


- [Complete Deployment Guide](#)
 -  [CRITICAL INFORMATION - READ FIRST](#)
 - [Table of Contents](#)
 - [Infrastructure Overview](#)
 - [Prerequisites](#)
 - [Local Development with Docker](#)
 - [AWS Deployment with Terraform](#)
 - [Destroying Infrastructure](#)
 - [Troubleshooting](#)
 - [Quick Reference Commands](#)
 - [CI/CD Pipeline \(GitHub Actions\)](#)
 - [Important Notes](#)
 - [Support Resources](#)

Complete Deployment Guide

Secure Messaging Application - Terraform Infrastructure as Code


Developed by DevOps Engineer Naeem Dosh | Fiverr



CRITICAL INFORMATION - READ FIRST

Application is Already Deployed

The secure messaging application is **currently live and running**:

- **Production URL:** <https://taxplanner.app>
- **AWS Region:** us-east-2 (Ohio)
- **Status:** Production Ready 
- **Infrastructure:** 100% Terraform-managed
- **Backend State:** S3 bucket hipaa-poc-tfstate-730543776652

Before Making Any Changes



YOU MUST DESTROY EXISTING INFRASTRUCTURE BEFORE REDEPLOYING

To redeploy or make infrastructure changes, follow this order:

1. **FIRST:** Backup database and data
2. **SECOND:** Destroy existing infrastructure (see [Destroying Infrastructure](#))
3. **THIRD:** Make configuration changes if needed
4. **FOURTH:** Deploy fresh infrastructure using Terraform

DO NOT attempt to deploy without destroying first - this will cause state conflicts and deployment failures.

Infrastructure-as-Code Philosophy

ALL INFRASTRUCTURE AND APPLICATION DEPLOYMENT IS DONE VIA TERRAFORM

✓ **What Terraform manages:** - VPC, subnets, routing, NAT gateway - Application Load Balancer (ALB) with SSL/TLS - EC2 instance (Amazon Linux 2023, t3.small) - Security groups and network ACLs - IAM roles and policies - S3 buckets for backups and state - AWS Secrets Manager for credentials - CloudWatch logs and monitoring - ACM certificates for HTTPS - **Automated Docker deployment** via user_data script - **Application code deployment** (clones from GitHub) - **Environment configuration** (.env from Secrets Manager)

✗ **Do NOT do manually:** - EC2 instance configuration - Security group changes - Application deployment - Docker setup - .env file creation (automated via user_data)

Table of Contents

1. [Infrastructure Overview](#)
 2. [Prerequisites](#)
 3. [Local Development with Docker](#)
 4. [AWS Deployment with Terraform](#)
 5. [Destroying Infrastructure](#)
 6. [Troubleshooting](#)
 7. [Quick Reference](#)
-

Infrastructure Overview

What Gets Deployed

When you run `terraform apply`, the following infrastructure is created:

Network Layer: - VPC (10.0.0.0/16) - 2 Public subnets (across 2 AZs) - 2 Private subnets (across 2 AZs) - Internet Gateway - NAT Gateway (for private subnet internet access) - Route tables and associations

Compute Layer: - EC2 instance (t3.small, Amazon Linux 2023) - Private subnet deployment - 30GB encrypted root volume - 10GB encrypted data volume for database - IMDSv2 required for enhanced security

Application Layer: - Docker & Docker Compose (auto-installed) - Application code (cloned from GitHub) - PHP-FPM + Nginx containers - SQLite database on encrypted volume

Load Balancing & SSL: - Application Load Balancer (ALB) - ACM certificate for taxplanner.app - HTTP to HTTPS redirect - Health checks

Security & Secrets: - AWS Secrets Manager (stores credentials) - Security groups (ALB, EC2) - IAM roles and instance profiles - SSM Session Manager access

Storage & Backups: - S3 bucket for database backups - Automated daily backups (2 AM UTC) - Server-side encryption (AES-256)

Monitoring: - CloudWatch Log Groups - Application logs (90-day retention) - Audit logs (365-day retention) - CloudWatch agent on EC2

State Management: - S3 backend (hipaa-poc-tfstate-730543776652) - DynamoDB table for state locking - Encryption at rest

