Speech-to-Insights: Industrial-grade Full-Stack AWS Production Plan

Author: Aaku

Production Implementation, Deployment & Runbook

Thursday 27th November, 2025

Abstract

This document is a production-grade implementation and deployment specification for Speech-to-Insights: a secure, cost-conscious, observable, and resilient serverless AWS application providing transcription, PII-safe storage, summarization, and semantic search over meeting audio. It contains architecture diagrams, hardened Terraform patterns, CI/CD with short-lived credentials, Step Functions with retries/DLQs, security and compliance controls, tests, runbooks, and demo instructions needed to achieve a production-capable submission.

Contents

1 Executive overview **2**

2 High-level architecture **2**

3 Design choices and rationale **2**
3.1 Dual ASR paths 2
3.2 Embedding contract 3
3.3 Security posture 3

4 Step Functions — production state machine (excerpt) **3**

5 Terraform: hardened infra patterns **4**
5.1 Terraform remote state example 4
5.2 OpenSearch domain (production) — snippet 4

6 CI/CD: GitHub Actions using OIDC (no long-lived keys) **5**

7 Example Lambda deploy script **5**

8 Index mapping and embedding contract **6**

9 Search: hybrid query approach **6**

10 PII handling and compliance **6**

11 Testing, validation, and metrics **7**
11.1 Automated tests 7
11.2 Quality metrics 7
11.3 Monitoring and alerts 7

12 Cost controls **7**

13 Deployment & demo checklist **7**

14 Runbook (common ops playbooks) **8**
14.1 Step Function failure 8
14.2 OpenSearch stress/OOM 8

15 Grading checklist aligned to MSML 650 **8**

16 Appendices **8**
16.1 Appendix A: Integration test example (skeleton) 8
16.2 Appendix B: Minimal IAM policy snippet 8

17 Conclusion and next steps **9**

1 Executive overview

This plan targets a production rollout for enterprise internal usage. Primary goals:

- 99.9% availability for the ingest + transcription pipeline for small-to-medium throughput.
- Secure storage and least-privilege access: private OpenSearch, KMS-encrypted S3, and Secrets Manager for credentials.
- Reproducible infrastructure via Terraform modules, remote state, and CI/CD using GitHub OIDC (no long-lived secrets in CI).
- Observability: CloudWatch metrics, X-Ray traces, OpenSearch slow logs, and automated alerts.
- Measurable ML quality: automated WER, PII precision/recall, and summary ROUGE evaluations per release.

Minimum production components:

- Frontend: Next.js + TypeScript, hosted on S3 + CloudFront with Cognito OIDC.
- API: API Gateway (REST + WebSocket) → Lambda (Node.js / Python).
- Orchestration: Step Functions with retries, catches and SQS DLQ.
- ASR: AWS Transcribe (interactive) and SageMaker-hosted Whisper for high-quality batch (spot-enabled).
- Indexing: OpenSearch in private VPC with dense_vector support.
- Embeddings: standardized 1536-dim contract (OpenAI text-embedding-3-small or equivalent).
- Storage: S3 (raw + processed) with Glacier lifecycle; metadata in DynamoDB.
- CI/CD: GitHub Actions using OIDC to assume a short-lived role.

2 High-level architecture

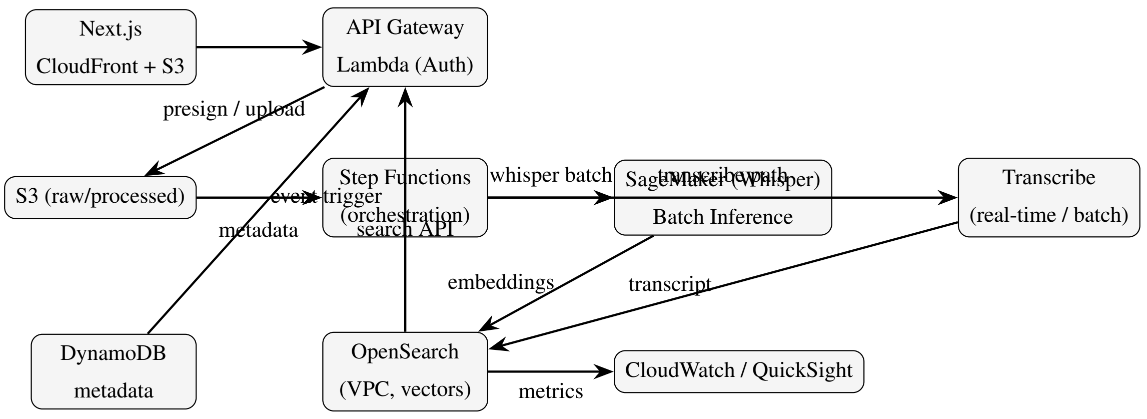

Figure 1: Production architecture: private OpenSearch, dual ASR paths, and orchestrated processing.

