Deploy a Containerized Application with ECS Fargate

Yash Mutatkar **Nov 12 2024**

To deploy a containerized application using Amazon ECS with Fargate, we'll go through the steps for defining a task and service, configuring an Application Load Balancer (ALB), and setting up auto-scaling in the AWS Console.

Prerequisites

1. Ensure you have a Docker image (`nov12deployacontainarizedapplicationwithecsfargate_php-app:latest`).
2. Have an active AWS account with the necessary permissions for ECS, ECR, ALB, and CloudWatch.

Step-by-Step Deployment Guide

1. Push Docker Image to Amazon ECR

- Go to the **Elastic Container Registry (ECR)** service in AWS Console.
- Click on **Create repository**, name the repository (e.g., `php-app`), and create it.
- Follow the **View push commands** on your ECR repository page to push your image. Replace `<your-repo-url>` with the actual repository URL.

```
bash
Copy code
aws ecr get-login-password --region <region> | docker login --username AWS
--password-stdin <your-repo-url>
docker tag nov12deployacontainarizedapplicationwithecsfargate_php-
app:latest <your-repo-url>:latest
docker push <your-repo-url>:latest
```

2. Create an ECS Cluster

- Go to the **Elastic Container Service (ECS)** section.
- Select **Clusters** > **Create Cluster**.
- Choose **Networking only** for Fargate.
- Enter a cluster name (e.g., `php-fargate-cluster`) and create the cluster.

3. Define a Task Definition

- In ECS, go to **Task Definitions** and select **Create new Task Definition**.
- Choose **Fargate** as the launch type.
- Specify a task name (e.g., `php-fargate-task`).
- Set up the **Task execution role** if not already available. This role should have permissions to pull images from ECR.
- Define container settings:
    - Under **Container Definitions**, add a container:

    - **Name**: `php-container`.
    - **Image**: Enter the ECR image URL (e.g., `<your-repo-url>:latest`).
    - **Memory Limits**: Set memory limit (e.g., `512 MiB`).
    - **Port Mappings**: Set container port to `80`.
  - Click **Add** and then **Create**.

4. Create an Application Load Balancer (ALB)

- Go to **Load Balancers** under the EC2 service.
- Click **Create Load Balancer** and select **Application Load Balancer**.
- Set **Name**, **Scheme** (Internet-facing), and **IP address type** (IPv4).
- Under **Network mapping**, select the VPC and subnets.
- In **Security Groups**, choose an existing group or create a new one allowing HTTP traffic on port `80`.
- Under **Listeners and routing**, select `HTTP: 80`, and for **Default action**, choose **Create target group**:
  - In the target group creation page, select **Target type** as IP, protocol as HTTP, and set `Port` to 80.
  - Complete the setup and go back to finalize ALB creation.

5. Set Up the ECS Service

- Return to **ECS** > **Clusters**, select the cluster (`php-fargate-cluster`), and choose **Create** for **Service**.
- Select the **Fargate** launch type.
- Configure the service:
  - **Service name**: `php-fargate-service`.
  - **Number of tasks**: Start with `1`.
  - **Minimum Healthy Percent**: 50%.
  - **Maximum Percent**: 200%.
- In **Load balancing**:
  - Enable the **Application Load Balancer**.
  - Choose the target group created in Step 4.
- Under **Auto Scaling**:
  - Enable auto-scaling and configure it to **Add scaling policy** based on **CPU utilization**.
  - Set the target CPU utilization percentage (e.g., 50%).
- Click **Create Service**.

6. Test the Application

- Navigate to **Load Balancers** and find the ALB's DNS name.
- Access the DNS name in a browser to confirm the application is runnin

