



UNIVERSITY OF
CENTRAL
MISSOURI
LEARNING TO A GREATER DEGREE

Manage Modern Applications Using AWS Fargate

Abburi Venkata Praveen

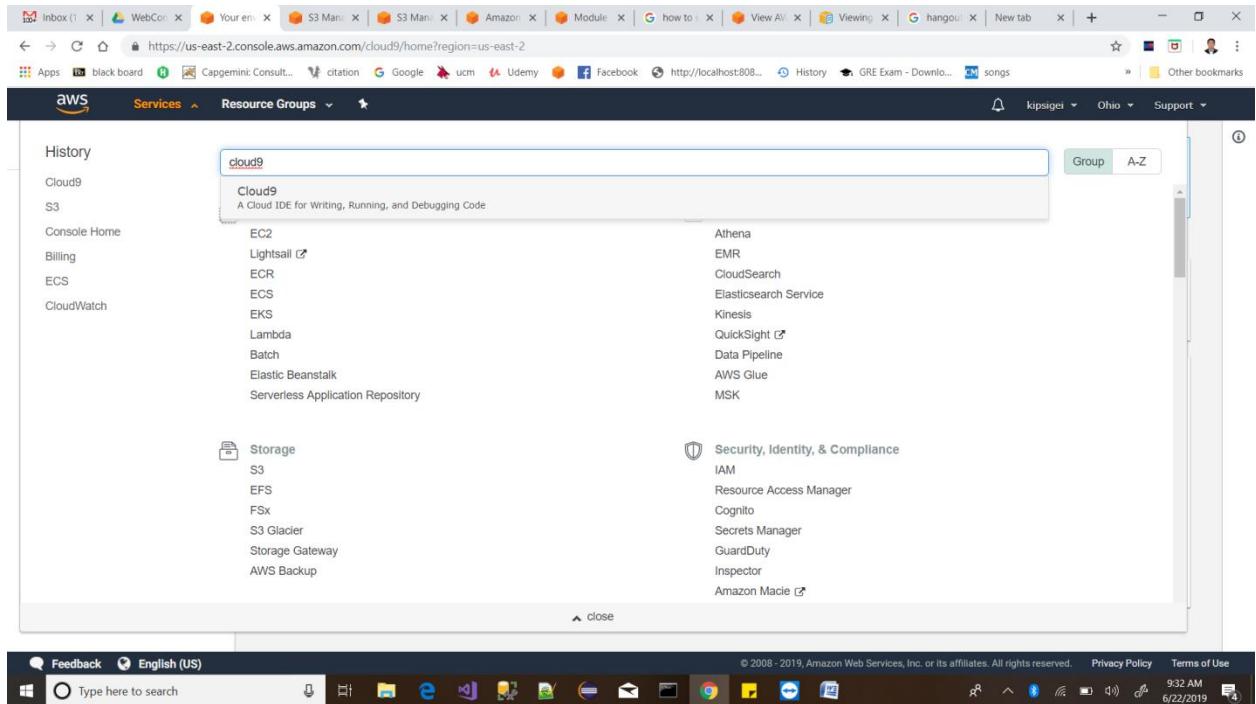
Vxa91760@ucmo.edu

700689176

Step-1:

We need to setup a Cloud 9 IDE.

For the cloud 9 IDE we need to search in services and it will be displayed as below



Step 2: Once we click on cloud9 IDE need to create an environment by selecting create environment. Here we can view the existing IDE's which we have created for testing purpose.

The screenshot shows the AWS Cloud9 dashboard. A prominent message at the top reads: "AWS root account login detected. We do not recommend using your AWS root account to create or work with environments. Use an IAM user instead. This is an AWS security best practice. For more information, see [Setting Up to Use AWS Cloud9](#)". Below this, the "Your environments" section displays three environments: "pr" (Type: EC2, Permissions: Owner), "Ana" (Type: EC2, Permissions: Owner), and "Sam" (Type: EC2, Permissions: Owner). Each environment card includes an "Open IDE" button. The browser's address bar shows the URL: https://us-east-2.console.aws.amazon.com/cloud9/home?region=us-east-2.

Step 3: Here we give the name of the environment

The screenshot shows the "Create environment" process. Step 1 is titled "Name environment". It contains a form for "Environment name and description". The "Name" field is populated with "LiveDemo". The "Description - Optional" field contains "demo project". There are "Cancel" and "Next step" buttons at the bottom. The browser's address bar shows the URL: https://us-east-2.console.aws.amazon.com/cloud9/home/create.

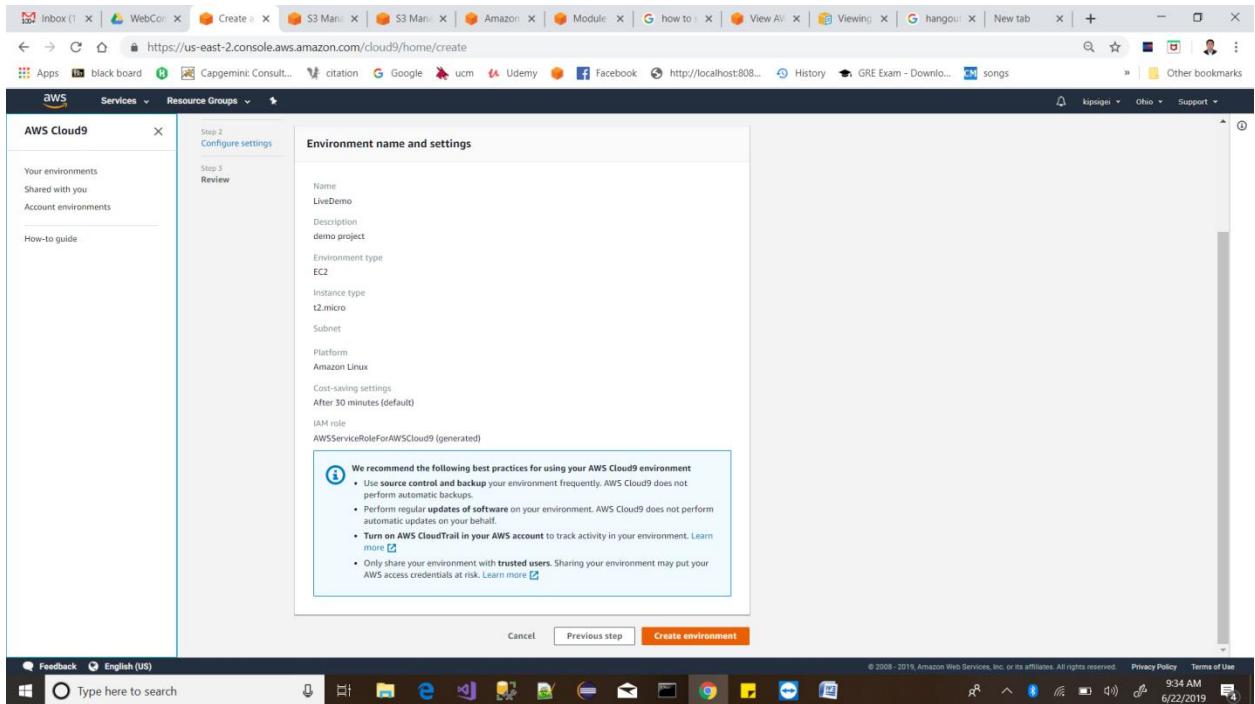
Step 4: Here we have the configuration ready for IDE.

The screenshot shows the 'Configure settings' step of the AWS Cloud9 environment creation wizard. The left sidebar lists 'Your environments', 'Shared with you', and 'Account environments'. The main panel is titled 'Configure settings' and contains the following sections:

- Environment type:** Set to 'Create new instance for environment (EC2)'. Other options include 'Connect and run in remote server (SSH)' and 'Connect and run in local server (SSH)'.
- Instance type:** Set to 't2.micro (1 GB RAM + 1 vCPU)'. Other options include 't2.small (2 GB RAM + 1 vCPU)', 'm4.large (8 GB RAM + 2 vCPU)', and 'Other instance type' (with a dropdown menu showing 't2.nano').
- Platform:** Set to 'Amazon Linux'.
- Cost-saving settings:** Set to 'After 30 minutes (default)'.
- IAM role:** A note states that Cloud9 creates a service-linked role for the environment. It includes a link to 'AWS IAM service-linked role for AWS Cloud9'.
- Network settings (advanced):** A link to 'Edit network settings'.

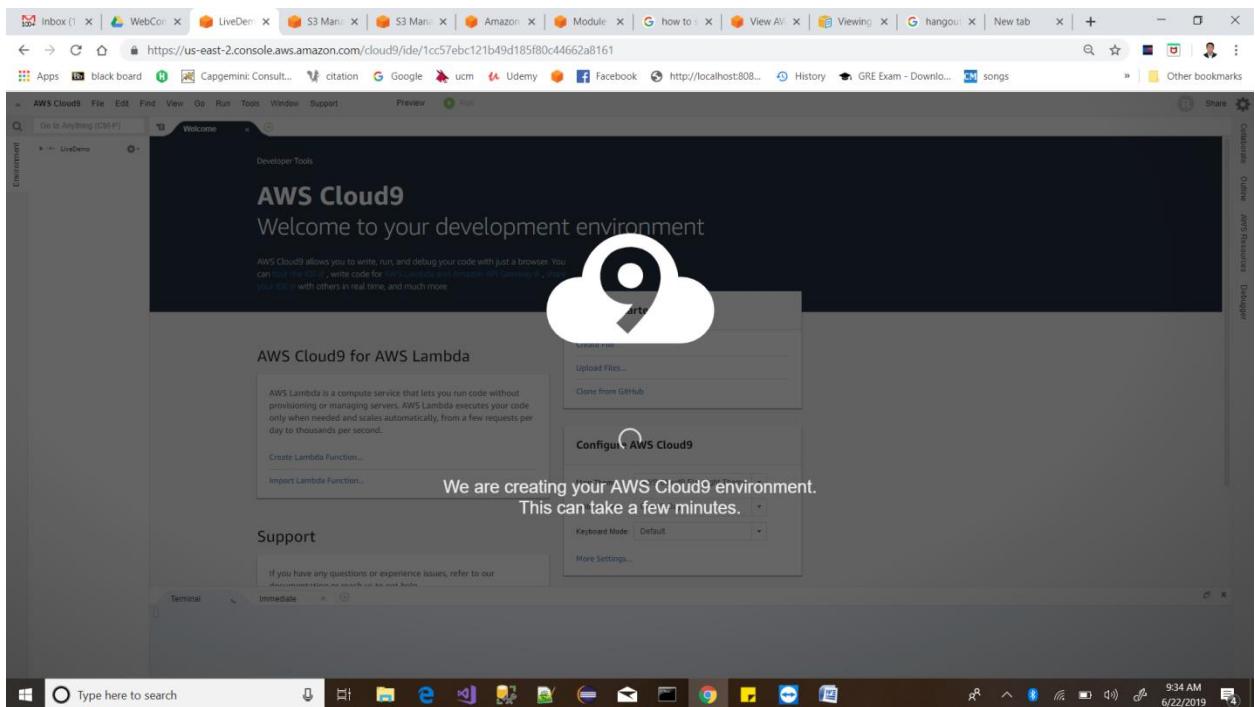
At the bottom are 'Cancel', 'Previous step', and 'Next step' buttons.

Here is the review of IDE Environment.

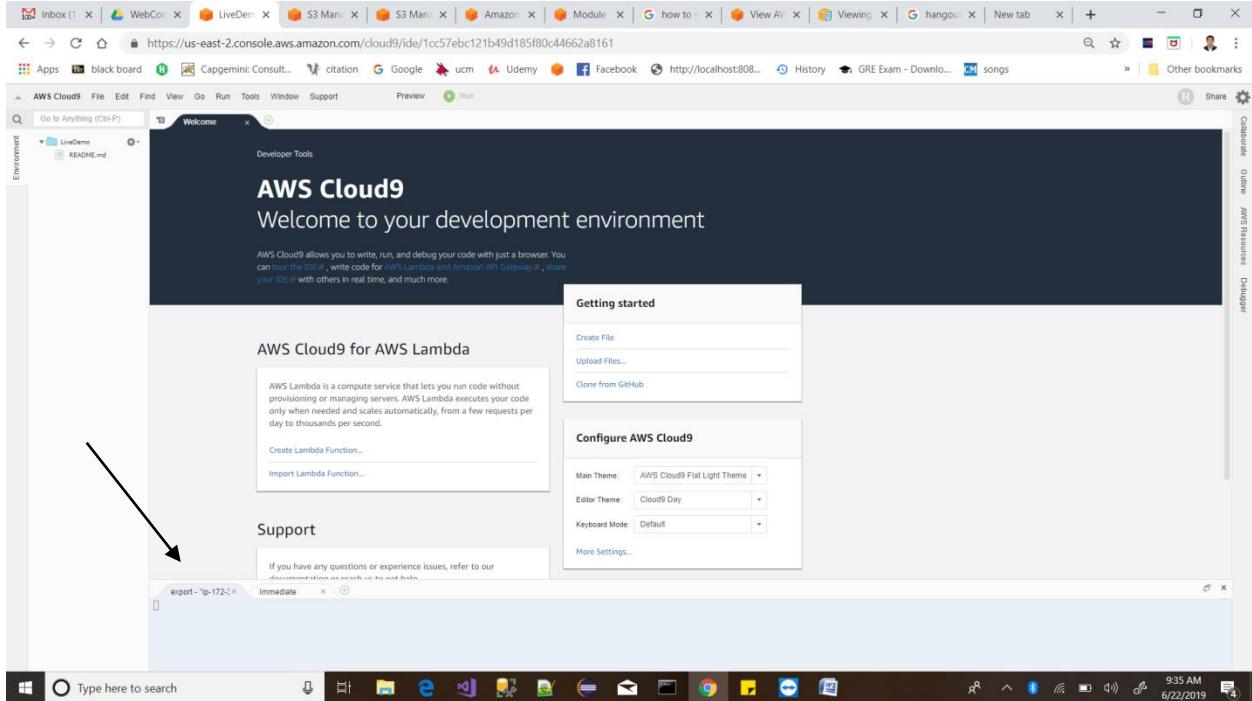


Step 5: continue to creation of IDE. This shows the environment creation step.

