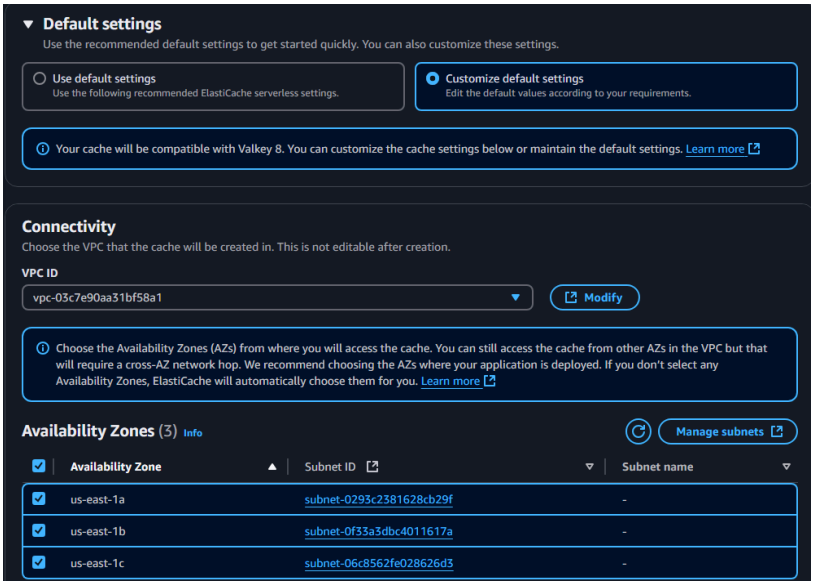
To keep things simple, name the lambdas after their directory name in the repo to follow along, make sure to update api links, elasticache links, and service names in the lambdas and the scribble.html static files where necessary. Lambda breakdown at the bottom. Any fields not mentioned or pictured should be left as default or configured based on preference/use case.

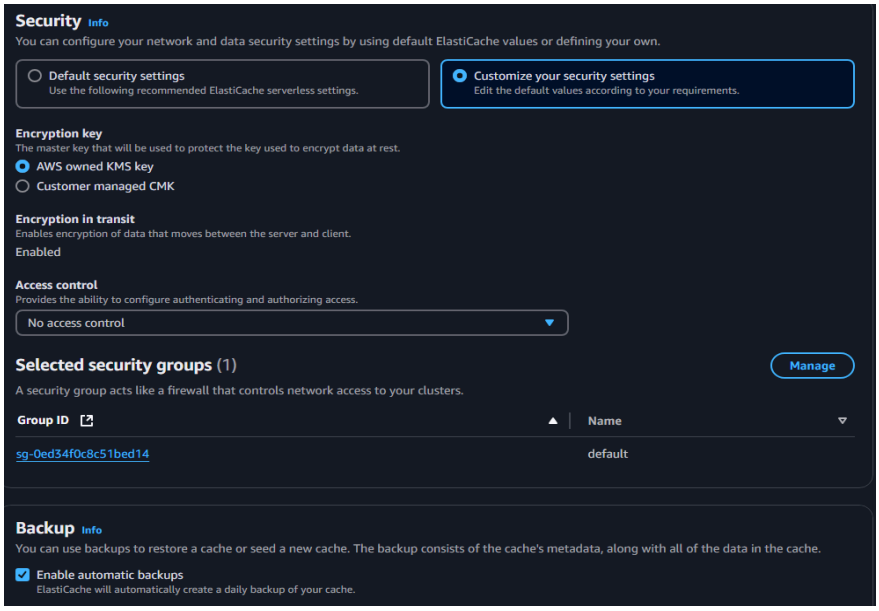
1. Starting with user with AdministratorAccess permission policy in order to seamlessly use all the services. The default security group amazon provides should be sufficient anytime a “security group” field is request, so long as that entity is also added to the VPC created with the steps below.

2. Create a vpc following the instructions in this link under “I don’t have a vpc yet”, which allows our lambda functions in our vpc access to the internet, which is important to allow them to post back to the websocket api gateway, otherwise the lambdas will timeout.