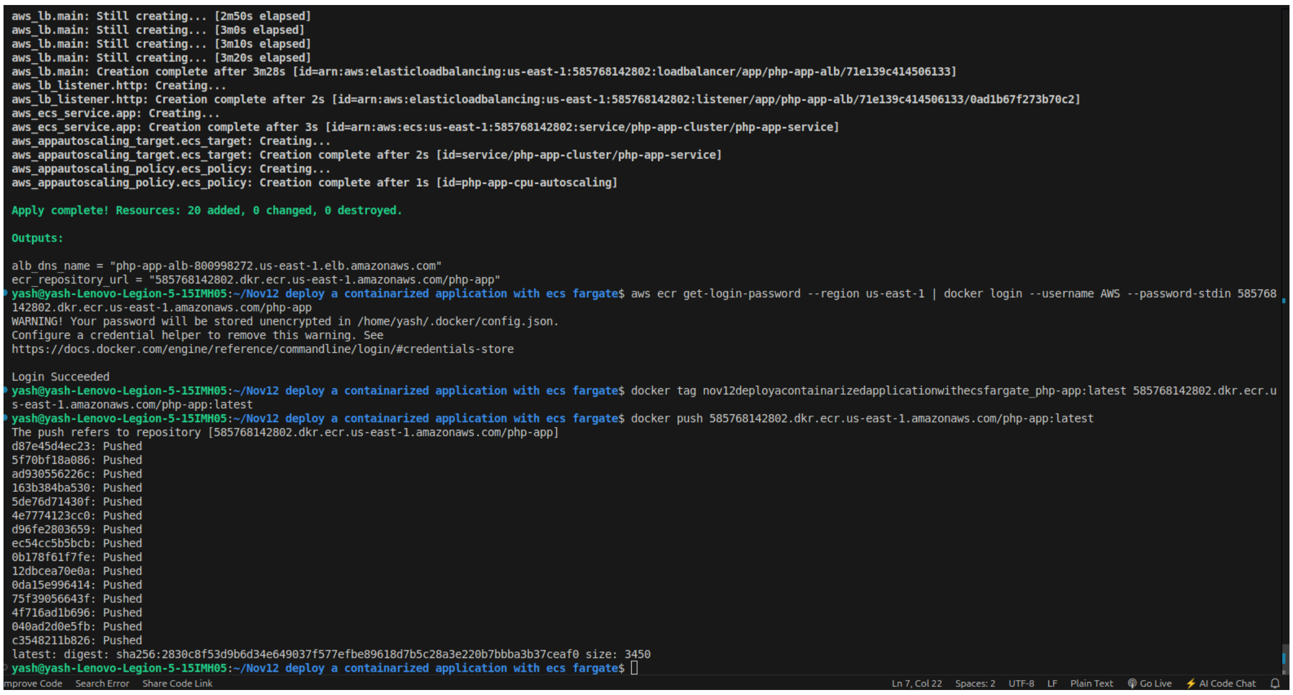
}

Using Terraform

```
# provider.tf
provider "aws" {
  region = "us-east-1"
}

# variables.tf
variable "app_name" {
  description = "Application name"
  default     = "php-app"
}

variable "environment" {
  description = "Environment name"
  default     = "prod"
}

variable "container_port" {
  description = "Container port"
  default     = 80
}

variable "container_cpu" {
  description = "Container CPU units"
  default     = 256
}

variable "container_memory" {
  description = "Container memory in MiB"
  default     = 512
}

variable "desired_count" {
  description = "Desired task count"
  default     = 1
}

# vpc.tf
data "aws_availability_zones" "available" {
  state = "available"
}

resource "aws_vpc" "main" {
  cidr_block           = "10.0.0.0/16"
  enable_dns_hostnames = true
  enable_dns_support   = true
  tags = {
    Name = "${var.app_name}-vpc"
  }
}

resource "aws_subnet" "public" {
  count                   = 2
  vpc_id                  = aws_vpc.main.id
  cidr_block              = "10.0.${count.index + 1}.0/24"
  availability_zone       = data.aws_availability_zones.available.names[count.index]
  map_public_ip_on_launch = true

  tags = {
    Name = "${var.app_name}-public-subnet-${count.index + 1}"
  }
}

resource "aws_internet_gateway" "main" {
  vpc_id = aws_vpc.main.id

  tags = {
    Name = "${var.app_name}-igw"
  }
}

resource "aws_route_table" "public" {
  vpc_id = aws_vpc.main.id

  route {
    cidr_block = "0.0.0.0/0"
    gateway_id = aws_internet_gateway.main.id
  }

  tags = {
    Name = "${var.app_name}-public-rt"
  }
}

resource "aws_route_table_association" "public" {
  count          = 2
  subnet_id      = aws_subnet.public[count.index].id
  route_table_id = aws_route_table.public.id
}

# ecr.tf
resource "aws_ecr_repository" "app" {
  name = var.app_name

  image_scanning_configuration {
    scan_on_push = true
  }
```

```
}

# security.tf
resource "aws_security_group" "alb" {
  name        = "${var.app_name}-alb-sg"
  description = "ALB Security Group"
  vpc_id      = aws_vpc.main.id

  ingress {
    protocol    = "tcp"
    from_port   = 80
    to_port     = 80
    cidr_blocks = ["0.0.0.0/0"]
  }

  egress {
    protocol    = "-1"
    from_port   = 0
    to_port     = 0
    cidr_blocks = ["0.0.0.0/0"]
  }
}

resource "aws_security_group" "ecs_tasks" {
  name        = "${var.app_name}-ecs-tasks-sg"
  description = "ECS Tasks Security Group"
  vpc_id      = aws_vpc.main.id

  ingress {
    protocol        = "tcp"
    from_port       = var.container_port
    to_port         = var.container_port
    security_groups = [aws_security_group.alb.id]
  }

  egress {
    protocol    = "-1"
    from_port   = 0
    to_port     = 0
    cidr_blocks = ["0.0.0.0/0"]
  }
}

# iam.tf
resource "aws_iam_role" "ecs_task_execution_role" {
  name = "${var.app_name}-ecs-task-execution-role"

  assume_role_policy = jsonencode({
    Version = "2012-10-17"
    Statement = [
      {
        Action = "sts:AssumeRole"
        Effect = "Allow"
        Principal = {
          Service = "ecs-tasks.amazonaws.com"
