Architecture & AWS Deployment Guide

# **CAS Website - Complete Architecture & AWS Deployment Guide**

## **Table of Contents**

1. [Current Architecture Overview](https://www.google.com/search?q=%23current-architecture-overview)
2. [Technology Stack](https://www.google.com/search?q=%23technology-stack)
3. [Application Components](https://www.google.com/search?q=%23application-components)
4. [Data Flow](https://www.google.com/search?q=%23data-flow)
5. [AWS Deployment Architecture](https://www.google.com/search?q=%23aws-deployment-architecture)
6. [Deployment Process](https://www.google.com/search?q=%23deployment-process)
7. [GitHub Bridge Deployment (Recommended)](https://www.google.com/search?q=%23github-bridge-deployment-recommended)
8. [Environment Configuration](https://www.google.com/search?q=%23environment-configuration)
9. [Monitoring & Maintenance](https://www.google.com/search?q=%23monitoring--maintenance)

## **Current Architecture Overview**

The Canadian Amyloidosis Society (CAS) website is a full-stack web application that serves as a platform for:

* Healthcare professional registration (CAS & CANN membership)
* Educational resources and events
* Healthcare facility directory with interactive map
* Zoho CRM integration for lead management

### 

### 

### 

### 

### 

### 

### 

### 

### 

### 

### **High-Level Architecture Diagram**

┌─────────────────────────────────────────────────────────────────────────────┐

│ USER BROWSER │

│ (Desktop / Mobile) │

└─────────────────────────────────┬───────────────────────────────────────────┘

│ HTTPS (Port 5000)

▼

┌─────────────────────────────────────────────────────────────────────────────┐

│ EXPRESS.JS SERVER │

│ ┌─────────────────────────────────────────────────────────────────────┐ │

│ │ FRONTEND (React + Vite) │ │

│ │ • Single Page Application │ │

│ │ • Tailwind CSS + shadcn/ui components │ │

│ │ • Wouter for client-side routing │ │

│ │ • TanStack Query for data fetching │ │

│ └─────────────────────────────────────────────────────────────────────┘ │

│ │ │

│ ┌─────────────────────────────────▼───────────────────────────────────┐ │

│ │ BACKEND API (/api/\*) │ │

│ │ • RESTful endpoints for forms, resources, events │ │

│ │ • Session management (connect-pg-simple) │ │

│ │ • Form validation with Zod schemas │ │

│ └─────────────────────────────────────────────────────────────────────┘ │

│ │ │

│ ┌─────────────────────────────────▼───────────────────────────────────┐ │

│ │ BACKGROUND WORKERS │ │

│ │ • TokenManager: OAuth token refresh (Zoho) │ │

│ │ • ZohoSyncWorker: Async form submission sync │ │

│ │ • FieldMetadataCache: CRM field caching │ │

│ └─────────────────────────────────────────────────────────────────────┘ │

└─────────────────────────────────────────────────────────────────────────────┘

│

┌───────────────────────┼───────────────────────┐

│ │ │

▼ ▼ ▼

┌─────────────────┐ ┌─────────────────┐ ┌─────────────────┐

│ NEON POSTGRES │ │ ZOHO CRM API │ │ FILE STORAGE │

│ (Database) │ │ (External) │ │ (Local/Assets) │

│ │ │ │ │ │

│ • form\_submissions│ │ • Leads module │ │ • Uploaded docs │

│ • oauth\_tokens │ │ • Accounts │ │ • Stock images │

│ • resources │ │ • Custom fields │ │ • PDF resources │

│ • event\_registrations│ OAuth v2 auth │ │ │

└─────────────────┘ └─────────────────┘ └─────────────────┘

## 

## 

## 

## **Technology Stack**

### **Frontend**

| **Technology** | **Purpose** |
| --- | --- |
| React 18 | UI framework |
| Vite | Build tool & dev server |
| TypeScript | Type safety |
| Tailwind CSS | Utility-first styling |
| shadcn/ui | UI component library |
| Radix UI | Accessible primitives |
| Wouter | Client-side routing |
| TanStack Query | Server state management |
| Framer Motion | Animations |
| Lucide React | Icons |

### **Backend**

| **Technology** | **Purpose** |
| --- | --- |
| Node.js | Runtime environment |
| Express.js | Web framework |
| TypeScript | Type safety |
| Drizzle ORM | Database queries |
| Zod | Schema validation |
| Passport | Authentication |
| express-session | Session management |

### **Database**

| **Technology** | **Purpose** |
| --- | --- |
| PostgreSQL | Primary database |
| Neon | Serverless Postgres hosting |
| Drizzle Kit | Schema migrations |

### **External Integrations**

| **Service** | **Purpose** |
| --- | --- |
| Zoho CRM | Lead management, contact storage |
| Nodemailer | Email notifications (pending SMTP) |

## **Application Components**

### **1. Frontend Pages**

/ → Homepage with interactive map

/about-amyloidosis → Disease information

/about-cas → Organization information

/join-cas → Unified registration form

/cann → CANN network page

/events → Events listing

/events/cann-townhall/register → Event registration

/clinical-tools → Resources for clinicians

/upload-resource → Resource submission portal

/admin/resources/moderation → Resource moderation (admin)

/eventsdownload → Event registrations export (admin)

### **2. API Endpoints**

POST /api/form-submissions → Submit any form

GET /api/form-configs → Get form configurations

GET /api/form-configs/:formName → Get specific form config

POST /api/event-registrations → Register for event

GET /api/event-registrations → List registrations (admin)

POST /api/resources → Submit resource

GET /api/resources → List resources

PATCH /api/resources/:id/status → Approve/reject resource

GET /api/health → Health check

GET /api/ping → Simple ping

### **3. Background Services**

**TokenManager** (server/token-manager.ts)

* Manages OAuth tokens for Zoho CRM
* Auto-refreshes tokens before expiry
* Health monitoring every 3.3 minutes

**ZohoSyncWorker** (server/zoho-sync-worker.ts)

* Polls for pending form submissions every 10 seconds
* Syncs to Zoho CRM with retry logic
* Exponential backoff on failures

**FieldMetadataCache** (server/field-metadata-cache-service.ts)

* Caches Zoho CRM field definitions
* Daily sync to keep field mappings current

## **Data Flow**

### **Form Submission Flow**

┌──────────┐ ┌──────────┐ ┌──────────┐ ┌──────────┐ ┌──────────┐

│ User │───▶│ React │───▶│ Express │───▶│ Postgres │───▶│ Zoho │

│ Browser │ │ Form │ │ API │ │ (save) │ │ CRM │

└──────────┘ └──────────┘ └──────────┘ └──────────┘ └──────────┘

│ ▲

│ Background │

└──────────Worker──────────────┘

(async sync)

1. **User submits form** → React form with validation
2. **API receives data** → Express validates with Zod
3. **Save to PostgreSQL** → Immediate local storage (bulletproof)
4. **Return success** → User sees confirmation
5. **Background sync** → Worker syncs to Zoho CRM asynchronously
6. **Retry on failure** → Exponential backoff ensures delivery