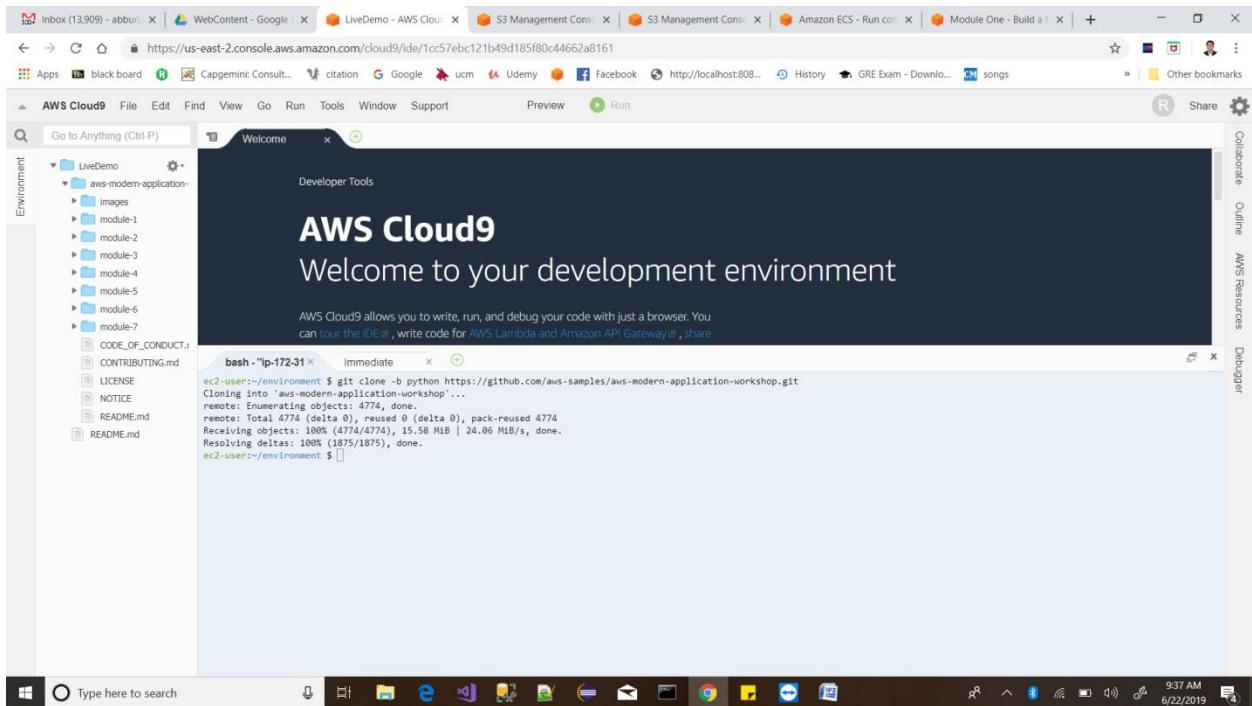
Usually it will take 3 minutes to set up the IDE.



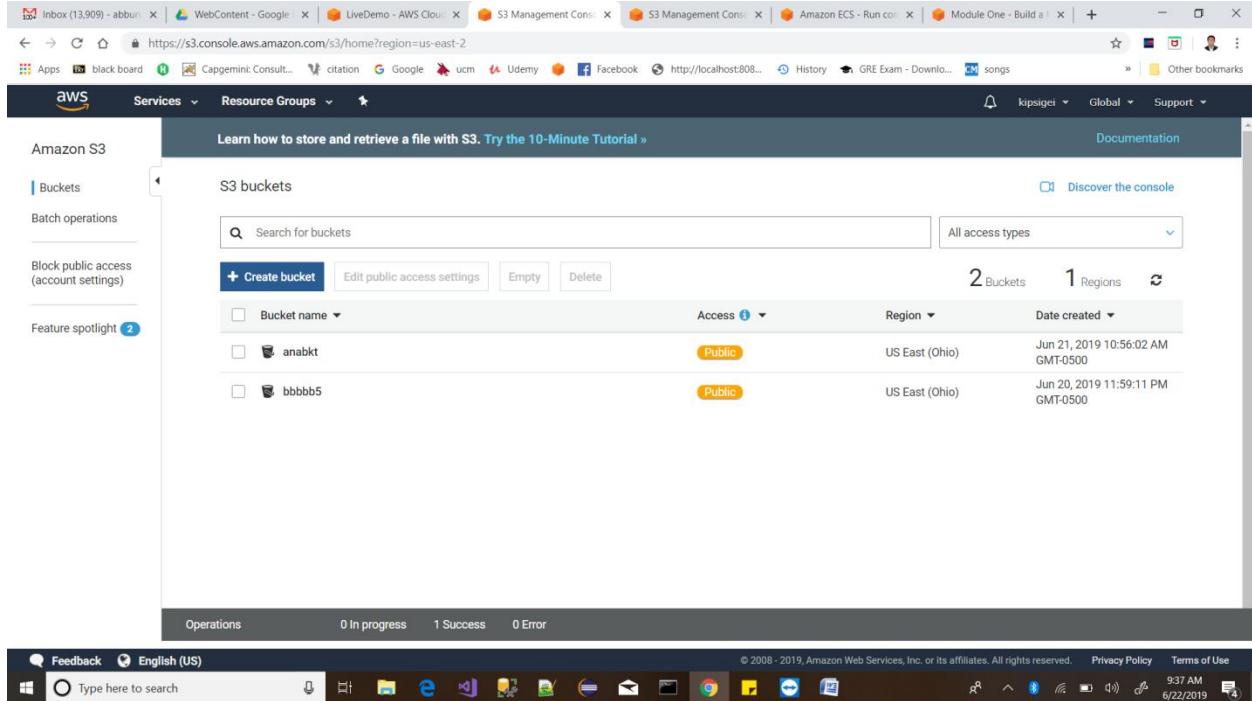
Step 6: Here is the cloud9 development Environment. The down arrow represent the command line interface for the cloud9 environment.



Step 7: First we will deploy the static website on s3 bucket for this one we are downloading the static web application package from github repository.

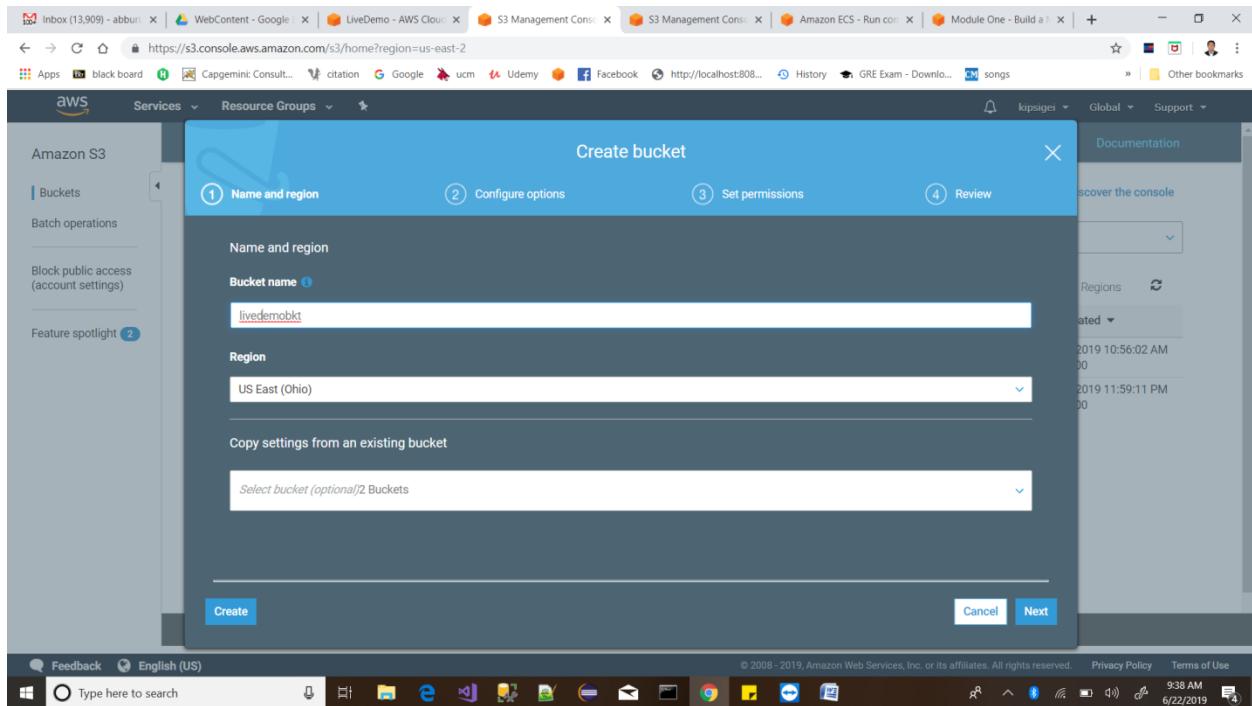


Step 8: Here we create a s3 bucket to store the application files and to host the files. Click on create bucket.



The screenshot shows the AWS S3 Management Console. On the left, there's a sidebar with 'Amazon S3' and options like 'Buckets', 'Batch operations', 'Block public access (account settings)', and 'Feature spotlight'. The main area is titled 'S3 buckets' with a sub-header 'Learn how to store and retrieve a file with S3. Try the 10-Minute Tutorial ». It features a search bar and buttons for '+ Create bucket', 'Edit public access settings', 'Empty', and 'Delete'. Below this is a table showing two buckets: 'anabkt' and 'bbbb5'. The table includes columns for 'Bucket name', 'Access', 'Region', and 'Date created'. The 'anabkt' bucket is listed under 'Public' access. The 'Region' column shows 'US East (Ohio)' for both buckets. The 'Date created' column shows 'Jun 21, 2019 10:56:02 AM GMT-0500' for 'anabkt' and 'Jun 20, 2019 11:59:11 PM GMT-0500' for 'bbbb5'. At the bottom, there are 'Operations' buttons for '0 In progress', '1 Success', and '0 Error'. The status bar at the bottom right shows '9:37 AM 6/22/2019'.

Step 9: Here we name the bucket as (Here I am giving **livedemobkt**) and click on create button. Here the bucket name is unique.



The screenshot shows the 'Create bucket' wizard. Step 1, 'Name and region', is active. It has four tabs: 'Name and region' (selected), 'Configure options', 'Set permissions', and 'Review'. Under 'Name and region', there is a 'Bucket name' field containing 'livedemobkt' and a 'Region' dropdown set to 'US East (Ohio)'. Below these are sections for 'Copy settings from an existing bucket' and a dropdown for selecting a bucket. At the bottom are 'Create' and 'Next' buttons. The status bar at the bottom right shows '9:37 AM 6/22/2019'.

Here the s3 bucket has created.

The screenshot shows the AWS S3 Management Console interface. On the left, there's a sidebar with options like 'Buckets', 'Batch operations', 'Block public access (account settings)', and 'Feature spotlight'. The main area is titled 'Learn how to store and retrieve a file with S3. Try the 10-Minute Tutorial ». It displays a list of 'S3 buckets' with the following details:

Bucket name	Access	Region	Date created
anabkt	public	US East (Ohio)	Jun 21, 2019 10:56:02 AM GMT-0500
bbbbbb5	public	US East (Ohio)	Jun 20, 2019 11:59:11 PM GMT-0500
livedemobkt	Bucket and objects not public	US East (Ohio)	Jun 22, 2019 9:38:26 AM GMT-0500

At the bottom, there are tabs for 'Operations' (0 In progress, 1 Success, 0 Error) and a status bar showing the URL https://s3.console.aws.amazon.com/s3/buckets/livedemobkt/?region=us-east-2&tab=overview, the date 6/22/2019, and the time 9:38 AM.

Step 10: We need to change the permission of the bucket to access publicly and to connect to cloud 9 IDE.

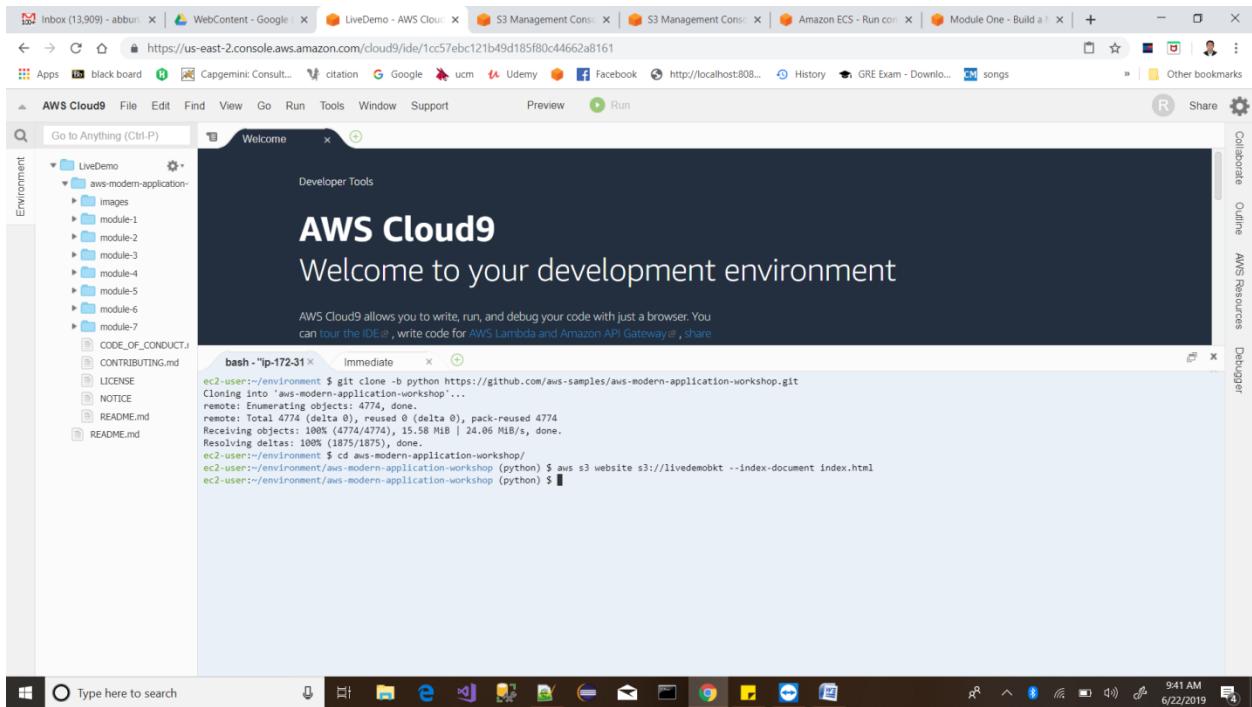
The screenshot shows the 'Permissions' tab of the AWS S3 bucket configuration for 'livedemobkt'. Under the 'Block public access' section, the 'Block all public access' checkbox is selected. A detailed description explains that turning this setting on is the same as turning on all four settings below. The settings listed are:

- 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 using ACLs.
- Block public access to buckets and objects granted through any access control lists (ACLs)**: S3 will ignore all ACLs that grant public access to buckets and objects.
- Block public access to buckets and objects granted through new public bucket policies**: S3 will block new bucket policies that grant public access to buckets and objects. This setting doesn't change any existing policies that allow public access to S3 resources.

