



School Of Computing

ST505

ESDE (Cloud Computing)

WEB APPLICATION CLOUD COMPUTING REPORT

AY2021S2 2A/23

Team Members:

Sara Golkaran (P1935657)

Yap Min Ru Vicki (P1904556)

1. Deploy Existing Web Application on AWS (Static)	3
2. Deploy Existing Web Application on AWS (Dynamic)	9
3. Troubleshoot and Test during Deployment Stage	23
3.1 Changing Modules from Bcrypt to BcryptJS	23
3.2 Faced 403 Forbidden Error Issue	25
4. Refactoring Application using Lambda Function	26
4.1 Creation of Amazon DynamoDB	26
4.2 Creation of Lambda Function to Get All Users Submission	29
4.3 Creation of AWS Web API Gateway	30
5. Cloud Security Features	33
5.1 Centralised Logging for Web API Access	33
5.2 MySQL RDS Database Security	37
5.3 Storing Private Key using AWS Secrets Manager	39
5.4 Server Side Encryption (SSE) on S3 Bucket	43
5.5 Enforcing CORS	46
6. Additional Feature	48
6.1 Created Button in Frontend to execute Lambda Function	48
7. AWS Cognito service for Authentication (Advanced)	50

GitHub repository link: <https://github.com/saragolkaran/ESDE-Cloud-Computing-CA2->

1. Deploy Existing Web Application on AWS (Static)

In this section, we will be showing a step-by-step approach on how to create a S3 bucket, adding a bucket policy to make its content publicly available and the deployment process of our existing web application using static API.

Step 1: After logging into our AWS account, we navigated to the S3 dashboard and clicked on the “Create bucket” button. We initially applied the default settings for our bucket configuration which includes blocking of all public access and disabled bucket versioning before naming it “testing1.com”.

The screenshot shows the "Create bucket" wizard in the AWS S3 console. In the "General configuration" step, the bucket name is set to "testing1.com". The "Region" is selected as "US East (N. Virginia) us-east-1". Under "Block Public Access settings for bucket", the "Block all public access" checkbox is checked. This setting applies to buckets and objects granted through new access control lists (ACLS). It also applies to buckets and objects granted through new public bucket or access point policies. It ignores public and cross-account access for buckets or access points with policies that grant public access to buckets and objects. The "Bucket Versioning" section is visible below.

The screenshot continues the "Create bucket" wizard. The "Bucket Versioning" section is expanded, showing that versioning is a means of keeping multiple variants of an object in the same bucket. It can be used to preserve, retrieve, and restore every version of every object stored in your Amazon S3 bucket. With versioning, you can easily recover from both unintended user actions and application failures. The "Tags (0) - optional" section shows that no tags are associated with this bucket, with an "Add tag" button.

The screenshot shows the "Default encryption" section, where server-side encryption is disabled. Below it is the "Advanced settings" section, which contains a note: "After creating the bucket you can upload files and folders to the bucket, and configure additional bucket settings." At the bottom of the page are "Cancel" and "Create bucket" buttons.

Step 2: After we have created a bucket, we can enable static website hosting for our bucket. We navigated to “Static website hosting” and clicked on the “edit” button to change to “enabled”.

The screenshot shows the AWS S3 Bucket Properties page for a bucket named 'testing1.com'. The 'Static website hosting' section is expanded, showing the following configuration:

- Static website hosting**: Enabled
- Hosting type**: Bucket hosting
- Bucket website endpoint**: <http://testing1.com.s3-website-us-east-1.amazonaws.com>

Step 3: We clicked on the link at the bottom of the previous page and encountered a 403 forbidden error that prevents us from accessing the bucket website endpoint. This means that our permission settings need to be changed to allow universally everyone to access.



Step 4: We cleared all the ticked checkboxes under “Block all public access” before choosing “save changes”. We were prompted to enter “confirm” to authorize the edit.

Edit Block public access (bucket settings)

Block public access (bucket settings)

Public access is granted to buckets and objects through several mechanisms: bucket policies, access point policies, or ACLs. In order to ensure that public access to all your S3 buckets and objects is blocked, turn on these all public access. These settings apply only to this bucket. We recommend that you turn on Block all public access, but before applying any of these settings, ensure that your applications will work correctly without public access. If you require some level of public access to your buckets or objects within, you can customize the individual settings below to suit your specific storage use cases. [Learn more](#)

- Block all public access** Turning this setting on at the same time as turning on all four settings below. Each of the following settings are independent of one another.
- Block public access to buckets and objects granted through new access control lists (ACLs)** S3 will block public access permissions applied to newly added buckets or objects, and prevent the creation of new public access ACLs for existing buckets and objects. This setting doesn't change any existing permissions that allow public access to S3 resources.
- Block public access to buckets and objects granted through new public bucket or access point policies** S3 will block new bucket and access point policies that grant public access to buckets and objects. This setting doesn't change any existing policies that grant public access to buckets and objects.
- Block public access to buckets and objects granted through any access control lists (ACLs)** S3 will ignore public and cross-account access for buckets or access points with policies that grant public access to buckets and objects.

[Cancel](#) [Save changes](#)

Edit Block public access (bucket settings)

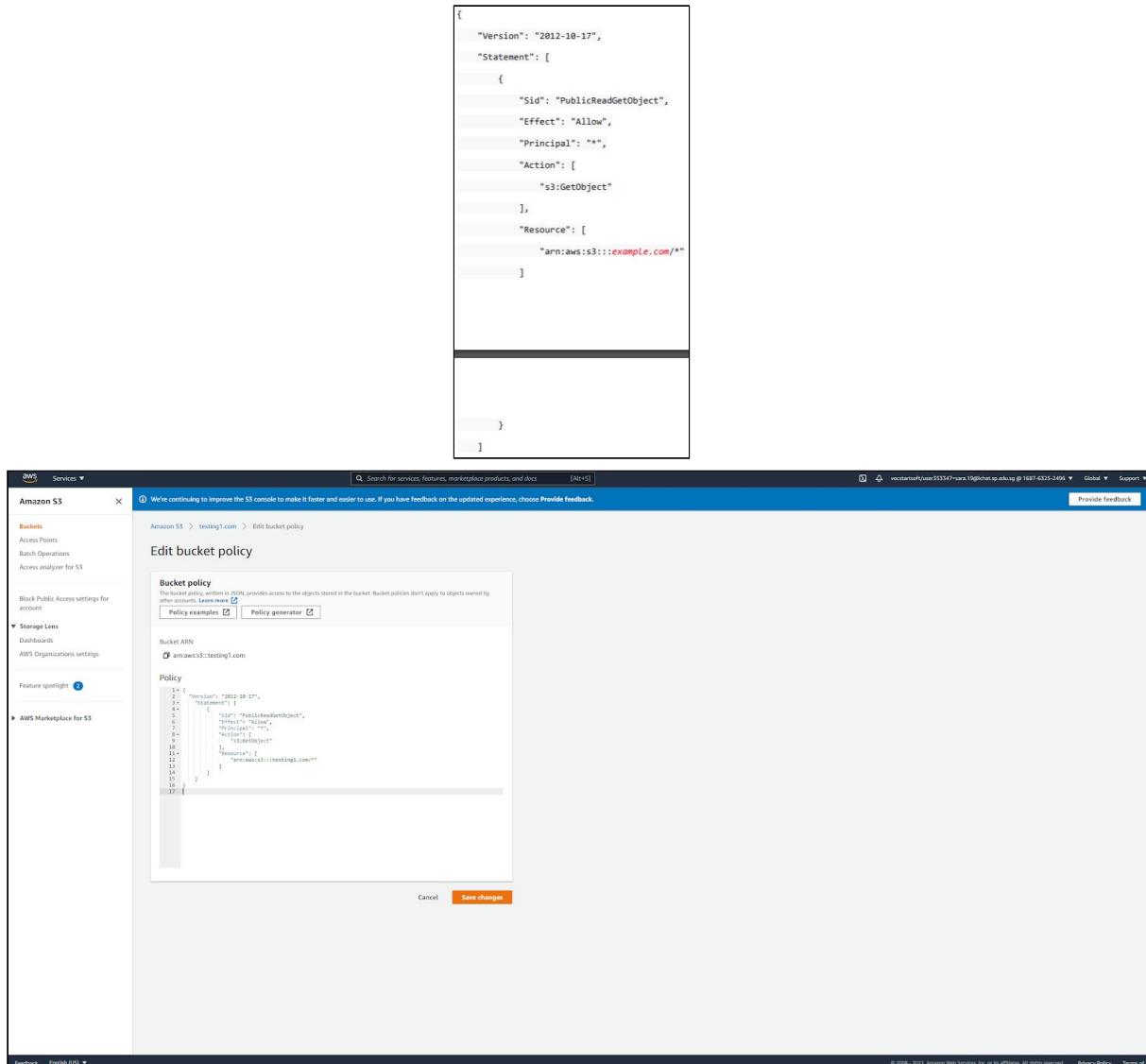
⚠️ Updating the Block Public Access settings for this bucket will affect this bucket and all objects within. This may result in some objects becoming public.

To confirm the settings, enter confirm in the field:

confirm

[Cancel](#) [Confirm](#)

Step 5: In the interests of availability which is one of the top 5 business impacts in Cloud Computing, we also decided to add a bucket policy to grant public read access to your bucket. With reference to the following bucket policy, we modified the `example.com` to our `testing1.com` and pasted the policy within the Bucket policy editor.



The screenshot shows the AWS S3 Bucket Policy Editor interface. On the left, there's a sidebar with navigation links like Buckets, Access Points, Batch Operations, and others. The main area is titled "Edit bucket policy". It contains a "Bucket policy" section with a detailed description and two buttons: "Policy examples" and "Policy generator". Below this is a "Policy ARN" field containing the ARN `arn:aws:s3:::testing1.com`. The central part of the screen is a large text area displaying the following JSON policy code:

```

{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "PublicReadGetObject",
            "Effect": "Allow",
            "Principal": "*",
            "Action": [
                "s3:GetObject"
            ],
            "Resource": [
                "arn:aws:s3:::testing1.com/*"
            ]
        }
    ]
}

```

At the bottom of the policy editor, there are "Cancel" and "Save changes" buttons. The status bar at the bottom right indicates the URL `vocstartsoft/user551547-sara.v@lightchat.up.edu.sg @ 167.43.25.2496` and provides links for "Global", "Support", "Provide feedback", and the AWS footer with copyright information.

Step 6: We were informed that the editing of the bucket policy is successful and that the contents of `testing1.com` is now “publicly available”.

The screenshot shows the AWS S3 console interface. On the left, the navigation pane includes 'Amazon S3', 'Backets', 'Access Points', 'Batch Operations', 'Access analyzer for S3', 'Storage Lens', 'Dashboards', 'AWS Organizations settings', 'Feature spotlight', and 'AWS Marketplace for S3'. The main content area has a green banner at the top stating 'Successfully edited bucket policy.' Below this, the 'testing1.com' bucket is selected. The 'Permissions' tab is active, showing a 'Permissions overview' section with a 'Public' access level indicator. Under 'Block public access (bucket settings)', it shows 'Block off public access' is 'Off'. The 'Bucket policy' section displays a JSON policy document:

```

{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "PublicReadGetObject",
      "Effect": "Allow",
      "Principal": "*",
      "Action": "S3.GetObject",
      "Resource": "arn:aws:s3:::testing1.com/*"
    }
  ]
}

```

Step 7: We managed to upload the individual HTML and Javascript files within the frontend folder into the S3 bucket which totals up to 3 pages as seen below.

The screenshot shows the AWS S3 console interface. The left navigation pane is identical to the previous screenshot. The main content area shows the 'Upload' page for the 'testing1.com' bucket. It displays a list of uploaded files and folders:

Name	Folder	Type	Size
admin.png	images/	image/png	2.0 KB
admin_manage_user.js	js/	text/javascript	0.1 KB
admin_update_user.js	js/	text/javascript	3.8 KB
axios.js	js/	text/javascript	13.4 KB
beadesignerstudologo.png	images/	image/png	107.3 KB
desktop.ini	images/	-	92.0 B
global.js	js/	text/javascript	1.0 KB
home.html	-	text/html	3.7 KB
login.html	-	text/html	5.4 KB
login.js	js/	text/javascript	2.8 KB

Below the file list, the 'Destination' section shows the path `s3://testing1.com`. The 'Destination details' section contains settings for 'Bucket Versioning', 'Default encryption', and 'Object Lock'. A note at the bottom encourages enabling Bucket Versioning to protect against unintentional overwriting or deleting objects.

We're continuing to improve the S3 console to make it faster and easier to use. If you have feedback on the updated experience, choose [Provide feedback](#).

[Amazon S3 > testing1.com > Upload](#)

Upload

Add the files and folders you want to upload to S3. To upload a file larger than 1600GB, use the AWS CLI, AWS SDK or Amazon S3 REST API. Learn more [\[?\]](#)

Drag and drop files and folders you want to upload here, or choose [Add files](#), or [Add folder](#).

Files and folders (25 Total, 212.8 KB)					
All files and folders in this table will be uploaded.					
Find by name					
Name	Folder	Type	Size	Actions	
logo.png	images/	image/png	537.0 B	Edit	
manage_submission.html	user/	text/html	5.5 kB	Edit	
manage_users.html	admin/	text/html	4.8 kB	Edit	
profile.html	user/	text/html	5.1 kB	Edit	
profile.html	admin/	text/html	4.7 kB	Edit	
profile.js	js/	text/javascript	1.6 kB	Edit	
register.html	-	text/html	4.6 kB	Edit	
register.js	js/	text/javascript	2.0 kB	Edit	
site.css	css/	text/css	954.0 B	Edit	
submit_design.html	user/	text/html	5.1 kB	Edit	

Destination

Destination: [s3://testing1.com](#)

Destination details

The following bucket settings impact new objects stored in the specified destination.

Bucket Versioning	When enabled, multiple variants of an object can be stored in the bucket to easily recover deleted or modified objects. Learn more [?]	Default encryption	When enabled, new objects stored in this bucket are automatically encrypted. Learn more [?]	Object Lock	When enabled, objects in this bucket might be prevented from being deleted or overwritten for a fixed amount of time or indefinitely. Learn more [?]
Disabled	Enabled	Disabled	Enabled	Disabled	Enabled

⚠ We recommend that you enable Bucket Versioning to help protect against unintentionally overwriting or deleting objects. Learn more [\[?\]](#)

[Enable Bucket Versioning](#)

[Feedback](#) [English \(US\) ▾](#)

[Search for services, features, marketplace products, and docs](#) [Alt+D]

vocaturo@ph-user353347:~\$@highchatup.edu.qg@1687-6325-2496 ~ Global Support [Provide feedback](#)

© 2008 - 2021, Amazon Web Services, Inc. or its affiliates. All rights reserved. [Privacy Policy](#) [Terms of Use](#)

We're continuing to improve the S3 console to make it faster and easier to use. If you have feedback on the updated experience, choose [Provide feedback](#).

[Amazon S3 > testing1.com > Upload](#)

Upload

Add the files and folders you want to upload to S3. To upload a file larger than 1600GB, use the AWS CLI, AWS SDK or Amazon S3 REST API. Learn more [\[?\]](#)

Drag and drop files and folders you want to upload here, or choose [Add files](#), or [Add folder](#).

Files and folders (12 Total, 212.8 KB)					
All files and folders in this table will be uploaded.					
Find by name					
Name	Folder	Type	Size	Actions	
submit_design.js	js/	text/javascript	2.9 kB	Edit	
update_design.html	user/	text/html	5.7 kB	Edit	
update_design.js	js/	text/javascript	4.7 kB	Edit	
update_user.html	admin/	text/html	4.9 kB	Edit	
user_manage_submission.html	js/	text/javascript	12.0 kB	Edit	

Destination

Destination: [s3://testing1.com](#)

Destination details

The following bucket settings impact new objects stored in the specified destination.

Bucket Versioning	When enabled, multiple variants of an object can be stored in the bucket to easily recover deleted or modified objects. Learn more [?]	Default encryption	When enabled, new objects stored in this bucket are automatically encrypted. Learn more [?]	Object Lock	When enabled, objects in this bucket might be prevented from being deleted or overwritten for a fixed amount of time or indefinitely. Learn more [?]
Disabled	Enabled	Disabled	Enabled	Disabled	Enabled

⚠ We recommend that you enable Bucket Versioning to help protect against unintentionally overwriting or deleting objects. Learn more [\[?\]](#)

[Enable Bucket Versioning](#)

[Feedback](#) [English \(US\) ▾](#)

[Search for services, features, marketplace products, and docs](#) [Alt+D]

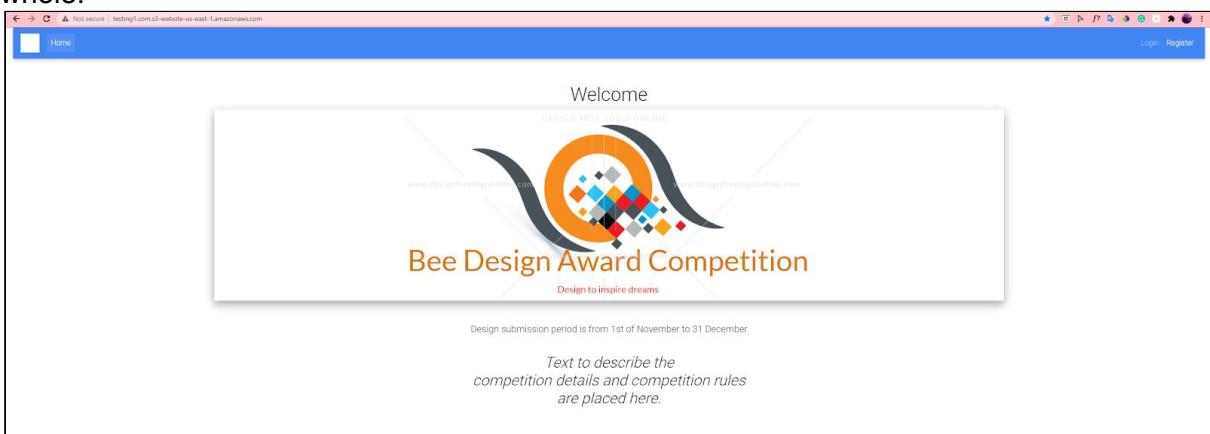
vocaturo@ph-user353347:~\$@highchatup.edu.qg@1687-6325-2496 ~ Global Support [Provide feedback](#)

© 2008 - 2021, Amazon Web Services, Inc. or its affiliates. All rights reserved. [Privacy Policy](#) [Terms of Use](#)

The screenshot shows the AWS S3 console interface. At the top, a green banner indicates "Upload successful" and "View details below". Below this, the page title is "Upload: status". A summary table shows the destination as "s3://testing1.com" with 25 files uploaded (100.00%) and 0 files failed. The "Files and folders" tab is selected, displaying a detailed list of 25 files and their properties. The files include various JavaScript, CSS, and image files, all marked as "Succeeded".

Name	Folder	Type	Size	Status	Error
admin.png	image/	image/png	2.0 kB	Succeeded	-
admin_manage_user.js	/js/	text/javascript	8.1 kB	Succeeded	-
admin_update_user.js	/js/	text/javascript	3.8 kB	Succeeded	-
awes.js	/js/	text/javascript	15.4 kB	Succeeded	-
beedesignawardlogo.png	image/	image/png	107.5 kB	Succeeded	-
desktop.ini	image/	-	92.0 kB	Succeeded	-
global.js	/js/	text/javascript	1.0 kB	Succeeded	-
home.html	-	text/html	3.7 kB	Succeeded	-
login.html	-	text/html	5.4 kB	Succeeded	-
login.js	/js/	text/javascript	2.8 kB	Succeeded	-

Step 8: We uploaded our ESDE CA1 Front-End files into AWS S3 and it was shown that the upload was successful. As shown previously, since only our static website was hosted successfully, we proceeded to make the dynamic portion in order to run it properly as a whole.



