# Digital Identity API on AWS (Fargate + Aurora MySQL + API Gateway + SNS/SQS)

Region: **af-south-1 (Cape Town)**  
AZs: **af-south-1a, af-south-1b**

This document gives you: - A step‑by‑step AWS configuration guide. - Production‑grade Terraform (modular) to deploy the VPC, networking, RDS Aurora MySQL, SNS+SQS, ECS on Fargate (API + Worker), internal ALB, API Gateway (HTTP API) with VPC Link, and supporting IAM/Secrets/Logs.  
- Minimal .NET 8 code skeletons (API & Worker) + Dockerfiles so you can build/push/run.

**Architecture Note**: Because API Gateway is the internet‑facing front door, the ALB is configured as **internal** and placed in private application subnets. API Gateway uses **VPC Link** to privately reach the ALB, which targets the ECS Fargate API service. The API writes PENDING identity records to Aurora and publishes messages to **SNS**. SNS fans out to **SQS** (with DLQ). A separate **Worker** Fargate service consumes SQS, calls the third‑party API, and updates the record status + response in Aurora.

## 1) High‑Level Architecture

Client → API Gateway (HTTP API)  
 │  
 └── VPC Link (ENIs in private subnets)  
 │  
 └── Internal ALB (private)  
 │  
 └── Target Group → ECS Fargate Service: api  
  
Aurora MySQL (Multi‑AZ) ←───────────── api writes PENDING + publishes SNS  
 └─> SNS Topic ──> SQS Queue (DLQ)  
 │  
 └── ECS Fargate Service: worker  
 ├─ Calls 3rd‑party API (via NAT)  
 └─ Updates Aurora MySQL row  
  
Internet Gateway (for public egress) + NAT Gateways (one per AZ for HA)  
VPC Endpoints (Interface) for SNS/SQS to reduce NAT traffic

## 2) Step‑by‑Step AWS Configuration Guide

The Terraform below automates everything, but here are the manual steps conceptually in the order Terraform will execute.

### Networking & Security

1. **Create VPC (10.0.0.0/16)** in **af-south-1** with **2 AZs**.
2. **Create Subnets**:
   * **Public** (10.0.0.0/24, 10.0.1.0/24) – for IGW, NAT GW.
   * **Private-App** (10.0.10.0/24, 10.0.11.0/24) – for ALB + ECS tasks.
   * **Private-DB** (10.0.20.0/24, 10.0.21.0/24) – for Aurora.
3. **Attach Internet Gateway** to VPC.
4. **Create NAT Gateways (x2)** in each public subnet; allocate **EIPs**.
5. **Route Tables**:
   * Public RT: 0.0.0.0/0 → IGW (assoc. to public subnets)
   * Private-App RTs (one per AZ): 0.0.0.0/0 → NAT GW (same AZ)
   * Private-DB RTs: **no internet route** (only internal VPC traffic)
6. **Security Groups**:
   * **sg\_alb**: allow inbound **HTTP 80** from **sg\_vpclink** (API GW ENIs), allow egress to **sg\_ecs\_api**.
   * **sg\_ecs\_api**: allow inbound **8080** from **sg\_alb**; egress to DB (3306), SNS/SQS (via VPC endpoints/NAT).
   * **sg\_ecs\_worker**: no inbound; egress to DB (3306), 3rd‑party API via NAT, and SNS/SQS.
   * **sg\_db**: allow inbound **3306** from **sg\_ecs\_api** and **sg\_ecs\_worker**.
   * **sg\_vpclink**: used by API GW VPC Link ENIs; outbound to **sg\_alb**.
7. **(Optional but Recommended) VPC Interface Endpoints** for **SNS** and **SQS** to keep that traffic private and reduce NAT egress.

### Data Layer (Aurora MySQL)

1. **Create DB Subnet Group** over the two Private‑DB subnets.
2. **Create an Aurora MySQL Cluster** (Multi‑AZ), define engine version, admin creds, storage, backup.
3. **Store DB creds** in **AWS Secrets Manager** for ECS tasks to read (also supplied to RDS at create time).

### Messaging (SNS + SQS)

1. **Create SNS Topic** identity-topic.
2. **Create SQS Queue** identity-queue and **DLQ** identity-dlq with redrive policy (e.g., maxReceiveCount=5).
3. **Subscribe SQS to SNS** (queue policy must allow sns:Publish from the topic ARN).

### Containers & Compute (ECR, ECS on Fargate)

1. **Create ECR repositories**: identity-api and identity-worker.
2. **Build & push images** (see commands below).
3. **Create ECS Cluster**.
4. **Create IAM Roles**:
   * **ecsTaskExecutionRole** (pull from ECR, write logs, decrypt if needed).
   * **ecsTaskRole\_api** (publish to SNS, read Secrets Manager, DB access via network).
   * **ecsTaskRole\_worker** (read from SQS, delete/visibility‑change messages, read Secrets Manager, DB via network, call 3rd‑party API).
5. **CloudWatch Log Groups** for both tasks.
6. **Task Definitions** (Fargate): API (containerPort 8080), Worker (no listener).

### Load Balancing & API Front Door

1. **Create Internal ALB** (scheme: internal) in **Private-App** subnets; target group on port 8080, health check /health.
2. **ALB Listener (HTTP 80)** → forward to API target group.
3. **API Gateway (HTTP API)** with **VPC Link**:
   * **VPC Link**: attach to Private-App subnets with **sg\_vpclink**.
   * **Integrations**: HTTP proxy to **ALB Listener ARN**.
   * **Routes**: POST /identities, POST /identities/batch, GET /identities/{id} → single integration.
   * **Stage**: $default with auto deploy; enable access logs.

### Scale, Observability, and Ops

1. **ECS Service Autoscaling** (target tracking on CPU or SQS queue depth for worker).
2. **CloudWatch Alarms** on DLQ, 5xx at API GW/ALB, DB CPU/storage, etc.
3. **Parameterize** 3rd‑party API base URL and credentials via Secrets/SSM.

## 3) Terraform — File Layout

terraform/  
 providers.tf  
 variables.tf  
 locals.tf  
 vpc.tf  
 security\_groups.tf  
 endpoints.tf  
 rds.tf  
 secrets.tf  
 sns\_sqs.tf  
 ecr.tf  
 iam.tf  
 ecs\_cluster.tf  
 alb.tf  
 apigw\_vpclink.tf  
 ecs\_task\_api.tf  
 ecs\_task\_worker.tf  
 outputs.tf

**Apply order** is handled by Terraform’s graph. Use terraform init, terraform plan, terraform apply.

### providers.tf

terraform {  
 required\_version = ">= 1.6.0"  
 required\_providers {  
 aws = {  
 source = "hashicorp/aws"  
 version = ">= 5.50"  
 }  
 }  
}  
  
provider "aws" {  
 region = var.region  
}

### variables.tf

variable "project\_name" { type = string }  
variable "region" { type = string default = "af-south-1" }  
variable "azs" {  
 type = list(string)  
 default = ["af-south-1a", "af-south-1b"]  
}  
variable "vpc\_cidr" { type = string default = "10.0.0.0/16" }  
variable "public\_subnets" { type = list(string) default = ["10.0.0.0/24", "10.0.1.0/24"] }  
variable "private\_app\_subnets" { type = list(string) default = ["10.0.10.0/24", "10.0.11.0/24"] }  
variable "private\_db\_subnets" { type = list(string) default = ["10.0.20.0/24", "10.0.21.0/24"] }  
  
variable "db\_name" { type = string default = "identitydb" }  
variable "db\_username" { type = string default = "identity\_admin" }  
variable "db\_password" { type = string sensitive = true }  
  
variable "container\_port" { type = number default = 8080 }  
variable "desired\_count\_api" { type = number default = 2 }  
variable "desired\_count\_worker" { type = number default = 2 }  
  
variable "third\_party\_api\_base\_url" { type = string }

### locals.tf

locals {  
 tags = {  
 Project = var.project\_name  
 Env = "prod"  
 }  
}

### vpc.tf

resource "aws\_vpc" "this" {  
 cidr\_block = var.vpc\_cidr  
 enable\_dns\_hostnames = true  
 enable\_dns\_support = true  
 tags = merge(local.tags, { Name = "${var.project\_name}-vpc" })  
}  
  
resource "aws\_internet\_gateway" "igw" {  
 vpc\_id = aws\_vpc.this.id  
 tags = merge(local.tags, { Name = "${var.project\_name}-igw" })  
}  
  
resource "aws\_subnet" "public" {  
 for\_each = { for idx, cidr in var.public\_subnets : idx => cidr }  
 vpc\_id = aws\_vpc.this.id  
 cidr\_block = each.value  
 availability\_zone = var.azs[tonumber(each.key)]  
 map\_public\_ip\_on\_launch = true  
 tags = merge(local.tags, {  
 Name = "${var.project\_name}-public-${each.key}",  
 Tier = "public"  
 })  
}  
  
resource "aws\_subnet" "private\_app" {  
 for\_each = { for idx, cidr in var.private\_app\_subnets : idx => cidr }  
 vpc\_id = aws\_vpc.this.id  
 cidr\_block = each.value  
 availability\_zone = var.azs[tonumber(each.key)]  
 tags = merge(local.tags, {  
 Name = "${var.project\_name}-private-app-${each.key}",  
 Tier = "private-app"  
 })  
}  
  
resource "aws\_subnet" "private\_db" {  
 for\_each = { for idx, cidr in var.private\_db\_subnets : idx => cidr }  
 vpc\_id = aws\_vpc.this.id  
 cidr\_block = each.value  
 availability\_zone = var.azs[tonumber(each.key)]  
 tags = merge(local.tags, {  
 Name = "${var.project\_name}-private-db-${each.key}",  
 Tier = "private-db"  
 })  
}  
  
