ThreatCompass Production Deployment Guide

This guide walks you through deploying ThreatCompass to AWS using the provided Terraform templates.

Prerequisites

1. **AWS Account** with appropriate permissions
2. **Domain Name** (optional but recommended)
3. **Terraform** installed (v1.0+)
4. **AWS CLI** configured
5. **Docker** for building container images

Phase 1: Pre-Deployment Setup

1. Configure AWS CLI

```bash
# Configure AWS CLI with your credentials
aws configure
# Enter your Access Key ID, Secret Access Key, and preferred region
```

2. Create Terraform State Management

First, create the S3 bucket and DynamoDB table for Terraform state:

```bash
# Create S3 bucket for Terraform state
aws s3 mb s3://threatcompass-terraform-state --region us-east-1

# Enable versioning
aws s3api put-bucket-versioning \
  --bucket threatcompass-terraform-state \
  --versioning-configuration Status=Enabled

# Create DynamoDB table for state locking
aws dynamodb create-table \
  --table-name threatcompass-terraform-locks \
  --attribute-definitions AttributeName=LockID,AttributeType=S \
  --key-schema AttributeName=LockID,KeyType=HASH \
  --billing-mode PAY_PER_REQUEST \
  --region us-east-1
```

3. Prepare Your Variables

Create a `terraform.tfvars` file:

hcl

```
# terraform.tfvars
aws_region = "us-east-1"
environment = "production"
project_name = "threatcompass"

# Domain configuration (optional)
domain_name = "threatcompass.yourdomain.com"

# Database configuration
db_instance_class = "db.t3.medium"
db_allocated_storage = 100

# Redis configuration
redis_node_type = "cache.t3.medium"

# ECS configuration
flask_cpu = 512
flask_memory = 1024
flask_min_capacity = 2
flask_max_capacity = 10

celery_worker_cpu = 256
celery_worker_memory = 512
celery_worker_min_capacity = 1
celery_worker_max_capacity = 5

# API Keys (store these securely)
virustotal_api_key = "your-virustotal-api-key"
```

```
abuseipdb_api_key = "your-abuseipdb-api-key"
smtp_password = "your-smtp-password"
```

Phase 2: Infrastructure Deployment

1. Initialize Terraform

bash

```
# Clone your ThreatCompass repository
git clone https://github.com/your-org/threatcompass.git
cd threatcompass/terraform

# Initialize Terraform
terraform init

# Validate configuration
terraform validate

# Plan deployment
terraform plan -var-file="terraform.tfvars"
```

2. Deploy Infrastructure

```bash
# Deploy infrastructure
terraform apply -var-file="terraform.tfvars"

# This will take 15-20 minutes to complete
# Review the plan carefully before confirming
```

3. Verify Infrastructure

```bash
# Get deployment outputs
terraform output

# Test database connectivity
aws rds describe-db-instances \
  --db-instance-identifier threatcompass-production-postgresql \
  --region us-east-1

# Test Redis connectivity
aws elasticache describe-replication-groups \
  --replication-group-id threatcompass-production-redis \
  --region us-east-1
```

Phase 3: Container Image Preparation

1. Build Application Images

Create optimized Dockerfiles for production:

dockerfile

```dockerfile
# Dockerfile.flask-app
FROM python:3.11-slim

# Set environment variables
ENV PYTHONUNBUFFERED=1
ENV PYTHONDONTWRITEBYTECODE=1

# Install system dependencies
RUN apt-get update && apt-get install -y \
    gcc \
    libpq-dev \
    curl \
    && rm -rf /var/lib/apt/lists/*

# Create app user
RUN useradd --create-home --shell /bin/bash app

# Set work directory
WORKDIR /app

# Copy requirements and install Python dependencies
COPY requirements.txt .
RUN pip install --no-cache-dir -r requirements.txt

# Copy application code
COPY . .

# Change ownership
```

```
RUN chown -R app:app /app
USER app

# Health check
HEALTHCHECK --interval=30s --timeout=10s --start-period=30s --retries=3 \
    CMD curl -f http://localhost:5000/health || exit 1

# Expose port
EXPOSE 5000

# Run application
CMD ["gunicorn", "--bind", "0.0.0.0:5000", "--workers", "4", "--timeout", "120", "ap
```

