😊 Amazon SageMaker

Amazon SageMaker is a fully managed machine learning (ML) service provided by AWS that helps developers and data scientists to quickly and efficiently build, train, and deploy ML models at scale. It is designed to simplify the complex steps involved in the ML lifecycle—data preparation, model building, training, tuning, and deployment—without the need to manage underlying infrastructure.

Core Features of Amazon SageMaker:

1. **SageMaker Studio**: A web-based integrated development environment (IDE) that allows users to build, train, and deploy models from a single interface. It supports the full ML workflow, including code editing, experimentation, training, tuning, and monitoring.
2. **SageMaker Autopilot**: Automatically builds, trains, and tunes the best machine learning models based on your data, allowing you to get started without deep ML expertise.
3. **SageMaker Data Wrangler**: Helps streamline the data preparation process, allowing you to import, transform, and analyze your data quickly.
4. **SageMaker Experiments**: Tracks and organizes experiments, helping you compare different model versions and settings to find the best-performing model.
5. **SageMaker Model Monitor**: Continuously monitors deployed models for data and performance drift, ensuring the model predictions stay accurate over time.
6. **SageMaker Debugger**: Provides insights into model training, automatically identifying and notifying you of issues such as overfitting, so you can take corrective actions.
7. **SageMaker Pipelines**: Helps automate ML workflows by defining, automating, and scaling end-to-end ML pipelines, from data preparation to model training and deployment.
8. **Built-in Algorithms**: SageMaker provides optimized implementations of popular machine learning algorithms such as XGBoost, k-means clustering, linear regression, and image classification.
9. **Elastic Inference**: Allows you to reduce the cost of deploying machine learning models by scaling inference acceleration across different instance types.

Advantages of Amazon SageMaker:

1. **End-to-End Machine Learning Workflow**: SageMaker covers all stages of the machine learning pipeline, from data preparation and model training to deployment and monitoring, which helps developers manage the full lifecycle in one platform.
2. **Managed Infrastructure**: SageMaker abstracts the complexity of managing and scaling the underlying infrastructure for training and deploying models, so developers can focus on the ML part without worrying about resource allocation or server setup.

3. **Supports Multiple Frameworks**: SageMaker supports popular ML and deep learning frameworks like TensorFlow, PyTorch, MXNet, and Scikit-learn. Users can also bring their own algorithms and custom frameworks.
4. **Pay-As-You-Go**: You only pay for the resources you use, such as computing power for training or inference, without the need to invest in expensive on-premises hardware.
5. **Distributed Training**: SageMaker can distribute large training jobs across multiple machines, allowing faster training for complex models with large datasets.
6. **Automatic Model Tuning (Hyperparameter Optimization)**: SageMaker's hyperparameter tuning automatically searches for the best model parameters, helping to optimize model performance.
7. **Flexible Deployment Options**: SageMaker enables you to deploy models in various ways, such as real-time endpoints for interactive applications, batch transformations for large datasets, or multi-model endpoints to host multiple models on the same instance.

Use Cases of Amazon SageMaker:

1. **Financial Services (Fraud Detection)**: SageMaker can be used to build models that detect fraudulent transactions in real-time, helping banks and financial institutions reduce fraud risk by analyzing patterns of behavior.
2. **Healthcare (Medical Imaging)**: In healthcare, SageMaker can help build models that analyze medical images (e.g., X-rays, MRIs) to detect diseases or abnormalities like tumors, automating and speeding up the diagnostic process.
3. **Retail (Personalized Recommendations)**: Retailers can use SageMaker to analyze customer behavior and purchase history to provide personalized product recommendations, increasing sales and customer satisfaction.
4. **Manufacturing (Predictive Maintenance)**: SageMaker can help manufacturers predict equipment failure by analyzing machine sensor data, reducing downtime and maintenance costs.
5. **Autonomous Vehicles (Model Training for Self-driving Cars)**: Companies working on autonomous driving can use SageMaker to train deep learning models that process sensor data from cameras and LiDAR to help vehicles navigate.

Disadvantages of Amazon SageMaker:

1. **Complexity for Beginners**: Although SageMaker simplifies many aspects of ML, it still requires a certain level of ML and AWS expertise to use effectively, especially when dealing with custom models and advanced features.
2. **Cost Management**: While SageMaker offers cost-efficiency, managing costs can become a challenge, particularly for long-running training jobs, large datasets, or high-demand inference models. Mismanaging resource allocation (e.g., using large, expensive instances unnecessarily) can result in higher-than-expected costs.
3. **Dependency on AWS Ecosystem**: SageMaker works best within the AWS ecosystem. If you are using non-AWS tools or services, integration may be more challenging.

Migrating workloads or integrating with third-party cloud providers can add complexity.

4. **Limited Control for Custom Solutions**: While SageMaker's pre-built tools and algorithms are powerful, some users might find them restrictive if they need more granular control or customization over their ML pipelines, such as specific hardware choices or algorithm optimizations.

Example:

Example 1: Building a Sentiment Analysis Model

Let's say an e-commerce company wants to understand customer feedback by analyzing product reviews. Using Amazon SageMaker, they can:

1. Import customer review data from Amazon S3.
2. Use SageMaker's built-in natural language processing (NLP) algorithms to train a sentiment analysis model.
3. Deploy the model using SageMaker's real-time endpoint feature.
4. Continuously monitor the model using SageMaker Model Monitor to ensure its predictions remain accurate.

Example 2: Image Recognition for Quality Control

A manufacturing company wants to detect defects in products from images taken on the assembly line. Using SageMaker:

1. They upload images of products to Amazon S3.
2. Use SageMaker to train a custom image classification model that can detect whether a product is defective or not.
3. Deploy the model to detect defects in real-time, improving the quality control process and reducing human error.

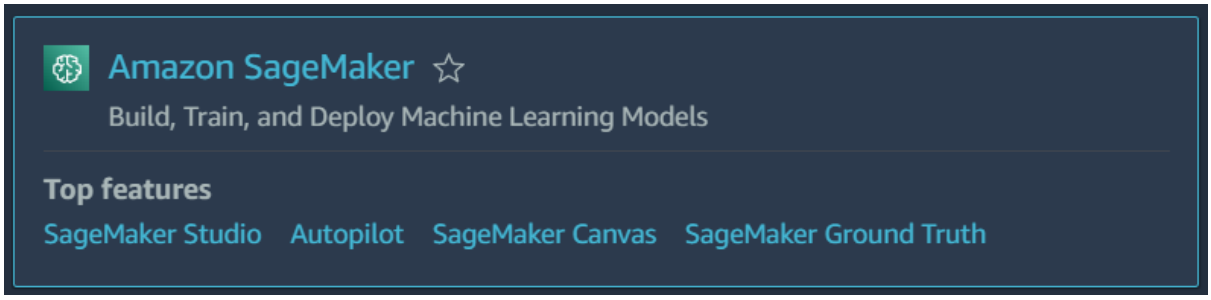
Conclusion:

Amazon SageMaker is a powerful service for managing machine learning workflows at scale, providing tools for data preparation, model training, tuning, deployment, and monitoring. It offers benefits like scalability, cost efficiency, and ease of integration with other AWS services. However, it requires some ML and cloud computing expertise to use effectively and can become costly if not managed well.

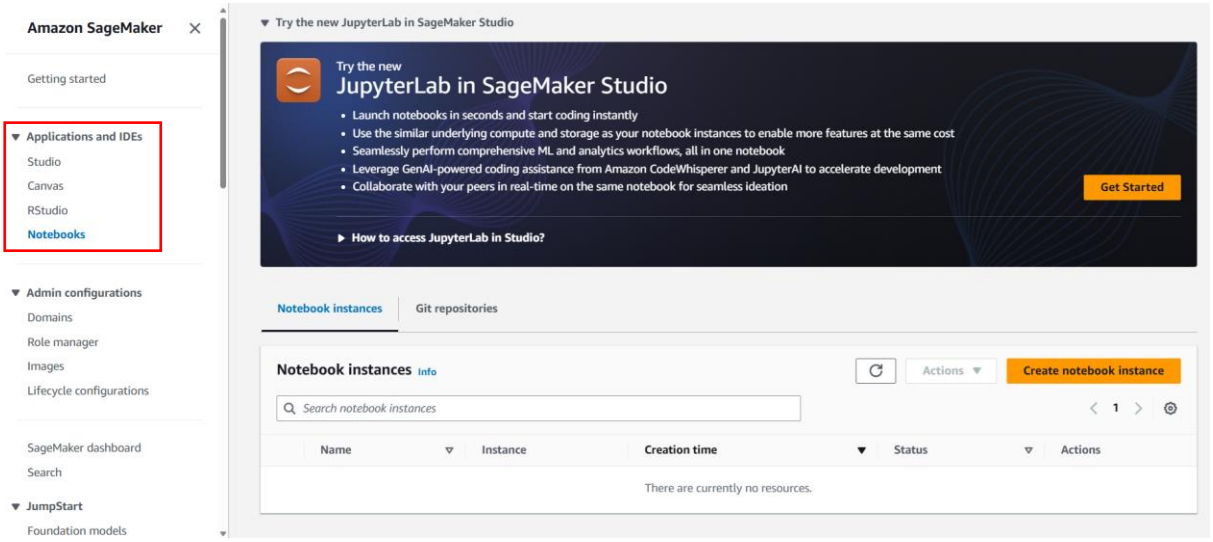
😊 To begin with the Lab:

1. In this lab, we are going to launch a SageMaker Notebook instance and then we are going to upload our notebook. We will push the datafile that we got into our S3 bucket and then we are going to consume the data from the S3 bucket.
2. We will prepare the training data that is required for us in order to perform the fir on our model and we will push the data back to our S3 bucket.

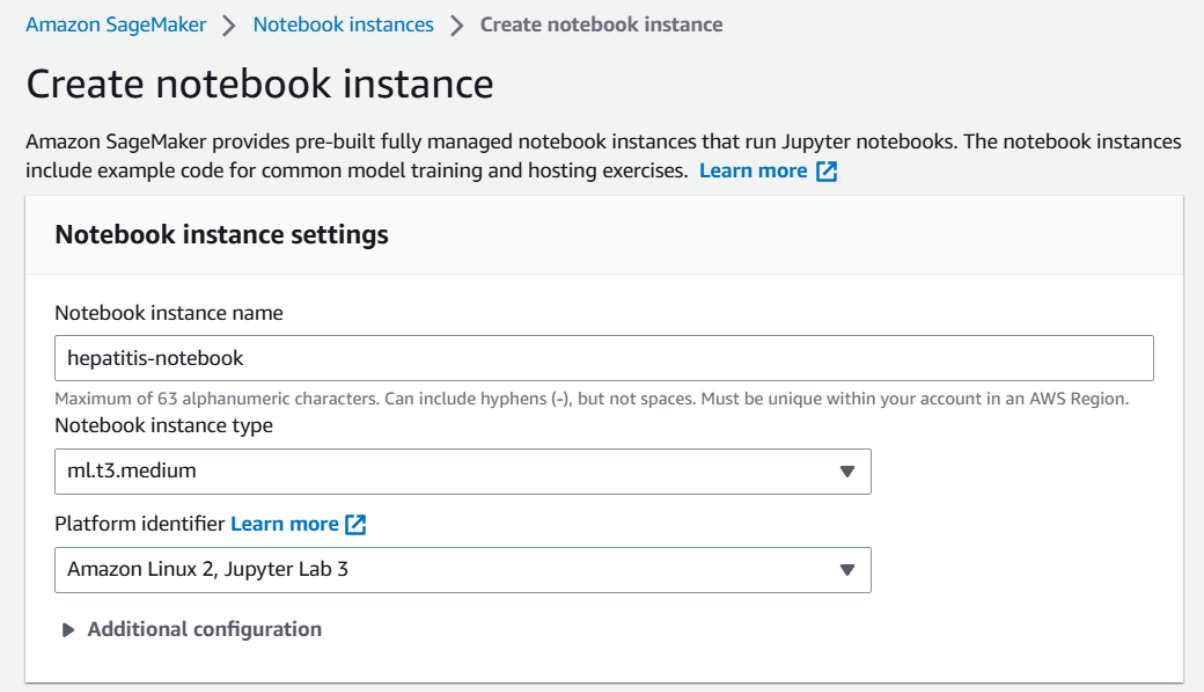
3. And once we perform the training, we will then push the model artifacts back into our S3 bucket.
4. So, we'll be using this S3 bucket as a source and as a place where we can store all those data files and we can also perform the versioning if you want to capture the lineage.
5. You will also have a folder with this lab which contains some files that we need to use for this lab.
6. Now go to your AWS Console and search for Amazon SageMaker and the service accordingly.