2. Deploy Existing Web Application on AWS (Dynamic)

In this section, we will be showing a step-by-step approach on how to configure the Ubuntu 18.04 Operating System on Amazon Web Services (AWS) Elastic Cloud Compute (EC2) instance and the deployment process of our existing web application using dynamic API.

Step 1: After logging into our AWS account, we navigated to the EC2 dashboard and clicked on the “Launch Instance” button.

The screenshot shows the AWS EC2 Dashboard. On the left, there's a sidebar with 'Instances' selected, showing options like 'Instances', 'Launch Templates', 'Spot Requests', and 'Scheduled Instances'. The main area has a 'Launch instance' card with the sub-section 'Scheduled events' showing 'US East (N. Virginia)' and 'No scheduled events'. To the right, there's an 'Explore AWS' sidebar with sections for 'Launch Custom AMIs with Fast Snapshot Restore (FSR)', 'Enable Best Price-Performance with AWS Graviton2', and 'Zones'.

Step 2: We choose to use Ubuntu Server 18.04 LTS (64-bit) for our Amazon Web Image (AMI).

The screenshot shows the 'Choose an Amazon Machine Image (AMI)' step of the EC2 wizard. It lists several AMI options:

- Ubuntu Server 20.04 LTS (HVM), SSD Volume Type** - ami-00ddb0e5626798373 (64-bit x86) / ami-074db80f0dc9b5f40 (64-bit Arm)
 - Select** button
 - 64-bit (x86)
 - 64-bit (Arm)
- Ubuntu Server 18.04 LTS (HVM), SSD Volume Type** - ami-00ddb0e5626798373 (64-bit x86) / ami-074db80f0dc9b5f40 (64-bit Arm)
 - Select** button
 - 64-bit (x86)
 - 64-bit (Arm)
- Microsoft Windows Server 2019 Base** - ami-0f5761c546ea1265a
 - Select** button
 - 64-bit (x86)
- Deep Learning AMI (Ubuntu 18.04) Version 39.0** - ami-03e0fdb8c9d235984
 - Select** button

Step 3: We chose to use t2.micro instance type, which is a 1vCPU and 1GB memory server offered by AWS as an eligible free tier. We proceeded to click the “next” button to configure instance details.

Step 2: Choose an Instance Type

Currently selected: t2.micro (- ECUs, 1 vCPUs, 2.5 GHz, 1 GiB memory, EBS only)

	Family	Type	vCPUs	Memory (GiB)	Instance Storage (GB)	EBS-Optimized Available	Network Performance	IPv6 Support
<input type="checkbox"/>	t2	t2.nano	1	0.5	EBS only	-	Low to Moderate	Yes
<input checked="" type="checkbox"/>	t2	t2.micro <small>Free tier eligible</small>	1	1	EBS only	-	Low to Moderate	Yes
<input type="checkbox"/>	t2	t2.small	1	2	EBS only	-	Low to Moderate	Yes
<input type="checkbox"/>	t2	t2.medium	2	4	EBS only	-	Low to Moderate	Yes
<input type="checkbox"/>	t2	t2.large	2	8	EBS only	-	Low to Moderate	Yes
<input type="checkbox"/>	t2	t2.xlarge	4	16	EBS only	-	Moderate	Yes

Cancel Previous Review and Launch Next: Configure Instance Details

Step 4: We applied the default details for the instance and proceeded to click the “next” button to add storage.

Step 3: Configure Instance Details

File systems i Add file system C Create new file system

Advanced Details

- Enclave i Enable
- Metadata accessible i Enabled
- Metadata version i V1 and V2 (token optional)
- Metadata token response hop limit i 1
- User data i As text As file Input is already base64 encoded
(Optional)

Cancel Previous Review and Launch Next: Add Storage

Step 5: We applied the default details for the storage and proceeded to click the “next” button to add tags.

Step 4: Add Storage

Your instance will be launched with the following storage device settings. You can attach additional EBS volumes and instance store volumes to your instance, or edit the settings of the root volume. You can also attach additional EBS volumes after launching an instance, but not instance store volumes. [Learn more](#) about storage options in Amazon EC2.

Volume Type	Device	Snapshot	Size (GiB)	Volume Type	IOPS	Throughput (MB/s)	Delete on Termination	Encryption
Root	/dev/sda1	snap-0b071e09e1285af85	8	General Purpose SSD (gp2)	100 / 3000	N/A	<input checked="" type="checkbox"/>	Not Encrypt

Add New Volume

Free tier eligible customers can get up to 30 GB of EBS General Purpose (SSD) or Magnetic storage. [Learn more](#) about free usage tier eligibility and usage restrictions.

Cancel Previous Review and Launch Next: Add Tags

Step 6: We applied the default for the tags and proceeded to click the “next” button to configure the security group.

Step 5: Add Tags

A tag consists of a case-sensitive key-value pair. For example, you could define a tag with key = Name and value = Webserver. A copy of a tag can be applied to volumes, instances or both. Tags will be applied to all instances and volumes. [Learn more](#) about tagging your Amazon EC2 resources.

Key	(128 characters maximum)	Value	(256 characters maximum)	Instances	Volumes
This resource currently has no tags					

Choose the Add tag button or [click to add a Name tag](#). Make sure your [IAM policy](#) includes permissions to create tags.

Add Tag (Up to 50 tags maximum)

Cancel Previous Review and Launch Next: Configure Security Group

Step 7: We applied the default for the security group, which is 0.0.0.0/0 as a source out of convenience as our IP addresses are prone to changes in different environments. We proceeded to click the “review and launch” button to launch the EC2 instance.

Step 6: Configure Security Group

A security group is a set of firewall rules that control the traffic for your instance. On this page, you can add rules to allow specific traffic to reach your instance. For example, if you want to set up a web server and allow Internet traffic to reach your instance, add rules that allow unrestricted access to the HTTP and HTTPS ports. You can create a new security group or select from an existing one below. [Learn more](#) about Amazon EC2 security groups.

Assign a security group: Create a new security group Select an existing security group

Security group name:

Description:

Type	Protocol	Port Range	Source	Description
SSH	TCP	22	Custom 0.0.0.0/0	e.g. SSH for Admin Desktop

[Add Rule](#)

⚠️ Warning
Rules with source of 0.0.0.0/0 allow all IP addresses to access your instance. We recommend setting security group rules to allow access from known IP addresses only.

[Cancel](#) [Previous](#) [Review and Launch](#)

Step 8: We did a final check of the settings for the instance type before proceeding to click the “launch” button to launch the EC2 instance.

Step 7: Review Instance Launch

Ubuntu Server 18.04 LTS (HVM), SSD Volume Type - ami-00ddb0e5626798373

Free tier eligible Ubuntu Server 18.04 LTS (HVM).EBS General Purpose (SSD) Volume Type. Support available from Canonical (<http://www.ubuntu.com/cloud/services>).
Root Device Type: ebs Virtualization type: hvm

Instance Type [Edit instance type](#)

Instance Type	ECUs	vCPUs	Memory (GiB)	Instance Storage (GB)	EBS-Optimized Available	Network Performance
t2.micro	-	1	1	EBS only	-	Low to Moderate

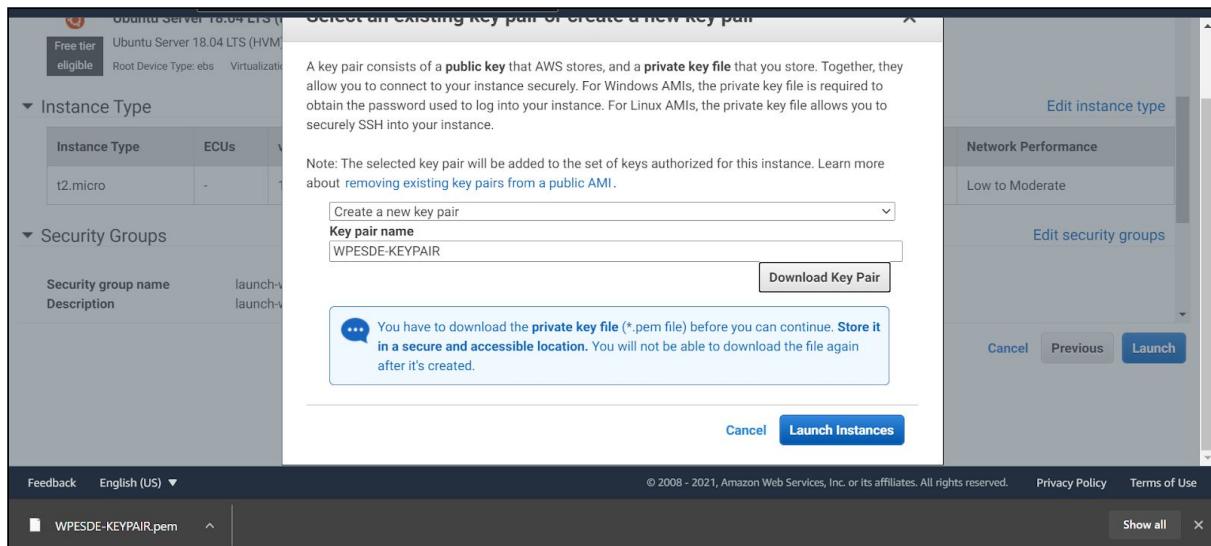
Security Groups [Edit security groups](#)

Security group name	Description
launch-wizard-1	launch-wizard-1 created 2021-01-25T11:58:46.131+08:00

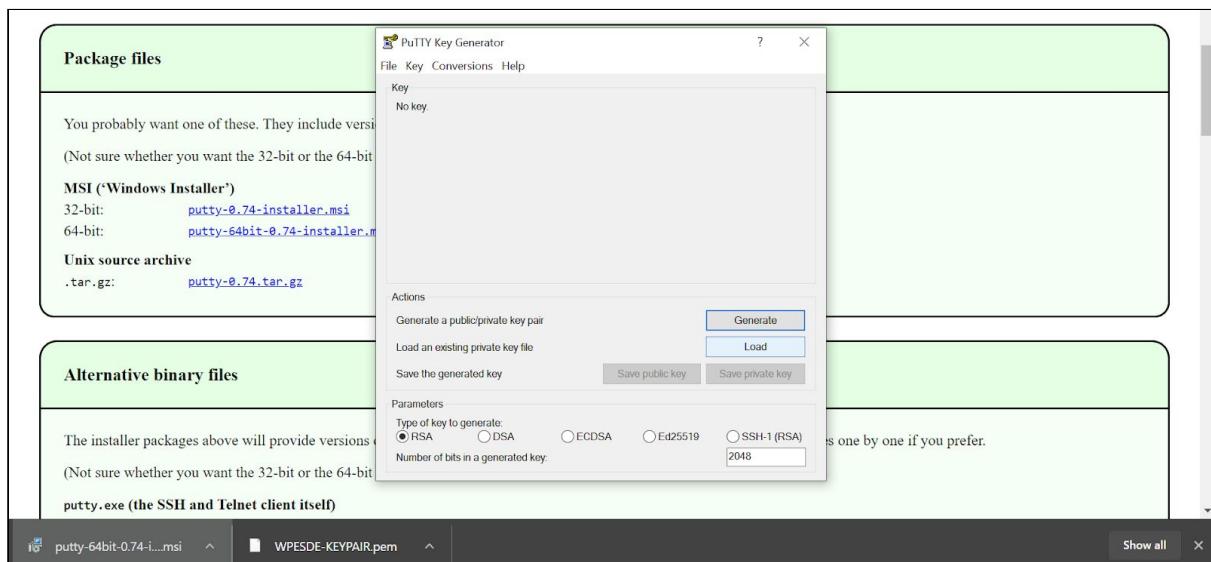
Type	Protocol	Port Range	Source	Description
------	----------	------------	--------	-------------

[Cancel](#) [Previous](#) [Launch](#)

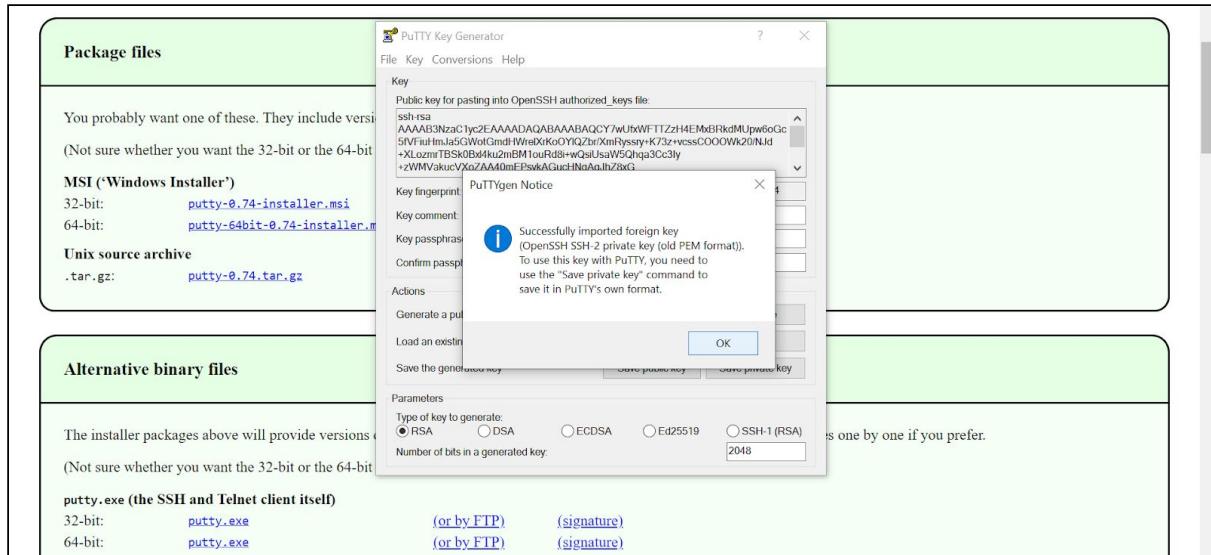
Step 9: We were prompted to create and download a private key, which allows us to connect with the EC2 instances using SSH. From the dropdown menu, we selected “create a new key pair” and named it “WPESDE-KEYPAIR” before clicking the “download key pair” button.



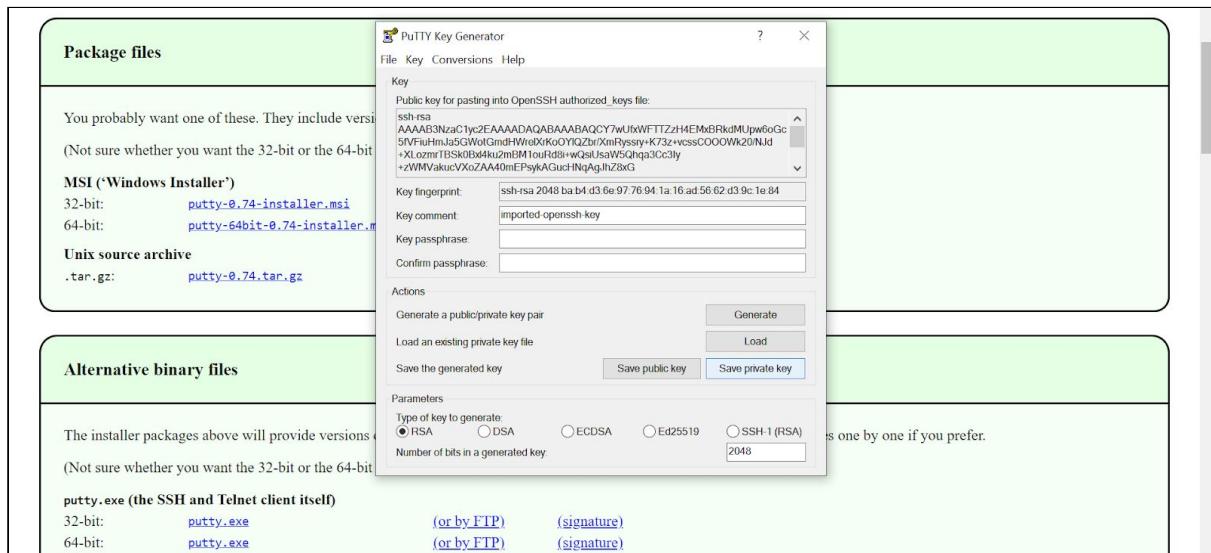
Step 10: We chose the 64-bit MSI to install putty which also installs puttygen. As the key pair downloaded previously is of .pem, we need to generate a private key to login to the remote instance as putty does not accept that kind of extension. After opening puttygen, we clicked on the “file” button then “load private key” button to convert it.



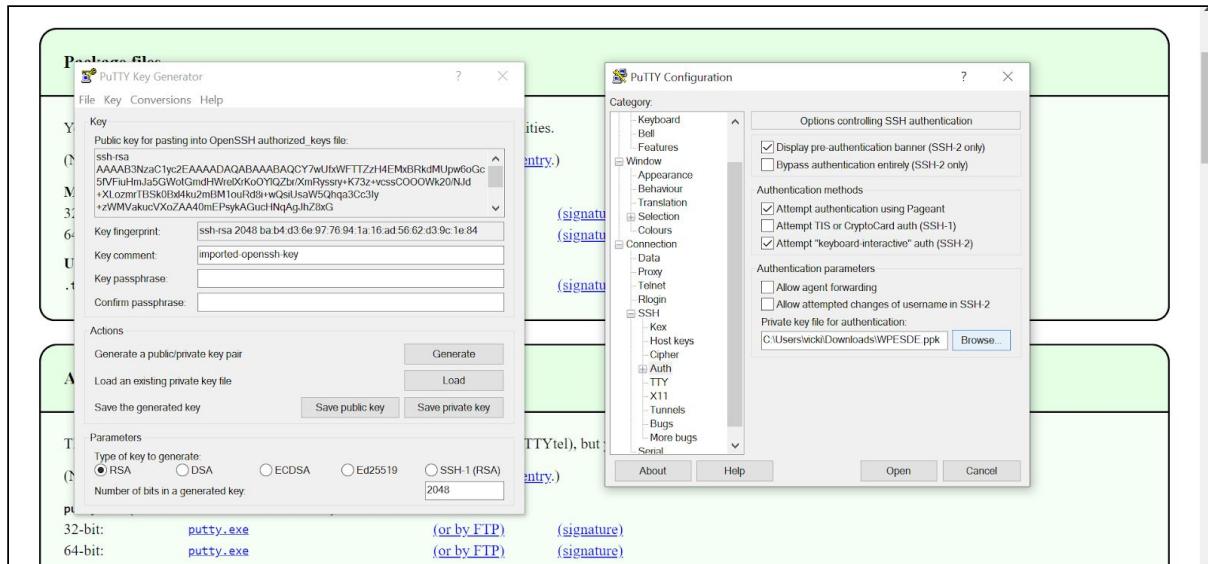
Step 11: We were informed that our private key of RSA type has been generated and proceeded to click “ok” to close off the notice.



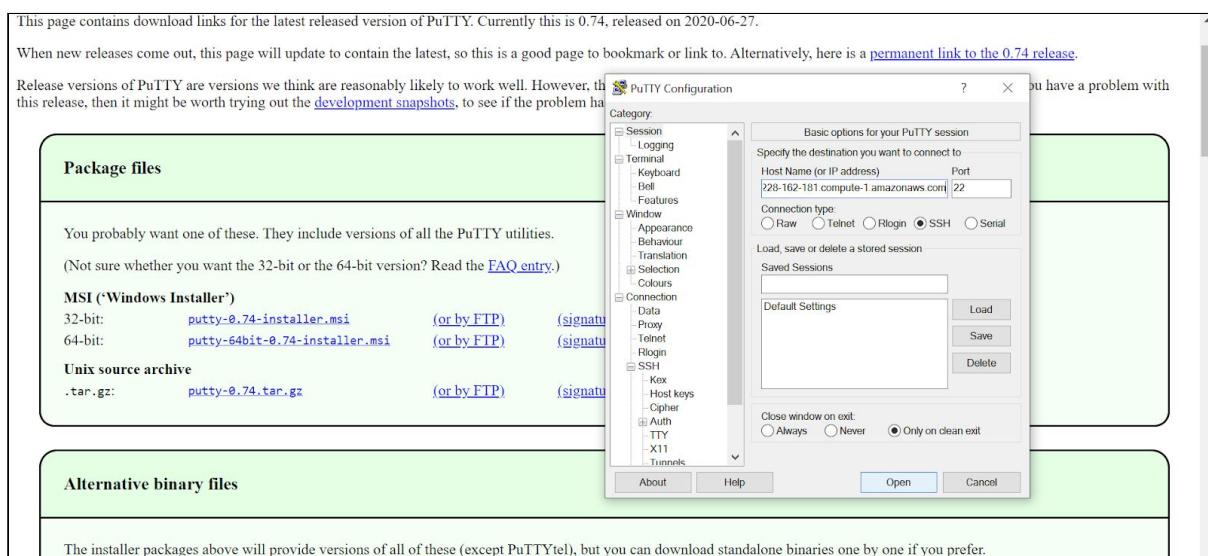
Step 12: We clicked the “save private key” button to save it to a file of .ppk extension and named it “WPESDE”. We also opted for no passphrase so that it will be less troublesome as manual human intervention is not needed to log on or copy files to an instance.



