tags = {
6     Name = "${var.project_name}-rds-subnet-group"
7   }
8 }
9
10 # MySQL Instance
11 resource "aws_db_instance" "mysql" {
12   identifier = "${var.project_name}-mysql"
13   engine = "mysql"
14   instance_class = "db.t3.micro"
15   allocated_storage = 20
16   username = var.db_username
17   password = var.db_password
18   skip_final_snapshot = true
19   publicly_accessible = false
20   db_subnet_group_name = aws_db_subnet_group.rds_subnet_group.name
21   vpc_security_group_ids = [var.rds_sg_id]
22 }
23
24 # PostgreSQL Instance
25 resource "aws_db_instance" "postgres" {
26   identifier = "${var.project_name}-postgres"
27   engine = "postgres"
28   instance_class = "db.t3.micro"
29   allocated_storage = 20
30   username = var.db_username
31   password = var.db_password
32   skip_final_snapshot = true
33   publicly_accessible = false
34   db_subnet_group_name = aws_db_subnet_group.rds_subnet_group.name
35   vpc_security_group_ids = [var.rds_sg_id]
36 }
37
```

RDS Main.tf (file)

AWS:

DB identifier	Status	Role	Engine	Region	Size
masadm-devops-project-mysql	Available	Instance	MySQL Co...	us-east-1a	db.t3.micro
masadm-devops-project-postgres	Available	Instance	PostgreSQL	us-east-1b	db.t3.micro

RDS Databases (AWS Console)

Name	Description	Status	VPC
default	default	Complete	vpc-075a56389b46d1e69
default-vpc-075a56389b46d1e69	Created from the RDS Management Console	Complete	vpc-075a56389b46d1e69
masadm-devops-project-rds-subnet-group	Managed by Terraform	Complete	vpc-065b3e9641a73b158

RDS Subnet-Groups (AWS Console)

3. Security Groups

- EC2 Security Group:
 - Ingress: Port 22 (SSH), 80 (HTTP), 443 (HTTPS)
- RDS Security Group:
 - Ingress: Ports 3306 (MySQL) and 5432 (PostgreSQL)
 - Access allowed only from EC2 security group
- ALB Security Group:
 - Ingress: Port 80 and 443 open to all
- Egress open to all (0.0.0.0/0) for all SGs.

Screenshot:

Code/Script:

```
modules > security_groups > main.tf > ...
1 # EC2 Security Group
2 resource "aws_security_group" "ec2_sg" {
3   name = "${var.project_name}-ec2-sg"
4   description = "Allow SSH, HTTP, HTTPS"
5   vpc_id = var.vpc_id
6
7   ingress {
8     description = "SSH"
9     from_port = 22
10    to_port = 22
11    protocol = "tcp"
12    cidr_blocks = ["0.0.0.0/0"]
13  }
14
15  ingress {
16    description = "HTTP"
17    from_port = 80
18    to_port = 80
19    protocol = "tcp"
20    cidr_blocks = ["0.0.0.0/0"]
21  }
22
23  ingress {
24    description = "HTTPS"
25    from_port = 443
26    to_port = 443
27    protocol = "tcp"
28    cidr_blocks = ["0.0.0.0/0"]
29  }
30
31  egress {
32    from_port = 0
33    to_port = 0
34    protocol = "-1"
35    cidr_blocks = ["0.0.0.0/0"]
36  }
37
38  tags = {
39    Name = "${var.project_name}-ec2-sg"
40  }
41 }
42
43 # ALB Security Group
44 resource "aws_security_group" "alb_sg" {
45   name = "${var.project_name}-alb-sg"
46   description = "Allow inbound from internet on ports 80 and 443"
47   vpc_id = var.vpc_id
48
49   ingress {
50     from_port = 80
51     to_port = 80
52     protocol = "tcp"
53     cidr_blocks = ["0.0.0.0/0"]
54   }
55
56   ingress {
57     from_port = 443
58     to_port = 443
59     protocol = "tcp"
60     cidr_blocks = ["0.0.0.0/0"]
61   }
62
63   egress {
64     from_port = 0
65     to_port = 0
66     protocol = "-1"
67     cidr_blocks = ["0.0.0.0/0"]
68   }
69
70   tags = {
71     Name = "${var.project_name}-alb-sg"
72   }
73 }
74
75 # RDS Security Group
76 resource "aws_security_group" "rds_sg" {
77   name = "${var.project_name}-rds-sg"
78   description = "Allow MySQL and PostgreSQL from EC2"
79   vpc_id = var.vpc_id
80
81   # MySQL
82   ingress {
83     from_port = 3306
84     to_port = 3306
85     protocol = "tcp"
86     security_groups = [aws_security_group.ec2_sg.id]
87   }
88
89   # PostgreSQL
90   ingress {
91     from_port = 5432
92     to_port = 5432
93     protocol = "tcp"
94     security_groups = [aws_security_group.ec2_sg.id]
95   }
96
97   egress {
98     from_port = 0
99     to_port = 0
100    protocol = "-1"
101    cidr_blocks = ["0.0.0.0/0"]
102  }
103
104  tags = {
105    Name = "${var.project_name}-rds-sg"
106  }
107 }
108
```



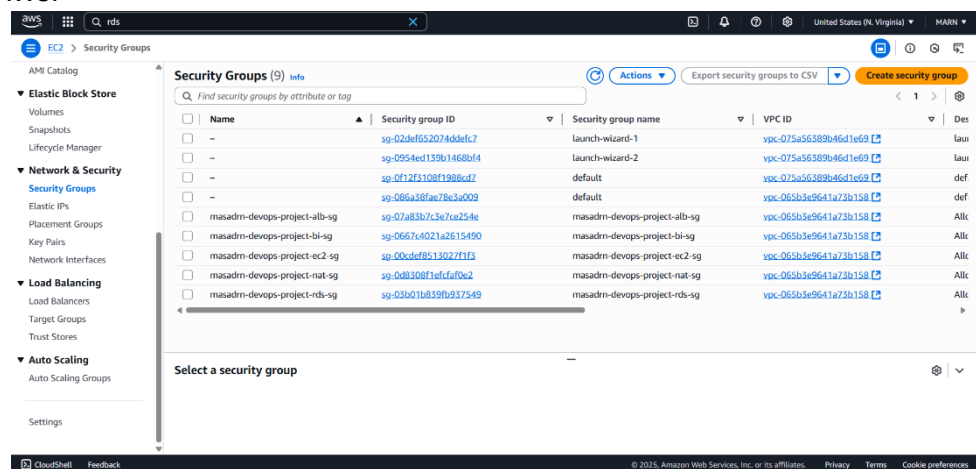
```

109 # BI-EC2 Security Group
110 resource "aws_security_group" "bi_sg" {
111   name               = "${var.project_name}-bi-sg"
112   description        = "Allow access for Metabase"
113   vpc_id             = var.vpc_id
114
115   ingress {
116     from_port = 3000
117     to_port   = 3000
118     protocol  = "tcp"
119     cidr_blocks = ["0.0.0.0/0"]
120   }
121
122   # Optional: Allow SSH
123   ingress {
124     from_port = 22
125     to_port   = 22
126     protocol  = "tcp"
127     cidr_blocks = ["0.0.0.0/0"]
128   }
129
130   egress {
131     from_port = 0
132     to_port   = 0
133     protocol  = "-1"
134     cidr_blocks = ["0.0.0.0/0"]
135   }
136
137   tags = {
138     Name = "${var.project_name}-bi-sg"
139   }
140 }
141

```

Security Group Main.tf (file)

AWS:



Security-Groups (AWS Console)

4. Load Balancer (ALB)

- Application Load Balancer deployed via Terraform.