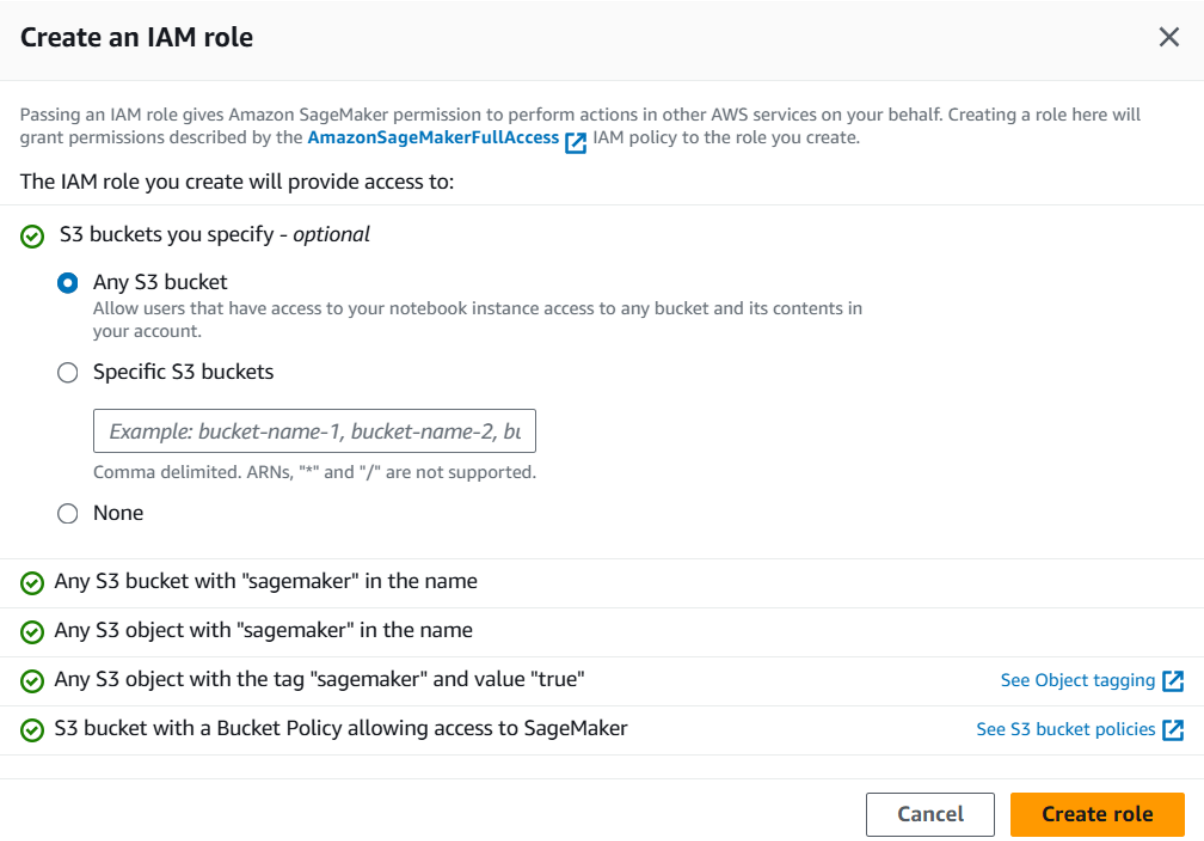
7. In SageMaker from the left pane expand Application and IDEs and choose Notebooks. Then click on create notebook instances.



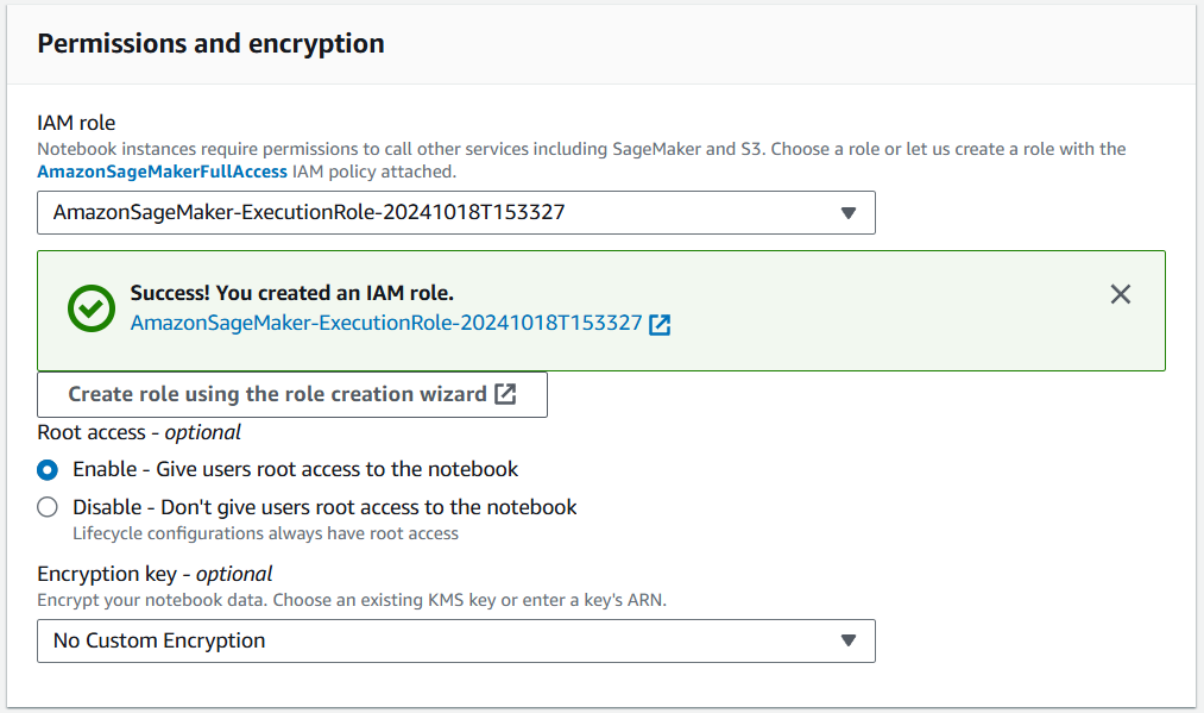
8. Now give your notebook a name and keep instance type and platform identifier to default.



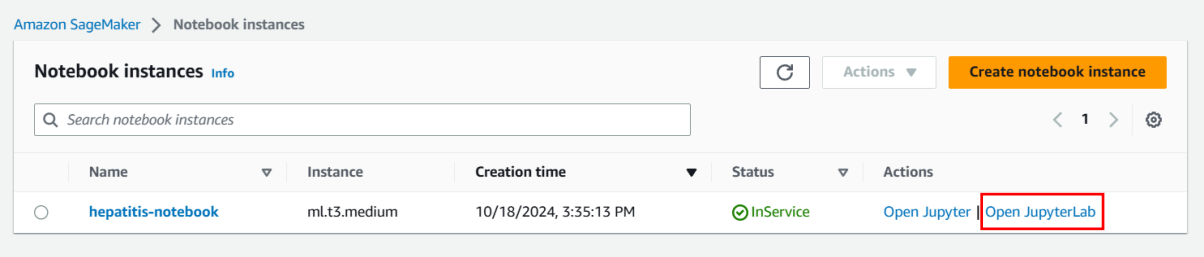
9. For Permission and Encryption section you need to click on create new IAM role and then choose the same settings as shown below and create you new role.



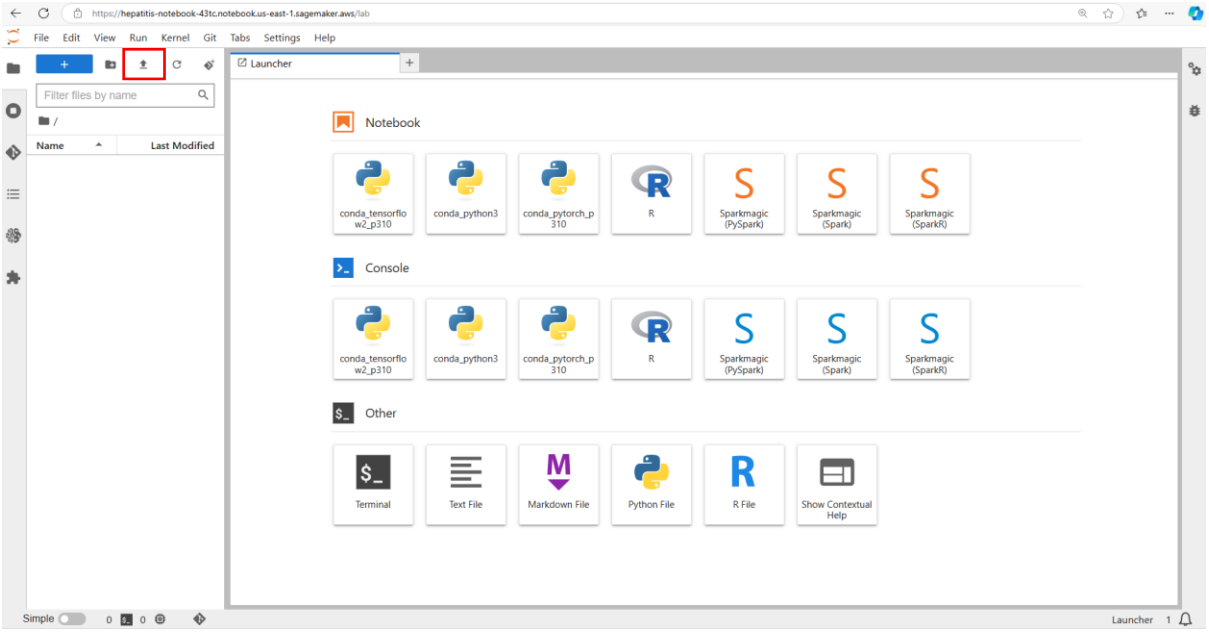
10. After that keep rest of the things to default and create your Notebook. It will take at least 5 minutes to be in ready state.



