



EUGENE PLATFORM

AWS Hosting Guide

End-to-end deployment runbook: S3 + CloudFront for the static frontend, ECS Fargate for services, EC2 for Neo4j

PREPARED FOR	CSL Behring — Business Development & R&D
PROGRAMME	Eugene Knowledge Graph & Agentic AI Platform
DATE	May 2, 2026
CLASSIFICATION	Confidential — Internal Use Only
VERSION	2.1 (Book Edition)

TABLE OF CONTENTS

FRONT MATTER	3
About this Guide	3
Target Architecture (Summary)	3
SECTION 1 — PREREQUISITES	3
1. Local Workstation Setup	3
2. Pre-flight Checks	4
SECTION 2 — NETWORKING FOUNDATION	4
3. Create a VPC (One-Time)	4
4. Security Groups	4
SECTION 3 — STATIC FRONTEND ON S3 + CLOUDFRONT	5
5. Build the Next.js UI for Static Export	5
6. Create the S3 Bucket	5
7. Upload the Build	5
8. Request a TLS Certificate (ACM)	6
9. CloudFront Distribution	6
10. Route 53 — Point Domain at CloudFront	6
SECTION 4 — BACKEND SERVICES ON ECS FARGATE	6
11. Create ECR Repositories	6
12. Build & Push Images	7
13. Create the ECS Cluster	7
14. Task Definitions	7
15. Application Load Balancer + Target Groups	8
16. ECS Services	8
SECTION 5 — NEO4J KNOWLEDGE GRAPH	8
17. Launch a Neo4j EC2 Instance	8
18. Internal DNS for Neo4j	9
19. Seed Initial Data	9
SECTION 6 — SECRETS, IAM, AND OBSERVABILITY	9
20. Secrets Manager	10

21. IAM Roles	10
22. CloudWatch Logs + Metrics	10
SECTION 7 — CI / CD	10
23. GitHub Actions → ECR	10
24. Frontend Sync on Every Release	11
SECTION 8 — SMOKE TESTS & CUTOVER	11
25. End-to-End Smoke Test	11
26. Rollback Plan	12
SECTION 9 — OPERATIONS	12
27. Cost Outlook (us-east-1, monthly)	12
28. Monitoring + Alerts	12
29. Security Hardening Checklist	12
APPENDICES	13
Appendix A — Terraform Module Layout	13
Appendix B — Troubleshooting	13
Appendix C — Useful CLI One-Liners	13

FRONT MATTER

About this Guide

This guide walks through hosting the Eugene biomedical knowledge graph and agentic AI platform on AWS. It is written as a runbook: every step is a concrete action with the exact AWS service, console navigation, and CLI commands required.

■ **Important** — Amazon S3 alone is a static-object store. It can host the compiled Next.js UI (HTML / JS / CSS / images), but it cannot run the Eugene backend services (FastAPI Core API, MCP Server, Strands Agent, Streamlit UI, Neo4j). This guide therefore uses S3 for the static frontend and ECS Fargate + EC2 for the backend. If you literally need only an S3 deployment, you can host the docs and the Next.js UI on S3 and point it at an existing Eugene backend running elsewhere.

Target Architecture (Summary)

The deployment splits the stack across the following AWS services:

Tier	AWS Service	Why
Static frontend	S3 + CloudFront + Route 53 + ACM	Cheap, global, cached; perfect for a Next.js static export
Agent + Core API + MCP	ECS Fargate	Containerised Python services, auto-scale, no servers to patch
Streamlit UI (optional)	ECS Fargate	Streamlit is a Python server — cannot live on S3
Knowledge graph (Neo4j)	EC2 (or Amazon Neptune)	Neo4j needs persistent storage + tuning
Load balancing	Application Load Balancer	TLS termination + path-based routing
Container registry	ECR	Stores agent / core / mcp / ui images
Secrets	AWS Secrets Manager + KMS	Tokens, DB credentials, OpenAI / Anthropic keys
Observability	CloudWatch + X-Ray	Logs, metrics, distributed tracing
Identity	Microsoft Entra ID + Cognito (federated)	Existing CSL SSO
CI / CD	CodePipeline + CodeBuild + CodeDeploy	Automated image build + ECS rolling deploy

SECTION 1 — PREREQUISITES

1. Local Workstation Setup

- AWS account with admin access (or scoped role allowing IAM, S3, EC2, ECS, ECR, ALB, CloudFront, ACM, Route53, Secrets Manager, KMS, CloudWatch).
- AWS CLI v2 installed and configured (`aws configure` with an Access Key + region — e.g. eu-central-1 or us-east-1).
- Docker Desktop ≥ 4.30 running locally.