### **Zoho CRM Data Structure**

**Leads Module (249 records)**

Lead\_Source options:

├── Excel Import - CAS Registration (Historical) [63]

├── Website - CAS Registration [62]

├── Excel Import - CAS Registration (2025) [54]

├── Excel Import - CAS Registration (French 2025) [21]

├── Excel Import - PANN Membership (Historical) [21]

├── Website - CAS & CANN Registration [17]

├── Website - PANN Membership [5]

├── Website - Join CAS Today (Historical) [5]

└── Website - CANN Membership [1]

Key Fields:

├── Professional\_Designation (text)

├── Institution\_Name (text)

├── Sub\_Specialty (text)

├── CAS\_Member (boolean)

├── PANN\_Member (boolean)

├── CAS\_Communications (picklist: Yes/No)

└── Services\_Map\_Inclusion (picklist: Yes/No)

**Accounts Module (101 records)**

* Healthcare centers across Canada
* Used for Services Map display

## 

## 

## 

## 

## 

## 

## **AWS Deployment Architecture**

### **Target AWS Infrastructure**

┌─────────────────────────────────────────────────────────────────────────────┐

│ INTERNET │

└─────────────────────────────────┬───────────────────────────────────────────┘

│

┌────────▼────────┐

│ Route 53 │ DNS: amyloid.ca

│ (DNS Zone) │ \*.amyloid.ca

└────────┬────────┘

│

┌────────▼────────┐

│ ACM Certificate│ SSL/TLS termination

│ (SSL/HTTPS) │ Auto-renewal

└────────┬────────┘

│

┌────────▼────────┐

│ Application │ Port 443 (HTTPS)

│ Load Balancer │ Health: /health

└────────┬────────┘

│

┌────────────────────────┼────────────────────────┐

│ │ │

▼ ▼ ▼

┌─────────────────┐ ┌─────────────────┐ ┌─────────────────┐

│ ECS Task 1 │ │ ECS Task 2 │ │ ECS Task N │

│ (Container) │ │ (Container) │ │ (Container) │

│ │ │ │ │ │

│ Port 5000 │ │ Port 5000 │ │ Port 5000 │

└────────┬────────┘ └────────┬────────┘ └────────┬────────┘

│ │ │

└──────────────────────┼──────────────────────┘

│

┌─────────────────┼─────────────────┐

│ │ │

▼ ▼ ▼

┌─────────────┐ ┌─────────────┐ ┌─────────────┐

│ Neon │ │ Secrets │ │ CloudWatch │

│ PostgreSQL │ │ Manager │ │ Logs │

│ (External) │ │ (ENV vars) │ │ (Monitoring)│

└─────────────┘ └─────────────┘ └─────────────┘

### 

### **AWS Services Required**

| **Service** | **Purpose** | **Estimated Cost** |
| --- | --- | --- |
| **ECR** | Docker image registry | ~$0.10/GB/month |
| **ECS Fargate** | Container hosting | ~$30-50/month (1 vCPU, 2GB) |
| **Application Load Balancer** | Traffic distribution | ~$20/month + data |
| **Route 53** | DNS hosting | ~$0.50/month |
| **ACM** | SSL certificates | Free |
| **Secrets Manager** | Secure credentials | ~$0.40/secret/month |
| **CloudWatch** | Logging & monitoring | ~$5-10/month |
| **VPC** | Network isolation | Free |

**Estimated Total: ~$60-100/month** (depending on traffic)

## 

## 

## **Deployment Process**

### **"Republish" Workflow**

┌───────────────────────────────────────────────────────────────────────────┐

│ REPLIT (Development) │

│ │

│ 1. Developer makes changes │

│ 2. Tests locally on Replit │

│ 3. Clicks "Republish" button │

└─────────────────────────────────┬─────────────────────────────────────────┘

│

▼

┌───────────────────────────────────────────────────────────────────────────┐

│ BUILD PHASE │

│ │

│ 1. npm run build (Vite frontend) │

│ 2. esbuild (Bundle server) │

│ 3. docker build (Create image) │

│ 4. docker push (Upload to ECR) │

└─────────────────────────────────┬─────────────────────────────────────────┘

│

▼

┌───────────────────────────────────────────────────────────────────────────┐

│ DEPLOY PHASE │

│ │

│ 1. aws ecs update-service --force-new-deployment │

│ 2. ECS pulls new image from ECR │

│ 3. Rolling deployment (zero downtime) │

│ 4. Old containers drain, new containers start │

│ 5. Health checks pass → traffic shifts │

└───────────────────────────────────────────────────────────────────────────┘

### **Dockerfile (Multi-stage Build)**

# Stage 1: Build frontend

FROM node:20-alpine AS frontend-builder

WORKDIR /app

COPY package\*.json ./

RUN npm ci

COPY . .

RUN npm run build

# Stage 2: Build backend

FROM node:20-alpine AS backend-builder

WORKDIR /app

COPY package\*.json ./

RUN npm ci --production

COPY --from=frontend-builder /app/dist ./dist

COPY server ./server

COPY shared ./shared

RUN npx esbuild server/index.ts --bundle --platform=node --outfile=dist/server.js

# Stage 3: Production runtime

FROM node:20-alpine AS production

WORKDIR /app

COPY --from=backend-builder /app/dist ./dist

COPY --from=backend-builder /app/node\_modules ./node\_modules

COPY package.json ./

ENV NODE\_ENV=production

ENV PORT=5000

EXPOSE 5000

HEALTHCHECK --interval=30s --timeout=5s --start-period=10s \

CMD wget -q --spider http://localhost:5000/health || exit 1

CMD ["node", "dist/server.js"]

### **ECS Task Definition**

{

"family": "cas-website",

"networkMode": "awsvpc",

"requiresCompatibilities": ["FARGATE"],

"cpu": "1024",

"memory": "2048",

"containerDefinitions": [

{

"name": "cas-app",

"image": "[123456789.dkr.ecr.us-east-1.amazonaws.com/cas-website:latest](https://123456789.dkr.ecr.us-east-1.amazonaws.com/cas-website:latest)",

"portMappings": [

{

"containerPort": 5000,

"protocol": "tcp"

}

],

"environment": [

{"name": "NODE\_ENV", "value": "production"},

{"name": "PORT", "value": "5000"}

],

"secrets": [

{

"name": "DATABASE\_URL",

"valueFrom": "arn:aws:secretsmanager:us-east-1:123456789:secret:cas/database-url"

},

{

"name": "ZOHO\_CLIENT\_ID",

"valueFrom": "arn:aws:secretsmanager:us-east-1:123456789:secret:cas/zoho-client-id"

},

{

"name": "ZOHO\_CLIENT\_SECRET",

"valueFrom": "arn:aws:secretsmanager:us-east-1:123456789:secret:cas/zoho-client-secret"

}

],

"logConfiguration": {

"logDriver": "awslogs",

"options": {

"awslogs-group": "/ecs/cas-website",

"awslogs-region": "us-east-1",

"awslogs-stream-prefix": "ecs"

}

},

"healthCheck": {

"command": ["CMD-SHELL", "wget -q --spider http://localhost:5000/health || exit 1"],

"interval": 30,

"timeout": 5,

"retries": 3,

"startPeriod": 60

}

}

]

}

### 

### **Republish Script (scripts/deploy-aws.sh)**

#!/bin/bash

set -e

# Configuration

AWS\_REGION="us-east-1"

ECR\_REPOSITORY="cas-website"

ECS\_CLUSTER="cas-production"

ECS\_SERVICE="cas-website-service"

AWS\_ACCOUNT\_ID="123456789012"

echo "Starting AWS deployment..."

