AWS Spotify - Security Architecture & Infrastructure Runbook

Microscaled Spotify Architecture on AWS | IAM, Terraform, and Security Controls

Nathan Lim

March 2026 - v3

Contents

Architecture Overview

Figure 1: Architecture Overview

1 Architecture Overview

1.1 System Context

A microscaled implementation of Spotify's backend architecture deployed on AWS. The system serves audio content to 50 simulated users across 28 tracks, with the primary design objective being security control depth rather than scale.

Parameter	Value
Region	us-east-1
Budget	$10/month
IaC	Terraform 1.14.7 (AWS Provider 5.100.0)
Compute	EC2 t3.micro
Database	PostgreSQL 16 (self-hosted)
Auth	Cognito + JWT hybrid

1.2 Architectural Decision Records

1.2.1 ADR-001: EC2 over Lambda

Context: The system requires a persistent API server with VPC-resident networking, security group enforcement, and SSH-accessible OS hardening. Lambda abstracts away the network layer, eliminating the ability to demonstrate security group configuration, NACL design, and OS-level controls.

Decision: Single EC2 t3.micro instance in a custom VPC public subnet.

Consequence: Richer security surface for demonstrating IAM instance profiles, security group rules, encrypted EBS, and systemd service management. Tradeoff is no auto-scaling - acceptable at 50-user microscale.

1.2.2 ADR-002: Self-Hosted PostgreSQL over RDS

Context: RDS db.t3.micro costs $12.41/month post-free-tier, exceeding the $10 total budget before any other service. Self-hosting on the EC2 instance costs $0 incremental.

Decision: PostgreSQL 16 installed on the EC2 instance with `pg_hba.conf` configured for local TCP/md5 authentication. Database credentials stored in application config (Secrets Manager integration planned).

Consequence: Demonstrates understanding of the operational tradeoffs: no automated backups (mitigated by reproducible seed scripts), no Multi-AZ failover (accepted for portfolio scope), no managed patching (mitigated by Amazon Linux 2023 auto-update). The RDS migration path is documented in the Terraform modules - the `terraform/environments/prod/` directory is reserved for this upgrade.

1.2.3 ADR-003: Cognito + Custom JWT Hybrid Authentication

Context: Enterprise identity architectures separate human identity (managed IdP) from machine identity (service tokens). Using only Cognito or only custom JWT would demonstrate half the pattern.

Decision: AWS Cognito User Pool handles user authentication (sign-up, sign-in, MFA, token lifecycle). The API server validates Cognito-issued JWTs against the JWKS endpoint for request authorization. A separate API app client with a secret handles server-side operations.

Consequence: Two app clients with different security postures: the frontend client is public (no secret, SRP auth flow), the API client is confidential (with secret, admin auth flow). Token validity is set to 1 hour for access/ID tokens and 30 days for refresh tokens. `prevent_user_existence_errors` is enabled on both clients to block email enumeration attacks.

1.2.4 ADR-004: CloudFront Dual-Origin with Path-Based Routing

Context: The frontend (React SPA) and API server have different caching, security, and origin requirements. A single origin cannot serve both efficiently.

Decision: CloudFront distribution with two origins: S3 via Origin Access Control (OAC) for static assets (default behavior, 24h TTL), and EC2 via custom origin for `/api/*` requests (0s TTL, Authorization header forwarded).

Consequence: HTTPS enforced at the viewer edge. S3 bucket remains fully private - no public access, no legacy OAI. API requests pass through with zero caching, preserving auth header integrity. SPA routing handled via custom error responses (403/404 rewrite to `index.html`).

IAM Trust Chain

Figure 2: IAM Trust Chain

2 IAM Architecture

2.1 Policy Evaluation Logic

IAM policy evaluation follows a deterministic three-tier model. Every access decision resolves through this chain:

1. **Explicit Deny** - Any policy containing a Deny statement for the requested action terminates evaluation immediately. No Allow can override it. This is the foundation of defense-in-depth in IAM.
2. **Explicit Allow** - If no Deny exists and at least one policy grants Allow for the action, access is permitted.
3. **Implicit Deny** - If no policy mentions the action at all, access is denied by default. AWS is deny-by-default.