At the bottom, there are tabs for 'Overview', 'Properties', 'Permissions', and 'Management'. The 'Permissions' tab is active. The status bar at the bottom shows the URL https://s3.console.aws.amazon.com/s3/buckets/livedemobkt/?region=us-east-2&tab=permissions, the date 6/22/2019, and the time 9:39 AM.

Step 11: To enable the bucket website hosting we set configuration option by running the following command as shown below.

```
aws s3 website s3://livedemobkt --index-document index.html
```



The screenshot shows the AWS Cloud9 IDE interface. On the left, there's a file explorer with a tree view of a project named 'LiveDemo'. The 'aws-modern-application' folder is expanded, showing subfolders like 'images', 'module-1', etc., and files such as 'CODE_OF_CONDUCT.J', 'CONTRIBUTING.md', 'LICENSE', 'NOTICE', 'README.md', and 'README.md'. In the center, the main editor window displays the 'Welcome' page of the AWS Cloud9 environment. At the bottom of the editor, a terminal window titled 'bash - *ip-172-31' is open, showing the command being run and its output:

```
ec2-user:~/environment $ git clone -b python https://github.com/aws-samples/aws-modern-application-workshop.git
Cloning into 'aws-modern-application-workshop'...
remote: Enumerating objects: 4774, done.
remote: Total 4774 (delta 0), reused 0 (delta 0), pack-reused 4774
Receiving objects: 100% (4774/4774), 15.58 MiB | 24.06 MiB/s, done.
Resolving deltas: 100% (1875/1875), done.
ec2-user:~/environment $ cd aws-modern-application-workshop/
ec2-user:~/environment/aws-modern-application-workshop (python) $ aws s3 website s3://livedemobkt --index-document index.html
ec2-user:~/environment/aws-modern-application-workshop (python) $
```

Step 13: The Policy Json file looks like below with the bucket name

```
1 {  
2   "Id": "MyPolicy",  
3   "Version": "2012-10-17",  
4   "Statement": [  
5     {  
6       "Sid": "PublicReadForGetBucketObjects",  
7       "Effect": "Allow",  
8       "Principal": "*",  
9       "Action": "s3:GetObject",  
10      "Resource": "arn:aws:s3:::livedemobkt/*"  
11    }  
12  ]  
13 }
```

```
bash -"ip-172-31" Immediate x  
Cloning into 'aws-modern-application-workshop'...  
remote: Enumerating objects: 4774, done.  
remote: Total 4774 (delta 0), reused 0 (delta 0), pack-reused 4774  
Receiving objects: 100% (4774/4774), 15.58 MiB | 24.00 MiB/s, done.  
Resolving deltas: 100% (1875/1875), done.  
ec2-user:~/environment $ cd aws-modern-application-workshop/  
ec2-user:~/environment/aws-modern-application-workshop (python) $ aws s3 website s3://livedemobkt --index-document index.html  
ec2-user:~/environment/aws-modern-application-workshop (python) $
```

Step 14: Now we need to add bucket policy to our website.

```
aws s3api put-bucket-policy --bucket livedemobkt --policy file://~/environment/aws-modern-application-workshop/module-1/aws-cli/website-bucket-policy.json
```

```
1 {  
2   "Id": "MyPolicy",  
3   "Version": "2012-10-17",  
4   "Statement": [  
5     {  
6       "Sid": "PublicReadForGetBucketObjects",  
7       "Effect": "Allow",  
8       "Principal": "*",  
9       "Action": "s3:GetObject",  
10      "Resource": "arn:aws:s3:::livedemobkt/*"  
11    }  
12  ]  
13 }
```

```
bash -"ip-172-31" Immediate x  
remote: Enumerating objects: 4774, done.  
remote: Total 4774 (delta 0), reused 0 (delta 0), pack-reused 4774  
Receiving objects: 100% (4774/4774), 15.58 MiB | 24.06 MiB/s, done.  
Resolving deltas: 100% (1875/1875), done.  
ec2-user:~/environment $ cd aws-modern-application-workshop/  
ec2-user:~/environment/aws-modern-application-workshop (python) $ aws s3api put-bucket-policy --bucket livedemobkt --policy file://~/environment/aws-modern-application-workshop/module-1/aws-cli/website-bucket-policy.json  
An error occurred (MalformedPolicy) when calling the PutBucketPolicy operation: Policy has invalid resource  
ec2-user:~/environment/aws-modern-application-workshop (python) $ aws s3api put-bucket-policy --bucket livedemobkt --policy file://~/environment/aws-modern-application-workshop/module-1/aws-cli/website-bucket-policy.json  
ec2-user:~/environment/aws-modern-application-workshop (python) $
```

Step 15: Now we need copy the website content from cloud9 IDE to s3 bucket to access static application in s3 bucket.

If we observe the bucket files initially empty.

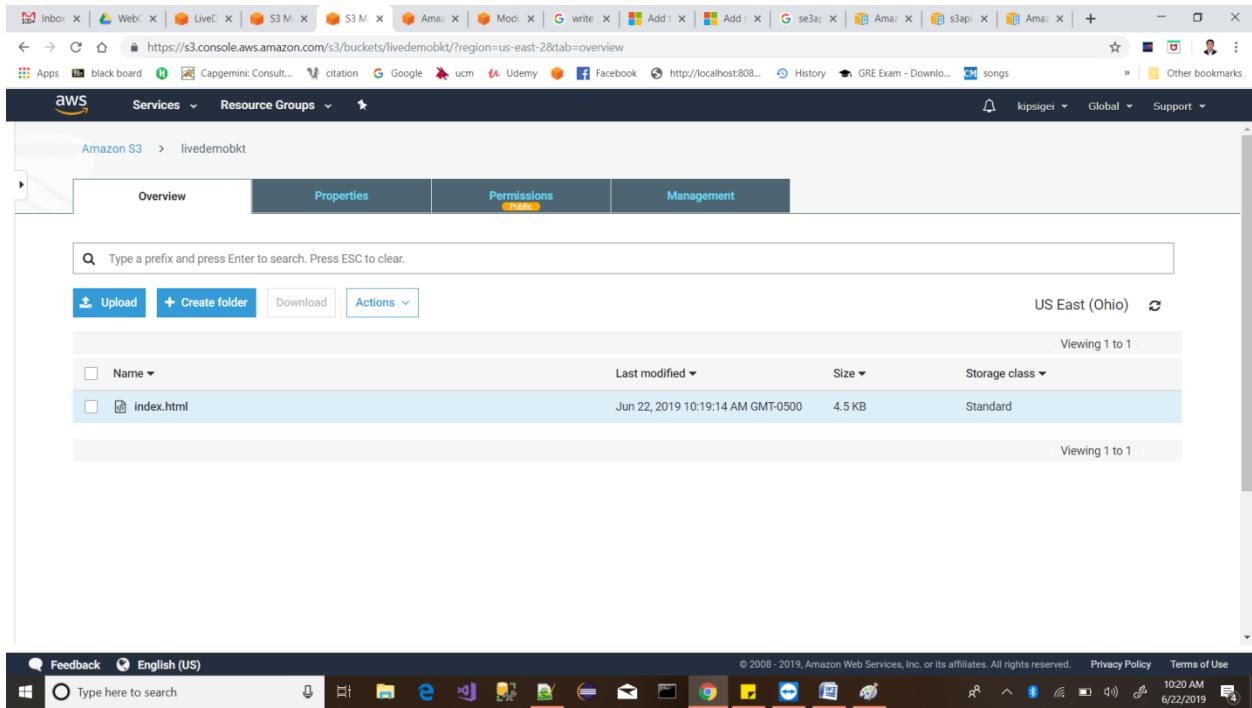
The screenshot shows the AWS S3 console interface. The URL is https://s3.console.aws.amazon.com/s3/buckets/livedemobkt/?region=us-east-2&tab=overview. The page title is "Amazon S3 > livedemobkt". The navigation bar includes "Services", "Resource Groups", and "AWS". The main content area has tabs for "Overview", "Properties", "Permissions" (which is selected), and "Management". Below these are buttons for "Upload", "+ Create folder", "Download", and "Actions". A status message "US East (Ohio)" is shown. The central area displays the message "This bucket is empty. Upload new objects to get started." with three icons: "Upload an object" (a paperclip icon), "Set object properties" (two user icons), and "Set object permissions" (a database icon). Below each icon is a brief description. At the bottom, there's a search bar, a toolbar with various icons, and a footer with copyright information and links to "Feedback", "English (US)", "Privacy Policy", and "Terms of Use".

Step 16: Now we are copying the files from IDE to S3 bucket by running following command.

```
aws s3 cp~/environment/aws-modern-application-workshop/module-1/web/index.html  
s3://livedemobkt/index.html
```

The screenshot shows the AWS Cloud9 IDE interface. The terminal window displays the command "aws s3 cp~/environment/aws-modern-application-workshop/module-1/web/index.html s3://livedemobkt/index.html". The output shows the file being uploaded: "Receiving objects: 100% (1774/1774), 15.58 MB | 24.06 MB/s, done." An error message follows: "An error occurred (MalformedPolicy) when calling the PutBucketPolicy operation: Policy has invalid resource". The terminal also shows the creation of a policy file: "ec2-user:~/environment/aws-modern-application-workshop\$ cd aws-modern-application-workshop/". The status bar at the bottom indicates the time as 10:19 AM and the date as 6/22/2019.

Step 17: Now refresh the s3 bucket we can view the index.html file which we copied from the IDE.



The screenshot shows the AWS S3 console interface. At the top, there's a browser-like header with multiple tabs open. Below it is the AWS navigation bar with 'Services' and 'Resource Groups'. The main area shows the 'Amazon S3' service with the path 'livedemobkt'. There are four tabs: 'Overview', 'Properties', 'Permissions', and 'Management'. The 'Management' tab is selected. A search bar says 'Type a prefix and press Enter to search. Press ESC to clear.' Below it are buttons for 'Upload', '+ Create folder', 'Download', and 'Actions'. To the right, it says 'US East (Ohio)'. A table lists one file: 'index.html' with a size of '4.5 KB' and a last modified date of 'Jun 22, 2019 10:19:14 AM GMT-0500'. The table has columns for 'Name', 'Last modified', 'Size', and 'Storage class'. At the bottom, it says 'Viewing 1 to 1'.

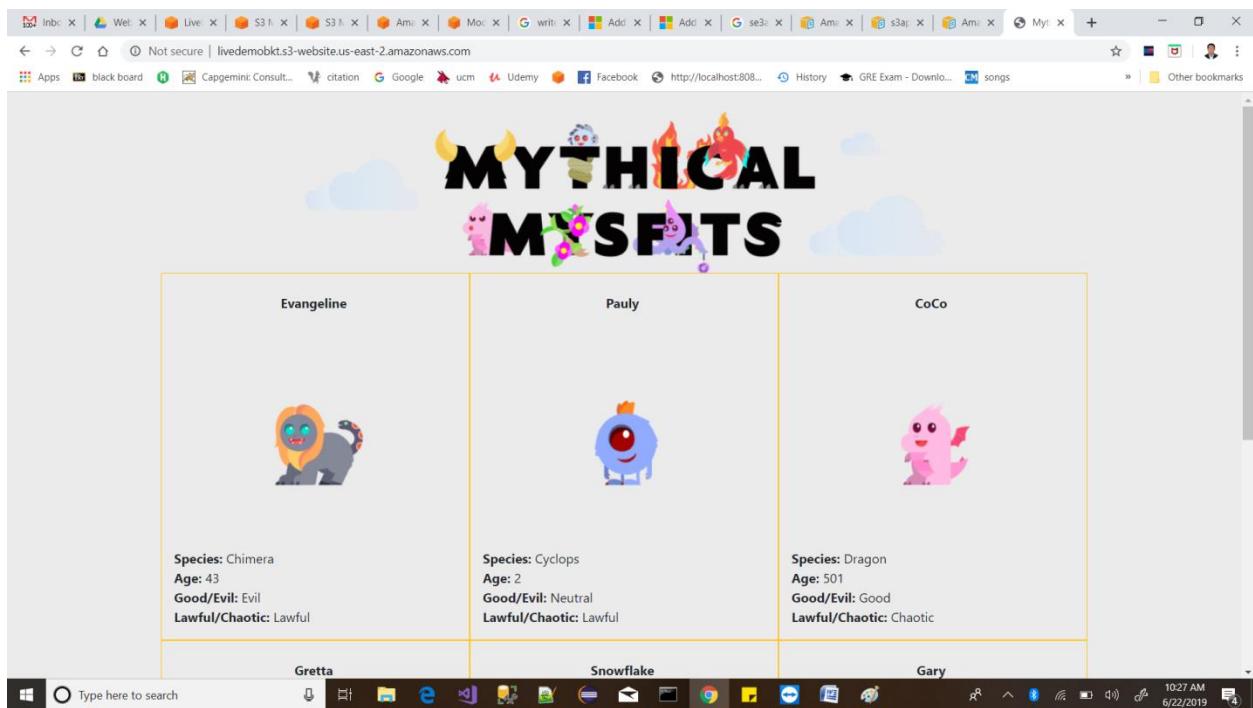
Step 18: Now we can run the application from s3 bucket.