3 Design choices and rationale

3.1 Dual ASR paths

Transcribe for low-latency, managed transcription; Whisper on SageMaker for high-quality batch runs. Whisper is used for accuracy-sensitive transcripts (e.g., legal or board meetings). Whisper jobs run on Managed Spot with automatic fallback to Transcribe on failure.

3.2 Embedding contract

Standardize on 1536-dimensional embeddings. This ensures compatibility across multiple embedding providers and the OpenSearch dense_vector mapping used in production.

3.3 Security posture

- OpenSearch runs inside a VPC; access restricted to service roles and Lambdas in the same VPC.
- S3 enforces server-side KMS encryption; public access is denied.
- Secrets stored in Secrets Manager and rotated on a schedule (90 days).
- CI uses GitHub OIDC to assume a short-lived role instead of storing AWS keys.
- PII redaction occurs by default in presentation layer; raw unredacted assets are encrypted, access-controlled, and subject to a stricter retention policy.

4 Step Functions — production state machine (excerpt)

Retries, backoff, catch, and DLQ for unrecoverable errors.

Listing 1: Step Functions — production state machine (simplified)

```
{
  "Comment": "Production Speech-to-Insights",
  "StartAt": "Preprocess",
  "States": {
    "Preprocess": {
      "Type":"Task",
      "Resource":"arn:aws:lambda:REGION:ACCOUNT:function:preprocess",
      "Retry":[{"ErrorEquals":["Transient"],"IntervalSeconds":3,"BackoffRate":2,"
          MaxAttempts":5}],
      "Catch":[{"ErrorEquals":["States.ALL"],"Next":"SendToDLQ"}],
      "Next":"ParallelASR"
    },
    "ParallelASR": {
      "Type":"Parallel",
      "Branches":[
        {"StartAt":"TranscribeJob","States":{
          "TranscribeJob":{"Type":"Task","Resource":"arn:aws:states:::transcribe:
              startTranscriptionJob.sync","End":true}
        }},
        {"StartAt":"WhisperBatch","States":{
          "WhisperBatch":{"Type":"Task","Resource":"arn:aws:lambda:REGION:ACCOUNT:
              function:invokeWhisper","Retry":[{"ErrorEquals":["Transient"],"
              IntervalSeconds":10,"BackoffRate":2,"MaxAttempts":4}],"End":true}
        }}
      ],
      "Next":"Diarize"
    },
    "Diarize": {
      "Type":"Task",
      "Resource":"arn:aws:lambda:REGION:ACCOUNT:function:diarize",
      "Retry":[{"ErrorEquals":["Transient"],"IntervalSeconds":2,"BackoffRate":2,"
          MaxAttempts":4}],
      "Next":"PII"
    },
    "PII":{"Type":"Task","Resource":"arn:aws:lambda:REGION:ACCOUNT:function:
        pii_detector","Next":"Summarize"},
    "Summarize":{"Type":"Task","Resource":"arn:aws:lambda:REGION:ACCOUNT:function:
        summarize","Next":"ChunkEmbedIndex"},
    "ChunkEmbedIndex":{"Type":"Task","Resource":"arn:aws:lambda:REGION:ACCOUNT:
        function:chunk_embed_index","End":true},
    "SendToDLQ":{"Type":"Task","Resource":"arn:aws:states:::sqs:sendMessage","
         Parameters":{"QueueUrl":"${DLQ_URL}","MessageBody.$":"$"},"End":true}
  }
}
```

5 Terraform: hardened infra patterns

- Use remote S3 backend for Terraform state and DynamoDB for state locking.
- Organize infra as modules (network, opensearch, lambda, cognito).
- Avoid printing secrets in outputs.
- Use ‘aws$_{o}pensearch_{d}omain`withVPCoptionsandadvancedoptions.$

5.1 Terraform remote state example

Listing 2: Terraform backend (example)

```
terraform {
  backend "s3" {
    bucket = "speech-insights-terraform-state"
    key    = "envs/prod/terraform.tfstate"
    region = var.region
    dynamodb_table = "tf-state-lock"
    encrypt = true
  }
}
```