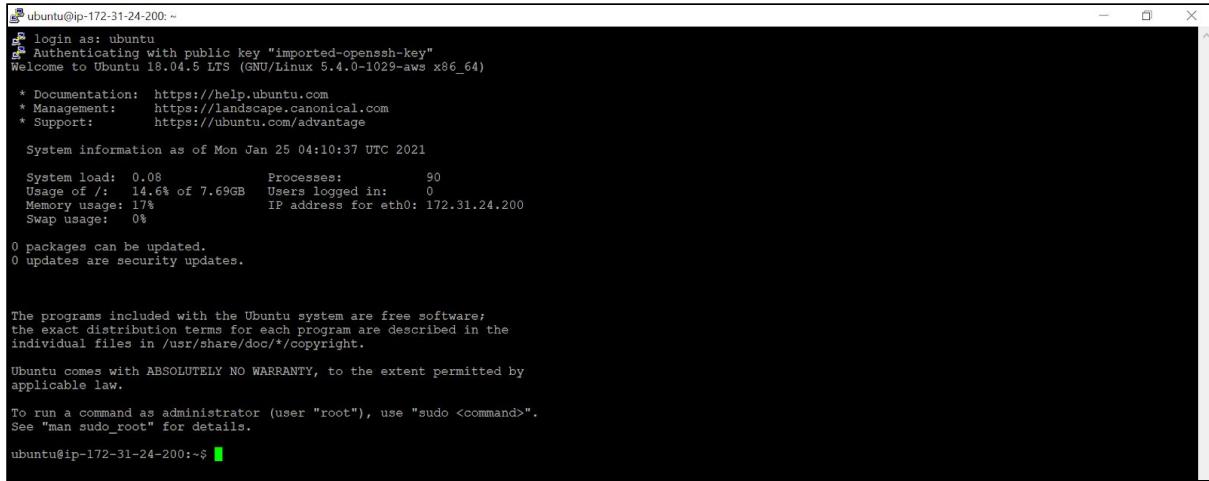
Step 13: After opening putty, we clicked on the “connection” button then the “SSH” button then the “Auth” button and lastly the “browse” button to search for the previously downloaded .ppk file and uploaded it to provide a key file for authentication.



Step 14: We clicked on the “session” button to fill in the details for the putty session. We proceeded to fill in the host name (or IP address) field with our public DNS stated in AWS for the EC2 instance.



Step 15: We clicked on the “open” button at the bottom of the previous putty session configuration page and successfully logged into putty with the username of “ubuntu”.



```
ubuntu@ip-172-31-24-200: ~
SSH login as: ubuntu
Authenticating with public key "imported-openssh-key"
Welcome to Ubuntu 18.04.5 LTS (GNU/Linux 5.4.0-1029-aws x86_64)

 * Documentation: https://help.ubuntu.com
 * Management: https://landscape.canonical.com
 * Support: https://ubuntu.com/advantage

System information as of Mon Jan 25 04:10:37 UTC 2021

System load: 0.08      Processes:         90
Usage of /: 14.6% of 7.69GB   Users logged in:    0
Memory usage: 1%          IP address for eth0: 172.31.24.200
Swap usage: 0%

0 packages can be updated.
0 updates are security updates.

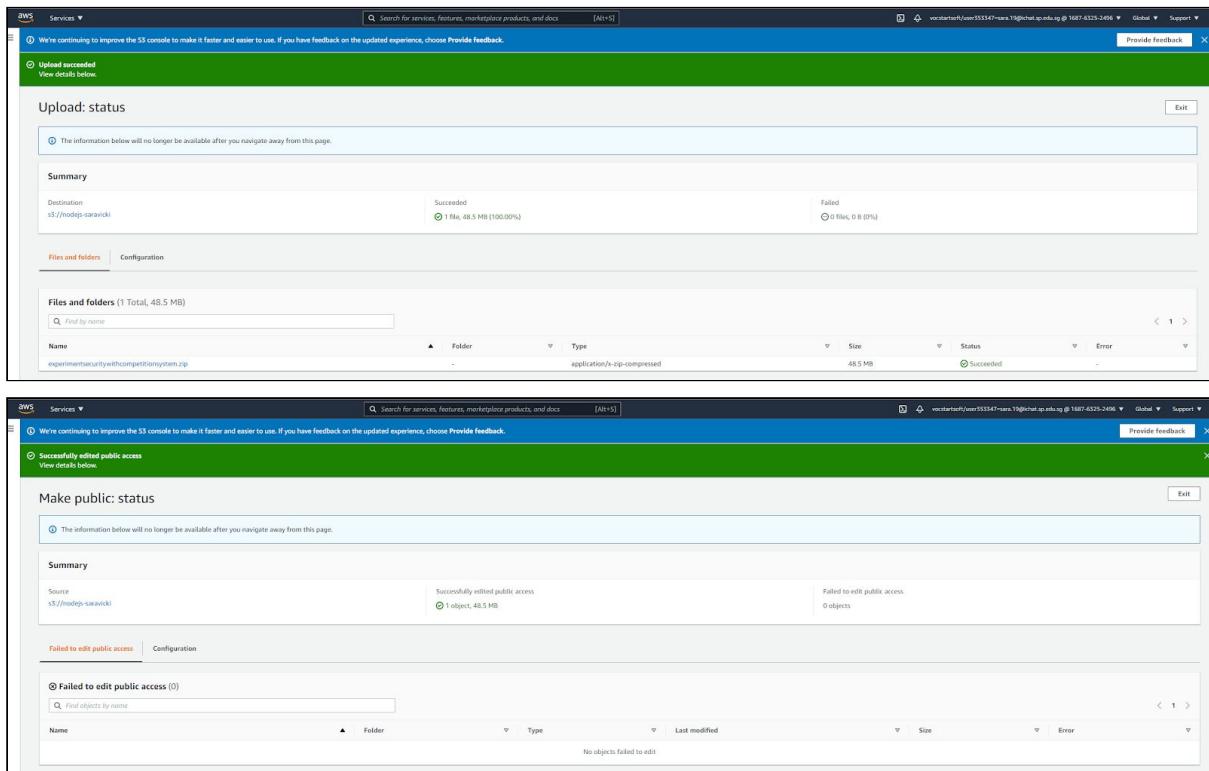
The programs included with the Ubuntu system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/*copyright.

Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by
applicable law.

To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.

ubuntu@ip-172-31-24-200:~$
```

Step 16: We changed the status of our previously created S3 bucket that we used to upload our CA1 zipped folder into and changed the status of it to enable public access. We were notified that the status edit is successful.



The screenshots show the AWS S3 console interface. The top screenshot displays a successful file upload of 'status' to the 's3://nodejs-saravicki' bucket. The bottom screenshot shows the successful editing of the object's public access status, changing it from 'Failed' to 'Succeeded'.

Step 17: We verified that the node application is copied to AWS by entering “wget https://nodejs-saravicki.s3.amazonaws.com/experimentsecuritywithcompetitionsystem.zip”.

```

root@raspberrypi: ~#
[1]+ 0 wget https://nodejs-saravicki.s3.amazonaws.com/experimentsecuritywithcompetitionsystem.zip

```

Step 18: We saw that the contents are fetched accordingly from our home directory. This shows that we managed to upload our zipped folder to the S3 bucket successfully.

```

root@raspberrypi: ~#
[1]+ 0 wget https://nodejs-saravicki.s3.amazonaws.com/experimentsecuritywithcompetitionsystem.zip

```

Step 19: We removed a zipped folder consisting of both frontend and backend files which we uploaded incorrectly previously by entering “rm CA1.zip”. We proceeded to unzip the recently uploaded correct backend folder by entering “unzip experimentsecuritywithcompetitionsystem.zip”.

```

root@raspberrypi: ~#
[1]+ 0 rm CA1.zip
[2]+ 0 unzip experimentsecuritywithcompetitionsystem.zip

```

Step 20: We saw that the contents are extracted accordingly from the backend folder. We also installed the node version manager (nvm) by entering “`nvm install 12.18.3`” and checked the node version previously installed by entering “`node-v`”. We proceeded to run the command “`pm2 start /home/ubuntu/experimentsecuritywithcompetition/index.js`” to start the server automatically after each restart.

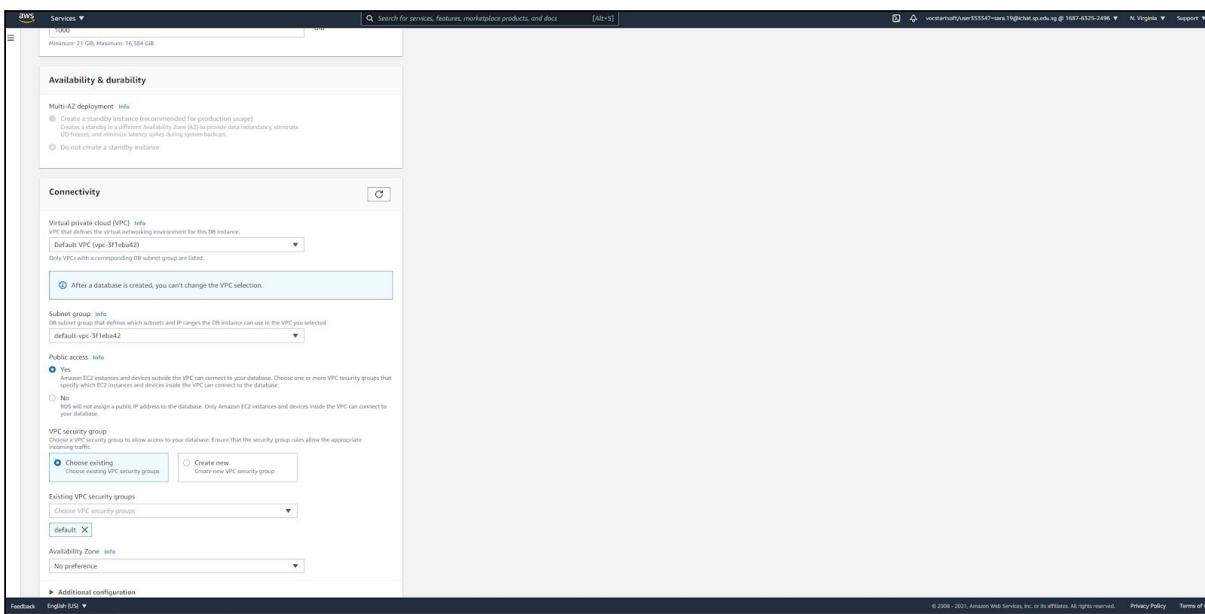
Step 21: We saw the following message that “this project provides only backend API support” after opening our browser at port 5000 and entering our public IP address and domain name stated in AWS for the EC2 instance. This shows that our backend web API is successfully hosted on EC2.

In this section, we will be showing a step-by-step approach on how to use Amazon RDS to create a MySQL DB Instance.

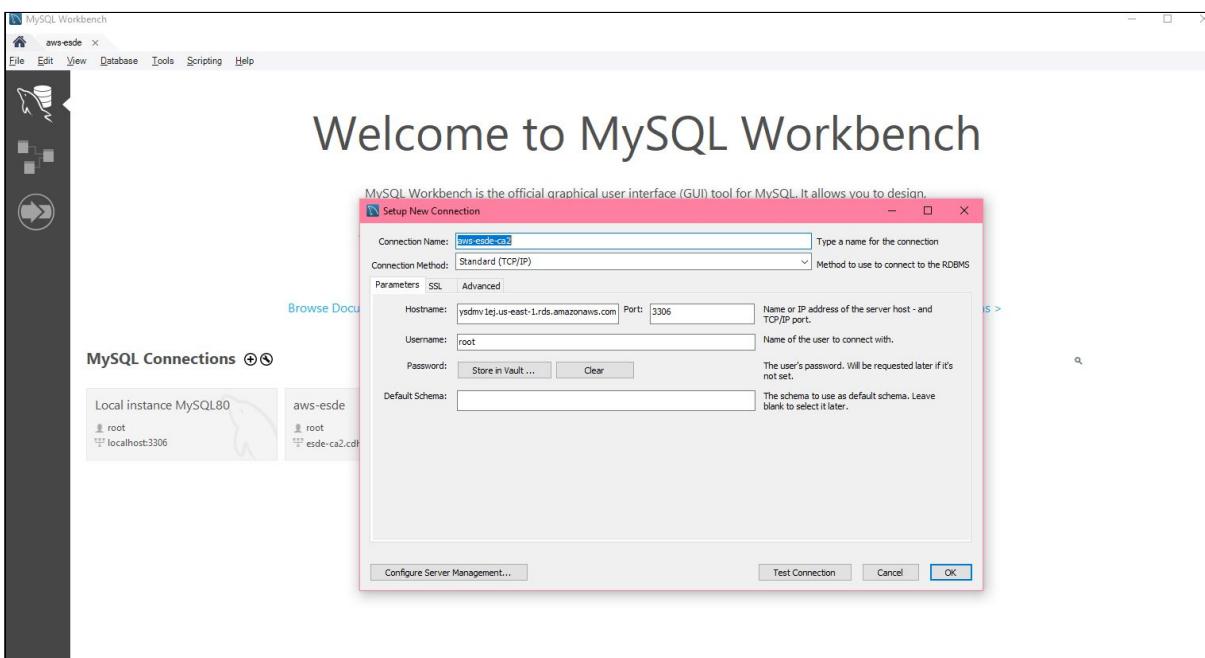
Although DynamoDB makes use of partitions to store data, we use Amazon RDS as it has bigger read-write capacity and is more flexible in terms of storage and faster in retrieving the datasets queried by us.

Step 22: After logging into our AWS account, we navigated to the RDS dashboard and clicked on the “Create database” button. For our engine options, we clicked the MySQL icon and selected the Free Tier template which is the most cost effective option to help us save up our limited \$50 credits. We opted for the default settings which include public access and existing VPC security group, which allowed us to directly connect to the database from our own devices by allocating an IP address just for our database instance. We proceeded to give our database a unique name “esde-ca2-db”

The screenshot shows the AWS RDS "Create database" wizard. The first step, "Create database", is displayed. Under "Choose a database creation method", the "Standard create" option is selected. In the "Engine options" section, the "MySQL" icon is chosen. The "Edition" is set to "MySQL Community". The "Version" dropdown shows "MySQL 5.7.22". The "Template" section has "Free tier" selected. Other sections like "DB instance identifier", "Master password", and "DB instance class" are partially visible below.



Step 23: As it will be more useful to access a RDS instance to remotely send SQL commands, we need to connect AWS to MySQL. After opening MySQL Workbench, we clicked the “+” icon next to “MySQL Connections” to set up a new connection. We proceeded to fill in the hostname, port and username fields with the endpoint, port number and principal username stated at AWS RDS instance.



Step 24: We opened our SQL codes dump for CA1 and executed it by clicking the “lightning bolt” icon at the top of the page. We see that the files are successfully inserted into the created database.

The screenshot shows the MySQL Workbench interface with a pink header bar. The main window has two tabs: 'Query 1' and 'new_competition_system_security_concept_db.sql'. The 'Query 1' tab contains the SQL code for creating the database and its schema. The 'Output' tab displays the results of the execution, showing 60 rows affected for various SET statements like `SET TIME_ZONE` and `SET FOREIGN_KEY_CHECKS`.

#	Action	Time	Message	Duration / Fetch
53	/'40103 SET TIME_ZONE=@OLD_TIME_ZONE '/	17:23:47	0 row(s) affected	0.250 sec
54	/'40101 SET SQL_MODE=@OLD_SQL_MODE '/	17:23:47	0 row(s) affected	0.235 sec
55	/'40014 SET FOREIGN_KEY_CHECKS=@OLD_FOREIGN_KEY_CHECKS '/	17:23:48	0 row(s) affected	0.234 sec
56	/'40014 SET UNIQUE_CHECKS=@OLD_UNIQUE_CHECKS '/	17:23:48	0 row(s) affected	0.234 sec
57	/'40101 SET CHARACTER_SET_CLIENT=@OLD_CHARACTER_SET_CLIENT '/	17:23:48	0 row(s) affected	0.250 sec
58	/'40101 SET CHARACTER_SET_RESULTS=@OLD_CHARACTER_SET_RESULTS '/	17:23:48	0 row(s) affected	0.250 sec
59	/'40101 SET COLLATION_CONNECTION=@OLD_COLLATION_CONNECTION '/	17:23:49	0 row(s) affected	0.234 sec
60	/'40111 SET SQL_NOTES=@OLD_SQL_NOTES '/	17:23:49	0 row(s) affected	0.250 sec

Step 25: We used Ubuntu to modify the connection info string for host in databaseConfig.js with that of our RDS instance.

The terminal window shows the modification of the 'databaseConfig.js' file. The host value is changed from 'localhost' to 'aws-easte1-1.rds.amazonaws.com'. After saving the file, the command 'node app' is run, and the application starts successfully.

```
git clone https://github.com/awesomeness/competition-security-concept-db.git
cd competition-security-concept-db
cd database
vi databaseConfig.js
// Change host to RDS instance
module.exports = {
  pool: mysql.createPool({
    host: "aws-easte1-1.rds.amazonaws.com",
    port: 3306,
    user: "root",
    password: "password",
    database: "competition_security_concept_db"
  })
}

node app
```

Step 26: We can now perform a login action as Rita on the frontend login page. This shows that our website is hosted on AWS as we are able to access our own submissions using the account of user id 100.

The top screenshot shows a 'Login' page with the following note:

- Public can register as a user
- The database has been **seeded** with user test data.
- user:rita@designer.com password:password
normal user role
- user:robert@admin.com password:password
admin user role
- Inspect the user table for more information on all the **seeded** user test records
- Use the SQL script `SQLscript_CreateDbAndTables_bee_competition_system_security_concept_db.sql` in the MySQL workbench to prepare a new database and necessary tables.
- Use node `seeddata.js` to run the script file to provide test data and test files at cloudinary.
If you are recreating the database. Please drop the existing schema first. Also manually delete all the test files at Cloudinary.