# Step 1: Build frontend

echo "Building frontend..."

npm run build

# Step 2: Login to ECR

echo "Logging into ECR..."

aws ecr get-login-password --region $AWS\_REGION | \

docker login --username AWS --password-stdin $AWS\_ACCOUNT\_ID.dkr.ecr.$AWS\_REGION.amazonaws.com

# Step 3: Build Docker image

echo "Building Docker image..."

docker build -t $ECR\_REPOSITORY:latest .

docker tag $ECR\_REPOSITORY:latest $AWS\_ACCOUNT\_ID.dkr.ecr.$AWS\_[REGION.amazonaws.com/$ECR\_REPOSITORY:latest](https://REGION.amazonaws.com/$ECR\_REPOSITORY:latest)

# Step 4: Push to ECR

echo "Pushing to ECR..."

docker push $AWS\_ACCOUNT\_ID.dkr.ecr.$AWS\_[REGION.amazonaws.com/$ECR\_REPOSITORY:latest](https://REGION.amazonaws.com/$ECR\_REPOSITORY:latest)

# Step 5: Update ECS service

echo "Updating ECS service..."

aws ecs update-service \

--cluster $ECS\_CLUSTER \

--service $ECS\_SERVICE \

--force-new-deployment \

--region $AWS\_REGION

echo "Deployment initiated! Monitor progress in AWS Console."

echo "Link: [https://console.aws.amazon.com/ecs/home?region=$AWS\_REGION#/clusters/$ECS\_CLUSTER/services/$ECS\_SERVICE/deployments](https://console.aws.amazon.com/ecs/home?region=$AWS\_REGION#/clusters/$ECS\_CLUSTER/services/$ECS\_SERVICE/deployments)"

## **GitHub Bridge Deployment (Recommended)**

Using GitHub as a bridge between Replit and AWS provides version control, automated deployments, and easier collaboration. This is the **recommended approach** for production deployments.

### **GitHub Bridge Architecture**

┌───────────────────────────────────────────────────────────────────────────┐

│ REPLIT (Development) │

│ │

│ 1. Developer makes changes │

│ 2. Tests locally on Replit │

│ 3. Commits and pushes to GitHub │

└─────────────────────────────────┬─────────────────────────────────────────┘

│ git push

▼

┌───────────────────────────────────────────────────────────────────────────┐

│ GITHUB REPOSITORY │

│ │

│ • Version control & history │

│ • Branch protection rules │

│ • Pull request reviews │

│ • Triggers GitHub Actions on push │

└─────────────────────────────────┬─────────────────────────────────────────┘

│ GitHub Actions (automatic)

▼

┌───────────────────────────────────────────────────────────────────────────┐

│ GITHUB ACTIONS WORKFLOW │

│ │

│ 1. Checkout code │

│ 2. Build frontend (npm run build) │

│ 3. Build Docker image │

│ 4. Push to AWS ECR │

│ 5. Deploy to ECS (rolling update) │

└─────────────────────────────────┬─────────────────────────────────────────┘

│

▼

┌───────────────────────────────────────────────────────────────────────────┐

│ AWS ECS │

│ │

│ • Pulls new image from ECR │

│ • Rolling deployment (zero downtime) │

│ • Health checks pass → traffic shifts │

└───────────────────────────────────────────────────────────────────────────┘

### **Benefits of GitHub Bridge**

| **Feature** | **Direct Deploy** | **GitHub Bridge** |
| --- | --- | --- |
| Version History | Manual tags | Full git history |
| Rollback | Re-deploy old image | git revert + auto-deploy |
| Code Review | None | Pull requests |
| Audit Trail | CloudWatch only | Git commits + Actions logs |
| Team Collaboration | Share AWS creds | GitHub permissions |
| Automation | Manual script | Fully automatic |

### **Step 1: Connect Replit to GitHub**

1. In Replit, click **Git** in the sidebar (branch icon)
2. Click **Connect to GitHub**
3. Authorize Replit to access your GitHub account
4. Create a new repository: cas-website
5. Push your code: git push -u origin main

### **Step 2: Set Up GitHub Repository Secrets**

Go to your GitHub repository → **Settings** → **Secrets and variables** → **Actions**

Add these secrets:

| **Secret Name** | **Description** | **How to Get It** |
| --- | --- | --- |
| AWS\_ACCESS\_KEY\_ID | AWS IAM access key | AWS Console → IAM → Users → Security credentials |
| AWS\_SECRET\_ACCESS\_KEY | AWS IAM secret key | Generated with access key |
| AWS\_REGION | AWS region (e.g., us-east-1) | Your chosen region |
| AWS\_ACCOUNT\_ID | 12-digit AWS account ID | AWS Console → top right menu |
| ECR\_REPOSITORY | ECR repo name (cas-website) | From ECR setup |
| ECS\_CLUSTER | ECS cluster name (cas-production) | From ECS setup |
| ECS\_SERVICE | ECS service name (cas-website-service) | From ECS setup |

### **Step 3: Create GitHub Actions Workflow**

Create the file .github/workflows/deploy.yml in your repository:

name: Deploy to AWS ECS

on:

push:

branches:

- main

workflow\_dispatch:

env:

AWS\_REGION: ${{ secrets.AWS\_REGION }}

ECR\_REPOSITORY: ${{ secrets.ECR\_REPOSITORY }}

ECS\_CLUSTER: ${{ secrets.ECS\_CLUSTER }}

ECS\_SERVICE: ${{ secrets.ECS\_SERVICE }}

jobs:

deploy:

name: Build and Deploy

runs-on: ubuntu-latest

steps:

- name: Checkout code

uses: actions/checkout@v4

- name: Configure AWS credentials

uses: aws-actions/configure-aws-credentials@v4

with:

aws-access-key-id: ${{ secrets.AWS\_ACCESS\_KEY\_ID }}

aws-secret-access-key: ${{ secrets.AWS\_SECRET\_ACCESS\_KEY }}

aws-region: ${{ secrets.AWS\_REGION }}

- name: Login to Amazon ECR

id: login-ecr

uses: aws-actions/amazon-ecr-login@v2

- name: Build, tag, and push Docker image

id: build-image

env:

ECR\_REGISTRY: ${{ steps.login-ecr.outputs.registry }}

IMAGE\_TAG: ${{ github.sha }}

run: |

docker build -t $ECR\_REGISTRY/$ECR\_REPOSITORY:$IMAGE\_TAG .

docker build -t $ECR\_REGISTRY/$ECR\_REPOSITORY:latest .

docker push $ECR\_REGISTRY/$ECR\_REPOSITORY:$IMAGE\_TAG

docker push $ECR\_REGISTRY/$ECR\_REPOSITORY:latest

echo "image=$ECR\_REGISTRY/$ECR\_REPOSITORY:$IMAGE\_TAG" >> $GITHUB\_OUTPUT

- name: Deploy to ECS

run: |

aws ecs update-service \

--cluster $ECS\_CLUSTER \

--service $ECS\_SERVICE \

--force-new-deployment

- name: Wait for deployment

run: |

echo "Waiting for ECS deployment to stabilize..."

aws ecs wait services-stable \

--cluster $ECS\_CLUSTER \

--services $ECS\_SERVICE

echo "Deployment complete!"