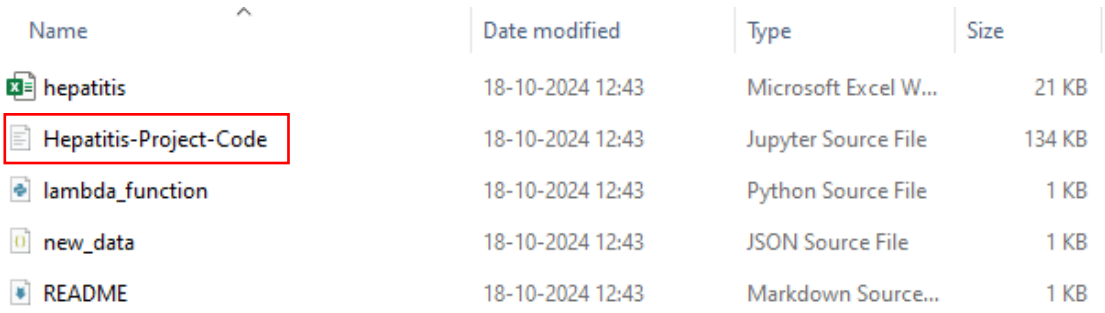
11. Below you can see that your notebook has been created. Now click on JupyterLab to open the Jupyter Notebook.



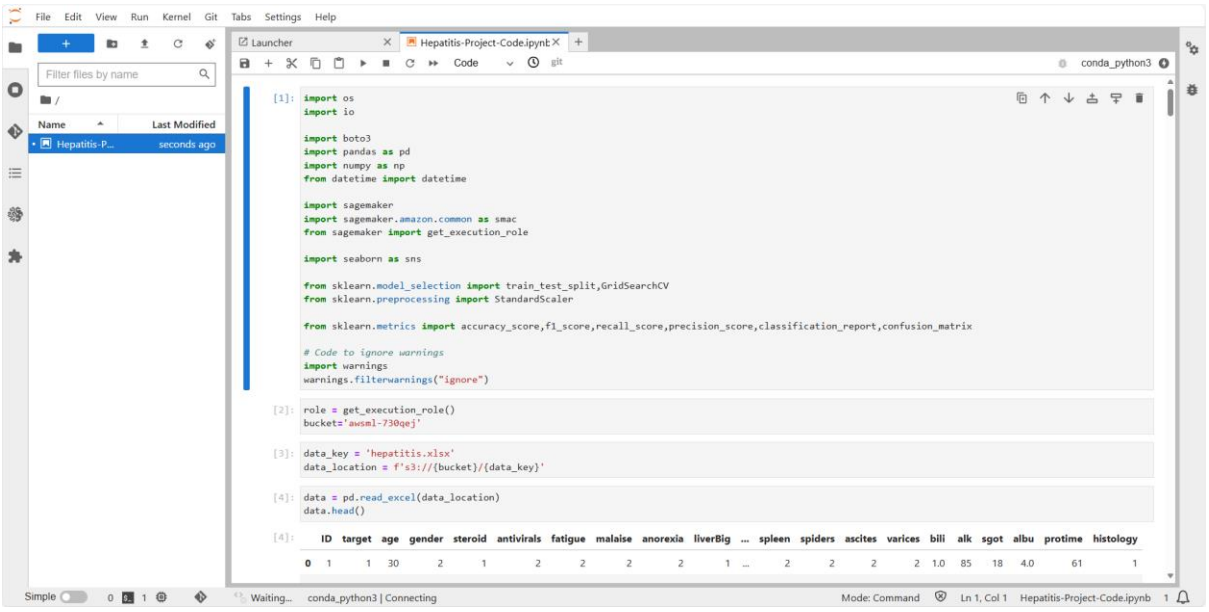
12. This is how the Notebook would look like. So, it is similar to Jupyter Notebook interface but if you look at the link you can say that this notebook is powered by AWS.
13. Now we are going to click on upload icon which is highlighted.



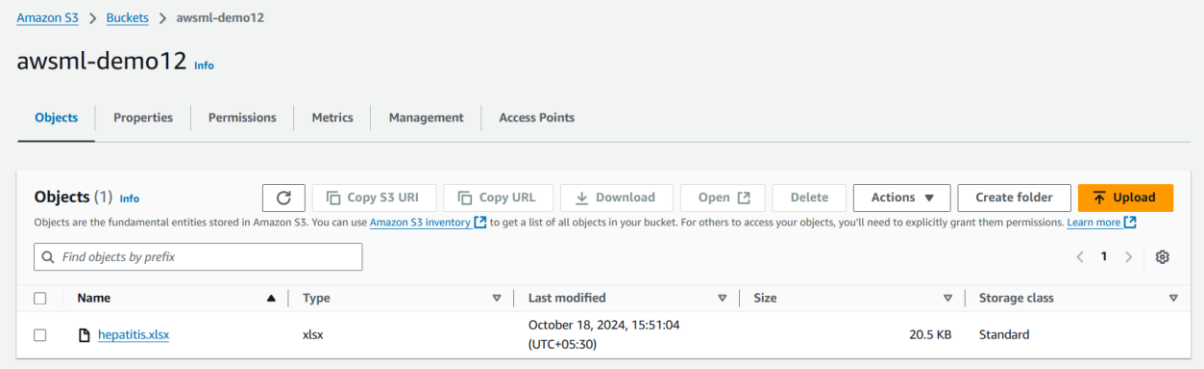
14. Then choose the Hepatitis Project Code and upload it to your notebook.



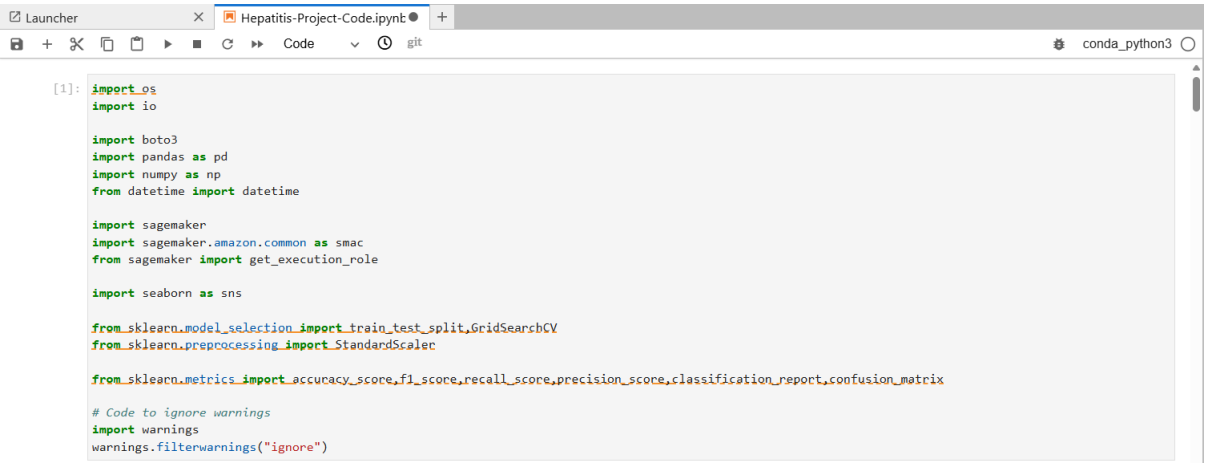
15. Below you can see that our project code have been uploaded and this is how our notebook would look like.



16. Next thing, you need to create an S3 bucket and upload the hepatitis.xlsx file to it.



17. Now we just need to execute our Notebook. First things first, import the necessary libraries.



18. Then we specified our bucket name and get the execution role attached with our SageMaker. After that we accessed the file from our S3 bucket.

```
[2]: role = get_execution_role()
     bucket='awsml-demo12' #specify the bucket name

[3]: data_key = 'hepatitis.xlsx'
     data_location = f's3://awsml-demo12/hepatitis.xlsx' #S3 object URI
```

19. Here we read few entries from our dataset and check its shape for rows and columns.

```
[4]: data = pd.read_excel(data_location)
     data.head()
```

[4]:

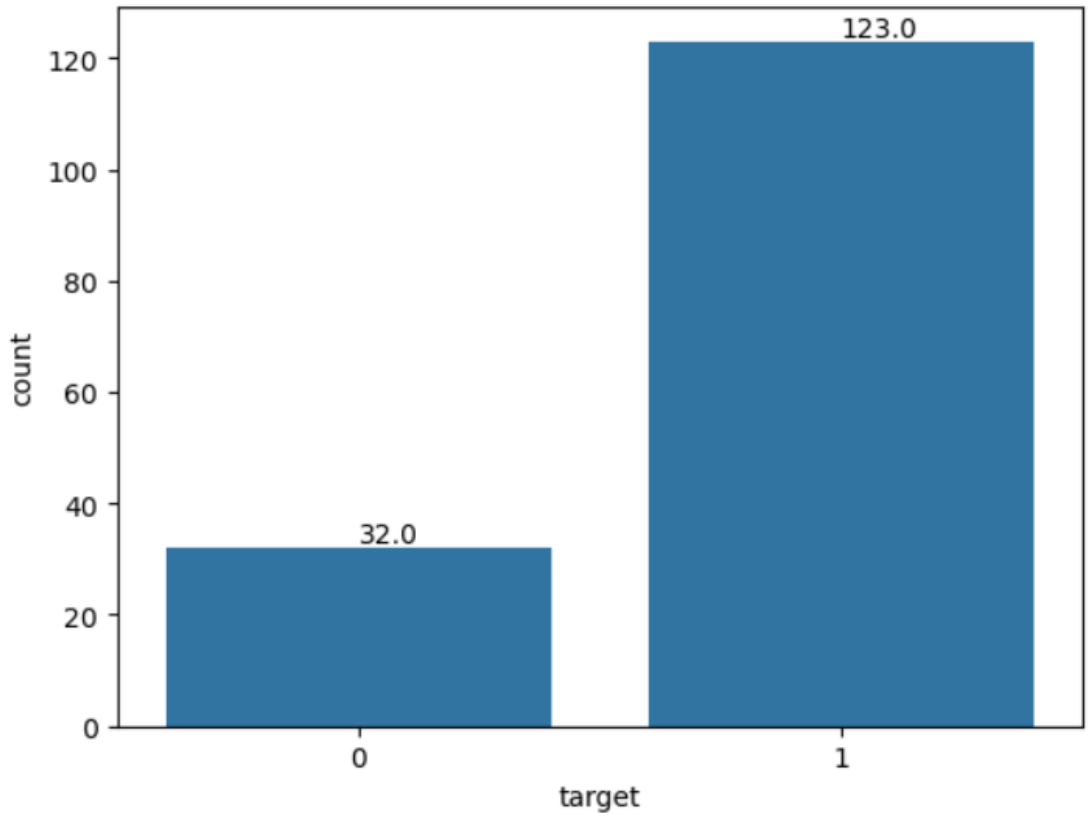
	ID	target	age	gender	steroid	antivirals	fatigue	malaise	anorexia	liverBig	...	spleen	spiders	ascites	varices	bili	alk	sgot	albu	protime	histology
0	1	1	30	2	1	2	2	2	2	1	...	2	2	2	2	1.0	85	18	4.0	61	1
1	2	1	50	1	1	2	1	2	2	1	...	2	2	2	2	0.9	135	42	3.5	61	1
2	3	1	78	1	2	2	1	2	2	2	...	2	2	2	2	0.7	96	32	4.0	61	1
3	4	1	31	1	1	1	2	2	2	2	...	2	2	2	2	0.7	46	52	4.0	80	1
4	5	1	34	1	2	2	2	2	2	2	...	2	2	2	2	1.0	46	200	4.0	61	1

