









Full ECS + ECR Deployment Steps for now - later with using Copilot

Step Task.

- 1  Create ECR repositories.
- 2  Authenticate Docker to ECR.
- 3  Build and push multi-arch Docker images.
- 4  Create IAM execution role for ECS tasks.
- 5  Define ECS Fargate-compatible task definition.
- 6  Launch ECS service on Fargate.

Set up the underpinning AWS infrastructure

Sets up temporary environment variables in your current shell session.

```
export AWS_REGION=eu-west-1
export VPC_CIDR=10.0.0.0/16
export SUBNET_CIDR=10.0.1.0/24
export CLUSTER_NAME=quote-app-cluster
export BACKEND_PORT=8080
export FRONTEND_PORT=80
```

✓ To make them persistent across sessions (Zsh example):

```
echo 'export AWS_REGION=eu-west-1' >> ~/.zshrc
source ~/.zshrc
```

✓ How to check they're set:

```
echo $AWS_REGION
echo $VPC_CIDR
```

Expected output:

```
eu-west-1
10.0.0.0/16
```

1. Create VPC networking (or use existing)

- A VPC and one or more Subnet IDs.
- A Security Group.

- The subnet must be public (with IGW access), SG must allow inbound HTTP (80) and backend port (8080).

1.1 - Create a VPC and tag it.

```
VPC_ID=$(aws ec2 create-vpc --cidr-block $VPC_CIDR --query 'Vpc.VpcId' --
output text)
aws ec2 create-tags --resources $VPC_ID --tags Key=Name,Value=ecs-test-vpc
```

1.2 - Create a public subnet and tag it

```
SUBNET_ID=$(aws ec2 create-subnet \
--vpc-id $VPC_ID \
--cidr-block $SUBNET_CIDR \
--availability-zone ${AWS_REGION}a \
--query 'Subnet.SubnetId' --output text)
aws ec2 create-tags --resources $SUBNET_ID --tags Key=Name,Value=subnet-pub
```

1.3 - Enable auto-assign public IP

```
aws ec2 modify-subnet-attribute --subnet-id $SUBNET_ID --map-public-ip-on-
launch
```

1.4 - Create and attach Internet Gateway

```
IGW_ID=$(aws ec2 create-internet-gateway --query
'InternetGateway.InternetGatewayId' --output text)
aws ec2 attach-internet-gateway --internet-gateway-id $IGW_ID --vpc-id
$VPC_ID
```

1.5 - Create Route Table and add default route

```
RT_ID=$(aws ec2 create-route-table --vpc-id $VPC_ID --query
'RouteTable.RouteTableId' --output text)
aws ec2 create-route --route-table-id $RT_ID --destination-cidr-block 0.
0.0.0/0 --gateway-id $IGW_ID
aws ec2 associate-route-table --route-table-id $RT_ID --subnet-id $SUBNET_ID
```

1.6 - Create Security Group with Ingress Rules

```
SG_ID=$(aws ec2 create-security-group \
--group-name quote-sg \
--description "Allow HTTP ports" \
--vpc-id $VPC_ID \
--query 'GroupId' --output text)
```

```
# Allow 80 (frontend)
aws ec2 authorize-security-group-ingress --group-id $SG_ID --protocol tcp --
port 80 --cidr 0.0.0.0/0

# Allow 8080 (backend)
aws ec2 authorize-security-group-ingress --group-id $SG_ID --protocol tcp --
port 8080 --cidr 0.0.0.0/0
```

1.7 - IAM Execution Role for ECS Tasks

```
aws iam create-role --role-name ecsTaskExecutionRole \
  --assume-role-policy-document '{
    "Version": "2012-10-17",
    "Statement": [{
      "Effect": "Allow",
      "Principal": { "Service": "ecs-tasks.amazonaws.com" },
      "Action": "sts:AssumeRole"
    }]
  }'

aws iam attach-role-policy \
  --role-name ecsTaskExecutionRole \
  --policy-arn arn:aws:iam::aws:policy/service-role/
AmazonECSTaskExecutionRolePolicy
```