3. Troubleshoot and Test during Deployment Stage

3.1 Changing Modules from Bcrypt to BcryptJS

At first, we only managed to deploy the dynamic website into AWS. However, we face some issues in the backend side thus we are unable to do any login in the webpage. Basically, the front end is working but the backend is not working. After troubleshooting many times, we

realised there were some issues with our bcrypt modules, thus we replaced our bcrypt with bcryptjs instead. The following image is the process of uninstalling bcrypt modules.

```

ubuntu@ip-172-31-27-155:~/ESDE_CAI_DITFT2A23_SaraGolkaran_P1935657_YapMinRuVicki_P1904556/SimulatedFrontEnd
ywithcompetitionssystem/node_modules/node-pre-gyp/lib/util/napi.js:80
    throw new Error(
      ^

Error: The N-API version of this Node instance is 1. This module supports N-API version(s) 3.
This Node instance cannot run this module.
  at Object.exports.validate_package_json (/home/ubuntu/ESDE_CAI_DITFT2A23_SaraGolkaran_P1935657_YapMinRuVicki_P1904556/experimentssecuritywithcompetitionssystem/node_modules/node-pre-gyp/lib/util/napi.js:80:19)
  at Object.validateBinding (/home/ubuntu/ESDE_CAI_DITFT2A23_SaraGolkaran_P1935657_YapMinRuVicki_P1904556/experimentssecuritywithcompetitionssystem/node_modules/node-pre-gyp/lib/util/versioning.js:229:10)
  at Object.exports.find (/home/ubuntu/ESDE_CAI_DITFT2A23_SaraGolkaran_P1935657_YapMinRuVicki_P1904556/experimentssecuritywithcompetitionssystem/node_modules/node-pre-gyp/lib/pre-binding.js:21:15)
  at Object.anonymous> (/home/ubuntu/ESDE_CAI_DITFT2A23_SaraGolkaran_P1935657_YapMinRuVicki_P1904556/experimentssecuritywithcompetitionssystem/node_modules/bcrypt/bcrypt.js:5:27)
  at Module._compile (module.js:652:30)
  at Object.Module._extensions..js (module.js:663:10)
  at Module.load (module.js:555:32)
  at tryModuleLoad (module.js:505:12)
  at Function.Module.load (module.js:497:3)
  at Module.require (module.js:596:17)
ubuntu@ip-172-31-27-155:~/ESDE_CAI_DITFT2A23_SaraGolkaran_P1935657_YapMinRuVicki_P1904556/experimentssecuritywithcompetitionssystem$ npm uninstall bcrypt
npm WARN ws@7.2.5 requires a peer of bufferutil@^4.0.1 but none is installed. You must install peer dependencies yourself.
npm WARN ws@7.2.5 requires a peer of utf-8-validate@^5.0.2 but none is installed. You must install peer dependencies yourself.
npm WARN experimentssecuritywithdesignerstudio@1.0.0 No repository field.
npm WARN optional SKIPPING OPTIONAL DEPENDENCY: fsevents@2.1.3: wanted {"os":"darwin","arch":"any"} (current: {"os":"linux","arch":"x64"})
npm WARN optional SKIPPING OPTIONAL DEPENDENCY: fsevents@2.3.1 (node_modules/pm2/node_modules/fsevents):
npm WARN optional SKIPPING OPTIONAL DEPENDENCY: Unsupported platform for fsevents@2.3.1 wanted {"os":"darwin","arch":"any"} (current: {"os":"linux","arch":"x64"})
npm WARN optional SKIPPING OPTIONAL DEPENDENCY: Unsupported platform for fsevents@2.3.17 wanted {"os":"darwin","arch":"any"} (current: {"os":"linux","arch":"x64"})

removed 19 packages in 9.359s
ubuntu@ip-172-31-27-155:~/ESDE_CAI_DITFT2A23_SaraGolkaran_P1935657_YapMinRuVicki_P1904556/experimentssecuritywithcompetitionssystem$ 
# login as: ubuntu
# Authenticating with public Key "imported-openssh-key"
Welcome to Ubuntu 18.04.5 LTS (GNU/Linux 5.4.0-1029-aws x86_64)

 * Documentation: https://help.ubuntu.com
 * Management: https://landscape.canonical.com
 * Support: https://ubuntu.com/advantage

 System information as of Thu Jan 21 06:19:44 UTC 2021

 System load: 0.17 Processes: 107
 Usage of /: 33.8% of 7.69GB Users logged in: 1
 Memory usage: 27% IP address for eth0: 172.31.27.155
 Swap usage: 0%

```

The following image is the process of installing bcryptjs for the upper part of the image. The second part of the image is the “sudo nano” command to edit the authcontroller.js file directly in the ubuntu terminal without opening Visual Studio Code. As the authcontroller.js file used bcrypt modules, thus we also need to replace it with bcryptjs.

```

ubuntu@ip-172-31-27-155:~/ESDE_CAI_DITFT2A23_SaraGolkaran_P1935657_YapMinRuVicki_P1904556/SimulatedFrontEnd
Introducing self-healing high availability clusters in MicroK8s.
  simple, hardened, Kubernetes for production, from RaspberryPi to DC.

https://microk8s.io/high-availability

7 packages can be updated.
7 of these updates are security updates.
To see these additional updates run: apt list --upgradable

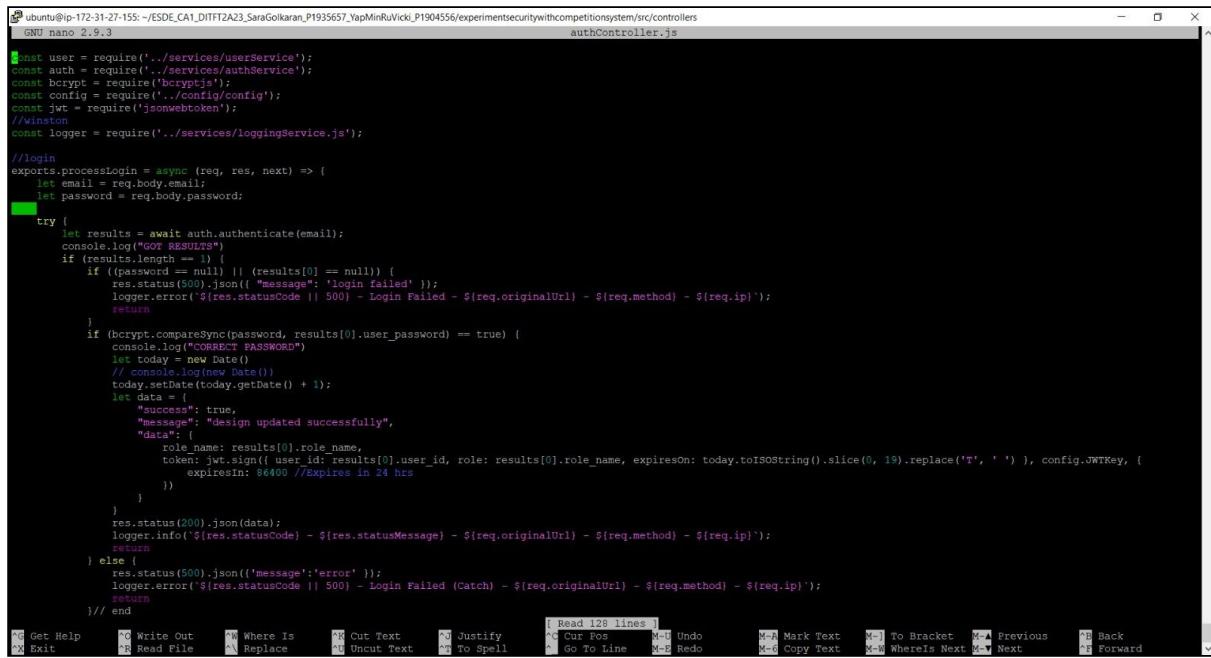
New release '20.04.1 LTS' available.
Run 'do-release-upgrade' to upgrade to it.

*** System restart required ***
Last login: Thu Jan 21 06:17:18 2021 from 129.126.58.249
ubuntu@ip-172-31-27-155:~/ESDE_CAI_DITFT2A23_SaraGolkaran_P1935657_YapMinRuVicki_P1904556/experimentssecuritywithcompetitionssystem$ npm install bcryptjs
npm WARN ws@7.2.5 requires a peer of bufferutil@^4.0.1 but none is installed. You must install peer dependencies yourself.
npm WARN ws@7.2.5 requires a peer of utf-8-validate@^5.0.2 but none is installed. You must install peer dependencies yourself.
npm WARN experimentssecuritywithdesignerstudio@1.0.0 No repository field.
npm WARN optional SKIPPING OPTIONAL DEPENDENCY: fsevents@2.1.3 (node_modules/fsevents):
npm WARN optional SKIPPING OPTIONAL DEPENDENCY: Unsupported platform for fsevents@2.1.3: wanted {"os":"darwin","arch":"any"} (current: {"os":"linux","arch":"x64"})
npm WARN optional SKIPPING OPTIONAL DEPENDENCY: fsevents@2.3.1 (node_modules/pm2/node_modules/fsevents):
npm WARN optional SKIPPING OPTIONAL DEPENDENCY: Unsupported platform for fsevents@2.3.1: wanted {"os":"darwin","arch":"any"} (current: {"os":"linux","arch":"x64"})

+ bcryptjs@2.4.3
added 1 package in 9.708s
ubuntu@ip-172-31-27-155:~/ESDE_CAI_DITFT2A23_SaraGolkaran_P1935657_YapMinRuVicki_P1904556/experimentssecuritywithcompetitionssystem$ cd src
ubuntu@ip-172-31-27-155:~/ESDE_CAI_DITFT2A23_SaraGolkaran_P1935657_YapMinRuVicki_P1904556/experimentssecuritywithcompetitionssystem$ srcs ls
assets bootstrap.js config controllers helpers middlewares routes.js services validation views
ubuntu@ip-172-31-27-155:~/ESDE_CAI_DITFT2A23_SaraGolkaran_P1935657_YapMinRuVicki_P1904556/experimentssecuritywithcompetitionssystem$ cd authController
-bash: cd: authController: No such file or directory
ubuntu@ip-172-31-27-155:~/ESDE_CAI_DITFT2A23_SaraGolkaran_P1935657_YapMinRuVicki_P1904556/experimentssecuritywithcompetitionssystem$ cd middleware
-bash: cd: middleware: No such file or directory
ubuntu@ip-172-31-27-155:~/ESDE_CAI_DITFT2A23_SaraGolkaran_P1935657_YapMinRuVicki_P1904556/experimentssecuritywithcompetitionssystem$ cd middlewares
ubuntu@ip-172-31-27-155:~/ESDE_CAI_DITFT2A23_SaraGolkaran_P1935657_YapMinRuVicki_P1904556/experimentssecuritywithcompetitionssystem$ cd middlewares$ sudo nano authController
ubuntu@ip-172-31-27-155:~/ESDE_CAI_DITFT2A23_SaraGolkaran_P1935657_YapMinRuVicki_P1904556/experimentssecuritywithcompetitionssystem$ cd middlewares$ cd ..
ubuntu@ip-172-31-27-155:~/ESDE_CAI_DITFT2A23_SaraGolkaran_P1935657_YapMinRuVicki_P1904556/experimentssecuritywithcompetitionssystem$ cd controllers
ubuntu@ip-172-31-27-155:~/ESDE_CAI_DITFT2A23_SaraGolkaran_P1935657_YapMinRuVicki_P1904556/experimentssecuritywithcompetitionssystem$ cd controllers$ sudo nano userController.js
ubuntu@ip-172-31-27-155:~/ESDE_CAI_DITFT2A23_SaraGolkaran_P1935657_YapMinRuVicki_P1904556/experimentssecuritywithcompetitionssystem$ cd controllers$ sudo nano authController.js
ubuntu@ip-172-31-27-155:~/ESDE_CAI_DITFT2A23_SaraGolkaran_P1935657_YapMinRuVicki_P1904556/experimentssecuritywithcompetitionssystem$ cd controllers$ sudo nano authController.js
ubuntu@ip-172-31-27-155:~/ESDE_CAI_DITFT2A23_SaraGolkaran_P1935657_YapMinRuVicki_P1904556/experimentssecuritywithcompetitionssystem$ cd controllers$ cd ..
ubuntu@ip-172-31-27-155:~/ESDE_CAI_DITFT2A23_SaraGolkaran_P1935657_YapMinRuVicki_P1904556/experimentssecuritywithcompetitionssystem$ pm2 restart index.js
Use --update-env to update environment variables
[EM2] Applying action restartProcessId on app [index.js](ids: 0)
[EM2] [index](0) ✓

```

The following image is editing the authcontroller.js file. We replace bcrypt with bcrypt.js.



```

ubuntu@ip-172-31-27-155:~/ESDE_CAs1-DITFT2A23_SaraGokaran_P1935657_YapMinRuVicki_P1904556/experimentsystem/src/controllers
GNU nano 2.9.3
authController.js

const user = require('../services/userService');
const auth = require('../services/authService');
const bcrypt = require('bcryptjs');
const config = require('../config');
const jwt = require('jsonwebtoken');
//winston
const logger = require('../services/loggingService.js');

//login
exports.processLogin = async (req, res, next) => {
    let email = req.body.email;
    let password = req.body.password;
    try {
        let results = await auth.authenticate(email);
        console.log("GOT RESULTS")
        if (results.length == 1) {
            if ((password == null) || (results[0] == null)) {
                res.status(500).json({ "message": 'login failed' });
                logger.error(`${res.statusCode} || 500 - Login Failed - ${req.originalUrl} - ${req.method} - ${req.ip}`);
                return;
            }
            if (bcrypt.compareSync(password, results[0].user_password) == true) {
                console.log("CORRECT PASSWORD")
                let today = new Date();
                // console.log(new Date())
                today.setDate(today.getDate() + 1);
                let data = {
                    "success": true,
                    "message": "design updated successfully",
                    "data": {
                        role_name: results[0].role_name,
                        token: jwt.sign({ user_id: results[0].user_id, role: results[0].role_name, expiresOn: today.toISOString().slice(0, 19).replace('T', ' ') }, config.JWTKey, {
                            expiresIn: 86400 //Expires in 24 hrs
                        })
                    }
                }
                res.status(200).json(data);
                logger.info(`${res.statusCode} - ${res.statusMessage} - ${req.originalUrl} - ${req.method} - ${req.ip}`);
                return;
            } else {
                res.status(500).json({ "message": "error" });
                logger.error(`${res.statusCode} || 500 - Login Failed (Catch) - ${req.originalUrl} - ${req.method} - ${req.ip}`);
                return;
            }
        }
    } catch {
        res.status(500).json({ "message": "error" });
        logger.error(`${res.statusCode} || 500 - Login Failed (Catch) - ${req.originalUrl} - ${req.method} - ${req.ip}`);
    }
}

Get Help   F1 Write Out   F2 Where Is   F3 Cut Text   F4 Justify   [ Read 128 lines ]   ^C Cur Pos   M-U Undo   M-A Mark Text   M-J To Bracket   M-P Previous   M-B Back
F5 Read File   F6 Replace   F7 Uncut Text   F8 To Spell   ^D Go To Line   M-Z Redo   M-C Copy Text   M-W WhereIs Next   M-V Next   M-F Forward
Exit

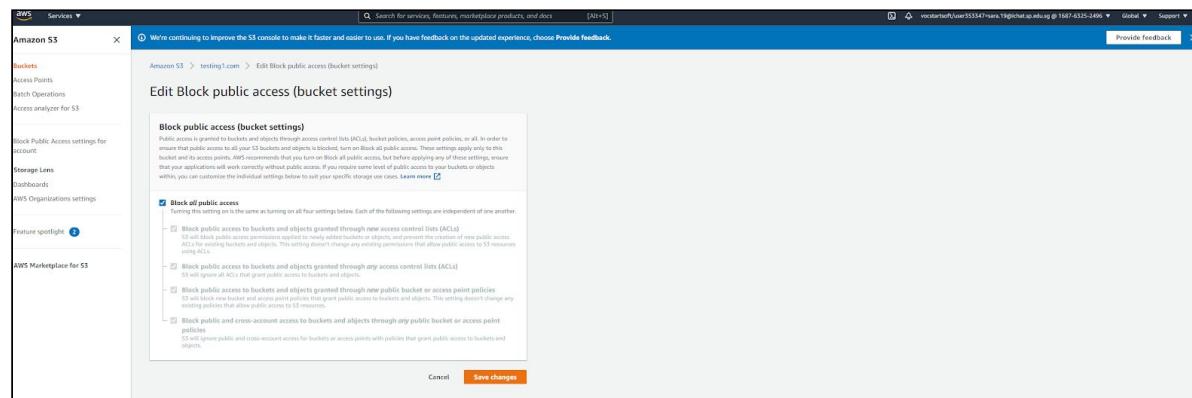
```

3.2 Faced 403 Forbidden Error Issue

At first, we encountered this 403 Forbidden error when we clicked on the link as a public user.



We troubleshooted and realised that this is due to the public access settings for our S3 bucket used to host the backend. Initially, we applied the default settings where we blocked public access. As such, we unchecked the checkbox to allow public access before clicking on the “save changes” button.



After unchecking the checkbox to allow public access, we can now access our hosted website.



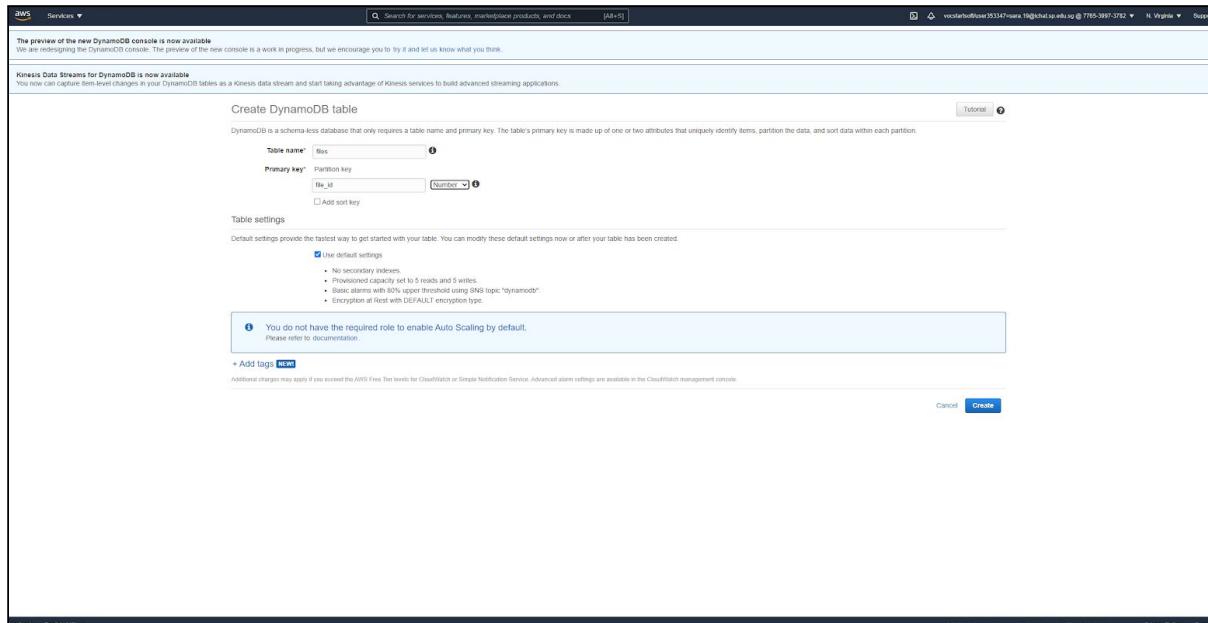
4. Refactoring Application using Lambda Function

In this section, we will be showing a step-by-step approach on how to develop using AWS Lambda Function and enable Cross-Origin-Resource-Sharing (CORS) on the endpoint from the Amazon API Gateway console.

We came across the error of the browser blocking our query request with the message “Failed due to CORS”. After troubleshooting many times, we realised that this is due to our browser enforcing a cross-domain policy rule.