Critical interaction with Permission Boundaries:

When a permission boundary is attached to an IAM entity, the effective permissions become the *intersection* of the identity policy and the boundary:

```
Effective Permissions =
  (Identity Policy ALLOWS)
  AND (Boundary ALLOWS)
  - (Any Explicit DENY)
```

This means attaching `AdministratorAccess` to an entity with a boundary that only allows S3 and EC2 results in access to S3 and EC2 only. The boundary is a ceiling, not a grant.

2.2 Role Architecture

2.2.1 EC2 Instance Role - Trust and Permission Chain

The EC2 instance assumes `SpotifyEC2Role` via the instance metadata service (IMDSv2). Credentials rotate automatically every ~6 hours. No static access keys exist on the instance.

Trust Policy:

```
{
  "Version": "2012-10-17",
  "Statement": [{
    "Effect": "Allow",
    "Principal": { "Service": "ec2.amazonaws.com" },
    "Action": "sts:AssumeRole"
  }]
}
```

The `Principal` field restricts assumption to the EC2 service only. Lambda, ECS, or human users cannot assume this role.

Permissions Policy - Scoped by Resource ARN:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "S3AudioBucketAccess",
      "Effect": "Allow",
      "Action": [
        "s3:GetObject",
        "s3:PutObject",
        "s3:DeleteObject",
        "s3:ListBucket"
      ],
      "Resource": [
        "arn:aws:s3:::spotify-audio-ACCT_ID",
        "arn:aws:s3:::spotify-audio-ACCT_ID/*"
      ]
    },
    {
      "Sid": "SecretsManagerDBCredentials",
      "Effect": "Allow",
      "Action": [
        "secretsmanager:GetSecretValue"
      ],
      "Resource": "arn:aws:secretsmanager:
        us-east-1:ACCT_ID:secret:spotify/db-*"
    },
    {
      "Sid": "CloudWatchLogging",
      "Effect": "Allow",
      "Action": [
        "logs:CreateLogGroup",
        "logs:CreateLogStream",
        "logs:PutLogEvents",
        "logs:DescribeLogStreams"
      ],
      "Resource": "arn:aws:logs:
        us-east-1:ACCT_ID:log-group:/spotify/*"
    },
    {
      "Sid": "CloudWatchMetrics",
      "Effect": "Allow",
      "Action": ["cloudwatch:PutMetricData"],
      "Resource": "*",
      "Condition": {
```

```
        "StringEquals": {
          "cloudwatch:namespace": "Spotify/API"
        }
      }
    }
  ]
}
```

Design notes on this policy:

- Two S3 ARNs are required: bucket-level (`arn:aws:s3:::bucket`) for `ListBucket`, object-level (`arn:aws:s3:::bucket/*`) for `GetObject/PutObject/DeleteObject`. Omitting either causes AccessDenied on the corresponding operation class.
- The Secrets Manager ARN uses a wildcard suffix (`-*`) because Secrets Manager appends a random 6-character suffix to secret ARNs at creation time.
- CloudWatch Metrics uses a namespace condition (`cloudwatch:namespace: "Spotify/API"`) to prevent the instance from writing metrics to other namespaces - a subtle least-privilege constraint that most implementations skip.
- `CreateLogGroup` is included for first-deployment convenience. In a production hardening pass, the log group would be pre-created via Terraform and this permission removed.

2.2.2 Permission Boundary

The boundary caps the maximum permissions any role in the project can have, regardless of attached policies:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "AllowedServices",
      "Effect": "Allow",
      "Action": [
        "s3:*", "ec2:*",
        "cognito-idp:*", "cloudfront:*",
        "cloudwatch:*", "logs:*",
        "secretsmanager:GetSecretValue",
        "kms:Decrypt",
        "kms:GenerateDataKey"
      ],
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "aws:RequestedRegion": "us-east-1"
        }
      }
    },
    {
```

```
      "Sid": "DenyDangerousActions",
      "Effect": "Deny",
      "Action": [
        "organizations:*",
        "account:*",
        "iam:CreateUser",
        "iam:DeleteUser",
        "iam:CreateAccountAlias",
        "iam:DeleteAccountAlias"
      ],
      "Resource": "*"
    }
  ]
}
```

The region lock (`aws:RequestedRegion: us-east-1`) prevents resource creation in other regions. The explicit Deny on Organizations and Account actions is a hard stop - even `AdministratorAccess` cannot override it.

