Setting up Highly Available RDS Service on AWS with Terraform

Prepared by

Sohail Abid

Sohailabid786@hotmail.Co.uk

Git Repository: https://github.com/sohailabid864/Terraform-AWS-RDS

Overview

In the realm of cloud infrastructure, achieving high availability and performance is paramount, particularly for critical services such as databases. This guide outlines the steps to establish a highly available RDS (Relational Database Service) on Amazon Web Services (AWS) using Terraform. The objective is to create an environment that not only ensures the continuous availability of the database but is also optimized for handling requests from a cluster of EC2 instances.

Key Considerations:

High Availability:

Multi-AZ Deployment: The RDS service will be configured for multi-Availability Zone (AZ) deployment. This architecture enhances fault tolerance and ensures that if one AZ becomes unavailable, the system seamlessly switches to another.

Performance Optimization:

Instance Class Selection: The RDS instance class, specified in Terraform, will be chosen to meet the performance requirements of the workload. For example, we may opt for an instance with higher compute and memory resources for improved database performance.

Allocated Storage: The allocated storage for the RDS instance will be tailored to accommodate the anticipated data volume and throughput, optimizing storage performance.

Terraform Automation:

Infrastructure as Code (IaC): Leveraging Terraform, we'll follow best practices of Infrastructure as Code, ensuring the reproducibility and manageability of our AWS environment.

Modular Configuration: The setup will be organized into modules, promoting reusability and maintainability. This modular structure enhances the scalability of the infrastructure.

Prerequisites

- Install Terraform
- AWS credentials configured on your machine
- AWS CLI set up with the correct region and permissions

Directory Structure

```
-terraform-rds-module
      -main.tf
      -variables.tf
      -outputs.tf
      -terraform.tfvars
      -provider.tf
```

Steps

1. Install Terraform:
   - Download and install Terraform from Terraform Downloads.
2. Create a Module Folder (terraform-rds-module):
   - Create a folder to contain your Terraform module.

```
mkdir terraform-rds-module
```

3. Create Variable File (variables.tf):
   - Define module variables in variables.tf.

```
Notepad variables.tf
```

```
variable "aws_region" {
  description = "AWS region for resources"
}

variable "db_instance_identifier" {
  description = "Identifier for the RDS instance"
}

variable "db_username" {
  description = "Username for the RDS instance"
}

variable "db_password" {
  description = "Password for the RDS instance"
}

variable "db_instance_class" {
  description = "RDS instance class"
  default     = "db.t3.medium"
}

variable "allocated_storage" {
  description = "Allocated storage for RDS instance"
  default     = 200
}
```

4. Create Terraform Variables File (terraform.tfvars):
    - Populate terraform.tfvars with values for your variables.

```
Notepad terraform.tfvars
```

```
aws_region            = "us-east-1"
db_instance_identifier = "ecommerce-db"
db_username           = "admin"
db_password           = "Sudd3n!y!!!"
```

5. Create Provider File (provider.tf):
    - Configure the AWS provider in provider.tf.

```
Notepad provider.tf
```

```
provider "aws" {
  region = var.aws_region
}
```

6. Create Main Configuration File (main.tf):
    - Define RDS resource and include the module in main.tf.

```
Notepad main.tf
```

```
module "rds" {
  source = "./terraform-rds-module"

  aws_region            = var.aws_region
  db_instance_identifier = var.db_instance_identifier
  db_username           = var.db_username
  db_password           = var.db_password
  db_instance_class     = var.db_instance_class
  allocated_storage     = var.allocated_storage
}
```

7. Create RDS Module (terraform-rds-module/main.tf):
    - Define the RDS module with the necessary RDS resource configuration.

```
Notepad main.tf
```

```
resource "aws_db_instance" "ecommerce_db" {
  identifier           = var.db_instance_identifier
  engine               = "mysql"
  instance_class       = var.db_instance_class
  multi_az             = true
  allocated_storage    = var.allocated_storage
  storage_type         = "gp2"
  username             = var.db_username
  password             = var.db_password
  publicly_accessible  = false
}
```