# NAT per AZ for HA  
resource "aws\_eip" "nat" {  
 for\_each = aws\_subnet.public  
 domain = "vpc"  
 tags = merge(local.tags, { Name = "${var.project\_name}-eip-nat-${each.key}" })  
}  
  
resource "aws\_nat\_gateway" "nat" {  
 for\_each = aws\_subnet.public  
 allocation\_id = aws\_eip.nat[each.key].id  
 subnet\_id = aws\_subnet.public[each.key].id  
 tags = merge(local.tags, { Name = "${var.project\_name}-nat-${each.key}" })  
 depends\_on = [aws\_internet\_gateway.igw]  
}  
  
# Route tables  
resource "aws\_route\_table" "public" {  
 vpc\_id = aws\_vpc.this.id  
 tags = merge(local.tags, { Name = "${var.project\_name}-rt-public" })  
}  
  
resource "aws\_route" "public\_internet" {  
 route\_table\_id = aws\_route\_table.public.id  
 destination\_cidr\_block = "0.0.0.0/0"  
 gateway\_id = aws\_internet\_gateway.igw.id  
}  
  
resource "aws\_route\_table\_association" "public\_assoc" {  
 for\_each = aws\_subnet.public  
 subnet\_id = each.value.id  
 route\_table\_id = aws\_route\_table.public.id  
}  
  
# Private-App RTs per AZ pointing to same‑AZ NAT  
resource "aws\_route\_table" "private\_app" {  
 for\_each = aws\_subnet.private\_app  
 vpc\_id = aws\_vpc.this.id  
 tags = merge(local.tags, { Name = "${var.project\_name}-rt-private-app-${each.key}" })  
}  
  
resource "aws\_route" "private\_app\_default" {  
 for\_each = aws\_route\_table.private\_app  
 route\_table\_id = each.value.id  
 destination\_cidr\_block = "0.0.0.0/0"  
 nat\_gateway\_id = aws\_nat\_gateway.nat[each.key].id  
}  
  
resource "aws\_route\_table\_association" "private\_app\_assoc" {  
 for\_each = aws\_subnet.private\_app  
 subnet\_id = each.value.id  
 route\_table\_id = aws\_route\_table.private\_app[each.key].id  
}  
  
# Private-DB RTs (no internet route)  
resource "aws\_route\_table" "private\_db" {  
 vpc\_id = aws\_vpc.this.id  
 tags = merge(local.tags, { Name = "${var.project\_name}-rt-private-db" })  
}  
  
resource "aws\_route\_table\_association" "private\_db\_assoc" {  
 for\_each = aws\_subnet.private\_db  
 subnet\_id = each.value.id  
 route\_table\_id = aws\_route\_table.private\_db.id  
}

### security\_groups.tf

resource "aws\_security\_group" "vpclink" {  
 name = "${var.project\_name}-sg-vpclink"  
 description = "SG for API GW VPC Link ENIs"  
 vpc\_id = aws\_vpc.this.id  
 egress { from\_port = 0 to\_port = 0 protocol = "-1" cidr\_blocks = [var.vpc\_cidr] }  
 tags = merge(local.tags, { Name = "${var.project\_name}-sg-vpclink" })  
}  
  
resource "aws\_security\_group" "alb" {  
 name = "${var.project\_name}-sg-alb"  
 vpc\_id = aws\_vpc.this.id  
 ingress {  
 description = "From VPC Link"  
 from\_port = 80  
 to\_port = 80  
 protocol = "tcp"  
 security\_groups = [aws\_security\_group.vpclink.id]  
 }  
 egress { from\_port = 0 to\_port = 0 protocol = "-1" cidr\_blocks = [var.vpc\_cidr] }  
 tags = merge(local.tags, { Name = "${var.project\_name}-sg-alb" })  
}  
  
resource "aws\_security\_group" "ecs\_api" {  
 name = "${var.project\_name}-sg-ecs-api"  
 vpc\_id = aws\_vpc.this.id  
 ingress {  
 description = "ALB to API"  
 from\_port = var.container\_port  
 to\_port = var.container\_port  
 protocol = "tcp"  
 security\_groups = [aws\_security\_group.alb.id]  
 }  
 egress {  
 from\_port = 0  
 to\_port = 0  
 protocol = "-1"  
 cidr\_blocks = ["0.0.0.0/0"]  
 }  
 tags = merge(local.tags, { Name = "${var.project\_name}-sg-ecs-api" })  
}  
  
resource "aws\_security\_group" "ecs\_worker" {  
 name = "${var.project\_name}-sg-ecs-worker"  
 vpc\_id = aws\_vpc.this.id  
 egress { from\_port = 0 to\_port = 0 protocol = "-1" cidr\_blocks = ["0.0.0.0/0"] }  
 tags = merge(local.tags, { Name = "${var.project\_name}-sg-ecs-worker" })  
}  
  
resource "aws\_security\_group" "db" {  
 name = "${var.project\_name}-sg-db"  
 vpc\_id = aws\_vpc.this.id  
 ingress {  
 description = "DB from ECS API"  
 from\_port = 3306  
 to\_port = 3306  
 protocol = "tcp"  
 security\_groups = [aws\_security\_group.ecs\_api.id, aws\_security\_group.ecs\_worker.id]  
 }  
 egress { from\_port = 0 to\_port = 0 protocol = "-1" cidr\_blocks = [var.vpc\_cidr] }  
 tags = merge(local.tags, { Name = "${var.project\_name}-sg-db" })  
}

### endpoints.tf (Optional VPC Interface Endpoints for SNS/SQS)

resource "aws\_vpc\_endpoint" "sqs" {  
 vpc\_id = aws\_vpc.this.id  
 service\_name = "com.amazonaws.${var.region}.sqs"  
 vpc\_endpoint\_type = "Interface"  
 subnet\_ids = [for s in aws\_subnet.private\_app : s.id]  
 security\_group\_ids = [aws\_security\_group.ecs\_api.id]  
 private\_dns\_enabled = true  
 tags = merge(local.tags, { Name = "${var.project\_name}-vpce-sqs" })  
}  
  
resource "aws\_vpc\_endpoint" "sns" {  
 vpc\_id = aws\_vpc.this.id  
 service\_name = "com.amazonaws.${var.region}.sns"  
 vpc\_endpoint\_type = "Interface"  
 subnet\_ids = [for s in aws\_subnet.private\_app : s.id]  
 security\_group\_ids = [aws\_security\_group.ecs\_api.id]  
 private\_dns\_enabled = true  
 tags = merge(local.tags, { Name = "${var.project\_name}-vpce-sns" })  
}

### rds.tf

resource "aws\_db\_subnet\_group" "this" {  
 name = "${var.project\_name}-db-subnets"  
 subnet\_ids = [for s in aws\_subnet.private\_db : s.id]  
 tags = merge(local.tags, { Name = "${var.project\_name}-db-subnets" })  
}  
  
resource "aws\_rds\_cluster" "this" {  
 cluster\_identifier = "${var.project\_name}-aurora"  
 engine = "aurora-mysql"  
 engine\_mode = "provisioned"  
 engine\_version = "8.0.mysql\_aurora.3.06.0" # adjust as needed  
 database\_name = var.db\_name  
 master\_username = var.db\_username  
 master\_password = var.db\_password  
 db\_subnet\_group\_name = aws\_db\_subnet\_group.this.name  
 vpc\_security\_group\_ids = [aws\_security\_group.db.id]  
 backup\_retention\_period = 7  
 preferred\_backup\_window = "03:00-04:00"  
 deletion\_protection = true  
 storage\_encrypted = true  
 apply\_immediately = true  
 tags = merge(local.tags, { Name = "${var.project\_name}-aurora" })  
}  
  
resource "aws\_rds\_cluster\_instance" "writer" {  
 identifier = "${var.project\_name}-aurora-writer"  
 cluster\_identifier = aws\_rds\_cluster.this.id  
 instance\_class = "db.r6g.large"  
 engine = aws\_rds\_cluster.this.engine  
 engine\_version = aws\_rds\_cluster.this.engine\_version  
 publicly\_accessible = false  
 db\_subnet\_group\_name = aws\_db\_subnet\_group.this.name  
 tags = merge(local.tags, { Role = "writer" })  
}  
  
resource "aws\_rds\_cluster\_instance" "reader" {  
 identifier = "${var.project\_name}-aurora-reader"  
 cluster\_identifier = aws\_rds\_cluster.this.id  
 instance\_class = "db.r6g.large"  
 engine = aws\_rds\_cluster.this.engine  
 engine\_version = aws\_rds\_cluster.this.engine\_version  
 publicly\_accessible = false  
 db\_subnet\_group\_name = aws\_db\_subnet\_group.this.name  
 tags = merge(local.tags, { Role = "reader" })  
}

