JARWIS AI PENTEST

AWS Deployment Plan

Comprehensive Step-by-Step Guide

Version: 1.0

Date: January 4, 2026

Project: Jarwis AI Penetration Testing Framework

Platform: Amazon Web Services (AWS)

TABLE OF CONTENTS

1. EXECUTIVE SUMMARY

This document provides a comprehensive step-by-step guide to deploy the Jarwis AI Penetration Testing Framework on AWS infrastructure. The deployment architecture includes:

- **Backend API** (FastAPI/Python) deployed on ECS Fargate
- **Frontend** (React) hosted on S3 with CloudFront CDN
- **Database** (PostgreSQL) on Amazon RDS with Multi-AZ
- **AI/LLM** powered by Amazon Bedrock (replacing Ollama)
- **Security scanning infrastructure** with proper network isolation

2. HIGH-LEVEL ARCHITECTURE

The following diagram illustrates the complete AWS infrastructure for Jarwis:

Component	AWS Service	Configuration
Frontend	S3 + CloudFront	Static React build with global CDN
API Gateway	Application Load Balancer	HTTPS termination, health checks
Backend API	ECS Fargate	2 tasks, 1 vCPU, 2GB RAM each
AI Engine	Amazon Bedrock	Claude 3.5 Sonnet model
Database	RDS PostgreSQL	db.t3.medium, Multi-AZ, encrypted
Secrets	Secrets Manager	Database credentials, API keys
Monitoring	CloudWatch	Logs, metrics, alarms
DNS	Route 53	Domain routing

Network Architecture

Subnet	CIDR Block	Purpose
VPC	10.0.0.0/16	Main virtual private cloud
Public Subnet 1a	10.0.1.0/24	ALB, NAT Gateway
Public Subnet 1b	10.0.4.0/24	ALB (Multi-AZ)
Private Subnet	10.0.2.0/24	ECS Fargate tasks
Database Subnet 1a	10.0.3.0/24	RDS Primary
Database Subnet 1b	10.0.5.0/24	RDS Standby (Multi-AZ)

3. PHASE 1: AWS ACCOUNT SETUP

3.1 Prerequisites

- AWS Account with billing enabled
- AWS CLI v2 installed and configured
- Docker Desktop installed
- Node.js 18+ for frontend build
- Domain name (optional but recommended)

3.2 Enable Required AWS Services

Navigate to AWS Console and enable:

EC2, VPC, RDS, S3, CloudFront, ECR, ECS, Amazon Bedrock, Secrets Manager, CloudWatch, Route 53, ACM (Certificate Manager)

3.3 Request Bedrock Model Access

Important: Amazon Bedrock requires explicit model access approval. Navigate to Amazon Bedrock → Model access → Request access to 'Anthropic Claude 3.5 Sonnet'. Approval is typically instant.

4. PHASE 2: INFRASTRUCTURE SETUP

4.1 Create VPC

Command:

```
aws ec2 create-vpc --cidr-block 10.0.0.0/16 \
    --tag-specifications 'ResourceType=vpc,Tags=[{Key=Name,Value=jarwis-vpc}]'
```

4.2 Create Subnets

Create 5 subnets across 2 availability zones for high availability:

```
# Public Subnet 1a
aws ec2 create-subnet --vpc-id vpc-xxx --cidr-block 10.0.1.0/24 \
    --availability-zone us-east-1a

# Private Subnet
aws ec2 create-subnet --vpc-id vpc-xxx --cidr-block 10.0.2.0/24 \
    --availability-zone us-east-1a

# Database Subnets (2 required for RDS)
aws ec2 create-subnet --vpc-id vpc-xxx --cidr-block 10.0.3.0/24 \
    --availability-zone us-east-1a
aws ec2 create-subnet --vpc-id vpc-xxx --cidr-block 10.0.5.0/24 \
    --availability-zone us-east-1b
```

4.3 Security Groups

Security Group	Inbound Rules	Purpose
jarwis-alb-sg	TCP 443, 80 from 0.0.0.0/0	Allow HTTPS/HTTP to load balancer
jarwis-api-sg	TCP 8000 from ALB SG only	API access from load balancer
jarwis-rds-sg	TCP 5432 from API SG only	Database from API only

5. PHASE 3: DATABASE SETUP (RDS)

5.1 Store Credentials in Secrets Manager

```
aws secretsmanager create-secret \
    --name jarwis/database/credentials \
    --secret-string '{"username":"jarwis_admin","password":"YOUR_SECURE_PASSWORD"}'
```