Here we are using ohio (us-east-2) region

after replacing the bucket name and region from above URL:

<http://livedemobkt.s3-website.us-east-2.amazonaws.com>

If we run the above URL in browser ,view the website as below.



Step 19: Now we are going to host a microservice using AWS Fargateserver. Before create a service ,we need setup an infrastructure environment with cloudFormationto build network infrastructure and vpc,iam roles for security,NAT gateways.

Here will create a stack in cloud formation . The stack is collection of resources can managed as a single resources. by creating stack it will internally create & mange VPC,NAT gate ways,iam roles.

If we run the following command to create stack :

```
awscloudformation create-stack --stack-name MythicalMysfitsLiveDemo--capabilities CAPABILITY_NAMED_IAM --template-body file://~/environment/aws-modern-application-workshop/module-2/cfn/core.yml
```

Usually the creation of stack will take 10 minutes.

We can view the status of creation of stack.

```
awscloudformation describe-stacks --stack-name MythicalMysfitsLiveDemo
```

The screenshot shows the AWS Cloud9 IDE interface. On the left, there's a file explorer with a tree view of files and folders under the 'LiveDemo' project. The 'aws-modern-application-workshop' folder contains several sub-folders like 'aws-cli', 'images', 'module-1', etc., and files like 'website-bucket-policy.json', 'README.md', and 'CODE_OF_CONDUCT.md'. In the center, the 'Welcome' page for AWS Cloud9 is displayed with the heading 'AWS Cloud9' and 'Welcome to your development environment'. Below it, a message says 'AWS Cloud9 allows you to write, run, and debug your code with just a browser. You can tour the IDE, write code for AWS Lambda and Amazon API Gateway, share...'. On the right, there are tabs for 'Collaborate', 'Outline', 'AWS Resources', and 'Debugger'. At the bottom, a terminal window titled 'python2.7 - "ip-1"' shows command-line output related to cloning a GitHub repository and creating a CloudFormation stack. The terminal output includes:

```

python2.7 - "ip-1" x Immediate x
Cloning into 'aws-modern-application-workshop...' ...
remote: Enumerating objects: 4774, done.
remote: Total 4774 (delta 0), pack-reused 4774
Receiving objects: 100% (4774/4774), 15.58 MB | 24.06 MB/s, done.
Resolving deltas: 100% (1875/1875), done.
ec2-user:~/environment/aws-modern-application-workshop (python) $ aws s3 website s3://livedemobkt --index-document index.html
ec2-user:~/environment/aws-modern-application-workshop (python) $ aws s3api put-bucket-policy --bucket livedemobkt --policy file://~/environment/aws-modern-application-workshop/module-1/aws-cli/website-bucket-policy.json
An error occurred (MalformedPolicy) when calling the PutBucketPolicy operation: Policy has invalid resource
ec2-user:~/environment/aws-modern-application-workshop (python) $ aws s3api put-bucket-policy --bucket livedemobkt --policy file://~/environment/aws-modern-application-workshop/module-1/aws-cli/website-bucket-policy.json
ec2-user:~/environment/aws-modern-application-workshop (python) $ aws s3 cp ~/environment/aws-modern-application-workshop/module-1/web/index.html to s3://livedemobkt/index.html
ec2-user:~/environment/aws-modern-application-workshop (python) $ aws cloudformation create-stack --stack-name MythicalMysfitsLiveDemo --capabilities CAPABILITY_NAMED_IAM --template-body file://~/environment/aws-modern-application-workshop/module-2/cfn.core.yml
{
  "StackId": "arn:aws:cloudformation:us-east-2:141332078835:stack/MythicalMysfitsLiveDemo/23a43120-950c-11e9-ba16-06e3439fefa0"
}
ec2-user:~/environment/aws-modern-application-workshop (python) $

```

At the bottom of the screen, the Windows taskbar shows various pinned icons like File Explorer, Edge, and File Explorer.

Step 20: Here we can check the status of stack creation with the following command.

`awscloudformation describe-stacks --stack-name MythicalMysfitsLiveDemo`

This screenshot is similar to the previous one but shows the terminal output after the stack has been created. The terminal window titled 'bash - "ip-172-31"' now displays the JSON response from the 'aws cloudformation describe-stacks' command. The output shows a single stack named 'MythicalMysfitsLiveDemo' with its ARN, stack ID, drift information, and creation time. The stack is in the 'CREATE_IN_PROGRESS' state. The terminal output is as follows:

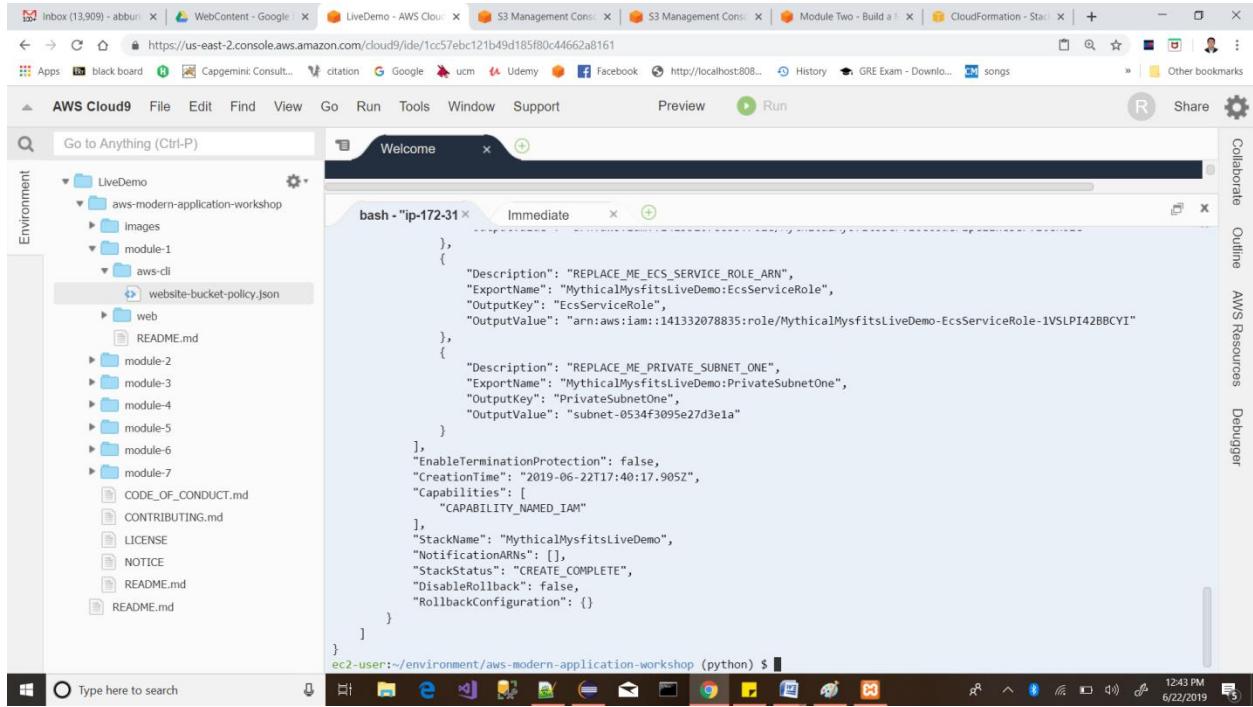
```

,
ec2-user:~/environment/aws-modern-application-workshop (python) $ ^
ec2-user:~/environment/aws-modern-application-workshop (python) $ aws cloudformation describe-stacks --stack-name MythicalMysfit
sLiveDemo
{
  "Stacks": [
    {
      "StackId": "arn:aws:cloudformation:us-east-2:141332078835:stack/MythicalMysfitsLiveDemo/ccc55600-9514-11e9-9c28-020c
f0d94894",
      "DriftInformation": {
        "StackDriftStatus": "NOT_CHECKED"
      },
      "Description": "This stack deploys the core network infrastructure and IAM resources to be used for a service hosted in Amazon ECS using AWS Fargate.",
      "Tags": [],
      "EnableTerminationProtection": false,
      "CreationTime": "2019-06-22T17:40:17.905Z",
      "Capabilities": [
        "CAPABILITY_NAMED_IAM"
      ],
      "StackName": "MythicalMysfitsLiveDemo",
      "NotificationARNs": [],
      "StackStatus": "CREATE_IN_PROGRESS",
      "DisableRollback": false,
      "RollbackConfiguration": {}
    }
  ]
}
ec2-user:~/environment/aws-modern-application-workshop (python) $

```

The taskbar at the bottom remains the same, showing the Windows Start button and pinned icons.

Step 21: We should wait until this step creation before proceeding to next step. We can run multiple times to check status of stack creation. Finally, after 10 minutes stack creation has completed.



The screenshot shows the AWS Cloud9 IDE interface. On the left, there's a file tree for an environment named 'aws-modern-application-workshop'. In the center, a terminal window titled 'bash - "ip-172-31...' displays the JSON output of a CloudFormation describe-stacks command. The output shows details about a stack named 'MythicalMysfitsCoreStack' with a status of 'CREATE_COMPLETE'. The JSON includes fields like 'Outputs', 'Outputs[REPLACE_ME_ECS_SERVICE_ROLE_ARN]', 'Outputs[REPLACE_ME_PRIVATE_SUBNET_ONE]', and 'Outputs[PrivateSubnetOne]'. The terminal prompt is 'ec2-user:~/environment/aws-modern-application-workshop (python) \$'. The bottom of the screen shows a Windows taskbar with various icons.

```
bash - "ip-172-31...  Immediate  +"
},
{
  "Description": "REPLACE_ME_ECS_SERVICE_ROLE_ARN",
  "ExportName": "MythicalMysfitsLiveDemo:EcsServiceRole",
  "OutputKey": "EcsServiceRole",
  "OutputValue": "arn:aws:iam::141332078835:role/MythicalMysfitsLiveDemo-EcsServiceRole-1VSLPI42BBCYI"
},
{
  "Description": "REPLACE_ME_PRIVATE_SUBNET_ONE",
  "ExportName": "MythicalMysfitsLiveDemo:PrivateSubnetOne",
  "OutputKey": "PrivateSubnetOne",
  "OutputValue": "subnet-0534f3095e27d3e1a"
},
],
"EnableTerminationProtection": false,
"CreationTime": "2019-06-22T17:40:17.905Z",
"Capabilities": [
  "CAPABILITY_NAMED_IAM"
],
"StackName": "MythicalMysfitsCoreStack",
"NotificationARNs": [],
"StackStatus": "CREATE_COMPLETE",
"DisableRollback": false,
"RollbackConfiguration": {}
}
]
}
ec2-user:~/environment/aws-modern-application-workshop (python) $
```

Step 22: We can save these out data into json file for furthursteps by running the following command.

```
awscloudformation describe-stacks --stack-name MythicalMysfitsCoreStack>
~/environment/cloudformation-core-output.json
```

we can see these details in cloudfomation click on output tab.

The screenshot shows the AWS CloudFormation console with the 'Outputs' tab selected for the 'MythicalMysfitsLiveDemo' stack. The table displays the following outputs:

Key	Value	Description
CodeBuildRole	arn:aws:iam::141332078835:role/MythicalMysfitsServiceCodeBuildServiceRole	REPLACE_ME_CODEBUILD_ROLE_ARN
CodePipelineRole	arn:aws:iam::141332078835:role/MythicalMysfitsServiceCodePipelineServiceRole	REPLACE_ME_CODEPIPELINE_ROLE_ARN
CurrentAccount	141332078835	REPLACE_ME_ACCOUNT_ID
CurrentRegion	us-east-2	REPLACE_ME_REGION
ECSTaskRole	arn:aws:iam::141332078835:role/MythicalMysfitsLiveDemo-ECSTaskRole-11YKVFT7DOSAO	REPLACE_ME_ECS_TASK_ROLE_ARN
EcsServiceRole	arn:aws:iam::141332078835:role/MythicalMysfitsLiveDemo-EcsServiceRole-1VSLP142BBCYI	REPLACE_ME_ECS_SERVICE_ROLE_ARN
FargateContainerSecurityGroup	sg-006c6d4204a461c9f	REPLACE_ME_SECURITY_GROUP_ID

Step 23: Now we are going to build a service. Docker comes inbuilt with the cloud9 IDE instead of building a new docker locally we will create the image tag with the following command.

Change the path

cd~/environment/aws-modern-application-workshop/module-2/app

docker build . -t 141332078835.dkr.ecr.us-east-2.amazonaws.com/mythicalmysfits/service:latest

If we run the above command it will download and install required dependency packages for application service.

The screenshot shows the AWS Cloud9 IDE interface. On the left, the file tree displays a project structure with folders like 'LiveDemo', 'aws-modern-application-workshop', 'module-1', 'module-2', and 'module-3' containing files such as 'Dockerfile', 'requirements.txt', and 'app.py'. The main editor window shows a Dockerfile with the following content:

```

FROM ubuntu:latest
RUN echo Updating existing packages, installing and upgrading python and pip...
RUN apt-get update -y
RUN apt-get install -y python-pip python-dev build-essential
RUN pip install --upgrade pip
RUN echo Copying the MythicalMysfits Flask service into a service directory...
COPY ./service /MythicalMysfitsService/
RUN echo Installing Python packages listed in requirements.txt...
RUN pip install -r requirements.txt
RUN echo Starting python and starting the Flask service...
ENTRYPOINT ["python"]

```

Below the editor, the terminal window shows the build process:

```

$ docker build -t mythicalmysfits .
Step 1/12 : FROM ubuntu:latest
Step 11/13 : RUN echo Starting python and starting the Flask service...
Step 12/13 : ENTRYPOINT ["python"]

```

The terminal output indicates successful builds for each step.