5 rows × 21 columns

```
[5]: print('Dataset has ' + str(data.shape[0]) + ' rows, and ' + str(data.shape[1]) + ' columns')
Dataset has 155 rows, and 21 columns
```

20. We also have the visualization of our data.

[7]:
```
y_count=sns.countplot(x='target',data=data)
# Shows the count of observations in each categorical bin using bars

for p in y_count.patches:
    height = p.get_height()
    # Add text to the axes
    y_count.text(p.get_x()+p.get_width()/2, height + 1, height)
# The y_count.text method takes an x position, a y position and a string
```



21. Afte that we dropped the ID column and we set the categorical column and displayed the data types.

```
[8]: data.drop(["ID"], axis = 1, inplace=True)
```

```
[9]: num_cols = ["age", "bili", "alk", "sgot", "protime"]
     cat_cols = ['gender', 'steroid', 'antivirals', 'fatigue', 'malaise', 'anorexia', 'liverBig',
                 'liverFirm', 'spleen', 'spiders', 'ascites', 'varices', 'histology']
```

```
[10]: data[cat_cols] = data[cat_cols].astype('category')
```

```
[11]: data.dtypes
```

```
[11]: target           int64
      age              int64
      gender        category
      steroid       category
      antivirals    category
      fatigue       category
      malaise       category
      anorexia      category
      liverBig      category
      liverFirm     category
      spleen        category
      spiders       category
      ascites       category
      varices       category
      bili           float64
      alk              int64
      sgot             int64
      albu           float64
      protime          int64
      histology     category
      dtype: object
```

22. We performed the one hot encoding and displayed the few entries.

```
[13]: # OneHotEncoding
      data = pd.get_dummies(data, columns=['gender', 'steroid', 'antivirals', 'fatigue', 'malaise', 'anorexia', 'liverBig',
                  'liverFirm', 'spleen', 'spiders', 'ascites', 'varices', 'histology'],drop_first=True)
```

```
[14]: display(data.head())
      display(data.shape)
```

	target	age	bili	alk	sgot	albu	protime	gender_2	steroid_2	antivirals_2	fatigue_2	malaise_2	anorexia_2	liverBig_2	liverFirm_2	spleen_2	spiders_2	ascites
0	1	30	1.0	85	18	4.0	61	True	False	True	True	True	True	False	True	True	True	Tr
1	1	50	0.9	135	42	3.5	61	False	False	True	False	True	True	False	True	True	True	Tr
2	1	78	0.7	96	32	4.0	61	False	True	True	False	True	True	True	True	True	True	Tr
3	1	31	0.7	46	52	4.0	80	False	False	False	True	True	True	True	True	True	True	Tr
4	1	34	1.0	46	200	4.0	61	False	True	True	True	True	True	True	True	True	True	Tr

```
(155, 20)
```

23. After that we split the dataset into two parts. So, before we use this data we need to perform a kind of formatting, so for that we are going to generate this training data and we are going to place it in our S3 bucket.
24. In this case, I'll be converting the data into a protobuf IO format. So this is one of the efficient way to process the dataset for training the sagemaker models.

```
[18]: X_train.shape
```

```
[18]: (127, 19)
```

```
[19]: X_test.shape
```

```
[19]: (28, 19)
```

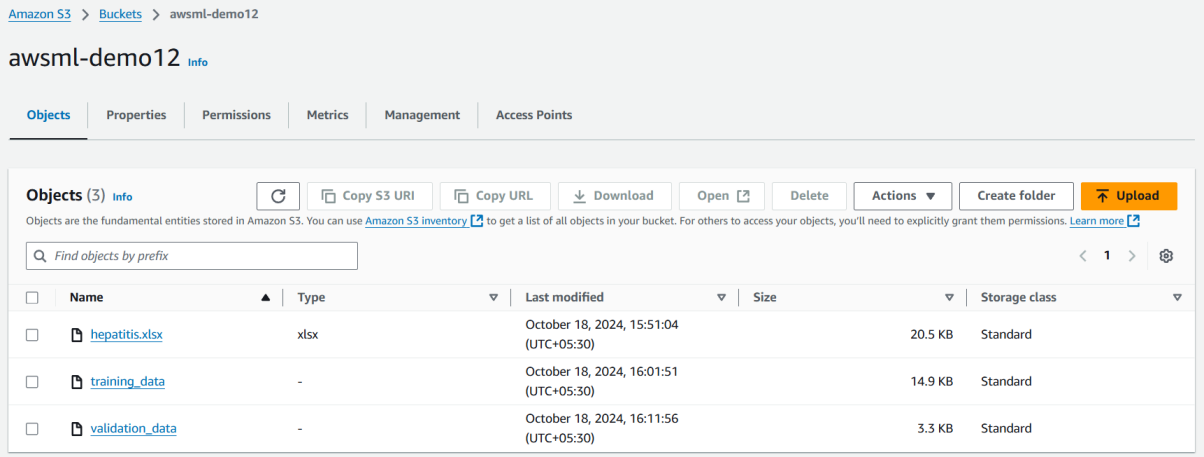
```
[20]: train_file = 'training_data'
f = io.BytesIO()
smac.write_numpy_to_dense_tensor(f, X_train.astype('float32'), y_train.astype('float32'))
f.seek(0)

boto3.Session().resource('s3').Bucket(bucket).Object(f'{train_file}').upload_fileobj(f)
training_recordIO_protobuf_location = f's3://{bucket}/{train_file}'
print('The Pipe mode recordIO protobuf training data: {}'.format(training_recordIO_protobuf_location))
```

```
The Pipe mode recordIO protobuf training data: s3://awsml-demo12/training_data
```

```
[21]: validation_file = 'validation_data'

f = io.BytesIO()
smac.write_numpy_to_dense_tensor(f, X_test.astype('float32'), y_test.astype('float32'))
f.seek(0)

boto3.Session().resource('s3').Bucket(bucket).Object(f'{validation_file}').upload_fileobj(f)
validate_recordIO_protobuf_location = f's3://{bucket}/{validation_file}'
print('The Pipe mode recordIO protobuf validation data: {}'.format(validate_recordIO_protobuf_location))
```

```
The Pipe mode recordIO protobuf validation data: s3://awsml-demo12/validation_data
```

25. After running the command if you go to your S3 bucket you will see that a training data and validation data files has been created.



26. Then just run all the commands until you see the endpoint name.

27. The code is setting up a machine learning training job using the K-Nearest Neighbors (KNN) algorithm on AWS SageMaker. It starts by identifying the necessary container for the KNN algorithm. Then, it creates a unique name for the training job based on the current date and time.
28. Next, it specifies where the trained model will be stored in an S3 bucket. The code also defines the number of features in the dataset and sets the hyperparameters for the KNN model, like the number of neighbors to consider.
29. A SageMaker session is initiated, and an estimator object is created to manage the training process. The training job is then launched with the specified data for training

and validation. Finally, once the model is trained, it is deployed to an endpoint for making predictions.

[22]:
```
from sagemaker import image_uris, model_uris, script_uris
```

[23]:
```
container = image_uris.retrieve('knn', boto3.Session().region_name, 'latest')
```

```
Defaulting to the only supported framework/algorithm version: 1. Ignoring framework/algorithm version: latest.
```

[24]:
```
# Create a training job name
job_name = 'hepatitis-job3-{}'.format(datetime.now().strftime("%Y%m%d%H%M%S"))
print('Here is the job name {}'.format(job_name))

# Here is where the model-artifact will be stored
output_location = f's3://{bucket}/model-artifact/'
```

```
Here is the job name hepatitis-job3-20241018120509
```

[25]:
```
print('The feature_dim hyperparameter needs to be set to {}.'.format(X_train.shape[1]))
```

```
The feature_dim hyperparameter needs to be set to 19.
```

[26]:
```
X_train.shape
```

[26]:
```
(127, 19)
```

[27]:
```
hyperparams = {"feature_dim": 19, "k": 10, "sample_size": 20, "predictor_type": "classifier"}
```

[28]:
```
sess = sagemaker.Session()

# Setup the KNNLeaner algorithm from the ECR container
knn = sagemaker.estimator.Estimator(container,
                                    role,
                                    instance_count=1,
                                    instance_type='ml.m4.xlarge',
                                    output_path=output_location,
                                    sagemaker_session=sess,
                                    input_mode='Pipe')
```

[29]:
```
# Setup the hyperparameters
knn.set_hyperparameters(feature_dim=19,
                        k=3,
                        sample_size=20,
                        predictor_type='classifier')
```

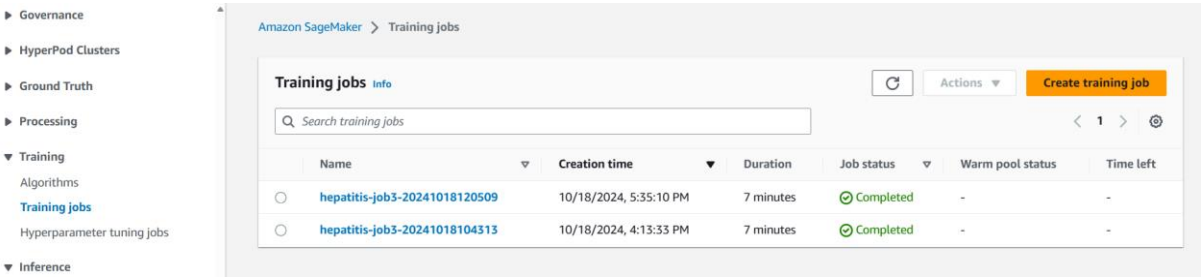
[30]:
```
# Launch a training job. This method calls the CreateTrainingJob API call
data_channels = {'train': training_recordIO_protobuf_location,
                 'validation': validate_recordIO_protobuf_location}
```

