AWS Cloud Solution Design for Chess Game App

Frontend (Next.js):
Use **CloudFront** for a CDN to serve the app globally with low latency. This will ensure that users can access the app with minimal delay, regardless of their geographic location.

Backend (NestJS):
Deploy using **ECS (Elastic Container Service)** with **Fargate** for a serverless, scalable approach. Fargate eliminates the need to manage servers and provides on-demand scaling, which is ideal for handling varying traffic loads.

Database (PostgreSQL):
Use **Amazon RDS** with **multi-AZ** for high availability and automatic backups. This ensures your data is secure and your application remains available even during an AZ failure.

Caching (Redis):
Use **ElastiCache for Redis** to store session data and manage task queues efficiently. This will reduce the load on the database and enhance the overall performance of the application.

WebSockets:
Deploy **WebSockets** with **Application Load Balancer (ALB)** for sticky sessions and scalability. ALB will distribute WebSocket connections effectively while maintaining session persistence, which is crucial for real-time interactions in the chess game.

Scaling & Cost Efficiency

ECS Fargate with **Auto-Scaling** to dynamically adjust the backend resources according to the traffic. This will handle traffic spikes and maintain a smooth user experience during high-demand periods.

RDS will scale using **Read Replicas** and **Multi-AZ** for high availability. This ensures that your database can handle increasing read-heavy workloads while maintaining resilience.

ElastiCache scaling via **Auto-Scaling** based on demand will ensure that Redis can scale as required without manual intervention.

Cost Optimization: Leverage **Spot Instances** for ECS tasks to reduce costs while maintaining performance. Use **CloudFront** with caching to minimize data transfer costs and improve load times for users.

Security

IAM Roles & Policies: Implement least privilege access to restrict permissions, ensuring that resources are securely accessed.

VPC: Place **RDS** and **Redis** in private subnets for added security, while the ECS tasks run in public subnets behind the **Application Load Balancer (ALB)**.

Secrets Manager: Use **Secrets Manager** to securely store and rotate credentials (RDS, Redis) automatically.

CloudWatch: Set up **CloudWatch** for monitoring performance metrics and error logging. Utilize alarms for proactive responses to potential issues, ensuring high availability and performance.

Database Scaling

Use **RDS Read Replicas** for read-heavy operations to offload the primary database and improve performance.

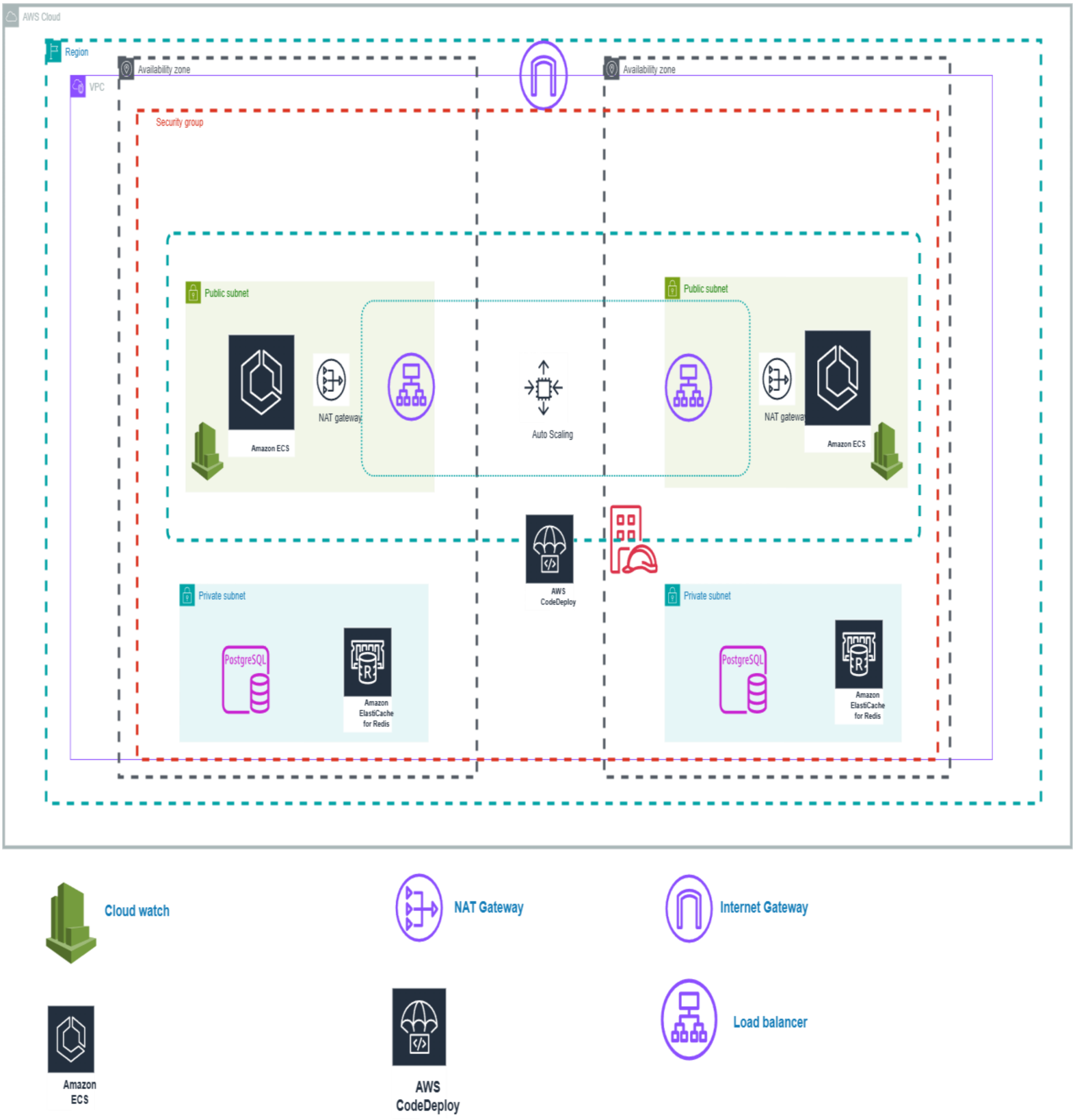
Auto-Scaling storage for RDS will ensure that the database capacity grows dynamically based on demand, without manual intervention.

Redis Task Scheduling

Use **ElastiCache Redis** to manage task queues and scheduling. Pair this with **Amazon EventBridge** to trigger periodic tasks, such as background jobs or session cleanup, at regular intervals.

This architecture ensures **scalability**, **cost efficiency**, and **security**, making it a robust solution for deploying a chess game application in AWS.

Architecture Diagram


AWS Cloud
Region
VPC
Availability zone
Availability zone
Security group
Public subnet
Public subnet
Amazon ECS
NAT gateway
Auto Scaling
NAT gateway
Amazon ECS
AWS CodeDeploy
Private subnet
PostgreSQL
Amazon ElastiCache for Redis
Private subnet
PostgreSQL
Amazon ElastiCache for Redis
Cloud watch
NAT Gateway
Internet Gateway
Amazon ECS
AWS CodeDeploy
Load balancer