Ace the Exam Series®

2022

FIRST EDITION

HANDS-ON LABS

MLS-C01: AWS CERTIFIED MACHINE LEARNING SPECIALTY

BASED ON

REAL WORLD CASE STUDIES

IPSpecialist
Sharpen Your Skills for the Digital Future

Hands-On Labs

MLS-CO1: AWS Certified Machine Learning Specialty

Based on Real-World Case Studies

Lab Guide

www.ipspecialist.net

Document Control

Proposal Name	:	AWS Certified Machine Leaning Specialty – Hands-on Labs
Document Edition	:	First Edition
Document Release Date	:	16th August 2022
Reference	:	MLS-C01

Registered in England and Wales
Company Registration No: 10883539
Registration Office at: Office 32, 19-21 Crawford Street, London W1H 1PJ, United Kingdom
www.ipspecialist.net

Feedback:
If you have any comments regarding the quality of this book, or otherwise alter it to better suit your needs, you can contact us through email at info@ipspecialist.net
Please make sure to include the book's title and ISBN in your message.

About IPSpecialist

IPSPECIALIST LTD. IS COMMITTED TO EXCELLENCE AND DEDICATED TO YOUR SUCCESS.

Our philosophy is to treat our customers like family. We want you to succeed, and we are willing to do everything possible to help you make it happen. We have the proof to back up our claims. We strive to accelerate billions of careers with great courses, accessibility, and affordability. We believe that continuous learning and knowledge evolution are the most important things to keep re-skilling and up-skilling the world.

Planning and creating a specific goal are where IPSpecialist helps. We can create a career track that suits your visions as well as develop the competencies you need to become a professional Network Engineer. We can also assist you with the execution and evaluation of your proficiency level, based on the career track you choose, as they are customized to fit your specific goals.

We help you STAND OUT from the crowd through our detailed IP training content packages.

Course Features:

- Self-Paced Learning
  - Learn at your own pace and in your own time
- Covers Complete Exam Blueprint
  - Prep-up for the exam with confidence
- Case Study Based Learning
  - Relate the content with real-life scenarios
- Subscriptions that Suit You
  - Get more and pay less with IPS subscriptions
- Career Advisory Services
  - Let the industry experts plan your career journey
- Virtual Labs to test your skills
  - With IPS vRacks, you can evaluate your exam preparations
- Practice Questions
  - Practice questions to measure your preparation standards
- On Request Digital Certification
  - On request digital certification from IPSpecialist LTD.

About the Authors:

This book has been compiled with the help of multiple professional engineers. These engineers specialize in different fields. For example, Networking, Security, Cloud, Big Data, IoT, etc. Each engineer develops content in its specialized field that is compiled to form a comprehensive certification guide.

About the Technical Reviewers:

Nouman Ahmed Khan

AWS-Architect, CCDE, CCIEX5 (R&S, SP, Security, DC, Wireless), CISSP, CISA, CISM is a Solution Architect working with a major telecommunication provider in Qatar. He works with enterprises, mega-projects, and service providers, helping them select the best-fit technology solutions. He also works closely as a consultant to understand customer business processes and helps select an appropriate technology strategy to support business goals. He has more than 14 years of experience working in Pakistan/Middle-East & UK. He holds a Bachelor of Engineering Degree from NED University, Pakistan, and an M.Sc. in Computer Networks from the UK.

Abubakar Saeed

Abubakar Saeed has more than twenty-five years of experience, Managing, Consulting, Designing, and implementing large-scale technology projects, extensive experience heading ISP operations, solutions integration, heading Product Development, Presales, and Solution Design. Emphasizing adhering to Project timelines and delivering as per customer expectations, he always leads the project in the right direction with his innovative ideas and excellent management.

Dr. Fahad Abdali

Dr. Fahad Abdali is a seasoned leader with extensive experience managing and growing software development teams in high-growth start-ups. He is a business entrepreneur with more than 18 years of experience in management and marketing. He holds a Bachelor's Degree from NED University of Engineering and Technology and a Doctor of Philosophy (Ph.D.) from the University of Karachi.

