Assignment 3 Report

/r/place

🏗️ Architecture

Our app uses the following AWS Services:

- EC2 Instances
- Lambda
- Elasticache
- API Gateway
- S3

(Architecture diagram provided at the bottom of document)

We store the state of the board in **ElastiCache**, In a **Redis** instance. Our client is hosted in **S3** and served statically. We use **Lambda** functions for writing pixels to the board. Our **Lambda** functions are behind an **API Gateway**. We use **EC2 Instances** to host websocket servers, which serve the entire board as well as live updates to the clients. The **Lambdas** and **EC2 Instances** communicate using the **Redis** publisher/subscriber mechanism.

🏁How to run the System (Orchestration)

Production Setup:

To start EC2:
Add new instance using auto scale group UI, run

```
> ./initInstance <new instance’s public dns addr.>
```

To take down EC2:

```
ec2shell> pm2 rm npm
```

Remove instance through AWS if needed

Everything else is done through AWS UI.

📡Broadcast Update

The user can send a pixel by picking a color from the palette and clicking on the canvas. This is what happens in order from a click.

1. Client sends a PUT request with a body in the form of `{x, y, colorId}` to Lambda.
2. Lambda validates the request and inserts the pixel into Redis if valid.

3. Lambda then publishes the update to Redis.
4. EC2 servers subscribed to Redis see the pixel being updated, and broadcast the update to all clients through their websocket connections.
5. Clients receive the update message, and draws the pixel to canvas.

🛡️Spam Protection

We limit the user to be able to only send 1 pixel every 5 minutes by blocking any incoming requests from their IP for 5 minutes once they have successfully sent a pixel in. If the user attempts to paint more pixels during this time, their requests will be blocked and they will be notified.

📈Scalability

Redis: We have a single Redis instance, however we can easily scale horizontally by using a Redis with replication, or redis in cluster mode (need to switch libraries). We can vertically scale the Redis instance through AWS.

EC2 Instances: Our websocket servers sit behind a TCP load balancer. We can add or remove EC2 nodes, and the load balancer will distribute new clients to them. For vertical scaling we can select the appropriate machine capabilities.

Lamda: Lambda functions scale automatically. If a new request is received and all existing Lambdas are busy handling requests, a new instance will be spun up by AWS. If the request load is lower, and there are instances that aren't being used, AWS will automatically take the unused instances.

□Availability

Our client is served through S3, and hosted and served from multiple availability zones.

If any individual EC2 instance goes down, it will fail health checks from the load balancer and the load balancer will stop routing requests to it. We are running our servers with pm2, so if the server crashes, but the instance remains operational then pm2 will revive the server.
Our EC2 instances are hosted in multiple availability zones so if an availability zone goes down we remain operational.

AWS Lambda is supported across multiple availability zones. If a Lambda instance isn't functioning, the API Gateway won't route requests to it and a new instance will be created.

We only have a single Redis instance, so if that availability zone goes down we will run into problems.

Another issue is since we handle the code deployment ourselves, all instances receive updates at the same time, so on updates there will be a brief moment of downtime. This can be mitigated by changing our scripts.

🔒Security

Lambda functions are accessible only within the VPC. EC2 is only accessible within the VPC, unless from developer IPs using port 22 (SSH). Redis is accessible only within the VPC. Our client files (S3) are completely public.

⚖️Weakness/Strengths of our System

- One of our weaknesses is the way that we retrieve the state of the board from redis. There is no method to retrieve an entire bitfield, so instead we have to request each portion of the board individually. We request 32 bits at a time, and batch thousands of these requests together into a single request. Even with the request batching we believe retrieving the state of the board is quite expensive.

