API Gateway Implementation Steps - FreedmAI CI/CD System

API Gateway Implementation Steps - FreedmAI CI/CD System

Overview

This document provides step-by-step instructions for implementing the API Gateway service as the first microservice in the FreedmAI CI/CD system. The implementation includes complete CI/CD pipeline, security scanning, and deployment automation.

Prerequisites

- AWS Account with appropriate permissions
- GitHub organization: `freedmai`
- Local development environment with Node.js, Docker, and Terraform
- AWS CLI configured
- GitHub CLI installed

Phase 1: Repository Setup

1.1 Create GitHub Repository

```
        # Create repository in freedmai organization
        gh repo create freedmai/api-gateway --public --description "API Gateway for FreedmAI microservices"

        # Clone the repository
        git clone https://github.com/freedmai/api-gateway.git
        cd api-gateway
```

1.2 Copy Implementation Files

```
        # Copy all files from local implementation
        cp -r /var/Freedm/project/api-gateway/* .
        cp -r /var/Freedm/project/api-gateway/.* . 2>/dev/null || true

        # Remove node_modules if copied
        rm -rf node_modules/
```

1.3 Initial Commit

```
        git add .
        git commit -m "Initial API Gateway implementation with CI/CD pipeline"
        git push origin main
```

Phase 2: AWS Infrastructure Setup

2.1 Deploy Terraform Infrastructure

```
cd terraform/

# Initialize Terraform
terraform init

# Review the plan
terraform plan

# Apply infrastructure
terraform apply
```

Expected Resources Created: - ECR Repository: `freedmai-api-gateway` - IAM Role: `GitHubActionsRole-FreedmAI` - OIDC Provider for GitHub Actions

2.2 Note Important Outputs

```
# Save these values for GitHub secrets
terraform output ecr_repository_url
terraform output github_actions_role_arn
```

Phase 3: GitHub Configuration

3.1 Configure Repository Secrets

```
# Add AWS role ARN
gh secret set AWS_ROLE_ARN --body
"arn:aws:iam::ACCOUNT_ID:role/GitHubActionsRole-FreedmAI"

# Add ECR registry URL
gh secret set ECR_REGISTRY --body "ACCOUNT_ID.dkr.ecr.us-east-
1.amazonaws.com"
```

3.2 Configure Environments

```
# Create UAT environment
gh api repos/freedmai/api-gateway/environments/uat --method PUT

# Create Production environment with protection rules
gh api repos/freedmai/api-gateway/environments/production --method
PUT \
  --field
protection_rules='[{"type":"required_reviewers","reviewers":
[{"type":"Team","id":TEAM_ID}]}]'
```

3.3 Enable GitHub Actions

```
# Enable Actions if not already enabled
gh api repos/freedmai/api-gateway/actions/permissions --method PUT \
  --field enabled=true --field allowed_actions=all
```

Phase 4: EC2 Instance Setup

4.1 Launch EC2 Instances

UAT Instance (t3.small):

```
        # Launch UAT instance
        aws ec2 run-instances \
          --image-id ami-0c02fb55956c7d316 \
          --instance-type t3.small \
          --key-name your-key-pair \
          --security-group-ids sg-xxxxxxxxx \
          --subnet-id subnet-xxxxxxxxx \
          --tag-specifications 'ResourceType=instance,Tags=
[{Key=Name,Value=FreedmAI-UAT},{Key=Environment,Value=uat}]'
```

Production Instance (t3.small):

```
        # Launch Production instance
        aws ec2 run-instances \
          --image-id ami-0c02fb55956c7d316 \
          --instance-type t3.small \
          --key-name your-key-pair \
          --security-group-ids sg-xxxxxxxxx \
          --subnet-id subnet-xxxxxxxxx \
          --tag-specifications 'ResourceType=instance,Tags=
[{Key=Name,Value=FreedmAI-Prod},{Key=Environment,Value=production}]'
```