Mehwish Jawed

Mehwish Jawed is working as a Senior Research Analyst. She holds a Master's and Bachelors of Engineering degree in Telecommunication Engineering from NED University of Engineering and Technology. She also worked under the supervision of HEC Approved supervisor. She has more than three published papers, including both conference and journal papers. She has a great knowledge of TWDM Passive Optical Network (PON). She also worked as a Project Engineer, Robotic Trainer in a private institute and has research skills in the field of communication networks. She has both technical knowledge and industry-sounding information, which she utilizes effectively when needed. She also has expertise in cloud platforms like AWS, GCP, Oracle, and Microsoft Azure.

Syeda Fariha Ashrafi

Syeda Fariha Ashrafi is working as a technical content developer. She has completed bachelor’s degree in telecommunication engineering from NED University of Engineering and Technology. She has also completed the CCNA (Routing and Switching) course. During her bachelor’s program, she has worked on the project “Smart metering using PLC (Power Line Communication).

Free Resources

For Free Resources: Please visit our website and register to access your desired Resources Or contact us at: helpdesk@ipspecialist.net

Career Report: This report is a step-by-step guide for a novice who wants to develop his/her career in the field of computer networks. It answers the following queries:

- What are the current scenarios and future prospects?
- Is this industry moving towards saturation, or are new opportunities knocking at the door?
- What will the monetary benefits be?
- Why get certified?
- How to plan, and when will I complete the certifications if I start today?
- Is there any career track that I can follow to accomplish the specialization level?

Furthermore, this guide provides a comprehensive career path towards being a specialist in networking and highlights the tracks needed to obtain certification.

IPS Personalized Technical Support for Customers: Good customer service means helping customers efficiently, in a friendly manner. It is essential to be able to handle issues for customers and do your best to ensure they are satisfied. Providing good service is one of the most important things that can set our business apart from the others of its kind.

Excellent customer service will result in attracting more customers and attain maximum customer retention.

IPS offers personalized TECH support to its customers to provide better value for money. If you have any queries related to technology and labs, you can simply ask our technical team for assistance via Live Chat or Email.

Our Products

Study Guides

IPSpecialist Study Guides are the ideal guides to developing the hands-on skills necessary to pass the exam. Our workbooks cover the official exam blueprint and explain the technology with real-life case study-based labs. The content covered in each workbook consists of individually focused technology topics presented in an easy-to-follow, goal-oriented, step-by-step approach. Every scenario features detailed breakdowns and thorough verifications to help you completely understand the task and associated technology.

We extensively used mind maps in our workbooks to visually explain the technology. Our workbooks have become a widely used tool to learn and remember information effectively.

vRacks

Our highly scalable and innovative virtualized lab platforms let you practice the IPSpecialist Study Guide at your own time and your own place as per your convenience.

Exam-Cram Notes

Our Exam Cram notes are concise bundling of condensed notes of the complete exam blueprint. It is an ideal and handy document to help you remember the most important technology concepts related to the certification exam.

Practice Questions

IP Specialists' Practice Questions are dedicatedly designed from a certification exam perspective. The collection of these questions from our Study Guides is prepared to keep the exam blueprint in mind, covering important and necessary topics. It's an ideal document to practice and revise your certification.

Hans on Labs

IPSpecialist Hands-on Labs are the fastest and easiest way to learn the real-world use cases. These labs are carefully designed to prepare you for the certification exams and your next job role. Whether you are starting to learn technology or solving a real-world scenario, our labs will help you learn the core concepts in no time.

IPSpecialist self-paced labs were designed by subject matter experts and provide an opportunity to use products in a variety of pre-designed scenarios and common use cases, giving you hands-on practice in a simulated environment to help you gain confidence. You have the flexibility to choose from topics and products about which you want to learn more.

Content at a glance

Table of Contents

AWS Cloud Certifications

AWS Certifications are industry-recognized credentials that validate your technical cloud skills and expertise while assisting in your career growth. These are one of the most valuable IT certifications right now since AWS has established an overwhelming lead in the public cloud market. Even with the presence of several tough competitors such as Microsoft Azure, Google Cloud Engine, and Rackspace, AWS is by far the dominant public cloud platform today, with an astounding collection of proprietary services that continues to grow.