- name: Deployment Summary

run: |

echo "## Deployment Summary" >> $GITHUB\_STEP\_SUMMARY

echo "- \*\*Image:\*\* ${{ steps.build-image.outputs.image }}" >> $GITHUB\_STEP\_SUMMARY

echo "- \*\*Cluster:\*\* $ECS\_CLUSTER" >> $GITHUB\_STEP\_SUMMARY

echo "- \*\*Service:\*\* $ECS\_SERVICE" >> $GITHUB\_STEP\_SUMMARY

echo "- \*\*Status:\*\* Deployed successfully" >> $GITHUB\_STEP\_SUMMARY

### **Step 4: Deployment Workflow**

After setup, deployments are fully automatic:

Developer Experience:

1. Make changes in Replit

2. Test locally (workflow running)

3. Commit changes:

$ git add .

$ git commit -m "Add new feature"

$ git push

4. GitHub Actions automatically:

→ Builds Docker image

→ Pushes to ECR

→ Deploys to ECS

→ Verifies health

5. Done! Live in ~5-10 minutes

### **Optional: Branch Protection**

For team safety, add branch protection rules:

1. Go to **Settings** → **Branches** → **Add rule**
2. Branch name pattern: main
3. Enable:
   * Require pull request before merging
   * Require status checks to pass
   * Require branches to be up to date

### **Optional: Staging Environment**

Add a staging environment that deploys from develop branch:

# .github/workflows/deploy-staging.yml

name: Deploy to Staging

on:

push:

branches:

- develop

env:

AWS\_REGION: ${{ secrets.AWS\_REGION }}

ECR\_REPOSITORY: ${{ secrets.ECR\_REPOSITORY }}

ECS\_CLUSTER: cas-staging # Different cluster

ECS\_SERVICE: cas-website-staging

jobs:

deploy:

# Same steps as production...

### **Rollback via GitHub**

To rollback to a previous version:

# Option 1: Revert the bad commit

git revert HEAD

git push

# → Automatic re-deploy with previous code

# Option 2: Deploy specific commit

# Go to Actions → Run workflow → Select branch/tag

# Option 3: AWS Console

# ECS → Service → Update → Select previous task definition

### **Comparison: Deployment Methods**

| **Method** | **When to Use** | **Complexity** |
| --- | --- | --- |
| **GitHub Actions (Recommended)** | Production deployments | Medium (one-time setup) |
| **Manual Script** | Testing, emergency fixes | Low |
| **Terraform + GitHub** | Full infrastructure as code | High |

## **Environment Configuration**

### **Required Secrets (AWS Secrets Manager)**

| **Secret Name** | **Description** |
| --- | --- |
| cas/database-url | PostgreSQL connection string |
| cas/zoho-client-id | Zoho CRM OAuth client ID |
| cas/zoho-client-secret | Zoho CRM OAuth client secret |
| cas/zoho-refresh-token | Zoho CRM refresh token |
| cas/session-secret | Express session secret |
| cas/event-admin-password | Event admin dashboard password |

### **Environment Variables**

| **Variable** | **Value** | **Description** |
| --- | --- | --- |
| NODE\_ENV | production | Runtime environment |
| PORT | 5000 | Server port |
| REPLIT\_DEPLOYMENT | 1 | Signals production mode |

### **Current Secrets (from Replit)**

These need to be migrated to AWS Secrets Manager:

DATABASE\_URL → cas/database-url

ZOHO\_CLIENT\_ID → cas/zoho-client-id

ZOHO\_CLIENT\_SECRET → cas/zoho-client-secret

SESSION\_SECRET → cas/session-secret

EVENT\_ADMIN\_USERNAME → cas/event-admin-username

EVENT\_ADMIN\_PASSWORD → cas/event-admin-password

## **Monitoring & Maintenance**

### **Health Endpoints**

| **Endpoint** | **Purpose** | **Response** |
| --- | --- | --- |
| GET /health | Full health check | JSON with DB, Zoho status |
| GET /ping | Simple ping | "pong" |

### **CloudWatch Alarms (Recommended)**

| **Alarm** | **Threshold** | **Action** |
| --- | --- | --- |
| High CPU | >80% for 5 min | Scale up / Alert |
| Memory Usage | >85% | Alert |
| 5xx Errors | >10/min | Alert |
| Health Check Failures | >3 consecutive | Replace container |
| Response Time | >2s avg | Alert |

### **Log Groups**

/ecs/cas-website → Application logs

/ecs/cas-website/access → HTTP access logs

/aws/alb/cas-website → Load balancer logs

### **Backup Strategy**

| **Component** | **Strategy** | **Frequency** |
| --- | --- | --- |
| Database (Neon) | Point-in-time recovery | Continuous |
| Docker Images | ECR retention | Keep last 10 |
| Secrets | AWS backup | Daily |

## **Infrastructure as Code (Terraform)**

### **Main Configuration (infrastructure/main.tf)**

provider "aws" {

region = "us-east-1"

}

# VPC

module "vpc" {

source = "terraform-aws-modules/vpc/aws"

version = "5.0.0"

name = "cas-vpc"

cidr = "10.0.0.0/16"

azs = ["us-east-1a", "us-east-1b"]

private\_subnets = ["10.0.1.0/24", "10.0.2.0/24"]

public\_subnets = ["10.0.101.0/24", "10.0.102.0/24"]

enable\_nat\_gateway = true

single\_nat\_gateway = true

}

# ECR Repository

resource "aws\_ecr\_repository" "cas\_website" {

name = "cas-website"

image\_tag\_mutability = "MUTABLE"

image\_scanning\_configuration {

scan\_on\_push = true

}

}

# ECS Cluster

resource "aws\_ecs\_cluster" "cas\_production" {

name = "cas-production"

setting {

name = "containerInsights"

value = "enabled"

