

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

The screenshot shows the AWS ElastiCache console interface. At the top, under 'Default settings', there are two radio buttons: 'Use default settings' (selected) and 'Customize default settings'. Below this is a message: 'Your cache will be compatible with Valkey 8. You can customize the cache settings below or maintain the default settings. Learn more'. The 'Connectivity' section shows a dropdown for 'VPC ID' with the value 'vpc-03c7e90aa31bf58a1' and a 'Modify' button. Below this is a message about choosing Availability Zones. The 'Availability Zones' section shows a table with three rows, each representing an availability zone and its corresponding subnets.

Availability Zone	Subnet ID	Subnet name
us-east-1a	subnet-0293c2381628cb29f	-
us-east-1b	subnet-0f33a3dbc4011617a	-
us-east-1c	subnet-06c8562fe028626d3	-

**Security** [Info](#)

You can configure your network and data security settings by using default ElastiCache values or defining your own.

☐ Default security settings  
Use the following recommended ElastiCache serverless settings.

☒ Customize your security settings  
Edit the default values according to your requirements.

**Encryption key**

The master key that will be used to protect the key used to encrypt data at rest.

☒ AWS owned KMS key

☐ Customer managed CMK

**Encryption in transit**

Enables encryption of data that moves between the server and client.

Enabled

**Access control**

Provides the ability to configure authenticating and authorizing access.

No access control

**Selected security groups (1)** [Manage](#)

A security group acts like a firewall that controls network access to your clusters.

Group ID	Name
sg-0ed34f0c8c51bed14	default

**Backup** [Info](#)

You can use backups to restore a cache or seed a new cache. The backup consists of the cache's metadata, along with all of the data in the cache.

☒ Enable automatic backups  
ElastiCache will automatically create a daily backup of your cache.

- 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) :

**Origin**

**Origin domain**

Choose an AWS origin, or enter your origin's domain name. [Learn more](#)

scribblev3.s3.us-east-1.amazonaws.com

Enter a valid DNS domain name, such as an S3 bucket, HTTP server, or VPC origin ID.

**Origin path - optional**

Enter a URL path to append to the origin domain name for origin requests.

Enter the origin path

**Name**

Enter a name for this origin.

scribblev3.s3.us-east-1.amazonaws.com

**Origin access** [Info](#)

☒ Public  
Bucket must allow public access.

☐ Origin access control settings (recommended)  
Bucket can restrict access to only CloudFront.

☐ Legacy access identities  
Use a CloudFront origin access identity (OAI) to access the S3 bucket.

**Add custom header - optional**

CloudFront includes this header in all requests that it sends to your origin.

[Add header](#)

**Enable Origin Shield**

Origin shield is an additional caching layer that can help reduce the load on your origin and help protect its availability.

☒ No

☐ Yes

### Default cache behavior

**Path pattern** [Info](#)

Default (\*)

**Compress objects automatically** [Info](#)

☐ No

☒ Yes

**Viewer**

**Viewer protocol policy**

☒ HTTP and HTTPS

☐ Redirect HTTP to HTTPS

☐ HTTPS only

**Allowed HTTP methods**

☒ GET, HEAD

☐ GET, HEAD, OPTIONS

☐ GET, HEAD, OPTIONS, PUT, POST, PATCH, DELETE

**Restrict viewer access**

If you restrict viewer access, viewers must use CloudFront signed URLs or signed cookies to access your content.

☒ No

☐ Yes

**Cache key and origin requests**

We recommend using a cache policy and origin request policy to control the cache key and origin requests.

☒ Cache policy and origin request policy (recommended)

☐ Legacy cache settings

**Cache policy**

Choose an existing cache policy or create a new one.

CachingOptimized Recommended for S3

Policy with caching enabled. Supports Gzip and Brotli compression.

### Web Application Firewall (WAF) [Info](#)

☐ Enable security protections

Keep your application secure from the most common web threats and security vulnerabilities using AWS WAF. Blocked requests are stopped before they reach your web servers.

☒ Do not enable security protections

Select this option if your application does not need security protections from AWS WAF.

**Settings**

**Anycast static IP list - optional** [Info](#)

Deliver traffic from a small set of IP addresses

There are no Anycast static IP lists available

Create an Anycast static IP list

There are no Anycast static IP lists available

**Price class** [Info](#)

Choose the price class associated with the maximum price that you want to pay.

☒ Use all edge locations (best performance)

☐ Use only North America and Europe

☐ Use North America, Europe, Asia, Middle East, and Africa

- 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.

- 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:

### Select trusted entity [Info](#)

**Trusted entity type**

☒ AWS service

Allow AWS services like EC2, Lambda, or others to perform actions in this account.

☐ AWS account

Allow entities in other AWS accounts to you or a 3rd party to perform actions in this account.

☐ SAML 2.0 federation

Allow users federated with SAML 2.0 from a corporate directory to perform actions in this account.

☐ Custom trust policy

Create a custom trust policy to perform actions in this account.

**Use case**

Allow an AWS service like EC2, Lambda, or others to perform actions in this account.

**Service or use case**

Lambda

Choose a use case for the specified service.

**Use case**

☒ Lambda

Allows Lambda functions to call AWS services on your behalf.

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:

The screenshot shows the 'Configuration' tab for an AWS SQS queue. The settings are as follows:

Setting	Value	Unit	Range
Visibility timeout	1	Minutes	Should be between 0 seconds and 12 hours.
Message retention period	4	Days	Should be between 1 minute and 14 days.
Delivery delay	0	Seconds	Should be between 0 seconds and 15 minutes.
Maximum message size	256	KB	Should be between 1 KB and 256 KB.
Receive message wait time	1	Seconds	Should be between 0 and 20 seconds.

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 elasticsearch “connections” list
- Queues the ID to receive catchup data by queuing a message to the catchUpQueueV3 queue to be processed by catchUpClientV3

DisconnectV3:

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

updateImageV3:

- Triggered by the eventbridge scheduler instance every minute
- Reads in the bitmap string from elasticsearch 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