Prerequisites

Required Tools

1. Terraform (≥ 1.0)

```
# Install Terraform
wget https://releases.hashicorp.com/terraform/1.6.0/
      terraform_1.6.0_linux_amd64.zip
unzip terraform_1.6.0_linux_amd64.zip
sudo mv terraform /usr/local/bin/
terraform --version
```

2. AWS CLI (≥ 2.0)

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

3. Git

```
sudo apt-get install git # Ubuntu/Debian
sudo yum install git     # RHEL/Amazon Linux
```

AWS Account Requirements

- **Account:** Active AWS account
- **Region:** us-east-2 (Ohio)
- **Permissions:** Administrator access or these IAM permissions:
 - EC2 (VPC, Instances, Security Groups, ALB)
 - S3 (Buckets, Objects)
 - IAM (Roles, Policies, Instance Profiles)
 - Secrets Manager
 - CloudWatch Logs

- ACM (Certificate Manager)
- DynamoDB (for state locking)

Google OAuth Credentials

1. Go to [Google Cloud Console](#)
 2. Create/select project
 3. Enable Google+ API
 4. Create OAuth 2.0 credentials
 5. Configure:
 - **Authorized redirect URIs:** `https://taxplanner.app/login.php`
 - Copy Client ID and Client Secret
-

Local Development with Docker

For local testing before deploying to AWS:

Step 1: Clone Repository

```
git clone https://github.com/appcropolisdevops/awspoc.git
cd awspoc
```

Step 2: Create .env File

```
cat > .env << 'EOF'
# Google OAuth
GOOGLE_CLIENT_ID=your-google-client-id.apps.googleusercontent.com
GOOGLE_CLIENT_SECRET=your-google-client-secret
GOOGLE_REDIRECT_URI=http://localhost:8080/login.php

# App Secret
APP_SECRET=your-random-secret-key-here

# Database
DB_PATH=/var/www/data/app.sqlite

# Database Encryption (optional)
DB_ENCRYPTION_KEY=your-encryption-key-here
EOF
```

Step 3: Run with Docker Compose

```
# Start containers
docker-compose up -d

# View logs
docker-compose logs -f
```

```
# Stop containers
docker-compose down
```

Step 4: Access Application

Open browser: <http://localhost:8080>

AWS Deployment with Terraform

Step 1: Configure AWS Credentials

```
aws configure
```

Enter: - **AWS Access Key ID:** Your access key - **AWS Secret Access Key:** Your secret key - **Default region:** us-east-2 - **Output format:** json

Verify:

```
aws sts get-caller-identity
```

Step 2: Store Secrets in AWS Secrets Manager

IMPORTANT: Secrets must be created BEFORE running Terraform!

```
cd terraform
```

```
# Create secrets in Secrets Manager
aws secretsmanager create-secret \
  --name hipaa-poc/app-secrets \
  --region us-east-2 \
  --secret-string '{
    "GOOGLE_CLIENT_ID": "your-google-client-
      id.apps.googleusercontent.com",
    "GOOGLE_CLIENT_SECRET": "your-google-client-secret",
    "APP_SECRET": "your-random-secret-key-min-32-chars",
    "DB_ENCRYPTION_KEY": "your-db-encryption-key-32-chars"
  }'
```

Generate secure secrets:

```
# Generate APP_SECRET
openssl rand -base64 32
```

```
# Generate DB_ENCRYPTION_KEY
openssl rand -base64 32
```

Step 3: Initialize Terraform Backend

FIRST TIME ONLY - Create S3 backend:

```
cd terraform/backend-setup
```

```
# Initialize backend setup
```

```
terraform init
```

```
# Create S3 bucket and DynamoDB table
```

```
terraform apply
```

This creates: - S3 bucket: hipaa-poc-tfstate-730543776652 - DynamoDB table: hipaa-poc-tfstate-locks