### secrets.tf

resource "aws\_secretsmanager\_secret" "db" {  
 name = "${var.project\_name}/db"  
 tags = local.tags  
}  
  
resource "aws\_secretsmanager\_secret\_version" "db" {  
 secret\_id = aws\_secretsmanager\_secret.db.id  
 secret\_string = jsonencode({  
 username = var.db\_username,  
 password = var.db\_password,  
 engine = "mysql",  
 host = aws\_rds\_cluster.this.endpoint,  
 port = 3306,  
 dbname = var.db\_name  
 })  
}

### sns\_sqs.tf

resource "aws\_sns\_topic" "identity" {  
 name = "${var.project\_name}-identity-topic"  
 tags = local.tags  
}  
  
resource "aws\_sqs\_queue" "dlq" {  
 name = "${var.project\_name}-identity-dlq"  
 message\_retention\_seconds = 1209600  
 tags = local.tags  
}  
  
resource "aws\_sqs\_queue" "queue" {  
 name = "${var.project\_name}-identity-queue"  
 visibility\_timeout\_seconds = 60  
 message\_retention\_seconds = 345600  
 receive\_wait\_time\_seconds = 20  
 redrive\_policy = jsonencode({  
 deadLetterTargetArn = aws\_sqs\_queue.dlq.arn,  
 maxReceiveCount = 5  
 })  
 tags = local.tags  
}  
  
resource "aws\_sns\_topic\_subscription" "sub" {  
 topic\_arn = aws\_sns\_topic.identity.arn  
 protocol = "sqs"  
 endpoint = aws\_sqs\_queue.queue.arn  
 raw\_message\_delivery = true  
}  
  
# Allow SNS to publish to SQS  
data "aws\_iam\_policy\_document" "sqs\_policy" {  
 statement {  
 actions = ["SQS:SendMessage"]  
 resources = [aws\_sqs\_queue.queue.arn]  
 principals { type = "Service" identifiers = ["sns.amazonaws.com"] }  
 condition {  
 test = "ArnEquals"  
 variable = "aws:SourceArn"  
 values = [aws\_sns\_topic.identity.arn]  
 }  
 }  
}  
  
resource "aws\_sqs\_queue\_policy" "queue" {  
 queue\_url = aws\_sqs\_queue.queue.id  
 policy = data.aws\_iam\_policy\_document.sqs\_policy.json  
}

### ecr.tf

resource "aws\_ecr\_repository" "api" {  
 name = "${var.project\_name}-identity-api"  
 image\_scanning\_configuration { scan\_on\_push = true }  
 tags = local.tags  
}  
  
resource "aws\_ecr\_repository" "worker" {  
 name = "${var.project\_name}-identity-worker"  
 image\_scanning\_configuration { scan\_on\_push = true }  
 tags = local.tags  
}

### iam.tf

# Execution role for ECS tasks  
resource "aws\_iam\_role" "ecs\_execution" {  
 name = "${var.project\_name}-ecs-exec-role"  
 assume\_role\_policy = jsonencode({  
 Version = "2012-10-17",  
 Statement = [{  
 Effect = "Allow",  
 Principal = { Service = "ecs-tasks.amazonaws.com" },  
 Action = "sts:AssumeRole"  
 }]  
 })  
}  
  
resource "aws\_iam\_role\_policy\_attachment" "exec\_base" {  
 role = aws\_iam\_role.ecs\_execution.name  
 policy\_arn = "arn:aws:iam::aws:policy/service-role/AmazonECSTaskExecutionRolePolicy"  
}  
  
# Allow execution role to read Secrets Manager for DB string (pull at startup)  
resource "aws\_iam\_policy" "exec\_secrets" {  
 name = "${var.project\_name}-ecs-exec-secrets"  
 policy = jsonencode({  
 Version = "2012-10-17",  
 Statement = [{  
 Effect = "Allow",  
 Action = ["secretsmanager:GetSecretValue"],  
 Resource = [aws\_secretsmanager\_secret.db.arn]  
 }]  
 })  
}  
  
resource "aws\_iam\_role\_policy\_attachment" "exec\_secrets\_attach" {  
 role = aws\_iam\_role.ecs\_execution.name  
 policy\_arn = aws\_iam\_policy.exec\_secrets.arn  
}  
  
# Task role for API  
resource "aws\_iam\_role" "task\_api" {  
 name = "${var.project\_name}-ecs-task-role-api"  
 assume\_role\_policy = aws\_iam\_role.ecs\_execution.assume\_role\_policy  
}  
  
resource "aws\_iam\_policy" "api\_policy" {  
 name = "${var.project\_name}-api-policy"  
 policy = jsonencode({  
 Version = "2012-10-17",  
 Statement = [  
 {  
 Effect: "Allow",  
 Action: ["sns:Publish"],  
 Resource: [aws\_sns\_topic.identity.arn]  
 },  
 {  
 Effect: "Allow",  
 Action: ["secretsmanager:GetSecretValue"],  
 Resource: [aws\_secretsmanager\_secret.db.arn]  
 }  
 ]  
 })  
}  
  
resource "aws\_iam\_role\_policy\_attachment" "api\_attach" {  
 role = aws\_iam\_role.task\_api.name  
 policy\_arn = aws\_iam\_policy.api\_policy.arn  
}  
  
# Task role for Worker  
resource "aws\_iam\_role" "task\_worker" {  
 name = "${var.project\_name}-ecs-task-role-worker"  
 assume\_role\_policy = aws\_iam\_role.ecs\_execution.assume\_role\_policy  
}  
  
resource "aws\_iam\_policy" "worker\_policy" {  
 name = "${var.project\_name}-worker-policy"  
 policy = jsonencode({  
 Version = "2012-10-17",  
 Statement = [  
 {  
 Effect: "Allow",  
 Action: [  
 "sqs:ReceiveMessage", "sqs:DeleteMessage", "sqs:GetQueueUrl", "sqs:ChangeMessageVisibility"  
 ],  
 Resource: [aws\_sqs\_queue.queue.arn]  
 },  
 {  
 Effect: "Allow",  
 Action: ["secretsmanager:GetSecretValue"],  
 Resource: [aws\_secretsmanager\_secret.db.arn]  
 }  
 ]  
 })  
}  
  
resource "aws\_iam\_role\_policy\_attachment" "worker\_attach" {  
 role = aws\_iam\_role.task\_worker.name  
 policy\_arn = aws\_iam\_policy.worker\_policy.arn  
}

### ecs\_cluster.tf

resource "aws\_ecs\_cluster" "this" {  
 name = "${var.project\_name}-cluster"  
 setting { name = "containerInsights" value = "enabled" }  
 tags = local.tags  
}  
  
resource "aws\_cloudwatch\_log\_group" "api" {  
 name = "/${var.project\_name}/api"  
 retention\_in\_days = 30  
}  
  
resource "aws\_cloudwatch\_log\_group" "worker" {  
 name = "/${var.project\_name}/worker"  
 retention\_in\_days = 30  
}

### alb.tf

resource "aws\_lb" "internal" {  
 name = "${var.project\_name}-alb-int"  
 internal = true  
 load\_balancer\_type = "application"  
 subnets = [for s in aws\_subnet.private\_app : s.id]  
 security\_groups = [aws\_security\_group.alb.id]  
 tags = local.tags  
}  
  
resource "aws\_lb\_target\_group" "api" {  
 name = "${var.project\_name}-tg-api"  
 port = var.container\_port  
 protocol = "HTTP"  
 vpc\_id = aws\_vpc.this.id  
 target\_type = "ip"  
 health\_check {  
 path = "/health"  
 healthy\_threshold = 2  
 unhealthy\_threshold = 2  
 timeout = 5  
 interval = 20  
 matcher = "200-399"  
 }  
 tags = local.tags  
}  
  
resource "aws\_lb\_listener" "http" {  
 load\_balancer\_arn = aws\_lb.internal.arn  
 port = 80  
 protocol = "HTTP"  
 default\_action {  
 type = "forward"  
 target\_group\_arn = aws\_lb\_target\_group.api.arn  
 }  
}

### apigw\_vpclink.tf

resource "aws\_apigatewayv2\_vpc\_link" "this" {  
 name = "${var.project\_name}-vpclink"  
 security\_group\_ids = [aws\_security\_group.vpclink.id]  
 subnet\_ids = [for s in aws\_subnet.private\_app : s.id]  
 tags = local.tags  
}  
  
resource "aws\_apigatewayv2\_api" "http" {  
 name = "${var.project\_name}-httpapi"  
 protocol\_type = "HTTP"  
 tags = local.tags  
}  
  
# Integration to ALB Listener via VPC Link  
resource "aws\_apigatewayv2\_integration" "alb" {  
 api\_id = aws\_apigatewayv2\_api.http.id  
 integration\_type = "HTTP\_PROXY"  
 integration\_method = "ANY"  
 connection\_type = "VPC\_LINK"  
 connection\_id = aws\_apigatewayv2\_vpc\_link.this.id  
 integration\_uri = aws\_lb\_listener.http.arn  
 payload\_format\_version = "1.0"  
}  
  