[31]:
```
knn.fit(data_channels, job_name=job_name)
```

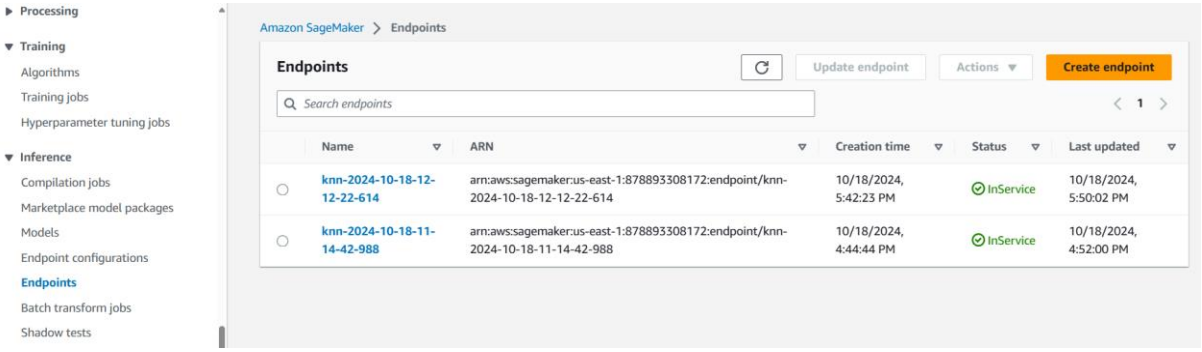
```
INFO:sagemaker:Creating training-job with name: hepatitis-job3-20241018120509
2024-10-18 12:05:12 Starting - Starting the training job...
2024-10-18 12:05:26 Starting - Preparing the instances for training...
2024-10-18 12:06:00 Downloading - Downloading input data...
2024-10-18 12:06:25 Downloading - Downloading the training image.....................
2024-10-18 12:11:03 Training - Training image download completed. Training in progress....
2024-10-18 12:11:46 Uploading - Uploading generated training model
2024-10-18 12:11:46 Completed - Training job completed
```

```
[32]: print('Here is the location of the trained KNN model: {}/{}/output/model.tar.gz'.format(output_location, job_name))

Here is the location of the trained KNN model: s3://awsml-demo12/model-artifact//hepatitis-job3-20241018120509/output/model.tar.gz

[33]: hepatitis_predictor = knn.deploy(initial_instance_count=1, instance_type='ml.m4.xlarge')

INFO:sagemaker:Creating model with name: knn-2024-10-18-12-12-22-614
INFO:sagemaker:Creating endpoint-config with name knn-2024-10-18-12-12-22-614
INFO:sagemaker:Creating endpoint with name knn-2024-10-18-12-12-22-614
---------------!

[34]: hepatitis_predictor.endpoint_name

[34]: 'knn-2024-10-18-12-12-22-614'
```

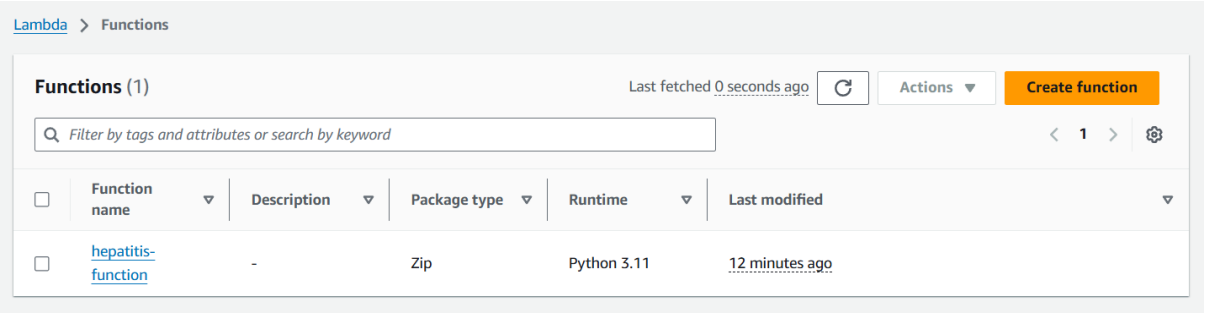
30. So, when we run the command on 31st cell we did the KNN fit and by doing that we create a Training job that took 7 minutes to completed. You can see that job in your SageMaker by expanding the training section and going to training jobs.



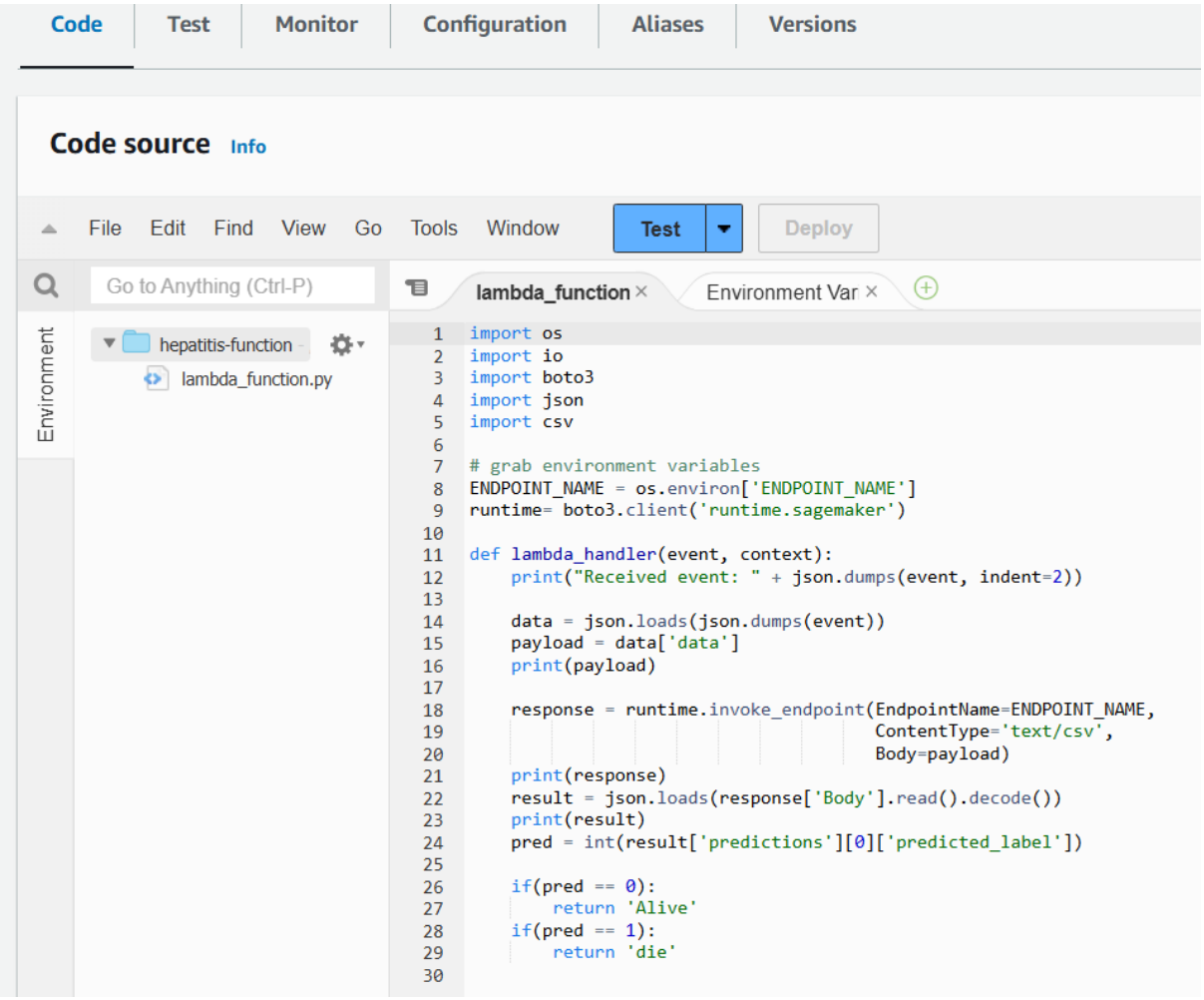
31. When we run the command on 33rd cell we created an endpoint which took almost 15 minutes and make a note of this endpoint.



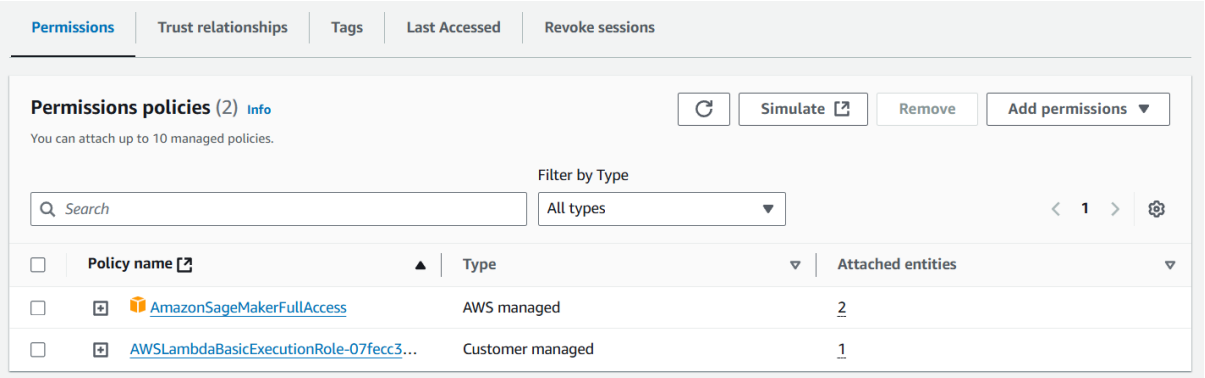
32. Then in you AWS Console go to Lambda and create a new function using Python 3.11 as your environment.



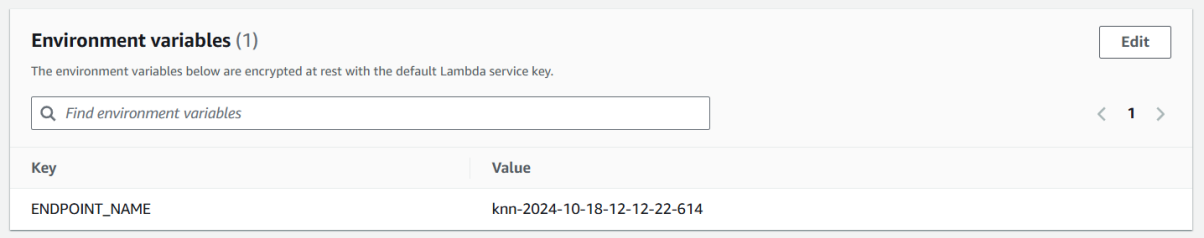
33. Once your function is created then you need to change the code of it. For that you can open the folder you get with this lab and open the lambda function file. Copy and paste the code in your function and click on deploy.



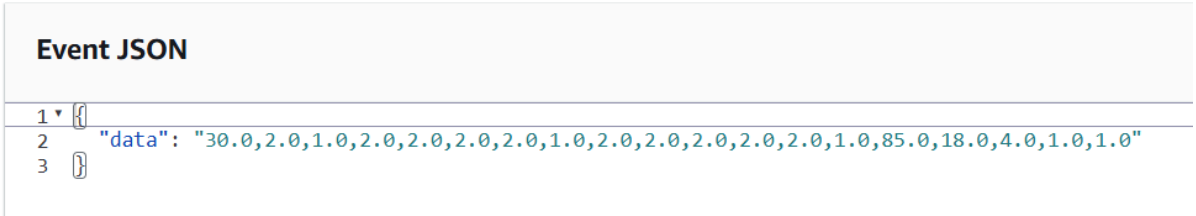
34. Now go to configurations then to Permission open the IAM role attached to your Lambda function then add sage maker full access policy to your role and come back to your lambda function.