The two key reasons as to why AWS certifications are prevailing in the current cloud-oriented job market:

- There's a dire need for skilled cloud engineers, developers, and architects – and the current shortage of experts is expected to continue into the foreseeable future.
- AWS certifications stand out for their thoroughness, rigor, consistency, and appropriateness for critical cloud engineering positions.

Value of AWS Certifications

AWS places equal emphasis on sound conceptual knowledge of its entire platform, as well as on hands-on experience with the AWS infrastructure and its many unique and complex components and services.

For Individuals

- Demonstrates your expertise to design, deploy, and operate highly available, cost-effective, and secure applications on AWS.
- Gain recognition and visibility for your proven skills and proficiency with AWS.
- Earn tangible benefits such as access to the AWS Certified LinkedIn Community, invite to AWS Certification Appreciation Receptions and Lounges, AWS Certification Practice Exam Voucher, Digital Badge for certification validation, AWS Certified Logo usage, access to AWS Certified Store.
- Foster credibility with your employer and peers.

For Employers

- Identify skilled professionals to lead IT initiatives with AWS technologies.
- Reduce risks and costs to implement your workloads and projects on the AWS platform.
- Increase customer satisfaction.

Types of Certification

Role-Based Certifications:

- ***Foundational*** - Validates overall understanding of the AWS Cloud. Prerequisite to achieving Specialty certification or an optional start towards Associate certification.
- ***Associate*** - Technical role-based certifications. No prerequisite.
- ***Professional*** - Highest level technical role-based certification. Relevant Associate certification required.

Specialty Certifications:

- Validate advanced skills in specific technical areas.
- Require one active role-based certification.

About AWS – Certified Machine Learning – Specialty Exam

Exam Questions	Case study, short answer, repeated answer, MCQs
Number of Questions	100-120
Time to Complete	180 minutes
Exam Fee	300 USD

Overview of AWS Machine Learning – Specialty Certification

Individuals who work in artificial intelligence/machine learning (AI/ML) development or data science should take the AWS Certified Machine Learning – Specialty (MLS-C01) exam. The exam verifies a candidate's competence to use the AWS Cloud to design, construct, deploy, optimize, train, tune, and manage machine learning solutions for specific business challenges.

A candidate's ability to accomplish the following tasks is also validated by the exam:

- Select and justify the appropriate ML approach for a given business problem
- Identify appropriate AWS services to implement ML solutions
- Design and implement scalable, cost-optimized, reliable, and secure ML solutions

The target candidate is expected to have 2 or more years of hands-on experience developing, architecting, and running ML or deep learning workloads in the AWS Cloud.

Recommended AWS knowledge

The target candidate should have the following knowledge:

- The ability to express the intuition behind basic ML algorithms
- Experience performing basic hyperparameter optimization
- Experience with ML and deep learning frameworks
- The ability to follow model-training best practices
- The ability to follow deployment best practices
- The ability to follow operational best practices

What is considered out of scope for the target candidate?

The following is a non-exhaustive list of related job tasks that the target candidate is not expected to be able to perform. These items are considered out of scope for the exam:

- Extensive or complex algorithm development
- Extensive hyperparameter optimization
- Complex mathematical proofs and computations
- Advanced networking and network design
- Advanced database, security, and DevOps concepts
- DevOps-related tasks for Amazon EMR

AWS Knowledge

- Minimum one year of hands-on experience with the AWS platform
- Professional experience managing/operating production systems on AWS

- A firm grasp of the seven AWS tenets – architecting for the cloud
- Hands-on experience with the AWS CLI and SDKs/API tools
- Understanding of network technologies as they relate to AWS
- Good grasp of fundamental Security concepts with hands-on inexperience in implementing Security controls and compliance requirements

General IT Knowledge

- 1-2 years' experience as a system's administrator in a systems operations role
- Experience in understanding virtualization technology
- Monitoring and auditing system's experience
- Knowledge of networking concepts (DNS, TCP/IP, and Firewalls)
- Ability to collaborate with developers

