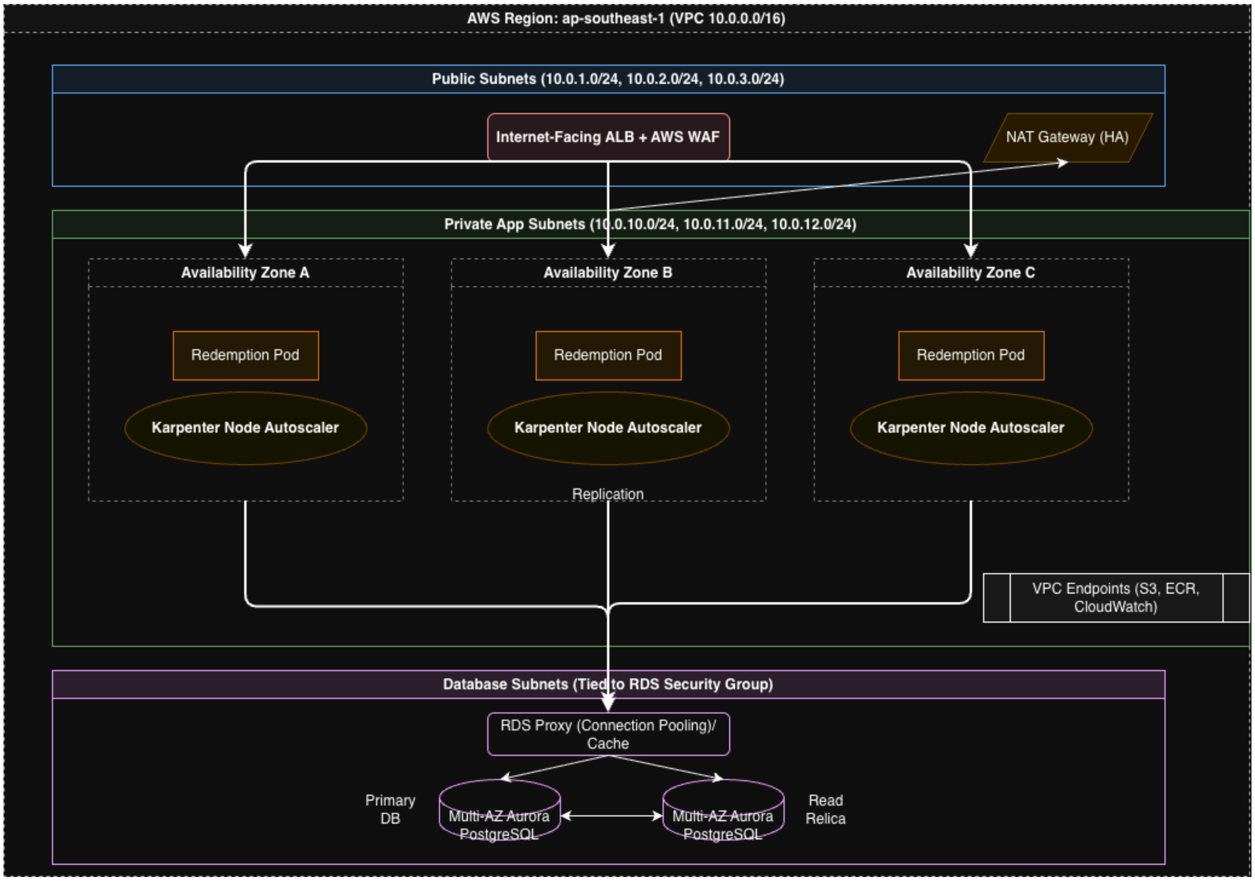
DESIGN DOCUMENT: "The Redemption" Microservice

1. Executive Summary

This document explains the architecture for "The Redemption" microservice. The main goal is zero downtime even when we have 10x traffic spikes during Flash Sales. We use a **Three-Tier Multi-AZ** setup on AWS EKS, focusing on automatic scaling and security to protect the revenue.