Step 4: Configure Terraform Variables

Create terraform/terraform.tfvars:

```
# AWS Configuration
```

```
aws_region = "us-east-2"
```

```
# Project Configuration
```

```
project_name = "hipaa-poc"
```

```
domain_name  = "taxplanner.app"
```

```
# EC2 Configuration
```

```
instance_type = "t3.small"
```

```
# Google OAuth Credentials (from Secrets Manager)
```

```
google_client_id      = "your-google-client-id.apps.googleusercontent.com"
```

```
google_client_secret = "your-google-client-secret"
```

```
# Application Secret
```

```
app_secret = "your-app-secret-from-above"
```

```
# Network Configuration
```

```
vpc_cidr = "10.0.0.0/16"
```

```
# SSH Access (optional - SSM Session Manager is preferred)
```

```
admin_ip = "" # Leave empty to use SSM only
```

Security Note: - Never commit terraform.tfvars to Git (already in .gitignore) - Use same values as stored in Secrets Manager

Step 5: Initialize Terraform

```
cd terraform
```

```
# Download providers and initialize
```

```
terraform init
```

```
# Validate configuration
```

```
terraform validate
```

Step 6: Plan Deployment

Create execution plan

```
terraform plan -out=tfplan
```

Review the plan carefully

Review what will be created: - 30+ resources - Estimated costs: ~\$50-80/month - t3.small EC2: ~\$15/month - ALB: ~\$20/month - NAT Gateway: ~\$35/month - Data transfer: Variable - Other resources: <\$5/month

Step 7: Deploy Complete Application with Terraform

Apply the plan

```
terraform apply tfplan
```

OR apply directly with confirmation










```
terraform apply
```

Enter yes when prompted.

 **IMPORTANT: This deploys the COMPLETE APPLICATION, not just infrastructure!**








Deployment time: 8-12 minutes



What terraform apply deploys:

Phase 1: Infrastructure (3-4 minutes) 1.  VPC with public/private subnets 2.  Internet Gateway and NAT Gateway 3.  Route tables and security groups 4.  Application Load Balancer (ALB) 5.  ACM certificate for HTTPS 6.  S3 buckets for backups 7.  IAM roles and policies 8.  CloudWatch log groups 9.  EC2 instance (t3.small, Amazon Linux 2023)







Phase 2: Automated Application Deployment (5-8 minutes)







The EC2 instance automatically runs `user_data.sh` which:

1.  Installs Docker & Docker Compose (2-3 min)
2.  Installs AWS CLI and required tools (1 min)
3.  **Clones your application from GitHub** (30 sec) `bash git clone https://github.com/appcropolisdevops/awspoc.git /app`
4.  **Retrieves secrets from AWS Secrets Manager** (10 sec)
5.  **Creates .env file automatically** (5 sec)
 - Google OAuth credentials
 - App secrets
 - Database configuration
6.  **Installs Composer dependencies** (30 sec) `bash composer install --no-dev --optimize-autoloader`
7.  **BUILDS AND STARTS THE APPLICATION** (2-3 min) `bash docker-compose -f docker-compose.prod.yml up -d --build`
 - Builds PHP-FPM container

- Builds Nginx container
- Starts application on port 80
- 8.  Sets up automated daily database backups to S3 (30 sec)
- 9.  Configures CloudWatch monitoring and logging (1 min)

 **RESULT: Your application is FULLY DEPLOYED and RUNNING!**

After terraform apply completes, your application is: -  Running in Docker containers -  Accessible via ALB (after DNS configuration) -  Connected to Secrets Manager for credentials -  Logging to CloudWatch -  Backing up to S3 daily -  Ready for production traffic

You do NOT need to: -  SSH into the server -  Manually install Docker -  Clone the repository -  Create .env files -  Run docker-compose manually -  Configure backups

Everything is 100% automated via Terraform!

Step 8: Review Outputs

View all outputs
terraform output

Important outputs:

terraform output alb_dns_name	# ALB DNS name
terraform output acm_validation_records	# For DNS validation
terraform output ssm_connect_command	# To connect via SSM
terraform output next_steps	# Post-deployment

Step 9: Configure DNS

Add CNAME record in your DNS provider:

Type: CNAME
Name: taxplanner.app
Value: [alb_dns_name from terraform output]
TTL: 300

Step 10: Validate ACM Certificate

Add DNS validation record (from acm_validation_records output):

Type: CNAME
Name: [from output]
Value: [from output]
TTL: 300

Wait for validation (usually 5-10 minutes).