# Routes  
resource "aws\_apigatewayv2\_route" "identities\_post" {  
 api\_id = aws\_apigatewayv2\_api.http.id  
 route\_key = "POST /identities"  
 target = "integrations/${aws\_apigatewayv2\_integration.alb.id}"  
}  
  
resource "aws\_apigatewayv2\_route" "identities\_batch" {  
 api\_id = aws\_apigatewayv2\_api.http.id  
 route\_key = "POST /identities/batch"  
 target = "integrations/${aws\_apigatewayv2\_integration.alb.id}"  
}  
  
resource "aws\_apigatewayv2\_route" "identities\_get" {  
 api\_id = aws\_apigatewayv2\_api.http.id  
 route\_key = "GET /identities/{id}"  
 target = "integrations/${aws\_apigatewayv2\_integration.alb.id}"  
}  
  
resource "aws\_apigatewayv2\_stage" "default" {  
 api\_id = aws\_apigatewayv2\_api.http.id  
 name = "$default"  
 auto\_deploy = true  
 access\_log\_settings {  
 destination\_arn = aws\_cloudwatch\_log\_group.api.arn  
 format = jsonencode({  
 requestId = "$context.requestId",  
 httpMethod = "$context.httpMethod",  
 path = "$context.path",  
 status = "$context.status",  
 ip = "$context.identity.sourceIp"  
 })  
 }  
}

### ecs\_task\_api.tf

resource "aws\_ecs\_task\_definition" "api" {  
 family = "${var.project\_name}-api"  
 requires\_compatibilities = ["FARGATE"]  
 network\_mode = "awsvpc"  
 cpu = 512  
 memory = 1024  
 execution\_role\_arn = aws\_iam\_role.ecs\_execution.arn  
 task\_role\_arn = aws\_iam\_role.task\_api.arn  
  
 container\_definitions = jsonencode([  
 {  
 name = "api",  
 image = "${aws\_ecr\_repository.api.repository\_url}:latest",  
 essential = true,  
 portMappings = [{ containerPort = var.container\_port, protocol = "tcp" }],  
 logConfiguration = {  
 logDriver = "awslogs",  
 options = {  
 awslogs-group = aws\_cloudwatch\_log\_group.api.name,  
 awslogs-region = var.region,  
 awslogs-stream-prefix = "api"  
 }  
 },  
 environment = [  
 { name = "ASPNETCORE\_URLS", value = "http://0.0.0.0:${var.container\_port}" },  
 { name = "SNS\_TOPIC\_ARN", value = aws\_sns\_topic.identity.arn },  
 { name = "DB\_SECRET\_ARN", value = aws\_secretsmanager\_secret.db.arn }  
 ]  
 }  
 ])  
}  
  
resource "aws\_ecs\_service" "api" {  
 name = "${var.project\_name}-svc-api"  
 cluster = aws\_ecs\_cluster.this.id  
 task\_definition = aws\_ecs\_task\_definition.api.arn  
 desired\_count = var.desired\_count\_api  
 launch\_type = "FARGATE"  
  
 network\_configuration {  
 subnets = [for s in aws\_subnet.private\_app : s.id]  
 security\_groups = [aws\_security\_group.ecs\_api.id]  
 assign\_public\_ip = false  
 }  
  
 load\_balancer {  
 target\_group\_arn = aws\_lb\_target\_group.api.arn  
 container\_name = "api"  
 container\_port = var.container\_port  
 }  
  
 deployment\_minimum\_healthy\_percent = 50  
 deployment\_maximum\_percent = 200  
  
 depends\_on = [aws\_lb\_listener.http]  
}

### ecs\_task\_worker.tf

resource "aws\_ecs\_task\_definition" "worker" {  
 family = "${var.project\_name}-worker"  
 requires\_compatibilities = ["FARGATE"]  
 network\_mode = "awsvpc"  
 cpu = 512  
 memory = 1024  
 execution\_role\_arn = aws\_iam\_role.ecs\_execution.arn  
 task\_role\_arn = aws\_iam\_role.task\_worker.arn  
  
 container\_definitions = jsonencode([  
 {  
 name = "worker",  
 image = "${aws\_ecr\_repository.worker.repository\_url}:latest",  
 essential = true,  
 logConfiguration = {  
 logDriver = "awslogs",  
 options = {  
 awslogs-group = aws\_cloudwatch\_log\_group.worker.name,  
 awslogs-region = var.region,  
 awslogs-stream-prefix = "worker"  
 }  
 },  
 environment = [  
 { name = "SQS\_QUEUE\_URL", value = aws\_sqs\_queue.queue.id },  
 { name = "DB\_SECRET\_ARN", value = aws\_secretsmanager\_secret.db.arn },  
 { name = "THIRD\_PARTY\_BASE\_URL", value = var.third\_party\_api\_base\_url }  
 ]  
 }  
 ])  
}  
  
resource "aws\_appautoscaling\_target" "worker\_sqs" {  
 max\_capacity = 10  
 min\_capacity = var.desired\_count\_worker  
 resource\_id = "service/${aws\_ecs\_cluster.this.name}/${aws\_ecs\_service.worker.name}"  
 scalable\_dimension = "ecs:service:DesiredCount"  
 service\_namespace = "ecs"  
}  
  
resource "aws\_appautoscaling\_policy" "worker\_scale\_on\_sqs" {  
 name = "${var.project\_name}-worker-queue-scaling"  
 policy\_type = "StepScaling"  
 resource\_id = aws\_appautoscaling\_target.worker\_sqs.resource\_id  
 scalable\_dimension = aws\_appautoscaling\_target.worker\_sqs.scalable\_dimension  
 service\_namespace = aws\_appautoscaling\_target.worker\_sqs.service\_namespace  
  
 step\_scaling\_policy\_configuration {  
 adjustment\_type = "ChangeInCapacity"  
 cooldown = 60  
 metric\_aggregation\_type = "Average"  
 step\_adjustment { scaling\_adjustment = 1 metric\_interval\_lower\_bound = 0 }  
 }  
}  
  
# CloudWatch metric alarm on ApproximateNumberOfMessagesVisible (not shown) should trigger policy (add if desired)  
  
resource "aws\_ecs\_service" "worker" {  
 name = "${var.project\_name}-svc-worker"  
 cluster = aws\_ecs\_cluster.this.id  
 task\_definition = aws\_ecs\_task\_definition.worker.arn  
 desired\_count = var.desired\_count\_worker  
 launch\_type = "FARGATE"  
  
 network\_configuration {  
 subnets = [for s in aws\_subnet.private\_app : s.id]  
 security\_groups = [aws\_security\_group.ecs\_worker.id]  
 assign\_public\_ip = false  
 }  
}

### outputs.tf

output "api\_gateway\_endpoint" {  
 value = aws\_apigatewayv2\_api.http.api\_endpoint  
}  
  
output "aurora\_writer\_endpoint" {  
 value = aws\_rds\_cluster.this.endpoint  
}  
  
output "sqs\_queue\_url" {  
 value = aws\_sqs\_queue.queue.id  
}  
  
output "sns\_topic\_arn" {  
 value = aws\_sns\_topic.identity.arn  
}

## 4) .NET 8 — Minimal API & Worker (Skeleton)

These are intentionally concise to show the message flow and DB update loop. Use EF Core or Dapper as you prefer. Below uses EF Core.

### Domain Model (shared)

// Models/IdentityRecord.cs  
public class IdentityRecord  
{  
 public long Id { get; set; }  
 public string NationalId { get; set; } = default!;  
 public string FirstName { get; set; } = default!;  
 public string LastName { get; set; } = default!;  
 public string Status { get; set; } = "PENDING"; // PENDING, VALIDATED, FAILED  
 public string? ExternalReference { get; set; }  
 public string? ValidationResponseJson { get; set; }  
 public DateTime CreatedAt { get; set; } = DateTime.UtcNow;  
 public DateTime UpdatedAt { get; set; } = DateTime.UtcNow;  
}

// Data/AppDbContext.cs  
using Microsoft.EntityFrameworkCore;  
public class AppDbContext : DbContext  
{  
 public DbSet<IdentityRecord> Identities => Set<IdentityRecord>();  
 public AppDbContext(DbContextOptions<AppDbContext> options) : base(options) {}  
 protected override void OnModelCreating(ModelBuilder b)  
 {  
 b.Entity<IdentityRecord>().ToTable("identities");  
 b.Entity<IdentityRecord>().Property(p => p.Id).ValueGeneratedOnAdd();  
 b.Entity<IdentityRecord>().HasIndex(p => p.NationalId);  
 }  
}

### API Project (Minimal APIs)

// Program.cs  
using Amazon;  
using Amazon.SecretsManager;  
using Amazon.SecretsManager.Model;  
using Amazon.SimpleNotificationService;  
using Amazon.SimpleNotificationService.Model;  
using Microsoft.EntityFrameworkCore;  
using System.Text.Json;  
  
var builder = WebApplication.CreateBuilder(args);  
  