4.2 Configure EC2 Instances

For both UAT and Production instances:

```
        # SSH into instance
        ssh -i your-key.pem ec2-user@INSTANCE_IP

        # Update system
        sudo yum update -y

        # Install Docker
        sudo yum install -y docker
        sudo systemctl start docker
        sudo systemctl enable docker
        sudo usermod -a -G docker ec2-user

        # Install Docker Compose
        sudo curl -L
"https://github.com/docker/compose/releases/latest/download/docker-
compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose
        sudo chmod +x /usr/local/bin/docker-compose

        # Install AWS CLI v2
        curl "https://awscli.amazonaws.com/awscli-exe-linux-x86_64.zip" -o
"awscliv2.zip"
        unzip awscliv2.zip
        sudo ./aws/install

        # Install GitHub Actions Runner (UAT only)
        mkdir actions-runner && cd actions-runner
        curl -o actions-runner-linux-x64-2.311.0.tar.gz -L
https://github.com/actions/runner/releases/download/v2.311.0/actions-
runner-linux-x64-2.311.0.tar.gz
        tar xzf ./actions-runner-linux-x64-2.311.0.tar.gz

        # Configure runner (get token from GitHub)
        ./config.sh --url https://github.com/freedmai/api-gateway --token
YOUR_TOKEN

        # Install as service
        sudo ./svc.sh install
        sudo ./svc.sh start
```

4.3 Configure IAM Role for EC2

```
        # Create IAM role for EC2 instances
        aws iam create-role --role-name FreedmAI-EC2-Role --assume-role-
policy-document '{
          "Version": "2012-10-17",
          "Statement": [
            {
              "Effect": "Allow",
              "Principal": {
                "Service": "ec2.amazonaws.com"
              },
              "Action": "sts:AssumeRole"
            }
          ]
        }'

        # Attach ECR permissions
        aws iam attach-role-policy --role-name FreedmAI-EC2-Role --policy-
arn arn:aws:iam::aws:policy/AmazonEC2ContainerRegistryReadOnly

        # Create instance profile
        aws iam create-instance-profile --instance-profile-name FreedmAI-
EC2-Profile
        aws iam add-role-to-instance-profile --instance-profile-name
FreedmAI-EC2-Profile --role-name FreedmAI-EC2-Role

        # Attach to instances
        aws ec2 associate-iam-instance-profile --instance-id i-xxxxxxxxx --
iam-instance-profile Name=FreedmAI-EC2-Profile
```

Phase 5: Load Balancer Setup

5.1 Create Application Load Balancer

```
        # Create ALB
        aws elbv2 create-load-balancer \
          --name FreedmAI-ALB \
          --subnets subnet-xxxxxxxxx subnet-yyyyyyyyy \
          --security-groups sg-xxxxxxxxx \
          --scheme internet-facing \
          --type application \
          --ip-address-type ipv4
        # Create target groups
        aws elbv2 create-target-group \
          --name FreedmAI-UAT-TG \
          --protocol HTTP \
          --port 80 \
          --vpc-id vpc-xxxxxxxxx \
          --health-check-path /health

        aws elbv2 create-target-group \
          --name FreedmAI-Prod-TG \
          --protocol HTTP \
          --port 80 \
          --vpc-id vpc-xxxxxxxxx \
          --health-check-path /health

        # Register targets
        aws elbv2 register-targets --target-group-arn
arn:aws:elasticloadbalancing:us-east-1:ACCOUNT:targetgroup/FreedmAI-
UAT-TG/xxxxxxxxx --targets Id=i-xxxxxxxxx
        aws elbv2 register-targets --target-group-arn
arn:aws:elasticloadbalancing:us-east-1:ACCOUNT:targetgroup/FreedmAI-
Prod-TG/xxxxxxxxx --targets Id=i-yyyyyyyyy
```

5.2 Configure Listeners

```
        # Create listeners with path-based routing
        aws elbv2 create-listener \
          --load-balancer-arn arn:aws:elasticloadbalancing:us-east-
1:ACCOUNT:loadbalancer/app/FreedmAI-ALB/xxxxxxxxx \
          --protocol HTTP \
          --port 80 \
          --default-actions
Type=forward,TargetGroupArn=arn:aws:elasticloadbalancing:us-east-
1:ACCOUNT:targetgroup/FreedmAI-Prod-TG/xxxxxxxxx

        # Add rules for UAT routing
        aws elbv2 create-rule \
          --listener-arn arn:aws:elasticloadbalancing:us-east-
1:ACCOUNT:listener/app/FreedmAI-ALB/xxxxxxxxx/yyyyyyyyy \
          --priority 100 \
          --conditions Field=host-header,Values=uat.freedmai.com \
          --actions
Type=forward,TargetGroupArn=arn:aws:elasticloadbalancing:us-east-
1:ACCOUNT:targetgroup/FreedmAI-UAT-TG/xxxxxxxxx
```