Step 11: Verify Deployment

1. Check DNS propagation:

```
dig taxplanner.app
nslookup taxplanner.app
```

2. Access application:

```
https://taxplanner.app
```

3. Connect to EC2 via SSM:

```
aws ssm start-session --target [instance-id] --region us-east-2
```

4. Check Docker containers:

```
# Inside EC2 via SSM
sudo docker ps
sudo docker logs awspoc-php-1
sudo docker logs awspoc-nginx-1
```

5. Test Google OAuth:

- Visit <https://taxplanner.app>
- Click "Sign in with Google"
- Verify login works
- Send test message

Destroying Infrastructure

WARNING: DATA LOSS

Destroying infrastructure will **permanently delete**: - All EC2 instances and data - Database files (SQLite) - Application files - Log files - Network configuration

ALWAYS backup before destroying!

Step 1: Backup Data

```
# Connect to EC2 via SSM
aws ssm start-session --target [instance-id] --region us-east-2

# Inside EC2, backup database
sudo su -
cd /data/db
sqlite3 app.sqlite ".backup /tmp/backup-$(date +%Y%m%d).sqlite"
```

Upload to S3

```
aws s3 cp /tmp/backup-$(date +%Y%m%d).sqlite \
    s3://hipaa-poc-backups-[account-id]/manual-backups/ \
    --sse AES256
```

Exit SSM session

```
exit
exit
```

Step 2: Download Backups (Optional)

List backups

```
aws s3 ls s3://hipaa-poc-backups-[account-id]/backups/ --region
    us-east-2
```

Download latest backup

```
aws s3 cp s3://hipaa-poc-backups-[account-id]/backups/[latest-
    file] \
    ./backup.sqlite.gz --region us-east-2
```

Extract

```
gunzip backup.sqlite.gz
```

Step 3: Destroy Infrastructure

```
cd terraform
```

Preview what will be destroyed

```
terraform plan -destroy
```

Destroy all resources

```
terraform destroy
```

Review the plan, then type yes to confirm.

Destruction time: 5-8 minutes

What gets destroyed: 1. EC2 instance and volumes (data volume NOT deleted by default) 2. Load Balancer 3. VPC and networking 4. Security groups 5. IAM roles 6. S3 buckets (if empty) 7. CloudWatch log groups 8. ACM certificate

What persists: - S3 backend state bucket (manual deletion required) - DynamoDB state lock table - Secrets Manager secrets (manual deletion required) - S3 backup buckets (if configured to retain) - EBS data volume (delete_on_termination = false)

Step 4: Clean Up Backend (Optional)

To completely remove everything:

Delete Secrets Manager secret

```
aws secretsmanager delete-secret \  
  --secret-id hipaa-poc/app-secrets \  
  --region us-east-2 \  
  --force-delete-without-recovery
```

Empty and delete backup bucket

```
aws s3 rm s3://hipaa-poc-backups-[account-id] --recursive --  
  region us-east-2
```

```
aws s3 rb s3://hipaa-poc-backups-[account-id] --region us-east-2
```

Empty and delete state bucket

```
cd backend-setup
```

```
terraform destroy # Destroys S3 and DynamoDB
```

Or manually:

```
aws s3 rm s3://hipaa-poc-tfstate-730543776652 --recursive --  
  region us-east-2
```

```
aws s3 rb s3://hipaa-poc-tfstate-730543776652 --region us-east-2
```

```
aws dynamodb delete-table --table-name hipaa-poc-tfstate-locks --  
  region us-east-2
```

Selective Resource Destruction

To destroy specific resources only:

Destroy only EC2 instance

```
terraform destroy -target=aws_instance.app
```

Destroy ALB only

```
terraform destroy -target=aws_lb.main
```

Remove from state without destroying

```
terraform state rm aws_instance.app
```

Troubleshooting

Terraform Issues

Issue: “Backend initialization required”

Solution: Initialize Terraform

```
cd terraform
```

```
terraform init
```

Issue: “State lock error”

Solution: Check DynamoDB for locks

```
aws dynamodb scan --table-name hipaa-poc-tfstate-locks --region  
us-east-2
```