// DB secret  
var secretArn = Environment.GetEnvironmentVariable("DB\_SECRET\_ARN")!;  
var snsTopicArn = Environment.GetEnvironmentVariable("SNS\_TOPIC\_ARN")!;  
  
var sm = new AmazonSecretsManagerClient(RegionEndpoint.AFSouth1);  
var sec = await sm.GetSecretValueAsync(new GetSecretValueRequest{ SecretId = secretArn });  
var cfg = JsonSerializer.Deserialize<DbSecret>(sec.SecretString!);  
  
builder.Services.AddDbContext<AppDbContext>(opt =>  
 opt.UseMySql($"server={cfg!.host};port={cfg.port};database={cfg.dbname};user={cfg.username};password={cfg.password}",  
 ServerVersion.AutoDetect($"server={cfg!.host};port={cfg.port};database={cfg.dbname}"))  
);  
  
builder.Services.AddSingleton<IAmazonSimpleNotificationService>(new AmazonSimpleNotificationServiceClient(RegionEndpoint.AFSouth1));  
  
var app = builder.Build();  
app.MapGet("/health", () => Results.Ok(new { status = "ok" }));  
  
app.MapPost("/identities", async (IdentityDto dto, AppDbContext db, IAmazonSimpleNotificationService sns) =>  
{  
 var rec = new IdentityRecord {  
 NationalId = dto.NationalId,  
 FirstName = dto.FirstName,  
 LastName = dto.LastName,  
 Status = "PENDING"  
 };  
 db.Identities.Add(rec);  
 await db.SaveChangesAsync();  
  
 var msg = JsonSerializer.Serialize(new IdentityMessage { IdentityId = rec.Id });  
 await sns.PublishAsync(new PublishRequest { TopicArn = snsTopicArn, Message = msg });  
 return Results.Accepted($"/identities/{rec.Id}", new { id = rec.Id, status = rec.Status });  
});  
  
app.MapPost("/identities/batch", async (List<IdentityDto> dtos, AppDbContext db, IAmazonSimpleNotificationService sns) =>  
{  
 var records = dtos.Select(d => new IdentityRecord{  
 NationalId = d.NationalId, FirstName = d.FirstName, LastName = d.LastName, Status = "PENDING"  
 }).ToList();  
  
 await db.AddRangeAsync(records);  
 await db.SaveChangesAsync();  
  
 foreach (var r in records)  
 {  
 var msg = JsonSerializer.Serialize(new IdentityMessage { IdentityId = r.Id });  
 await sns.PublishAsync(new PublishRequest { TopicArn = snsTopicArn, Message = msg });  
 }  
  
 return Results.Accepted("/identities", new { count = records.Count });  
});  
  
app.MapGet("/identities/{id:long}", async (long id, AppDbContext db) =>  
{  
 var rec = await db.Identities.FindAsync(id);  
 return rec is null ? Results.NotFound() : Results.Ok(rec);  
});  
  
app.Run();  
  
record DbSecret(string username, string password, string host, int port, string dbname);  
record IdentityDto(string NationalId, string FirstName, string LastName);  
record IdentityMessage { public long IdentityId { get; set; } }

### Worker Project (Background SQS Processor)

// Program.cs  
using Amazon;  
using Amazon.SecretsManager;  
using Amazon.SecretsManager.Model;  
using Amazon.SQS;  
using Amazon.SQS.Model;  
using Microsoft.EntityFrameworkCore;  
using System.Net.Http.Json;  
using System.Text.Json;  
  
var builder = Host.CreateApplicationBuilder(args);  
var secretArn = Environment.GetEnvironmentVariable("DB\_SECRET\_ARN")!;  
var queueUrl = Environment.GetEnvironmentVariable("SQS\_QUEUE\_URL")!;  
var baseUrl = Environment.GetEnvironmentVariable("THIRD\_PARTY\_BASE\_URL")!;  
  
var sm = new AmazonSecretsManagerClient(RegionEndpoint.AFSouth1);  
var sec = await sm.GetSecretValueAsync(new GetSecretValueRequest{ SecretId = secretArn });  
var cfg = JsonSerializer.Deserialize<DbSecret>(sec.SecretString!);  
  
builder.Services.AddDbContext<AppDbContext>(opt =>  
 opt.UseMySql($"server={cfg!.host};port={cfg.port};database={cfg.dbname};user={cfg.username};password={cfg.password}",  
 ServerVersion.AutoDetect($"server={cfg!.host};port={cfg.port};database={cfg.dbname}"))  
);  
  
builder.Services.AddSingleton<IAmazonSQS>(new AmazonSQSClient(RegionEndpoint.AFSouth1));  
builder.Services.AddHttpClient("third", c => c.BaseAddress = new Uri(baseUrl));  
  
builder.Services.AddHostedService<Worker>();  
var app = builder.Build();  
await app.RunAsync();  
  
class Worker : BackgroundService  
{  
 private readonly IAmazonSQS \_sqs; private readonly IServiceProvider \_sp; private readonly IHttpClientFactory \_http;  
 private readonly string \_queueUrl = Environment.GetEnvironmentVariable("SQS\_QUEUE\_URL")!;  
 public Worker(IAmazonSQS sqs, IServiceProvider sp, IHttpClientFactory http){ \_sqs=sqs; \_sp=sp; \_http=http; }  
  
 protected override async Task ExecuteAsync(CancellationToken stoppingToken)  
 {  
 while(!stoppingToken.IsCancellationRequested)  
 {  
 var resp = await \_sqs.ReceiveMessageAsync(new ReceiveMessageRequest{  
 QueueUrl = \_queueUrl, MaxNumberOfMessages = 10, WaitTimeSeconds = 20, VisibilityTimeout = 60  
 }, stoppingToken);  
  
 foreach (var m in resp.Messages)  
 {  
 try {  
 var msg = JsonSerializer.Deserialize<IdentityMessage>(m.Body)!;  
 using var scope = \_sp.CreateScope();  
 var db = scope.ServiceProvider.GetRequiredService<AppDbContext>();  
 var rec = await db.Identities.FindAsync(new object?[]{ msg.IdentityId }, stoppingToken);  
 if (rec == null) { await \_sqs.DeleteMessageAsync(\_queueUrl, m.ReceiptHandle, stoppingToken); continue; }  
  
 var client = \_http.CreateClient("third");  
 var third = await client.PostAsJsonAsync("/validate", new { rec.NationalId, rec.FirstName, rec.LastName }, stoppingToken);  
 var payload = await third.Content.ReadAsStringAsync(stoppingToken);  
  
 rec.ValidationResponseJson = payload;  
 rec.Status = third.IsSuccessStatusCode ? "VALIDATED" : "FAILED";  
 rec.UpdatedAt = DateTime.UtcNow;  
 await db.SaveChangesAsync(stoppingToken);  
  
 await \_sqs.DeleteMessageAsync(\_queueUrl, m.ReceiptHandle, stoppingToken);  
 }  
 catch(Exception)  
 {  
 // Let visibility timeout expire; after 5 receives DLQ gets it per redrive policy.  
 }  
 }  
 }  
 }  
}  
  
record DbSecret(string username, string password, string host, int port, string dbname);  
record IdentityMessage { public long IdentityId { get; set; } }

### Simple EF Core Migration (Optional CLI)

# in API project folder  
dotnet add package Pomelo.EntityFrameworkCore.MySql  
dotnet ef migrations add Init --project ./Api --startup-project ./Api  
# apply via app startup or separate migration runner connecting to Aurora

### Dockerfiles

**API**

# Api/Dockerfile  
FROM mcr.microsoft.com/dotnet/aspnet:8.0 AS base  
WORKDIR /app  
EXPOSE 8080  
  
FROM mcr.microsoft.com/dotnet/sdk:8.0 AS build  
WORKDIR /src  
COPY . .  
RUN dotnet restore Api/Api.csproj && dotnet publish Api/Api.csproj -c Release -o /out  
  
FROM base AS final  
WORKDIR /app  
COPY --from=build /out .  
ENV ASPNETCORE\_URLS=http://+:8080  
ENTRYPOINT ["dotnet", "Api.dll"]

**Worker**

# Worker/Dockerfile  
FROM mcr.microsoft.com/dotnet/runtime:8.0 AS base  
WORKDIR /app  
  
FROM mcr.microsoft.com/dotnet/sdk:8.0 AS build  
WORKDIR /src  
COPY . .  
RUN dotnet restore Worker/Worker.csproj && dotnet publish Worker/Worker.csproj -c Release -o /out  
  
FROM base AS final  
WORKDIR /app  
COPY --from=build /out .  
ENTRYPOINT ["dotnet", "Worker.dll"]

### Build & Push to ECR

AWS\_REGION=af-south-1  
ACCOUNT\_ID=$(aws sts get-caller-identity --query Account --output text)  
  
# Login  
aws ecr get-login-password --region $AWS\_REGION | \  
 docker login --username AWS --password-stdin $ACCOUNT\_ID.dkr.ecr.$AWS\_REGION.amazonaws.com  
  
# API  
API\_REPO=$(aws ecr describe-repositories --repository-names ${project\_name}-identity-api --query 'repositories[0].repositoryUri' --output text)  
docker build -t identity-api:latest -f Api/Dockerfile .  
docker tag identity-api:latest $API\_REPO:latest  
docker push $API\_REPO:latest  
  
# Worker  
WORKER\_REPO=$(aws ecr describe-repositories --repository-names ${project\_name}-identity-worker --query 'repositories[0].repositoryUri' --output text)  
docker build -t identity-worker:latest -f Worker/Dockerfile .  
docker tag identity-worker:latest $WORKER\_REPO:latest  
docker push $WORKER\_REPO:latest

## 5) Operational Notes

* **Idempotency**: Include a messageId and dedupe on IdentityId + version to avoid double updates when SQS redelivers.
* **Retries**: Third‑party API failures rely on SQS redelivery → DLQ after maxReceiveCount. Add circuit breaker/backoff in Worker.
* **Security**: Add WAF to API Gateway, rotate DB credentials via Secrets Manager, enforce TLS from API GW to ALB if needed.
* **Migrations**: Run schema migrations on deploy (one‑off task) before rolling out new app versions.
* **Scaling**: Tie Worker scaling to ApproximateNumberOfMessagesVisible via CloudWatch alarm or EventBridge.
* **Costs**: NAT x2 + Aurora + Fargate + API GW + ALB. Consider VPC endpoints to reduce NAT egress for AWS‑to‑AWS.