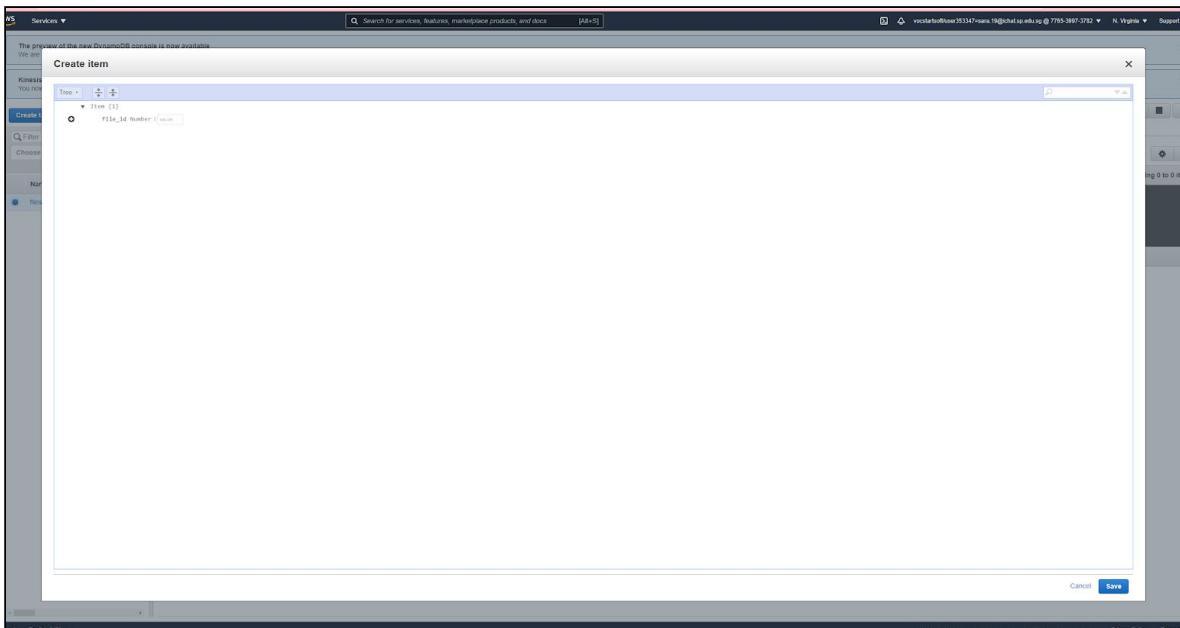
4.1 Creation of Amazon DynamoDB

Step 1: After logging into our AWS account, we navigated to the Amazon API Gateway console to select the API type. We chose to “build” a REST API that works with Lambda, HTTP and other AWS services.



Step 2: After logging into our AWS account, we navigated to the Amazon API Gateway console to select the API type. We chose to “build” a REST API that works with Lambda, HTTP and other AWS services.

Step 3: After creating the REST API for “files”, we navigated back to the items tab and clicked on the “create item” button to add rows to the table where we saw an example of a column attribute required to be added in: file_id.

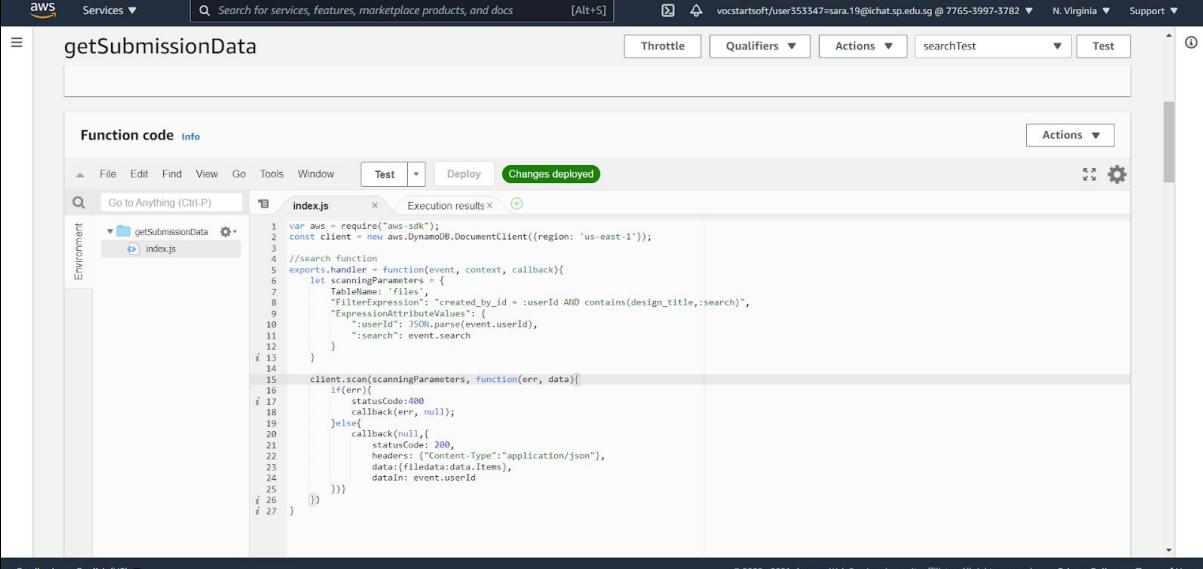


Step 4: We successfully added in the previous Cloudinary submissions of file id 100 to 117 to the current “files” table in DynamoDB as seen below.

file_id	cloudinary_file_id	cloudinary_url	created_by_id	design_title	design_description
100	Design/cbtlmraclz2mwegzb	http://res.cloudinary.com/sweeteara/image/upload/v1607580863/Design...	100	rita design 1	rita design 1 description text 1 text 2 text 3 text 4 text 5...
101	Design/zgsjewox4b2buackdu	http://res.cloudinary.com/sweeteara/image/upload/v1607585863/Design...	100	rita design 2	design 2 description text 1 text 2 text 3 text 4...
102	Design/w0fbdbzehosyvght	http://res.cloudinary.com/sweeteara/image/upload/v1607585863/Design...	100	rita design 3	design 3 description text 1 text 2 text 3 text 4...
103	Design/fniztusabobkavk88d	http://res.cloudinary.com/sweeteara/image/upload/v1607585863/Design...	100	Design 4	design 4 description text 1 text 2 text 3 text 4...
104	Design/rplguo1jahrcaeqfpx	http://res.cloudinary.com/sweeteara/image/upload/v1607585867/Design...	100	Design 4	four
105	Design/tfrsyxknh5bdqhd	http://res.cloudinary.com/sweeteara/image/upload/v1607585872/Design...	100	rita design 5	rita design 7 description text 1 text 2 text 3 text 4...
106	Design/gz1bpqff6ko1teqap	http://res.cloudinary.com/sweeteara/image/upload/v1607585873/Design...	100	rita design 7	rita design 7 description text 1 text 2 text 3 text 4...
107	Design/dthmaxxz72mwegzb	http://res.cloudinary.com/sweeteara/image/upload/v1607585875/Design...	100	rita design 8	rita design 8 description text 1 text 2 text 3 text 4...
108	Design/yw4ys1nqflh2yqmhq	http://res.cloudinary.com/sweeteara/image/upload/v1607585876/Design...	100	rita design 9	rita design 9 description text 1 text 2 text 3 text 4...
109	Design/pqkaypknodkrogrqfgy	http://res.cloudinary.com/sweeteara/image/upload/v1607585878/Design...	100	rita design 10	rita design 10 description text 1 text 2 text 3 text 4...
110	Design/ivxtkcdvz2ah2dn1	http://res.cloudinary.com/sweeteara/image/upload/v1607585879/Design...	100	rita design 11	rita design 11 description text 1 text 2 text 3 text 4...
111	Design/nqglofopswewvnt3bg	http://res.cloudinary.com/sweeteara/image/upload/v1607585880/Design...	100	rita design 12	rita design 12 description text 1 text 2 text 3 text 4...
112	Design/qeessj3xmbdsham	http://res.cloudinary.com/sweeteara/image/upload/v1607586449/Design...	100	Design A	Description A
113	Design/hcwcoxiwkympcu9zm	http://res.cloudinary.com/sweeteara/image/upload/v160813125/Design...	105	Testing	Testing
114	Design/hcwcoxiwkympcu9zm	http://res.cloudinary.com/sweeteara/image/upload/v1608299902/Design...	102	Testing B	Design B
115	Design/vk3n12099yqoc7wrm	http://res.cloudinary.com/sweeteara/image/upload/v160827249/Design...	105	test	random
116	Design/3twdewco3ncv03pffl	http://res.cloudinary.com/sweeteara/image/upload/v1608778059/Design...	105	Testing Design C	Design C
117	Design/hzy27mrpr95kr4vpc	http://res.cloudinary.com/sweeteara/image/upload/v1609145227/Design...	102	Testing 123	Design 7

4.2 Creation of Lambda Function to Get All Users Submission

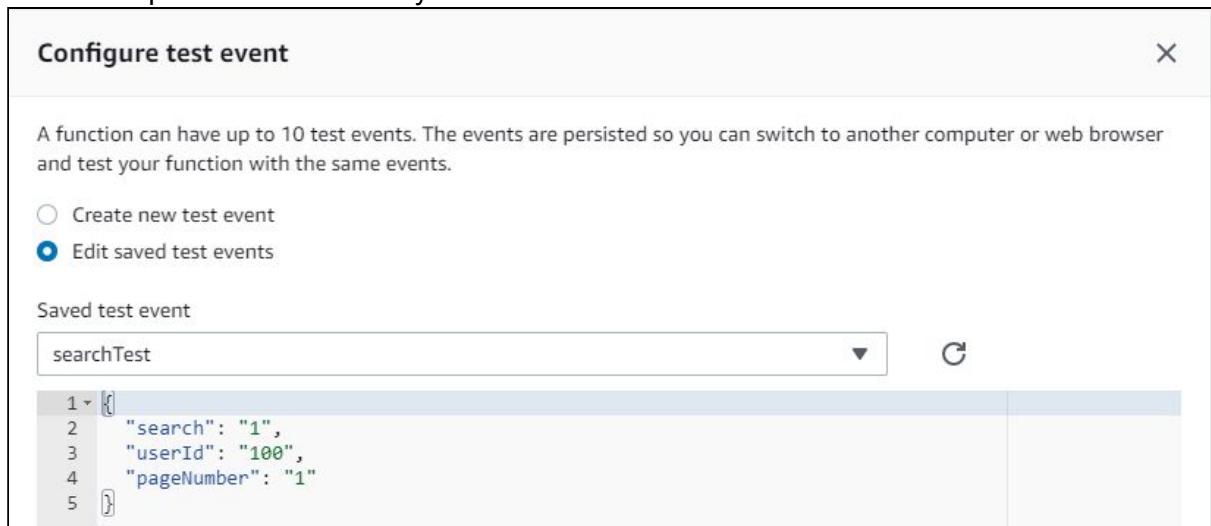
Step 1: By initiating a client that operates DynamoDB in our user account's region, we modified the index.js file to write a Lambda Function called "getSubmissionData" that serves as a search to return all submitted files from the database. We saw that our changes had been deployed.



The screenshot shows the AWS Lambda function editor for the "getSubmissionData" function. The "Function code" tab is selected, displaying the "index.js" file. The code implements a search logic using AWS SDK for DynamoDB:

```
1 var aws = require("aws-sdk");
2 const client = new aws.DynamoDB.DocumentClient({region: 'us-east-1'});
3
4 //search function
5 exports.handler = function(event, context, callback){
6     let scanningParameters = {
7         "TableName": "file",
8         "FilterExpression": "created_by_id = :userId AND contains(design_title,:search)",
9         "ExpressionAttributeValues": {
10             ":userId": JSON.parse(event.userId),
11             ":search": event.search
12         }
13     }
14     client.scan(scanningParameters, function(err, data){
15         if(err)
16             if(err.statusCode==400)
17                 callback(err, null);
18             else{
19                 callback(null, {
20                     statusCode: 200,
21                     headers: {"Content-Type":"application/json"},
22                     data:{fileData:data.Items},
23                     dataIn: event.userId
24                 })
25             }
26     })
27 }
```

Step 2: We then proceed to configure a test event in order to test on the lambda function. After that, we executed the result. Based on the below screenshot, we can see that the results are printed out accurately.



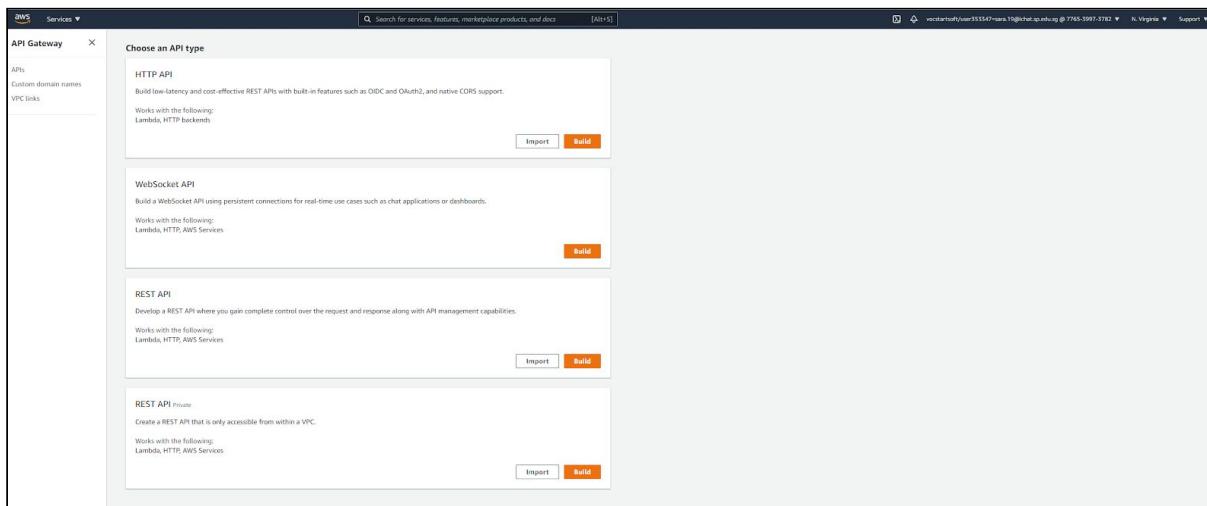
```

{
  "statusCode": 200,
  "headers": {
    "Content-Type": "application/json"
  },
  "data": [
    {
      "filedata": [
        {
          "design_title": "rita design 1",
          "cloudinary_url": "http://res.cloudinary.com/sweetiesara/image/upload/v1607585879/Design/dxtct1cdvv13iu0kh1.png",
          "cloudinary_file_id": "Design/dxtct1cdvv13iu0kh1",
          "created_by_id": 100,
          "design_description": "rita design 11 description text 1 text 2 text 3 text 4 ....",
          "file_id": 110
        },
        {
          "design_title": "rita design 1",
          "cloudinary_url": "http://res.cloudinary.com/sweetiesara/image/upload/v1607585863/Design/c8rt9mxad75mwsq1b.png",
          "cloudinary_file_id": "Design/c8rt9mxad75mwsq1b",
          "created_by_id": 100,
          "file_id": 100,
          "design_description": "rita design 1 description text 1 text 2 text 3 text 4 text 5..6"
        },
        {
          "design_title": "rita design 12",
          "cloudinary_url": "http://res.cloudinary.com/sweetiesara/image/upload/v1607585881/Design/mgalsfopikaenvr38g.png",
          "cloudinary_file_id": "Design/mgalsfopikaenvr38g",
          "created_by_id": 100,
          "design_description": "rita design 12 description text 1 text 2 text 3 text 4 ....",
          "file_id": 111
        },
        {
          "design_title": "rita design 10",
          "cloudinary_url": "http://res.cloudinary.com/sweetiesara/image/upload/v1607585878/Design/gukshasyno6roogrgrgf.png",
          "cloudinary_file_id": "Design/gukshasyno6roogrgrgf"
        }
      ]
    }
  ]
}

```

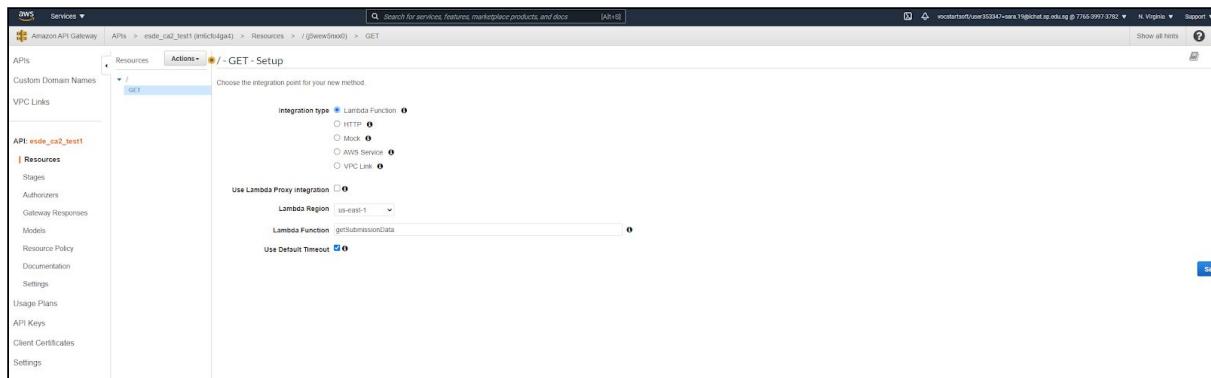
4.3 Creation of AWS Web API Gateway

Step 1: After logging into our AWS account, we navigated to the Amazon API Gateway console to select the API type. We chose to “build” a REST API that works with Lambda, HTTP and other AWS services.

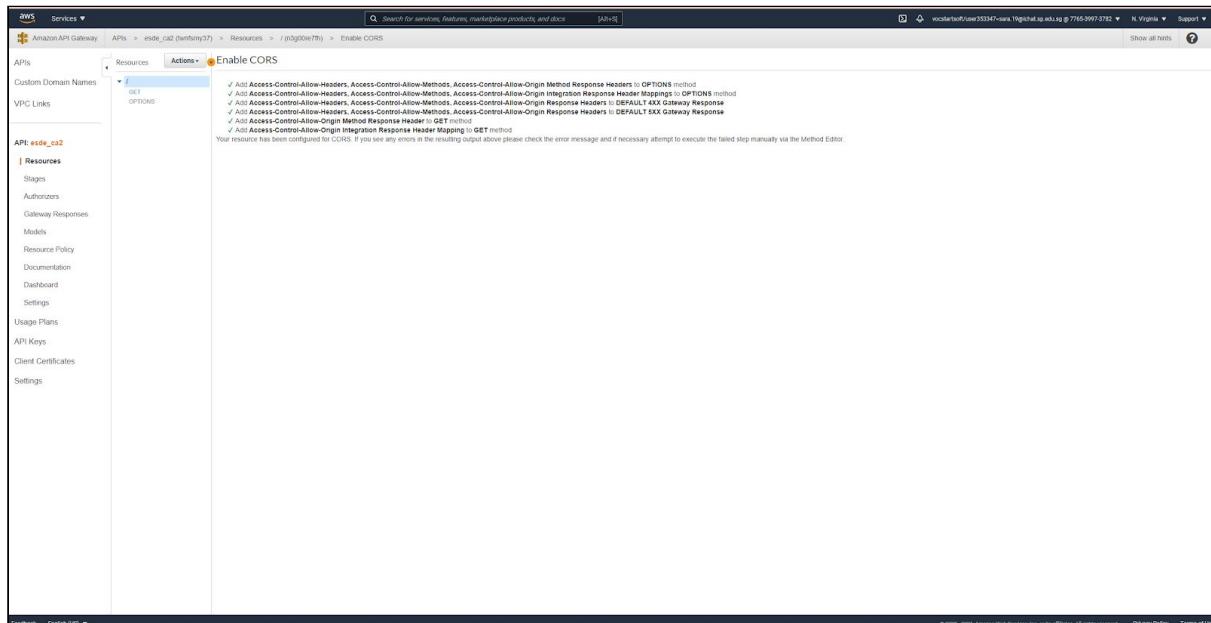


Step 2: We selected the option to create a “new API” and named it “esde_ca2_test1”.

Step 3: We clicked on the name of the created API and selected “Lambda Function” as the integration type.



Step 4: Above GET, we choose the resource / then select “actions”, “enable CORS” and “yes” to replace existing CORS headers. We applied the default settings and only ticked the checkboxes for Default 4XX and Default 5XX. We then ignored the warnings and proceeded to click the “deploy API” button to deploy it into the production stage.



Step 5: Using Postman, we tested the serverless application by entering the Amazon Gateway API endpoint to get all submission files for Rita whose user id is 100. As seen below, the headers successfully returned an application/json type of response.

The screenshot shows the Postman interface with a successful API call. The URL is https://unverified.execute-api.us-east-1.amazonaws.com/ESDE_C42/search?userId=100. The response body contains JSON data representing design submissions for user ID 100. The response status is 200 OK, and the content type is application/json.