- HTTPS enforced:
 - HTTP (port 80) redirected to HTTPS (port 443)
- Listener rules forward traffic to target groups based on subdomain.
- HTTPS configured with ACM certificate for secure access.

Screenshot:

Code/Script:

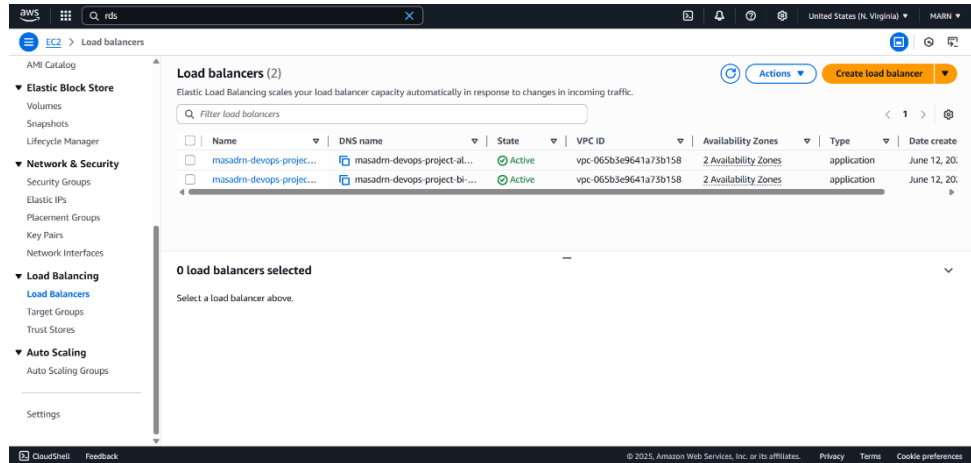
```
main.tf x
modules> alb> main.tf> resource "aws_lb_listener" "https" > default_action > target_group_arn
1 resource "aws_lb" "app_alb" {
2   name = "${var.project_name}-alb"
3   load_balancer_type = "application"
4   subnets = var.subnet_ids
5   security_groups = [var.alb_sg_id]
6
7   enable_deletion_protection = false
8   idle_timeout = 60
9
10  tags = {
11    Name = "${var.project_name}-alb"
12  }
13
14
15  # HTTP -> HTTPS Redirect
16  resource "aws_lb_listener" "http" {
17    port = 80
18    protocol = "HTTP"
19    load_balancer_arn = aws_lb.app_alb.arn
20
21    default_action {
22      type = "redirect"
23      redirect {
24        port = "443"
25        protocol = "HTTPS"
26        status_code = "HTTP_301"
27      }
28    }
29  }
30
31  # HTTPS Listener
32  resource "aws_lb_listener" "https" {
33    port = 443
34    protocol = "HTTPS"
35    # ssl_policy = "ELBSecurityPolicy-TLS-1-2-Ext-2018-06"
36    certificate_arn = var.acm_certificate_arn
37    load_balancer_arn = aws_lb.app_alb.arn
38
39    default_action {
40      type = "forward"
41      target_group_arn = var.target_group_arn
42    }
43  }
44
```

ALB Main.tf (file)

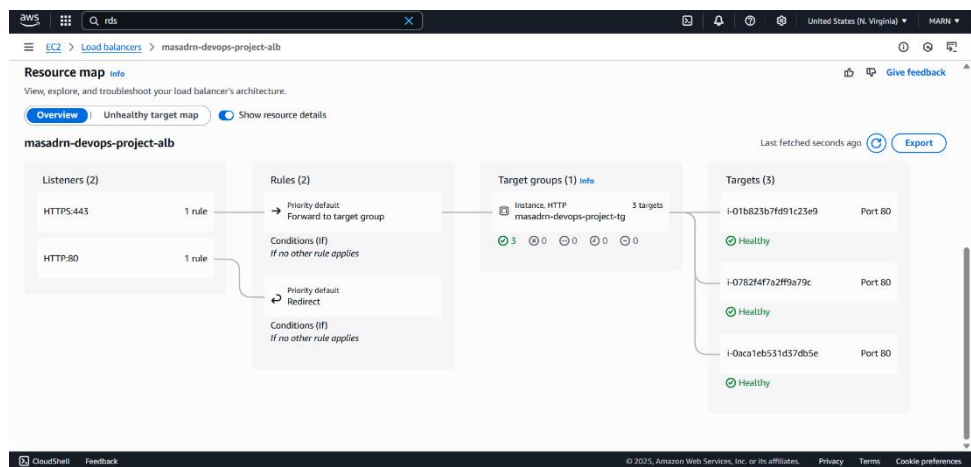
```
main.tf x
modules> alb-bi> main.tf> resource "aws_lb" "bi_alb" > subnets
1 resource "aws_lb" "bi_alb" {
2   name = "${var.project_name}-bi-alb"
3   load_balancer_type = "application"
4   subnets = var.subnet_ids
5   security_groups = [var.alb_sg_id]
6   enable_deletion_protection = false
7   idle_timeout = 60
8
9   tags = {
10    Name = "${var.project_name}-bi-alb"
11  }
12
13