Successfully built 4009b6ed7d42

Successfully tagged 141332078835.dkr.ecr.us-east-2.amazonaws.com/mythicalmysfits/service:latest

Now we can test this service locally by running following command

```
docker run -p 8080:8080 141332078835.dkr.ecr.us-east-2.amazonaws.com/mythicalmysfits/service:latest
```

here we can view the service output with preview application option from cloud9.

Or we can hit in web browser with the following URL

<https://1cc57ebc121b49d185f80c44662a8161.vfs.cloud9.us-east-2.amazonaws.com/mysfits>

The screenshot shows the AWS Cloud9 IDE interface. On the left, the file structure for 'LiveDemo' is visible, including 'aws-modern-application-workshop', 'module-1', 'module-2', and 'module-3'. The 'Dockerfile' tab is selected, displaying the following Dockerfile content:

```

FROM ubuntu:latest
RUN echo Updating existing packages, installing and upgrading python and pip...
RUN apt-get update -y
RUN apt-get install -y python pip python-dev build-essential
RUN pip install --upgrade pip
RUN echo Copying the MythicalMysfits Flask service into a service directory...
COPY ./service /MythicalMysfitsService
WORKDIR /MythicalMysfitsService
RUN echo Installing Python packages listed in requirements.txt
RUN pip install -r requirements.txt
RUN echo Starting python and starting the Flask service...
ENTRYPOINT ["python"]
CMD ["mythicalMysfitsService.py"]

```

The terminal tab shows the execution of the Dockerfile and the resulting service status:

```

docker - ip-172-  
ec2-user:~/environment/aws-modern-application-workshop/module-2/app$ docker run -p 8080:8080 Successfully tagged 141332078835.dkr.ecr.us-east-2.amazonaws.com/mythicalmysfits/service:latest
docker: invalid reference format: repository name must be lowercase.
See 'docker run --help'.
ec2-user:~/environment/aws-modern-application-workshop/module-2/app$ docker run -p 8080:8080 141332078835.dkr.ecr.us-east-2.amazonaws.com/mythicalmysfits/service:latest
* Running on http://0.0.0.0:8080/ (Press CTRL+C to quit)
172.17.0.1 - - [23/Jun/2019 01:23:32] "GET / HTTP/1.1" 200 -
172.17.0.1 - - [23/Jun/2019 01:23:48] "GET /myfits HTTP/1.1" 200 -
*ec2-user:~/environment/aws-modern-application-workshop/module-2/app$ docker run -p 8080:8080 141332078835.dkr.ecr.us-east-2.amazonaws.com/mythicalmysfits/service:latest
* Running on http://0.0.0.0:8080/ (Press CTRL+C to quit)

```

The system tray at the bottom right indicates the date and time as 6/22/2019 8:25 PM.

Step 24: We can see the running service from IDE browser

The screenshot shows the AWS Cloud9 IDE interface with the 'Preview File' tab selected. It displays the JSON configuration for the 'mythicalMysfits' service, specifically the 'chimera' character:

```

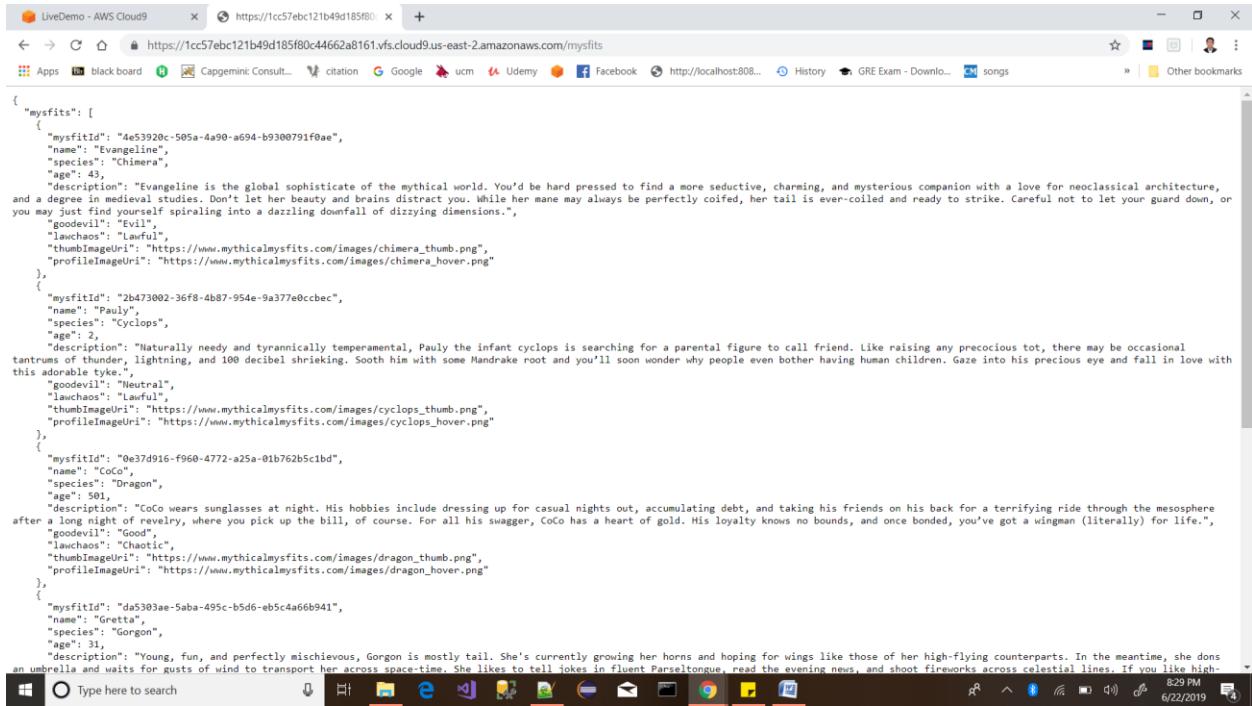
{
  "myfitId": "4e53920c-505a-4a90-a694-b9300791f0ae",
  "name": "Evangeline",
  "species": "Chimera",
  "age": 43,
  "description": "Evangeline is the global sophisticate of the mythical world. You'd be hard pressed to find a more seductive, charming, and mysterious companion with a love for neoclassical architecture, and a degree in medieval studies. Don't let her beauty and brains distract you. While her mane may always be perfectly coiffed, her tail is ever-coiled and ready to strike. Careful not to let your guard down, or you may just find yourself spiraling into a dazzling downfall of dizzying dimensions.",
  "gender": "Female",
  "lawchos": "Lawful",
  "thumbImageUri": "https://www.mythicalmysfits.com/images/chimera_thumb.png",
  "profileImageUri": "https://www.mythicalmysfits.com/images/chimera_hover.png"
},
{
  "myfitId": "2b473002-36f8-4b67-954e-9a377e0ccbec",
  "name": "Chimera"
}

```

The browser tab shows the preview URL: <https://1cc57ebc121b49d185f80c44662a8161.wf.e.cloud9.us-east-2.amazonaws.com>. The terminal tab shows the service is running on port 8080.

The system tray at the bottom right indicates the date and time as 6/22/2019 8:34 PM.

Step 25: The service output from docker image from browser view.

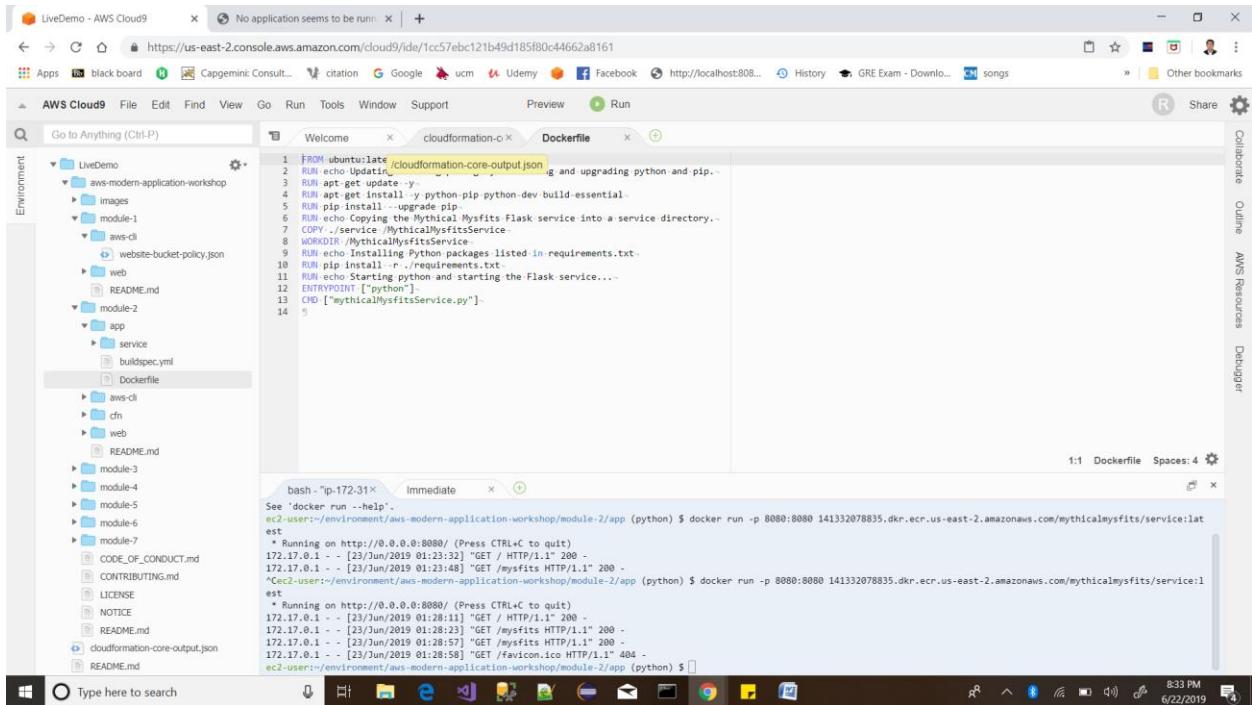


The screenshot shows a Microsoft Edge browser window with the URL <https://1cc57ebc121b49d185f80c44662a8161.vfs.cloud9.us-east-2.amazonaws.com/mysfits>. The page displays a JSON document representing a collection of mythical creatures (Mysfits). The JSON structure includes fields like myfitId, name, species, age, description, gender, and URLs for thumbs and profile images. The description for each creature is detailed, such as 'Evangeline' being a global sophisticate and 'Pauly' being naturally needy and tyrannically temperamental.

```
{
  "mysfits": [
    {
      "myfitId": "4e53920c-505a-4a90-a694-b9300791f0ae",
      "name": "Evangeline",
      "species": "Chimera",
      "age": 43,
      "description": "Evangeline is the global sophisticate of the mythical world. You'd be hard pressed to find a more seductive, charming, and mysterious companion with a love for neoclassical architecture, and a degree in medieval studies. Don't let her beauty and brains distract you. While her mane may always be perfectly coifed, her tail is ever-coiled and ready to strike. Careful not to let your guard down, or you may just find yourself spiraling into a dazzling downfall of dizzying dimensions.",
      "gender": "Female",
      "luchchas": "Useful",
      "thumbImageUri": "https://www.mythicalmysfits.com/images/chimera_thumb.png",
      "profileImageUri": "https://www.mythicalmysfits.com/images/chimera_hover.png"
    },
    {
      "myfitId": "b2473002-36f8-4b87-954e-9a377e0ccbec",
      "name": "Pauly",
      "species": "Cyclops",
      "age": 10,
      "description": "Naturally needy and tyrannically temperamental, Pauly the infant cyclops is searching for a parental figure to call friend. Like raising any precocious tot, there may be occasional tantrums of thunder, lightning, and 100 decibel shrieking. Sooth him with some Mandrake root and you'll soon wonder why people even bother having human children. Gaze into his precious eye and fall in love with this adorable tyke.",
      "gender": "Neutral",
      "luchchas": "Useful",
      "thumbImageUri": "https://www.mythicalmysfits.com/images/cyclops_thumb.png",
      "profileImageUri": "https://www.mythicalmysfits.com/images/cyclops_hover.png"
    },
    {
      "myfitId": "0e37d916-f960-4772-a25a-01b762b5c1bd",
      "name": "CoCo",
      "species": "Dragon",
      "age": 50,
      "description": "CoCo wears sunglasses at night. His hobbies include dressing up for casual nights out, accumulating debt, and taking his friends on his back for a terrifying ride through the mesosphere after a long night of revelry, where you pick up the bill, of course. For all his swagger, CoCo has a heart of gold. His loyalty knows no bounds, and once bonded, you've got a wingman (literally) for life.",
      "gender": "Female",
      "luchchas": "Chaotic",
      "thumbImageUri": "https://www.mythicalmysfits.com/images/dragon_thumb.png",
      "profileImageUri": "https://www.mythicalmysfits.com/images/dragon_hover.png"
    },
    {
      "myfitId": "da5303ae-5aba-495c-b5d6-eb5c4a6b941",
      "name": "Gretta",
      "species": "Gorgon",
      "age": 20,
      "description": "Young, fun, and perfectly mischievous, Gorgon is mostly tail. She's currently growing her horns and hoping for wings like those of her high-flying counterparts. In the meantime, she dons an umbrella and waits for gusts of wind to transport her across space-time. She likes to tell jokes in fluent Parseltongue, read the evening news, and shoot fireworks across celestial lines. If you like high-", "gender": "Female"
    }
  ]
}
```

Step 26: We can observe this response from the service that returns the JSON document stored at `/aws-modern-application-workshop/module-2/app/service/mysfits-response.json`

Once testing is done, we can press **ctrl+c** to stop the docker service.



