Jane Doe
jdoe
CS 40
Feb 25, 2024

CS 40 Final Project: Yoctogram

Overview

For my project, I deployed scalable infrastructure for Yoctogram, a barebones Instagram clone. With Yoctogram, users around the world can upload private and public images and view their own private posts as well as other users' public posts. The entire infrastructure for Yoctogram is deployed via my submitted AWS CDK code (except for my use of the CS40 DNS provider). The application is live now at https://yoctogram.jdoe.infracourse.cloud.

Architecture

Services

Compute Resources

The frontend is stored in an AWS S3 bucket and served via Cloudfront for caching. The backend application layer is run on Docker containers on AWS ECS, using Fargate for elasticity and scalability. An AWS ALB is used to route requests efficiently to the application containers. Additionally, an AWS Lambda function is used to automatically compress images uploaded to the S3 buckets storing user images.

Data Resources

User information and image metadata is stored in a Postgres database running on AWS Aurora Serverless, allowing for improved elasticity for database compute resources. User images are stored in AWS S3 buckets. Cloudfront is also leveraged to cache the images from S3.

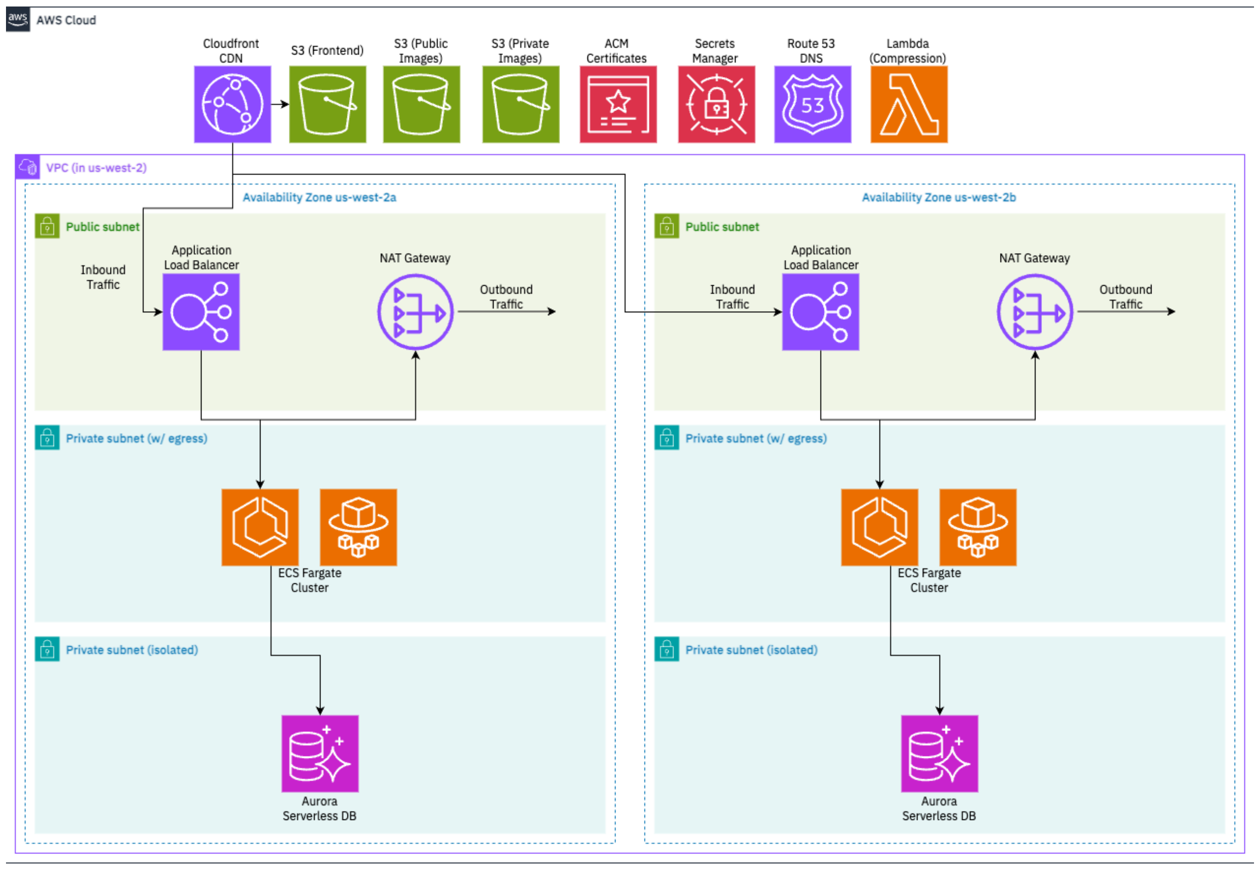
Network Resources

An AWS VPC is provisioned for this infrastructure, with parallel deployments in two availability zones. The ALBs are positioned in public subnets, the ECS clusters in private subnets with egress access, and the Aurora DB instances are placed in private isolated subnets. AWS Route 53 is used for managing the necessary DNS records, and Amazon Certificate Manager is used for creating TLS certificates.

Observability

Datadog agent containers are deployed as sidecars for each of the Fargate clusters, allowing the collection of resource metrics as well as logging of network traffic.

Diagram



Operation

Client requests first go through the Cloudfront endpoints, and are then routed to fetch the frontend hosted in S3. When API requests are made, Cloudfront routes the requests to the ALBs, which then direct them to a Fargate container for processing. The Fargate containers make requests to the Aurora DB, performing any necessary reads/writes in order to perform the API request. When a user posts a new image, it gets uploaded to an S3 bucket, thus triggering the compression Lambda function to compress the image and put the compressed version back into the same bucket.

Process

Difficulties Encountered

Integrating the backend and the frontend turned out to be somewhat tricky. In order to avoid issues with Cross-Origin Resource Sharing, both the Cloudfront frontend and ECS/ALB backend needed to be served from the same domain `yoctogram.jdoe.infracourse.cloud`. This in theory would lend itself to the use of the same ACM certificate for both entities; however, Cloudfront has a limitation that requires certificates to be provisioned in `us-east-1` despite the backend being in `us-west-2`. Thus, a second certificate needed to be generated for the frontend using a deprecated CDK API.

A second integration issue ended up being caching when fronting the backend API responses with Cloudfront. By default, Cloudfront applies caching to all `GET` requests, regardless of authentication status. However, this meant that loading the main feed after uploading an image would not show the new image, as Cloudfront cached the API feed response before the new image was uploaded. This meant that caching needed to be disabled for the backend API.

Lessons Learned

Deploying web applications to the cloud is tricky. While the general architecture might be similar between one app and the next, there is a lot of intricate configuration work that needs to be specific to the particular application at hand. For example, Yoctogram stores many images per user, compression needs to be used to minimize the storage cost, but this may not be necessary for other applications. Additionally, the Yoctogram application expects its secrets to be provided as environment variables with specific names that are different from AWS's generic secret names, so some translation work needs to be done within CDK to provide this.