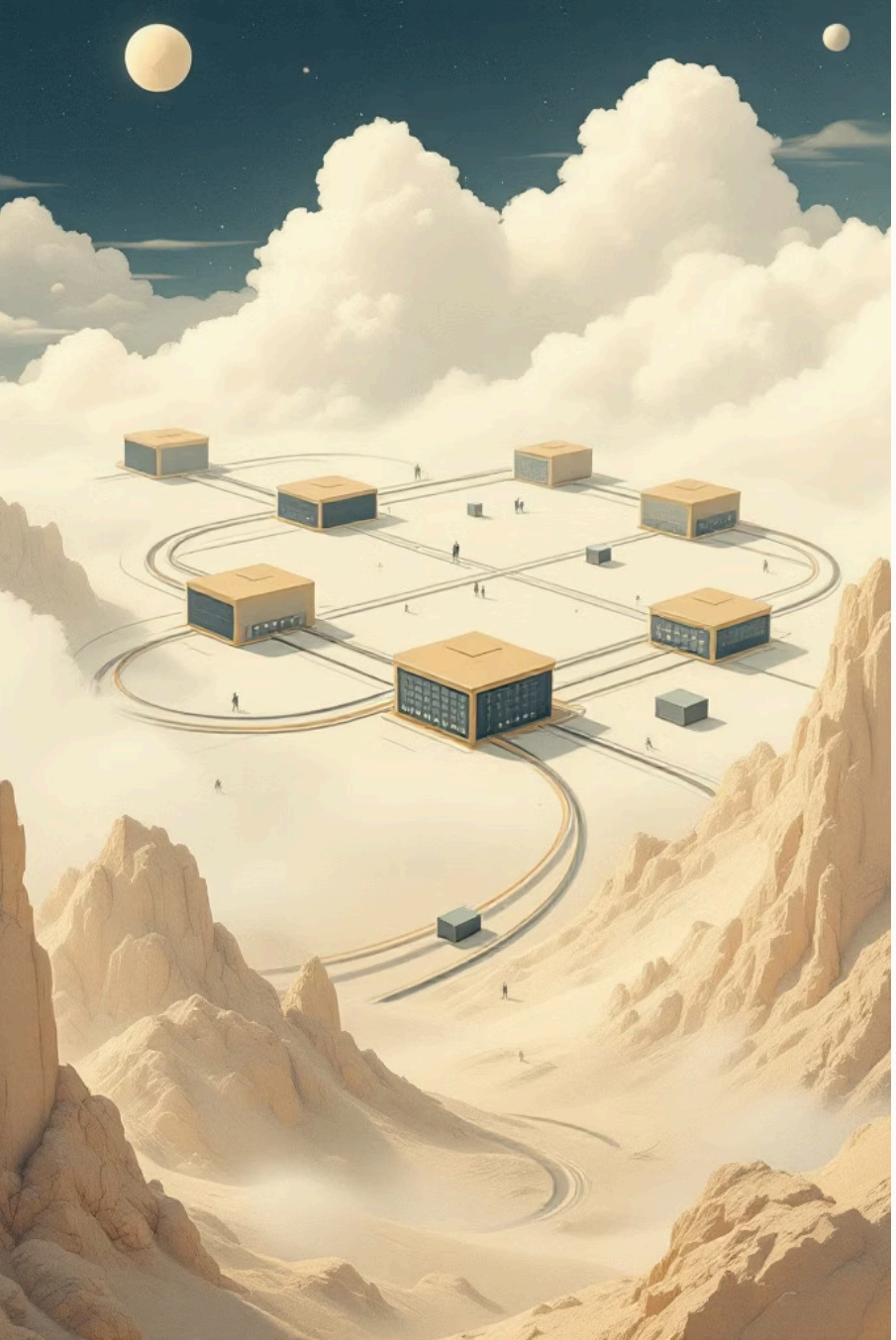
Three-Tier Architecture on AWS

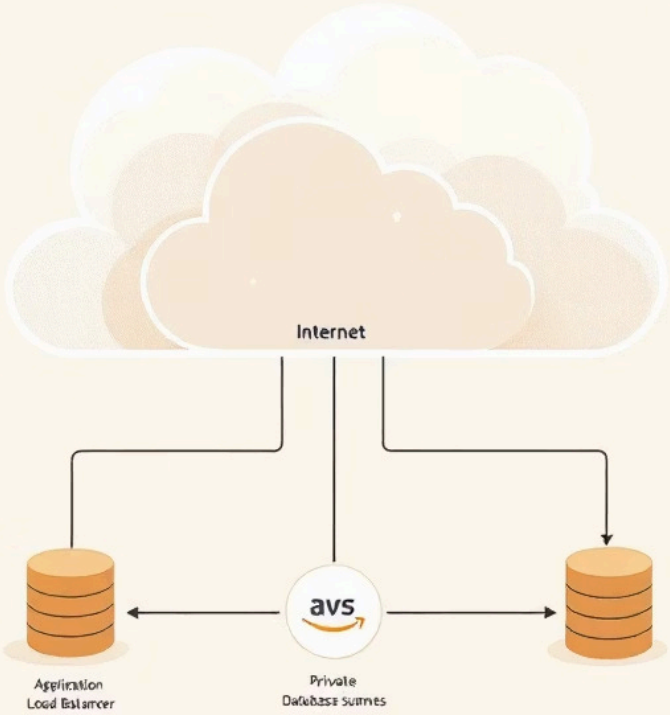


Three-Tier Architecture on AWS — Production Blueprint

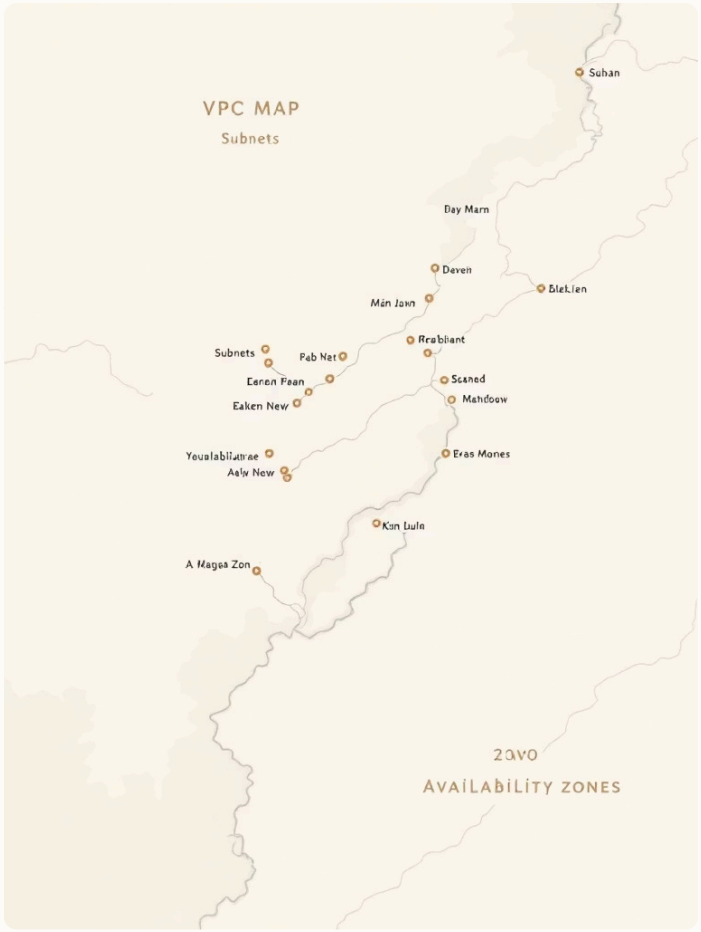
Concise technical guide for cloud engineers and architects. This deck explains a resilient, highly available three-tier web application on AWS: Presentation (ALB), Application (EC2 / Node.js), and Database (private subnets / RDS-ready).

The Big Picture — Logical Architecture

Traffic from the internet terminates at an internet-facing Application Load Balancer (ALB) in public subnets. ALB routes HTTP(S) to EC2 application servers in private app subnets across two Availability Zones. Database tier resides in private DB subnets (reserved for RDS).