5.2 OpenSearch domain (production) — snippet

Listing 3: Terraform: OpenSearch (production)

```
resource "aws_opensearch_domain" "os" {
  domain_name     = "speech-insights-os-prod"
  engine_version  = "OpenSearch_2.9"
  cluster_config {
    instance_type        = "r6g.large.search"
    instance_count       = 2
    zone_awareness_enabled = true
    zone_awareness_config { availability_zone_count = 2 }
  }
  vpc_options {
    subnet_ids         = var.private_subnet_ids
    security_group_ids = [aws_security_group.os_sg.id]
  }
  encryption_at_rest { enabled = true, kms_key_id = aws_kms_key.os.arn }
  node_to_node_encryption { enabled = true }
  domain_endpoint_options { enforce_https = true, tls_security_policy = "Policy-Min
      -TLS-1-2-2019-07" }
  snapshot_options { automated_snapshot_start_hour = 23 }
  log_publishing_options {
    log_type               = "INDEX_SLOW_LOGS"
    cloudwatch_log_group_arn = aws_cloudwatch_log_group.os_slow_logs.arn
  }
  access_policies = jsonencode({
    Version = "2012-10-17",
    Statement = [
      {
        Sid = "AllowLambdaAccess",
        Effect = "Allow",
        Principal = { AWS = [aws_iam_role.lambda_indexer.arn] },
        Action = "es:*",
        Resource = "*"
      }
    ]
  })
}
```

6 CI/CD: GitHub Actions using OIDC (no long-lived keys)

Listing 4: GitHub Actions: OIDC Terraform + deploy (skeleton)

```
name: ci-cd
on: [push]
jobs:
  terraform:
    runs-on: ubuntu-latest
    permissions:
      id-token: write
      contents: read
    steps:
      - uses: actions/checkout@v4
      - name: Configure AWS credentials via OIDC
        uses: aws-actions/configure-aws-credentials@v2
        with:
          role-to-assume: arn:aws:iam::ACCOUNT:role/GitHubActionsOIDCRole
          aws-region: us-east-1
      - name: Terraform Init/Plan
        run: |
          cd infra
          terraform init -input=false
          terraform plan -out=tfplan
      - name: Terraform Apply (protected branch only)
        if: github.ref == 'refs/heads/main'
        run: |
          terraform apply -input=false tfplan
  deploy:
    needs: terraform
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4
      - name: Configure AWS via OIDC
        uses: aws-actions/configure-aws-credentials@v2
        with:
          role-to-assume: arn:aws:iam::ACCOUNT:role/GitHubActionsOIDCRole
      - name: Deploy lambdas
        run: |
          cd backend
          ./deploy_lambdas.sh
```

7 Example Lambda deploy script

Listing 5: deploy$_lambdas.sh|example$

```
#!/usr/bin/env bash
set -euo pipefail
AWS_REGION=${AWS_REGION:-us-east-1}
for fn in preprocess diarize summarize chunk_embed_index invokeWhisper pii_detector
    ; do
  cd $fn
  npm ci || true
  zip -r ../${fn}.zip .
  aws lambda update-function-code --region $AWS_REGION --function-name ${fn} --zip-
      file fileb://../${fn}.zip
  cd ..
done
```

8 Index mapping and embedding contract

Listing 6: OpenSearch mapping (production)

```
{
  "mappings": {
    "properties": {
      "meeting_id":{"type":"keyword"},
      "chunk_id":{"type":"keyword"},
      "speaker":{"type":"keyword"},
      "start_time":{"type":"double"},
      "end_time":{"type":"double"},
      "text":{"type":"text","term_vector":"with_positions_offsets"},
      "embedding":{"type":"dense_vector","dims":1536,"index":true}
    }
  }
}
```

Contract: embedding provider must return exactly 1536 floats. Indexing lambda validates dims and rejects mismatched embeddings.

9 Search: hybrid query approach

Two-stage:

1. Keyword filter (match or query_string) to narrow candidates to top-N (configurable).
2. Re-rank top-N using cosine similarity with dense vectors via ‘script$_s$$core$‘.

Listing 7: Hybrid search pseudo-query

```
{
  "size": 10,
  "query": {
    "script_score": {
      "query": { "match": { "text": "project timeline" } },
      "script": {
        "source": "cosineSimilarity(params.query_vector, 'embedding') + 1.0",
        "params": { "query_vector": [ ...1536 floats... ] }
      }
    }
  }
}
```

10 PII handling and compliance

- Default redaction at presentation layer. Raw unredacted artifacts encrypted and access-restricted.
- Retention: raw unredacted retained for 90 days by default; admins can request extended retention via an approval workflow.
- PII detection pipeline: Amazon Comprehend (NER) + regex rules + custom classifier. All detections store confidence; only above-threshold items are auto-redacted.