2. Architectural Decisions & Trade-offs


AWS Region: ap-southeast-1 (VPC 10.0.0.0/16)
Public Subnets (10.0.1.0/24, 10.0.2.0/24, 10.0.3.0/24)
Internet-Facing ALB + AWS WAF
NAT Gateway (HA)
Private App Subnets (10.0.10.0/24, 10.0.11.0/24, 10.0.12.0/24)
Availability Zone A
Availability Zone B
Availability Zone C
Redemption Pod
Redemption Pod
Redemption Pod
Karpenter Node Autoscaler
Karpenter Node Autoscaler
Karpenter Node Autoscaler
Replication
VPC Endpoints (S3, ECR, CloudWatch)
Database Subnets (Tied to RDS Security Group)
RDS Proxy (Connection Pooling)/ Cache
Primary DB
Multi-AZ Aurora PostgreSQL
Multi-AZ Aurora PostgreSQL
Read Relica

A. Compute: EKS with Karpenter

- **Decision:** I chose **Amazon EKS** using **Karpenter** for scaling the nodes. Standard Cluster Autoscaler is too slow for 10x spikes.
- **Trade-off:** Karpenter is a bit harder for setup than the default one, but it can start new EC2 instances in less than a minute. For Flash Sales, we cannot wait 5 minutes for a node to be ready.

- **High Availability:** Pods are spread across **3 Availability Zones** (AZ). I will use `podTopologySpreadConstraints` so if one AZ has a problem, the other two keep the service running.

B. Networking & Security

- **Isolation:** All the application and database are inside **Private Subnets**. Only the ALB is in the public subnet.
- **Least Privilege:** We use **IRSA** (IAM Roles for Service Accounts). This means the pod only has permission for what it needs. If a pod is compromised, the hacker cannot touch other parts of the AWS account.
- **Egress:** I added **NAT Gateways** so private nodes can get updates, and **VPC Endpoints** for S3/ECR so traffic stays inside the AWS network for better security.

C. Database: RDS Proxy

- **Decision:** We put **RDS Proxy** in front of our Aurora DB.
- **Why:** When we scale to 100s of pods during a spike, the database connections will hit the limit very fast. RDS Proxy manages the "connection pooling" so the database doesn't crash from too many connections.

3. Scalability Strategy

To handle 10x spikes without someone doing manual work:

1. **HPA (Horizontal Pod Autoscaler):** Scales the pods based on CPU and memory.
2. **Infrastructure Scaling:** Karpenter sees "Pending" pods and adds new nodes to the cluster immediately.
3. **Flash Sale Prep:** For scheduled events, we can use a small script to "pre-warm" the scaling min-size so we are ready before the traffic hits.

4. Reliability & Day 2 Ops

- **Self-Healing:** We use Liveness and Readiness probes. If a pod is stuck, Kubernetes will kill and restart it automatically.
- **Observability:** We will use **Prometheus and Grafana** for monitoring the "Golden Signals" (Latency, Traffic, Errors).
- **Bad Deployments:** Using **ArgoCD**. If a new version is bad, we can rollback to the old version with one click (or auto-rollback if error rate is high).

5. Team Task Assignment

Since we have 1 Senior and 2 Juniors, I divide the tasks like this:

Person	Task
Senior (Lead)	Infrastructure setup (VPC, EKS, IAM), Karpenter config, and the main Terraform modules.
Junior 1	Creating the K8s manifests (HPA, Deployment), and setting up the CI/CD pipeline in GitHub.
Junior 2	Setting up Monitoring (Grafana dashboards) and running Load Tests to check if 10x spike works.

6. Conclusion

This design is robust and focuses on automation. By using Karpenter and RDS Proxy, we handle the technical bottlenecks of "The Redemption" service. The team of 3 can manage this because most of the scaling is automatic.