Step 6: We navigated back to our Amazon S3 website in the browser. As seen below, we were able to view the design submission files of the user Rita upon entering the query string.

The screenshot shows the Amazon S3 website interface for user Rita. It displays three design submissions: 'RITA DESIGN 5', 'RITA DESIGN 1', and 'RITA DESIGN 3'. Each submission has an 'UPDATE' button next to it. The designs are described as 'Design File for testing'.

5. Cloud Security Features

In this section, we will be addressing the design and implementation of 4 such cloud security features in our web application.

5.1 Centralised Logging for Web API Access

In this section, we will be showing a step-by-step approach on how to setup CloudWatch API logging using the API Gateway console.

Step 1: After logging into our AWS account, we navigated to the IAM dashboard and clicked on “Roles” at the side navigation panel. We selected “AWS Service” as the type of trusted entity and “API Gateway” as the service that will use the newly created role. We proceeded to click the “next” button to edit the permissions.

The screenshot shows the AWS IAM 'Create role' wizard, Step 1: Set permissions type. The 'AWS service' option is selected under 'Select type of trusted entity'. Below, 'Common use cases' show 'EC2' and 'Lambda' selected. A large table lists various AWS services and their corresponding permissions. At the bottom, 'API Gateway' is selected under 'Select your use case'.

Create role

Select type of trusted entity

AWS service
EC2, Lambda and others

Another AWS account
Belonging to you or 3rd party

Web identity
Cognito or any OpenID provider

SAML SAML 2.0 federation
Your corporate directory

Allows AWS services to perform actions on your behalf. Learn more

Choose a use case

Common use cases

EC2
Allows EC2 instances to call AWS services on your behalf.

Lambda
Allows Lambda functions to call AWS services on your behalf.

Or select a service to view its use cases

API Gateway	CloudWatch Events	EKS	IoT Things Graph	Redshift
AWS Backup	CodeBuild	EMR	KMS	Rekognition
AWS Chatbot	CodeDeploy	ElastiCache	Kinesis	RoboMaker
AWS Marketplace	CodeGuru	Elastic Beanstalk	Lake Formation	S3
AWS Support	CodeStar Notifications	Elastic Container Registry	Lambda	SMS

* Required Cancel Next: Permissions

AppSync Connect ElasticLoadBalancing MQ SageMaker

Application Auto Scaling DMS Forecast Machine Learning Security Hub

Application Discovery Service Data Lifecycle Manager GameLift Macie Service Catalog

Batch Data Pipeline Global Accelerator Managed Blockchain Step Functions

Braket DataBrew Glue MediaConvert Storage Gateway

Budgets DataSync Greengrass Migration Hub Systems Manager

Certificate Manager DeepLens GuardDuty Network Firewall Textract

Chime Directory Service Health Organizational View OpsWorks Transfer

CloudFormation DynamoDB Honeycode Personalize Trusted Advisor

CloudHSM EC2 IAM Access Analyzer Purchase Orders VPC

CloudTrail EC2 - Fleet Inspector QLDB WorkLink

CloudWatch Alarms EC2 Auto Scaling IoT RAM WorkMail

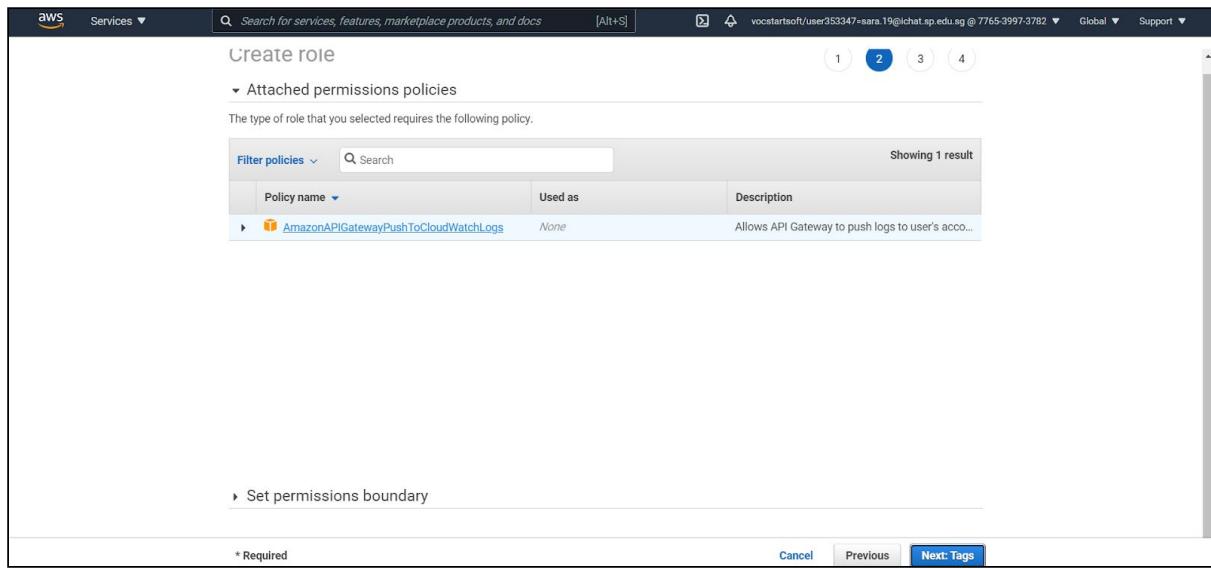
CloudWatch Application Insights EC2 Image Builder IoT SiteWise RDS

Select your use case

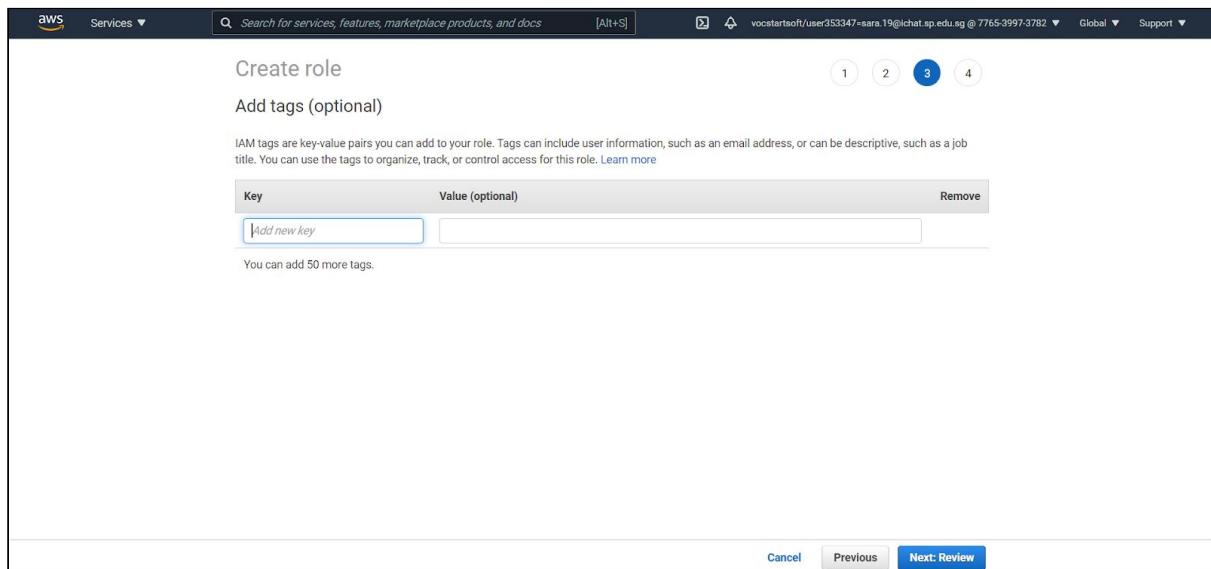
API Gateway
Allows API Gateway to push logs to CloudWatch Logs.

* Required Cancel Next: Permissions

Step 2: We clicked on the policy name that appeared. By establishing API Gateway as the use case, this service will be entitled to push logs to CloudWatch. We proceeded to click the “next” button to view the settings for tags.



Step 3: Since adding tags is an optional step, we applied the default settings for it and proceeded to click on the “next” button to review the summary of the role created.



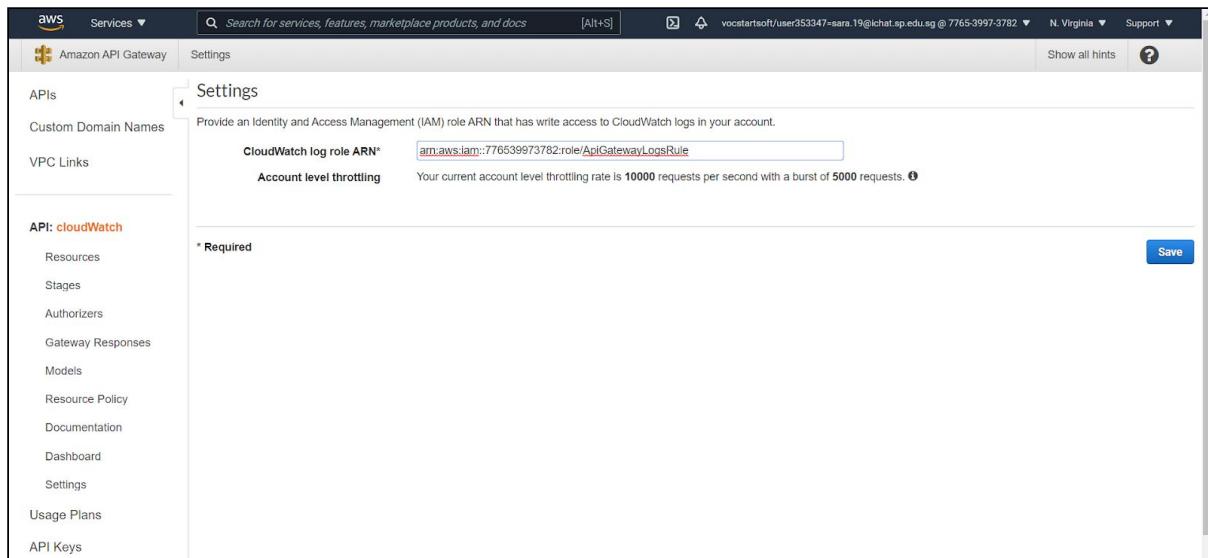
Step 4: We named our role as “ApiGatewayLogsRule” and applied the default role description. We proceeded to click on the “next” button to confirm the role creation.

The screenshot shows the 'Create role' review step in the AWS IAM console. The role name is 'ApiGatewayLogsRule'. The role description is 'Allows API Gateway to push logs to CloudWatch Logs.' The trusted entity is 'AWS service: apigateway.amazonaws.com'. The policy is 'AmazonAPIGatewayPushToCloudWatchLogs'. The permissions boundary is not set. There are no tags added. The status bar at the bottom indicates 'Required' fields and provides 'Cancel', 'Previous', and 'Create role' buttons.

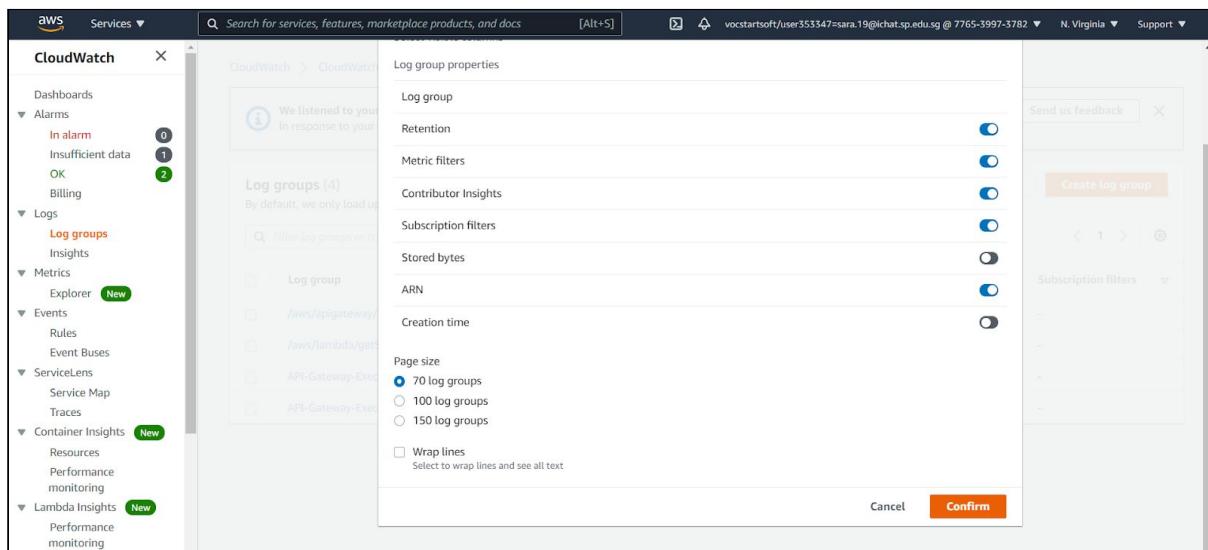
Step 5: We viewed the summary of the role and took note of the IAM's Role ARN which needs to be transferred over to the API Gateway console to enable access logging later on.

The screenshot shows the 'Summary' page for the 'ApiGatewayLogsRule' role in the AWS IAM console. The role ARN is 'arn:aws:iam::776539973782:role/ApiGatewayLogsRule'. The role description is 'Allows API Gateway to push logs to CloudWatch Logs.'. The instance profile ARNs are listed as 'None'. The path is '/'. The creation time is '2021-02-02 14:18 UTC+0800'. The last activity is 'Not accessed in the tracking period'. The maximum session duration is '1 hour'. The permissions tab is selected, showing one policy applied: 'AmazonAPIGatewayPushToCloudWatchLogs'. This is a managed policy. The status bar at the bottom indicates 'Required' fields and provides 'Delete role' and 'Permissions' buttons.

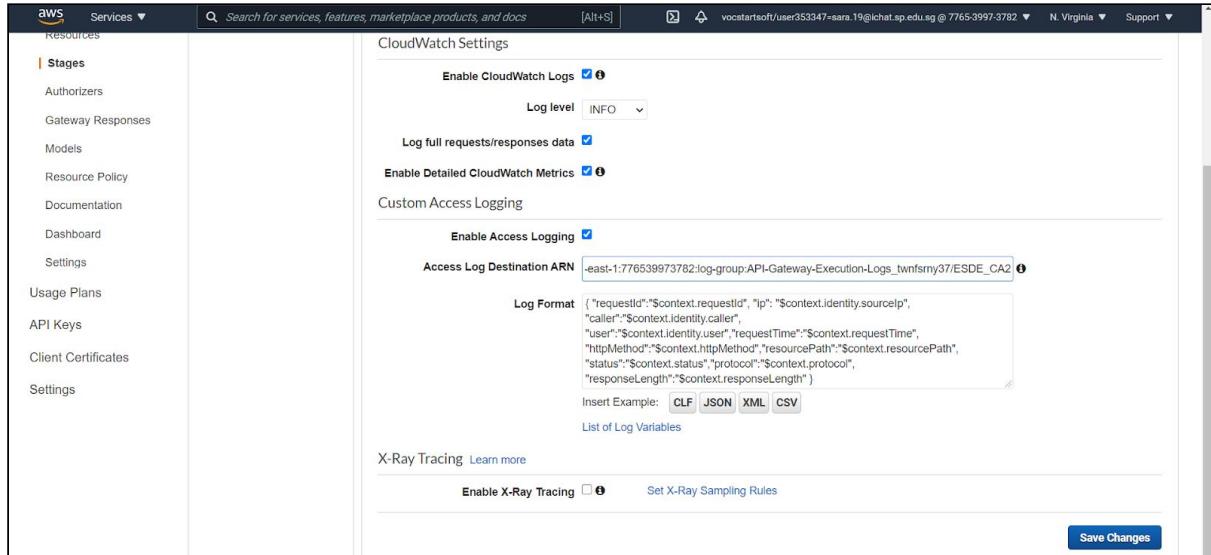
Step 6: We navigated to the API Gateway dashboard and clicked on the settings in the side navigation panel and transferred over the Role ARN to fill in the CloudWatch log role ARN before clicking the “save” button to confirm the changes.



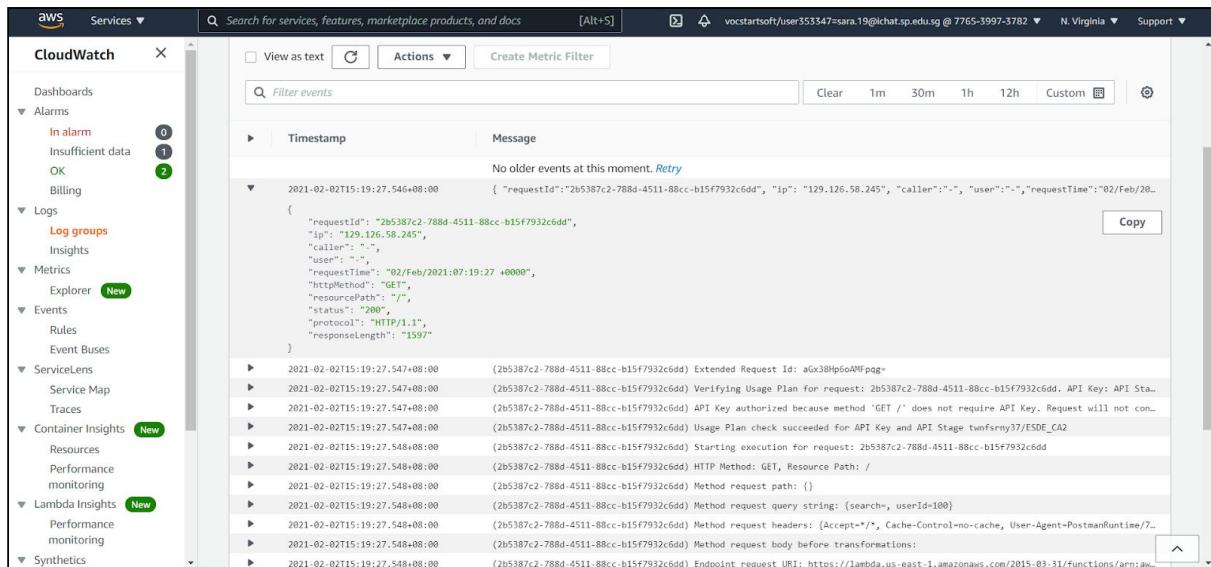
Step 7: We navigated back to the CloudWatch dashboard and clicked on the settings in the side navigation panel to change the ARN column from “hidden” to “visible” so that we can see the log destination of the Access Log Destination ARN and strictly follow its format of arn:aws:logs:{region}:{account-id}:log-group:log-group-name.



Step 8: We clicked on the previously created API called “esde_ca2_test” to access its stage editor and “logs/tracing” tab section. Under the CloudWatch settings, we selected “Info” instead of “Error” from the dropdown menu and ticked the checkboxes to enable CloudWatch logs and log full requests and responses data. Under the Custom Access Logging, we transferred over ARN from the previous step and opted for the log to be printed in JSON format. We proceeded to click on the “save changes” button to finish the setup.



Step 9: We navigated back to the CloudWatch dashboard and clicked on the log group for the above mentioned “esde_ca2_test” API. We are able to view the following metrics for “where”, “when”, “why” and “how” in the logging. For example, in the most recent log for getting all submissions, the details included 129.126.58.245 as the source IP address, the request time made at 3.19pm, the HTTP method and resource path called “GET” and “/” respectively and the response length is 1597.



5.2 MySQL RDS Database Security

In this section, we will be showing a step-by-step approach on the features we implemented to make our MySQL RDS Database more secure.

Step 1: After logging into our AWS account, we navigated to the EC2 dashboard and clicked on the security groups for the previously launched instance. We see that the inbound rules for it involve only the TCP protocol of SSH and the EC2 instance running on port 22 and 5000 respectively. This means that to access the RDS Database, the instance must also run NodeJS codes.