Intended Audience

Eligible candidates for this exam must have:

- One or more years of hands-on experience in operating AWS-based applications
- Experience in provisioning, operating, and maintaining systems running on AWS
- Ability to identify and gather requirements to define a solution to be built and operated on AWS
- Capabilities to provide AWS operations and deployment guidance and best practices throughout the life cycle of a project

Recommended AWS Knowledge

- Create data repositories for machine learning.
- Identify and implement a data ingestion solution.
- Identify and implement a data transformation solution.
- Sanitize and prepare data for modeling.
- Perform feature engineering.
- Analyze and visualize data for machine learning.
- Frame business problems as machine learning problems.
- Select the appropriate model(s) for a given machine learning problem.
- Train machine learning models.
- Perform hyperparameter optimization.
- Evaluate machine learning models.
- Build machine learning solutions for performance, availability, scalability, resiliency, and fault tolerance.
- Recommend and implement the appropriate machine learning services and features for a given problem.
- Apply basic AWS security practices to machine learning solutions.
- Deploy and operationalize machine learning solutions.

The table below lists the main content domains and their weightings on the exam.

	Domain	Percentage
Domain 1	Data Engineering	20%
Domain 2	Exploratory Data Analysis	24%
Domain 3	Modeling	36%

Domain 4	Machine Learning Implementation and Operations	20%
Total		100%

Course Introduction (MLS-C01)

Machine Learning (ML) is a form of Artificial Intelligence (AI) that allows software programs to improve their prediction accuracy without being expressly designed to do so. Machine learning algorithms use historical data to anticipate new output values.

Machine learning is critical because it helps companies see trends in consumer behavior and business operations while developing new products. Many of today's most successful firms, such as Facebook, Google, and Uber, use machine learning. ML has become a major competitive differentiator for many organizations.

In this course, you will get extensive knowledge of machine learning on AWS. Suppose you are an individual who works in development or data science. In that case, you should pursue the AWS Certified Machine Learning - Specialty Certification as it verifies the candidate's ability to create, deploy, and manage machine learning solutions for specific business issues.

Certification Approved Skills

1. Choose and justify the best machine learning technique for a particular business challenge.
2. Implement ML solutions and identify relevant AWS services.
3. Create scalable, cost-effective, dependable, and secure machine learning systems.

In this course, you will learn about data engineering, data analysis, model design, algorithm selection, optimization, and implementation of machine learning on AWS. This course covers SageMaker, Kinesis Glue, and all the other services that come together to create one of the most powerful ML platforms in the industry.

Recommended AWS Knowledge

The ideal applicant should have one to two years of hands-on experience building, architecting, or operating machine learning/deep learning workloads on the AWS Cloud, as well as the following skills:

- The ability to communicate the intuition underlying fundamental machine learning techniques
- Working knowledge of fundamental hyperparameter optimization
- Working knowledge of machine learning and deep learning frameworks is advantageous
- The capacity to follow best practices in model training
- The ability to adhere to optimal deployment and operating procedures

The AWS Certified Machine Learning Specialty test includes:

- Machine learning algorithms built into Amazon SageMaker (XGBoost, BlazingText, Object Detection, etc.)
- Imputation, outliers, binning, and normalizing are examples of feature engineering techniques
- Comprehend, Translate, Polly, Transcribe, Lex, Rekognition, and other high-level ML services
- S3, Glue, Kinesis, and DynamoDB are all useful for data engineering
- Data exploration using scikit learn, Athena, Apache Spark, and EMR
- Deep neural network hyperparameter tweaking and deep learning
- SageMaker automates model tuning and operations
- Regularization at the L1 and L2 levels
- Machine learning pipelines using security best practices

Amazon Machine Learning

AWS now offers Amazon SageMaker, a sophisticated cloud-based solution that allows developers of all skill levels to leverage machine learning technologies. SageMaker is a fully managed machine learning service that assists you in developing sophisticated machine learning models. Data scientists and developers may use SageMaker to create and train machine learning models, then deploy them into a production-ready hosted environment.