VPC Layout & IP Plan

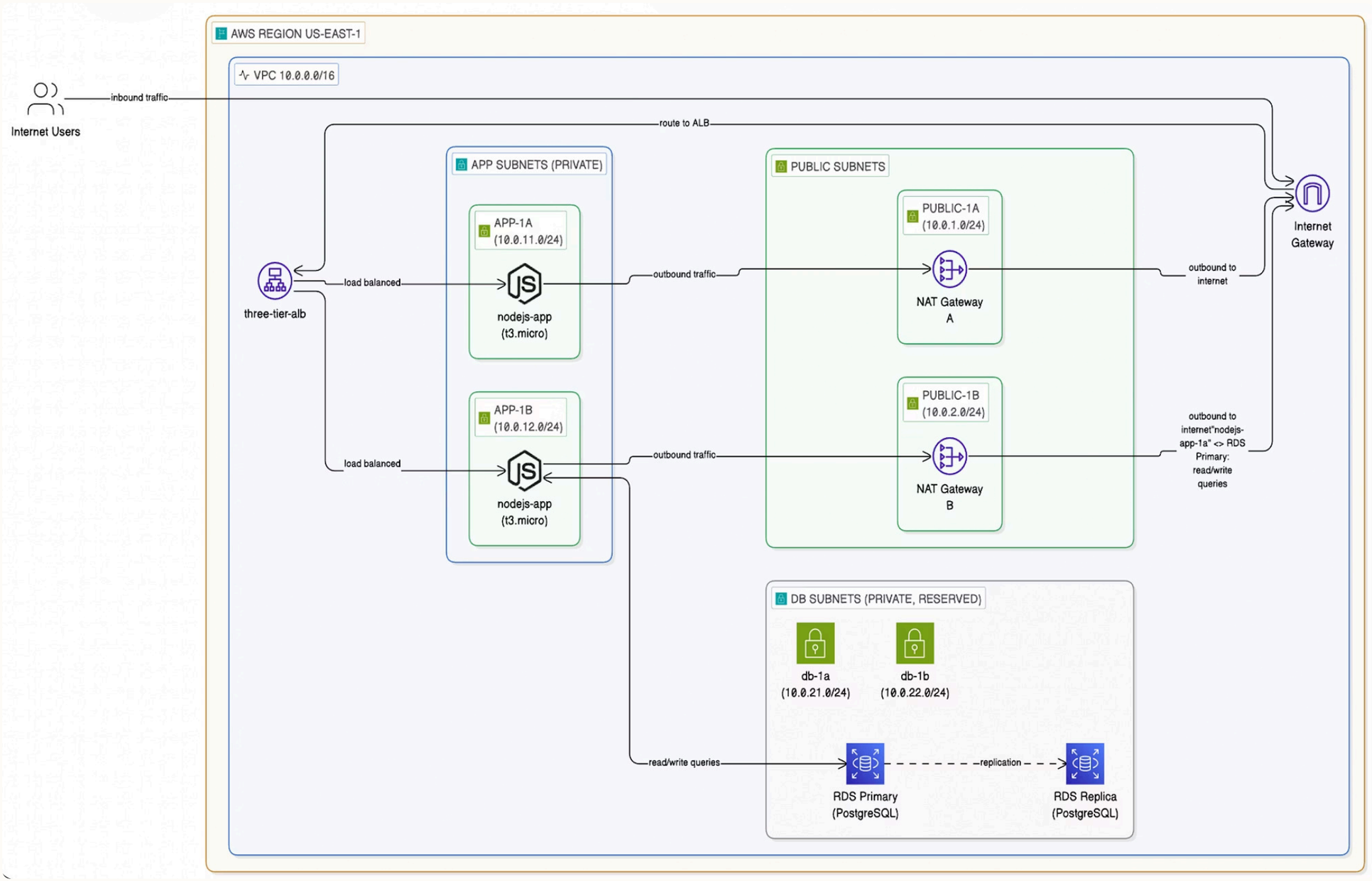


Network CIDR and Subnets

- VPC: 10.0.0.0/16 — 65,536 addresses
- Public subnets (Tier 1): 10.0.1.0/24 (1a), 10.0.2.0/24 (1b)
- Private app subnets (Tier 2): 10.0.11.0/24 (1a), 10.0.12.0/24 (1b)
- Private DB subnets (Tier 3): 10.0.21.0/24 (1a), 10.0.22.0/24 (1b)

Six subnets provide AZ-level isolation for each tier — resilient to single AZ failure.

Architecture



Traffic Flow — Step-by-Step

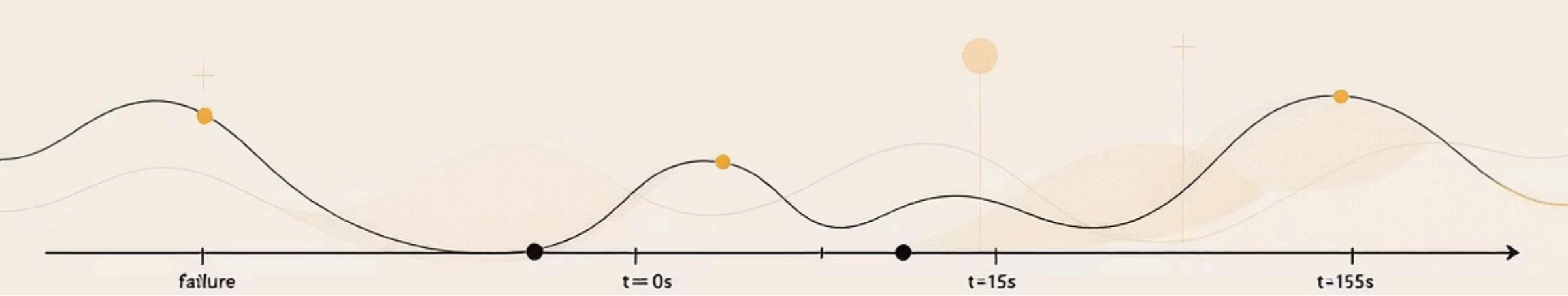
The ALB performs health checks (/health). It routes only to healthy targets using round-robin across AZs. The Node.js app serves HTML at “/” and JSON at “/health” for ALB monitoring.