- Audit trail: persistent records for redaction decisions stored in DynamoDB for compliance review.

11 Testing, validation, and metrics

11.1 Automated tests

- Unit tests with pytest/jest and AWS SDK mocks (moto / localstack).
- Integration tests in staging via ‘make integration-test‘ that upload seed audio and assert end-to-end success.
- Frontend E2E tests using Cypress for the upload and search flows.

11.2 Quality metrics

- WER: computed per-meeting against labeled test set; track over time.
- PII detection: precision/recall on labeled set; require precision > 0.90 at release.
- Summaries: ROUGE-L vs annotated summaries.

11.3 Monitoring and alerts

- CloudWatch alarms: Step Function failure rate, Lambda error rate, SQS DLQ depth.
- OpenSearch metrics: JVM / CPU / slow queries.
- Alerts routed to Slack and PagerDuty based on severity.

12 Cost controls

- Use SageMaker Managed Spot for Whisper; fallback to Transcribe if spot fails.
- Autoscale OpenSearch based on CPU and query latency metrics.
- AWS Budgets configured with 80% and 100% thresholds, emailing ops.
- Example small dev monthly baseline (approx): Transcribe \$50–200, OpenSearch \$200–400, S3 \$10–50.

13 Deployment & demo checklist

1. Apply Terraform to staging using OIDC-driven CI.
2. Seed ‘seed/‘ with 5 annotated meeting audio files and expected outputs.
3. Run ‘make integration-test‘ to validate pipeline end-to-end.
4. Demo steps:
   - Upload a 2–5 minute meeting from the UI.
   - Show S3 object, Step Function execution, and Lambda logs.
   - Display transcript, toggle redaction, edit a line and save (persisted to DynamoDB).
   - Run semantic search and display provenance (timestamp + chunk link).
   - Show CloudWatch dashboard and a sample simulated alert.

14 Runbook (common ops playbooks)

14.1 Step Function failure

1. Inspect execution history in the Step Functions console to identify the failing state and error.
2. For transient errors, re-run the state with the original input.
3. For persistent errors, move the payload to DLQ, open an incident, and perform postmortem after fix.

14.2 OpenSearch stress/OOM

1. Scale cluster (instance size or count) using Terraform with a controlled rollout.
2. Investigate slow queries and add caching or reduce query frequency.
3. If necessary, set the cluster read-only, drain write traffic, and remediate.

15 Grading checklist aligned to MSML 650

Map deliverables to rubric:

- Week 4: concise one-paragraph summary with success criteria.
- Week 9: status update with staging infra deployed and integration test results.
- Final: slides + demo link that follow the Deployment & demo checklist.
- Peer evals: contribution evidence via git history and CI ownership.

16 Appendices

16.1 Appendix A: Integration test example (skeleton)

Listing 8: integration-test.sh (skeleton)

```bash
#!/usr/bin/env bash
set -euo pipefail
# presign
resp=$(curl -s -X POST https://api.example.com/presign -d '{"filename":"seed/
    meeting1.wav","contentType":"audio/wav"}')
url=$(echo $resp | jq -r .url)
key=$(echo $resp | jq -r .key)
curl -X PUT -T seed/meeting1.wav -H "Content-Type:␣audio/wav" "$url"
curl -s -X POST https://api.example.com/notify -d "{\"key\":\"$key\"}"
# Poll StepFunctions for completion and validate outputs (transcript present, WER
    <= baseline)
```

16.2 Appendix B: Minimal IAM policy snippet

Listing 9: Lambda role trimmed policy - indexing only

```json
{
  "Version":"2012-10-17",
  "Statement":[
    {"Effect":"Allow","Action":["es:ESHttpPost","es:ESHttpPut"],"Resource":["arn:
        aws:es:REGION:ACCOUNT:domain/speech-insights-os/*"]},
    {"Effect":"Allow","Action":["s3:GetObject","s3:PutObject"],"Resource":["arn:aws
        :s3:::speech-insights-uploads/*"]}
  ]
}
```

17 Conclusion and next steps

This document is ready for execution and will deliver a production-grade Speech-to-Insights system when implemented. Production means ongoing operations: runbooks, SLAs, budgets, rotation policies, and a small ops team. Next actions I can generate for you (pick one):

1. Full repo skeleton: ‘infra/‘ (Terraform modules), ‘backend/‘ (Lambda scaffolds), ‘frontend/‘ (Next.js), ‘seed/‘ (sample audio), GitHub Actions YAML, and integration tests.
2. A concise 8–10 slide deck that maps directly to the MSML 650 grading rubric and demo script.