2.2.3 Security Auditor Role - Read-Only with Explicit Deny Guardrail

The auditor role demonstrates a pattern common in enterprise SOCs: broad read access to security services with an explicit deny on all mutation actions.

```
{
  "Sid": "DenyMutations",
  "Effect": "Deny",
  "Action": [
    "iam:Create*", "iam:Delete*",
    "iam:Update*", "iam:Put*",
    "iam:Attach*", "iam:Detach*",
    "iam:Add*",    "iam:Remove*"
  ],
  "Resource": "*"
}
```

This Deny statement is the safety net. Even if `AdministratorAccess` is accidentally attached to the auditor group, IAM mutation actions remain blocked. The explicit Deny always wins per the evaluation logic above.

2.3 IAM Group Structure

Five groups model enterprise job functions:

Group	Policy	Access
Admins	AdministratorAccess	Full (break-glass)
Developers	EC2 Describe, S3, CW, Cognito	Build/debug
DevOps	EC2/S3/CF full, no IAM	Deploy/operate

Group	Policy	Access
SecurityAuditors	IAM/CT/Config/GD + deny mutate	Audit only
ReadOnly	AWS ReadOnlyAccess	Observe only

3 Network Security

3.1 VPC Design

```
VPC: 10.0.0.0/16
|---- Public Subnet: 10.0.1.0/24 (us-east-1a)
|     \---- EC2 t3.micro + Elastic IP
\---- (Reserved) Private Subnet: 10.0.2.0/24 (us-east-1b)
    \---- Future: RDS PostgreSQL
Internet Gateway -> Route Table: 0.0.0.0/0 -> IGW
```

Segmentation rationale: The public subnet hosts the API server which requires inbound internet access. The reserved private subnet demonstrates awareness of the production pattern (data stores isolated from direct internet access). A NAT Gateway ($32.40/month) is omitted due to budget - documented as a known gap.

3.2 Security Group Rules

Dir	Proto	Port	Source/Dest	Purpose
In	TCP	22	Admin IP/32	SSH - single IP only
In	TCP	3000	0.0.0.0/0	API server (CF origin)
In	TCP	443	0.0.0.0/0	HTTPS
In	TCP	80	0.0.0.0/0	HTTP (ACME cert)
In	TCP	5432	10.0.0.0/16	PostgreSQL - VPC only
Out	All	All	0.0.0.0/0	AWS services + updates

Port 22 restricted to /32 is non-negotiable. Automated scanners hit open SSH ports within minutes of instance launch. Port 5432 scoped to VPC CIDR means the database is unreachable from the internet regardless of PostgreSQL authentication configuration.

3.3 S3 Presigned URL Security Model

The audio streaming workflow uses S3 presigned URLs to provide time-limited, per-object access without exposing the S3 bucket or requiring client-side AWS credentials.

How it works:

1. Client requests `GET /api/songs/{id}` with Cognito JWT in Authorization header
2. API server validates JWT against Cognito JWKS endpoint
3. API generates a presigned URL for the audio file's S3 key using the EC2 instance role credentials
4. Presigned URL expires in 15 minutes
5. Client streams audio directly from S3 using the presigned URL - no API server bandwidth consumed

Security properties:

- The presigned URL inherits the permissions of the signer (the EC2 instance role). If the role loses `s3:GetObject` permission, all outstanding presigned URLs immediately fail with 403.

- Each URL is bound to a specific object key. A presigned URL for `artist/1/album/1/track.ogg` cannot be used to access `artist/2/album/1/track.ogg`.
- The 15-minute expiry window limits the blast radius of URL leakage. A leaked URL is useless after expiry.
- The S3 bucket has Block Public Access enabled on all four settings. The presigned URL is the *only* access path for audio content.

Terraform (S3 module - bucket policy enforcing encryption and HTTPS):

```
resource "aws_s3_bucket_policy" "audio" {
  bucket = aws_s3_bucket.audio.id
  policy = jsonencode({
    Version = "2012-10-17"
    Statement = [
      {
        Sid       = "DenyUnencryptedUploads"
        Effect    = "Deny"
        Principal = "*"
        Action    = "s3:PutObject"
        Resource  = "${aws_s3_bucket.audio.arn}/*"
        Condition = {
          StringNotEquals = {
            "s3:x-amz-server-side-encryption" = "AES256"
          }
        }
      },
      {
        Sid       = "DenyInsecureTransport"
        Effect    = "Deny"
        Principal = "*"
        Action    = "s3:*"
        Resource  = [
          aws_s3_bucket.audio.arn,
          "${aws_s3_bucket.audio.arn}/*"
        ]
        Condition = {
          Bool = {
            "aws:SecureTransport" = "false"
          }
        }
      }
    ]
  })
}
```