14  resource "aws_lb_target_group" "bi_tg" {
15    name = "${var.project_name}-bi-tg"
16    port = 3000
17    protocol = "HTTP"
18    vpc_id = var.vpc_id
19    target_type = "instance"
20
21    health_check {
22      path = "/"
23      protocol = "HTTP"
24      port = "3000"
25      interval = 30
26      timeout = 5
27      healthy_threshold = 2
28      unhealthy_threshold = 2
29    }
30
31    tags = {
32      Name = "${var.project_name}-bi-tg"
33    }
34  }
35
36  resource "aws_lb_target_group_attachment" "bi_attach" {
37    target_group_arn = aws_lb_target_group.bi_tg.arn
38    target_id = var.instance_id
39    port = 3000
40  }
41
42  resource "aws_lb_listener" "http" {
43    load_balancer_arn = aws_lb.bi_alb.arn
44    port = 80
45    protocol = "HTTP"
46
47    default_action {
48      type = "redirect"
49      redirect {
50        port = "443"
51        protocol = "HTTPS"
52        status_code = "HTTP_301"
53      }
54    }
55  }
56
57  resource "aws_lb_listener" "https" {
58    load_balancer_arn = aws_lb.bi_alb.arn
59    port = 443
60    protocol = "HTTPS"
61    ssl_policy = "ELBSecurityPolicy-2016-08"
62    certificate_arn = var.acm_certificate_arn
63
64    default_action {
65      type = "forward"
66      target_group_arn = aws_lb_target_group.bi_tg.arn
67    }
68  }
69
```

ALB-BI Main.tf (file)

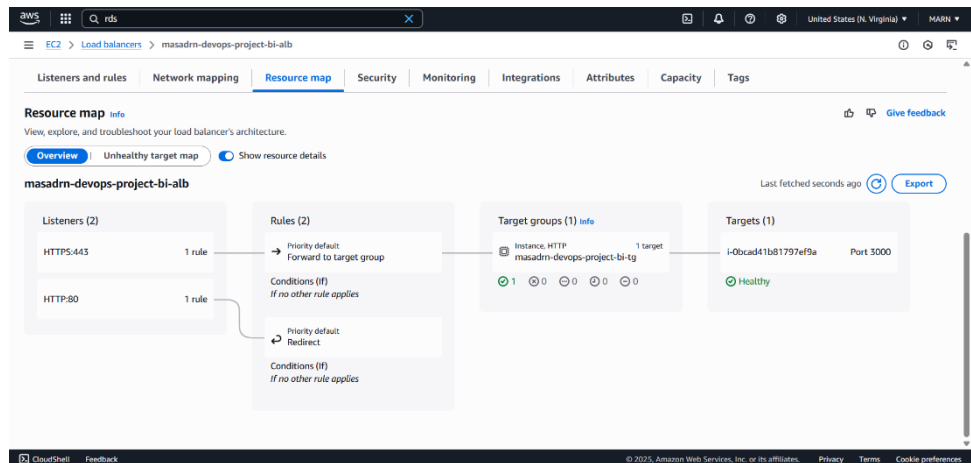
AWS:



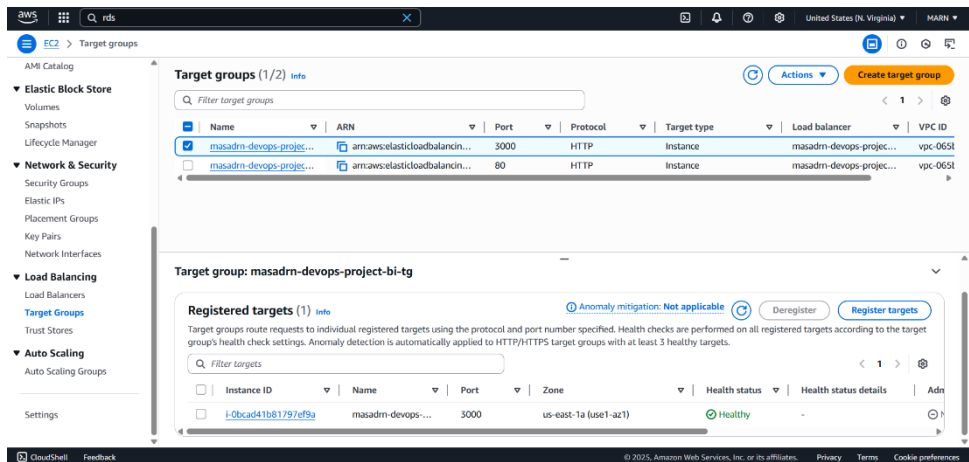
Load Balancers (AWS Console)



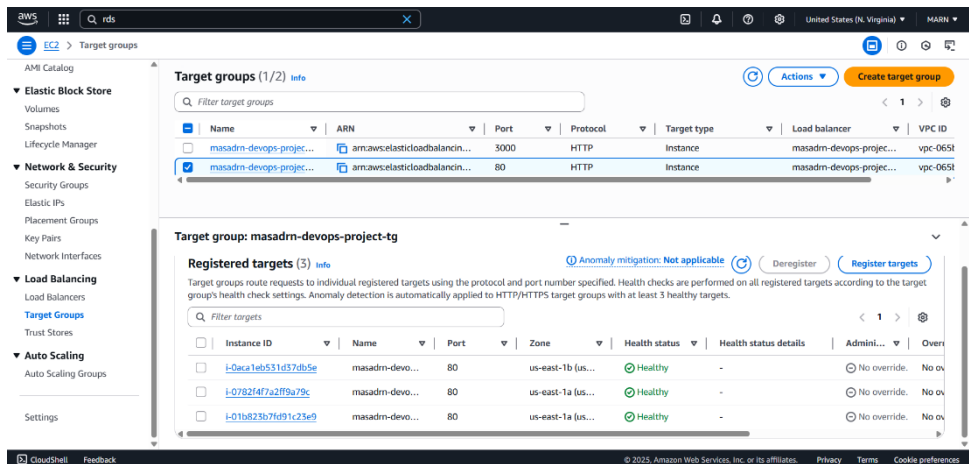
Application Load Balancer – Resource Map (AWS Console)



BI Load Balancer – Resource Map (AWS Console)



BI Target Group, with Instance Health (AWS Console)



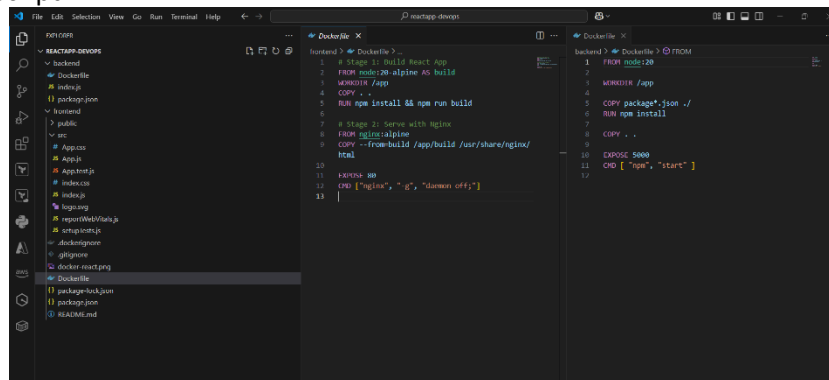
Application Target Group, with Instance Health (AWS Console)

5. Dockerized App Deployment

- Used multi-stage Dockerfiles for React frontend and Nomral for Node backend (Comparison).