Type	Protocol	Port range	Source	Description - optional
SSH	TCP	22	0.0.0.0/0	-
Custom TCP	TCP	5000	0.0.0.0/0	-
Custom TCP	TCP	5000	/0	-

Step 2: After seeing that the source is set to 0.0.0.0/0 for the inbound rule of RDS Database, we realised that this is not a good measure as the contents are publicly available and anyone can have read-write access to it anywhere. Thus, to ensure the least amount of privileges for the database connection credentials, we proceeded to click on the “edit inbound rule” button to customize the source from the previous 0.0.0.0/0 to only allow the user account’s number.

Step 3: We see that the details for the security groups of the RDS Database has been successfully modified to allow only 1 permission entry for all inbound and outbound rules. This verifies that the data stored within it is secured.

The screenshot shows the AWS CloudWatch Metrics Insights interface with a query result titled "Inbound security group rules successfully modified on security group sg-0eb1dc68f1a15c53 - esde-database". The results table displays three security groups:

Name	Security group ID	VPC ID	Description	Owner	Inbound rules count	Outbound rules count
-	sg-0ba7498543b5bd00	vpc-3f1feab42	launch-wizard-1 create...	108763252496	3 Permission entries	1 Permission entry
-	sg-0eb1dc68f1a15c53	vpc-3f1feab42	for AWS ESDE RDS	108763252496	1 Permission entry	1 Permission entry
-	sg-d7ab55df	vpc-3f1feab42	default VPC security gr...	108763252496	1 Permission entry	1 Permission entry

Below the main table, there is a detailed view for the security group "sg-0eb1dc68f1a15c53 - esde-database" under the "Inbound rules" tab. It shows a single rule:

Type	Protocol	Port range	Source	Description - optional
MySQL/Aurora	TCP	3306	sg-0ba7498543b5bd00 (Launch-wizard-1)	

Step 4: We navigated to our terminal in Visual Studio Code and saw that our access to the database has been logged where the results are successfully retrieved after a TCP protocol handshake is established to call the connection. As shown from the screenshot, we could not run it on localhost due to permission blocked issues that we created.

The screenshot shows a Visual Studio Code terminal window with a Node.js script running. The output shows the following error:

```

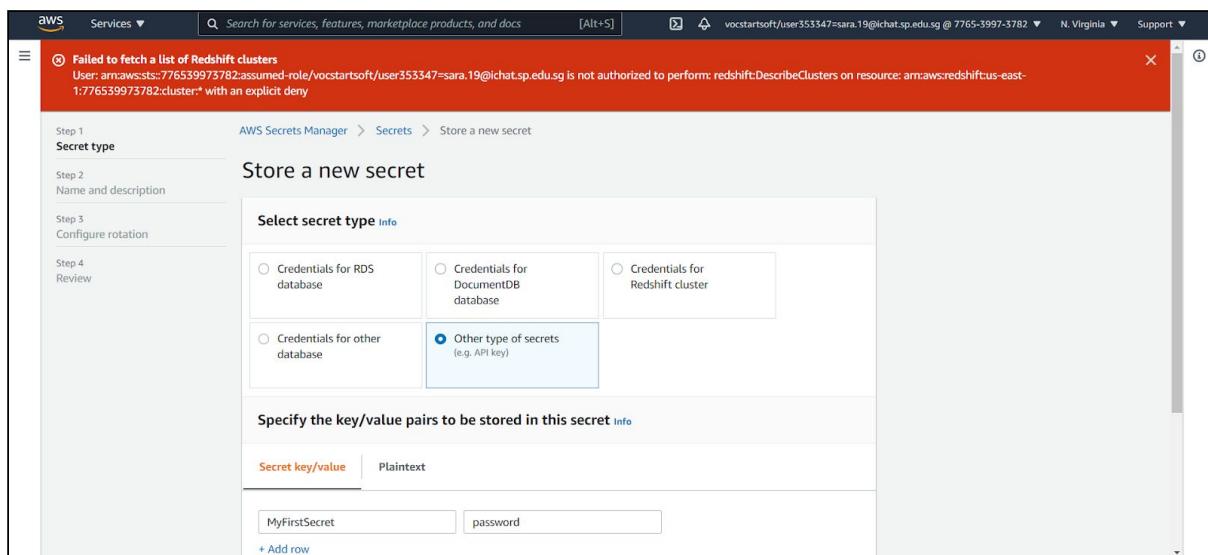
PS C:\Users\Sara.Gokkaran\Desktop\ESDE\Assignment\ESDE_CAI_DITTF2242_SaraGokkaran_P1935657_YapMinRuVicki_P1904556\experimentsecuritywithcompetitionsystem> node index.js
(node:1616) [DEP0018] DeprecationWarning: The 'nodejs' executable cannot be loaded because running scripts is disabled on this system. For more information, see about_Execution_Policies at https://go.microsoft.com/fwlink/?LinkID=135170.
+ nodemon index.js
+ nodemon index.js
+ +-----+
+ CategoryInfo : SecurityError: () [], PSSecurityException
+ FullyQualifiedErrorId : ProtocolHandlerFailedAccess
PS C:\Users\Sara.Gokkaran\Desktop\ESDE\Assignment\ESDE_CAI_DITTF2242_SaraGokkaran_P1935657_YapMinRuVicki_P1904556\experimentsecuritywithcompetitionsystem> node index.js
Service worker registered at https://localhost:4444
at Protocol1.Handshake (C:\Users\Sara.Gokkaran\Desktop\ESDE\Assignment\ESDE_CAI_DITTF2242_SaraGokkaran_P1935657_YapMinRuVicki_P1904556\experimentsecuritywithcompetitionsystem\node_modules\mysql\lib\protocol\Protocol.js:51:23)
at PoolConnection.connect (C:\Users\Sara.Gokkaran\Desktop\ESDE\Assignment\ESDE_CAI_DITTF2242_SaraGokkaran_P1935657_YapMinRuVicki_P1904556\experimentsecuritywithcompetitionsystem\node_modules\mysql\lib\Connection.js:116:18)
at PoolConnection.connect (C:\Users\Sara.Gokkaran\Desktop\ESDE\Assignment\ESDE_CAI_DITTF2242_SaraGokkaran_P1935657_YapMinRuVicki_P1904556\experimentsecuritywithcompetitionsystem\node_modules\mysql\lib\Connection.js:116:18)
at new Pool (anonymous)
at Object.Pool.create (C:\Users\Sara.Gokkaran\Desktop\ESDE\Assignment\ESDE_CAI_DITTF2242_SaraGokkaran_P1935657_YapMinRuVicki_P1904556\experimentsecuritywithcompetitionsystem\node_modules\mysql\lib\connectionPool.js:6:14)
at exports.openConnection (C:\Users\Sara.Gokkaran\Desktop\ESDE\Assignment\ESDE_CAI_DITTF2242_SaraGokkaran_P1935657_YapMinRuVicki_P1904556\experimentsecuritywithcompetitionsystem\node_modules\mysql\lib\connection.js:15:34)
at Layer.handle [as handleRequest] (C:\Users\Sara.Gokkaran\Desktop\ESDE\Assignment\ESDE_CAI_DITTF2242_SaraGokkaran_P1935657_YapMinRuVicki_P1904556\experimentsecuritywithcompetitionsystem\node_modules\express\lib\router\layer.js:95:5)
at next (C:\Users\Sara.Gokkaran\Desktop\ESDE\Assignment\ESDE_CAI_DITTF2242_SaraGokkaran_P1935657_YapMinRuVicki_P1904556\experimentsecuritywithcompetitionsystem\node_modules\express\lib\router\route.js:137:13) {
  code: 'PROTOCOL_SEQUENCE_TIMEOUT',
  fatal: true,
  timeout: 10000
}
GOT RESULTS

```

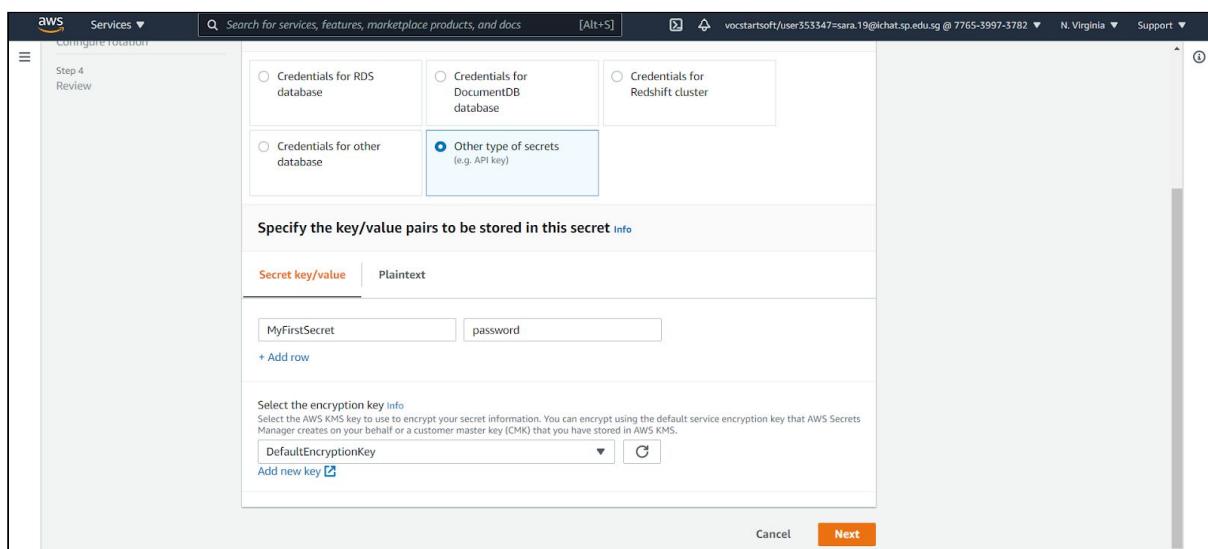
5.3 Storing Private Key using AWS Secrets Manager

In this section, we will be showing a step-by-step approach on how to create and store secret keys in AWS Secrets Manager, and retrieve it using its console. By doing so, the security aspect of our web application is improved as we can manage access to our private keys through the implementation of IAM's fine-grained created policies. This includes secret information, such as credentials, connection details, key name and value string pairs.

Step 1: After logging into our AWS API Gateway account, we navigated to the Secrets Manager dashboard and clicked on the “Store a new secret” button. We selected the option of “other type of secrets” as it has no relation to our database credentials.



Step 2: Under the first field for key/value pairs to be stored, we specified our secret as “MyFirstSecret” and configured a password by entering “password” in the next field. Under the section to select the encryption key, we chose the option of “DefaultEncryptionKey”. Using this cost-effective option where AWS Secrets Manager provides the standard encryption service at no charge, compared to using a custom key, we can spend our \$50 credits wisely. We proceeded to click the “next” button to set the name, description and tags.



Step 3: We named our secret “MyFirstSecretName” and described its purpose to allow us “access to secret”. Since tags is an optional thing, we applied the default settings to have no tags and proceeded to click on the “next” button to configure rotation.

Secret name and description

Secret name
Give the secret a name that enables you to find and manage it easily.

Description - optional

Tags - optional

Key	Value - optional
<input type="text" value="Enter key"/>	<input type="text" value="Enter value"/>

Resource Permissions (optional)

Add a new resource policy or edit resource policy to access secrets across AWS accounts securely.

Next

Step 4: We selected the “disable” option when configuring automatic rotation and did not choose any Lambda Function as we did not provide database credentials as our secret and so there is no need to invoke any functions along with it.

If you enable automatic rotation, the first rotation will happen immediately when you store this secret. If this secret is already in use, you must update your applications to retrieve it from AWS Secrets Manager. Read the [getting started guide](#) on rotation.

Configure automatic rotation

Disable automatic rotation
Recommended when your applications are using this secret and have not been updated to use AWS Secrets Manager.

Enable automatic rotation
Recommended when your applications are not using this secret yet.

Select rotation interval

This secret will be rotated based on the schedule you determine.

Must be a value between 1 and 365 days

Choose an AWS Lambda function

Select an AWS Lambda function that has permissions to rotate this secret.

Next

Step 5: We proceeded to click on the “next” button at the bottom of the previous page to review the summary of the created secret. We also viewed the code snippet to retrieve the secret using an API written in Javascript language but decided not to implement it within our codes as we have already uploaded it to putty previously. We proceeded to click the “store” button to save the secret.

Review

- Secret type: Other type of secret
- Encryption key: DefaultEncryptionKey
- Secret name: MyFirstSecretName
- Description: Access to secret
- Tags: -
- Resource Permissions: -
- Automatic rotation: Disabled
- Rotation interval: -

Sample code

View a code sample that illustrates how to retrieve the secret in your application.

```

1 // Use this code snippet in your app.
2 // If you need more information about configurations or implementing the sample code, visit the
3 // https://aws.amazon.com/developers/getting-started/nodejs/
4
5 // Load the AWS SDK
6 var AWS = require('aws-sdk'),
7     region = "us-east-1",
8     secretName = "MyFirstSecretName",
9     secret,
10    decodedBinarySecret;
11
12 // Create a Secrets Manager client
13 var client = new AWS.SecretsManager({
14     region: region
15 });
16
17 // In this sample we only handle the specific exceptions for the 'GetSecretValue' API.
18 // See https://docs.aws.amazon.com/secretsmanager/latest/apireference/API_GetSecretValue.html
19 // See more about them here: https://docs.aws.amazon.com/sdk-for-javascript/v2/developer-guide/global-error-handling.html
20

```

[Download AWS SDK for Javascript](#)

[Cancel](#) [Previous](#) [Store](#)

Step 6: We managed to successfully store the secret in AWS Secrets Manager as seen below.

Your secret MyFirstSecretName has been successfully stored.
Use the sample code to update your applications to retrieve this secret.

[See sample code](#) [View details](#)

Secrets

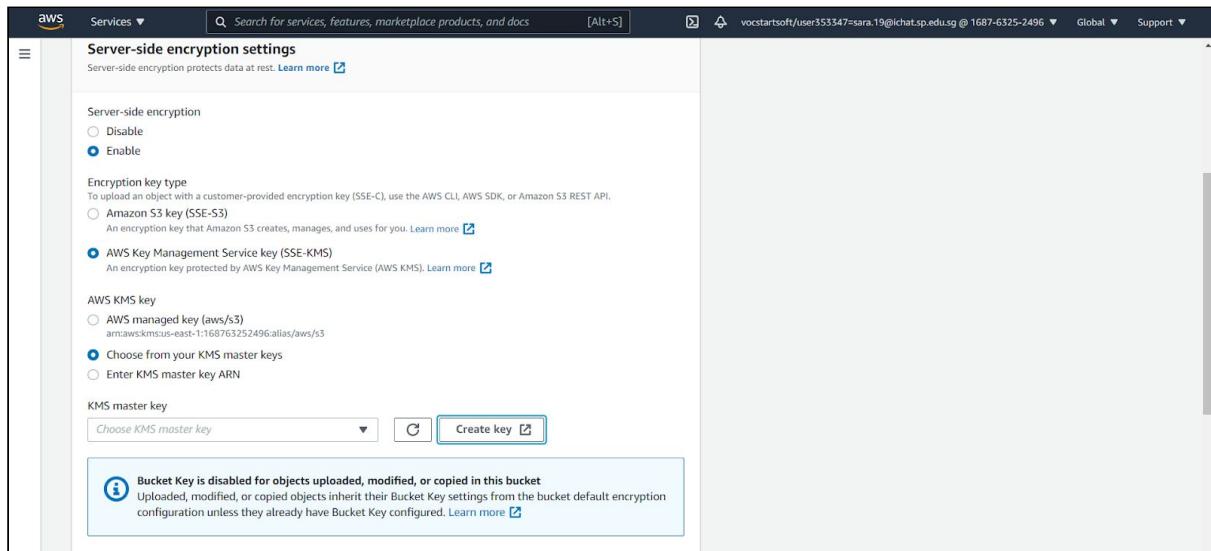
Secret name	Description	Last retrieved (UTC)
MyFirstSecretName	Access to secret	-

[Store a new secret](#)

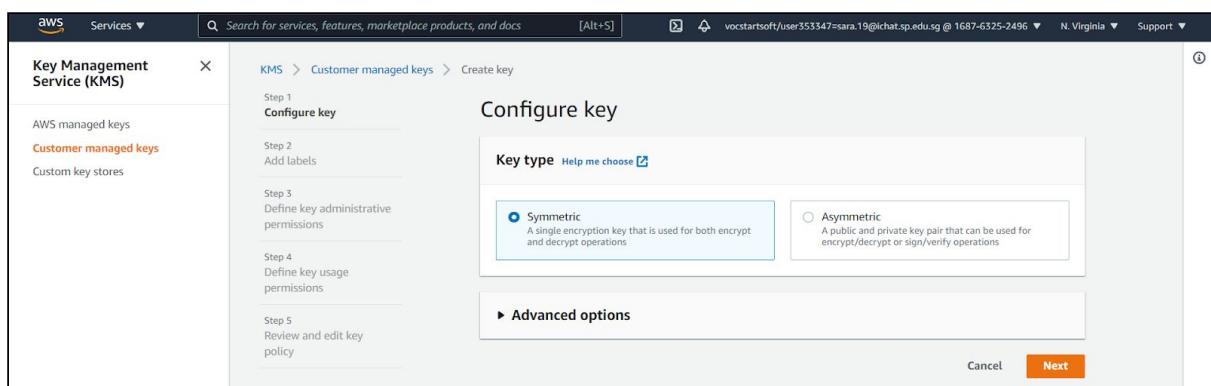
5.4 Server Side Encryption (SSE) on S3 Bucket

In this section, we will be showing a step-by-step approach on how to enhance the security of an S3 Bucket by using AWS Key Management Service (KMS) to encrypt the data within it. By doing so, the data is transformed into a way that can only be read by the administrator who has the correct decryption key, which further prevents unauthorized access to sensitive information such as private keys.

Step 1: After logging into our AWS **main** account, we navigated to the S3 dashboard and clicked on the name of the previously created bucket for the backend zipped folder. We navigated to its server-side encryption settings, clicked on “enable” and selected “SSE-KMS” for the type. As we do not have an existing KMS master key, we proceeded to click on the “create key” button to create the key policy.



Step 2: We were redirected to the KMS dashboard where we had to decide between a symmetric or asymmetric key type. The main difference is that symmetric encryption uses a single key that needs to be shared among the users who receive the message while asymmetric encryption uses a pair of public and private keys to encrypt and decrypt messages when communicating. Thus, we chose to configure a symmetric key type as it takes a faster time to set up and proceeded to ignore the advanced options and click on the “next” button to add labels.



Step 3: In the alias field, we named our key “backend-test” so that it will be easier to identify which part of the application the key is encrypting. Since the description and tags were optional, we just skipped the settings for it and applied the default. We proceeded to click the “next” button to define administrator and user permissions.

Key Management Service (KMS)

Step 1 Configure key

Step 2 Add labels

Step 3 Define key administrative permissions

Step 4 Define key usage permissions

Step 5 Review and edit key policy

Add labels

Create alias and description

Enter an alias and a description for this key. You can change the properties of the key at any time. [Learn more](#)

Alias: backend-test

Description - optional

Description of the key

Tags - optional

You can use tags to categorize and identify your CMKs and help you track your AWS costs. When you add tags to AWS resources, AWS generates a cost allocation report for each tag. [Learn more](#)

This key has no tags.

Add tag

You can add up to 50 more tags

Cancel Previous Next

Step 4: For the administrative and usage permissions, we simply skipped through them as we already have other existing IAM policies that can manage our roles. We proceeded to click the “next” button to review the key policy created.

