Company Inc. — AWS Architecture Design

AWS cloud provider was chosen due to a variety of available services and better expertise, but a pretty similar setup could be achieved within any other major cloud provider (Google Cloud or Azure).

Cloud Organizations & Accounts:

- Management – payer account, IAM Identity Center (SSO), Organizations, SCPs.
- Security & Logs – centralized CloudTrail, Config, S3 log buckets with KMS and lifecycle/Glacier.
- Services – CI/CD runners, ECR, Route 53.
- QA and Prod (at least) – workload accounts with their own VPCs/EKS clusters and quotas.

Network Design

- VPC per environment, split across at least 2 AZs:
- 2 Public subnets (per-AZ) for ALB and NAT Gateways / 2 Private subnets (per-AZ) for EKS node groups and pods (no public IPs).
- add VPC Endpoints (Gateway/Interface) for as many used services as we can (S3, ECR, STS, CloudWatch, Secrets Manager/SSM) to reduce NAT costs.
- Transit Gateway for VPN access or SSM Session Manager for better security
- Traffic goes CloudFront (SPA) → AWS WAF → ALB (ingress) → EKS services.

EKS

- EKS control plane private endpoint
- Karpenter pools
  1. On-Demand System nodes
  2. Application nodes (75% Spot + 25% On-Demand) for web/API
- Enable horizontal Autoscaling based on CPU/Mem and Karpenter for node scale-out
- Immutable deploys for critical services and spread load across AZs
- AWS Load Balancer Controller

Security & Monitoring (on EKS and VPC level)

- Security Groups: least privilege, widely use TLS encryption
- IRSA (IAM Roles for Service Accounts)
- Secrets from AWS Secrets Manager/SSM Parameter Store
- Kyverno policies and Kubernetes NetworkPolicies
- CloudWatch for AWS metrics
- Prometheus/Grafana stack
- OpenTelemetry

Containerization & CI/CD

- Use for image builds multi-stage Dockerfiles (Node build for React → NGINX dist; Python/Flask slim base), pinned versions, non-root user, distroless where possible.
- Push to private Amazon ECR, enable scan-on-push, immutable tags, lifecycle rules.
- Pipelines – GitHub Actions or CodePipeline
    1. Lint/test → SCA/SAST → build & push to ECR
    2. Helm chart render → sign → deploy to Dev/QA → progressive delivery (staging, then prod) with blue/green or canary
- use Terraform for all infra; Helm for Kubernetes.

Database (MongoDB)

- use MongoDB Atlas to retain the full MongoDB feature set, native drivers, and operational tooling. No public access must be allowed, use PrivateLink or VPC peering to the app VPC.
- HA – Replica set across 3 AZs in the chosen region (PSA architecture).
- Encryption enabled at rest (Atlas), in transit (TLS), and customer-managed KMS keys if required.
- Backups & retention – automated continuous backups with Point-in-Time Recovery; retention tiers (e.g., 7/30/90 days), scheduled logical backups for compliance exports to S3 (encrypted).
- Disaster recovery – start with cross-region snapshot replication; evolve to “Global Clusters” or cross-region replica for low RTO/RPO.
- Quarterly restore drills (tabletop + actual restore) and documented **RPO/RTO** targets (e.g., RPO ≤ 5 min, RTO ≤ 60 min as budget allows).
- Role-based database users; short-lived credentials injected via Secrets Manager
- IP/endpoint restrictions via PrivateLink; auditing enabled.

Cost Strategy (initial → scale)

- Start with 2 → 3 AZs, small Graviton instances, Spot for burst.
- Consolidate NAT early (1×), move to per-AZ NAT as HA needs rise.
- Prefer VPC Endpoints to cut NAT egress.
- Rightsize via Karpenter, adopt “Savings Plans” once steady state emerges.
- CloudFront caching to reduce origin/API load; aggressive SPA caching with cache-busting on deploy.