- Deployed both apps on same EC2 using Docker:
 - Frontend: app.nendo.fun (port 3000 → NGINX → 80)
 - Backend: appback.nendo.fun (port 5000 → NGINX → 80)
- NGINX set up as reverse proxy for each app.

Screenshot:

Code/Script:



React Application Frontend (Multi-Stage) and Backend (Normal) Dockerfile (file)

AWS:



```

AWS [Alt+S]
Amazon Linux 2
AL2 End of life is 2026-06-30.
A newer version of Amazon Linux is available!
Amazon Linux 2023, GA and supported until 2028-03-15.
https://aws.amazon.com/linux/amazon-linux-2023/

[ec2-user@ip-10-0-2-68 ~]$ docker ps
CONTAINER ID   IMAGE          COMMAND                  CREATED        STATUS        PORTS                               NAMES
6f7b92d31de   backend-app   "docker-entrypoint.sh"   20 minutes ago Up 20 minutes 0.0.0.0:5000->5000/tcp, :::5000->5000/tcp   backend-app
7f6d3a1a16d   frontend-app  "/docker-entrypoint..." 20 minutes ago Up 20 minutes 0.0.0.0:3000->3000/tcp, :::3000->3000/tcp   frontend-app

[ec2-user@ip-10-0-2-68 ~]$ docker images
REPOSITORY    TAG       IMAGE ID       CREATED        SIZE
backend-app    latest   6a0fe99d1800   20 minutes ago 1.11GB
frontend-app   latest   a31379ca7522   20 minutes ago 49.1MB

[ec2-user@ip-10-0-2-68 ~]$
```

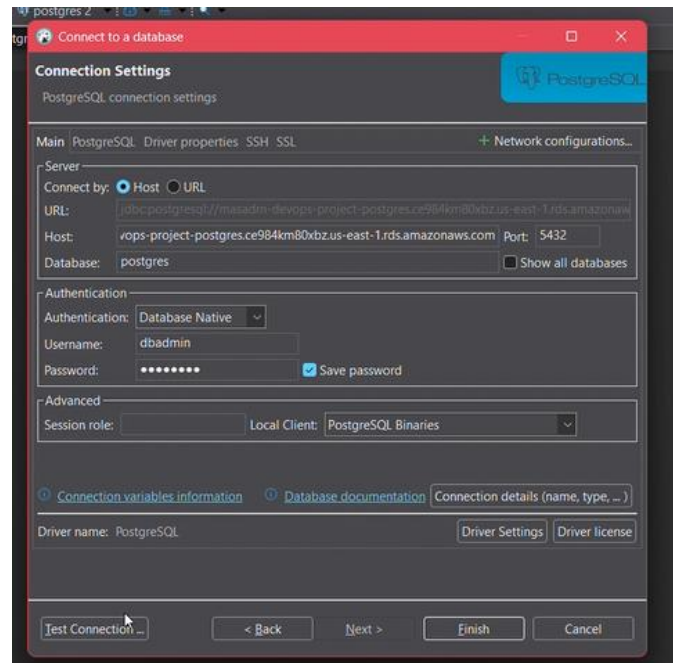
EC2 Instance SSH terminal via Instance Connect (AWS Console)
Docker Containers Running and Image Size Comparison

6. Database Access & Dummy Data

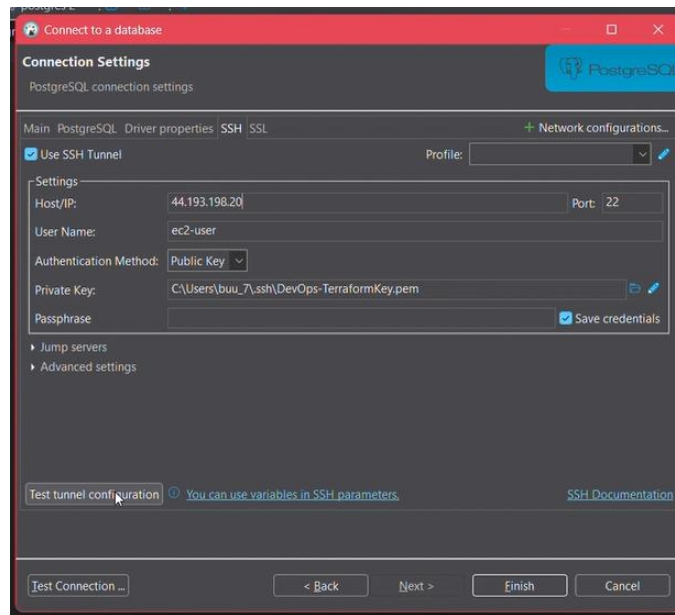
- Used DBeaver to connect to RDS PostgreSQL via SSH tunnel through EC2.
- Dummy data added to a sales table:
 - Console sales for PlayStation, Xbox, Steam Deck, etc.
- Verified live updates in Metabase dashboard after inserting records.

Screenshot:

DBeaver:



Connecting to our deployed Database on DBeaver by setting up Main and SSH settings.
Host will be our DB Endpoint provided by AWS RDS
Username and Password is set by us in TF-variables.