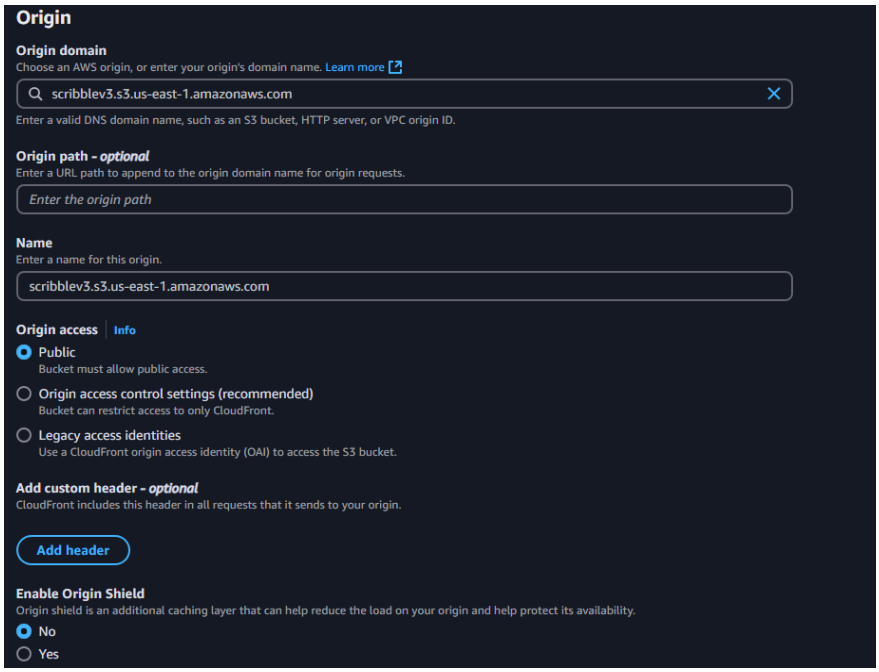
3. (Come back to this step after step 10) We then create a websocket api, with “Route selection expression” set to “request.body.action”, next page add the predefined “$connect” and “$disconnect” routes, and a custom sendMessage route. Next page, we set all integration types to lambda and select the corresponding lambdas as defined at the bottom of this document. Then simply name the stage and create. The url for the websocket api will be on the stages page, and the url for the lambdas to connect using boto3 will be right below, but make sure to remove the “@connect” at the end

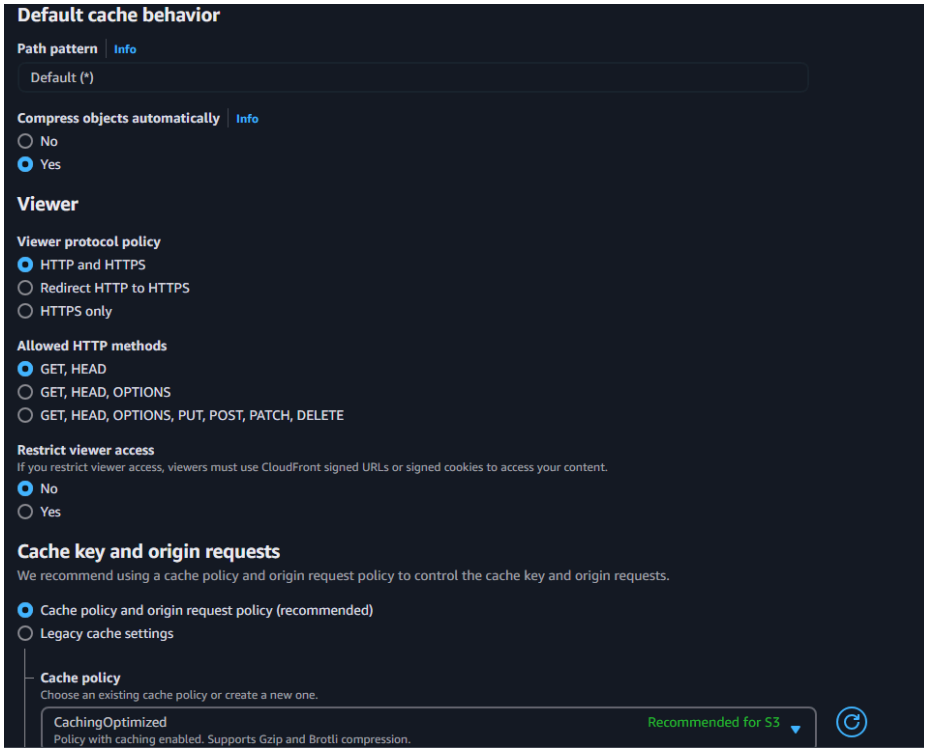
4. Create ElastiCache valkey serverless instance, with version 8, selecting the private subnets present within the created vpc (They will be in different AZ’s) and choose default security group as in the 2 images below