- Node.js 20 LTS + npm 10 for building the Next.js frontend.
- Terraform $\geq$ 1.7 OR AWS CDK 2.x if you prefer programmatic IaC. CloudFormation works too.
- A domain name you control (e.g. eugene.csl.example) for the public frontend. Route 53-managed is easiest.
- Microsoft Entra ID tenant + an App Registration (same as the current Eugene OAuth setup).
- An OpenAI or Anthropic API key (used by the agent for LLM inference).

2. Pre-flight Checks

```
# 1. Confirm AWS identity
aws sts get-caller-identity

# 2. Confirm region
aws configure get region

# 3. Confirm Docker is running
docker info | head -5

# 4. Confirm Node + npm versions
node --version # v20.x
npm --version # 10.x
```

SECTION 2 — NETWORKING FOUNDATION

3. Create a VPC (One-Time)

Eugene needs a private VPC with both public subnets (for the ALB) and private subnets (for ECS tasks and Neo4j). Use AWS-managed VPC creation for sane defaults.

1. Console → VPC → Create VPC → 'VPC and more'.
2. Name tag: eugene-prod-vpc; IPv4 CIDR: 10.0.0.0/16.
3. Availability zones: 2 (e.g. eu-central-1a, eu-central-1b).
4. Public subnets: 2 (one per AZ); private subnets: 2 (one per AZ).
5. NAT gateway: 1 per AZ (or 1 to save cost — accept the SPOF in dev).
6. VPC endpoints: enable S3 Gateway endpoint (free).
7. Click 'Create VPC'. Note the VPC ID, subnet IDs, and route-table IDs that come out.

4. Security Groups

Create three security groups via Console → EC2 → Security Groups:

Name	Inbound rules
eugene-alb-sg	443 from 0.0.0.0/0 (HTTPS); 80 from 0.0.0.0/0 (HTTP redirect)
eugene-ecs-sg	8000-8501 from eugene-alb-sg only
eugene-neo4j-sg	7687 from eugene-ecs-sg only; 7474 from your bastion IP only

SECTION 3 — STATIC FRONTEND ON S3 + CLOUDFRONT

5. Build the Next.js UI for Static Export

The Next.js v2 UI ships with server-side features (API routes, SSR). For S3 hosting, configure static export so it produces pure HTML, CSS, and JS. Server-only API routes will be retired (the agent and core API live on ECS and the frontend talks to them directly).

```
# In agents/eugene-agent-ui-next/next.config.js, add:
module.exports = {
  output: 'export',
  images: { unoptimized: true },
  trailingSlash: true,
  env: {
    NEXT_PUBLIC_AGENT_API: 'https://api.eugene.csl.example/agent/api',
    NEXT_PUBLIC_CORE_API: 'https://api.eugene.csl.example/core',
  },
};
```

```
cd agents/eugene-agent-ui-next
npm install
npm run build # produces ./out/ directory
ls out # confirm static HTML / _next / assets
```

■ *Streamlit cannot be statically exported — it must run as a Python server. If you need the Streamlit UI in production, deploy it as an additional ECS Fargate service (see Section 5).*

6. Create the S3 Bucket

```
# Replace <region> and <bucket-name>; the bucket name must be globally unique.
aws s3api create-bucket \
--bucket eugene-frontend-prod \
--region eu-central-1 \
--create-bucket-configuration LocationConstraint=eu-central-1

# Block public access (CloudFront will read via OAC — Origin Access Control)
aws s3api put-public-access-block \
--bucket eugene-frontend-prod \
--public-access-block-configuration \
BlockPublicAcls=true,IgnorePublicAcls=true,BlockPublicPolicy=true,RestrictPublicBuckets=true

# Enable versioning (rollback safety) and default encryption
aws s3api put-bucket-versioning --bucket eugene-frontend-prod \
--versioning-configuration Status=Enabled
aws s3api put-bucket-encryption --bucket eugene-frontend-prod \
--server-side-encryption-configuration \
'{"Rules":[{"ApplyServerSideEncryptionByDefault":{"SSEAlgorithm":"AES256"}}}]'
```