4 Encryption Strategy

Data State	Mechanism	Key Mgmt	Cost
Audio at rest	SSE-S3 (AES-256)	AWS-managed	$0
EBS at rest	AES-256 via EBS	AWS default key	$0
Client to CF	TLS 1.2+	AWS cert	$0
CF to EC2	HTTP (origin)	N/A	$0
DB credentials	Secrets Manager	AWS-managed	$0.40/mo

SSE-S3 vs SSE-KMS tradeoff: SSE-KMS provides customer-managed keys with CloudTrail audit trails for key usage, but costs $1/key/month plus API call charges. At this budget, SSE-S3 provides equivalent AES-256 encryption at zero cost. The upgrade path to KMS is documented for production environments where key rotation audit trails are compliance-required.

Security Controls

Figure 3: Security Controls

5 Detective and Monitoring Controls

5.1 CloudTrail

Multi-region trail with log file validation. Captures every API call across all regions - an attacker creating resources in `ap-southeast-1` while you monitor `us-east-1` is still logged. Log file validation creates a hash chain in S3 that detects tampering.

5.2 GuardDuty

Enabled with 15-minute finding publication. Uses ML to detect anomalous behavior: unusual API calls, cryptocurrency mining, C2 communication. At this scale, effectively $0/month.

5.3 AWS Config

Four compliance rules continuously evaluating resource state:

Rule	Checks
s3-bucket-sse-enabled	S3 buckets have encryption
ec2-instances-in-vpc	EC2 not outside VPC
root-account-mfa-enabled	Root has MFA
iam-root-access-key-check	No root access keys

5.4 IAM Access Analyzer

Continuously scans resource policies for external access grants. Flags S3 buckets, IAM roles, KMS keys, Lambda functions, and SQS queues that are accessible from outside the account.

5.5 Budget and Cost Alarms

Alarm	Threshold	Action
Budget 80%	$8.00	Email
Budget 100%	$10.00	Email
Budget 150%	$15.00	Email
CW billing	$10.00	SNS email
CW billing	$20.00	SNS email (emergency)
CW CPU	80% avg (2x)	SNS email

6 Terraform Infrastructure

6.1 Module Architecture

All infrastructure is defined as reusable Terraform modules:

Module	Provisions	Outputs
`vpc`	VPC, IGW, subnet, RT, SG (6 rules)	vpc_id, sg_id
`ec2`	t3.micro, encrypted gp3, EIP	instance_id, ip
`s3`	Audio + Frontend buckets	bucket names
`cloudfront`	Dual-origin, OAC, SPA routing	domain_name
`cognito`	User pool, MFA, 2 clients	pool_id, endpoint
`monitoring`	SNS, Budget, 3 CW alarms	sns_topic_arn

Total Terraform-managed resources: 37

6.2 State Management

State is stored locally (`terraform.tfstate`). For a team environment, this would migrate to an S3 backend with DynamoDB state locking:

```
backend "s3" {
  bucket         = "spotify-terraform-state"
  key            = "dev/terraform.tfstate"
  region         = "us-east-1"
  dynamodb_table = "terraform-locks"
  encrypt        = true
}
```

The state file is excluded from version control via `.gitignore`. It contains resource IDs, IP addresses, and potentially sensitive configuration values.

6.3 Provider Configuration

```
terraform {
  required_version = ">= 1.14.0"
  required_providers {
    aws = {
      source  = "hashicorp/aws"
      version = "~> 5.0"
    }
  }
}

provider "aws" {
  region  = "us-east-1"
  profile = "spotify-admin"
  default_tags {
```

```
    tags = {
      Project     = "spotify"
      Environment = "dev"
      ManagedBy   = "terraform"
    }
  }
}
```

`default_tags` ensures every resource is tagged for cost allocation and ownership identification without per-resource tag blocks.

7 Incident Response - Simulated Postmortem

7.1 INC-001: S3 Block Public Access Removed by Terraform

Severity: P1 | **Duration:** 2h 15m | **Data exposed:** None

What happened: During a Terraform update to add CORS configuration, the `aws_s3_bucket_public_access_block` resource was accidentally deleted from the `.tf` file. `terraform apply` destroyed the real resource to match desired state.