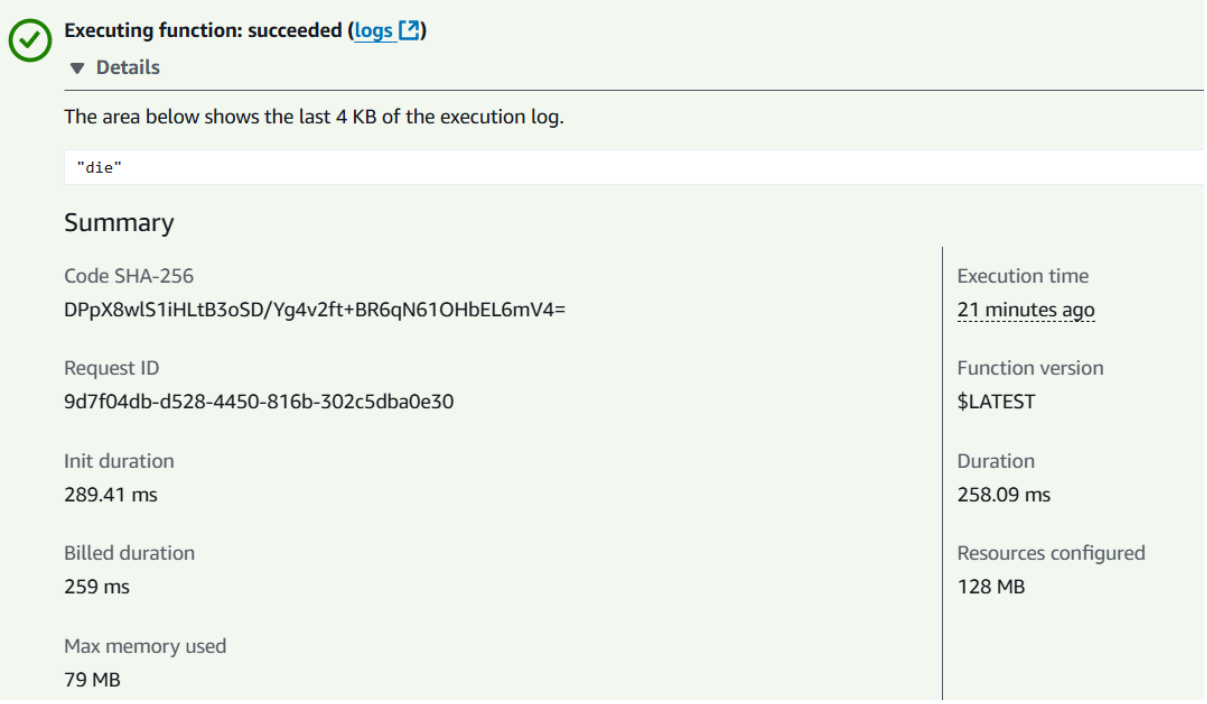
35. After that open the Environment variable tab and add a variable for the endpoint we got in SageMaker as you can see below. We have defined the key in our code and our code will get the value from the environment variable.



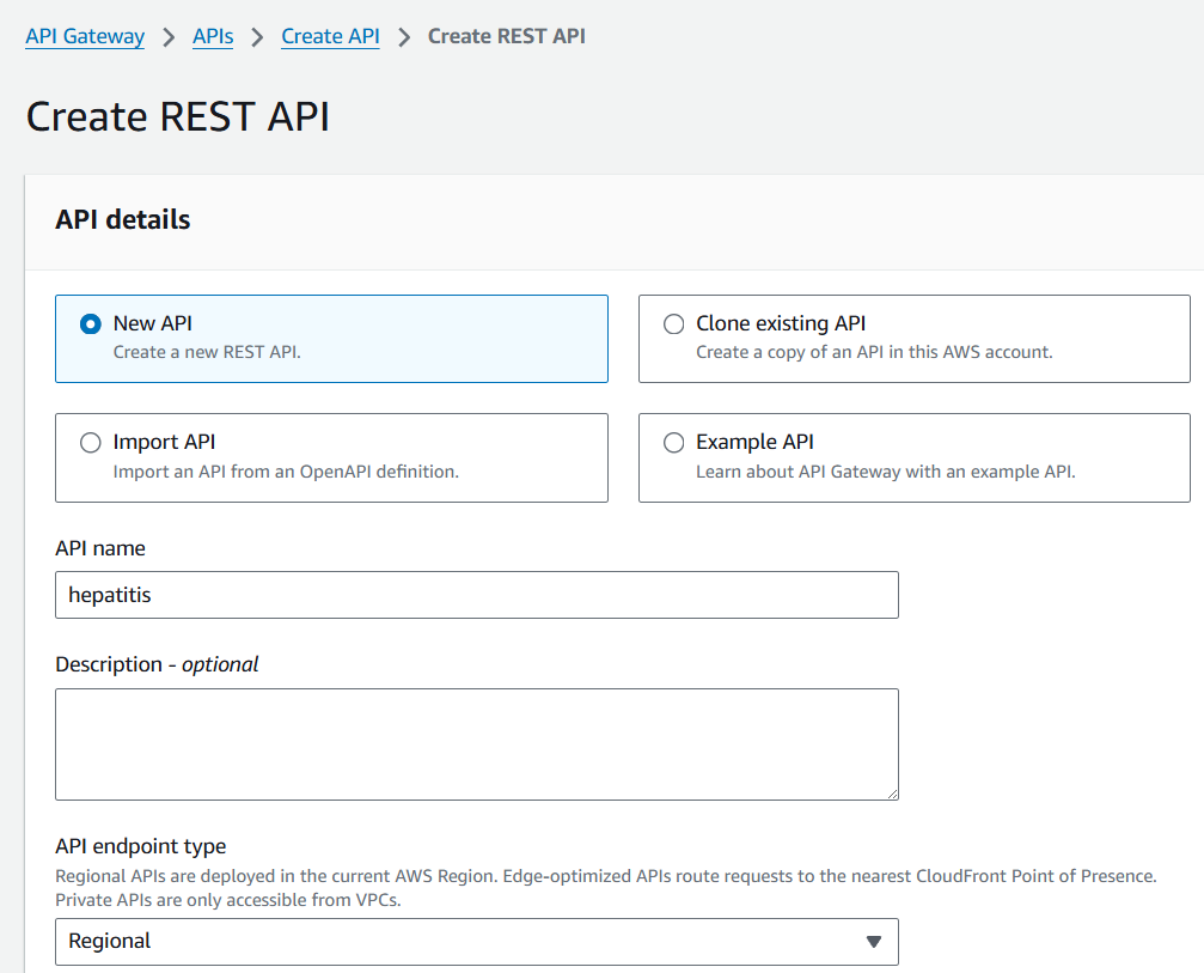
36. Once this is all done come to Test tab and we are going to test our function. Now we need click on create new event for testing and then in the JOSN area we need to paste this script. You can find this script in the file as well.



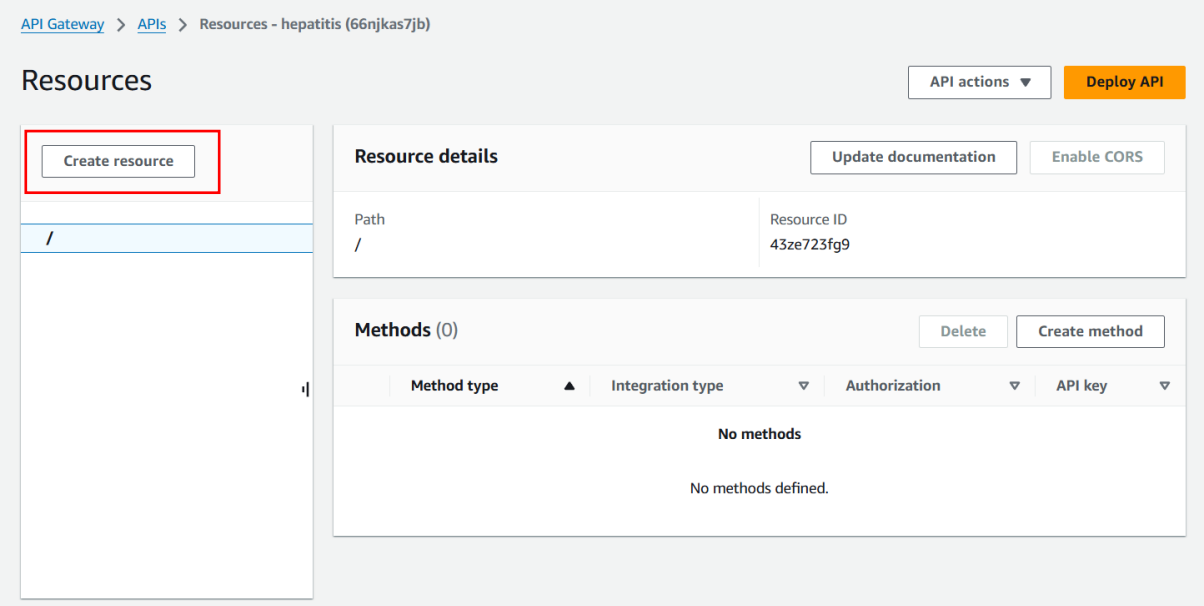
37. So, after running the test you can see that our test got executed and we got the output.



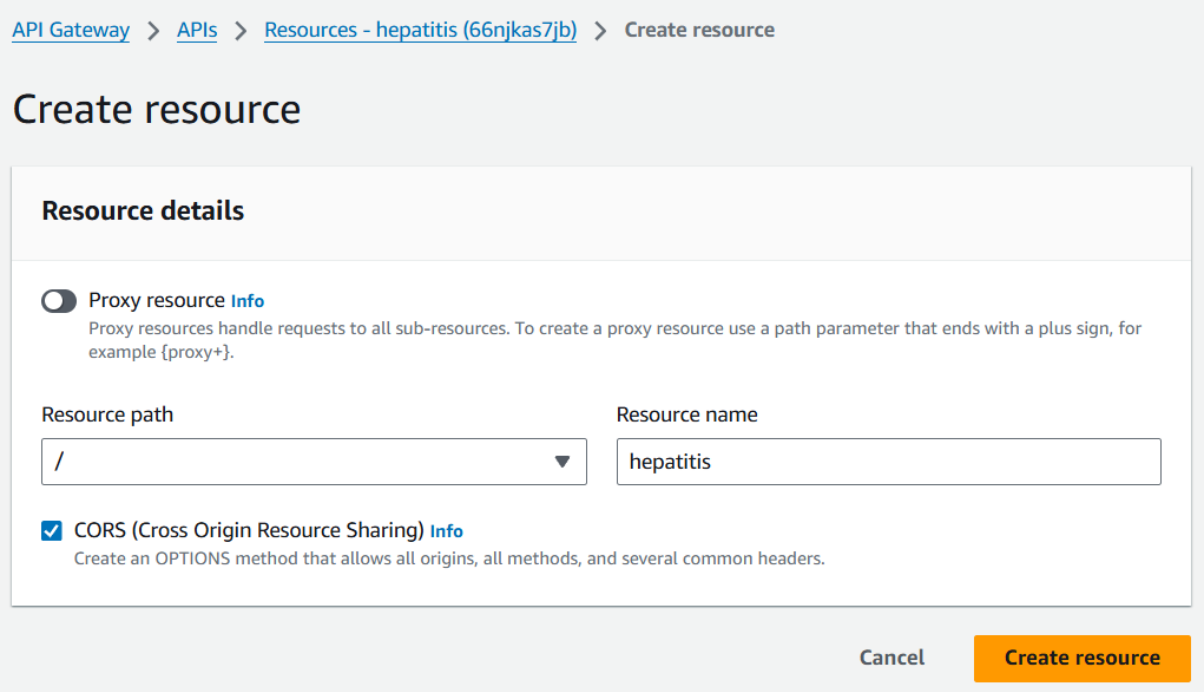
38. After that search and navigate to API gateway in your Console and click on build REST API. Just give it a name and then click on create.