7. Upload the Build

```
aws s3 sync agents/eugene-agent-ui-next/out/ s3://eugene-frontend-prod/ \
--delete \
--cache-control 'public, max-age=31536000, immutable' \
--exclude '*.html' \
--exclude '*.json'

# Re-upload HTML / JSON with shorter cache so deployments propagate fast
aws s3 sync agents/eugene-agent-ui-next/out/ s3://eugene-frontend-prod/ \
--cache-control 'public, max-age=60, must-revalidate' \
```

```
--content-type 'text/html' \  
--exclude '*' \  
--include '*.html' --include '*.json'
```

8. Request a TLS Certificate (ACM)

CloudFront requires ACM certificates in us-east-1 regardless of where the backend lives.

```
aws acm request-certificate \  
--domain-name eugene.csl.example \  
--subject-alternative-names api.eugene.csl.example www.eugene.csl.example \  
--validation-method DNS \  
--region us-east-1
```

Then go to ACM → click the pending cert → 'Create records in Route 53' to auto-add the DNS validation CNAME records. Wait ~5 minutes for validation.

9. CloudFront Distribution

1. Console → CloudFront → Create distribution.
2. Origin domain: select the eugene-frontend-prod S3 bucket (NOT the website endpoint).
3. Origin access: 'Origin access control settings' → Create OAC named eugene-frontend-oac.
4. After distribution is created, copy the OAC bucket policy back to the S3 bucket (CloudFront shows a 'Copy policy' button).
5. Default cache behaviour: redirect HTTP → HTTPS; allowed methods GET, HEAD; cache policy CachingOptimized.
6. Default root object: index.html.
7. Custom error pages: 403 → /index.html (200) and 404 → /index.html (200). This is the SPA-routing pattern that lets Next.js client-side routes work.
8. Alternate domain names (CNAMEs): eugene.csl.example.
9. Custom SSL certificate: pick the ACM cert created in step 8.
10. WAF: optional but recommended — attach AWS Managed Rules core rule set.
11. Click 'Create distribution'. Distribution domain looks like dxxxxx.cloudfront.net — note it.

10. Route 53 — Point Domain at CloudFront

```
# Console: Route 53 → Hosted zones → csl.example → Create record  
# Record name: eugene  
# Type: A — IPv4  
# Alias: Yes  
# Route traffic: Alias to CloudFront distribution → dxxxxx.cloudfront.net  
# Or CLI:  
aws route53 change-resource-record-sets --hosted-zone-id Z123 --change-batch file://route53.json
```

SECTION 4 — BACKEND SERVICES ON ECS FARGATE

11. Create ECR Repositories

```
for svc in eugene_ws eugene_mcp eugene_agent_ws eugene_agent_ui; do  
aws ecr create-repository --repository-name eugene/${svc} --region eu-central-1 \  
done
```

```
--image-scanning-configuration scanOnPush=true \  
--encryption-configuration encryptionType=KMS  
done
```

12. Build & Push Images

```
ACCT=$(aws sts get-caller-identity --query Account --output text)  
REGION=eu-central-1  
REGISTRY=${ACCT}.dkr.ecr.${REGION}.amazonaws.com  
  
aws ecr get-login-password --region ${REGION} | \  
docker login --username AWS --password-stdin ${REGISTRY}  
  
# Build (the project already has these Dockerfiles)  
for svc in eugene_ws eugene_mcp eugene_agent_ws eugene_agent_ui; do  
docker build -f docker/${svc}/Dockerfile -t ${REGISTRY}/eugene/${svc}:v1.0 . \  
--platform linux/amd64  
docker push ${REGISTRY}/eugene/${svc}:v1.0  
done
```