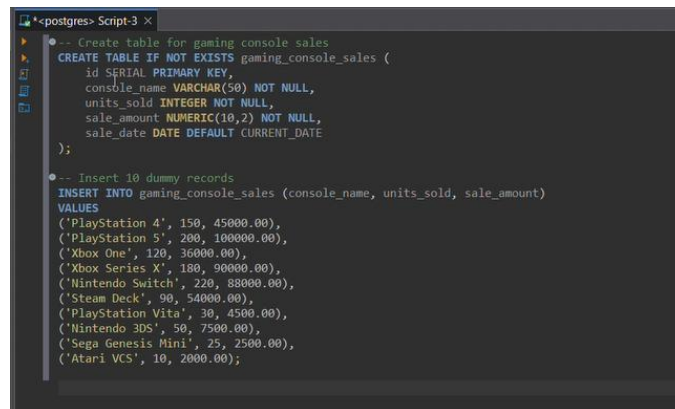


SSH tunneling because RDS are not deployed publicly.

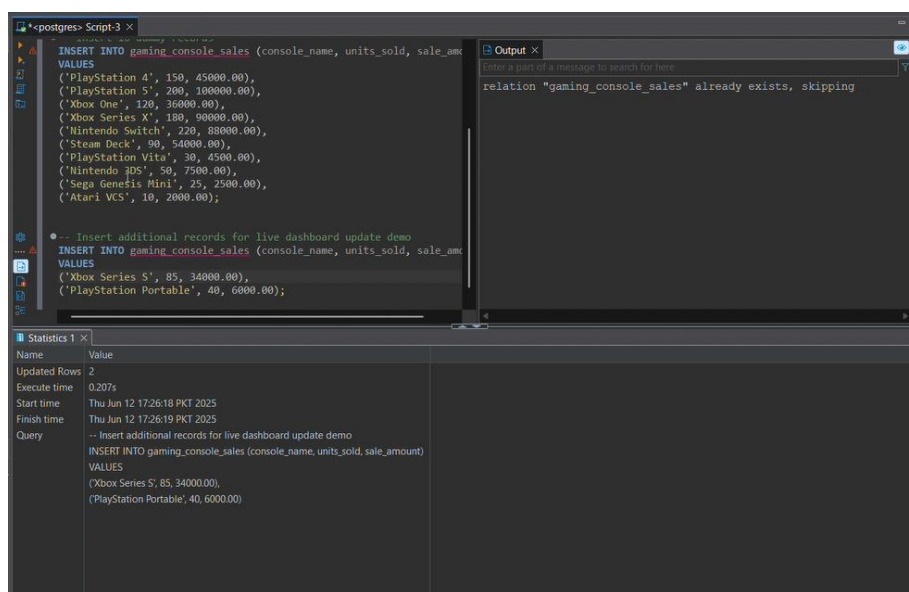
Host is EC2 Instance public IP on which RDS are deployed

Username will be ec2-user (default user of instance)

For authentication we will use key .pem file of key-pair we are using in Terraform

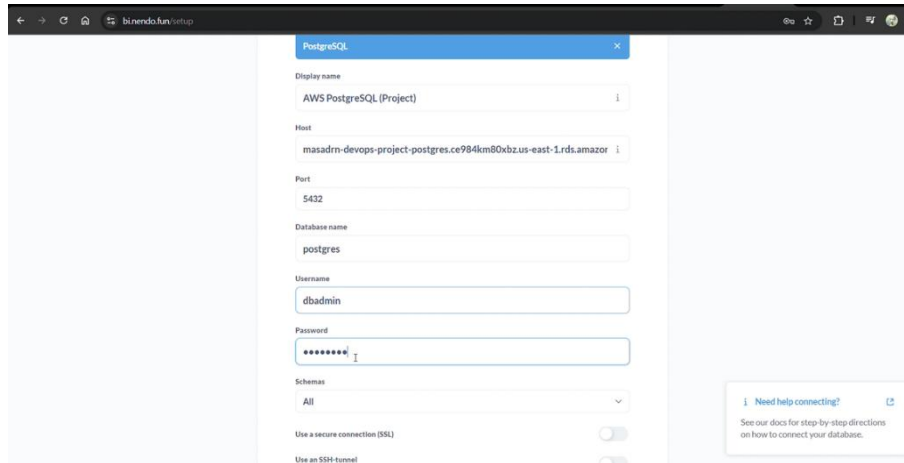


Dummy Data Script to create table and add dummy records.



Dummy Data Script Insert with Showing Result Successful and Outputting Table already exists on running same script again.

Metabase:

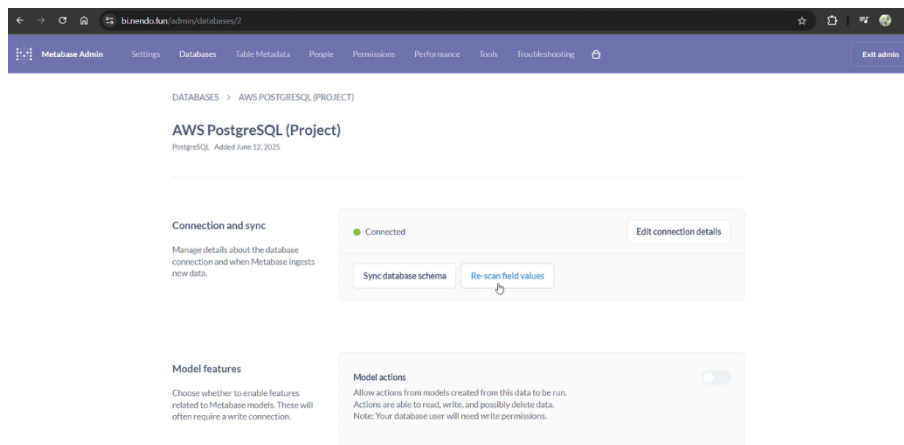


The screenshot shows the Metabase PostgreSQL connection setup form. The fields are as follows:

- Display name: AWS PostgreSQL (Project)
- Host: masadm-devops-project-postgres.ce984km80xbz.us-east-1.rds.amazonaws.com
- Port: 5432
- Database name: postgres
- Username: dbadmin
- Password: [masked]
- Schemas: All
- Use a secure connection (SSL): [checked]
- Use an SSH tunnel: [unchecked]

A help box on the right says: "Need help connecting? See our docs for step-by-step directions on how to connect your database."

Using DB endpoint we will setup Database on Metabase
Username and Password is set by us in TF-variables.



