- Healthcare SOA AWS Infrastructure Deployment Guide
    - Prerequisites
    - 1. AWS Authentication Setup
    - 2. Building Docker Images
    - 3. ECR Repository Setup & Image Pushing
    - 4. Creating terraform.tfvars File
    - 5. Deploying AWS Infrastructure with Terraform
    - 6. Verifying the Deployment
    - 7. Setting Up CI/CD (Optional)
    - 8. Cleaning Up Resources
    - Troubleshooting
        - Common Issues:
    - Best Practices
    - Next Steps

Healthcare SOA AWS Infrastructure Deployment Guide

This guide explains how to deploy the Healthcare SOA project to AWS using Terraform. The deployment process includes building Docker images, pushing them to Amazon ECR, and using Terraform to create and manage the AWS infrastructure.

Prerequisites

Before starting the deployment process, ensure you have the following:

1. **AWS CLI** configured with appropriate credentials
2. **Terraform** (version 1.0.0 or newer) installed
3. **Docker** installed locally
4. **AWS Account** with appropriate permissions
5. **Git** installed

1. AWS Authentication Setup

First, ensure your AWS CLI is configured with the appropriate credentials:

```
aws configure
```

You'll need to provide:

- AWS Access Key ID
- AWS Secret Access Key
- Default region (e.g., us-east-1)
- Default output format (json)

2. Building Docker Images

Before deploying to AWS, build your Docker images locally to ensure they work as expected:

```
# Navigate to project root
cd <PROJECT-DIR>\twelve-factor\healthcare-soa

# Build images using docker-compose
docker-compose build
```

3. ECR Repository Setup & Image Pushing

Use Terraform to create ECR repositories for your services:

```
# Navigate to the Terraform directory
cd <PROJECT-DIR>\twelve-factor\healthcare-soa\terraform

# Initialize Terraform
terraform init

# Create only the ECR repositories first
terraform apply -target=module.ecr
```

Once the ECR repositories are created, tag and push your images:

```
# Get AWS account ID
$AWS_ACCOUNT_ID=$(aws sts get-caller-identity --query Account --output text)
$AWS_REGION="us-east-1"  # Replace with your region

# Log in to ECR
aws ecr get-login-password --region $AWS_REGION | docker login --username AWS --
password-stdin "$AWS_ACCOUNT_ID.dkr.ecr.$AWS_REGION.amazonaws.com"

# Tag and push each service image
$SERVICES = @("esb", "patient-service", "appointment-service")

foreach ($SERVICE in $SERVICES) {
    # Tag image
    docker tag "healthcare-$SERVICE"
"$AWS_ACCOUNT_ID.dkr.ecr.$AWS_REGION.amazonaws.com/healthcare-soa-$SERVICE:latest"

    # Push image
    docker push "$AWS_ACCOUNT_ID.dkr.ecr.$AWS_REGION.amazonaws.com/healthcare-
soa-$SERVICE:latest"

    Write-Host "Successfully pushed $SERVICE image to ECR"
}
```

4. Creating terraform.tfvars File

Create a `terraform.tfvars` file to customize deployment variables:

```
# Navigate to the Terraform directory
cd <PROJECT-DIR>\twelve-factor\healthcare-soa\terraform

# Create terraform.tfvars file
@"
aws_region = "us-east-1"
project_name = "healthcare-soa"
environment = "dev"
vpc_cidr = "10.0.0.0/16"
availability_zones = ["us-east-1a", "us-east-1b"]
alert_email = "your-email@example.com"
"@ | Out-File -FilePath terraform.tfvars -Encoding utf8
```

5. Deploying AWS Infrastructure with Terraform

Deploy the complete infrastructure:

```
# Navigate to the Terraform directory if not already there
cd <PROJECT-DIR>\twelve-factor\healthcare-soa\terraform

# Initialize Terraform (if not done already)
terraform init

# Validate the Terraform configuration
terraform validate

# Create an execution plan
terraform plan -out=tfplan

# Apply the execution plan
terraform apply tfplan
```

The deployment will create:

1. A new VPC with public and private subnets
2. ECR repositories for your services
3. ECS cluster and services
4. PostgreSQL RDS instance
5. Redis ElastiCache cluster
6. MongoDB EC2 instance
7. Application Load Balancer
8. CloudWatch monitoring and alarms

6. Verifying the Deployment

After the deployment completes, verify that everything is working:

```
# Get the ALB DNS name
$ALB_DNS=$(terraform output -raw module.ecs.alb_dns_name)

# Test the endpoints
Invoke-WebRequest -Uri "http://$ALB_DNS/patient-service/actuator/health"
Invoke-WebRequest -Uri "http://$ALB_DNS/appointment-service/actuator/health"
```

7. Setting Up CI/CD (Optional)

For continuous deployment, consider setting up AWS CodePipeline:

1. Create a `buildspec.yml` file in each service directory
2. Set up CodeBuild projects to build and push Docker images
3. Configure CodePipeline to trigger ECS deployments

8. Cleaning Up Resources

To avoid unnecessary AWS costs, clean up resources when they're no longer needed:

```
# Navigate to the Terraform directory
cd <PROJECT-DIR>\twelve-factor\healthcare-soa\terraform

# Destroy all resources
terraform destroy
```

Troubleshooting

Common Issues:

1. **ECR Authentication Failures**

   ```
   aws ecr get-login-password --region $AWS_REGION | docker login --username AWS
   --password-stdin "$AWS_ACCOUNT_ID.dkr.ecr.$AWS_REGION.amazonaws.com"
   ```

2. **ECS Service Deployment Failures**

   - Check CloudWatch Logs for service errors
   - Ensure container health checks are passing

3. **Database Connection Issues**

   - Verify security group rules
   - Check environment variables in ECS task definitions

4. **CloudWatch Alarms**

    - Review the CloudWatch dashboard for service metrics
    - Check SNS topics for alert notifications

Best Practices

1. **Infrastructure as Code**: Keep all infrastructure defined in Terraform
2. **Secrets Management**: Use AWS Secrets Manager for database credentials
3. **Environment Isolation**: Use separate environments (dev, staging, prod)
4. **Monitoring**: Set up proper monitoring and alerting with CloudWatch
5. **Backup Strategy**: Configure regular backups for RDS and other data stores

Next Steps

1. Implement a Blue/Green deployment strategy
2. Set up AWS WAF for web application firewall protection
3. Configure VPC flow logs for network monitoring
4. Implement AWS Config for compliance monitoring
5. Set up AWS CloudTrail for auditing

Note: This guide assumes a basic understanding of AWS services and Terraform. Adjust the configurations based on your specific requirements and AWS best practices.