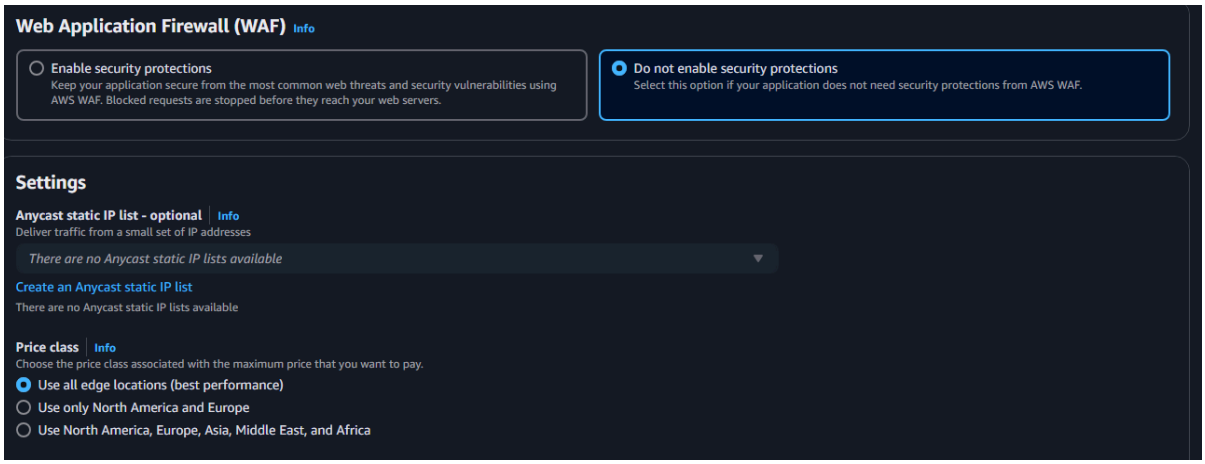




5. Next, create an S3 bucket called “scribblev3” for static files to be served up by cloudfront ensuring that they have public access unblocked and ACLs enabled and set to Bucket-Owner preferred. After creating the bucket, click on it and under Permissions, find the ACL tab and add read permissions to “Everyone”. After this, simply upload static files, then set the same permissions for each file if not already set by clicking on it and clicking permissions. Then proceed to setup your cloudfront distribution as such (make sure to set “default root object” to scribble.html) :

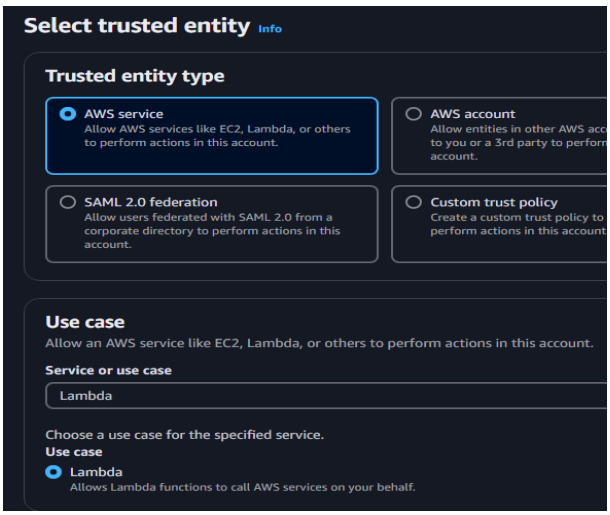






6. Following this, go to the policies section of Cloudfront and set up a caching policy with a 60 second max and default ttl, then change the cache policy for the created distribution (visible in the image above and possible to find in the settings) to the one you just made.

7. We must create a role that allows our lambda functions to invoke api actions (for managing and responding back to connections) along with any other permissions it needs to have such as accessing dynamoDB. Under the IAM service, roles, create role named scribbleV3Funcs:



8. And on the next page add the “AmazonAPIGatewayInvokeFullAccess”, “AmazonS3FullAccess”, “AmazonSQSFullAccess” and “AWSLambdaVPCAccessExecutionRole” policies and no permission boundaries. After creating the role, click on it and under add permissions select “create inline policy”. From there, we must select the dynamoDB service and under the read and write access levels, select getitem, batchgetitem and updateItem, putitem actions, respectively. For arn resources, check the “any from this account” box on the right of the line.