Machine Learning Concepts

Machine learning can assist you in making better business decisions by utilizing previous data. ML algorithms look for patterns in data and use that information to build mathematical models. The models can then be used to create predictions based on future data. For instance, a machine learning model may be used to forecast how likely a buyer is to purchase a specific product based on their previous behavior.

Solving Business Problems with Amazon Machine Learning

Supervised machine learning is a method of learning from data that has been labeled with the correct answer. You can use Amazon Machine Learning to apply machine learning to issues when you already have examples of real-world solutions. To use Amazon ML to predict if an email is spam, for example, you will need to gather email examples that have been appropriately tagged as spam or not. Then, using machine learning, you can generalize from these email examples to forecast whether a new email is likely to be spam or not.

For these specific machine learning problems, you can utilize supervised ML approaches binary classification (predicting one of two possible outcomes), multi-class classification (predicting one of more than two possible outcomes), and regression (predicting a numeric value).

Lab 01: Use Amazon Rekognition

Certifications	AWS Machine Learning - Specialty
Cloud Provider	AWS
Time Required	1 Hr.
Tags	Rekognition, AWS

Lab Prerequisites

- Familiarity with basic AWS Cloud Computing concepts and terminology
- An AWS account with an active subscription

Service Introduction

Amazon Rekognition is a way of converting images into tags or texts. It is a way of allowing your software to see the image. You can upload an image, and Rekognition will tell you what it thinks the image is with a certain degree of confidence. It can be used with lots of apps.

You can recognize objects, scenes, people, activities, and language in photographs and videos and detect inappropriate content using Amazon Rekognition. You may utilize Amazon Rekognition's highly accurate facial analysis and facial search capabilities to recognize, analyze, and compare faces for a range of user verification, people counting, and public safety use cases.

Case Study Enterprise Healthcare - Cleveland Medical Center

<u>Background</u>

Cleveland Medical Centers are stretched across the three states of California, Oregon, and Hawaii. The centers provide service to millions of lives through hospital admissions, emergency, clinic, outpatient, and home care visits. Cleveland uses advanced technology that helps restore patient health and empowers physicians to deliver better care to achieve better outcomes.

Cleveland is currently in the middle of a digital transformation to bring patient care closer to home. They aim to treat patients at the nearest hospital or resource center in their area or dispatch healthcare workers to assist them in their homes. To do this, they plan to use video conferencing, IoT, remote monitoring, and mobile solutions that will enable people to get the healthcare they need, thus limiting the number of patients who have to travel to one of their hospitals. This way is not only cost-effective, but it also gives patients a better experience.

An external consultant has done an extensive study on the Cleveland infrastructure. The consultant has advised the board to build a “Cloud First” strategy. This will reduce the need for organizations to invest in and maintain their on-premises IT resources, scalability to meet workload and user demands, and fewer wasted resources because the company will only pay for what they use.