The screenshot shows the AWS Cloud9 IDE interface. On the left, the file tree shows the project structure with files like Dockerfile, requirements.txt, and various configuration files. The central pane shows the Dockerfile content:

```
FROM ubuntu:latest
RUN echo Updating, installing python and upgrading python and pip...
RUN apt-get update -y
RUN apt-get install -y python-pip python-dev build-essential
RUN pip install --upgrade pip
RUN echo Copying the Mythical Mysfits Flask Service into a service directory...
COPY ./service /mythicalMysfitsService
WORKDIR /mythicalMysfitsService
RUN echo Installing python packages listed in requirements.txt
RUN pip install -r ./requirements.txt
RUN echo Starting python and starting the Flask service...
ENTRYPOINT ["python"]
CMD ["mythicalMysfitsService.py"]
```

The right pane shows the terminal output of running the Dockerfile:

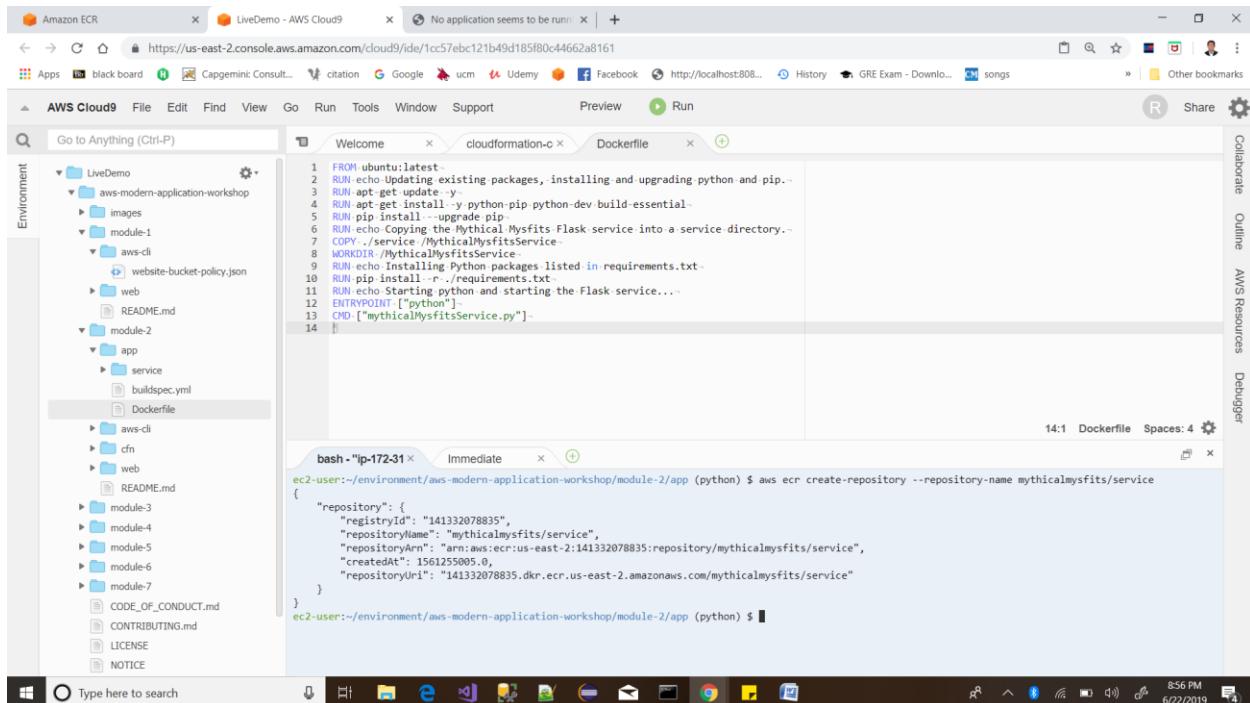
```
bash -ip-172-31x Immediate x
See 'docker run --help'.
ec2-user:~/environment/aws-modern-application-workshop/module-2/app (python) $ docker run -p 8080:8080 141332078835.dkr.ecr.us-east-2.amazonaws.com/mythicalmysfits/service:latest
* Running on http://0.0.0.0:8080/ (Press CTRL+C to quit)
172.17.0.1 - - [23/Jun/2019 01:23:32] "GET / HTTP/1.1" 200 -
172.17.0.1 - - [23/Jun/2019 01:23:48] "GET /mysfits HTTP/1.1" 200 -
*ec2-user:~/environment/aws-modern-application-workshop/module-2/app (python) $ docker run -p 8080:8080 141332078835.dkr.ecr.us-east-2.amazonaws.com/mythicalmysfits/service:latest
* Running on http://0.0.0.0:8080/ (Press CTRL+C to quit)
172.17.0.1 - - [23/Jun/2019 01:28:11] "GET / HTTP/1.1" 200 -
172.17.0.1 - - [23/Jun/2019 01:28:23] "GET /mysfits HTTP/1.1" 200 -
172.17.0.1 - - [23/Jun/2019 01:28:57] "GET /mysfits HTTP/1.1" 200 -
172.17.0.1 - - [23/Jun/2019 01:28:58] "GET /favicon.ico HTTP/1.1" 404 -
*ec2-user:~/environment/aws-modern-application-workshop/module-2/app (python) $
```

Step 27: Now we need to push this docker image into container image repository in Amazon elastic container registry (ECR).

Amazon Elastic Container Registry (ECR) is a fully-managed Docker container registry that makes it easy for developers to store, manage, and deploy Docker container images.

By running the following command, creates a new repository in the default AWS ECR registry created for your account.

```
aws ecr create-repository --repository-name mythicalmysfits/service
```



The screenshot shows the AWS Cloud9 IDE interface. On the left, the file structure of a project named 'aws-modern-application-workshop' is visible, including 'LiveDemo', 'aws-modern-application-workshop', 'images', 'module-1' (containing 'aws-cli', 'website-bucket-policy.json', 'web', and 'Dockerfile'), 'module-2' (containing 'app', 'aws-cli', 'cfn', 'web', and 'README.md'), 'module-3', 'module-4', 'module-5', 'module-6', 'module-7', 'CODE_OF_CONDUCT.md', 'CONTRIBUTING.md', 'LICENSE', and 'NOTICE'. In the center, a terminal window titled 'bash - "ip-172-31' is open, showing the command: `aws ecr create-repository --repository-name mythicalmysfits/service`. The output of the command is displayed below the command line, showing the creation of a new repository with details like repository ID, name, ARN, creation date, and URL. On the right, there are tabs for 'Collaborate', 'Outline', 'AWS Resources', and 'Debugger'. The status bar at the bottom indicates the time as 8:56 PM and the date as 6/2/2019.

Step 28: After running the above command repository has been created and we can see the created repository in aws ECR.

The screenshot shows the AWS ECR (Amazon Container Registry) service in the AWS Management Console. The left sidebar shows 'Amazon Container Services' with 'Amazon ECR' selected. The main pane displays a table titled 'Repositories (1 of 1)' with one item listed:

Repository name	URI	Created at
mythicalmysfits/service	141332078835.dkr.ecr.us-east-2.amazonaws.com/mythicalmysfits/service	06/22/19, 08:56:45 PM

Step 29: Now we need to push the docker image into repository with the following command.

To push image file into Repository we need authentication to with docker by executing following command.

`$ aws ecr get-login --no-include-email)`

The screenshot shows the AWS Cloud9 IDE interface. On the left, the file tree shows a project structure with several modules (module-1, module-2, module-3, module-4, module-5, module-6, module-7) and associated files like Dockerfile, buildspec.yml, and README.md. In the center, the 'Dockerfile' tab is open, displaying the Dockerfile content:

```

FROM ubuntu:latest
RUN echo Updating existing packages, installing and upgrading python and pip...
RUN apt-get update -y
RUN apt-get install -y python-pip python-dev build-essential
RUN pip install --upgrade pip
RUN echo Copying the MythicalMysfits Flask service into a service directory...
COPY ./service /MythicalMysfitsService-
WORKDIR /MythicalMysfitsService-
RUN echo Installing Python packages listed in requirements.txt
RUN pip install -r requirements.txt
RUN echo Starting python and starting the Flask service...
ENTRYPOINT ["python"]
CMD ["mythicalMysfitsService.py"]

```

Below the Dockerfile, the terminal window shows the output of the command:

```

bash - "ip-172-31-1-10" Immediate
$ $(aws ecr get-login --no-include-email)
WARNING! Using --password via the CLI is insecure. Use --password-stdin.
WARNING! Your password will be stored unencrypted in /home/ec2-user/.docker/config.json.
Configure a credential helper to remove this warning. See
https://docs.docker.com/engine/reference/commandline/login/#credentials-store

Login Succeeded
ec2-user:~/environment/aws-modern-application-workshop/module-2/app (python) $

```

Step 30: Now we are pushing the docker image to ECR repository.

```
docker push 141332078835.dkr.ecr.us-east-2.amazonaws.com/mythicalmysfits/service:latest
```

The screenshot shows the AWS Cloud9 IDE interface. On the left, the file tree displays a project structure with modules, configuration files, and a Dockerfile. The Dockerfile content is as follows:

```
FROM ubuntu:latest
RUN echo Updating existing packages, installing and upgrading python and pip...
RUN apt-get update -y
RUN apt-get install -y python-pip python-dev build-essential
RUN pip install --upgrade pip
RUN echo Copying the MythicalMysfits Flask service into a service directory...
COPY ./service /MythicalMysfitsService
WORKDIR /MythicalMysfitsService
RUN echo Installing Python packages listed in requirements.txt
RUN pip install -r ./requirements.txt
RUN echo Starting python and starting the Flask service...
ENTRYPOINT ["python"]
CMD ["mythicalMysfitsService.py"]
```

In the bottom terminal window, the command `docker push` is run, and the output shows the image being pushed to the ECR repository:

```
ec2-user:~/environment/aws-modern-application-workshop/module-2/app (python) $ docker push 141332078835.dkr.ecr.us-east-2.amazonaws.com/mythicalmysfits/service:latest
The push refers to repository [141332078835.dkr.ecr.us-east-2.amazonaws.com/mythicalmysfits/service]
aa85e75a5d27: Preparing
aa85e75a5d27: Pushed
13cc05649660: Pushed
5ba32150be00: Pushed
44d26594a7: Pushed
634990f72318: Pushed
75a79aa52609: Pushed
ddaf51859818: Pushed
fb2d732ad777: Pushed
ba9de90847: Pushed
latest: digest: sha256:ac2a5936787df3ba878762d191e71fb3a10ecc87f5b97e01ac95810a36e0eac0 size: 2208
ec2-user:~/environment/aws-modern-application-workshop/module-2/app (python) $
```

Step 31: Now we can view the docker image in ECR Repository.

The screenshot shows the AWS Cloud9 IDE interface. The terminal window displays the JSON response from the ECR API, which provides details about the pushed Docker image:

```
{
  "imageDetails": [
    {
      "imageSizeInBytes": 194101041,
      "imageDigest": "sha256:ac2a5936787df3ba878762d191e71fb3a10ecc87f5b97e01ac95810a36e0eac0",
      "imageTags": [
        "latest"
      ],
      "registryId": "141332078835",
      "repositoryName": "mythicalmysfits/service",
      "imagePushedAt": 1561255179.0
    }
  ]
}
```

The timestamp in the JSON output is 1561255179.0, corresponding to June 22, 2019, at 9:02 PM.

Step 32: Now service available in local. We are going to deploy the service hosted on Amazon ECS using AWS Fargate and will make the service available to publicly behind Network load balancer.

First, we need to create cluster in ECS. This represents the cluster of “servers” that your service containers will be deployed to. Servers is in "quotations" because you will be using AWS Fargate. Fargate allows us to specify that containers be deployed to a cluster without having to provision or manage any servers yourself.

To create a cluster, we need to execute below command.

```
awsecs create-cluster --cluster-name MythicalMysfitslivedemo-Cluster
```

if we observe the before running command , The ECS console shows available clusters and services.

The screenshot shows the AWS Management Console with the URL <https://us-east-2.console.aws.amazon.com/ecs/home?region=us-east-2#/clusters>. The left sidebar is the Amazon ECS navigation menu. The main content area is titled 'Clusters' and shows the 'MythicalMysfits-Cluster'. Below the title, it says 'FARGATE'. It displays the following metrics:

EC2	Running tasks	Pending tasks	CPUUtilization	MemoryUtilization	Container instances
1 Services	1	0	No data	No data	0

Step 33: Now we are executing the command to create the cluster

The screenshot shows the AWS Cloud9 IDE with the URL <https://us-east-2.console.aws.amazon.com/cloud9/ide/1cc57ebc121b49d185f80c44662a8161>. The left sidebar shows the project structure for 'LiveDemo - AWS Cloud9'. The right pane contains a terminal window with the following command and output:

```
bash -ip-172-31 x Immediate x
ec2-user:~/environment $ aws ecs create-cluster --cluster-name MythicalMysfitslivedemo-Cluster
{
  "cluster": {
    "status": "ACTIVE",
    "statistics": [],
    "tags": [],
    "clusterName": "MythicalMysfitslivedemo-Cluster",
    "registeredContainerInstancesCount": 0,
    "pendingTasksCount": 0,
    "runningTasksCount": 0,
    "activeServicesCount": 0,
    "clusterArn": "arn:aws:ecs:us-east-2:14133207835:cluster/MythicalMysfitslivedemo-Cluster"
  }
}
ec2-user:~/environment $
```

Step 34: Now we can see that the cluster is created in ECS console.

The screenshot shows the AWS Management Console with the ECS service selected. It displays two clusters: 'MythicalMysfits-Cluster' and 'MythicalMysfitslivedemo-Cluster'. Both clusters are using the FARGATE provider. The first cluster has 1 service, 1 running task, and 0 pending tasks. The second cluster has 0 services, 0 running tasks, and 0 pending tasks. There is no data for CPUUtilization and MemoryUtilization, and 0 container instances.

Step 35: Next we need to create the logs for the serviceThat we are configuring with AWS cloudWatch Logs. It is a collection of logs and we can monitor the system behavior in graph format.Execute the following command.

```
aws logs create-log-group --log-group-name mythicalmysfits-logs
```

The screenshot shows the AWS Cloud9 IDE interface. On the left, there's a file tree with files like Dockerfile, README.md, and various module files. In the center, a terminal window is open with the following command being run:

```
aws logs create-log-group --log-group-name mythicalmysfits-logs
```

Step 36: Now we can see the cloud watch logs file in AWS cloud watch console.