## 6) Testing the Flow

1. POST https://{api\_id}.execute-api.af-south-1.amazonaws.com/identities with JSON { "nationalId": "123", "firstName": "A", "lastName": "B" } → **202 Accepted**.
2. Record appears in Aurora as **PENDING**.
3. Message appears in **SQS** (via SNS fanout).
4. Worker consumes, calls third‑party /validate, updates row to **VALIDATED/FAILED** with response JSON.
5. GET /identities/{id} returns final status.

## 7) TLS on Internal ALB + WAF on API Gateway (Add‑On)

This section adds **TLS termination** on the internal ALB and attaches an **AWS WAFv2 Web ACL** to the API Gateway HTTP API stage.

**Cert Strategy**: Provide an ACM certificate ARN via var.alb\_cert\_arn. You can: - **Import** an internally issued certificate into ACM (same region). - Or use **ACM Private CA** (paid) to issue a private cert for an internal domain. In both cases, create a **Route 53 private hosted zone** (e.g., svc.internal.example) and map a record (e.g., identity.svc.internal.example) to the **internal ALB** using an alias.

### Variables — add to variables.tf

variable "alb\_cert\_arn" {  
 type = string  
 description = "ACM certificate ARN for the internal ALB HTTPS listener"  
}  
  
variable "enable\_waf" {  
 type = bool  
 default = true  
 description = "Attach WAFv2 Web ACL to API Gateway stage"  
}

### Security Groups — update security\_groups.tf

Add 443 from VPC Link to ALB SG (keep 80 for health or redirect):

resource "aws\_security\_group" "alb" {  
 name = "${var.project\_name}-sg-alb"  
 vpc\_id = aws\_vpc.this.id  
  
 ingress {  
 description = "From VPC Link HTTP"  
 from\_port = 80  
 to\_port = 80  
 protocol = "tcp"  
 security\_groups = [aws\_security\_group.vpclink.id]  
 }  
 ingress {  
 description = "From VPC Link HTTPS"  
 from\_port = 443  
 to\_port = 443  
 protocol = "tcp"  
 security\_groups = [aws\_security\_group.vpclink.id]  
 }  
 egress { from\_port = 0 to\_port = 0 protocol = "-1" cidr\_blocks = [var.vpc\_cidr] }  
 tags = merge(local.tags, { Name = "${var.project\_name}-sg-alb" })  
}

### ALB — update alb.tf

Add an HTTPS listener using the ACM cert; optionally redirect 80→443:

# Existing internal ALB and target group remain unchanged  
  
resource "aws\_lb\_listener" "http" {  
 load\_balancer\_arn = aws\_lb.internal.arn  
 port = 80  
 protocol = "HTTP"  
 default\_action {  
 type = "redirect"  
 redirect {  
 port = "443"  
 protocol = "HTTPS"  
 status\_code = "HTTP\_301"  
 }  
 }  
}  
  
resource "aws\_lb\_listener" "https" {  
 load\_balancer\_arn = aws\_lb.internal.arn  
 port = 443  
 protocol = "HTTPS"  
 ssl\_policy = "ELBSecurityPolicy-TLS13-1-2-2021-06"  
 certificate\_arn = var.alb\_cert\_arn  
 default\_action {  
 type = "forward"  
 target\_group\_arn = aws\_lb\_target\_group.api.arn  
 }  
}

### API Gateway Integration — update apigw\_vpclink.tf

Point the HTTP API integration to the **HTTPS listener ARN** instead of HTTP:

resource "aws\_apigatewayv2\_integration" "alb" {  
 api\_id = aws\_apigatewayv2\_api.http.id  
 integration\_type = "HTTP\_PROXY"  
 integration\_method = "ANY"  
 connection\_type = "VPC\_LINK"  
 connection\_id = aws\_apigatewayv2\_vpc\_link.this.id  
 integration\_uri = aws\_lb\_listener.https.arn # switched to HTTPS listener  
 payload\_format\_version = "1.0"  
}

**Note**: API Gateway over VPC Link will now speak HTTPS to the ALB using the private certificate you supplied.

### WAFv2 — new file waf.tf

This creates a sane default Web ACL with AWS Managed Rule groups and associates it to the $default stage when enable\_waf=true.

resource "aws\_wafv2\_web\_acl" "httpapi" {  
 name = "${var.project\_name}-httpapi-waf"  
 description = "WAF for HTTP API"  
 scope = "REGIONAL"  
 default\_action { allow {} }  
  
 rule {  
 name = "AWS-AWSManagedRulesCommonRuleSet"  
 priority = 1  
 statement { managed\_rule\_group\_statement { name = "AWSManagedRulesCommonRuleSet" vendor\_name = "AWS" } }  
 visibility\_config { cloudwatch\_metrics\_enabled = true metric\_name = "common" sampled\_requests\_enabled = true }  
 }  
 rule {  
 name = "AWS-AWSManagedRulesKnownBadInputsRuleSet"  
 priority = 2  
 statement { managed\_rule\_group\_statement { name = "AWSManagedRulesKnownBadInputsRuleSet" vendor\_name = "AWS" } }  
 visibility\_config { cloudwatch\_metrics\_enabled = true metric\_name = "badinputs" sampled\_requests\_enabled = true }  
 }  
 rule {  
 name = "AWS-AWSManagedRulesSQLiRuleSet"  
 priority = 3  
 statement { managed\_rule\_group\_statement { name = "AWSManagedRulesSQLiRuleSet" vendor\_name = "AWS" } }  
 visibility\_config { cloudwatch\_metrics\_enabled = true metric\_name = "sqli" sampled\_requests\_enabled = true }  
 }  
 rule {  
 name = "AWS-AWSManagedRulesAmazonIpReputationList"  
 priority = 4  
 statement { managed\_rule\_group\_statement { name = "AWSManagedRulesAmazonIpReputationList" vendor\_name = "AWS" } }  
 visibility\_config { cloudwatch\_metrics\_enabled = true metric\_name = "iprep" sampled\_requests\_enabled = true }  
 }  
  
 visibility\_config {  
 cloudwatch\_metrics\_enabled = true  
 metric\_name = "httpapi-waf"  
 sampled\_requests\_enabled = true  
 }  
 tags = local.tags  
}  
  
# Associate WAF to API Gateway HTTP API $default stage  
resource "aws\_wafv2\_web\_acl\_association" "httpapi\_assoc" {  
 count = var.enable\_waf ? 1 : 0  
 resource\_arn = "arn:aws:apigateway:${var.region}::/apis/${aws\_apigatewayv2\_api.http.id}/stages/${aws\_apigatewayv2\_stage.default.name}"  
 web\_acl\_arn = aws\_wafv2\_web\_acl.httpapi.arn  
}

### DNS (internal) — optional helper dns.tf

If you manage a private domain in Route 53, map your ALB to a friendly name used in the cert SANs:

variable "private\_zone\_id" { type = string }  
variable "alb\_private\_fqdn" { type = string } # e.g., "identity.svc.internal.example"  
  
resource "aws\_route53\_record" "alb\_alias" {  
 zone\_id = var.private\_zone\_id  
 name = var.alb\_private\_fqdn  
 type = "A"  
 alias {  
 name = aws\_lb.internal.dns\_name  
 zone\_id = aws\_lb.internal.zone\_id  
 evaluate\_target\_health = true  
 }  
}

### Apply Notes

1. Import/provision your ACM cert in **af-south-1** and set -var alb\_cert\_arn=....
2. Ensure the certificate SAN matches the hostname you’ll use internally (and that API Gateway over VPC Link can resolve it if needed; ALB validation uses SNI on the listener).
3. terraform apply — ECS/ALB will briefly recycle to add the HTTPS listener; API routes remain the same.

## 8) Advanced Protections — Rate Limiting, Bot Control, and mTLS

This section adds: (a) **WAFv2 rate limiting**, (b) **AWS Managed Bot Control**, and (c) **mutual TLS (mTLS)** for **API Gateway** via a **custom domain**.  
> Note: mTLS **on ALB** is not supported behind API Gateway **VPC Link** (API Gateway does not present a client certificate to your ALB). To achieve true client‑cert auth, terminate mTLS at **API Gateway custom domain** and keep the private hop to ALB on TLS server‑auth only.