}

}

# Application Load Balancer

resource "aws\_lb" "cas\_alb" {

name = "cas-alb"

internal = false

load\_balancer\_type = "application"

security\_groups = [aws\_security\_group.alb\_sg.id]

subnets = module.vpc.public\_subnets

}

resource "aws\_lb\_target\_group" "cas\_tg" {

name = "cas-tg"

port = 5000

protocol = "HTTP"

vpc\_id = module.vpc.vpc\_id

target\_type = "ip"

health\_check {

enabled = true

healthy\_threshold = 2

interval = 30

matcher = "200"

path = "/health"

port = "traffic-port"

protocol = "HTTP"

timeout = 5

unhealthy\_threshold = 3

}

}

# HTTPS Listener

resource "aws\_lb\_listener" "https" {

load\_balancer\_arn = aws\_lb.cas\_alb.arn

port = "443"

protocol = "HTTPS"

ssl\_policy = "ELBSecurityPolicy-TLS-1-2-2017-01"

certificate\_arn = aws\_acm\_certificate.cas\_cert.arn

default\_action {

type = "forward"

target\_group\_arn = aws\_lb\_target\_group.cas\_tg.arn

}

}

# ECS Service

resource "aws\_ecs\_service" "cas\_service" {

name = "cas-website-service"

cluster = aws\_ecs\_cluster.cas\_production.id

task\_definition = aws\_ecs\_task\_definition.cas\_task.arn

desired\_count = 2

launch\_type = "FARGATE"

network\_configuration {

subnets = module.vpc.private\_subnets

security\_groups = [aws\_security\_group.ecs\_sg.id]

assign\_public\_ip = false

}

load\_balancer {

target\_group\_arn = aws\_lb\_target\_group.cas\_tg.arn

container\_name = "cas-app"

container\_port = 5000

}

deployment\_configuration {

maximum\_percent = 200

minimum\_healthy\_percent = 100

}

}

## **Migration Checklist**

### **Phase 1: Preparation**

* [ ] Create AWS account (if not exists)
* [ ] Install AWS CLI and configure credentials
* [ ] Install Docker locally for testing
* [ ] Create ECR repository

### **Phase 2: Infrastructure Setup**

* [ ] Create VPC with public/private subnets
* [ ] Set up security groups
* [ ] Create Application Load Balancer
* [ ] Configure Route 53 hosted zone
* [ ] Request ACM certificate for domain
* [ ] Create ECS cluster

### **Phase 3: Secrets & Configuration**

* [ ] Create secrets in AWS Secrets Manager
* [ ] Copy all environment variables from Replit
* [ ] Create IAM roles for ECS tasks
* [ ] Set up CloudWatch log groups

### **Phase 4: GitHub Bridge Setup**

* [ ] Connect Replit to GitHub
* [ ] Create GitHub repository
* [ ] Push code to GitHub
* [ ] Add AWS credentials as GitHub Secrets
* [ ] Create .github/workflows/deploy.yml
* [ ] Test workflow with manual trigger

### **Phase 5: Build & Deploy**

* [ ] Create Dockerfile in repository
* [ ] Test Docker build locally
* [ ] Push to main branch (triggers deployment)
* [ ] Verify GitHub Actions workflow succeeds
* [ ] Verify ECS service health checks

### **Phase 6: DNS & Go Live**

* [ ] Update DNS to point to ALB
* [ ] Verify SSL certificate
* [ ] Test all functionality
* [ ] Monitor for 24-48 hours
* [ ] Decommission Replit hosting

## **Summary**

### **Current State (Replit)**

* Development environment with hot reload
* PostgreSQL on Neon (external)
* Zoho CRM integration working
* 350 total records in CRM (249 leads + 101 accounts)

### **Target State (AWS)**

* Production-grade container hosting
* Zero-downtime deployments
* Auto-scaling capability
* Professional monitoring & logging
* ~$60-100/month estimated cost

### **Key Benefits of AWS Migration**

1. **Reliability**: Multi-AZ deployment with auto-recovery
2. **Scalability**: Easy to scale up during traffic spikes
3. **Security**: VPC isolation, IAM roles, encrypted secrets
4. **Monitoring**: CloudWatch metrics, logs, and alarms
5. **Control**: Full infrastructure ownership

AWS DevOps Runbook

# **CAS Website - AWS DevOps Runbook**

## **Table of Contents**

1. [Quick Reference](https://www.google.com/search?q=%23quick-reference)
2. [Application Overview](https://www.google.com/search?q=%23application-overview)
3. [AWS Resource Inventory](https://www.google.com/search?q=%23aws-resource-inventory)
4. [Infrastructure Setup](https://www.google.com/search?q=%23infrastructure-setup)
5. [IAM Configuration](https://www.google.com/search?q=%23iam-configuration)
6. [Secrets Configuration](https://www.google.com/search?q=%23secrets-configuration)
7. [Container Configuration](https://www.google.com/search?q=%23container-configuration)
8. [Health Checks & Monitoring](https://www.google.com/search?q=%23health-checks--monitoring)
9. [GitHub Actions Setup](https://www.google.com/search?q=%23github-actions-setup)
10. [Operational Runbook](https://www.google.com/search?q=%23operational-runbook)
11. [Troubleshooting Guide](https://www.google.com/search?q=%23troubleshooting-guide)

## **Quick Reference**

### **Critical Endpoints**

| **Endpoint** | **Purpose** | **Expected Response** |
| --- | --- | --- |
| GET /health | Application health | 200 OK with JSON |
| GET /ping | Simple liveness | 200 OK "pong" |

### **Key Ports**

| **Service** | **Port** | **Protocol** |
| --- | --- | --- |
| Application | 5000 | HTTP |
| ALB HTTPS | 443 | HTTPS |
| ALB HTTP | 80 | HTTP (redirect to 443) |

### **Resource Sizing**

| **Resource** | **Minimum** | **Recommended** |
| --- | --- | --- |
| CPU | 0.5 vCPU | 1 vCPU |
| Memory | 1 GB | 2 GB |
| Instances | 1 | 2 (multi-AZ) |

## **Application Overview**

### **Architecture Summary**

Internet → ALB (443) → ECS Fargate (5000) → Neon PostgreSQL (External)

→ Zoho CRM API (External)

### **Technology Stack**

* **Runtime**: Node.js 20 (Alpine)
* **Framework**: Express.js + React (Vite)
* **Database**: PostgreSQL (Neon - external, no AWS RDS needed)
* **External APIs**: Zoho CRM (OAuth 2.0)

### **Background Processes**

The application runs these in-process workers (no separate containers needed):

| **Worker** | **Interval** | **Purpose** |
| --- | --- | --- |
| ZohoSyncWorker | 10 seconds | Syncs pending form submissions to Zoho CRM |
| TokenManager | 3.3 minutes | Refreshes OAuth tokens before expiry |
| FieldMetadataCache | 24 hours | Caches Zoho CRM field definitions |

**Important**: These workers run inside the main container. Each container instance runs its own workers. Design is idempotent - multiple instances won't cause conflicts.