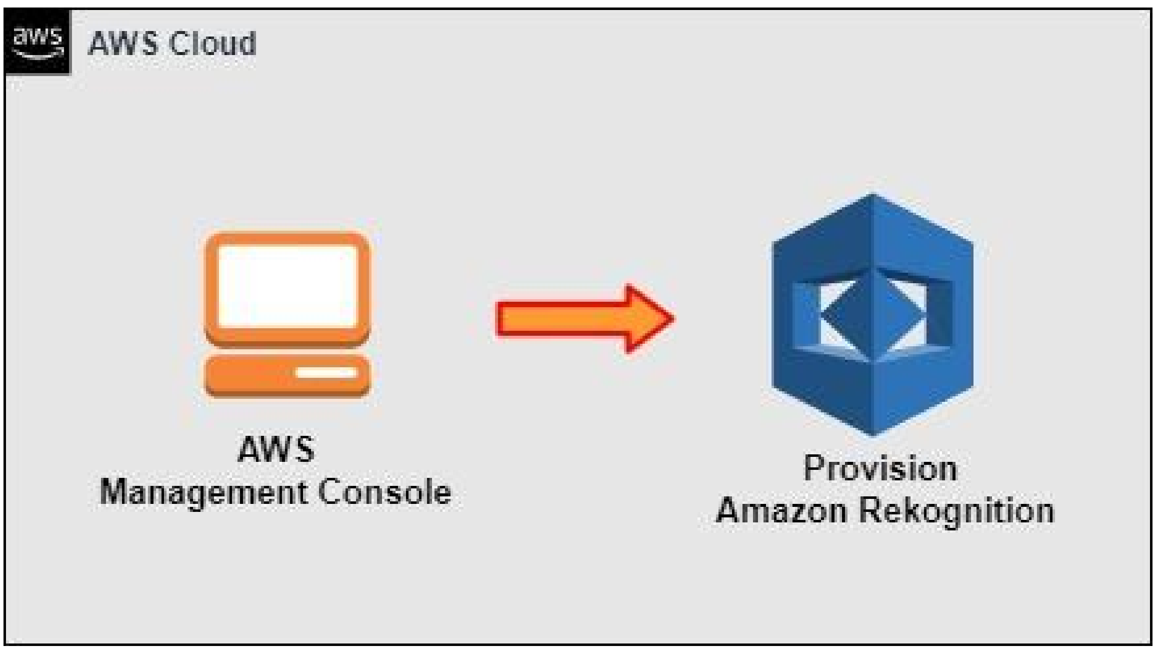
<u>Business Challenge</u>

Cleveland's business development team has deployed an online appointment portal where users conveniently schedule their appointments. Hundreds of patients come to the medical centers daily through online appointments, and management has faced trouble identifying the patients by appointment number only. Cleveland's business development team wants a solution that identifies the patient by face and shows the basic details. The management team hired a Cloud Specialist to propose a solution for the current requirement.

<u>Proposed Solution</u>

You are the Cloud Specialist for Cleveland medical center. You proposed a solution to this requirement by using Amazon Rekognition. Cleveland's management team can easily identify the patient by using the facial analysis feature of Amazon Rekognition. To use this feature, the picture of the patients should be uploaded at the time of the appointment. When patients come for medical consultancy, management identifies the patient through the Amazon Rekognition service.