Phase 6: Nginx Configuration

6.1 Create Nginx Configuration Files

On UAT Instance:

```
        sudo mkdir -p /opt/freedmai/nginx
        sudo tee /opt/freedmai/nginx/uat.conf > /dev/null <<EOF
        events {
            worker_connections 1024;
        }
        http {
            upstream api_gateway {
                server api-gateway:3000;
            }

            server {
                listen 80;
                server_name uat.freedmai.com;

                location /health {
                    proxy_pass http://api_gateway/health;
                    proxy_set_header Host \$host;
                    proxy_set_header X-Real-IP \$remote_addr;
                }

                location / {
                    proxy_pass http://api_gateway;
                    proxy_set_header Host \$host;
                    proxy_set_header X-Real-IP \$remote_addr;
                    proxy_set_header X-Forwarded-For
\$proxy_add_x_forwarded_for;
                    proxy_set_header X-Forwarded-Proto \$scheme;
                }
            }
        }
        EOF
```

On Production Instance:

```
        sudo mkdir -p /opt/freedmai/nginx
        sudo tee /opt/freedmai/nginx/prod.conf > /dev/null <<EOF
        events {
            worker_connections 1024;
        }

        http {
            upstream api_gateway {
                server api-gateway:3000;
            }

            server {
                listen 80;
                server_name api.freedmai.com;

                location /health {
                    proxy_pass http://api_gateway/health;
                    proxy_set_header Host \$host;
                    proxy_set_header X-Real-IP \$remote_addr;
                }

                location / {
                    proxy_pass http://api_gateway;
                    proxy_set_header Host \$host;
                    proxy_set_header X-Real-IP \$remote_addr;
                    proxy_set_header X-Forwarded-For
\$proxy_add_x_forwarded_for;
                    proxy_set_header X-Forwarded-Proto \$scheme;
                }
            }
        }
```

```
EOF
```

Phase 7: First Deployment

7.1 Trigger CI Pipeline

```
# Push code to trigger CI
git add .
git commit -m "Configure deployment settings"
git push origin main
```

Expected CI Pipeline Steps: 1. □ Code checkout 2. □ Node.js setup and dependency installation 3. □ ESLint code quality check 4. □ Jest unit tests 5. □ npm security audit 6. □ Trivy filesystem vulnerability scan 7. □ GitLeaks secret scan 8. □ Docker image build and push to ECR 9. □ Docker image vulnerability scan

7.2 Deploy to UAT

```
# Trigger UAT deployment via GitHub Actions
gh workflow run deploy-uat.yml -f image_tag=latest
```

Expected UAT Deployment Steps: 1. □ AWS credentials configuration 2. □ ECR login 3. □ Docker image pull 4. □ Container deployment with Docker Compose 5. □ Health check verification 6. □ Deployment notification

7.3 Verify UAT Deployment

```
# Check container status
ssh ec2-user@UAT_INSTANCE_IP "docker ps"

# Test health endpoint
curl http://uat.freedmai.com/health

# Check logs
ssh ec2-user@UAT_INSTANCE_IP "docker logs freedmai-api-gateway-uat"
```

Phase 8: Production Deployment

8.1 UAT Sign-off Process

1. **Manual Testing**: Verify all functionality in UAT
2. **Performance Testing**: Load test the UAT environment
3. **Security Validation**: Confirm security scans passed
4. **Stakeholder Approval**: Get approval from designated approvers

8.2 Deploy to Production

```
# Trigger production deployment (requires approval)
gh workflow run deploy-prod.yml -f image_tag=latest -f
uat_approval_id=12345
```

Expected Production Deployment Steps: 1. □ UAT approval validation 2. □ AWS credentials configuration 3. □ Current version backup 4. □ Blue-green deployment 5. □ Health check verification 6. □ Traffic switch 7. □ Old container cleanup 8. □ Rollback capability on failure

8.3 Verify Production Deployment

```
# Test production endpoint
curl http://api.freedmai.com/health

# Monitor logs
ssh ec2-user@PROD_INSTANCE_IP "docker logs freedmai-api-gateway-prod"
```

Phase 9: Monitoring Setup

9.1 CloudWatch Configuration

```
# Create log groups
aws logs create-log-group --log-group-name /freedmai/api-gateway/uat
aws logs create-log-group --log-group-name /freedmai/api-gateway/production

# Create custom metrics
aws cloudwatch put-metric-data \
  --namespace "FreedmAI/APIGateway" \
  --metric-data MetricName=HealthCheck,Value=1,Unit=Count
```