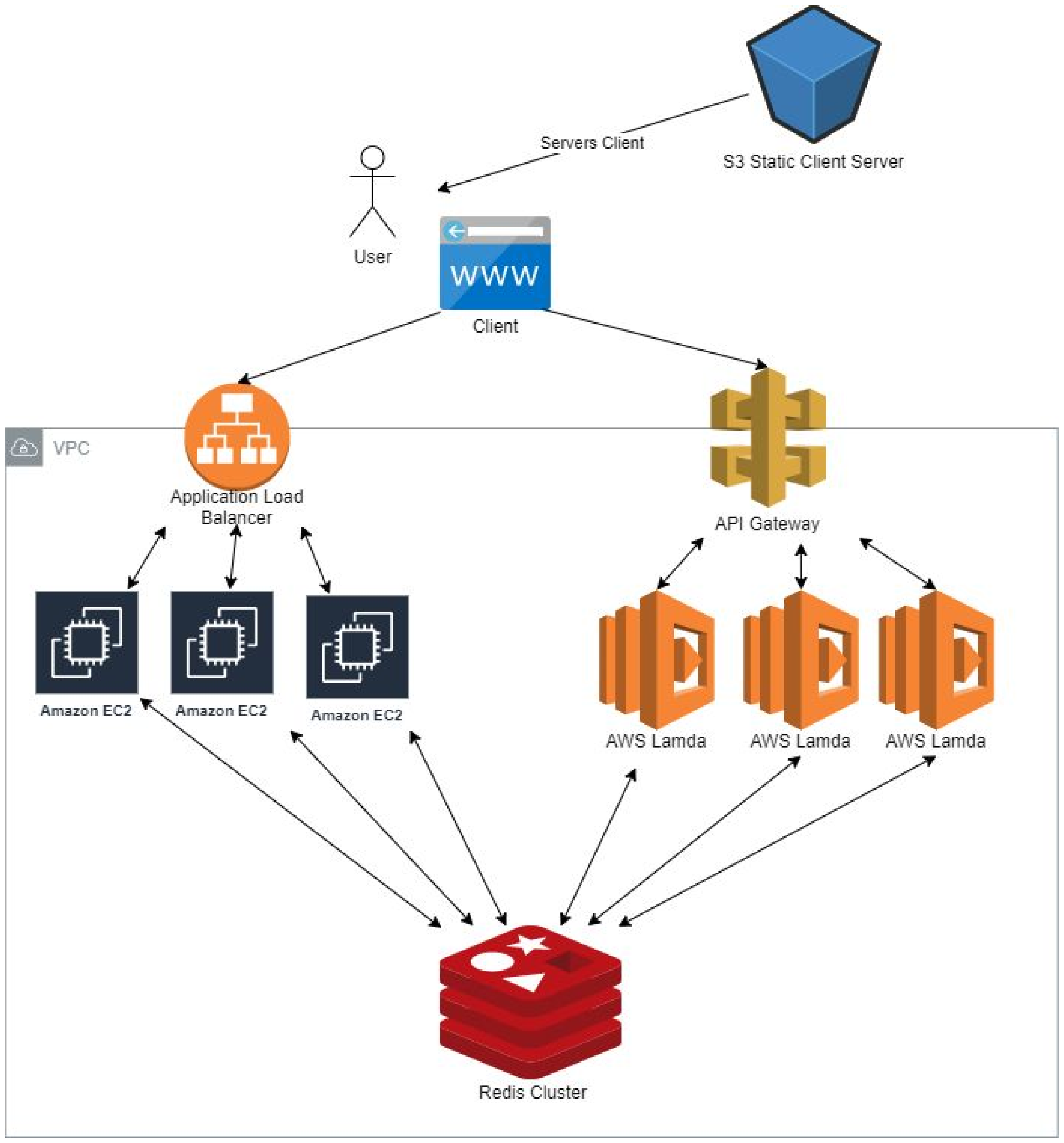
- Another weakness of the system is that each of our EC2 instances maintains its own cache of the board state. In the reddits original design of the board, they used a CDN to distribute the state of the board, and only maintained a single cache. We would've liked to implement this but we ran out of time.

- A strength of the system is the autoscaling of the Lambda functions. AWS Lambda scales incredibly quickly so if we suddenly start receiving a ton of write requests, (possibly because of a DDOS), our system can quickly scale to handle the load, and our regular users won't be impacted.

- Another strength of our system is that all portions are scalable both horizontally and vertically.

- Finally, while it's not necessarily a strength of our "system", we believe our UI is quite nice. The UI supports both scrolling and panning the canvas, so it's easier to edit the pixels.

Architecture Diagram


S3 Static Client Server
Servers Client
User
Client
Application Load Balancer
API Gateway
VPC
Amazon EC2
Amazon EC2
Amazon EC2
AWS Lamda
AWS Lamda
AWS Lamda
Redis Cluster