        }
      }
    ]
  })
}

resource "aws_iam_role_policy_attachment" "ecs_task_execution_role_policy" {
  role       = aws_iam_role.ecs_task_execution_role.name
  policy_arn = "arn:aws:iam::aws:policy/service-role/AmazonECSTaskExecutionRolePolicy"
}

# alb.tf
resource "aws_lb" "main" {
  name               = "${var.app_name}-alb"
  internal           = false
  load_balancer_type = "application"
  security_groups    = [aws_security_group.alb.id]
  subnets            = aws_subnet.public[*].id
}

resource "aws_lb_target_group" "app" {
  name        = "${var.app_name}-tg"
  port        = var.container_port
  protocol    = "HTTP"
  vpc_id      = aws_vpc.main.id
  target_type = "ip"

  health_check {
    path                = "/"
    healthy_threshold   = 2
    unhealthy_threshold = 10
  }
}

resource "aws_lb_listener" "http" {
  load_balancer_arn = aws_lb.main.arn
  port              = 80
  protocol          = "HTTP"

  default_action {
    type             = "forward"
    target_group_arn = aws_lb_target_group.app.arn
  }
}

# ecs.tf
resource "aws_ecs_cluster" "main" {
  name = "${var.app_name}-cluster"
}
```

```
resource "aws_ecs_task_definition" "app" {
  family                   = "${var.app_name}-task"
  network_mode             = "awsvpc"
  requires_compatibilities = ["FARGATE"]
  cpu                      = var.container_cpu
  memory                   = var.container_memory
  execution_role_arn       = aws_iam_role.ecs_task_execution_role.arn

  container_definitions = jsonencode([
    {
      name      = var.app_name
      image     = "${aws_ecr_repository.app.repository_url}:latest"
      cpu       = var.container_cpu
      memory    = var.container_memory
      essential = true
      portMappings = [
        {
          containerPort = var.container_port
          hostPort      = var.container_port
          protocol      = "tcp"
        }
      ]
    }
  ])
}

resource "aws_ecs_service" "app" {
  name            = "${var.app_name}-service"
  cluster         = aws_ecs_cluster.main.id
  task_definition = aws_ecs_task_definition.app.arn
  desired_count   = var.desired_count
  launch_type     = "FARGATE"

  network_configuration {
    security_groups = [aws_security_group.ecs_tasks.id]
    subnets         = aws_subnet.public[*].id
    assign_public_ip = true
  }

  load_balancer {
    target_group_arn = aws_lb_target_group.app.arn
    container_name   = var.app_name
    container_port   = var.container_port
  }

  depends_on = [aws_lb_listener.http]
}

# autoscaling.tf
resource "aws_appautoscaling_target" "ecs_target" {
  max_capacity       = 4
  min_capacity       = 1
  resource_id        = "service/${aws_ecs_cluster.main.name}/${aws_ecs_service.app.name}"
  scalable_dimension = "ecs:service:DesiredCount"
  service_namespace  = "ecs"
}

resource "aws_appautoscaling_policy" "ecs_policy" {
  name               = "${var.app_name}-cpu-autoscaling"
  policy_type        = "TargetTrackingScaling"
  resource_id        = aws_appautoscaling_target.ecs_target.resource_id
  scalable_dimension = aws_appautoscaling_target.ecs_target.scalable_dimension