### **Database Tables**

| **Table** | **Purpose** |
| --- | --- |
| users | Admin users |
| resources | Uploaded documents |
| form\_submissions | Lead capture data |
| submission\_logs | Sync audit trail |
| field\_mappings | CRM field mappings |
| form\_configurations | Dynamic form configs |
| field\_metadata\_cache | Zoho field cache |
| oauth\_tokens | OAuth credentials |
| automation\_workflows | Workflow definitions |
| workflow\_executions | Workflow runs |
| action\_executions | Action runs |
| campaign\_syncs | Marketing syncs |
| townhall\_registrations | Event signups |
| event\_admins | Event admin users |

## **AWS Resource Inventory**

### **Required AWS Services**

| **Service** | **Resource Name** | **Purpose** |
| --- | --- | --- |
| ECR | cas-website | Docker image registry |
| ECS Cluster | cas-production | Container orchestration |
| ECS Service | cas-website-service | Application service |
| ECS Task Definition | cas-website | Container spec |
| ALB | cas-alb | Load balancer |
| Target Group | cas-tg | Health checks |
| VPC | cas-vpc | Network isolation |
| Security Group | cas-alb-sg | ALB ingress |
| Security Group | cas-ecs-sg | ECS tasks |
| Route 53 | amyloid.ca | DNS zone |
| ACM | \*.amyloid.ca | SSL certificate |
| Secrets Manager | cas/\* | Credentials |
| CloudWatch Log Group | /ecs/cas-website | Logs |
| IAM Role | cas-ecs-task-role | Task permissions |
| IAM Role | cas-ecs-execution-role | Container startup |
| IAM User | cas-github-actions | CI/CD deployment |

## **Infrastructure Setup**

### **1. VPC Configuration**

# Terraform: vpc.tf

module "vpc" {

source = "terraform-aws-modules/vpc/aws"

version = "5.0.0"

name = "cas-vpc"

cidr = "10.0.0.0/16"

azs = ["us-east-1a", "us-east-1b"]

private\_subnets = ["10.0.1.0/24", "10.0.2.0/24"]

public\_subnets = ["10.0.101.0/24", "10.0.102.0/24"]

enable\_nat\_gateway = true

single\_nat\_gateway = true

enable\_dns\_hostnames = true

enable\_dns\_support = true

tags = {

Project = "CAS Website"

Environment = "production"

}

}

### **2. Security Groups**

# ALB Security Group

resource "aws\_security\_group" "alb\_sg" {

name = "cas-alb-sg"

description = "Allow HTTPS inbound"

vpc\_id = module.vpc.vpc\_id

ingress {

description = "HTTPS from internet"

from\_port = 443

to\_port = 443

protocol = "tcp"

cidr\_blocks = ["0.0.0.0/0"]

}

ingress {

description = "HTTP redirect"

from\_port = 80

to\_port = 80

protocol = "tcp"

cidr\_blocks = ["0.0.0.0/0"]

}

egress {

from\_port = 0

to\_port = 0

protocol = "-1"

cidr\_blocks = ["0.0.0.0/0"]

}

tags = {

Name = "cas-alb-sg"

}

}

# ECS Security Group

resource "aws\_security\_group" "ecs\_sg" {

name = "cas-ecs-sg"

description = "Allow traffic from ALB"

vpc\_id = module.vpc.vpc\_id

ingress {

description = "HTTP from ALB"

from\_port = 5000

to\_port = 5000

protocol = "tcp"

security\_groups = [aws\_security\_group.alb\_sg.id]

}

egress {

description = "All outbound (DB, Zoho API)"

from\_port = 0

to\_port = 0

protocol = "-1"

cidr\_blocks = ["0.0.0.0/0"]

}

tags = {

Name = "cas-ecs-sg"

}

}

### **3. ECR Repository**

# Create ECR repository

aws ecr create-repository \

--repository-name cas-website \

--image-scanning-configuration scanOnPush=true \

--encryption-configuration encryptionType=AES256 \

--region us-east-1

# Lifecycle policy (keep last 10 images)

aws ecr put-lifecycle-policy \

--repository-name cas-website \

--lifecycle-policy-text '{

"rules": [

{

"rulePriority": 1,

"description": "Keep last 10 images",

"selection": {

"tagStatus": "any",

"countType": "imageCountMoreThan",

"countNumber": 10

},

"action": {

"type": "expire"

}

}

]

}' \

--region us-east-1

### **4. Application Load Balancer**

# ALB

resource "aws\_lb" "cas\_alb" {

name = "cas-alb"

internal = false

load\_balancer\_type = "application"

security\_groups = [aws\_security\_group.alb\_sg.id]

subnets = module.vpc.public\_subnets

enable\_deletion\_protection = true

tags = {

Name = "cas-alb"

}

}

# Target Group

resource "aws\_lb\_target\_group" "cas\_tg" {

name = "cas-tg"

port = 5000

protocol = "HTTP"

vpc\_id = module.vpc.vpc\_id

target\_type = "ip"

health\_check {

enabled = true

healthy\_threshold = 2

unhealthy\_threshold = 3

timeout = 5

interval = 30

path = "/health"

port = "traffic-port"

protocol = "HTTP"

matcher = "200"

}

stickiness {

type = "lb\_cookie"

cookie\_duration = 86400

enabled = true

}

tags = {

Name = "cas-tg"

}

}

# HTTPS Listener

resource "aws\_lb\_listener" "https" {

load\_balancer\_arn = aws\_lb.cas\_alb.arn

port = "443"

protocol = "HTTPS"

ssl\_policy = "ELBSecurityPolicy-TLS13-1-2-2021-06"

certificate\_arn = aws\_acm\_certificate.cas\_cert.arn

default\_action {

type = "forward"

target\_group\_arn = aws\_lb\_target\_group.cas\_tg.arn

}

}

# HTTP to HTTPS Redirect

resource "aws\_lb\_listener" "http\_redirect" {

load\_balancer\_arn = aws\_lb.cas\_alb.arn

port = "80"

protocol = "HTTP"

default\_action {

type = "redirect"

redirect {

port = "443"

protocol = "HTTPS"

status\_code = "HTTP\_301"

}

}

}

### **5. ECS Cluster**

resource "aws\_ecs\_cluster" "cas\_production" {

name = "cas-production"

setting {

name = "containerInsights"

value = "enabled"

}

configuration {

execute\_command\_configuration {

logging = "OVERRIDE"

log\_configuration {

cloud\_watch\_log\_group\_name = "/ecs/cas-website/exec"

}

}

}

tags = {

Name = "cas-production"

Environment = "production"

}

}

resource "aws\_ecs\_cluster\_capacity\_providers" "cas\_fargate" {

cluster\_name = aws\_ecs\_cluster.cas\_production.name

capacity\_providers = ["FARGATE", "FARGATE\_SPOT"]

default\_capacity\_provider\_strategy {

base = 1

weight = 100

capacity\_provider = "FARGATE"