<u>Lab Diagram</u>



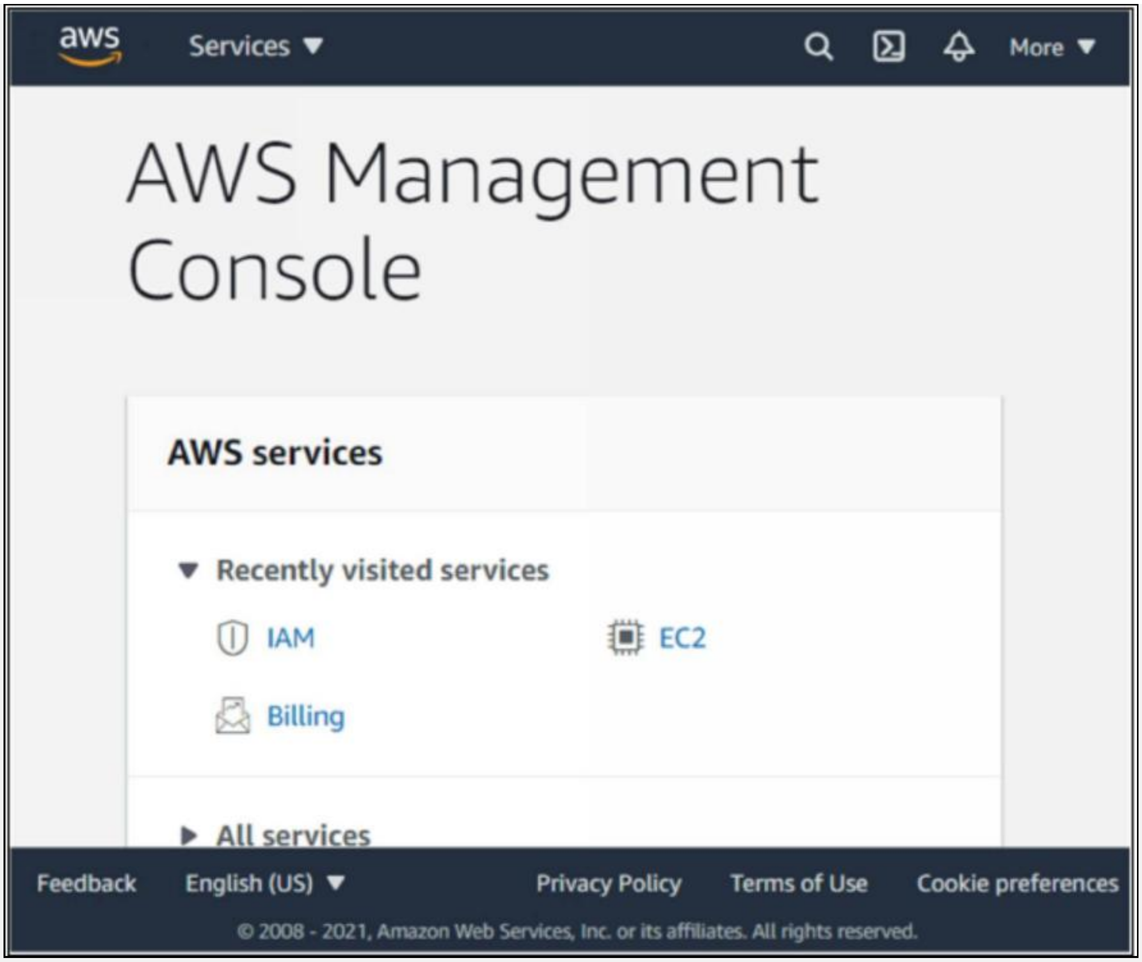
<u>Implementation Steps</u>

1. Navigate to the Amazon Rekognition.
2. Use Facial Analysis.

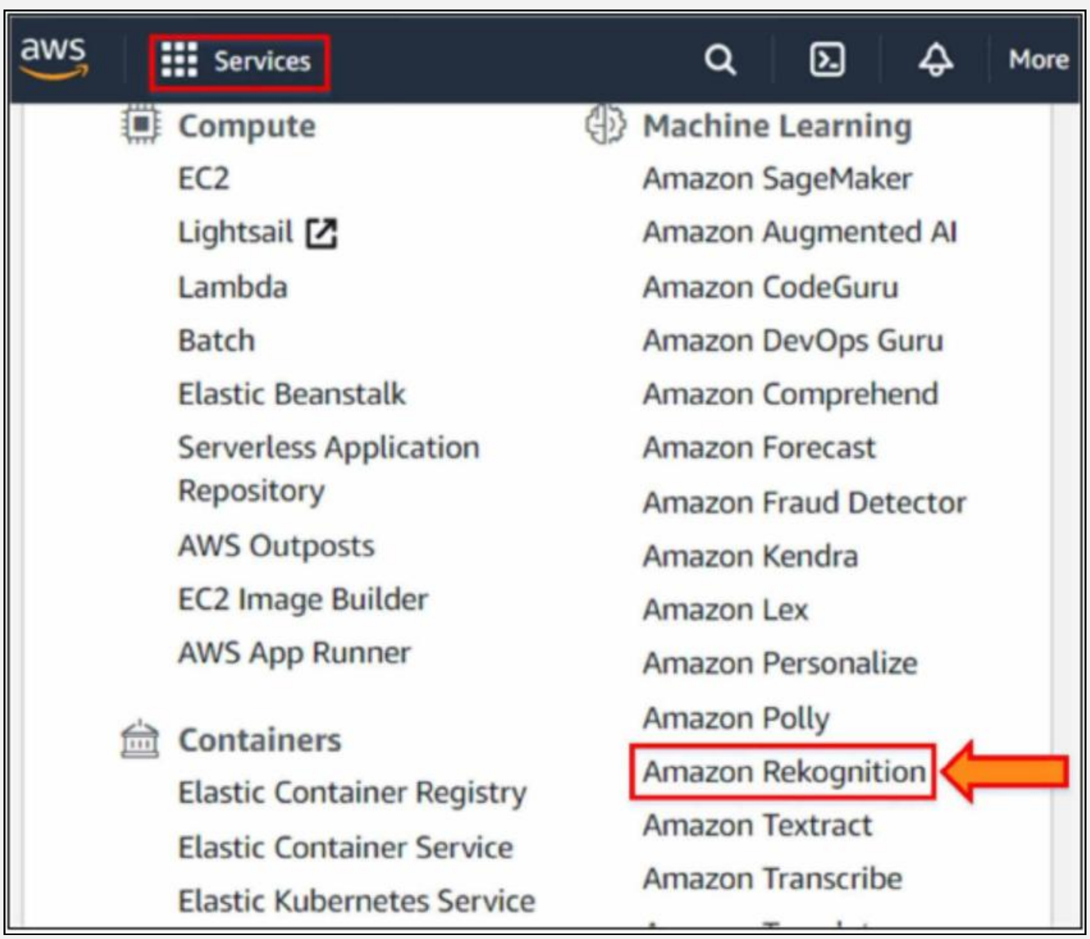
<u>Solution</u>