Force unlock (use with caution)

```
terraform force-unlock [LOCK_ID]
```

Issue: “Resource already exists”

Solution: Import existing resource

```
terraform import aws_instance.app i-1234567890abcdef0
```

Or remove from state

```
terraform state rm aws_instance.app
```

Issue: “Invalid credentials”

Solution: Reconfigure AWS CLI

```
aws configure  
aws sts get-caller-identity
```

Issue: “State file not found”

Solution: Verify S3 backend exists

```
aws s3 ls s3://hipaa-poc-tfstate-730543776652 --region us-east-2
```

Re-initialize if needed

```
terraform init -reconfigure
```

Deployment Issues

Issue: “ACM certificate validation pending”

Symptom: Certificate stuck in “Pending validation” status

Solution: 1. Check DNS records are correct 2. Verify CNAME record is propagated: `bash dig [validation-record-name]` 3. Wait up to 30 minutes for DNS propagation 4. Check ACM console for validation status

Issue: “ALB health checks failing”

Symptom: Targets showing unhealthy in ALB

Solution:

Connect to EC2

```
aws ssm start-session --target [instance-id] --region us-east-2
```

Check Docker containers

```
sudo docker ps
sudo docker logs awspoc-nginx-1
sudo docker logs awspoc-php-1
```

Check nginx is listening

```
sudo netstat -tlnp | grep 80
```

Restart containers if needed

```
cd /app
sudo docker-compose -f docker-compose.prod.yml restart
```

Issue: “Application not accessible”

Symptom: 502 Bad Gateway or timeout

Checklist: 1. Verify DNS is pointing to ALB 2. Check ACM certificate is validated 3. Verify security group allows traffic 4. Check Docker containers are running 5. Review CloudWatch logs: `bash aws logs tail /hipaa-poc/application --follow --region us-east-2`

Application Issues

Issue: “Google OAuth not working”

Solution: 1. Verify redirect URI in Google Console matches exactly: `https://taxplanner.app/login.php` 2. Check secrets in Secrets Manager are correct 3. Verify .env file on EC2: ````bash # Connect via SSM aws ssm start-session -target [instance-id] -region us-east-2`

`# Check .env sudo cat /app/.env ````

Issue: “Database errors”

Solution:

Connect to EC2

```
aws ssm start-session --target [instance-id] --region us-east-2
```

Check data volume is mounted

```
df -h | grep /data
```

Check database file exists

```
ls -lah /data/db/app.sqlite
```

Check permissions

```
sudo chown -R 1000:1000 /data/db
```

Issue: “Out of disk space”

Solution:

Check disk usage

```
df -h
```

Clean Docker images

```
sudo docker system prune -a
```

Check data volume

```
du -sh /data/*
```

Connection Issues

Issue: “Cannot connect via SSM”

Solution: 1. Verify SSM agent is running: `bash # On EC2 sudo systemctl status amazon-ssm-agent` 2. Check IAM instance profile has SSM permissions 3. Verify instance is in private subnet with NAT gateway 4. Check security group allows outbound HTTPS

Issue: “SSH not working”

Note: SSH is not configured by default. Use SSM Session Manager instead.

If SSH needed: 1. Add key pair to terraform variables 2. Update security group to allow port 22 3. Use SSM for secure access (recommended)

Quick Reference Commands

Terraform Commands

Initialize

```
terraform init
```

Validate configuration

```
terraform validate
```

Format code

```
terraform fmt -recursive
```

Plan changes

```
terraform plan
```

```
terraform plan -out=tfplan
```

Apply changes

```
terraform apply
```

```
terraform apply tfplan
terraform apply -auto-approve

# Destroy infrastructure
terraform destroy
terraform destroy -auto-approve

# View outputs
terraform output
terraform output alb_dns_name

# Show current state
terraform show

# List resources
terraform state list

# Show specific resource
terraform state show aws_instance.app

# Refresh state
terraform refresh

# Import existing resource
terraform import aws_instance.app i-1234567890

# Unlock state
terraform force-unlock [LOCK_ID]
```