9.2 Alerting Setup

```
# Create SNS topic
aws sns create-topic --name FreedmAI-Alerts

# Create CloudWatch alarms
aws cloudwatch put-metric-alarm \
  --alarm-name "API-Gateway-High-Error-Rate" \
  --alarm-description "API Gateway error rate too high" \
  --metric-name ErrorRate \
  --namespace FreedmAI/APIGateway \
  --statistic Average \
  --period 300 \
  --threshold 5.0 \
  --comparison-operator GreaterThanThreshold \
  --evaluation-periods 2
```

Phase 10: Security Hardening

10.1 Security Group Configuration

```
# Create security group for API Gateway
aws ec2 create-security-group \
  --group-name FreedmAI-API-Gateway-SG \
  --description "Security group for FreedmAI API Gateway"
```

```
# Allow HTTP traffic from ALB only
aws ec2 authorize-security-group-ingress \
  --group-id sg-xxxxxxxxx \
  --protocol tcp \
  --port 80 \
  --source-group sg-yyyyyyyyy
```

10.2 WAF Configuration

```
# Create WAF Web ACL
aws wafv2 create-web-acl \
  --name FreedmAI-WAF \
  --scope REGIONAL \
  --default-action Allow={} \
  --rules file://waf-rules.json
```

Phase 11: Cost Optimization

11.1 Resource Monitoring

```
# Set up cost alerts
aws budgets create-budget \
  --account-id ACCOUNT_ID \
  --budget file://budget-config.json
```

11.2 Lifecycle Policies

```
# ECR lifecycle policy
aws ecr put-lifecycle-policy \
  --repository-name freedmai-api-gateway \
  --lifecycle-policy-text file://ecr-lifecycle.json
```

Verification Checklist

□ Infrastructure

- [ ] ECR repository created and accessible
- [ ] IAM roles and policies configured
- [ ] EC2 instances running and accessible
- [ ] Load balancer configured with health checks
- [ ] Security groups properly configured

□ CI/CD Pipeline

- [ ] GitHub Actions workflows executing successfully
- [ ] Security scans passing (Trivy, GitLeaks, npm audit)
- [ ] Code quality checks passing (ESLint, tests)
- [ ] Docker images building and pushing to ECR
- [ ] Self-hosted runner operational on UAT instance

▯ Deployment

- ☐ UAT deployment working via GitHub Actions
- ☐ Production deployment with approval gates
- ☐ Blue-green deployment strategy functional
- ☐ Rollback capability tested
- ☐ Health checks responding correctly

▯ Monitoring

- ☐ CloudWatch logs streaming
- ☐ Custom metrics being recorded
- ☐ Alerts configured and tested
- ☐ Cost monitoring active

▯ Security

- ☐ No secrets in code repository
- ☐ OIDC authentication working
- ☐ Container vulnerability scans passing
- ☐ Network security properly configured
- ☐ WAF rules active (if implemented)

Troubleshooting Guide

Common Issues

1. GitHub Actions failing with AWS permissions:

```
# Verify OIDC trust relationship
aws iam get-role --role-name GitHubActionsRole-FreedmAI
```

2. Docker container not starting:

```
# Check container logs
docker logs freedmai-api-gateway-uat
# Verify environment variables
docker exec freedmai-api-gateway-uat env
```

3. Health check failing:

```
# Test directly on instance
curl http://localhost:3000/health
# Check application logs
docker logs freedmai-api-gateway-uat
```

4. ECR push failing:

```
# Re-authenticate with ECR
aws ecr get-login-password --region us-east-1 | docker login --username AWS --password-stdin ACCOUNT.dkr.ecr.us-east-1.amazonaws.com
```

Next Steps

1. **Add More Microservices**: Replicate this pattern for other services
2. **Implement Deployment UI**: Create web interface for deployments
3. **Add Database Integration**: Set up RDS for shared configuration
4. **Enhance Monitoring**: Add application performance monitoring
5. **Implement Service Mesh**: Consider Istio or AWS App Mesh for advanced routing

Cost Summary

Monthly Estimated Costs: - EC2 instances (2x t3.small): ~$30 - ALB: ~$16 - ECR storage: ~$2 - CloudWatch logs: ~$5 - Data transfer: ~$5 - **Total: ~$58/month**

Support Contacts

- **DevOps Team**: devops@freedmai.com
- **Security Team**: security@freedmai.com
- **AWS Support**: Use AWS Support Center

Document Version: 1.0
Date: September 19, 2025
Implementation Status: Complete
Next Review: October 19, 2025