CSC409 A3 Report

Dec 10, 2022

Group 47
Lakshya Gupta 1005569449
Ibrahim Fazili 1004882763

CSC409 A3 Report

- **Architecture**

  The system has multiple services running on AWS, here's a brief overview of them:

  - **Storage (Elasticache & Keyspaces):** We needed to store the board state and be able to fetch and modify it quickly. So, we store the board in elasticache's in-memory data store (using Redis as its storage engine). The board is stored as a long 1000 x 1000 bitfield which can be concurrently read and modified. For board persistence, every write to the bitfield was followed by a write to Keyspaces (Cassandra). We stored a single entry for every pixel (x, y) in the database. This means that in the event of Elasticache failure, the board state can be restored using data from keyspaces

  - **API (Lambda):** We needed to have functionality for processes such as getting the board, writing a pixel, or fetching the last time a user wrote to a board. We wrote each of these functionalities separately as Lambda functions which can read and write from Keyspaces and Elasticache depending on the use case. We have API routes attached to the load balancer to execute each lambada, i.e, http://{loadbalancerIP}.com/api/writepixel with the correct body will call and execute the WritePixel lambda. The lambda functions are stored as zip files which are packaged and stored in S3 where they are called from

  - **Routing:** All incoming requests go to CloudFront (CDN). CDN caches the results of recent API calls and the webpage requests which reduce client latency and help reduce the load on the system. If a request can't be satisfied by CDN, the request is forwarded to our application load balancer (ALB). ALB routes requests to different lambda functions depending on the request. If the client requests our webpage from the CDN, the CDN fetches the webpage from our S3 bucket which hosts our webpage

- **Socket Servers (ECS & EC2 Compute):** In order to provide real-time updates to all clients about all pixel updates, we have clients connect to our real-time update service (written in Golang) which runs on EC2 compute instances running a cluster of docker containers. EC2 compute instances provide the hardware for running containers, but the containers are managed by ECS. ECS uses all available compute instances to deploy containers depending on the load. The containers run on our custom update service docker image which ECS downloads from AWS's container registry (ECR). Depending on the load, the compute instances and containers can be scaled independently to keep the service functional and responsive. ECS scales the containers automatically and compute instances are part of an auto-scaling group (ASG) which controls scaling

- **Management (CloudWatch & CloudFormation):** Throughout the development process we needed to use logs for debugging as we could not use a debugger when testing the end-to-end flow with the ALB. To solve this, we used CloudWatch where we could monitor and investigate logs. In addition, we needed to be able to deploy our lambdas, ECS, Keyspace, ALB etc at one time using automation so it can be accessible. We used CloudFormation for this as we were able to create our resources in a 'stack' which would run under the same VPC.

