ENGINEERING REFERENCE ARCHITECTURE & CASE STUDY

Production Cloud Architecture & Serverless Containerization for B2B SaaS Licensing

A Defense-in-Depth AWS Fargate & PostgreSQL Deployment with Cost-Capped Security Controls

Client / System: KeyPulse Systems

Architected By: New Paradigm Systems (NPSys)

Cloud Provider & Region: AWS — `us-west-2` (Oregon)

Compute & Runtime: AWS ECS Fargate on Graviton (ARM64)

Database Engine: Amazon RDS PostgreSQL 16

Deployment Framework: Hardened OCI Containers & Parameter Store

---

1. Executive Summary

KeyPulse Systems developed a core B2B customer licensing engine engineered to issue cryptographically signed license keys, process billing webhooks, and provide programmatic validation APIs for enterprise client deployments. Following successful verification in local development containers, KeyPulse engaged New Paradigm Systems (NPSys) to design, commission, and validate a secure, highly available, and cost-controlled cloud hosting architecture on Amazon Web Services (AWS).

NPSys transitioned the application from an unmanaged local container environment into an enterprise-grade AWS cloud topology. The solution features managed serverless compute on AWS ECS Fargate (AWS Graviton ARM64), network micro-segmentation across public and isolated database subnets, an internet-facing Application Load Balancer with native automated health probing, zero-plaintext credential injection through AWS Systems Manager Parameter Store, and proactive budget guardrails that enforce zero unnecessary infrastructure expenditure.

2. Architectural Challenges & Strategic Objectives

Prior to engagement, the KeyPulse platform exhibited common risks typical of early-stage SaaS platforms migrating out of local testing:

- **Unmanaged Container Runtimes:** Local Docker setups executed containers as the `root` user without explicit health probe instrumentation, non-blocking I/O configuration, or compatibility testing for target cloud hypervisors.
- **Credential Exposure Risks:** Sensitive assets—including database master passwords, JWT cryptographic signing keys, and webhook verification tokens—resided in plaintext configuration files on local developer storage.
- **Network Layer Vulnerability:** The persistence layer lacked network isolation, leaving transactional database tables exposed to broader network routing without zero-trust microsegmentation.
- **Uncontrolled Cloud Cost Exposure:** Standard multi-tier cloud architectures frequently incur unintentional baseline fees (such as managed NAT Gateways at ~$65–$70/month), which would violate the client's strict startup operational budget.

3. Cloud Architecture & Network Topology

NPSys architected a high-security, dual-Availability Zone topology in `us-west-2` designed to deliver high availability, deterministic traffic routing, and a minimal cost footprint:

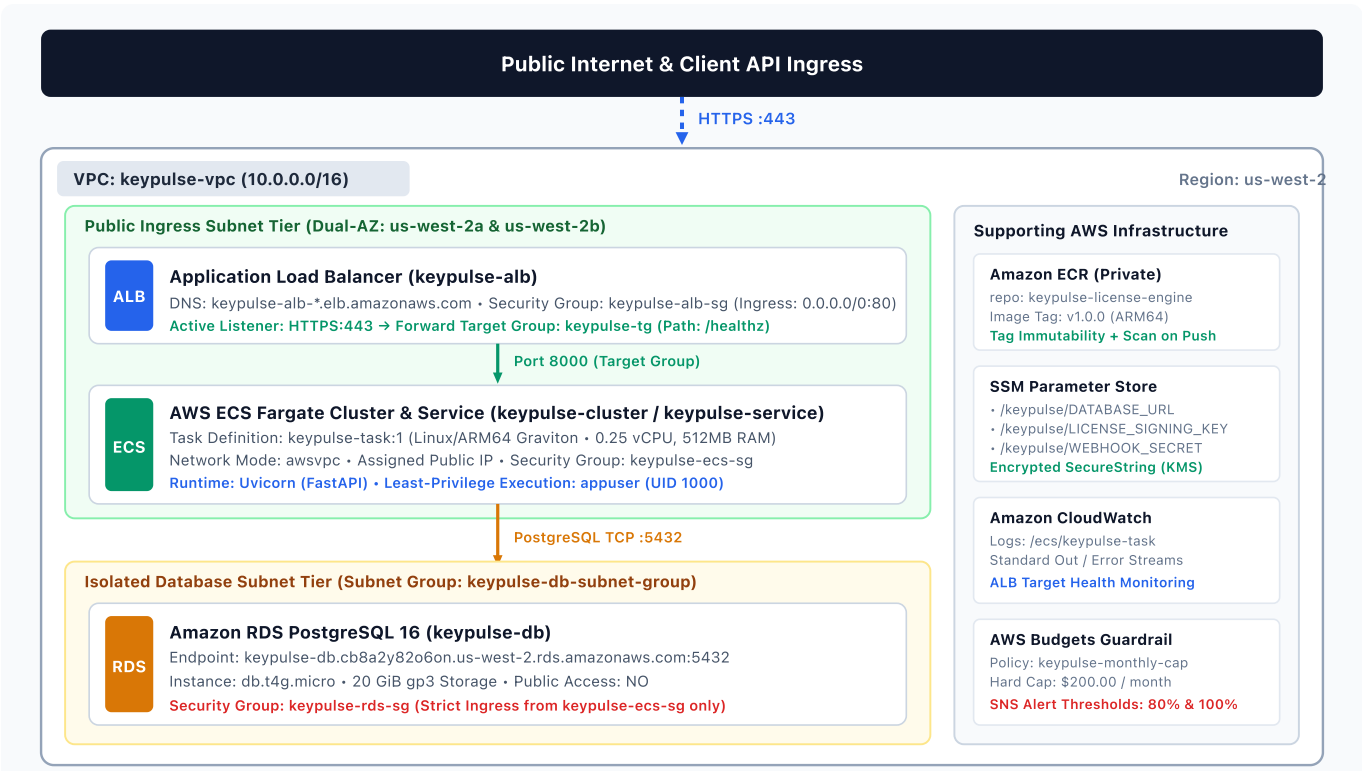

FIGURE 1: COMPLETE AWS NETWORK TOPOLOGY, CONTAINER ROUTING & SUPPORTING SERVICES

Component Technical Breakdown

- **Serverless Compute (AWS ECS Fargate):** Container tasks are executed across independent Availability Zones using AWS Fargate serverless infrastructure on ARM64 Graviton architecture. Graviton instances provide approximately 20% cost efficiency over x86 counterparts while matching local Apple Silicon development builds bit-for-bit.
- **Traffic Distribution (Application Load Balancer):** The public ALB handles ingress HTTPS traffic on port 443 and forwards requests to target group `keypulse-tg`. The ALB runs automated health checks against `GET /healthz` every 15 seconds, automatically deregistering any container task that fails to query the database.
- **Managed Relational Storage (Amazon RDS PostgreSQL 16):** Provisioned within private database subnets on a `db.t4g.micro` instance with 20 GiB of General Purpose SSD storage. Public routing is permanently disabled.
- **Zero-NAT Network Topology:** By provisioning ECS tasks with direct public IP capability inside public subnets while isolating incoming traffic via security groups, NPSys eliminated the need for managed NAT Gateways, immediately saving ~$65–$70/month in baseline networking fees.

4. Security Architecture & Governance Controls

The architecture implements a multi-tier defense model enforcing least privilege across container runtimes, IAM policies, and network communication:

Security Domain	Technical Safeguard Implementation	Impact & Compliance Result
Container Hardening	Multi-stage container build based on `python:3.11-slim`; unbuffered logging enabled; runs under dedicated non-root user `appuser` (UID 1000).	Mitigates container-breakout risks and host privilege escalation vulnerabilities.
Chained Security Groups	• `keypulse-alb-sg`: Ingress HTTPS:443 from `0.0.0.0/0`. • `keypulse-ecs-sg`: Ingress TCP:8000 restricted to `keypulse-alb-sg`. • `keypulse-rds-sg`: Ingress TCP:5432 restricted to `keypulse-ecs-sg`.	Establishes strict network microsegmentation. Database and containers are inaccessible to unauthorized external IP traffic.
Centralized Secrets	Database URLs, JWT signing tokens, and webhook secrets stored as `SecureString` parameters in AWS SSM Parameter Store. Injected in-memory at boot.	Zero credentials stored in version control, Docker images, or plaintext disk files.
IAM Least Privilege	ECS execution role attached with an inline policy granting `ssm:GetParameters` restricted specifically to the ARN resource `arn:aws:ssm:us-west-2:*:parameter/keypulse/*`.	Compute tasks cannot read unauthorized parameters or access external AWS services.
Registry Governance	Amazon ECR configured with **Tag Immutability** and **Scan on Push**. Continuous CVE reporting enabled.	Guarantees deterministic rollbacks, prevents image tag poisoning, and alerts on software CVEs.
Cost Guardrail	AWS Budgets configured with an absolute $200.00/month cap triggering multi-stage email notifications at 80% and 100% spend.	Guarantees protection against billing spikes or runaway resource consumption.

5. Implementation Roadmap & Execution Gates

NPSys structured the engagement across four discrete, verifiable milestones to ensure continuous stability:

Phase 1: Local Code Hygiene, Sanitization & Container Hardening

Purged development artifacts (virtual environments, test caches, OS metadata, local `.env` files). Constructed a production-grade `.gitignore`, initialized Git version control on branch `main`, and refactored the `Dockerfile` to execute as a non-root system user. Verified container health locally via Docker Compose against the `/healthz` database probe.

Phase 2: Cloud Foundations & Container Registry Commissioning

Established regional identity in `us-west-2`, provisioned the $200 AWS Budget alert, and created private ECR repository `keypulse-license-engine`. Authenticated Docker using temporary STS tokens and pushed the native ARM64 production image tagged `v1.0.0`.

Phase 3: Network Infrastructure, Persistence & Secrets Management

Provisioned `keypulse-vpc` across two Availability Zones with dedicated public and isolated private subnets. Configured chained security groups, deployed Amazon RDS PostgreSQL 16, and encrypted all sensitive application secrets in AWS SSM Parameter Store.

Phase 4: Serverless Orchestration & End-to-End Validation

Configured the ECS task definition (`keypulse-task:1`) with SSM secret injections and IAM execution role permissions. Launched `keypulse-service` on AWS Fargate attached to the Application Load Balancer. Verified automated target group health transition to `Healthy`, and executed live validation tests confirming successful database connectivity and administrative portal response.

6. Business Value & Engineering Outcomes

- **Fully Serverless Operations:** Zero EC2 instances to manage, patch, or maintain. Operating on AWS Fargate eliminates server administration overhead and operating system vulnerabilities.
- **Enterprise Credential Hygiene:** Completely decoupled application configuration from secrets, meeting enterprise software procurement security baselines.
- **Cost-Optimized Architecture:** By eliminating managed NAT Gateway infrastructure while maintaining strict security group firewalls, the deployment operates safely inside the client's budget framework with projected ongoing costs under $50.00/month.
- **Native Architecture Parity:** Running native ARM64 containers in production that match developer Apple Silicon machines eliminates cross-architecture emulation bugs and runtime performance regressions.
- **Production Readiness:** Centralized CloudWatch logging, container health metrics, and automated load balancer rerouting provide the client with immediate commercial deployment readiness.