AWS CLI Commands

```
# EC2 Commands
aws ec2 describe-instances --region us-east-2
aws ec2 describe-volumes --region us-east-2
aws ssm start-session --target [instance-id] --region us-east-2

# Secrets Manager
aws secretsmanager get-secret-value \
  --secret-id hipaa-poc/app-secrets \
  --region us-east-2

# S3 Commands
aws s3 ls s3://hipaa-poc-backups-[account-id]/ --region us-east-2
aws s3 cp local-file.sql s3://bucket/path/ --region us-east-2

# CloudWatch Logs
aws logs tail /hipaa-poc/application --follow --region us-east-2
aws logs filter-log-events \
  --log-group-name /hipaa-poc/application \
```

```
--region us-east-2
```

ACM

```
aws acm list-certificates --region us-east-2
```

```
aws acm describe-certificate --certificate-arn [arn] --region us-east-2
```

ALB

```
aws elbv2 describe-load-balancers --region us-east-2
```

```
aws elbv2 describe-target-health \  
    --target-group-arn [arn] \  
    --region us-east-2
```

Docker Commands (On EC2 via SSM)

View containers

```
sudo docker ps
```

```
sudo docker ps -a
```

View logs

```
sudo docker logs awspoc-php-1
```

```
sudo docker logs awspoc-nginx-1
```

```
sudo docker logs -f awspoc-php-1 # Follow
```

Restart containers

```
cd /app
```

```
sudo docker-compose -f docker-compose.prod.yml restart
```

Rebuild and restart

```
sudo docker-compose -f docker-compose.prod.yml up -d --build
```

Stop containers

```
sudo docker-compose -f docker-compose.prod.yml down
```

Clean up

```
sudo docker system prune -a
```

Database Commands (On EC2 via SSM)

Backup database

```
cd /data/db
```

```
sudo sqlite3 app.sqlite ".backup /tmp/backup-$(date +  
    %Y%m%d).sqlite"
```

Check database

```
sudo sqlite3 app.sqlite
```

```
.tables
```

```
.schema
```



```
SELECT COUNT(*) FROM messages;  
.quit
```

```
# Restore database
```

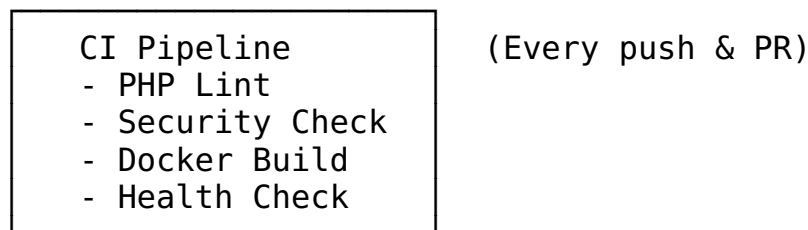
```
sudo cp /data/db/app.sqlite /data/db/app.sqlite.backup  
sudo sqlite3 /data/db/app.sqlite < backup.sql
```

CI/CD Pipeline (GitHub Actions)

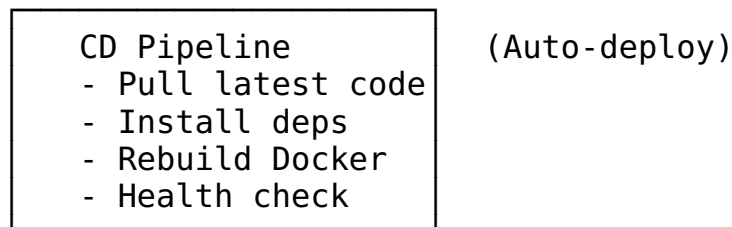
The project includes automated CI/CD pipelines using GitHub Actions. Every code push is automatically tested, built, and deployed.