13. Create the ECS Cluster

```
aws ecs create-cluster --cluster-name eugene-prod \  
--capacity-providers FARGATE FARGATE_SPOT \  
--default-capacity-provider-strategy capacityProvider=FARGATE,weight=1
```

14. Task Definitions

Each service needs its own task definition. Save the following JSON to a file (e.g. eugene-ws.task-def.json), then register:

```
aws ecs register-task-definition --cli-input-json file://eugene-ws.task-def.json  
# Repeat for eugene-mcp, eugene-agent-ws, eugene-agent-ui  
  
// eugene-ws.task-def.json – example  
{  
  "family": "eugene-ws",  
  "networkMode": "awsvpc",  
  "requiresCompatibilities": ["FARGATE"],  
  "cpu": "1024", "memory": "2048",  
  "executionRoleArn": "arn:aws:iam::ACCT:role/ecsTaskExecutionRole",  
  "taskRoleArn": "arn:aws:iam::ACCT:role/eugene-task-role",  
  "containerDefinitions": [{  
    "name": "eugene-ws",  
    "image": "ACCT.dkr.ecr.eu-central-1.amazonaws.com/eugene/eugene_ws:v1.0",  
    "portMappings": [{ "containerPort": 8000 }],  
    "essential": true,  
    "secrets": [  
      {"name": "NEO4J_PASSWORD",  
       "valueFrom": "arn:aws:secretsmanager:eu-central-1:ACCT:secret:eugene/neo4j-password"},  
      {"name": "EUGENE_CLIENT_SECRET",  
       "valueFrom": "arn:aws:secretsmanager:eu-central-1:ACCT:secret:eugene/jwt-secret"}  
    ],  
    "environment": [  
      {"name": "NEO4J_URI", "value": "bolt://eugene-neo4j.prod.local:7687"},  
      {"name": "NEO4J_USERNAME", "value": "neo4j"},  
      {"name": "ENVIRONMENT", "value": "production"}  
    ],  
    "logConfiguration": {  
      "logDriver": "awslogs",  
      "options": {
```



```
"awslogs-group": "/ecs/eugene-ws",
"awslogs-region": "eu-central-1",
"awslogs-stream-prefix": "ws"
},
"healthCheck": {
"command": ["CMD-SHELL", "curl -f http://localhost:8000/health || exit 1"],
"interval": 30, "timeout": 5, "retries": 3, "startPeriod": 30
}
}]
}
```

15. Application Load Balancer + Target Groups

1. Console → EC2 → Load Balancers → Create → Application Load Balancer.
2. Name: eugene-alb-prod; Scheme: internet-facing (or internal if behind corporate VPN).
3. VPC: eugene-prod-vpc; Mappings: 2 public subnets; Security group: eugene-alb-sg.
4. Listeners: HTTPS:443 (attach ACM cert from step 8); HTTP:80 → redirect to 443.
5. Create target groups (Type=IP, Port=8000/8001/8443/8501): eugene-ws-tg, eugene-mcp-tg, eugene-agent-tg, eugene-streamlit-tg.
6. Health-check path for each: /health.
7. On the HTTPS:443 listener, add path-based rules: /core/* → eugene-ws-tg; /agent/api/* → eugene-agent-tg; /mcp/* → eugene-mcp-tg; /streamlit/* → eugene-streamlit-tg; default → return-403.

16. ECS Services

```
for svc in eugene-ws eugene-mcp eugene-agent-ws eugene-agent-ui; do
aws ecs create-service \
--cluster eugene-prod \
--service-name ${svc} \
--task-definition ${svc} \
--desired-count 2 \
--launch-type FARGATE \
--network-configuration 'awsVpcConfiguration={subnets=[subnet-priv1,subnet-priv2],securityGroups=[sg-ecs],assignPublicIp=DISABLED}' \
--load-balancers "targetGroupArn=arn:aws:elasticloadbalancing:...:targetgroup/${svc}-tg,containerName=${svc},containerPort=8000" \
--deployment-configuration 'maximumPercent=200,minimumHealthyPercent=100' \
--health-check-grace-period-seconds 60
done
```

SECTION 5 — NEO4J KNOWLEDGE GRAPH

17. Launch a Neo4j EC2 Instance

Two viable options. The simpler one is to run Neo4j Enterprise on a dedicated EC2 instance with EBS storage. The fully-managed alternative is Amazon Neptune (Cypher-compatible), but it requires schema translation work first — start with Neo4j on EC2.

