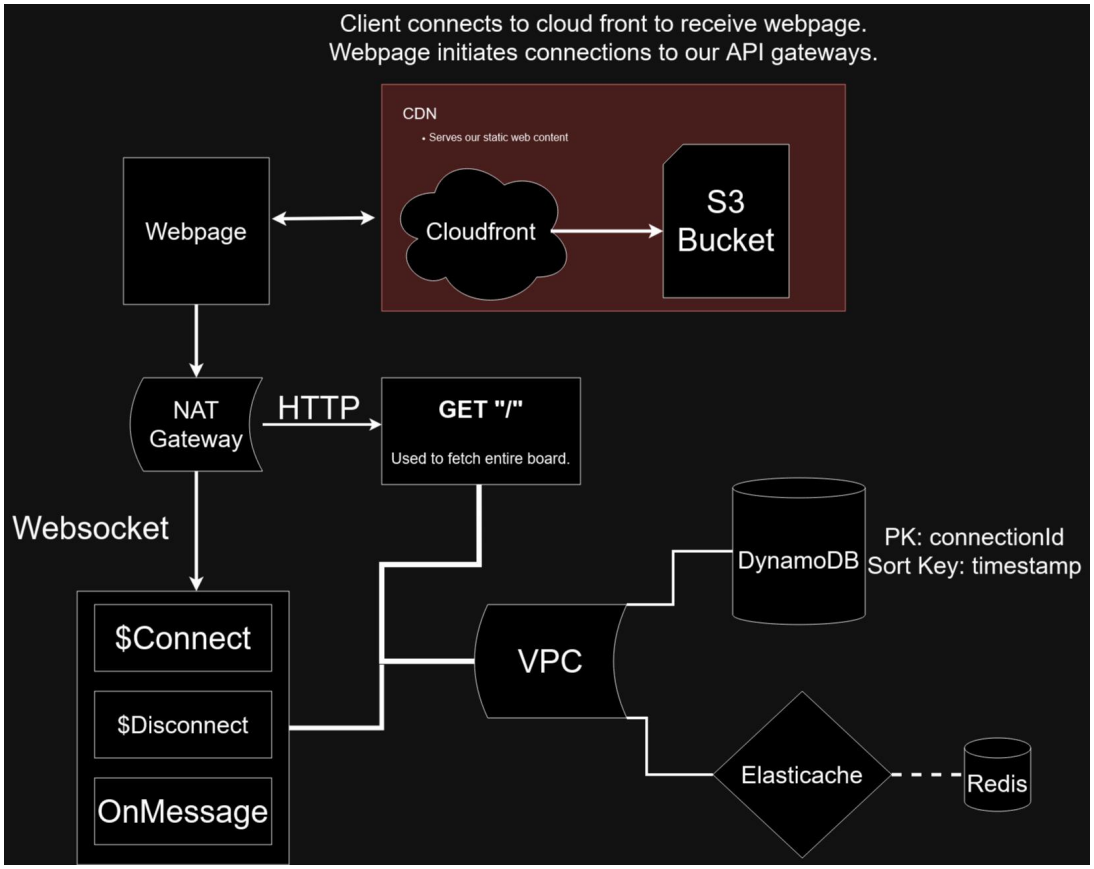
CSC409 A3 Report

duvallou, hokinmat, tabbahch
Louis-Phillippe Duval, Matthew Ho Kin, Christian Tabbah



For more diagrams see systemDiagram.pdf

Implementation Details:

Our group decided on a serverless approach for rPlace to help reduce overhead while also providing theoretically infinite scaling. The webpage and other static content, is served through CloudFront while our app is implemented through two API gateways.

One gateway is an REST gateway used only for retrieving the full board. We found that sending the entire board, whether it was as an array of integers or a bytestring, was too slow using websockets so we opted for an HTTP endpoint. The other gateway is a websocket gateway which invokes Lambda functions to manage and broadcast to active connections. Both websockets are configured to run on all Availability Zones in us-east-1.

For our backend we decided to use DynamoDB for persistent datastore and Redis Elasticache. Both of these resources are secured with a VPC and Lambda execution permissions. Additionally, these resources are also available across multiple Availability Zones.

Our CloudFront CDN is configured to serve content across all the regions in North America. It pulls from an S3 bucket which holds our static files.

Services:

Our system is built with Dynamo, Elasticache, Lambda, API Gateway, Cloudfront, CloudFormation, IAM, VPC (subnet and nat etc.), and S3 bucket.
See systemDiagram.pdf for more information.

Spam Protection:

Our system implements spam protection by logging user requests on DynamoDB, including the time stamp. When a user sends a message, the timestamp is compared against the timestamp of their most recent request. After processing the user's message, it is subsequently logged in the database. Currently, Lambda is locked to 10000 requests per second with a burst of 5000 requests per second. The timeout is 5 minutes as required.

Broadcast Updates:

Our API Gateway makes use of Javascripts aws-sdk to broadcast messages to all connected users. Connections are tracked on redis, the connect and disconnect functions add and remove the user's connectionId from the cache.

Scalability:

Our system has theoretically infinite vertical and horizontal scaling. Websockets are managed using Lambda functions, hence they scale with demand within the account concurrency settings. DynamoDB is set to On-Demand computing instead of provisioned computing, allowing it to accept a higher volume of requests during peak hours. Lastly, our system uses Elasticache to store the board and current connections. Normally, when Elasticache is configured to be serverless, Amazon will handle the autoscaling. The current configuration of our Elasticache is a t2.micro cache, to help save on cost. Elasticache stores our Board as a Bitfield helping us save on storage and improve performance.

Availability:

Our system runs on multiple availability zones. Elasticache has an automatic Multi-AZ setting, in which it creates nodes in different AZ's. DynamoDB is managed by Amazon and not locked to a single AZ. Lastly, we serve our API Gateway and Cloudfront on multiple subnets so that it works across multiple AZs.

Security:

Our VPC for internal resources, like the S3 bucket, and Elasticache, are configured so that they only allow network traffic from within the VPC. DynamoDB is secured through security policies and roles. Each Lambda function is given permission policies which give it the bare minimum permissions to access resources.

Some of the AWS managed security policies we used are:

- AmazonAPIGatewayInvokeFullAccess
- AmazonDynamoDBInvokeFullAccess
- AmazonElastiCacheFullAccess
- AmazonLambdaDynamoDBExecutionRole
- AmazonLambdaVPCAccessExecutionRole