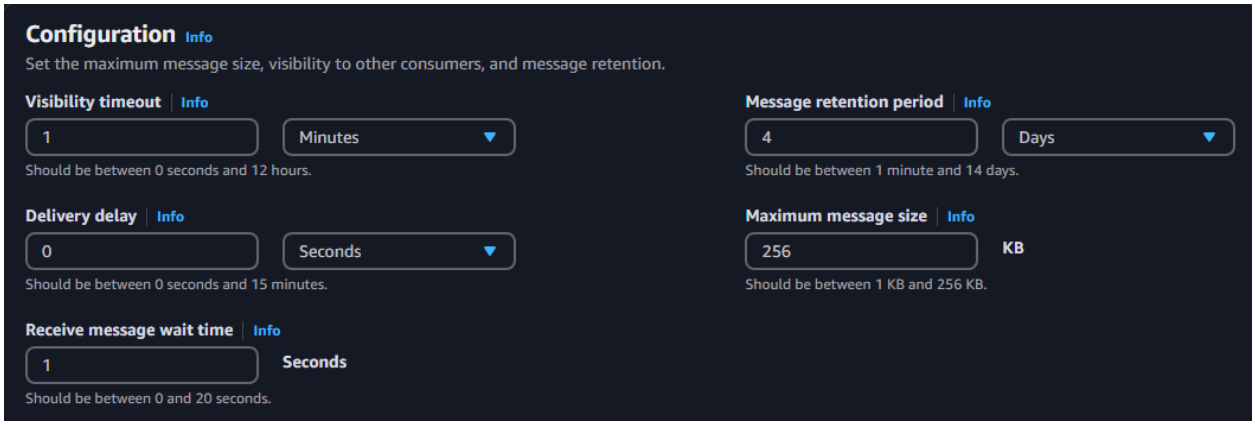
9. We then create a default dynamoDB table called scribbleV3DB. First, follow this link to create an endpoint in our vpc (adding the endpoint to the private route tables) that will automatically route any traffic bound for the dynamoDB service properly within the vpc (you may note that an s3 gateway has already been created for us), so that it will work as described in this link. After that, simply create a table named “scribbleV3DB” with a Number partition key named “idx”.

10. We can now create our functions, ensuring that we “change default execution role” to the one we made for our lambdas and under additional configurations, connecting to the same vpc, availability zones, and security groups as our elasticache instance. Ensure only to select private subnets when configuring the lambda on the vpc. The “updateImageV3” lambda requires these extra steps to add the “Pillow” (not “pillow”) package as a layer from the repository mentioned in the article to create the png image. Simply go to the repo linked in the article, make sure the selected runtime of the lambda is one of the ones listed under the “list of arns” link in the repo readme, select the html of your region, and ctrl+f the Pillow package’s arn, then simply add it as a layer to the “updateImageV3” lambda at the very bottom of its page. Ensure that updateImageV3, catchUpClientV3 and updateDBV3 lambdas have a higher timeout interval (about 1 minute) set under “general configuration” because they are intensive lambdas that can take quite some time to run sometimes. (Go back to step 3)

11. We will also create an “Amazon Eventbridge” “Scheduler” instance to run every minute to update our image.png that will be served to the front end. It will be a “recurring schedule” pattern on a rate-based schedule set to 1 minute and flexible time window off. After the first creation page, we select the “Aws lambda” target and select the “updateImageV3” lambda. We must create another role that allows the scheduler to invoke our lambda. Similar to steps 7 instead of selecting “AWS Service” select “custom trust policy” and on the right click add principle, then “aws service” from dropdown, then replace bracketed section with “scheduler” (the {} brackets and everything in between). Then simply attach the “AWSLambda_FullAccess” policy similar to step 8.

12. We then use sqs to decouple the database and image updates from the elasticache operations to make our application faster, modeled after this guide. We must create two standard queues, one called catchUpQueueV3 and another called scribbleV3Queue,

both with these settings:



And for the access policy, select “Only the specified…” and paste in the arn of the scribbleV3Func role, found by clicking on the role in the IAM service in another tab. For each queue, click on it, and add a lambda trigger under the corresponding tab, using the lambda breakdown below

Lambda breakdown:
ConnectV3:

- Triggered by the api $connect route
- Adds the connection ID of the connecting client to the elsaticahce “connections” list
- Queues the ID to receive catchup data by queuing a message to the catch catchUpQueueV3 queue to be processed by catchUpClientV3

DisconnectV3:

- Triggered by the api $disconnect endpoint
- Removes the connection ID of the disconnecting client from the elasticache “connections” list

updateImageV3:

- Triggered by the eventbridge scheduler instance every minute
- Reads in the bitmap string from ealsticache and uses it to update the png image in the S3 bucket

catchUpClientV3:

- Triggered by adding a message to the catchUpQueueV3 queue
- Digests the queue message and posts any points currently in the cache to the connectionID obtained from the queue message

updateDBV3:

- Triggered by adding a message to the scribbleV3Queue queue
- Digests the queue message to get point coordinates (x, y) which it puts into the dynamoDB table and also broadcasts to any currently connected clients

sendMessageV3:

- Triggered by the api “sendMessage” route
- Processes incoming messages to the websocket containing (x,y) coordinates

- Sets the bitmap index corresponding to coordinates in elasticache and adds the coordinates as a key that expires after 2 minutes
- Puts a message on the queue scribbleV3Queue to be processed by updateDBV3