Task 9: ECS Autoscaling with Terraform

Overview

This document outlines the process of setting up ECS (Elastic Container Service) Autoscaling with Terraform to dynamically adjust container instances based on workload demands. The goal is to ensure efficient resource utilization and cost management by automatically scaling the ECS service up or down based on the specified metrics.

Objectives

- Provision an ECS cluster with Autoscaling capabilities.
- Set up a target tracking scaling policy to manage the desired count of ECS tasks.
- Ensure that the system can scale between 1 and 3 instances based on CPU utilization.
- Use Terraform for Infrastructure as Code (IaC) to create and manage AWS resources.

Prerequisites

- An AWS account with the necessary permissions to create and manage ECS, IAM, and EC2 resources.
- Terraform installed on your local machine.
- A text editor or IDE to edit Terraform configuration files.
- A Docker image for the ECS task already pushed to Amazon ECR.

Implementation Steps

Step 1: Create a Terraform Configuration

1. **Create a Directory for the Project**

bash

```
mkdir my-terraform-project
cd my-terraform-project
```

2. **Create Main Terraform Configuration Files**

Create the following Terraform configuration files:

    - main.tf: For defining the primary AWS resources (ECS cluster, task definition, etc.).
    - variables.tf: For declaring input variables.
    - terraform.tfvars: For providing values for the input variables.
    - outputs.tf: For defining outputs from the Terraform configuration.

3. **Define the Provider**

In main.tf, specify the AWS provider and region.

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

4. **Declare Input Variables**

In variables.tf, declare the necessary input variables.

```
variable "region" {
  description = "AWS region"
  type        = string
  default     = "us-east-1"
}

variable "ecs_scaling_target_cpu_utilization" {
  description = "Target CPU utilization for ECS autoscaling"
  type        = number
  default     = 50
}

variable "min_capacity" {
  description = "Minimum number of tasks"
  type        = number
  default     = 1
}

variable "max_capacity" {
  description = "Maximum number of tasks"
  type        = number
```

```
  default     = 3
}

variable "image_uri" {
  description = "Docker image URI for the ECS task"
  type        = string
}

variable "vpc_id" {
  description = "VPC ID for the ECS service"
  type        = string
}

variable "public_subnet_ids" {
  description = "List of public subnet IDs for the ECS service"
  type        = list(string)
}
```

5. **Define the ECS Cluster and Task Definition**

In main.tf, define the ECS cluster and task definition.

```
resource "aws_ecs_cluster" "my_cluster" {
  name = "my-ecs-cluster"
}

resource "aws_ecs_task_definition" "medusa_task" {
  family                   = "medusa-task"
  network_mode             = "awsvpc"
  requires_compatibilities  = ["FARGATE"]
  cpu                      = "256"  # Adjust as necessary
  memory                   = "512"  # Adjust as necessary
```

```
  container_definitions = jsonencode([
   {
    name      = "medusa"
    image     = var.image_uri
    cpu       = 256
    memory    = 512
    essential = true
    portMappings = [
     {
      containerPort = 80
      hostPort      = 80
      protocol      = "tcp"
     }
    ]
   }
  ])
}
```

6. **Set Up ECS Service with Autoscaling**

Define the ECS service and autoscaling configurations.

```
resource "aws_ecs_service" "medusa_service" {
  name            = "medusa-service"
  cluster         = aws_ecs_cluster.my_cluster.id
  task_definition = aws_ecs_task_definition.medusa_task.arn
  desired_count   = var.min_capacity

  launch_type = "FARGATE"

  network_configuration {
```

```
    subnets         = var.public_subnet_ids
    security_groups  = [aws_security_group.app.id]  # Adjust if necessary
    assign_public_ip = "ENABLED"
  }

  health_check_grace_period_seconds = 60
}

resource "aws_appautoscaling_target" "ecs_autoscaling_target" {
  max_capacity       = var.max_capacity
  min_capacity       = var.min_capacity
  resource_id        =
"service/${aws_ecs_cluster.my_cluster.name}/${aws_ecs_service.medusa_service.name}"
  scalable_dimension  = "ecs:service:DesiredCount"
  service_namespace   = "ecs"
}

resource "aws_appautoscaling_policy" "ecs_autoscaling_policy" {
  name                = "ecs-autoscaling-policy"
  policy_type         = "TargetTrackingScaling"
  resource_id         = aws_appautoscaling_target.ecs_autoscaling_target.id
  scalable_dimension     = "ecs:service:DesiredCount"
  service_namespace      = "ecs"

  target_tracking_scaling_policy_configuration {
    target_value = var.ecs_scaling_target_cpu_utilization
    pre_scaling_adjustment = 0
    scale_in_cooldown  = 60
    scale_out_cooldown = 60

    metric_specifications {
```

```
    metric_name = "CPUUtilization"
    statistic   = "AVERAGE"
    period      = 60
    unit        = "Percent"
   }
  }
}
```