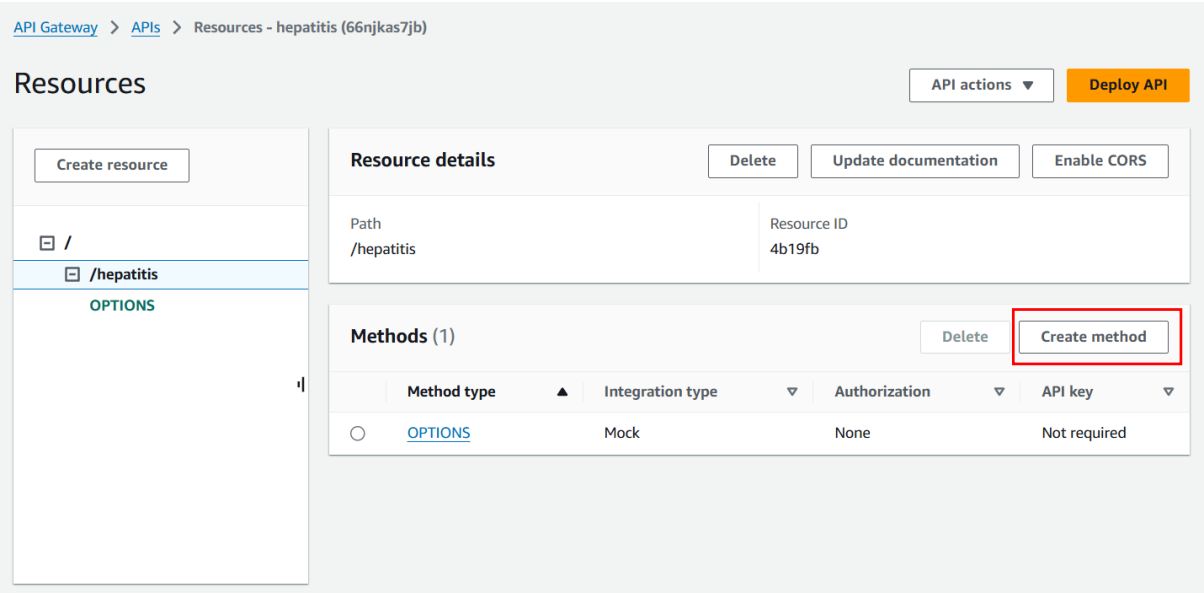
39. Here you need to create a resource, click on create resource.



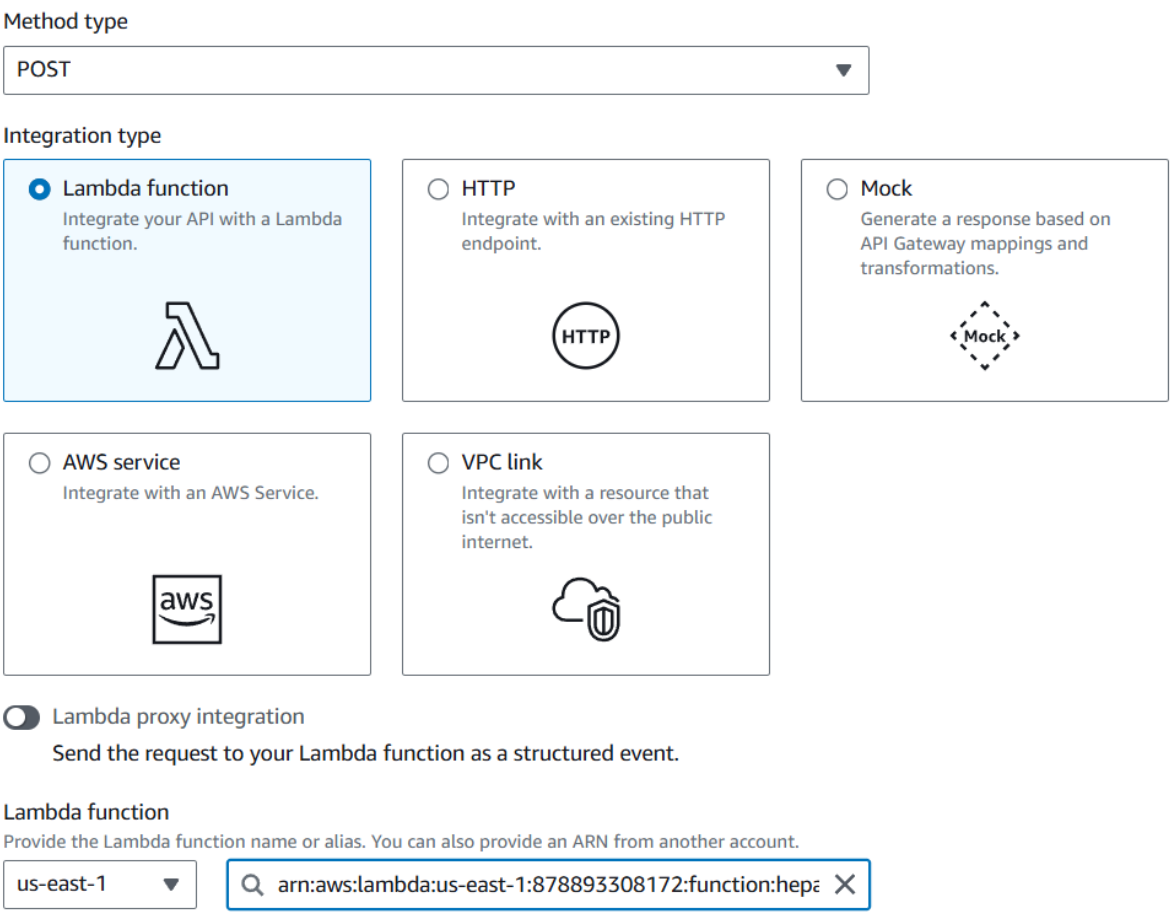
40. Then you need to give a resource name and enable CORS, click on create.



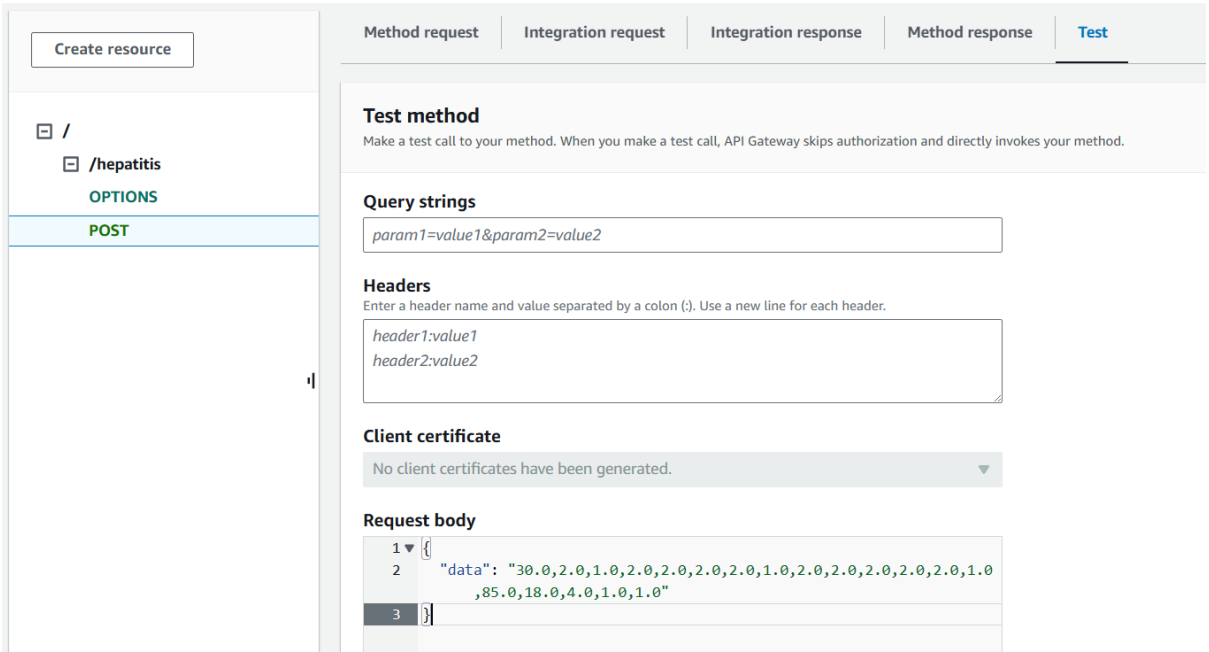
41. Now choose you hepatitis resource and click on create method.



42. You need to choose POST as your method type and scroll down then choose your Lambda function.



43. Then choose your POST method and click on Test tab. In the request body paste the same JOSN script you used in your Lambda Function.