Pipeline Overview

Developer pushes code to GitHub



(main branch only)



Application Live at
<https://taxplanner.app>

Three Pipelines

Pipeline	Trigger	What It Does
CI - Build & Test	Every push/PR	Lint, security check, Docker build, health check
CD - Deploy to AWS	Push to main	Deploys code to EC2 via SSM
Terraform	Changes in terraform/	Plans and applies infrastructure changes

Required GitHub Secrets

Go to **Repository Settings** → **Secrets and Variables** → **Actions** and add:

Secret Name	Description	Required
AWS_ACCESS_KEY_ID	AWS IAM access key	Yes
AWS_SECRET_ACCESS_KEY	AWS IAM secret key	Yes
EC2_INSTANCE_ID	EC2 instance ID (e.g., i-0abc123)	Yes
GOOGLE_CLIENT_ID	Google OAuth Client ID	Yes
GOOGLE_CLIENT_SECRET	Google OAuth Client Secret	Yes

How to Set Up Secrets

Get EC2 Instance ID from Terraform

```
cd terraform
terraform output ec2_instance_id
```

Then in GitHub: 1. Go to <https://github.com/appcropolisdevops/awspoc/settings/secrets/actions> 2. Click “New repository secret” 3. Add each secret listed above

CI Pipeline (ci.yml)

Triggers: Every push to main/develop and every PR

Steps: 1. **PHP Lint** - Checks all PHP files for syntax errors 2. **Security**

Check - Scans for hardcoded secrets, verifies .env is not committed 3.

Docker Build - Builds Docker images and runs containers 4. **Health Check**
- Verifies application responds on port 8080

CD Pipeline (deploy.yml)

Triggers: Push to main (ignores docs/ and .md changes)

Steps: 1. Runs CI checks first 2. Configures AWS credentials 3. Verifies EC2 instance is running 4. **Deploys via SSM** (no SSH needed): - Pulls latest code from GitHub - Installs Composer dependencies - Rebuilds and restarts Docker containers - Runs health check 5. Outputs deployment summary

Manual Trigger: Go to Actions → CD - Deploy to AWS → Run workflow

Terraform Pipeline (terraform.yml)

Triggers: Changes to terraform/ files

On Pull Request: - Runs terraform plan - Posts plan output as PR comment

On Push to Main: - Runs terraform apply automatically

Manual Options: - Plan (preview changes) - Apply (deploy infrastructure) - Destroy (teardown infrastructure)

Deployment Flow

Automatic (recommended):

```
# Make code changes
git add .
git commit -m "Update feature X"
git push origin main
# → CI runs → CD deploys automatically
```

Manual:

```
# Go to GitHub → Actions → CD - Deploy to AWS
# Click "Run workflow" → Select branch → Run
```

Important Notes

Cost Management







Estimated Monthly Costs (us-east-2): - EC2 t3.small: ~\$15 - ALB: ~\$20 - NAT Gateway: ~\$35 - EBS volumes: ~\$5 - Data transfer: Variable - S3, CloudWatch: <\$5 - **Total: ~\$80-100/month**

To reduce costs: - Use t3.micro instead of t3.small - Remove NAT gateway (deploy EC2 in public subnet) - Use VPC endpoints instead of NAT - Enable S3 lifecycle policies

Security Best Practices

1. **Secrets:** Always use Secrets Manager, never hardcode
2. **SSH:** Use SSM Session Manager, disable SSH if possible
3. **Updates:** Regularly update AMI and packages
4. **Backups:** Verify automated backups are working
5. **Monitoring:** Check CloudWatch logs regularly
6. **IAM:** Follow least privilege principle
7. **Encryption:** All data encrypted at rest and in transit

Compliance (HIPAA)

The infrastructure is designed with HIPAA compliance in mind: -  Encryption at rest (EBS, S3) -  Encryption in transit (HTTPS, TLS) -  Audit logging (CloudWatch, 365-day retention) -  Access controls (IAM, Security Groups) -  Data backups (automated daily) -  Network isolation (private subnets)

Note: Full HIPAA compliance requires additional operational procedures, BAA with AWS, and proper data handling policies.

Support Resources

- **Documentation:** /docs folder in repository
 - **Terraform Docs:** <https://registry.terraform.io/providers/hashicorp/aws/latest/docs>
 - **AWS Support:** <https://aws.amazon.com/support>
 - **Application:** <https://taxplanner.app>
 - **Repository:** <https://github.com/appcropolisdevops/awspoc>
-

Document Version: 2.0 **Last Updated:** February 4, 2026 **Application:** Secure Messaging Platform (taxplanner.app) **AWS Region:** us-east-2 (Ohio) **Infrastructure:** 100% Terraform-managed **DevOps Engineer:** Naeem Dosh **Platform:** Fiverr