}

}

### **6. ECS Service**

resource "aws\_ecs\_service" "cas\_service" {

name = "cas-website-service"

cluster = aws\_ecs\_cluster.cas\_production.id

task\_definition = aws\_ecs\_task\_definition.cas\_task.arn

desired\_count = 2

launch\_type = "FARGATE"

network\_configuration {

subnets = module.vpc.private\_subnets

security\_groups = [aws\_security\_group.ecs\_sg.id]

assign\_public\_ip = false

}

load\_balancer {

target\_group\_arn = aws\_lb\_target\_group.cas\_tg.arn

container\_name = "cas-app"

container\_port = 5000

}

deployment\_configuration {

maximum\_percent = 200

minimum\_healthy\_percent = 100

}

deployment\_circuit\_breaker {

enable = true

rollback = true

}

lifecycle {

ignore\_changes = [task\_definition]

}

tags = {

Name = "cas-website-service"

}

}

## **IAM Configuration**

### **ECS Task Execution Role**

{

"Version": "2012-10-17",

"Statement": [

{

"Sid": "ECRPull",

"Effect": "Allow",

"Action": [

"ecr:GetAuthorizationToken",

"ecr:BatchCheckLayerAvailability",

"ecr:GetDownloadUrlForLayer",

"ecr:BatchGetImage"

],

"Resource": "\*"

},

{

"Sid": "CloudWatchLogs",

"Effect": "Allow",

"Action": [

"logs:CreateLogStream",

"logs:PutLogEvents"

],

"Resource": "arn:aws:logs:us-east-1:\*:log-group:/ecs/cas-website:\*"

},

{

"Sid": "SecretsManager",

"Effect": "Allow",

"Action": [

"secretsmanager:GetSecretValue"

],

"Resource": "arn:aws:secretsmanager:us-east-1:\*:secret:cas/\*"

}

]

}

### **ECS Task Role**

{

"Version": "2012-10-17",

"Statement": [

{

"Sid": "CloudWatchMetrics",

"Effect": "Allow",

"Action": [

"cloudwatch:PutMetricData"

],

"Resource": "\*",

"Condition": {

"StringEquals": {

"cloudwatch:namespace": "CAS/Website"

}

}

}

]

}

### **GitHub Actions IAM User Policy**

{

"Version": "2012-10-17",

"Statement": [

{

"Sid": "ECRLogin",

"Effect": "Allow",

"Action": "ecr:GetAuthorizationToken",

"Resource": "\*"

},

{

"Sid": "ECRPush",

"Effect": "Allow",

"Action": [

"ecr:BatchCheckLayerAvailability",

"ecr:GetDownloadUrlForLayer",

"ecr:BatchGetImage",

"ecr:InitiateLayerUpload",

"ecr:UploadLayerPart",

"ecr:CompleteLayerUpload",

"ecr:PutImage"

],

"Resource": "arn:aws:ecr:us-east-1:\*:repository/cas-website"

},

{

"Sid": "ECSDeployment",

"Effect": "Allow",

"Action": [

"ecs:UpdateService",

"ecs:DescribeServices",

"ecs:DescribeClusters",

"ecs:ListTasks",

"ecs:DescribeTasks"

],

"Resource": [

"arn:aws:ecs:us-east-1:\*:cluster/cas-production",

"arn:aws:ecs:us-east-1:\*:service/cas-production/cas-website-service"

]

},

{

"Sid": "ECSWaitForStable",

"Effect": "Allow",

"Action": [

"ecs:DescribeServices"

],

"Resource": "\*"

}

]

}

### **Create GitHub Actions User**

# Create IAM user

aws iam create-user --user-name cas-github-actions

# Attach policy

aws iam put-user-policy \

--user-name cas-github-actions \

--policy-name CASGitHubActionsPolicy \

--policy-document file://github-actions-policy.json

# Create access key

aws iam create-access-key --user-name cas-github-actions

## **Secrets Configuration**

### **Secrets to Create in AWS Secrets Manager**

| **Secret Path** | **Description** | **Source** |
| --- | --- | --- |
| cas/database-url | PostgreSQL connection string | Neon dashboard |
| cas/zoho-client-id | Zoho OAuth client ID | Zoho API Console |
| cas/zoho-client-secret | Zoho OAuth client secret | Zoho API Console |
| cas/session-secret | Express session secret | Generate random |
| cas/event-admin-username | Event admin login | cannAdmin |
| cas/event-admin-password | Event admin password | Townhall2025! |

### **Create Secrets**

# Database URL (from Neon)

aws secretsmanager create-secret \

--name cas/database-url \

--description "PostgreSQL connection string for CAS website" \

--secret-string "postgresql://user:pass@host/db?sslmode=require"

# Zoho OAuth credentials

aws secretsmanager create-secret \

--name cas/zoho-client-id \

--description "Zoho CRM OAuth client ID" \

--secret-string "1000.XXXX"

aws secretsmanager create-secret \

--name cas/zoho-client-secret \

--description "Zoho CRM OAuth client secret" \

--secret-string "secret-value"

# Session secret (generate new)

aws secretsmanager create-secret \

--name cas/session-secret \

--description "Express session secret" \

--secret-string "$(openssl rand -base64 32)"

# Event admin credentials

aws secretsmanager create-secret \

--name cas/event-admin-username \

--description "Event admin username" \

--secret-string "cannAdmin"

aws secretsmanager create-secret \

--name cas/event-admin-password \

--description "Event admin password" \

--secret-string "Townhall2025!"

## **Container Configuration**

### **Dockerfile**

# Stage 1: Build frontend

FROM node:20-alpine AS frontend-builder

WORKDIR /app

COPY package\*.json ./

RUN npm ci

COPY . .

RUN npm run build

# Stage 2: Build backend

FROM node:20-alpine AS backend-builder

WORKDIR /app

COPY package\*.json ./

RUN npm ci --omit=dev

COPY --from=frontend-builder /app/dist ./dist

COPY server ./server

COPY shared ./shared

RUN npx esbuild server/index.ts --bundle --platform=node --outfile=dist/server.js --packages=external

# Stage 3: Production runtime

FROM node:20-alpine AS production

RUN apk add --no-cache wget

WORKDIR /app

COPY --from=backend-builder /app/dist ./dist

COPY --from=backend-builder /app/node\_modules ./node\_modules

COPY package.json ./

ENV NODE\_ENV=production

ENV PORT=5000

EXPOSE 5000

HEALTHCHECK --interval=30s --timeout=5s --start-period=60s --retries=3 \

CMD wget -q --spider http://localhost:5000/health || exit 1

CMD ["node", "dist/server.js"]

### **ECS Task Definition**

{

"family": "cas-website",

"networkMode": "awsvpc",

"requiresCompatibilities": ["FARGATE"],

"cpu": "1024",

"memory": "2048",

"executionRoleArn": "arn:aws:iam::ACCOUNT\_ID:role/cas-ecs-execution-role",

"taskRoleArn": "arn:aws:iam::ACCOUNT\_ID:role/cas-ecs-task-role",

"containerDefinitions": [

{

"name": "cas-app",

"image": "ACCOUNT\_[ID.dkr.ecr.us-east-1.amazonaws.com/cas-website:latest](https://ID.dkr.ecr.us-east-1.amazonaws.com/cas-website:latest)",