2. Task Definitions (Backend + Frontend) - Save and Register

```
ACCOUNT_ID=$(aws sts get-caller-identity --query Account --output text)
EXECUTION_ROLE_ARN="arn:aws:iam::${ACCOUNT_ID}:role/ecsTaskExecutionRole"
```

quote-backend-task.json

```
{
  "family": "quote-backend-task",
  "networkMode": "awsvpc",
  "requiresCompatibilities": ["FARGATE"],
  "cpu": "256",
  "memory": "512",
  "executionRoleArn": "${EXECUTION_ROLE_ARN}",
  "containerDefinitions": [
    {
      "name": "quote-backend",
      "image": "040929397520.dkr.ecr.eu-west-1.amazonaws.com/aws-quote-
backend:latest",
      "portMappings": [{ "containerPort": 8080 }]
    }
  ]
}
```

```
]
}
```

quote-frontend-task.json

```
{
  "family": "quote-frontend-task",
  "networkMode": "awsvpc",
  "requiresCompatibilities": ["FARGATE"],
  "cpu": "256",
  "memory": "512",
  "executionRoleArn": "${EXECUTION_ROLE_ARN}",
  "containerDefinitions": [
    {
      "name": "quote-frontend",
      "image": "040929397520.dkr.ecr.eu-west-1.amazonaws.com/aws-quote-frontend:latest",
      "portMappings": [{ "containerPort": 80 }]
    }
  ]
}
```

Register the task definitions

```
envsubst < quote-backend-task.json > backend-resolved.json
envsubst < quote-frontend-task.json > frontend-resolved.json

aws ecs register-task-definition --cli-input-json file://backend-resolved.json
aws ecs register-task-definition --cli-input-json file://frontend-resolved.json
```



3. Create ECS Cluster

```
# Commented: inconsistent name from earlier plan
# aws ecs create-cluster --cluster-name quote-app-cluster-copilot # <-
# Removed to keep consistent naming

aws ecs create-cluster --cluster-name $CLUSTER_NAME
```



4. Launch Fargate Services

```
# Backend
aws ecs create-service \
  --cluster $CLUSTER_NAME \
  --service-name quote-backend-service \
```

```

--task-definition quote-backend-task \
--desired-count 1 \
--launch-type FARGATE \
--network-configuration
"awsvpcConfiguration={subnets=[$SUBNET_ID],securityGroups=[$SG_ID],assignPublicIp=ENABLED}"

# Frontend
aws ecs create-service \
--cluster $CLUSTER_NAME \
--service-name quote-frontend-service \
--task-definition quote-frontend-task \
--desired-count 1 \
--launch-type FARGATE \
--network-configuration
"awsvpcConfiguration={subnets=[$SUBNET_ID],securityGroups=[$SG_ID],assignPublicIp=ENABLED}"

```

5. Post-Deployment Health Checks

```

echo "🕒 Checking backend service status..."
aws ecs describe-services \
--cluster $CLUSTER_NAME \
--services quote-backend-service \
--query "services[*].deployments[*].{status:status, running:runningCount,
pending:pendingCount}" \
--output table

echo "🕒 Checking frontend service status..."
aws ecs describe-services \
--cluster $CLUSTER_NAME \
--services quote-frontend-service \
--query "services[*].deployments[*].{status:status, running:runningCount,
pending:pendingCount}" \
--output table

```

6. Public IP Curl Tests

```

get_public_ip() {
    local CLUSTER=$1
    local SERVICE=$2

    TASK_ARN=$(aws ecs list-tasks --cluster $CLUSTER --service-name $SERVICE --
query 'taskArns[0]' --output text)
    ENI_ID=$(aws ecs describe-tasks --cluster $CLUSTER --tasks $TASK_ARN \
--query "tasks[0].attachments[0].details[?
name=='networkInterfaceId'].value" --output text)
    PUBLIC_IP=$(aws ec2 describe-network-interfaces --network-interface-ids
$ENI_ID \
--query "NetworkInterfaces[0].Association.PublicIp" --output text)
}

```

```
    echo $PUBLIC_IP
}

# Test backend
BACKEND_IP=$(get_public_ip $CLUSTER_NAME quote-backend-service)
echo "🌐 Backend Public IP: $BACKEND_IP"
sleep 5
curl --fail http://$BACKEND_IP:$BACKEND_PORT || echo "❌ Backend curl failed"

# Test frontend
FRONTEND_IP=$(get_public_ip $CLUSTER_NAME quote-frontend-service)
echo "🌐 Frontend Public IP: $FRONTEND_IP"
sleep 5
curl --fail http://$FRONTEND_IP:$FRONTEND_PORT || echo "❌ Frontend curl failed"
```

Bonus: Manual Browser Test

- Backend (API): `http://<backend-public-ip>:8080`
- Frontend (HTML page): `http://<frontend-public-ip>`