8. Create RDS Module Variables (terraform-rds-module/variables.tf):
    - Define input variables for the RDS module.

```
Notepad variables.tf
```

```
variable "aws_region" {
  description = "AWS region for resources"
}

variable "db_instance_identifier" {
  description = "Identifier for the RDS instance"
}

variable "db_username" {
  description = "Username for the RDS instance"
}

variable "db_password" {
  description = "Password for the RDS instance"
}

variable "db_instance_class" {
  description = "RDS instance class"
}

variable "allocated_storage" {
  description = "Allocated storage for RDS instance"
}
```

9. Create RDS Module Outputs (terraform-rds-module/output.tf):
   - ➢ Define outputs for the RDS module, such as the RDS endpoint.

```
Notepad variables.tf
```

```
output "rds_endpoint" {
  value = aws_db_instance.ecommerce_db.endpoint
}
```

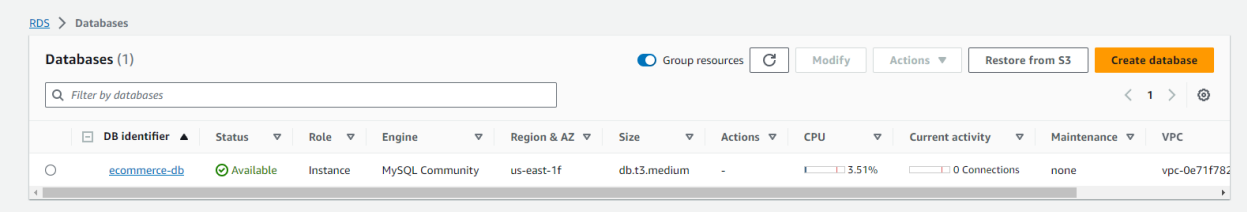
10. Run Terraform Commands:

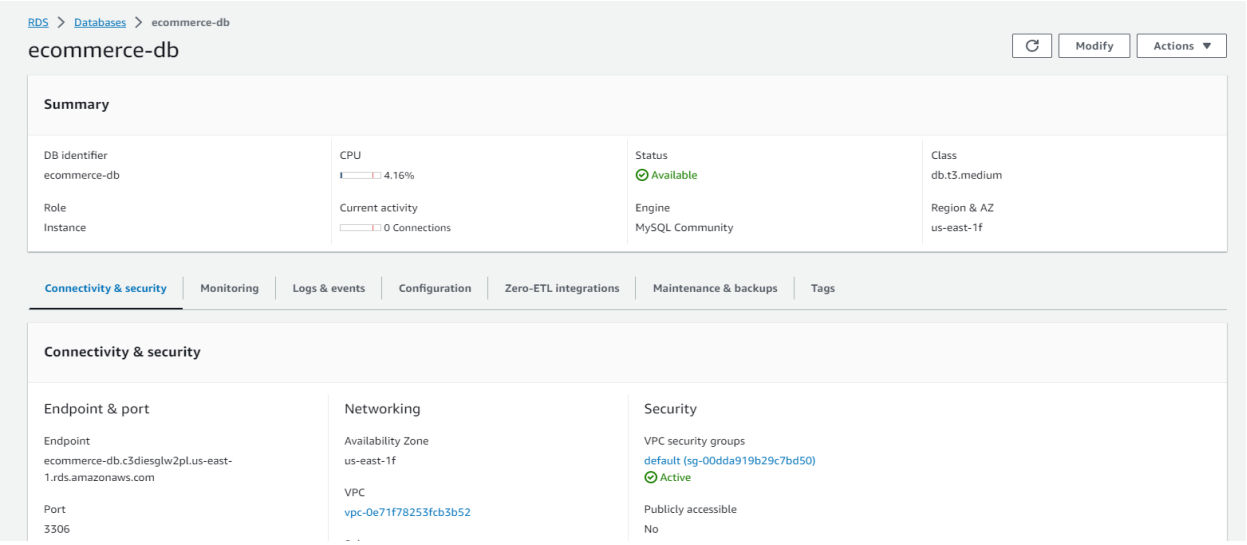
```
terraform init
```

```
terraform plan -out=tfplan #To save the existing plan
```

```
terraform apply “tfplan”
```

Final Output from AWS console





Conclusion:

By following this comprehensive document, you'll establish a highly available RDS service on AWS, finely tuned for optimal performance to serve requests from a cluster of EC2 instances. This approach combines the power of Terraform's Infrastructure as Code capabilities with AWS's robust RDS service, resulting in a scalable, reliable, and performant database solution.