1. Console → EC2 → Launch instance.
2. Name: eugene-neo4j-prod.
3. AMI: Amazon Linux 2023 (or Ubuntu 24.04 LTS).

4. Instance type: r6i.2xlarge for production (8 vCPU, 64 GB RAM). r6i.xlarge for staging.
5. Network: eugene-prod-vpc, private subnet, security group eugene-neo4j-sg.
6. Storage: 200 GB gp3 EBS, encrypted with the eugene-data-cmk KMS key.
7. User data: install Neo4j 5.x community or enterprise (see commands below).
8. IAM role: attach an instance profile granting CloudWatch Agent + Systems Manager access.

```
# Neo4j install user-data (Amazon Linux 2023)
#!/bin/bash
yum update -y
yum install -y java-17-amazon-corretto
rpm --import https://debian.neo4j.com/neotechnology.gpg.key
cat > /etc/yum.repos.d/neo4j.repo <<EOF
[neo4j]
name=Neo4j RPM Repository
baseurl=https://yum.neo4j.com/stable/5
enabled=1
gpgcheck=1
EOF
yum install -y neo4j-5.26.0
echo 'server.default_listen_address=0.0.0.0' >> /etc/neo4j/neo4j.conf
echo 'server.memory.pagecache.size=20G' >> /etc/neo4j/neo4j.conf
echo 'server.memory.heap.initial_size=8G' >> /etc/neo4j/neo4j.conf
echo 'server.memory.heap.max_size=8G' >> /etc/neo4j/neo4j.conf
neo4j-admin dbms set-initial-password "$(aws secretsmanager get-secret-value --secret-id
eugene/neo4j-password --query SecretString --output text)"
systemctl enable neo4j && systemctl start neo4j
```

18. Internal DNS for Neo4j

```
# Use Route 53 private hosted zone (created with the VPC)
# Add an A record for eugene-neo4j.prod.local → the EC2 private IP
aws route53 change-resource-record-sets \
--hosted-zone-id ZONE_ID \
--change-batch '{
"Changes":[{"Action":"CREATE","ResourceRecordSet":{"
"Name":"eugene-neo4j.prod.local","Type":"A","TTL":60,
"ResourceRecords":[{"Value":"10.0.20.42"}]}]}'
```

19. Seed Initial Data

From a developer workstation (over Session Manager or VPN):

```
# Open Session Manager port-forward
aws ssm start-session --target i-0abc123 \
--document-name AWS-StartPortForwardingSession \
--parameters portNumber=7687,localPortNumber=7687

# In another terminal: pump the seed scripts in
for f in bin/seed/csl_behring_assets.cypher \
bin/seed/biogen_assets.cypher \
bin/seed/biogen_assets_patch_labels.cypher \
bin/seed/csl_drug_node_properties.cypher; do
cat "$f" | cypher-shell -a bolt://localhost:7687 -u neo4j \
-p "$(aws secretsmanager get-secret-value --secret-id eugene/neo4j-password --query SecretString
--output text)"
done
```

SECTION 6 — SECRETS, IAM, AND OBSERVABILITY

20. Secrets Manager

```
aws secretsmanager create-secret --name eugene/neo4j-password \
--secret-string 'CHANGE_ME_STRONG_PASSWORD' \
--kms-key-id alias/eugene-data-cmk

aws secretsmanager create-secret --name eugene/jwt-secret \
--secret-string "$(openssl rand -hex 48)"

aws secretsmanager create-secret --name eugene/openai-key \
--secret-string 'sk-proj-...'

aws secretsmanager create-secret --name eugene/entra-client-secret \
--secret-string '...'
```