Step 2: Configure terraform.tfvars

Provide values for the input variables in terraform.tfvars.

```
region = "us-east-1"
image_uri = "jaswanth-medusa:latest"
vpc_id = "your-vpc-id"  # Replace with your actual VPC ID
public_subnet_ids = ["subnet-0a044fda974f92d5b", "subnet-0a0f031fdc6225420"]  # Replace with your actual subnet IDs
```

Step 3: Initialize Terraform

Run the following command to initialize Terraform:

bash

```
terraform init
```

Step 4: Plan and Apply the Configuration

1. **Plan the Infrastructure**

Run the command below to see what resources will be created:

bash

```
terraform plan
```

2. **Apply the Configuration**

Run the command below to create the resources:

bash

terraform apply

Confirm the action when prompted.

Step 5: Verify the Setup

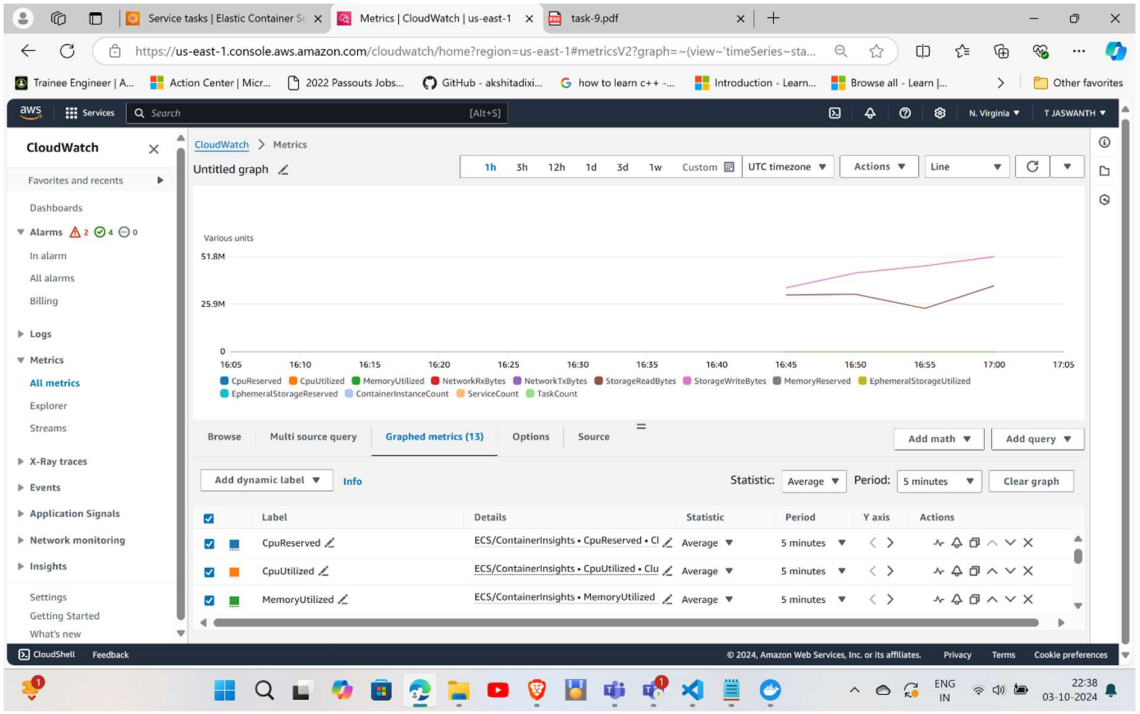
- Check the AWS Management Console for the ECS cluster and service.
- Monitor the ECS service to see if it scales based on CPU utilization.