The screenshot shows the AWS CloudWatch Management Console interface. The left sidebar is titled 'Logs' and includes options like 'CloudWatch', 'Dashboards', 'Alarms', 'Event Buses', 'Logs', 'Metrics', 'Settings', 'Favorites', and 'Add a dashboard'. The main content area is titled 'Log Groups' and shows two log groups: 'mythicalmysfits-logs' and 'mythicalmysfits-logs1'. Both groups have the 'Explore' status, 'Never Expire' for 'Expire Events After', and 0 filters and subscriptions. A search bar at the top says 'Filter: Log Group Name Prefix'.

Step 37: By selecting a specific log group we can see the logs of service. Below screen shot is sample of existing service. But the newly created one logs are empty.

The screenshot shows the AWS CloudWatch Management Console with the 'Log Event Viewer' open for the 'mythicalmysfits-logs' log group. A modal window titled 'Try CloudWatch Logs Insights' provides information about the service. The main view shows a table with columns 'Time (UTC +00:00)' and 'Message'. The table lists numerous log entries from June 21, 2019, at 05:42:51, all showing the same message: 'Running on http://0.0.0.0:8080/ (Press CTRL+C to quit)'. The log entries range from 05:42:51 to 05:43:56.

Step 38: Now we have created cluster and a log group where our container logs will be pushed to and we're ready to register an ECS task definition. A task in ECS is a set of container images

that should be scheduled together. A task definition declares that set of containers and the resources and configuration those containers require. You will use the AWS CLI to create a new task definition for how your new container image should be scheduled to the ECS cluster we just created.

A JSON file has been provided that will serve as the input to the CLI command.

Open `~/environment/aws-modern-application-workshop/module-2/aws-cli/task-definition.json` in the IDE.

Here we need replace the image tags in the above file and we need to run the following command.

```
awsecs register-task-definition --cli-input-json file:///~/environment/aws-modern-application-workshop/module-2/aws-cli/task-definition.json
```

Step 39:

These values we can see in cloud formation template

The screenshot shows the AWS CloudFormation console with the 'Outputs' section selected. The 'Outputs (12)' table lists the following key-value pairs:

Key	Value
CodeBuildRole	arn:aws:iam::141332078835:role/MythicalMysfitsServiceCodeBuildServiceRole
CodePipelineRole	arn:aws:iam::141332078835:role/MythicalMysfitsServiceCodePipelineServiceRole
CurrentAccount	141332078835
CurrentRegion	us-east-2
ECSTaskRole	arn:aws:iam::141332078835:role/MythicalMysfitsLiveDemo-ECSTaskRole
EcsServiceRole	arn:aws:iam::141332078835:role/MythicalMysfitsLiveDemo-EcsServiceRole
FargateContainerSecurityGroup	sg-006c6d4204a461c9f

Step 40: The task definition file has shown as below

```

{
  "cluster": {
    "status": "ACTIVE",
    "statistics": [],
    "tags": [],
    "clusterName": "MythicalMysfitsliveDemo-Cluster",
    "registeredContainerInstancesCount": 0,
    "pendingTasksCount": 0,
    "runningTasksCount": 0,
    "activeServicesCount": 0,
    "clusterArn": "arn:aws:ecs:us-east-2:141332078835:cluster/MythicalMysfitsliveDemo-Cluster"
  }
}

```

```

ec2-user:~/environment $ aws ecs create-cluster --cluster-name MythicalMysfitsliveDemo-Cluster
ec2-user:~/environment $ aws logs create-log-group --log-group-name mythicalmysfitsliveDemo-logs

```

Step 41: Now we are executing the below command to register an ECS Task definition.

```

{
  "taskDefinition": {
    "status": "ACTIVE",
    "memory": "512",
    "networkMode": "awsvpc",
    "family": "mythicalmysfitservice",
    "placementConstraints": [],
    "requiresAttributes": [
      {
        "name": "ecs.capability.execution-role-ecr-pull"
      }
    ]
  }
}

```

```

python2.7 -"ip-1" python2.7 -"ip-1" $ aws logs create-log-group --log-group-name mythicalmysfits-logs
ec2-user:~/environment $ aws ecs register-task-definition --cli-input-json file://~/environment/aws-modern-application-workshop/module-2/aws-cli/task-definition.json

```

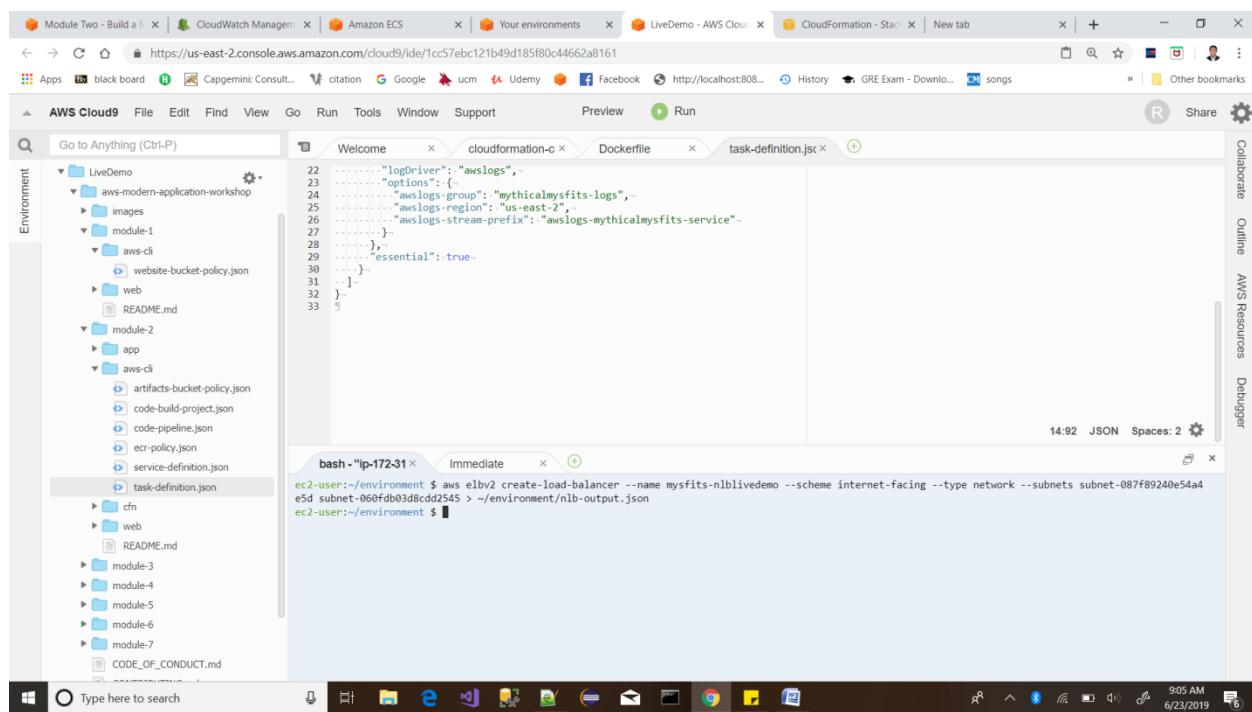
Step 42: Now we need to create the load balancer to fargate service.

If we create a Network load balancer with public access any one can access the website or services from outside the network.

To provision a NLB we need to execute following command.

```
aws elbv2 create-load-balancer --name mysfits-nlblivedemo --scheme internet-facing --type network --subnets subnet-087f89240e54a4e5d subnet-060fdb03d8cdd2545 > ~/environment/nlb-output.json
```

After executing the command we can see the nlb-output.json file in the environment. In this file we can see the DNS name and Loadbalncer ARN names



The screenshot shows the AWS Cloud9 IDE interface. The left sidebar displays a file tree for a project named 'LiveDemo'. The current file being edited is 'task-definition.json'. The code in the editor is as follows:

```
22     "logDriver": "awslogs",
23     "options": {
24       "awslogs-group": "mythicalmysfits-logs",
25       "awslogs-region": "us-east-2",
26       "awslogs-stream-prefix": "awslogs-mythicalmysfits-service"
27     },
28   },
29   "essential": true,
30 },
31 ],
32 }
33 }
```

Below the editor, a terminal window titled 'bash - "ip-172-31' shows the command that was run:

```
ec2-user:~/environment $ aws elbv2 create-load-balancer --name mysfits-nlblivedemo --scheme internet-facing --type network --subnets subnet-087f89240e54a4e5d subnet-060fdb03d8cdd2545 > ~/environment/nlb-output.json
ec2-user:~/environment $
```

Step 43: After executing the command we can see the nlb-output.json file in the environment. In this file we can see the DNS name and Loadbalncer ARN names

```

1   "LoadBalancers": [
2     {
3       "IpAddressType": "ipv4",
4       "VpcId": "vpc-00fa4cdf225ffa69",
5       "LoadBalancerArn": "arn:aws:elasticloadbalancing:us-east-2:141332078835:loadbalancer/net/mysfits-nlblivedemo/b03dca91add7c25a",
6       "State": {
7         "Code": "provisioning"
8       },
9       "DNSName": "mysfits-nlblivedemo-b03dca91add7c25a.elb.us-east-2.amazonaws.com",
10      "LoadBalancerName": "mysfits-nlblivedemo",
11      "CreatedTime": "2019-06-23T14:05:55.075Z",
12      "Scheme": "internet-facing",
13      "Type": "network",
14      "CanonicalHostedZoneId": "ZLMOA37VPKAMP",
15      "AvailabilityZones": [
16        {
17          "SubnetId": "subnet-066fdb03d8ccdd2545",
18          "ZoneName": "us-east-2a"
19        },
20        {
21          "SubnetId": "subnet-087f89240e54a4e5d",
22          "ZoneName": "us-east-2a"
23        }
24      ]
25    }
26  ],
27  "task-definition.json"
28
29 bash -lpi-172-31x  Immediate
30 ec2-user:~/environment $ aws elbv2 create-load-balancer --name mysfits-nlblivedemo --scheme internet-facing --type network --subnets subnet-087f89240e54a4e5d
31 ec2-user:~/environment $ aws elbv2 create-target-group --name MythicalMysfitsld-TargetGroup --port 8080 --protocol TCP --target-type ip --vpc-id vpc-00fa4cdf225ffa69 --health-check-interval-seconds 10 --health-check-path / --health-check-protocol HTTP --healthy-threshold-count 3 --unhealthy-threshold-count 3 > ~/environment/target-group-output.json
32 ec2-user:~/environment $ 

```

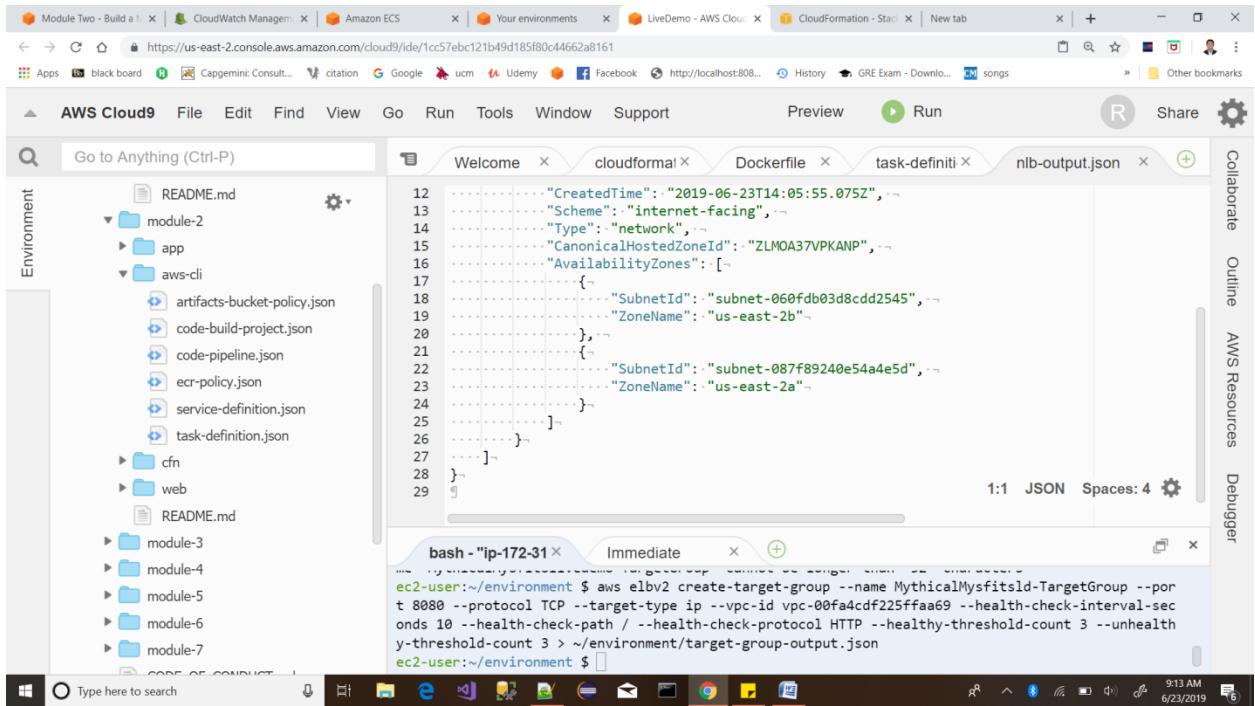
Step 44: Next, use the CLI to create an NLB target group. A target group allows AWS resources to register themselves as targets for requests that the load balancer receives to forward.