### Variables — add to variables.tf

variable "waf\_rate\_limit" {  
 type = number  
 default = 2000 # requests per 5‑min window per IP  
 description = "WAFv2 rate limit per IP"  
}  
  
variable "enable\_bot\_control" {  
 type = bool  
 default = false # Bot Control incurs additional cost  
 description = "Enable AWS Managed Bot Control rule set"  
}  
  
variable "mtls\_enabled" {  
 type = bool  
 default = false  
}  
  
variable "mtls\_domain\_name" {  
 type = string  
 default = ""  
 description = "API custom domain for mTLS (e.g., id.example.com)"  
}  
  
variable "mtls\_hosted\_zone\_id" {  
 type = string  
 default = ""  
 description = "Route53 public or private hosted zone ID for mtls\_domain\_name"  
}  
  
variable "mtls\_cert\_arn" {  
 type = string  
 default = ""  
 description = "ACM cert ARN for the API custom domain (server cert)"  
}  
  
variable "mtls\_truststore\_s3\_uri" {  
 type = string  
 default = ""  
 description = "s3://bucket/key for client CA bundle (PEM)"  
}  
  
variable "mtls\_truststore\_version" {  
 type = string  
 default = ""  
 description = "S3 object version for the truststore"  
}

### WAFv2 — update waf.tf with Rate Limit + Bot Control

# ... existing aws\_wafv2\_web\_acl httpapi ...  
  
# Rate limiting rule  
rule {  
 name = "RateLimitPerIp"  
 priority = 5  
 statement {  
 rate\_based\_statement {  
 limit = var.waf\_rate\_limit  
 aggregate\_key\_type = "IP"  
 }  
 }  
 action { block {} }  
 visibility\_config {  
 cloudwatch\_metrics\_enabled = true  
 metric\_name = "ratelimit"  
 sampled\_requests\_enabled = true  
 }  
}  
  
# Optional: Managed Bot Control (chargeable)  
rule {  
 count = var.enable\_bot\_control ? 1 : 0  
 name = "AWS-AWSManagedRulesBotControlRuleSet"  
 priority = 6  
 statement {  
 managed\_rule\_group\_statement {  
 name = "AWSManagedRulesBotControlRuleSet"  
 vendor\_name = "AWS"  
 }  
 }  
 override\_action { none {} } # start in count mode by using override\_action if you prefer; change to block {} to enforce  
 visibility\_config {  
 cloudwatch\_metrics\_enabled = true  
 metric\_name = "botcontrol"  
 sampled\_requests\_enabled = true  
 }  
}

Tip: Begin Bot Control in **count/override** mode to monitor impact, then switch to action { block {} } once you’re confident.

### API Gateway mTLS — new file apigw\_mtls.tf

This creates a **custom domain** with mTLS and maps it to the existing HTTP API stage. Clients must present a certificate chaining to the trust store you provide in S3.

# Only create when enabled  
resource "aws\_apigatewayv2\_domain\_name" "mtls" {  
 count = var.mtls\_enabled ? 1 : 0  
 domain\_name = var.mtls\_domain\_name  
  
 domain\_name\_configuration {  
 certificate\_arn = var.mtls\_cert\_arn  
 endpoint\_type = "REGIONAL"  
 security\_policy = "TLS\_1\_2"  
 }  
  
 mutual\_tls\_authentication {  
 truststore\_uri = var.mtls\_truststore\_s3\_uri  
 truststore\_version = var.mtls\_truststore\_version  
 }  
  
 tags = local.tags  
}  
  
resource "aws\_apigatewayv2\_api\_mapping" "mtls" {  
 count = var.mtls\_enabled ? 1 : 0  
 api\_id = aws\_apigatewayv2\_api.http.id  
 domain\_name = aws\_apigatewayv2\_domain\_name.mtls[0].id  
 stage = aws\_apigatewayv2\_stage.default.name  
}  
  
# Route53 record for the custom domain  
resource "aws\_route53\_record" "mtls" {  
 count = var.mtls\_enabled ? 1 : 0  
 zone\_id = var.mtls\_hosted\_zone\_id  
 name = var.mtls\_domain\_name  
 type = "A"  
 alias {  
 name = aws\_apigatewayv2\_domain\_name.mtls[0].domain\_name\_configuration[0].api\_gateway\_domain\_name  
 zone\_id = aws\_apigatewayv2\_domain\_name.mtls[0].domain\_name\_configuration[0].hosted\_zone\_id  
 evaluate\_target\_health = false  
 }  
}

### Client Certificates for mTLS

* Build a client cert that chains to the CA bundle in your **S3 truststore** (PEM).
* Example curl test once DNS resolves to your custom domain:

curl https://id.example.com/identities \  
 --cert client\_cert.pem \  
 --key client\_key.pem \  
 -H 'Content-Type: application/json' \  
 -d '{"nationalId":"8001015009087","firstName":"Lloyd","lastName":"Moti"}'

### Apply Notes

1. Upload your **truststore** (PEM) to S3 and capture the **object version** (versioning must be enabled).
2. Ensure your **ACM cert** for mtls\_domain\_name is in **af-south-1**.
3. Enable and apply:

terraform apply \  
 -var="enable\_bot\_control=true" \  
 -var="waf\_rate\_limit=1500" \  
 -var="mtls\_enabled=true" \  
 -var="mtls\_domain\_name=id.example.com" \  
 -var="mtls\_hosted\_zone\_id=Z0123456789ABCDEF" \  
 -var="mtls\_cert\_arn=arn:aws:acm:af-south-1:123456789012:certificate/xxxx" \  
 -var="mtls\_truststore\_s3\_uri=s3://my-bucket/mtls/truststore.pem" \  
 -var="mtls\_truststore\_version=3L4k..."

## 9) WAF: IP Allow/Deny, Geo Blocks, and Header-Based API Key

This section adds configurable **IP allow/deny**, **country geo rules**, and an optional **header-based API key** check in **WAFv2** for your HTTP API stage.

⚠️ **Secret caution**: A header value embedded in Terraform is stored in state. If you must keep it secret, consider a low-risk token with frequent rotation, a separate state backend with tight access, or move to **API Gateway usage plans + API keys** or **Cognito/OIDC/JWT** for stronger auth. The WAF header rule here is useful as an additional filter.

### Variables — add to variables.tf

variable "ip\_allow\_cidrs" { type = list(string) default = [] }  
variable "ip\_deny\_cidrs" { type = list(string) default = [] }  
  
variable "geo\_allow\_countries" { # ISO 3166-1 alpha-2 (e.g., ["ZA","NA"])   
 type = list(string)  
 default = []  
}  
variable "geo\_block\_countries" {  
 type = list(string)  
 default = []  
}  
  
variable "require\_api\_key\_header" { type = bool default = false }  
variable "api\_key\_header\_name" { type = string default = "x-api-key" }  
variable "api\_key\_header\_value" { type = string default = "" } # WARNING: stored in TF state

### WAFv2 IP Sets (optional) — add to waf.tf

# IP allowlist  
resource "aws\_wafv2\_ip\_set" "allow" {  
 count = length(var.ip\_allow\_cidrs) > 0 ? 1 : 0  
 name = "${var.project\_name}-waf-ip-allow"  
 scope = "REGIONAL"  
 ip\_address\_version = "IPV4"  
 addresses = var.ip\_allow\_cidrs  
 tags = local.tags  
}  
  
# IP denylist  
resource "aws\_wafv2\_ip\_set" "deny" {  
 count = length(var.ip\_deny\_cidrs) > 0 ? 1 : 0  
 name = "${var.project\_name}-waf-ip-deny"  
 scope = "REGIONAL"  
 ip\_address\_version = "IPV4"  
 addresses = var.ip\_deny\_cidrs  
 tags = local.tags  
}

### WAFv2 Rules — extend existing aws\_wafv2\_web\_acl.httpapi in waf.tf

Keep priorities unique and consistent with prior rules. The snippet below assumes earlier rules used priorities 1..6. Adjust if you changed them.

# 0) Block explicit IP denylist (if any)  
rule {  
 name = "IpDenyList"  
 priority = 7  
 statement {  
 ip\_set\_reference\_statement { arn = length(var.ip\_deny\_cidrs) > 0 ? aws\_wafv2\_ip\_set.deny[0].arn : "arn:aws:wafv2:${var.region}:000000000000:global/ipset/empty" }  
 }  
 action { block {} }  
 visibility\_config { cloudwatch\_metrics\_enabled = true metric\_name = "ipdeny" sampled\_requests\_enabled = true }  
 dynamic "rule\_label" {  
 for\_each = length(var.ip\_deny\_cidrs) > 0 ? [1] : []  
 content { name = "ipdeny" }  
 }  
}  
  
# 1) Enforce IP allowlist by blocking NOT in allowlist (if provided)  
rule {  
 name = "IpAllowListEnforce"  
 priority = 8  
 statement {  
 not\_statement {  
 statement {  
 ip\_set\_reference\_statement { arn = length(var.ip\_allow\_cidrs) > 0 ? aws\_wafv2\_ip\_set.allow[0].arn : "arn:aws:wafv2:${var.region}:000000000000:global/ipset/empty" }  
 }  
 }  
 }  
 action { block {} }  
 visibility\_config { cloudwatch\_metrics\_enabled = true metric\_name = "ipallow\_enforce" sampled\_requests\_enabled = true }  
 dynamic "rule\_label" {  
 for\_each = length(var.ip\_allow\_cidrs) > 0 ? [1] : []  
 content { name = "ipallow\_enforce" }  
 }  
}  
  
# 2) Block geo countries (if any)  
rule {  
 name = "GeoBlock"  
 priority = 9  
 statement { geo\_match\_statement { country\_codes = var.geo\_block\_countries } }  
 action { block {} }  
 visibility\_config { cloudwatch\_metrics\_enabled = true metric\_name = "geoblock" sampled\_requests\_enabled = true }  
}  
  
# 3) Enforce geo allowlist: block NOT in allow list (if provided)  
rule {  
 name = "GeoAllowEnforce"  
 priority = 10  
 statement {  
 not\_statement { statement { geo\_match\_statement { country\_codes = var.geo\_allow\_countries } } }  
 }  
 action { block {} }  
 visibility\_config { cloudwatch\_metrics\_enabled = true metric\_name = "geoallow\_enforce" sampled\_requests\_enabled = true }  
}  
  
# 4) Header-based API key (block when header missing or wrong), only if enabled  
rule {  
 name = "HeaderApiKeyRequired"  
 priority = 11  
 statement {  
 or\_statement {  
 statement {  
 not\_statement {  
 statement {  
 byte\_match\_statement {  
 positional\_constraint = "EXACTLY"  
 search\_string = var.api\_key\_header\_value  
 field\_to\_match { single\_header { name = lower(var.api\_key\_header\_name) } }  
 text\_transformations { priority = 0 type = "NONE" }  
 }  
 }  
 }  
 }  
 statement {  
 byte\_match\_statement {  
 positional\_constraint = "EXACTLY"  
 search\_string = ""  
 field\_to\_match { single\_header { name = lower(var.api\_key\_header\_name) } }  
 text\_transformations { priority = 0 type = "NONE" }  
 }  
 }  
 }  
 }  
 action { block {} }  
 visibility\_config { cloudwatch\_metrics\_enabled = true metric\_name = "header\_apikey" sampled\_requests\_enabled = true }  
}

**Header name must be lowercase** in WAF. The rule above blocks when the header is **missing** or **doesn’t equal** the configured value.