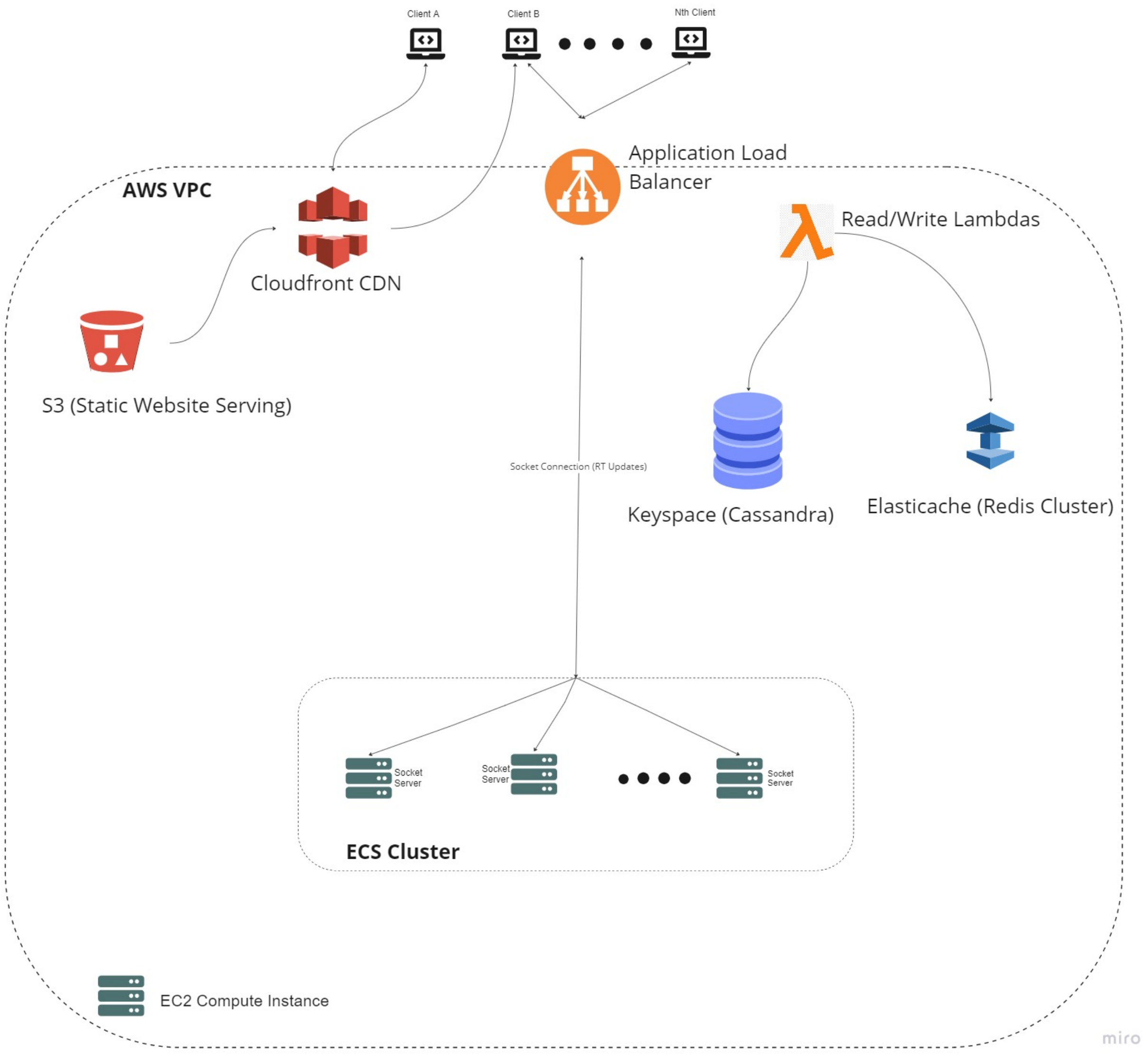
- Strengths

Every part of our system is a service contained within itself and is capable of scaling independently according to the client load. This allows the system to always be responsive and available. We use CDN for caching all API calls and our webpage which makes the request latency very low and it makes the system feel extremely responsive. Due to the self-contained nature of all the services, any one service going down doesn't lead to the entire system crashing. Critical services going down may make the system temporarily unavailable, but once those services are back, the system recovers and becomes functional without any manual intervention

- Weaknesses

The use of CDN to cache the board state means that if a client downloads the board right before it is about to expire, the client can have a board that’s behind the actual board. Also, using one-time websocket updates to update individual pixels can also lead to the board being out of sync from the actual state if any updates get dropped due to network (or other) errors. This architecture is good for performance and responsiveness, but in order to deal with sync issues, we have to periodically download the entire board which is expensive for both the client and our servers

Here’s a diagram showing the system:



Complete List of AWS Services used:

- Compute
  - Elastic Cloud Compute (EC2)
  - Lambda
- Containers
  - Elastic Container Registry (ECR)
  - Elastic Container Service (ECS)
  - Simple Storage Service (S3)
- Storage
  - S3
- Database
  - Elasticache (Redis)
  - Keyspaces
- Networking & Content Delivery
  - VPC
  - CloudFront
- Management
  - CloudWatch
  - CloudFormation
- Security, Identity & Compliance
  - IAM