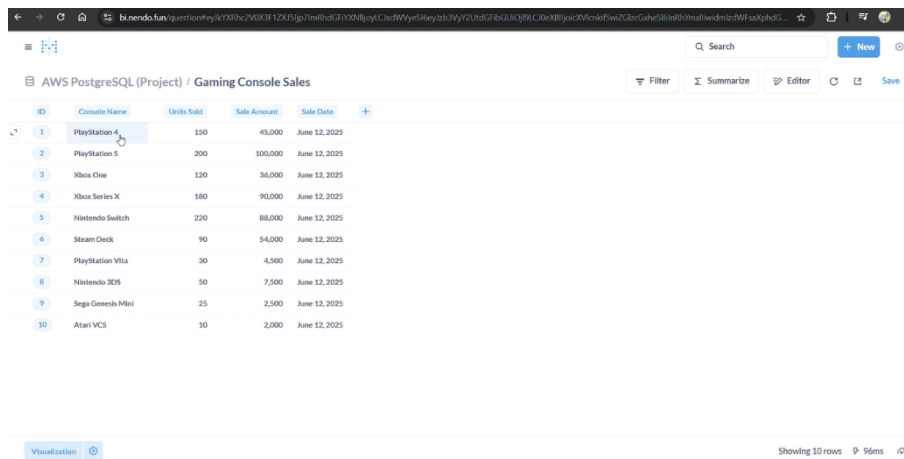
The screenshot shows the Metabase Admin page for the AWS PostgreSQL (Project) database. The page has a navigation bar with links: Metabase Admin, Settings, Databases, Table Metadata, People, Permissions, Performance, Tools, Troubleshooting, and Exit admin.

The main content area is titled "AWS PostgreSQL (Project)" and "PostgreSQL, Added June 12, 2025".

Under "Connection and sync", there is a "Connected" status and an "Edit connection details" button. Below this are two buttons: "Sync database schema" and "Re-scan field values".

Under "Model features", there is a toggle switch for "Model actions". The description says: "Allow actions from models created from this data to be run. Actions are able to read, write, and possibly delete data. Note: Your database user will need write permissions."

Sync and Scan Fields after adding updating DB First Time

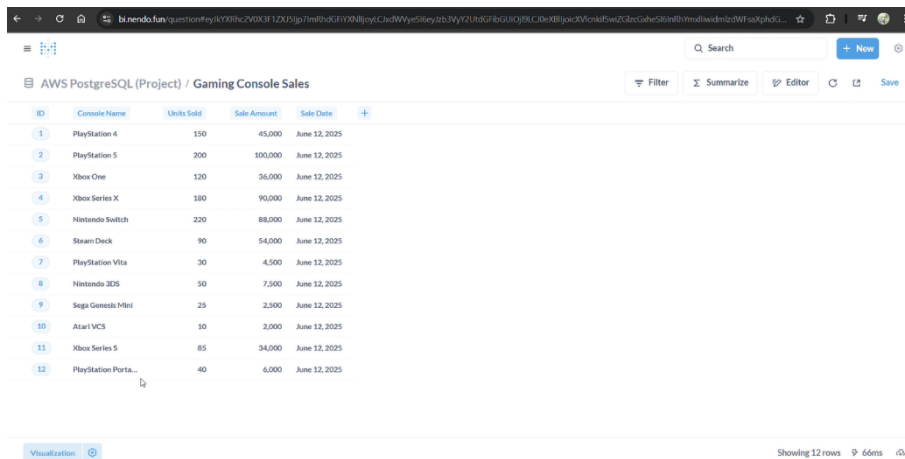


The screenshot shows the Metabase table view for "AWS PostgreSQL (Project) / Gaming Console Sales". The table has the following columns: ID, Console Name, Units Sold, Sale Amount, and Sale Date. The data is as follows:

ID	Console Name	Units Sold	Sale Amount	Sale Date
1	PlayStation 4	150	45,000	June 12, 2025
2	PlayStation 5	200	100,000	June 12, 2025
3	Xbox One	120	36,000	June 12, 2025
4	Xbox Series X	180	90,000	June 12, 2025
5	Nintendo Switch	220	88,000	June 12, 2025
6	Steam Deck	90	54,000	June 12, 2025
7	PlayStation Vita	30	4,500	June 12, 2025
8	Nintendo 3DS	50	7,500	June 12, 2025
9	Sega Genesis Mini	25	2,500	June 12, 2025
10	Atari VCS	10	2,000	June 12, 2025

At the bottom, there is a "Visualization" button and a status bar showing "Showing 10 rows" and "96ms".

Table Before Update



The screenshot shows the AWS PostgreSQL console interface. At the top, there's a search bar and a '+ New' button. Below that, the breadcrumb path is 'AWS PostgreSQL (Project) / Gaming Console Sales'. There are buttons for 'Filter', 'Summarize', 'Editor', and 'Save'. The main area displays a table with 12 rows and 5 columns: ID, Console Name, Units Sold, Sale Amount, and Sale Date. The data includes various gaming consoles like PlayStation 4, PlayStation 5, Xbox One, Xbox Series X, Nintendo Switch, Steam Deck, PlayStation Vita, Nintendo 3DS, Sega Genesis Mini, Atari VCS, Xbox Series S, and PlayStation Portables. At the bottom, there's a 'Visualization' button and a status bar indicating 'Showing 12 rows' and a refresh icon.

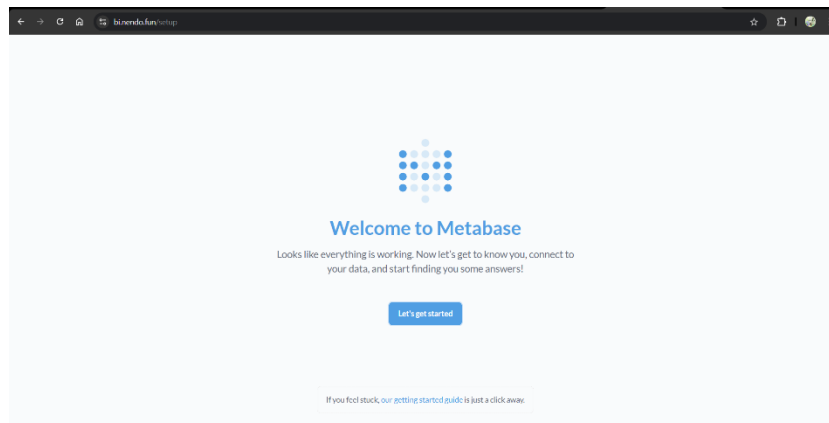
ID	Console Name	Units Sold	Sale Amount	Sale Date
1	PlayStation 4	150	45,000	June 12, 2025
2	PlayStation 5	200	100,000	June 12, 2025
3	Xbox One	120	36,000	June 12, 2025
4	Xbox Series X	180	90,000	June 12, 2025
5	Nintendo Switch	220	88,000	June 12, 2025
6	Steam Deck	90	54,000	June 12, 2025
7	PlayStation Vita	30	4,500	June 12, 2025
8	Nintendo 3DS	50	7,500	June 12, 2025
9	Sega Genesis Mini	25	2,500	June 12, 2025
10	Atari VCS	10	2,000	June 12, 2025
11	Xbox Series S	85	34,000	June 12, 2025
12	PlayStation Portables	40	6,000	June 12, 2025

Table After Update

7. BI Tool Deployment (Metabase)

- Deployed Metabase using Docker on a dedicated EC2 instance.