### Example: lock to South Africa + Namibia, deny a CIDR, and require header key

ip\_allow\_cidrs = []  
ip\_deny\_cidrs = ["203.0.113.0/24"]  
geo\_allow\_countries = ["ZA","NA"]  
require\_api\_key\_header = true  
api\_key\_header\_name = "x-internal-key"  
api\_key\_header\_value = "change-me-rotate-often"

### Apply

terraform apply \  
 -var='ip\_deny\_cidrs=["203.0.113.0/24"]' \  
 -var='geo\_allow\_countries=["ZA","NA"]' \  
 -var='require\_api\_key\_header=true' \  
 -var='api\_key\_header\_name=x-internal-key' \  
 -var='api\_key\_header\_value=change-me-rotate-often'

## 10) JWT (OIDC) Authorizer on HTTP API + (Optional) Usage Plans via REST API Front Door

**Important distinction**: - **HTTP API (v2)** supports **JWT/OIDC authorizers** natively but **does not support Usage Plans or API Keys**. - **REST API (v1)** supports **Usage Plans + API Keys** but **does not** have native OIDC JWT; you use **Cognito User Pools** or a **Lambda authorizer**.

Below we (A) add **JWT (OIDC)** to your existing **HTTP API** and (B) optionally create a **REST API front door** solely to enforce **Usage Plans** while proxying to the same internal ALB via **VPC Link (v1)**.

### (A) JWT (OIDC) for HTTP API — update variables.tf

variable "jwt\_issuer" { type = string default = "" } # e.g., https://issuer.example.com  
variable "jwt\_audience" { type = list(string) default = [] } # e.g., ["identity-api"]  
variable "jwt\_required" { type = bool default = true }

### Update apigw\_vpclink.tf — add JWT authorizer & secure routes

resource "aws\_apigatewayv2\_authorizer" "jwt" {  
 count = var.jwt\_required ? 1 : 0  
 api\_id = aws\_apigatewayv2\_api.http.id  
 authorizer\_type = "JWT"  
 identity\_sources = ["$request.header.Authorization"]  
 name = "oidc-jwt"  
 jwt\_configuration {  
 issuer = var.jwt\_issuer  
 audience = var.jwt\_audience  
 }  
}  
  
# Update routes to require JWT  
resource "aws\_apigatewayv2\_route" "identities\_post" {  
 api\_id = aws\_apigatewayv2\_api.http.id  
 route\_key = "POST /identities"  
 target = "integrations/${aws\_apigatewayv2\_integration.alb.id}"  
 authorization\_type = var.jwt\_required ? "JWT" : "NONE"  
 authorizer\_id = var.jwt\_required ? aws\_apigatewayv2\_authorizer.jwt[0].id : null  
}  
  
resource "aws\_apigatewayv2\_route" "identities\_batch" {  
 api\_id = aws\_apigatewayv2\_api.http.id  
 route\_key = "POST /identities/batch"  
 target = "integrations/${aws\_apigatewayv2\_integration.alb.id}"  
 authorization\_type = var.jwt\_required ? "JWT" : "NONE"  
 authorizer\_id = var.jwt\_required ? aws\_apigatewayv2\_authorizer.jwt[0].id : null  
}  
  
resource "aws\_apigatewayv2\_route" "identities\_get" {  
 api\_id = aws\_apigatewayv2\_api.http.id  
 route\_key = "GET /identities/{id}"  
 target = "integrations/${aws\_apigatewayv2\_integration.alb.id}"  
 authorization\_type = var.jwt\_required ? "JWT" : "NONE"  
 authorizer\_id = var.jwt\_required ? aws\_apigatewayv2\_authorizer.jwt[0].id : null  
}

### (B) REST API (v1) with Usage Plans + API Keys (Optional) — new apigw\_rest\_usageplan.tf

This creates a minimal **REST API** front door that enforces an **API key** and **usage plan** while proxying to your **internal ALB** over a **v1 VPC Link**.

# VPC Link (REST API v1)  
resource "aws\_api\_gateway\_vpc\_link" "v1" {  
 name = "${var.project\_name}-v1-vpclink"  
 target\_arns = [aws\_lb.internal.arn]  
 tags = local.tags  
}  
  
resource "aws\_api\_gateway\_rest\_api" "rest" {  
 name = "${var.project\_name}-rest-frontend"  
 description = "REST API with API keys/usage plans forwarding to internal ALB"  
}  
  
resource "aws\_api\_gateway\_resource" "root\_proxy" {  
 rest\_api\_id = aws\_api\_gateway\_rest\_api.rest.id  
 parent\_id = aws\_api\_gateway\_rest\_api.rest.root\_resource\_id  
 path\_part = "{proxy+}"  
}  
  
resource "aws\_api\_gateway\_method" "any\_proxy" {  
 rest\_api\_id = aws\_api\_gateway\_rest\_api.rest.id  
 resource\_id = aws\_api\_gateway\_resource.root\_proxy.id  
 http\_method = "ANY"  
 authorization = "NONE"  
 api\_key\_required = true  
}  
  
resource "aws\_api\_gateway\_integration" "alb\_proxy" {  
 rest\_api\_id = aws\_api\_gateway\_rest\_api.rest.id  
 resource\_id = aws\_api\_gateway\_resource.root\_proxy.id  
 http\_method = aws\_api\_gateway\_method.any\_proxy.http\_method  
 type = "HTTP\_PROXY"  
 integration\_http\_method = "ANY"  
 uri = "https://${aws\_lb.internal.dns\_name}/{proxy}"  
 connection\_type = "VPC\_LINK"  
 connection\_id = aws\_api\_gateway\_vpc\_link.v1.id  
}  
  
resource "aws\_api\_gateway\_deployment" "rest" {  
 rest\_api\_id = aws\_api\_gateway\_rest\_api.rest.id  
 triggers = { redeploy = timestamp() }  
 lifecycle { create\_before\_destroy = true }  
}  
  
resource "aws\_api\_gateway\_stage" "prod" {  
 rest\_api\_id = aws\_api\_gateway\_rest\_api.rest.id  
 deployment\_id = aws\_api\_gateway\_deployment.rest.id  
 stage\_name = "prod"  
 xray\_tracing\_enabled = true  
}  
  
# Usage Plan & API Key  
resource "aws\_api\_gateway\_usage\_plan" "plan" {  
 name = "${var.project\_name}-plan"  
 api\_stages { api\_id = aws\_api\_gateway\_rest\_api.rest.id stage = aws\_api\_gateway\_stage.prod.stage\_name }  
 throttle { burst\_limit = 200 rate\_limit = 100 }  
 quota { limit = 100000 period = "MONTH" }  
}  
  
resource "aws\_api\_gateway\_api\_key" "consumer" {  
 name = "${var.project\_name}-consumer"  
 enabled = true  
}  
  
resource "aws\_api\_gateway\_usage\_plan\_key" "bind" {  
 key\_id = aws\_api\_gateway\_api\_key.consumer.id  
 key\_type = "API\_KEY"  
 usage\_plan\_id = aws\_api\_gateway\_usage\_plan.plan.id  
}  
  
output "rest\_api\_invoke\_url" {  
 value = "https://${aws\_api\_gateway\_rest\_api.rest.id}.execute-api.${var.region}.amazonaws.com/${aws\_api\_gateway\_stage.prod.stage\_name}"  
}

**Pick your front door:** - If you only need **JWT/OIDC** and low latency → stick with **HTTP API** (kept from earlier sections). - If you need **Usage Plans/API keys** for partners → expose the **REST API** invoke URL (or map a custom domain). Both can coexist temporarily during migration.

### Example Apply — JWT + REST Usage Plan

terraform apply \  
 -var='jwt\_required=true' \  
 -var='jwt\_issuer=https://issuer.example.com' \  
 -var='jwt\_audience=["identity-api"]'  
# For REST usage plan, simply include the new tf file; the output will show rest\_api\_invoke\_url.

### Done ✅

* HTTP API is now protected by **JWT (OIDC)**.
* Optional REST API front door adds **API Keys + Usage Plans** without changing your backend. Map DNS to whichever entrypoint you prefer (or both during cutover).