21. IAM Roles

Two roles are needed: `ecsTaskExecutionRole` (pulls images, writes logs) and `eugene-task-role` (the task's own runtime role for Secrets Manager + KMS access).

```
# ecsTaskExecutionRole – managed policy + ECR secrets read
aws iam create-role --role-name ecsTaskExecutionRole \
--assume-role-policy-document file://ecs-trust.json
aws iam attach-role-policy --role-name ecsTaskExecutionRole \
--policy-arn arn:aws:iam::aws:policy/service-role/AmazonECSTaskExecutionRolePolicy

# eugene-task-role – fine-grained read for Eugene's secrets
aws iam create-role --role-name eugene-task-role \
--assume-role-policy-document file://ecs-trust.json
aws iam put-role-policy --role-name eugene-task-role \
--policy-name eugene-secrets-read \
--policy-document file://eugene-secrets-policy.json
```

22. CloudWatch Logs + Metrics

```
for svc in eugene-ws eugene-mcp eugene-agent-ws eugene-agent-ui; do
aws logs create-log-group --log-group-name /ecs/${svc} --region eu-central-1
aws logs put-retention-policy --log-group-name /ecs/${svc} --retention-in-days 30
done

# Container Insights for cluster-level metrics
aws ecs put-account-setting --name containerInsights --value enabled
```

SECTION 7 — CI / CD

23. GitHub Actions → ECR

```
# .github/workflows/deploy.yml (excerpt)
name: deploy
on:
push:
branches: [main]
permissions:
id-token: write
contents: read
jobs:
build-and-push:
```

```
runs-on: ubuntu-latest
steps:
- uses: actions/checkout@v4
- uses: aws-actions/configure-aws-credentials@v4
with:
  role-to-assume: arn:aws:iam::ACCT:role/eugene-github-deploy
  aws-region: eu-central-1
- uses: aws-actions/amazon-ecr-login@v2
- run: ./bin/docker/ecr/build.sh
- run: ./bin/docker/ecr/push.sh
- run: aws ecs update-service --cluster eugene-prod \
  --service eugene-ws --force-new-deployment
```

24. Frontend Sync on Every Release

```
# Append to the deploy workflow above
deploy-frontend:
  needs: build-and-push
  runs-on: ubuntu-latest
  steps:
  - uses: actions/checkout@v4
  - uses: actions/setup-node@v4
  with: { node-version: '20' }
  - run: cd agents/eugene-agent-ui-next && npm ci && npm run build
  - uses: aws-actions/configure-aws-credentials@v4
  with:
    role-to-assume: arn:aws:iam::ACCT:role/eugene-github-deploy
    aws-region: eu-central-1
  - run: aws s3 sync agents/eugene-agent-ui-next/out/ s3://eugene-frontend-prod/ --delete
  - run: aws cloudfront create-invalidation \
    --distribution-id E1ABC --paths '/*'
```

SECTION 8 — SMOKE TESTS & CUTOVER

25. End-to-End Smoke Test

```
# 1. Public frontend
curl -sI https://eugene.csl.example/ | head -3

# 2. ALB health
curl -sS https://api.eugene.csl.example/core/health
curl -sS https://api.eugene.csl.example/agent/api/health

# 3. Obtain JWT (Entra federated or local-bypass)
TOKEN=$(curl -sS https://api.eugene.csl.example/core/login | grep -oE
'eyJ[A-Za-z0-9_-]+\.[A-Za-z0-9_-]+\.[A-Za-z0-9_-]+' | head -1)

# 4. Hit a graph endpoint
curl -sS -H "Authorization: Bearer $TOKEN" \
https://api.eugene.csl.example/core/node/find/Hemophilia | head -200

# 5. Agent end-to-end
curl -sS -N -X POST https://api.eugene.csl.example/agent/api/query/stream \
-H "Authorization: Bearer $TOKEN" \
-H 'Content-Type: application/json' \
-d '{"prompt":"What is Andembry?","conversation_id":"00000000-0000-0000-0000-000000000001"}' |
head -10
```

26. Rollback Plan

- S3 versioning: `aws s3api list-object-versions` then `aws s3api copy-object` with `--version-id`.
- CloudFront cache flush: `aws cloudfront create-invalidation --paths '/*'`.
- ECS: each service tracks the previous task definition revision; `aws ecs update-service --task-definition`` rolls back instantly.
- Neo4j: take an automated snapshot of the EBS volume nightly; restore by creating a new EBS volume from the snapshot and remounting.