- Available via bi.nendo.fun, secured with HTTPS.
- Connected to PostgreSQL RDS to visualize sales data.
- Dashboard created to reflect real-time data.

Screenshot:



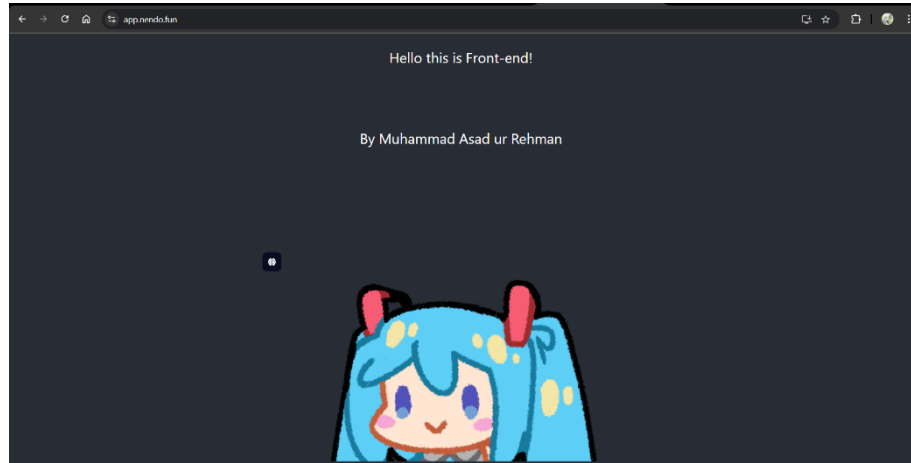
Metabase deployed on HTTPS://bi.nendo.fun (secure)

Metabase Deployment in part 1
Updates and data display in part 6

8. Domain & SSL Configuration

- Hosted zone: nendo.fun managed in Route 53.
- Subdomains routed using ALB DNS:
 - app.nendo.fun → Frontend
 - appback.nendo.fun → Backend
 - bi.nendo.fun → BI (Metabase)
- ACM certificate issued and validated for:
 - *.nendo.fun
 - nendo.fun

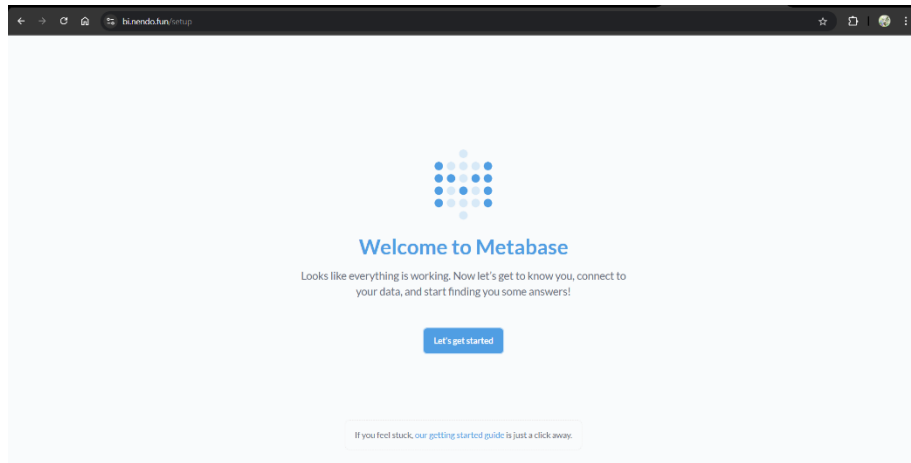
Screenshot:



Application Front-End (<https://app.nendo.fun>)



Application Back-End (<https://appback.nendo.fun>)



Metabase (<https://bi.nendo.fun>)

Code/Script:

```
main.tf
modules > route53 > main.tf > resource "aws_route53_record" "app"
1 # resource "aws_route53_zone" "main" {
2 #   name = var.domain_name # "nendo.fun"
3 # } # New Hosted Zone using TF
4
5 resource "aws_route53_record" "app" {
6   zone_id = var.hosted_zone_id
7   name = "app.${var.domain_name}" # app.nendo.fun
8   type = "A"
9
10  alias {
11    name = var.alb_dns_name
12    zone_id = var.alb_zone_id # us-east-1
13    evaluate_target_health = true
14  }
15 }
16
17 resource "aws_route53_record" "appback" {
18   zone_id = var.hosted_zone_id
19   name = "appback.${var.domain_name}"
20   type = "A"
21
22   alias {
23     name = var.alb_dns_name
24     zone_id = var.alb_zone_id
25     evaluate_target_health = true
26   }
27 }
28
29 # A Record for BI Tool - EC2 instance (optional for Metabase)
30 resource "aws_route53_record" "bi" {
31   zone_id = var.hosted_zone_id
32   name = "bi.${var.domain_name}" # bi.nendo.fun
33   type = "A"
34
35   alias {
36     name = var.bi_alb_dns_name
37     zone_id = var.bi_alb_zone_id
38     evaluate_target_health = true
39   }
40 }
41
```

RDS Main.tf (file)

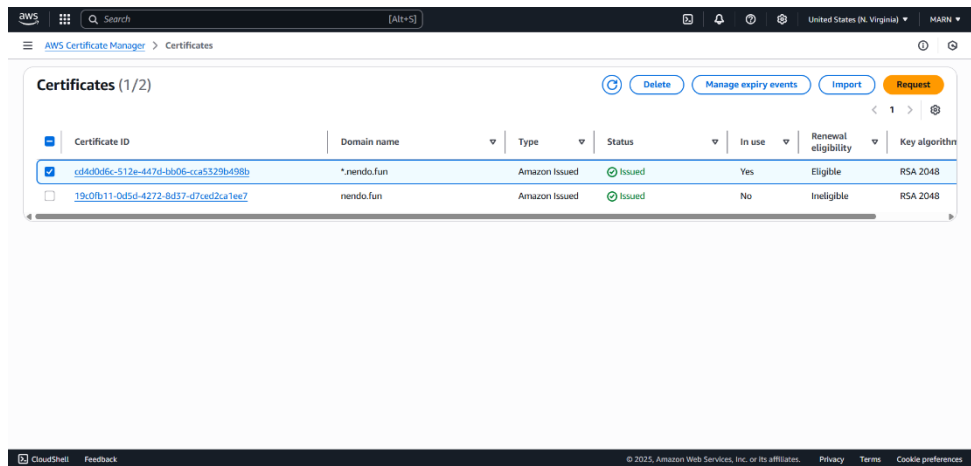
AWS:

The screenshot shows the AWS Management Console interface for Route 53. The left sidebar contains navigation links for Route 53, including Dashboard, Hosted zones, Health checks, Profiles, IP-based routing, Traffic flow, Domains, and Resolver. The main content area is titled 'Hosted zones (1)' and shows a table with one entry: 'nendo.fun' (Public, Route 53, 8 records, description '-').