  service_namespace  = aws_appautoscaling_target.ecs_target.service_namespace

  target_tracking_scaling_policy_configuration {
    predefined_metric_specification {
      predefined_metric_type = "ECSServiceAverageCPUUtilization"
    }
    target_value = 50.0
  }
}

# outputs.tf
output "alb_dns_name" {
  value = aws_lb.main.dns_name
}

output "ecr_repository_url" {
  value = aws_ecr_repository.app.repository_url
}
```



```
aws_lb.main: Still creating... [2m50s elapsed]
aws_lb.main: Still creating... [3m0s elapsed]
aws_lb.main: Still creating... [3m10s elapsed]
aws_lb.main: Still creating... [3m20s elapsed]
aws_lb.main: Creation complete after 3m28s [id=arn:aws:elasticloadbalancing:us-east-1:585768142802:loadbalancer/app/php-app-alb/71e139c414506133]
aws_lb_listener.http: Creating...
aws_lb_listener.http: Creation complete after 2s [id=arn:aws:elasticloadbalancing:us-east-1:585768142802:listener/app/php-app-alb/71e139c414506133/0ad1b67f273b70c2]
aws_ecs_service.app: Creating...
aws_ecs_service.app: Creation complete after 3s [id=arn:aws:ecs:us-east-1:585768142802:service/php-app-cluster/php-app-service]
aws_appautoscaling_target.ecs_target: Creating...
aws_appautoscaling_target.ecs_target: Creation complete after 2s [id=service/php-app-cluster/php-app-service]
aws_appautoscaling_policy.ecs_policy: Creating...
aws_appautoscaling_policy.ecs_policy: Creation complete after 1s [id=php-app-cpu-autoscaling]

Apply complete! Resources: 20 added, 0 changed, 0 destroyed.

Outputs:

alb_dns_name = "php-app-alb-800998272.us-east-1.elb.amazonaws.com"
ecr_repository_url = "585768142802.dkr.ecr.us-east-1.amazonaws.com/php-app"
yash@yash-Lenovo-Legion-5-15IMH05:~/Nov12 deploy a containarized application with ecs fargate$ aws ecr get-login-password --region us-east-1 | docker login --username AWS --password-stdin 585768
142802.dkr.ecr.us-east-1.amazonaws.com/php-app
WARNING! Your password will be stored unencrypted in /home/yash/.docker/config.json.
Configure a credential helper to remove this warning. See
https://docs.docker.com/engine/reference/commandline/login/#credentials-store

Login Succeeded
yash@yash-Lenovo-Legion-5-15IMH05:~/Nov12 deploy a containarized application with ecs fargate$ docker tag nov12deployacontainarizedapplicationwithecsfargate_php-app:latest 585768142802.dkr.ecr.u
s-east-1.amazonaws.com/php-app:latest
yash@yash-Lenovo-Legion-5-15IMH05:~/Nov12 deploy a containarized application with ecs fargate$ docker push 585768142802.dkr.ecr.us-east-1.amazonaws.com/php-app:latest
The push refers to repository [585768142802.dkr.ecr.us-east-1.amazonaws.com/php-app]
d87e45d4ec23: Pushed
5f70bf18a086: Pushed
ad930556226c: Pushed
163b384ba530: Pushed
5de76d71430f: Pushed
4e7774123cc0: Pushed
d96fe2803659: Pushed
ec54cc5b5bcb: Pushed
0b178f61f7fe: Pushed
12dbcea70e0a: Pushed
0da15e996414: Pushed
75f39056643f: Pushed
4f716ad1b696: Pushed
040ad2d0e5fb: Pushed
c3548211b826: Pushed
latest: digest: sha256:2830c8f53d9b6d34e649037f577efbe89618d7b5c28a3e220b7bbba3b37ceaf0 size: 3450
yash@yash-Lenovo-Legion-5-15IMH05:~/Nov12 deploy a containarized application with ecs fargate$ 
```