"essential": true,

"portMappings": [

{

"containerPort": 5000,

"protocol": "tcp"

}

],

"environment": [

{"name": "NODE\_ENV", "value": "production"},

{"name": "PORT", "value": "5000"},

{"name": "REPLIT\_DEPLOYMENT", "value": "1"}

],

"secrets": [

{

"name": "DATABASE\_URL",

"valueFrom": "arn:aws:secretsmanager:us-east-1:ACCOUNT\_ID:secret:cas/database-url"

},

{

"name": "ZOHO\_CLIENT\_ID",

"valueFrom": "arn:aws:secretsmanager:us-east-1:ACCOUNT\_ID:secret:cas/zoho-client-id"

},

{

"name": "ZOHO\_CLIENT\_SECRET",

"valueFrom": "arn:aws:secretsmanager:us-east-1:ACCOUNT\_ID:secret:cas/zoho-client-secret"

},

{

"name": "SESSION\_SECRET",

"valueFrom": "arn:aws:secretsmanager:us-east-1:ACCOUNT\_ID:secret:cas/session-secret"

},

{

"name": "EVENT\_ADMIN\_USERNAME",

"valueFrom": "arn:aws:secretsmanager:us-east-1:ACCOUNT\_ID:secret:cas/event-admin-username"

},

{

"name": "EVENT\_ADMIN\_PASSWORD",

"valueFrom": "arn:aws:secretsmanager:us-east-1:ACCOUNT\_ID:secret:cas/event-admin-password"

}

],

"logConfiguration": {

"logDriver": "awslogs",

"options": {

"awslogs-group": "/ecs/cas-website",

"awslogs-region": "us-east-1",

"awslogs-stream-prefix": "ecs",

"awslogs-create-group": "true"

}

},

"healthCheck": {

"command": ["CMD-SHELL", "wget -q --spider http://localhost:5000/health || exit 1"],

"interval": 30,

"timeout": 5,

"retries": 3,

"startPeriod": 60

}

}

]

}

## **Health Checks & Monitoring**

### **Health Check Endpoints**

**GET /health** - Full health check

{

"status": "healthy",

"timestamp": "2025-12-24T12:00:00.000Z",

"port": 5000,

"environment": "production",

"replitDeployment": true,

"databaseHost": "ep-xxx.us-east-2.aws.neon.tech"

}

**GET /ping** - Simple liveness

pong

### **ALB Health Check Configuration**

| **Setting** | **Value** |
| --- | --- |
| Protocol | HTTP |
| Port | traffic-port (5000) |
| Path | /health |
| Healthy threshold | 2 |
| Unhealthy threshold | 3 |
| Timeout | 5 seconds |
| Interval | 30 seconds |
| Success codes | 200 |

### **CloudWatch Alarms**

# High CPU Alarm

aws cloudwatch put-metric-alarm \

--alarm-name "CAS-HighCPU" \

--metric-name CPUUtilization \

--namespace AWS/ECS \

--statistic Average \

--period 300 \

--threshold 80 \

--comparison-operator GreaterThanThreshold \

--evaluation-periods 2 \

--dimensions Name=ClusterName,Value=cas-production Name=ServiceName,Value=cas-website-service \

--alarm-actions arn:aws:sns:us-east-1:ACCOUNT\_ID:cas-alerts

## **GitHub Actions Setup**

### **Required Repository Secrets**

| **Secret Name** | **Value** |
| --- | --- |
| AWS\_ACCESS\_KEY\_ID | From IAM user cas-github-actions |
| AWS\_SECRET\_ACCESS\_KEY | From IAM user cas-github-actions |
| AWS\_REGION | us-east-1 |
| AWS\_ACCOUNT\_ID | Your 12-digit account ID |
| ECR\_REPOSITORY | cas-website |
| ECS\_CLUSTER | cas-production |
| ECS\_SERVICE | cas-website-service |

### **Workflow File (.github/workflows/deploy.yml)**

name: Deploy to AWS ECS

on:

push:

branches: [main]

workflow\_dispatch:

env:

AWS\_REGION: ${{ secrets.AWS\_REGION }}

ECR\_REPOSITORY: ${{ secrets.ECR\_REPOSITORY }}

ECS\_CLUSTER: ${{ secrets.ECS\_CLUSTER }}

ECS\_SERVICE: ${{ secrets.ECS\_SERVICE }}

jobs:

deploy:

name: Build and Deploy

runs-on: ubuntu-latest

timeout-minutes: 30

steps:

- name: Checkout code

uses: actions/checkout@v4

- name: Configure AWS credentials

uses: aws-actions/configure-aws-credentials@v4

with:

aws-access-key-id: ${{ secrets.AWS\_ACCESS\_KEY\_ID }}

aws-secret-access-key: ${{ secrets.AWS\_SECRET\_ACCESS\_KEY }}

aws-region: ${{ secrets.AWS\_REGION }}

- name: Login to Amazon ECR

id: login-ecr

uses: aws-actions/amazon-ecr-login@v2

- name: Build and push Docker image

id: build-image

env:

ECR\_REGISTRY: ${{ steps.login-ecr.outputs.registry }}

IMAGE\_TAG: ${{ github.sha }}

run: |

docker build -t $ECR\_REGISTRY/$ECR\_REPOSITORY:$IMAGE\_TAG .

docker tag $ECR\_REGISTRY/$ECR\_REPOSITORY:$IMAGE\_TAG $ECR\_REGISTRY/$ECR\_REPOSITORY:latest

docker push $ECR\_REGISTRY/$ECR\_REPOSITORY:$IMAGE\_TAG

docker push $ECR\_REGISTRY/$ECR\_REPOSITORY:latest

echo "image=$ECR\_REGISTRY/$ECR\_REPOSITORY:$IMAGE\_TAG" >> $GITHUB\_OUTPUT

- name: Deploy to ECS

run: |

aws ecs update-service \

--cluster $ECS\_CLUSTER \

--service $ECS\_SERVICE \

--force-new-deployment

- name: Wait for deployment

run: |

aws ecs wait services-stable \

--cluster $ECS\_CLUSTER \

--services $ECS\_SERVICE

## **Operational Runbook**

### **Scale Service**

# Scale up

aws ecs update-service \

--cluster cas-production \

--service cas-website-service \

--desired-count 4

# Scale down

aws ecs update-service \

--cluster cas-production \

--service cas-website-service \

--desired-count 2

### **View Logs**

# Recent logs

aws logs tail /ecs/cas-website --follow

## **Troubleshooting Guide**

### **Container Won't Start**

**Symptoms**: Tasks fail to start, service stuck at 0 running tasks

**Check**:

1. View task stopped reason:

aws ecs describe-tasks \

--cluster cas-production \

--tasks $(aws ecs list-tasks --cluster cas-production --service-name cas-website-service --query 'taskArns[0]' --output text)

**Common Causes**:

* Missing secrets → Verify secrets exist in Secrets Manager
* Invalid DATABASE\_URL → Check Neon connection string