SECTION 9 — OPERATIONS

27. Cost Outlook (us-east-1, monthly)

Service	Sizing	Approx. USD / month
S3 + CloudFront	10 GB static + 200 GB egress	\$20
ALB	Always-on, 2 AZ	\$25
ECS Fargate (4 services × 2 tasks)	1 vCPU / 2 GB each	\$220
EC2 r6i.2xlarge (Neo4j)	1 instance, 24/7	\$390
EBS gp3 200 GB	Neo4j data	\$18
Secrets Manager	4 secrets	\$2
CloudWatch Logs	30-day retention, ~10 GB/mo	\$10
NAT Gateway	1 AZ	\$35
Route 53 hosted zone	1 zone	\$0.50
ACM	Public cert	Free
Total	**Single-AZ baseline**	**≈ \$720**

28. Monitoring + Alerts

- CloudWatch alarms: ALB 5xx > 1% over 5 minutes; ECS service CPU > 80%; Neo4j EC2 disk > 80%.
- Synthetics canary: every 5 minutes, GET `https://eugene.csl.example/` and hit `/core/health`.
- Log Insights queries (saved): error pattern by service, slow query > 2s, JWT validation failures.
- Service map via AWS X-Ray once OpenTelemetry is wired into `eugene-agent-ws` (see the Technical Recommendations doc).

29. Security Hardening Checklist

1. Enable AWS WAF on the CloudFront distribution with AWS Managed Rules: Core, Known Bad Inputs, SQL DB.
2. Turn on GuardDuty + Security Hub at the account level.
3. Force MFA on the AWS root account; create IAM users only via Identity Center.
4. Bedrock Guardrails on every LLM call (when migrating to Bedrock).
5. AWS Macie scan on S3 buckets (for sensitive-data classification).

6. Daily AWS Config rule scans against CIS benchmark.
7. AWS Backup with cross-region copy for Neo4j EBS volume and Secrets Manager.
8. Rotate KMS CMK every 90 days; rotate Secrets Manager values via Lambda hook.

APPENDICES

Appendix A — Terraform Module Layout

Recommended directory layout if you choose Terraform over manual console steps:

```
infrastructure/  
  envs/  
    prod/  
      backend.tf # remote-state S3 + DynamoDB lock  
      main.tf # composes all modules  
      terraform.tfvars # prod-specific values  
    staging/  
  modules/  
    vpc/  
    alb/  
    ecr/  
    ecs-service/  
    ec2-neo4j/  
    s3-cloudfront-spa/  
    secrets/  
    monitoring/
```

Appendix B — Troubleshooting

Symptom	Likely cause / fix
403 from CloudFront	OAC bucket policy not applied — copy from CloudFront console
S3 sync exits 'AccessDenied'	User missing s3:PutObject on the bucket
ECS task stuck in PROVISIONING	Subnet has no NAT / no route to ECR — fix VPC routing
ECS task exits with 'ResourceInitializationError'	Task role missing secretsmanager:GetSecretValue
ALB returns 504	Target health check failing; verify /health endpoint reachable on container port
Agent returns 'cannot connect to Neo4j'	Check Route 53 private record + eugene-neo4j-sg ingress from ECS sg
CloudFront SSL handshake fails	ACM cert is in wrong region (must be us-east-1) or domain mismatch
Frontend 404s on deep links	Add 403/404 → /index.html (200) custom error pages on CloudFront

Appendix C — Useful CLI One-Liners

```
# Force redeploy of an ECS service  
aws ecs update-service --cluster eugene-prod --service eugene-ws --force-new-deployment  
  
# Tail logs of a service  
aws logs tail /ecs/eugene-ws --follow --since 10m
```

```
# Invalidate CloudFront after a frontend deploy
aws cloudfront create-invalidation --distribution-id E1ABC --paths '/*'

# Get the public CloudFront domain
aws cloudfront list-distributions --query 'DistributionList.Items[?Aliases.Items &&
contains(Aliases.Items, `eugene.csl.example`)].DomainName' --output text

# Open Session Manager into the Neo4j box
aws ssm start-session --target i-0abc123

# Scale an ECS service
aws ecs update-service --cluster eugene-prod --service eugene-agent-ws --desired-count 4
```