dockerfile

```dockerfile
# Dockerfile.celery-worker
FROM python:3.11-slim

ENV PYTHONUNBUFFERED=1
ENV PYTHONDONTWRITEBYTECODE=1

RUN apt-get update && apt-get install -y \
    gcc \
    libpq-dev \
    && rm -rf /var/lib/apt/lists/*

RUN useradd --create-home --shell /bin/bash celery

WORKDIR /app

COPY requirements.txt .
RUN pip install --no-cache-dir -r requirements.txt

COPY . .
RUN chown -R celery:celery /app
USER celery

CMD ["celery", "-A", "app.celery", "worker", "--loglevel=info", "--concurrency=4"]
```

dockerfile

```dockerfile
# Dockerfile.celery-beat
FROM python:3.11-slim

ENV PYTHONUNBUFFERED=1
ENV PYTHONDONTWRITEBYTECODE=1

RUN apt-get update && apt-get install -y \
    gcc \
    libpq-dev \
    && rm -rf /var/lib/apt/lists/*

RUN useradd --create-home --shell /bin/bash celery

WORKDIR /app

COPY requirements.txt .
RUN pip install --no-cache-dir -r requirements.txt

COPY . .
RUN chown -R celery:celery /app
USER celery

CMD ["celery", "-A", "app.celery", "beat", "--loglevel=info"]
```

2. Build and Push Images

bash

```
# Get ECR login token
aws ecr get-login-password --region us-east-1 | docker login --username AWS --p

# Get ECR repository URLs from Terraform output
FLASK_REPO=$(terraform output -raw ecr_flask_app_url)
CELERY_WORKER_REPO=$(terraform output -raw ecr_celery_worker_url)
CELERY_BEAT_REPO=$(terraform output -raw ecr_celery_beat_url)

# Build and push Flask app
docker build -f Dockerfile.flask-app -t threatcompass-flask-app .
docker tag threatcompass-flask-app:latest $FLASK_REPO:latest
docker tag threatcompass-flask-app:latest $FLASK_REPO:prod-$(date +%Y%m%
docker push $FLASK_REPO:latest
docker push $FLASK_REPO:prod-$(date +%Y%m%d-%H%M%S)

# Build and push Celery worker
docker build -f Dockerfile.celery-worker -t threatcompass-celery-worker .
docker tag threatcompass-celery-worker:latest $CELERY_WORKER_REPO:latest
docker tag threatcompass-celery-worker:latest $CELERY_WORKER_REPO:prod-$
docker push $CELERY_WORKER_REPO:latest
docker push $CELERY_WORKER_REPO:prod-$(date +%Y%m%d-%H%M%S)

# Build and push Celery beat
docker build -f Dockerfile.celery-beat -t threatcompass-celery-beat .
docker tag threatcompass-celery-beat:latest $CELERY_BEAT_REPO:latest
docker tag threatcompass-celery-beat:latest $CELERY_BEAT_REPO:prod-$(date -
docker push $CELERY_BEAT_REPO:latest
docker push $CELERY_BEAT_REPO:prod-$(date +%Y%m%d-%H%M%S)
```

Phase 4: Database Initialization

1. Initialize Database Schema

bash

```
# Get database connection details
DB_SECRET_ARN=$(terraform output -raw db_secret_arn)
DB_ENDPOINT=$(aws secretsmanager get-secret-value --secret-id $DB_SECRE

# Create a temporary ECS task to run database migrations
aws ecs run-task \
  --cluster threatcompass-production-cluster \
  --task-definition threatcompass-production-flask-app \
  --launch-type FARGATE \
  --network-configuration "awsvpcConfiguration={subnets=[$(terraform output -r
  --overrides '{
    "containerOverrides": [
      {
        "name": "flask-app",
        "command": ["python", "-c", "from app import db; db.create_all(); print(\"D
      }
    ]
  }'
```

2. Create Initial Admin User

bash

```bash
# Run task to create admin user
aws ecs run-task \
  --cluster threatcompass-production-cluster \
  --task-definition threatcompass-production-flask-app \
  --launch-type FARGATE \
  --network-configuration "awsvpcConfiguration={subnets=[$(terraform output -r
  --overrides '{
    "containerOverrides": [
      {
        "name": "flask-app",
        "command": ["python", "scripts/create_admin_user.py", "--email", "admin(
      }
    ]
  }'
```