5.2 Create RDS Instance

```
aws rds create-db-instance \
    --db-instance-identifier jarwis-db \
    --db-instance-class db.t3.medium \
    --engine postgres \
    --engine-version 15.4 \
    --allocated-storage 100 \
    --storage-type gp3 \
    --multi-az \
    --storage-encrypted \
    --no-publicly-accessible
```

5.3 RDS Configuration

Setting	Value	Reason
Instance Class	db.t3.medium	2 vCPU, 4GB RAM - suitable for medium workloads
Storage	100GB gp3	Fast SSD with baseline 3000 IOPS
Multi-AZ	Yes	High availability with automatic failover
Encryption	Yes	Security compliance requirement
Backup Retention	7 days	Point-in-time recovery

6. PHASE 4: DOCKER & ECR SETUP

6.1 Create ECR Repositories

```
aws ecr create-repository --repository-name jarwis-api \
    --image-scanning-configuration scanOnPush=true

aws ecr create-repository --repository-name jarwis-worker \
    --image-scanning-configuration scanOnPush=true
```

6.2 Dockerfile.api (Backend)

```
FROM python:3.11-slim
WORKDIR /app

# Install system dependencies
RUN apt-get update && apt-get install -y wget curl nmap

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

# Install Playwright
RUN playwright install chromium
RUN playwright install-deps chromium

# Copy application
COPY api/ ./api/
COPY core/ ./core/
COPY attacks/ ./attacks/
COPY database/ ./database/

EXPOSE 8000
CMD ["uvicorn", "api.app:app", "--host", "0.0.0.0", "--port", "8000"]
```

6.3 Build and Push

```
# Login to ECR
aws ecr get-login-password --region us-east-1 | docker login --username AWS \
    --password-stdin ACCOUNT_ID.dkr.ecr.us-east-1.amazonaws.com

# Build and push
docker build -f Dockerfile.api -t jarwis-api:latest .
docker tag jarwis-api:latest ACCOUNT_ID.dkr.ecr.us-east-1.amazonaws.com/jarwis-api:latest
docker push ACCOUNT_ID.dkr.ecr.us-east-1.amazonaws.com/jarwis-api:latest
```

7. PHASE 5: AMAZON BEDROCK INTEGRATION

This section explains how to replace Ollama with Amazon Bedrock for AI-powered security analysis.

7.1 Bedrock Model Selection

Model ID	Use Case	Cost (per 1K tokens)
anthropic.claude-3-5-sonnet-20241022-v2:0	RECOMMENDED - Best for security analysis	$0.003 / $0.015
anthropic.claude-3-haiku-20240307-v1:0	Fast, cheaper alternative	$0.00025 / $0.00125
amazon.titan-text-premier-v1:0	AWS native option	$0.0005 / $0.0015

7.2 Code Changes to ai_planner.py

```
# Add to imports
import boto3
from botocore.config import Config

# Initialize Bedrock client
config = Config(region_name="us-east-1")
client = boto3.client("bedrock-runtime", config=config)

# Invoke model
response = client.invoke_model(
    modelId="anthropic.claude-3-5-sonnet-20241022-v2:0",
    contentType="application/json",
    body=json.dumps({
        "anthropic_version": "bedrock-2023-05-31",
        "max_tokens": 4096,
        "messages": messages
    })
)
```

7.3 IAM Policy for Bedrock

```
{
    "Version": "2012-10-17",
    "Statement": [{
        "Effect": "Allow",
        "Action": [
            "bedrock:InvokeModel",
            "bedrock:InvokeModelWithResponseStream"
        ],
        "Resource": [
            "arn:aws:bedrock:us-east-1::foundation-model/anthropic.claude-*"
        ]
    }]
}
```

8. PHASE 6: ECS FARGATE DEPLOYMENT

8.1 Create ECS Cluster

```
aws ecs create-cluster --cluster-name jarwis-cluster \
    --capacity-providers FARGATE FARGATE_SPOT
```

8.2 Task Definition Configuration

Setting	Value	Reason
CPU	1024 (1 vCPU)	Sufficient for API + light scanning
Memory	2048 MB	Playwright needs ~1GB + app overhead
Desired Count	2	High availability
Max Count	10	Auto-scaling for heavy scan loads
Launch Type	FARGATE	Serverless, no EC2 management

