Architectural Design Cache accessible to all services ostgreSQ -Build images for all services Automated Aurora Secondary tests Containe Registry Cluster Use packages for image builds Reader Reader Elasticache AZ 2 AZ3 Test containers for Expired token Build npm packagesrunning Jest, Cypress,. Code Artifact Aurora Global Database Github Actions Tasks are deployed in multiple AZ for availability Deploy using ECS blue/green deployment. ECS Deploy DB migrations using an ECS task. DDOS & bots /auth/login Aurora Primary Auth Store protection, CDN, Cluster compression,... token Authentication/Authorization refresh-token service /auth/certs Writer Reader Reader AZ 1 AZ 2 AZ 3 Data Store Cloudflare **API**GW API Service (Node.js) /svc2/xxx Async/long running tasks (i.e email,..) API Service 2 (Node.js) -SSR API calls Job Processors SQS queues media.domain.com/... Frontend cron like jobs (Next.js) NYT/OpenAI Static web resources EventBridge Scheduler Dashboards, monitors, anomaly detection, errors, metrics Cloud Watch Application events Logs, aws metrics-Opsgenie Datadog 3rd Party APIs

The above diagram is the proposed high level architecture for the new SaaS product. I'm looking forward to a deep dive into architecture during our interview.

I want to add a few notes on the proposed architecture:

- The Frontend is a monolith Next.js service but it could easily evolve to a micro frontend architecture for better scalability and ease/autonomous deployment with the ALB in front of it routing requests, based on the path, to different services.
- The CI/CD pipeline managed by Github Actions Workflows is also responsible for applying database migrations by launching ECS tasks and taking advantage of ECS blue/green deployment hooks to sync the rollout of new tasks with the migrations.
- All the infrastructure is provisioned with IAC (Terraform/Terragrunt) allowing it to easily support multiple environments (dev, test, staging, prod) and rebuild environments in case of disaster recovery.