Why impact was limited (defense-in-depth):

- Bucket policy `DenyInsecureTransport` remained in place
- No public ACLs existed - removing Block Public Access removes the guardrail, not the lock
- Presigned URLs still required for object access
- GuardDuty detected anomalous S3 access patterns within 15 minutes

Remediation: `lifecycle { prevent_destroy = true }` added to all security-critical resources. AWS Config rule `s3-bucket-level-public-access-prohibited` added for continuous monitoring.

Root cause: Process failure - no peer review of `terraform plan` output. The plan clearly showed a destroy action on a security resource. Mitigation: CI/CD pipeline with plan review gates.

8 Compliance Mapping

8.1 NIST 800-53 Controls

Control	Name	Implementation
AC-2	Account Mgmt	IAM groups, no shared accounts
AC-3	Access Enforcement	Resource-level ARN scoping
AC-6	Least Privilege	No * on sensitive actions
AC-6(1)	Security Function Auth	Permission boundaries
AC-17	Remote Access	SSH to single IP, ed25519
AU-2	Audit Events	CloudTrail multi-region
AU-9	Audit Protection	Log validation, S3 versioning
CA-7	Continuous Monitoring	Config, GuardDuty, CW alarms
CM-2	Baseline Config	Terraform, drift via plan
IA-2(1)	MFA Privileged	MFA on root + admin user
SC-7	Boundary Protection	VPC, SGs, subnet segmentation
SC-8	Transit Confidentiality	TLS 1.2+ on CloudFront
SC-28	At-Rest Protection	SSE-S3, encrypted EBS
SI-4	System Monitoring	CW metrics, GuardDuty

8.2 CIS AWS Foundations Benchmark v3.0

Control	Status
1.4 No root access key	Compliant
1.5 MFA on root	Compliant
2.1.1 CloudTrail all regions	Compliant
2.1.2 Log file validation	Compliant
2.2.1 GuardDuty enabled	Compliant
3.1 S3 Block Public Access	Compliant
3.3 S3 encryption enabled	Compliant
5.1 No SSH from 0.0.0.0/0	Compliant (admin IP only)

9 Cost Architecture

Service	Resource	Monthly Cost
EC2	t3.micro on-demand (730h)	$7.59
EBS	20 GB gp3	$1.60
S3	300 MB Standard + requests	$0.06
CloudFront	67.5 GB (1TB free tier)	$0.00
Cognito	50 MAU (50K free tier)	$0.00
CloudWatch	Basic metrics + alarms	$0.00
EIP	Attached to running instance	$0.00
Total		**~$9.25**

Optimization levers: Stop EC2 when not in use ($0.25/day saved). Reduce EBS to 10GB (-$0.80/mo). The Elastic IP remains free only while attached to a running instance - releasing it when the instance is stopped avoids the $0.005/hour unattached charge.

10 Appendix: Database Schema

Six tables matching Spotify's relational metadata model:

Table	Key Columns	Relations
users	user_id PK, email CITEXT UNIQUE	playlists FK
artists	artist_id PK, name, country	songs, artistsongs FK
songs	song_id PK, title, file_url, artist_id FK	playlistitems FK
playlists	playlist_id PK, owner_id FK	playlistitems FK
playlistitems	playlist_id + song_id (composite PK)	Ordered tracks
artistsongs	song_id + artist_id (composite PK)	Many-to-many

CITEXT on email prevents case-variant duplicate registrations (`User@Example.com` = `user@example.com`). Indexes on `songs(artist_id)`, `songs(title)`, `playlists(owner_id)`, `playlistitems(song_id)` support the primary query patterns.

11 Appendix: REST API Surface

```
GET  /api/health                Health check
GET  /api/songs?limit=N        Paginated listing
GET  /api/songs/:id            Metadata + presigned URL
GET  /api/artists/:id/songs    Songs by artist
GET  /api/playlists/:id        Playlist with tracks
GET  /api/search?q=term        Full-text search
POST /api/playlists            Create playlist (auth)
POST /api/songs/:id/like       Like song (auth)
```

12 Appendix: Terraform Quick Reference

```
cd terraform/environments/dev
terraform init              # Initialize
terraform plan -out=tfplan  # Dry run
terraform apply "tfplan"    # Execute plan
terraform state list        # Resource inventory
terraform output            # Output values
```