8.3 Environment Variables

Variable	Value
DB_TYPE	postgresql
POSTGRES_HOST	jarwis-db.xxx.rds.amazonaws.com
AI_PROVIDER	bedrock
AI_MODEL	anthropic.claude-3-5-sonnet-20241022-v2:0
AWS_REGION	us-east-1

9. PHASE 7: FRONTEND DEPLOYMENT

9.1 Update API Configuration

Modify jarwisfrontend/src/api.js:

```
// Change from:
const BASE_URL = "https://jarwis-api.onrender.com/api";

// To:
const BASE_URL = process.env.REACT_APP_API_URL || "https://api.jarwis.yourdomain.com/api";
```

9.2 Build and Deploy to S3

```
cd jarwisfrontend
npm install
npm run build

# Create S3 bucket
aws s3 mb s3://jarwis-frontend-prod

# Upload build
aws s3 sync build/ s3://jarwis-frontend-prod --delete
```

9.3 CloudFront Distribution

Create CloudFront distribution pointing to S3 bucket with:

- HTTPS only (redirect HTTP to HTTPS)
- Custom SSL certificate from ACM
- Error page redirect to index.html for SPA routing
- Gzip compression enabled
- Cache TTL: 86400 seconds (1 day)

10. PHASE 8: DNS & SSL SETUP

10.1 Request SSL Certificates

```
# Frontend certificate (MUST be in us-east-1 for CloudFront)
aws acm request-certificate --domain-name jarwis.yourdomain.com \
    --validation-method DNS --region us-east-1

# API certificate
aws acm request-certificate --domain-name api.jarwis.yourdomain.com \
    --validation-method DNS --region us-east-1
```

10.2 DNS Records (Route 53)

Record	Type	Target
jarwis.yourdomain.com	A (Alias)	CloudFront distribution
api.jarwis.yourdomain.com	A (Alias)	Application Load Balancer

13. DEPLOYMENT CHECKLIST

Phase	Task	Status
Pre-Deploy	AWS Account configured	■
Pre-Deploy	Bedrock model access approved	■
Pre-Deploy	Domain name available	■
Infrastructure	VPC and subnets created	■
Infrastructure	Security groups configured	■
Database	RDS instance running	■
Database	Credentials in Secrets Manager	■
Containers	ECR repositories created	■
Containers	Docker images pushed	■
ECS	Cluster created	■
ECS	Task definitions registered	■
ECS	Services running	■
Frontend	S3 bucket created	■
Frontend	CloudFront distribution active	■
DNS/SSL	Certificates issued	■
DNS/SSL	DNS records configured	■
Testing	Health checks passing	■
Testing	End-to-end scan completed	■

14. ESTIMATED MONTHLY COSTS

Service	Configuration	Est. Monthly Cost
ECS Fargate	2 tasks × 1 vCPU × 2GB	$70
RDS PostgreSQL	db.t3.medium, Multi-AZ	$120
Application Load Balancer	Per hour + LCU	$25
NAT Gateway	Per hour + data transfer	$45
S3 + CloudFront	10GB storage, 100GB transfer	$15
Amazon Bedrock	~500K tokens/month	$10
Secrets Manager	2 secrets	$1
CloudWatch	Logs + alarms	$10
Route 53	Hosted zone + queries	$2
TOTAL		**$298/month**

Cost Optimization Tips

- Use FARGATE_SPOT for scan workers (up to 70% savings)
- Use Reserved Capacity for RDS (up to 60% savings for 1-year commitment)
- Enable S3 Intelligent-Tiering for reports storage
- Use CloudFront caching to reduce origin requests
- Consider single-AZ RDS for development/staging environments

15. TROUBLESHOOTING GUIDE

15.1 ECS Task Fails to Start

Check CloudWatch logs:

```
aws logs get-log-events --log-group-name /ecs/jarwis-api \
    --log-stream-name ecs/jarwis-api/TASK_ID
```

Common causes:

- Database connection refused → Check security group rules
- Image pull failed → Verify ECR permissions
- Memory exceeded → Increase task memory in task definition

15.2 Bedrock Access Denied

Verify IAM role has Bedrock permissions:

```
aws iam get-role-policy --role-name jarwis-task-role \
    --policy-name BedrockAccess
```

Ensure model is enabled in Bedrock console under Model Access.

15.3 Frontend CORS Errors

Checklist:

- Ensure ALB has correct CORS headers configured
- Verify API URL in frontend matches ALB domain exactly
- Check CloudFront behavior settings allow OPTIONS method