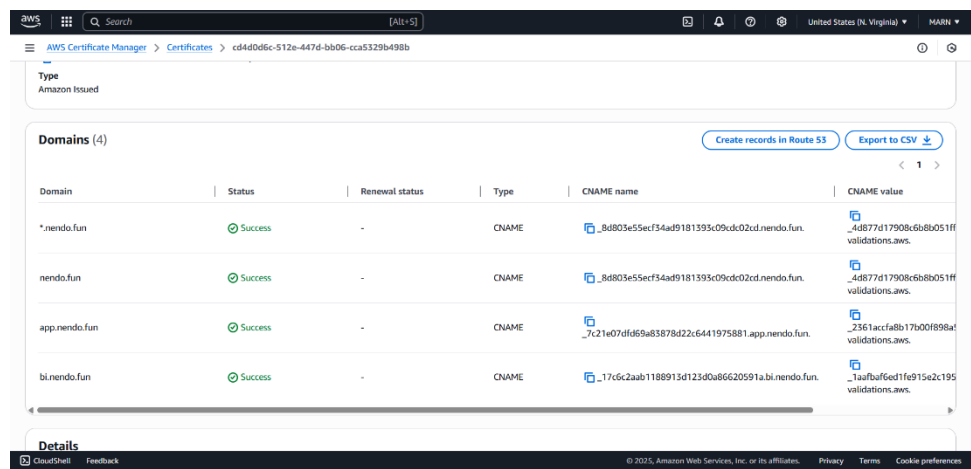
Hosted-Zone (AWS Console)

The screenshot shows the 'Records (8)' page for the 'nendo.fun' hosted zone. It displays a table of DNS records with columns for Record name, Type, Routing, Differ, Alias, Value/Route traffic to, TTL, and Health. The records include NS records for the zone's authoritative servers, a SOA record, and several CNAME and A records for various subdomains like 'app.nendo.fun' and 'bi.nendo.fun'.

Hosted-Zone Records (AWS Console)



ACM (AWS Console)



ACM Domains (AWS Console)

9. Snapshot Strategy (Extra)

- Shell script snapshot-and-destroy.sh created.
- Before destroying Terraform infra:
 - Takes RDS snapshots to preserve data and user config.
 - Then runs terraform destroy for clean teardown.
- Can restore from snapshot on re-deploy.

Screenshot:

Code/Script:

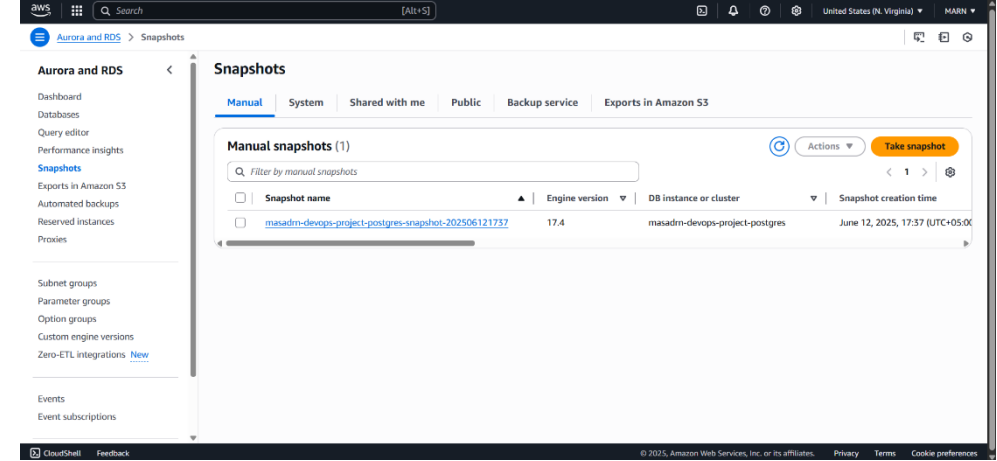
```

$ snapshot-and-destroy.sh
9 # Define snapshot name
10 SNAPSHOT_ID=$(RDS_INSTANCE)-snapshot-$(TIMESTAMP)
11
12 echo "Creating RDS snapshot: $SNAPSHOT_ID"
13
14 # Create the snapshot
15 aws rds create-db-snapshot \
16   --db-instance-identifier $RDS_INSTANCE \
17   --db-snapshot-identifier $SNAPSHOT_ID
18
19 # Optional: Wait for snapshot to complete
20 echo "Waiting for snapshot to become available..."
21 aws rds wait db-snapshot-available \
22   --db-snapshot-identifier $SNAPSHOT_ID
23
24 echo "Snapshot $SNAPSHOT_ID is available."
25
26 # Run Terraform destroy
27 echo "Destroying infrastructure using Terraform..."
28 terraform destroy -auto-approve
29

```

Snapshot script (file)

AWS:



RDS Snapshot (AWS Console)