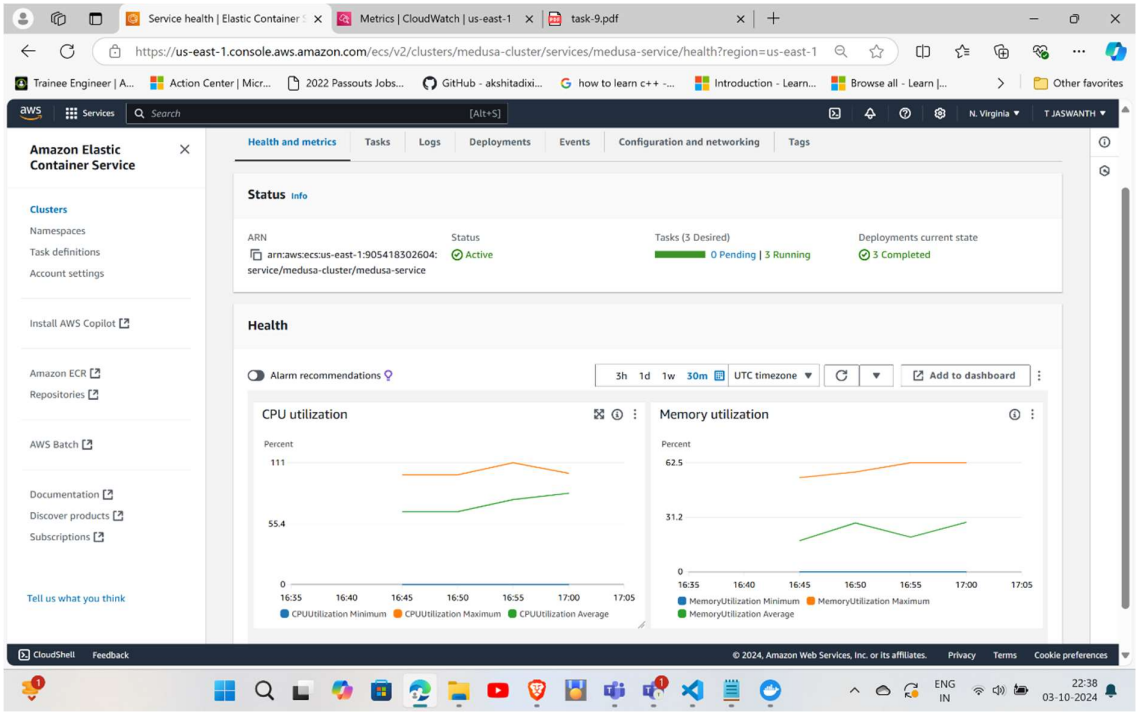
Outputs

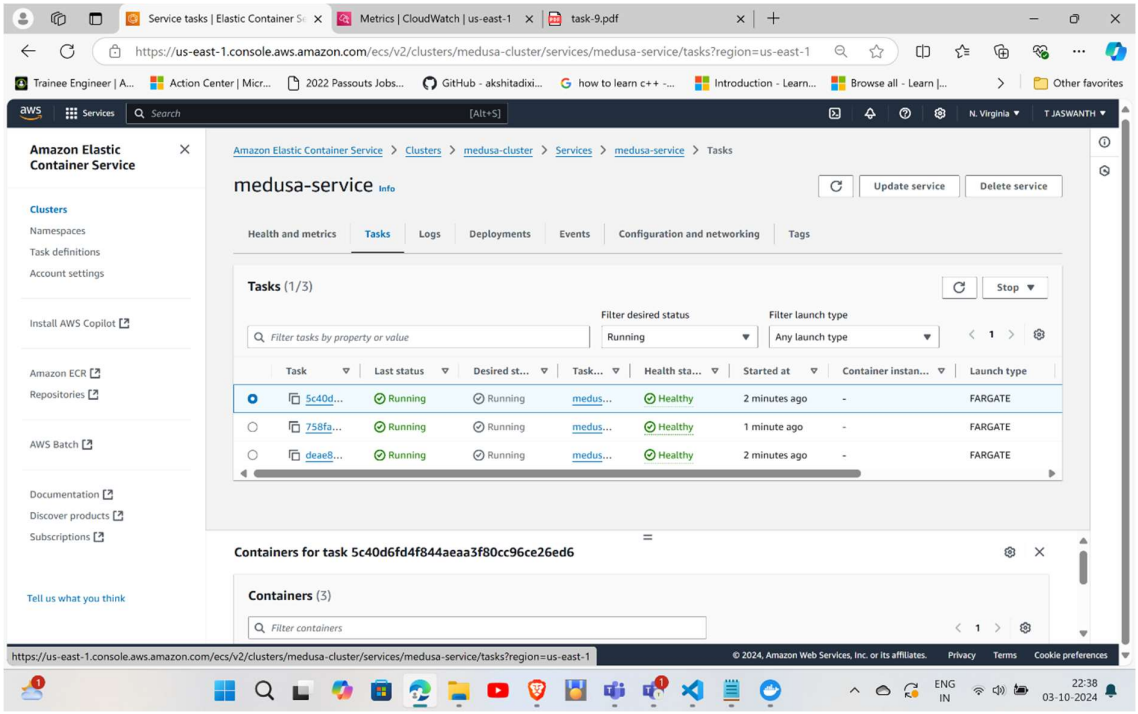
You can define outputs in outputs.tf to retrieve important information after the infrastructure is created:

```
output "cluster_name" {
  value = aws_ecs_cluster.my_cluster.name
}

output "service_name" {
  value = aws_ecs_service.medusa_service.name
}
```







Conclusion

This documentation provides a step-by-step guide to setting up ECS Autoscaling using Terraform. By following these instructions, you can effectively manage the scaling of your ECS services based on workload demands, ensuring optimal resource usage and cost management.