Key Management Service (KMS)

KMS > Customer managed keys > Create key

Step 1 Configure key

Step 2 Add labels

Step 3 Define key administrative permissions

Step 4 Define key usage permissions

Step 5 Review and edit key policy

Define key administrative permissions

Key administrators

Choose the IAM users and roles who can administer this key through the KMS API. You may need to add additional permissions for the users or roles to administer this key from this console. [Learn more](#)

<input type="checkbox"/>	Name	Path	Type
<input type="checkbox"/>	AWSServiceRoleForAWSCloud9	/aws-service-role/cloud9.amazonaws.com/	Role
<input type="checkbox"/>	AWSServiceRoleForCloudWatchEvents	/aws-service-role/events.amazonaws.com/	Role
<input type="checkbox"/>	AWSServiceRoleForElastiCache	/aws-service-role/elasticsearch.amazonaws.com/	Role
<input type="checkbox"/>	AWSServiceRoleForOrganizations	/aws-service-role/organizations.amazonaws.com/	Role
<input type="checkbox"/>	AWSServiceRoleForRDS	/aws-service-role/rds.amazonaws.com/	Role
<input type="checkbox"/>	AWSServiceRoleForSupport	/aws-service-role/support.amazonaws.com/	Role
<input type="checkbox"/>	AWSServiceRoleForTrustedAdvisor	/aws-service-role/trustedadvisor.amazonaws.com/	Role
<input type="checkbox"/>	EMR_AutoScaling_DefaultRole	/	Role

Cancel Previous Next

Define key usage permissions

This account

Select the IAM users and roles that can use the CMK in cryptographic operations. Learn more [?](#)

<input type="checkbox"/>	Name	Path	Type
<input type="checkbox"/>	AWSServiceRoleForAWSCloud9	/aws-service-role/cloud9.amazonaws.com/	Role
<input type="checkbox"/>	AWSServiceRoleForCloudWatchEvents	/aws-service-role/events.amazonaws.com/	Role
<input type="checkbox"/>	AWSServiceRoleForElastiCache	/aws-service-role/elasticache.amazonaws.com/	Role
<input type="checkbox"/>	AWSServiceRoleForOrganizations	/aws-service-role/organizations.amazonaws.com/	Role
<input type="checkbox"/>	AWSServiceRoleForRDS	/aws-service-role/rds.amazonaws.com/	Role
<input type="checkbox"/>	AWSServiceRoleForSupport	/aws-service-role/support.amazonaws.com/	Role
<input type="checkbox"/>	AWSServiceRoleForTrustedAdvisor	/aws-service-role/trustedadvisor.amazonaws.com/	Role
<input type="checkbox"/>	EMR_AutoScaling_DefaultRole	/	Role
<input type="checkbox"/>	EMR_DefaultRole	/	Role

Other AWS accounts

Specify the AWS accounts that can use this key. Administrators of the accounts you specify are responsible for managing the permissions that allow their IAM users and roles to use this key. Learn more [?](#)

Add another AWS account

Cancel Previous Next

Step 5: After viewing the key policy, we proceeded to click the “finish” button to complete the policy configuration process.

Review and edit key policy

```

1 {
2   "Id": "key-consolepolicy-3",
3   "Version": "2012-10-17",
4   "Statement": [
5     {
6       "Sid": "Enable IAM User Permissions",
7       "Effect": "Allow",
8       "Principal": {
9         "AWS": "arn:aws:iam::168763252496:root"
10      }
11      "Action": "kms:*",
12      "Resource": "*"
13    }
14  ]
15 }

```

Cancel Previous Finish

Step 6: We managed to successfully create a customer master key of our own for the S3 bucket used to store the backend of our web application as seen below.

The screenshot shows the AWS KMS service dashboard. A green success message at the top states: "Success Your customer master key was created with alias `backend-test` and key ID `69d60595-d40c-44e4-9d37-662cf53152be`". Below this, under "Customer managed keys", there is a table with one row. The table columns are "Aliases", "Key ID", "Status", "Key spec", and "Key usage". The data row shows "backend-test" as the alias, "69d60595-d40c-44e4-9d37-662cf53152be" as the key ID, "Enabled" as the status, "SYMMETRIC_DEFAULT" as the key spec, and "Encrypt and decrypt" as the key usage.

Step 7: We navigated back to the server-side encryption settings at the S3 dashboard and selected the option to “choose from our existing master key” so that we can upload our recently created “backend-test” master key. We proceeded to click on the “save changes” button to finalise the master key setup.

The screenshot shows the AWS S3 service dashboard under "Server-side encryption settings". Under "Encryption key type", the "AWS Key Management Service key (SSE-KMS)" option is selected. Under "AWS KMS key", the "Choose from your KMS master keys" option is selected. The "KMS master key" dropdown shows "arn:aws:kms:us-east-1:168763252496:key/69d60595-d40c-44e4-9d37-662cf53152be". At the bottom, a modal window is open for "Bucket Key". It contains an information icon and the text: "Bucket Key is disabled for objects uploaded, modified, or copied in this bucket. Uploaded, modified, or copied objects inherit their Bucket Key settings from the bucket default encryption configuration unless they already have Bucket Key configured." Below this, there is a "Bucket Key" section with "Disable" and "Enable" radio buttons, where "Disable" is selected. The "Specified objects" section shows a table with one row for "experimentsecuritywithcompetitionssystem.zip". The table columns are "Name", "Type", "Last modified", "Size", and "Storage class". The file name is "experimentsecuritywithcompetitionssystem.zip", type is "zip", last modified is "January 29, 2021, 16:54:35 (UTC+08:00)", size is "48.5 MB", and storage class is "Standard". At the bottom right of the modal, there are "Cancel" and "Save changes" buttons.

5.5 Enforcing CORS

In this section, we will be showing two pieces of evidence of CORS being employed to increase the security aspect of our web application in the cloud. Previously, CORS has caused us some problems for our query requests as our browser policy indicates that it only allows preflight requests. This means that it does not tolerate having our Amazon S3 website and API Gateway endpoints being hosted in different domains.

Step 1: In command prompt, we used the curl parameter to test the CORS OPTIONS request. We saw that our AWS host has completed the SSL/TLS handshake through the acceptance of curl as a user-agent, which activated the Access-Control-Allow-Origin.

```
curl --version
curl 7.30.0 (x86_64-pc-windows) libcurl/7.30.0 OpenSSL/1.0.2a WinSSL/1.0.2a zlib/1.2.8
Copyright (c) 2013, Daniel Stenberg, http://curl.haxx.se

  Usage: curl [options] [url]
  curl -V
  curl -v
  curl -V -v
  curl -v -V
  curl -V -V -V
  curl -V -V -V -V
  curl -V -V -V -V -V
  curl -V -V -V -V -V -V
  curl -V -V -V -V -V -V -V
  curl -V -V -V -V -V -V -V -V
  curl -V -V -V -V -V -V -V -V -V
  curl -V -V -V -V -V -V -V -V -V -V
  curl -V -V
  curl -V -V
  curl -V -V
  curl -V -V
  curl -V -V
  curl -V -V
  curl -V -V
  curl -V -V
  curl -V -V
  curl -V -V
  curl -V -V
  curl -V -V
  curl -V -V
  curl -V -V
  curl -V -V
  curl -V -V
  curl -V -V
  curl -V -V
  curl -V -V
  curl -V -V
  curl -V -V
  curl -V -V
  curl -V -V
  curl -V -V
  curl -V -V
  curl -V -V
  curl -V -V
  curl -V -V
  curl -V -V
  curl -V -V
  curl -V -V
  curl -V -V
  curl -V -V
  curl -V -V
  curl -V -V
  curl -V -V
  curl -V -V
  curl -V -V
  curl -V -V
  curl -V -V
  curl -V -V
  curl -V -V
  curl -V -V
  curl -V -V
  curl -V -V
  curl -V -V
  curl -V -V
  curl -V -V
  curl -V -V
  curl -V -V
  curl -V -V
```

Step 2: In Postman, we ran the AWS API created previously to get all submissions data and saw that Access-Control-Allow-Origin has been activated for that endpoint too.

The screenshot shows the Postman interface with a successful OPTIONS request to the specified endpoint. The response headers include:

- Access-Control-Allow-Origin: http://testing1.com:32959
- Access-Control-Allow-Methods: GET, HEAD, OPTIONS
- Access-Control-Allow-Headers: Cache-Control, Content-Type, Content-Length, Connection, Accept, Accept-Encoding, User-Agent

6. Additional Feature

6.1 Created Button in Frontend to execute Lambda Function

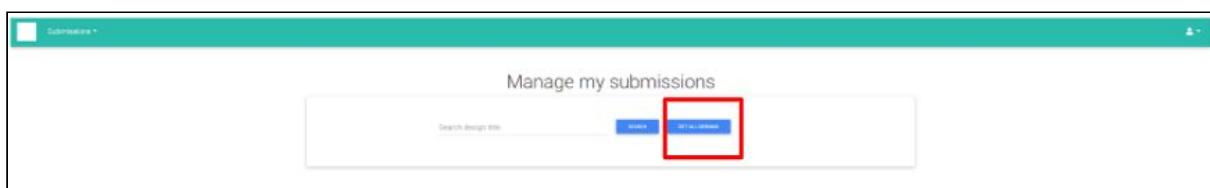
Step 1: We created a button called “get all designs” to execute the lambda function. The following image is the code snippet to print out all the submissions of Rita who is userId 100.

```
$('#getAllDesignButton').on('click', function(event) {
    event.preventDefault();
    axios({
        headers: {
            },
        method: 'get',
        url: 'https://twnfsrny37.execute-api.us-east-1.amazonaws.com/ESDE_CA2?search=&userId=100',
    })
    .then(function(response) {
        const records = response.data.data.filodata;
        const totalNumOfRecords = response.data.data.total_number_of_records;
        //Find the main container which displays page number buttons
        let $pageButtonContainer = $('#pagingButtonBlock');
        //Find the main container which has the id, dataBlock
        let $dataBlockContainer = $('#dataBlock');
        $dataBlockContainer.empty();
        $pageButtonContainer.empty();
        if (records.length == 0) {
            new Noty({
                type: 'information',
                layout: 'topCenter',
                timeout: '5000',
                theme: 'sunset',
                text: 'No submission records found.'
            }).show();
        }
        for (let index = 0; index < records.length; index++) {
            let record = records[index];
            console.log(record.cloudinary_url);
            let $card = $('<div></div>').addClass('card').attr('style', 'width: 18rem;');
            $card.append($('img').attr('src', record.cloudinary_url)).addClass('app_thumbnail');
            let $CardBody = $('<div></div>').addClass('card-body');
            let $editDesignButtonBlock = $('<div></div>').addClass('col-md-2 float-right');
            $editDesignButtonBlock.append($('a').attr('href', 'update_design.html?id=' + record.file_id));
            $CardBody.append($editDesignButtonBlock);
            $CardBody.append($('h5').addClass('card-title').text(record.design_title));
            $CardBody.append($('p').addClass('card-text').text(record.design_description));
            $card.append($CardBody);
            //After preparing all the necessary HTML elements to describe the file data,
            //I used the code below to insert the main parent element into the div element, dataBlock.
            $dataBlockContainer.append($card);
            $dataBlockContainer.append($('h5'));
        } //End of for loop
        let totalPages = Math.ceil(totalNumOfRecords / 4);

        for (let count = 1; count <= totalPages; count++) {
            let $button = $('<button class="btn btn-primary btn-sm" />');
            $button.text(count);
            $button.click(clickHandlerForPageButton);

            $pageButtonContainer.append($button);
        } //End of for loop to add page buttons
    })
    .catch(function(response) {
        //Handle error
        console.dir(response);
        new Noty({
            type: 'error',
            layout: 'topCenter',
            timeout: '5000',
            theme: 'sunset',
            text: 'Unable to search',
        }).show();
    });
});
```

Step 2: The manage submissions page is updated with the new button as seen below.



Step 3: We clicked on the button bordered in red and were able to retrieve all the submissions data that spans across 2 pages as seen below. This means that our Lambda Function has executed correctly.

The screenshot displays a web application interface titled "Manage my submissions". At the top, there is a search bar with the placeholder "Search design title" and two buttons: "SEARCH" and "GET ALL DESIGNS". Below the search bar, the page is divided into two main sections, each containing four items of submission data. Each item includes a small thumbnail image, a title, a subtitle, and a brief description. An "UPDATE" button is located at the bottom right of each item's card. The first section contains items: "rita design 11", "rita design 5", "Design 4", and "rita design 1". The second section contains items: "rita design 9", "rita design 12", "rita design 3", and "rita design 4". The entire page is framed by a thick black border.

7. AWS Cognito service for Authentication (Advanced)

In this section, we will be showing a step-by-step approach on how to create an Amazon Cognito Pool to manage the creation of first-time user accounts at the registration page.

Step 1: After logging into our AWS **API Gateway** account, we navigated to the Amazon Cognito dashboard and clicked on the “Manage your User Pools” button.

The screenshot shows the Amazon Cognito landing page. At the top, there is a search bar and navigation links for services, marketplace products, and documentation. Below the header is a large purple circular icon with a white house-like symbol. The title "Amazon Cognito" is centered below the icon. A descriptive text block explains that Cognito offers user pools and identity pools, detailing their functions. Two buttons at the bottom are "Manage User Pools" and "Manage Identity Pools". Below these buttons are two sections: "Add Sign-up and Sign-in" with a user icon and "Grant your users access to AWS services" with a computer monitor and padlock icon.

Step 2: We were redirected to the User Pools dashboard where we encountered this warning that we didn't have sufficient administrative privileges. We ignored it and proceeded to click on the “Create a user pool” button.

The screenshot shows the "Your User Pools" dashboard. The top navigation bar includes "User Pools | Federated Identities" and a "Create a user pool" button. A prominent orange warning box in the center states: "You don't have sufficient permission for this operation. Please contact your administrator." Below the warning box is a blue hexagonal icon.

Step 3: For the creation of our user pool, we named it “EsdeUser” before choosing the option to “review defaults” as we wish to apply the default settings for the majority.

The screenshot shows the "Create a user pool" wizard. On the left, a sidebar lists steps: Name (highlighted in yellow), Attributes, Policies, MFA and verifications, Message customizations, Tags, Devices, App clients, Triggers, and Review. The main area has a heading "What do you want to name your user pool?". It asks for a descriptive name and provides a placeholder "EsdeUser". Below this is a question "How do you want to create your user pool?". Two options are shown: "Review defaults" (selected) and "Step through settings". The "Review defaults" option includes a sub-note: "Start by reviewing the defaults and then customize as desired". The "Step through settings" option includes a sub-note: "Step through each setting to make your choices".

Step 4: We were redirected to the review page where we checked through each of the details before clicking on the “create pool” button.

The screenshot shows the 'Create a user pool' review step in the AWS User Pools console. The 'Review' tab is selected. The configuration details are as follows:

- Pool name:** EsdeUser
- Required attributes:** email
- Alias attributes:** Choose alias attributes...
- Username attributes:** Choose username attributes...
- Enable case insensitivity?**: Yes
- Custom attributes:** Choose custom attributes...
- Minimum password length:** 8
- Password policy:** uppercase letters, lowercase letters, special characters, numbers
- User sign ups allowed?**: Users can sign themselves up
- FROM email address:** Default
- Email Delivery through Amazon SES:** Yes
- MFA:** Enable MFA...
- Verifications:** Email

At the bottom right of the review section is a blue 'Create pool' button.

Step 5: We were informed that our user pool was successfully created as seen below. We also took note of the Pool Id that we need to use to update our config.js file later on.

The screenshot shows the AWS Cognito User Pools console. The navigation bar at the top includes the AWS logo, 'Services ▾', a search bar ('Search for services, features, marketplace products, and docs'), and account information ('[Alt+S] vocstartsoft/user353347@sara.19@ichat.sp.edu.sg @ 7765-3997-3782 N. Virginia Support'). The main area displays the 'EsdeUser' pool details. A success message 'Your user pool was created successfully.' is shown. Key pool details include the 'Pool Id' (us-east-1_hAllWSGV1) and 'Pool ARN' (arn:aws:cognito-idp:us-east-1:776539973782:userpool/us-east-1_hAllWSGV1). The 'General settings' section shows an 'Estimated number of users' of 0. Under 'App integration', 'Required attributes' are listed as 'email', 'Alias attributes' as 'none', and 'Username attributes' as 'none'. The 'Enable case insensitivity?' option is set to 'Yes'. There is a link to 'Custom attributes' with the option 'Choose custom attributes...'. The 'Minimum password length' is set to 8, and the 'Password policy' is described as 'uppercase letters, lowercase letters, special characters, numbers'. The 'User sign ups allowed?' setting is 'Users can sign themselves up'. A 'Delete pool' button is located in the top right corner.

Step 6: From the Pool Details page of our user pool, we selected “App Clients” from the General Settings section in the left navigation panel. We proceeded to click on the “Add an app client” to connect our web application to the previously created user pool.

The screenshot shows the 'Add an app client' page for the 'EsdeUser' pool. The navigation bar and pool details are identical to the previous screenshot. The left sidebar shows the 'App clients' section is selected. The main content area has a heading 'Which app clients will have access to this user pool?' and a sub-instruction 'The app clients that you add below will be given a unique ID and an optional secret key to access this user pool.' Below this, there is a button labeled 'Add an app client' and a 'Return to pool details' link. The rest of the page is currently empty, indicating no app clients have been added yet.

Step 7: We named our App client “EsdeUserWebApp” and applied the default settings for the Refresh, Access and ID token expiration. We also unchecked the checkbox for “Generate client secret” after learning that client secrets are not supported in terms of usage by browser-based applications.

The screenshot shows the AWS IAM User Pools console. On the left, there's a sidebar with various settings like General settings, Users and groups, Attributes, Policies, MFA and verifications, Advanced security, Message customizations, Tags, Devices, App clients (which is selected), Triggers, Analytics, App integration, App client settings, Domain name, UI customization, Resource servers, Federation, Identity providers, and Attribute mapping. The main area is titled "Which app clients will have access to this user pool?" and contains fields for "App client name" (set to "EsdeUserWebApp"), "Refresh token expiration" (set to "30 days and 0 minutes"), "Access token expiration" (set to "0 days and 60 minutes"), and "ID token expiration" (set to "0 days and 60 minutes"). A checkbox for "Generate client secret" is present but unchecked.

Step 8: We applied the default settings for Auth Flows Configuration and Security Configuration before clicking on the “Create app client” button.

The screenshot shows the "Create app client" configuration page. It has sections for "Auth Flows Configuration" (with checkboxes for "Enable username password auth for admin APIs for authentication (ALLOW_ADMIN_USER_PASSWORD_AUTH)", "Enable lambda trigger based custom authentication (ALLOW_CUSTOM_AUTH)", "Enable username password based authentication (ALLOW_USER_PASSWORD_AUTH)", "Enable SRP (secure remote password) protocol based authentication (ALLOW_USER_SRSP_AUTH)", and "Enable refresh token based authentication (ALLOW_REFRESH_TOKEN_AUTH)"), and "Security configuration" (with a radio button for "Enabled (Recommended)" selected). At the bottom, there are buttons for "Set attribute read and write permissions", "Cancel", and "Create app client".

Step 9: We successfully created the app client as seen below. We also took note of the App client Id that we need to use to update our config.js file later on.

The screenshot shows the AWS User Pools console interface. The top navigation bar includes the AWS logo, Services dropdown, search bar ('Search for services, features, marketplace products, and docs'), and user information ('[Alt+S] vocstartsoft/user353347@sara.19@ichat.ap.edu.sg @ 7765-3997-3782 N. Virginia Support').

The main page title is 'User Pools | Federated Identities' and the specific pool name is 'EsdeUser'. On the left, a sidebar lists various settings: General settings, Users and groups, Attributes, Policies, MFA and verifications, Advanced security, Message customizations, Tags, Devices, **App clients** (which is selected and highlighted in orange), Triggers, Analytics, App integration, Federation, Identity providers, and Attribute mapping.

The central content area is titled 'Which app clients will have access to this user pool?' and contains a note: 'The app clients that you add below will be given a unique ID and an optional secret key to access this user pool.' Below this, there is a table-like structure with two rows:

App client name	EsdeUserWebApp	X
App client id	52lk6cm19dqtejddh94thc6o0l	

Below the table are two buttons: 'Show Details' and 'Add another app client'. In the bottom right corner of the main content area, there is a link 'Return to pool details'.