For creating target group need to execute following command.

```

aws elbv2 create-target-group --name MythicalMysfitsld-TargetGroup --port 8080 --protocol TCP --target-type ip --vpc-id vpc-00fa4cdf225ffa69 --health-check-interval-seconds 10 --health-check-path / --health-check-protocol HTTP --healthy-threshold-count 3 --unhealthy-threshold-count 3 > ~/environment/target-group-output.json

```



Step 45: Now target-group-output.json file has created this consist of target group ARN Values and vpv id as follows:

"TargetGroups": [

```

{
  "HealthCheckPath": "/",
  "HealthCheckIntervalSeconds": 10,
  "VpcId": "vpc-00fa4cdf225ffaa69",
  "Protocol": "TCP",
  "HealthCheckTimeoutSeconds": 6,
  "TargetType": "ip",
  "HealthCheckProtocol": "HTTP",
  "Matcher": {
    "HttpCode": "200-399"
  },
  "UnhealthyThresholdCount": 3,
  "HealthyThresholdCount": 3,
}
```

```

    "TargetGroupArn": "arn:aws:elasticloadbalancing:us-east-
2:141332078835:targetgroup/MyticalMysfitsld-TargetGroup/97b6288f0abc266d",
    "HealthCheckEnabled": true,
    "HealthCheckPort": "traffic-port",
    "Port": 8080,
    "TargetGroupName": "MyticalMysfitsld-TargetGroup"
}

}

```

Step

Now we need to make this loadbalncer in Listening state:

In the following command need to replace the target group ARN which is there in target-group-output.json and Network Loadbalancer ARN which is aviable at nlb-output.json .

```

aws elbv2 create-listener --default-actions TargetGroupArn=arn:aws:elasticloadbalancing:us-
east-2:141332078835:targetgroup/MyticalMysfitsld-
TargetGroup/97b6288f0abc266d,Type=forward --load-balancer-arn
arn:aws:elasticloadbalancing:us-east-2:141332078835:loadbalancer/net/mysfits-
nlblivedemo/b03dca91add7c25a --port 80 --protocol TCP

```

```

{
  "LoadBalancers": [
    {
      "IpAddressType": "ipv4",
      "VpcId": "vpc-00f4acdff25ffa69",
      "LoadBalancerArn": "arn:aws:elasticloadbalancing:us-east-2:141332078835:loadbalancer/net/mysfits-nlblivedemo/b03dca91add7c25a",
      "State": {
        "Code": "provisioning"
      },
      "DNSName": "mysfits-nlblivedemo-b03dca91add7c25a.elb.us-east-2.amazonaws.com",
      "LoadBalancerName": "mysfits-nlblivedemo",
      "CreatedTime": "2019-06-23T14:05:55.075Z",
      "Scheme": "internet-facing",
      "Type": "network",
      "CanonicalHostedZoneId": "ZLMOA37VPKAMP",
      "AvailabilityZones": [
        {
          "SubnetId": "subnet-060fdb03d8cdd2545",
          "ZoneName": "us-east-2b"
        }
      ],
      "SubnetId": "subnet-087ff89240e54a4e5d"
    }
  ]
}

ec2-user:~/environment $ aws elbv2 create-listener --default-actions TargetGroupArn=arn:aws:elasticloadbalancing:us-east-2:141332078835:targetgroup/MyticalMysfitsld-TargetGroup/97b6288f0abc266d,Type=forward --load-balancer-arn arn:aws:elasticloadbalancing:us-east-2:141332078835:loadbalancer/net/mysfits-nlblivedemo/b03dca91add7c25a --port 80 --protocol TCP
{
  "Listeners": [
    {
      "Protocol": "TCP",
      "DefaultActions": [
        {
          "TargetGroupArn": "arn:aws:elasticloadbalancing:us-east-2:141332078835:targetgroup/MyticalMysfitsld-TargetGroup/97b6288f0abc266d",
          "Type": "forward"
        }
      ],
      "LoadBalancerArn": "arn:aws:elasticloadbalancing:us-east-2:141332078835:loadbalancer/net/mysfits-nlblivedemo/b03dca91add7c25a",
      "Port": 80
    }
  ]
}

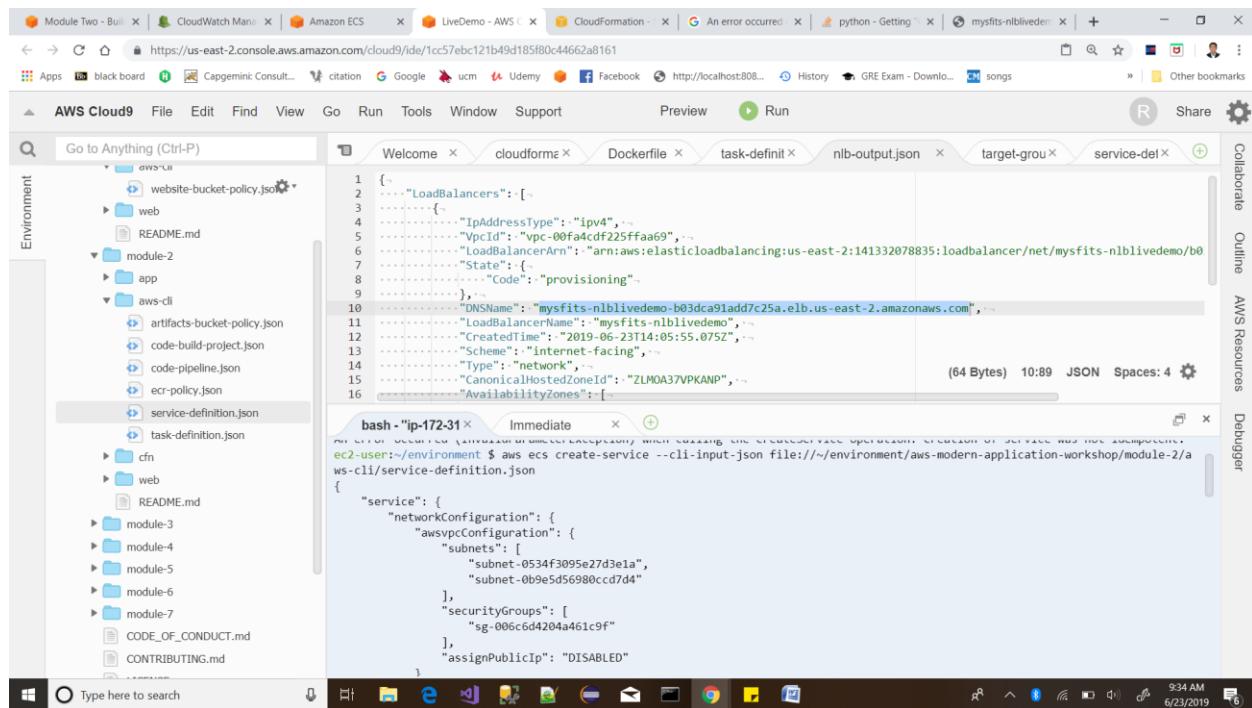
```

Step 46:

Grant the permission to the service with iam role.

```
awsiam create-service-linked-role --aws-service-name ecs.amazonaws.com
```

NLB has created and configured.now create the ECS service to be created and that service should be launched with AWS Fargate by running the following command.



```
1 {
2     "LoadBalancers": [
3         {
4             "IpAddressType": "ipv4",
5             "VpcId": "vpc-00fa4acd25ffaa69",
6             "LoadBalancerArn": "arn:aws:elasticloadbalancing:us-east-2:141332078835:loadbalancer/net/mysfits-nlblivedemo/b03dca91add7c25a",
7             "State": {
8                 "Code": "provisioning"
9             }
10            },
11            {
12                "DNSName": "mysfits-nlblivedemo-b03dca91add7c25a.elb.us-east-2.amazonaws.com",
13                "LoadBalancerName": "mysfits-nlblivedemo",
14                "CreatedTime": "2019-06-23T14:05:55.075Z",
15                "Scheme": "internet-facing",
16                "Type": "network",
17                "CanonicalHostedZoneId": "ZLMOA37VPKANP",
18                "AvailabilityZones": [
19                    "us-east-2a"
20                ]
21            }
22        ],
23        "service": {
24            "networkConfiguration": {
25                "awsvpcConfiguration": {
26                    "subnets": [
27                        "subnet-0534f3095e27d3e1a",
28                        "subnet-0b9e5d5d6980cccd7d4"
29                    ],
30                    "securityGroups": [
31                        "sg-006c6d4204a461c9f"
32                    ],
33                    "assignPublicIp": "DISABLED"
34                }
35            }
36        }
37    }
38}
```

```
bash *ip-172-31-1-1* Immediate
```

```
An error occurred (ResourceNotFoundException) when calling the CreateService operation: Creation of service was not successful.
ec2-user:~/environment $ aws ecs create-service --cli-input-json file://~/environment/aws-modern-application-workshop/module-2/aws-ml/service-definition.json
{
    "service": {
        "networkConfiguration": {
            "awsvpcConfiguration": {
                "subnets": [
                    "subnet-0534f3095e27d3e1a",
                    "subnet-0b9e5d5d6980cccd7d4"
                ],
                "securityGroups": [
                    "sg-006c6d4204a461c9f"
                ],
                "assignPublicIp": "DISABLED"
            }
        }
    }
}
```

Step 47:Now we can test the service with DNS configured for the service

<http://mysfits-nlblivedemo-b03dca91add7c25a.elb.us-east-2.amazonaws.com/mysfits>

we can see the response from server and the service output displayed in browser.

```

{
  "mysfits": [
    {
      "myfitId": "4e53920c-505a-4a90-a694-b9300791f0ae",
      "name": "Evangeline",
      "species": "Chimera",
      "age": 43,
      "description": "Evangeline is the global sophisticate of the mythical world. You'd be hard pressed to find a more seductive, charming, and mysterious companion with a love for neoclassical architecture, and a degree in medieval studies. Don't let her beauty and brains distract you. While her mane may always be perfectly coifed, her tail is ever-coiled and ready to strike. Careful not to let your guard down, or you may just find yourself spiraling into a dazzling downfall of dizzying dimensions.",
      "goodevil": "Evil",
      "lawchaos": "Lawful",
      "thumbImageUri": "https://www.mythicalmysfits.com/images/chimera_thumb.png",
      "profileImageUri": "https://www.mythicalmysfits.com/images/chimera_hover.png"
    },
    {
      "myfitId": "2b473002-36f8-4b87-954e-9a377e0ccbec",
      "name": "Pauly",
      "species": "Cyclops",
      "age": 100,
      "description": "Naturally needy and tyrannically temperamental, Pauly the infant cyclops is searching for a parental figure to call friend. Like raising any precocious tot, there may be occasional tantrums of thunder, lightning, and 100 decibel shrieking. Sooth him with some Mandrake root and you'll soon wonder why people even bother having human children. Gaze into his precious eye and fall in love with this adorable tyke.",
      "goodevil": "Neutral",
      "lawchaos": "Lawful",
      "thumbImageUri": "https://www.mythicalmysfits.com/images/cyclops_thumb.png",
      "profileImageUri": "https://www.mythicalmysfits.com/images/cyclops_hover.png"
    },
    {
      "myfitId": "0e37d916-f960-4772-a25a-01b762b5c1bd",
      "name": "CoCo",
      "species": "Dragon",
      "age": 501,
      "description": "CoCo wears sunglasses at night. His hobbies include dressing up for casual nights out, accumulating debt, and taking his friends on his back for a terrifying ride through the mesosphere after a long night of revelry, where you pick up the bill, of course. For all his swagger, CoCo has a heart of gold. His loyalty knows no bounds, and once bonded, you've got a wingman (literally) for life.",
      "goodevil": "Good",
      "lawchaos": "Chaotic",
      "thumbImageUri": "https://www.mythicalmysfits.com/images/dragon_thumb.png",
      "profileImageUri": "https://www.mythicalmysfits.com/images/dragon_hover.png"
    },
    {
      "myfitId": "da5303ae-5aba-495c-b5d6-eb5c4a6b941",
      "name": "Gretta",
      "species": "Gorgon",
      "age": 31,
      "description": "Young, fun, and perfectly mischievous, Gorgon is mostly tail. She's currently growing her horns and hoping for wings like those of her high-flying counterparts. In the meantime, she dons an umbrella and waits for gusts of wind to transport her across space-time. She likes to tell jokes in fluent Parseltongue, read the evening news, and shoot fireworks across celestial lines. If you like high-", "goodevil": "Neutral", "lawchaos": "Lawful", "thumbImageUri": "https://www.mythicalmysfits.com/images/gorgon_thumb.png", "profileImageUri": "https://www.mythicalmysfits.com/images/gorgon_hover.png"
    }
  ]
}

```

Step 48: Now we can see the service available in the cluster

Cluster	Services	Running tasks	Pending tasks	CPUUtilization	MemoryUtilization	Container instances
FARGATE	0	0	0	No data	No data	0
EC2	1	1	0	No data	No data	0
MythicalMysfits-Cluster1	0	0	0	No data	No data	0

Step 49: We can observe the logs of service.

Screenshot of the AWS CloudWatch Logs Insights interface.

The navigation bar shows:

- Module Two - Build
- CloudWatch Metrics
- Amazon ECS
- LiveDemo - AWS
- CloudFormation
- An error occurred
- python - Getting started
- mysfits-nblivedev

The browser tabs include:

- Apps
- black board
- Capgemini: Consult...
- citation
- Google
- ucm
- Udemy
- Facebook
- http://localhost:808...
- History
- GRE Exam - Downlo...
- songs
- Other bookmarks

The AWS CloudWatch interface shows:

- Services: CloudWatch, Resource Groups
- Resource Groups: mythicalmysfits-logs
- Log Stream: awslogs-mythicalmysfits-service/MyticalMysfits-Service/727e68f5-d84f-4f8f-b434-db7bce093e7f

A modal window titled "Try CloudWatch Logs Insights" provides information about the new log query language:

CloudWatch Logs Insights allows you to search and analyze your logs using a new, purpose-built query language. Click [here](#) to experience it. If you want to learn more, read the [AWS blog](#) or visit [our documentation](#).

The main log viewer table displays the following data:

Time (UTC +00:00)	Message
2019-06-23	10.0.1.187 - [23/Jun/2019 14:33:37] "GET / HTTP/1.1" 200 - 10.0.1.187 - [23/Jun/2019 14:33:38] "GET / HTTP/1.1" 200 - 10.0.1.187 - [23/Jun/2019 14:33:39] "GET / HTTP/1.1" 200 -

Below the table are filter options: "Expand all" (radio button selected), "Row" (radio button), "Text" (radio button), and a search bar with "all" and "2019-06-22 (14:30:37)" dropdowns.

The bottom of the screen shows the Windows taskbar with various pinned icons and the system tray indicating the date and time as 6/23/2019 at 9:37 AM.