44. Below you can see that you got the response.

/hepatitis - POST method test results

Request	Latency ms	Status
/hepatitis	908	200

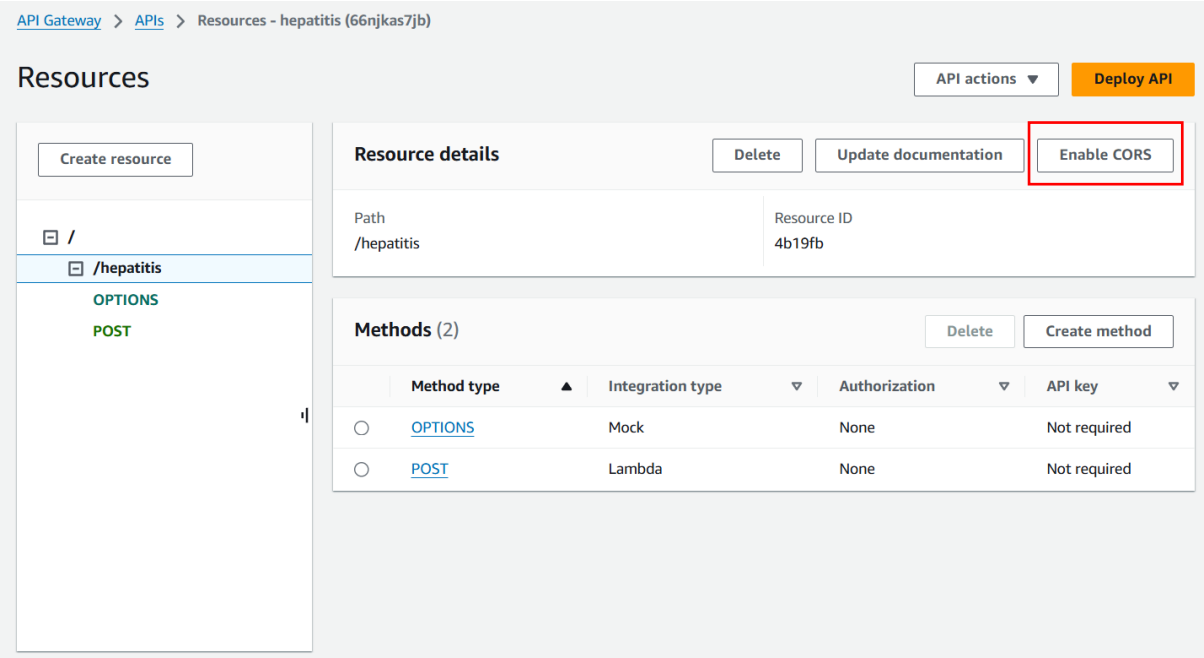
Response body

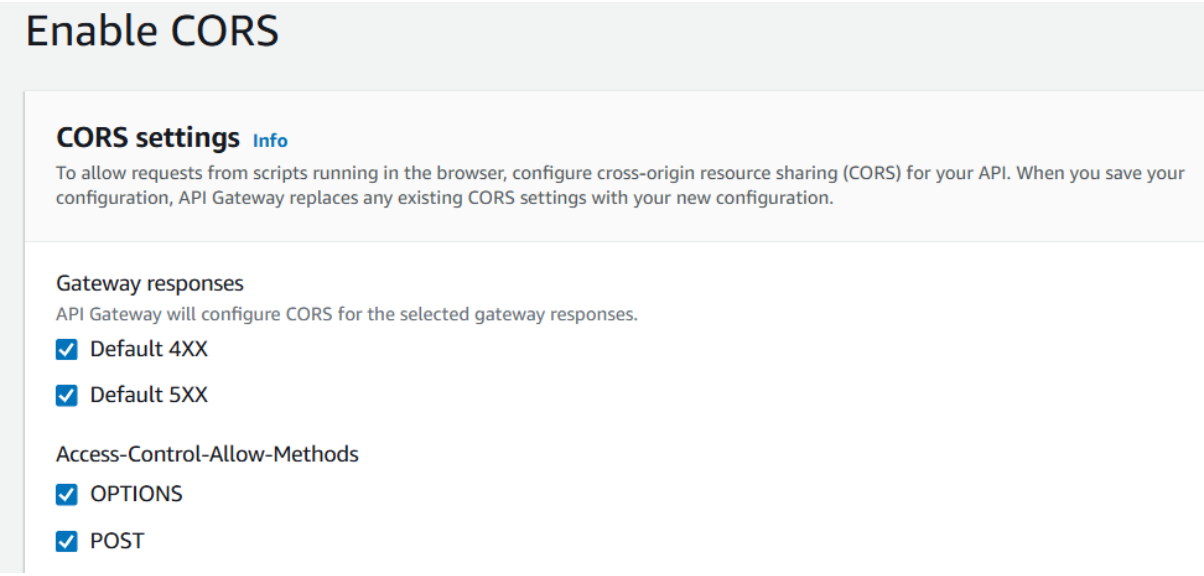
```
"die"
```

Response headers

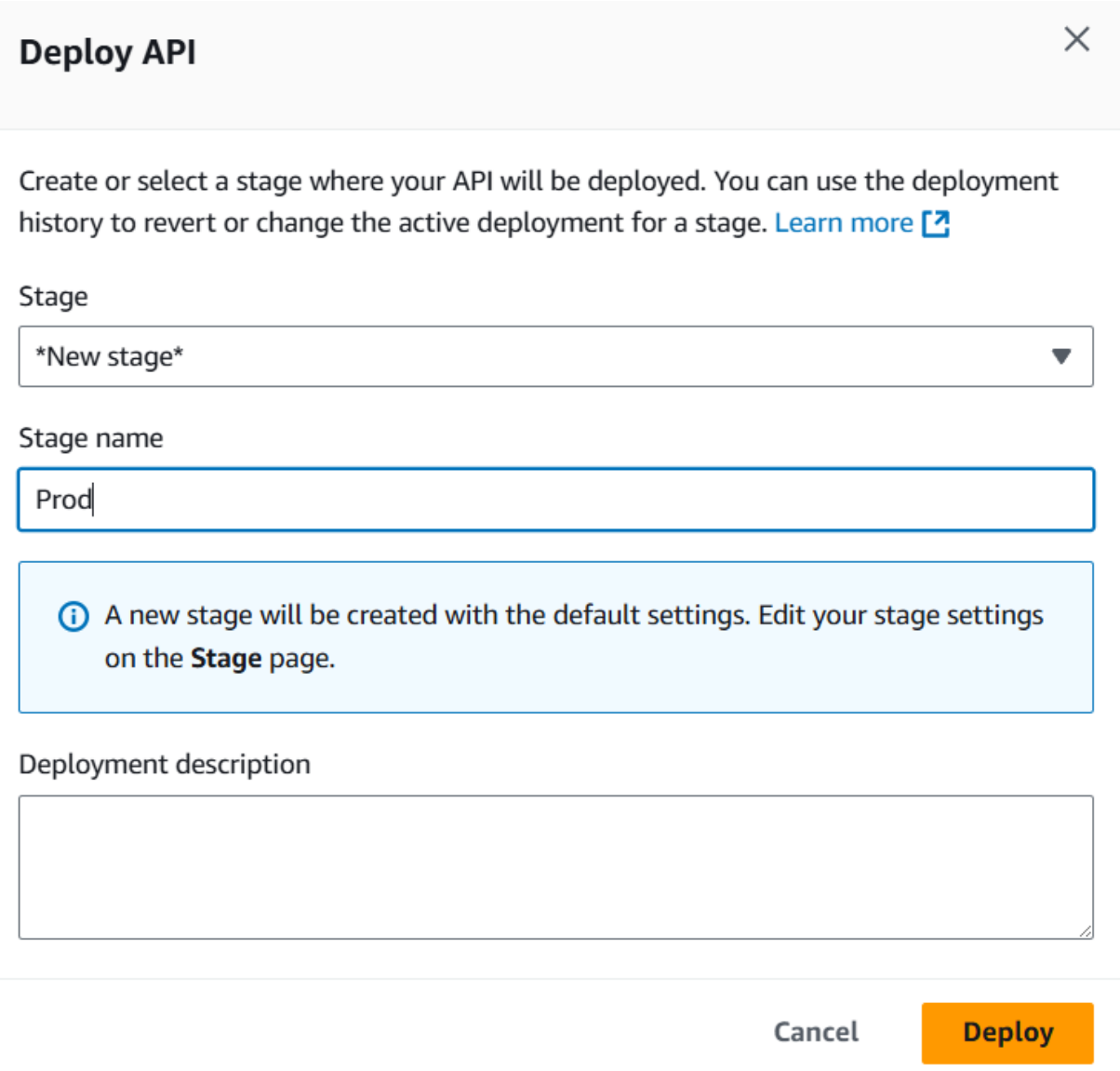
```
{
  "Content-Type": "application/json",
  "X-Amzn-Trace-Id": "Root=1-67125dd7-
5dc09b320ecb26d675c5ddf6;Parent=177f0a0fe65e1633;Sampled=0;Lineage=1:5f245cdc:0"
}
```

45. Now click on your hepatitis resource and click on Enable CORS. Here you need to enable the same things as you can see in the snapshot.

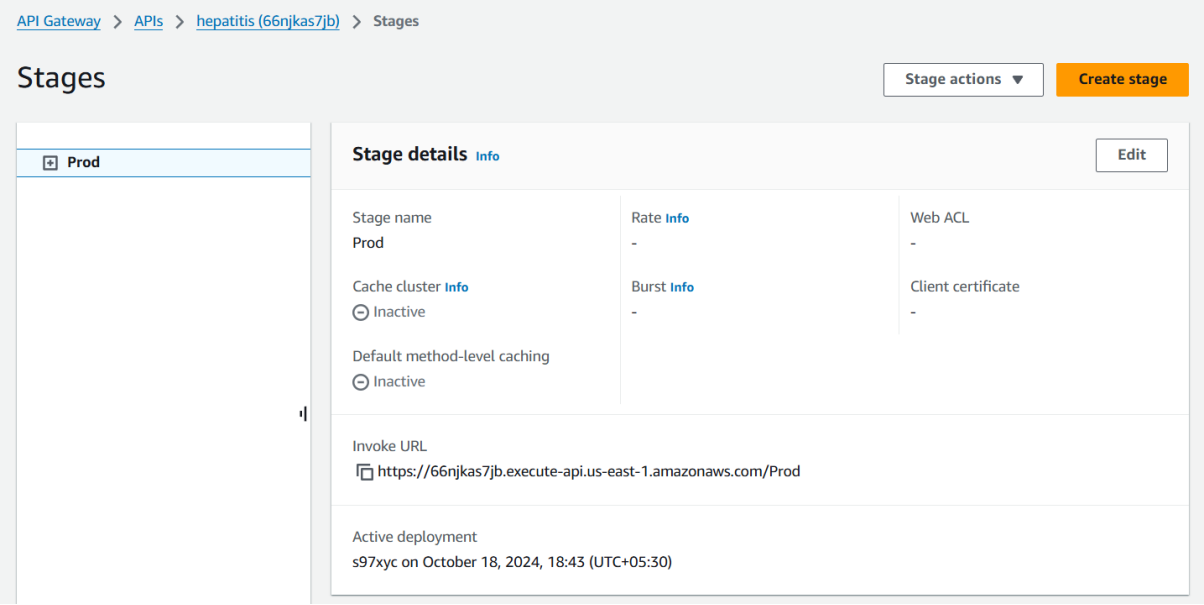




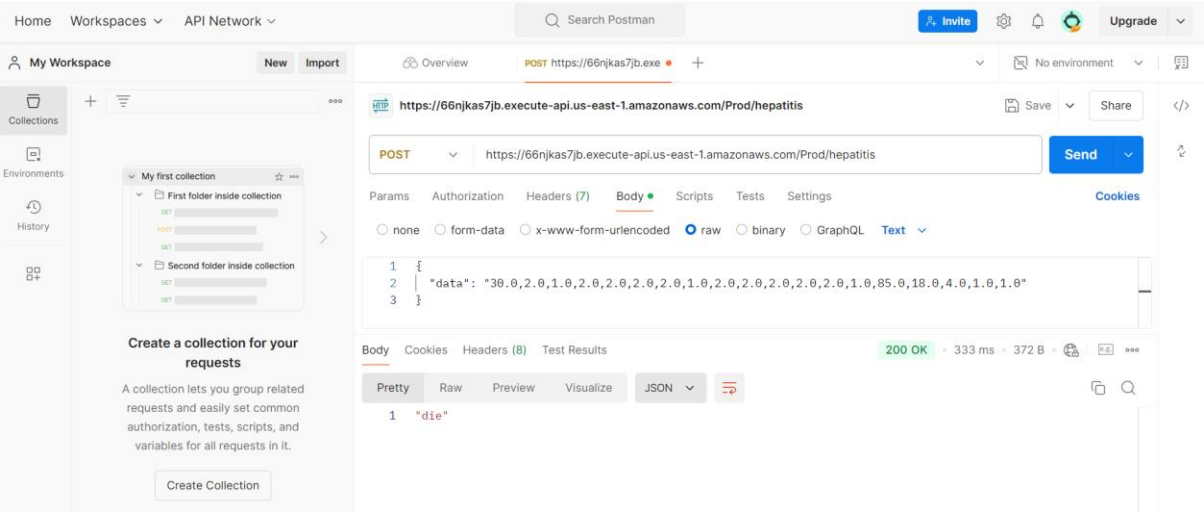
46. Then click on Deploy API, choose new stage give it a name and click on Deploy.



47. Here you can see that we got the stage ready. Now copy the Invoke URL.



48. Now open the PostMan in your Laptop. Create a new workspace in it.
49. Then choose POST as your method and paste the URL here then append it with the hepatitis name which is your resource name.
50. Then copy the JSON script and paste it in the body section. Click on Send.
51. Below you can see that you have the response.



52. Once you are done delete your Endpoints from SageMaker then stop your Notebook and delete it.
53. After that delete your lambda function and your API Gateway.