Phase 5: DNS and SSL Configuration

1. Configure DNS (if using custom domain)

bash

```bash
# Get ALB DNS name
ALB_DNS=$(terraform output -raw alb_dns_name)
ALB_ZONE_ID=$(terraform output -raw alb_zone_id)

# Create/update DNS records (if not using Terraform Route53 configuration)
aws route53 change-resource-record-sets \
  --hosted-zone-id Z123456789 \
  --change-batch '{
    "Changes": [
      {
        "Action": "UPSERT",
        "ResourceRecordSet": {
          "Name": "threatcompass.yourdomain.com",
          "Type": "A",
          "AliasTarget": {
            "DNSName": "'$ALB_DNS'",
            "EvaluateTargetHealth": true,
            "HostedZoneId": "'$ALB_ZONE_ID'"
          }
        }
      }
    ]
  }'
```

2. Verify SSL Certificate

bash

```bash
# Check certificate status
aws acm describe-certificate \
  --certificate-arn $(terraform output -raw certificate_arn) \
  --region us-east-1

# Test HTTPS connectivity
curl -I https://threatcompass.yourdomain.com/health
```

Phase 6: Verification and Testing

1. Health Checks

bash

```bash
# Test application health
HEALTH_URL=$(terraform output -raw application_url)/health
curl -f $HEALTH_URL

# Check ECS service status
aws ecs describe-services \
  --cluster threatcompass-production-cluster \
  --services threatcompass-production-flask-app \
  --region us-east-1

# Check target group health
aws elbv2 describe-target-health \
  --target-group-arn $(aws elbv2 describe-target-groups --names threatcompass
  --region us-east-1
```

2. Functional Testing

bash

```bash
# Test API endpoints
API_URL=$(terraform output -raw application_url)/api/v1

# Test authentication (replace with actual API key)
curl -H "X-API-Key: your-api-key" $API_URL/iocs

# Test database connectivity through application
curl -X POST $API_URL/iocs \
  -H "Content-Type: application/json" \
  -H "X-API-Key: your-api-key" \
  -d '{
    "value": "8.8.8.8",
    "type": "IP_ADDRESS",
    "source": "manual",
    "description": "Test IOC"
  }'
```

3. Monitoring Setup

```bash
# Check CloudWatch logs
aws logs describe-log-groups \
  --log-group-name-prefix "/ecs/threatcompass-production" \
  --region us-east-1

# View recent application logs
aws logs tail /ecs/threatcompass-production/flask-app --follow --region us-east-1
```

Phase 7: Post-Deployment Security

1. Enable GuardDuty

```bash
# Enable GuardDuty for threat detection
aws guardduty create-detector --enable --region us-east-1
```

2. Configure CloudTrail

bash

```bash
# Enable CloudTrail for audit logging
aws cloudtrail create-trail \
  --name threatcompass-audit-trail \
  --s3-bucket-name threatcompass-cloudtrail-logs \
  --include-global-service-events \
  --is-multi-region-trail \
  --enable-log-file-validation
```

3. Set Up Backup Verification

bash

```bash
# Test RDS snapshot restoration
aws rds describe-db-snapshots \
  --db-instance-identifier threatcompass-production-postgresql \
  --snapshot-type automated \
  --region us-east-1
```

Ongoing Maintenance

Daily Tasks

- Monitor CloudWatch dashboards
- Review application logs
- Check backup completion

Weekly Tasks

- Review security alerts
- Update container images
- Test disaster recovery procedures

Monthly Tasks

- Rotate secrets
- Review and update IAM policies
- Capacity planning review

Troubleshooting Common Issues

ECS Task Startup Issues

```bash
# Check task definition
aws ecs describe-task-definition \
  --task-definition threatcompass-production-flask-app

# Check service events
aws ecs describe-services \
  --cluster threatcompass-production-cluster \
  --services threatcompass-production-flask-app \
  --query 'services[0].events'
```

Database Connection Issues

bash

```bash
# Test database connectivity from ECS task
aws ecs execute-command \
  --cluster threatcompass-production-cluster \
  --task <task-arn> \
  --container flask-app \
  --interactive \
  --command "/bin/bash"
```

SSL Certificate Issues

bash

```bash
# Check certificate validation
aws acm describe-certificate \
  --certificate-arn <certificate-arn> \
  --
```