Security — Defence in Depth

Security Groups (least privilege)

- alb-sg: Inbound 80/443 from 0.0.0.0/0, Outbound: all
- app-tier-sg: Inbound 80/443 only from alb-sg, Outbound: all (for NAT)
- data-tier-sg: Inbound 3306/5432 only from app-tier-sg, Outbound: all

EC2 instances are not internet-reachable. All cross-tier access is restricted to explicit SG references.



High Availability & Failover Behaviour

Normal: ALB → EC2-A and EC2-B healthy. If EC2-A fails, ALB health checks mark it unhealthy within ~60s and route all traffic to EC2-B. When EC2-A recovers and passes health checks, traffic is redistributed. RTO ≈ 60s; RPO = 0 for stateless services.

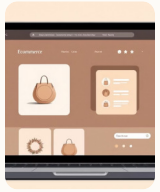
EC2 Bootstrap & Application Flow

userdata.sh (bootstrap)

- yum update -y
- Install NodeSource repo → yum install nodejs
- Fetch EC2 metadata (instance-id, availability-zone)
- Write server.js, start with nohup node server.js
- Self-check: curl localhost/health

Bootstrap automates dependency installation and service start; ensures instance is healthy before ALB directs traffic.

Real-World Use Cases

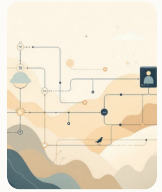


E-commerce

ALB handles traffic spikes (Black Friday). EC2 serves cart and business logic; RDS stores product and order data.

API Backend

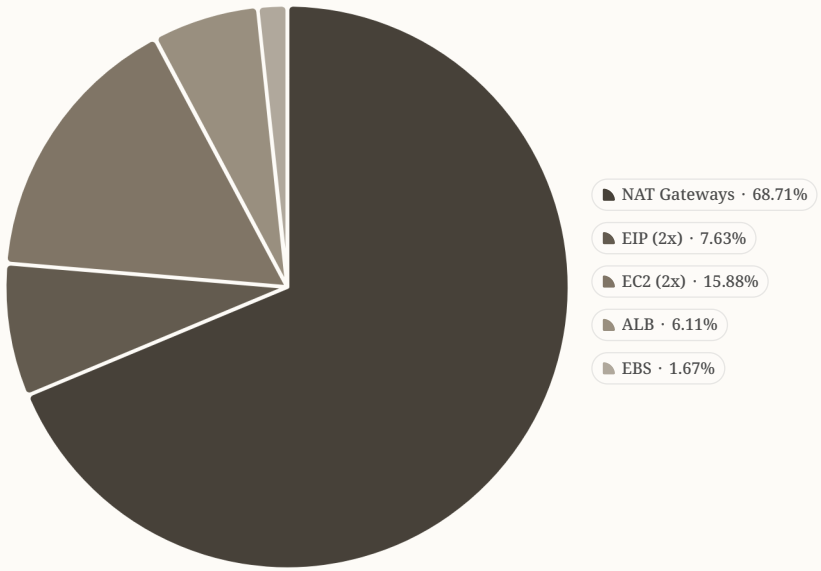
Mobile clients call REST APIs through ALB → EC2. Add ElastiCache for session and read performance.



Demo / Verification

Simple Node.js app returns instance and AZ: proves load balancing and health checks are configured correctly.

Cost Profile & Optimisation



Monthly estimate: ~US$96. Key cost driver: NAT Gateways (~68%). In development, consolidate to a single NAT Gateway to halve that portion. Other levers: smaller EC2 instance types, Spot instances, and reserve or use Savings Plans for steady workloads.

Production Roadmap & Key Takeaways

Security & Resilience

Enable HTTPS (ACM), add AWS WAF, and configure CloudFront for edge protection and caching.

Scalability

Replace fixed EC2s with an Auto Scaling Group (min 2 → max N), Health-checked by ALB.

Reliability & Observability

Deploy Multi-AZ RDS, CloudWatch Alarms & Dashboards, and structured logging/metrics.

Delivery & IaC

Implement CI/CD (CodePipeline) and author infrastructure in Terraform for repeatable, auditable deployments.

The reference architecture balances simplicity and production readiness: stateless app servers behind an ALB, strict security groups, AZ redundancy, and a clear roadmap to make the environment truly production-grade.