Step 1: Navigate to the Amazon Rekognition

1. Log in to **AWS Management Console**.

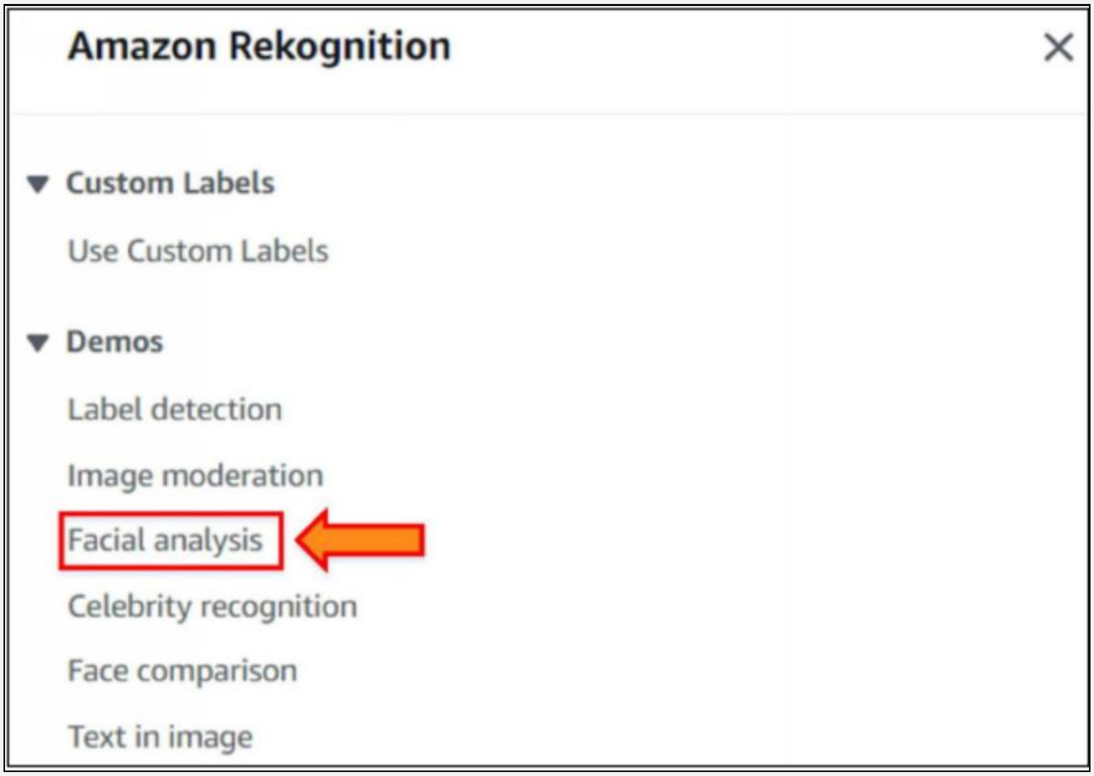


2. Click on **Services**.
3. Click on **Amazon Rekognition** under **Machine Learning**.



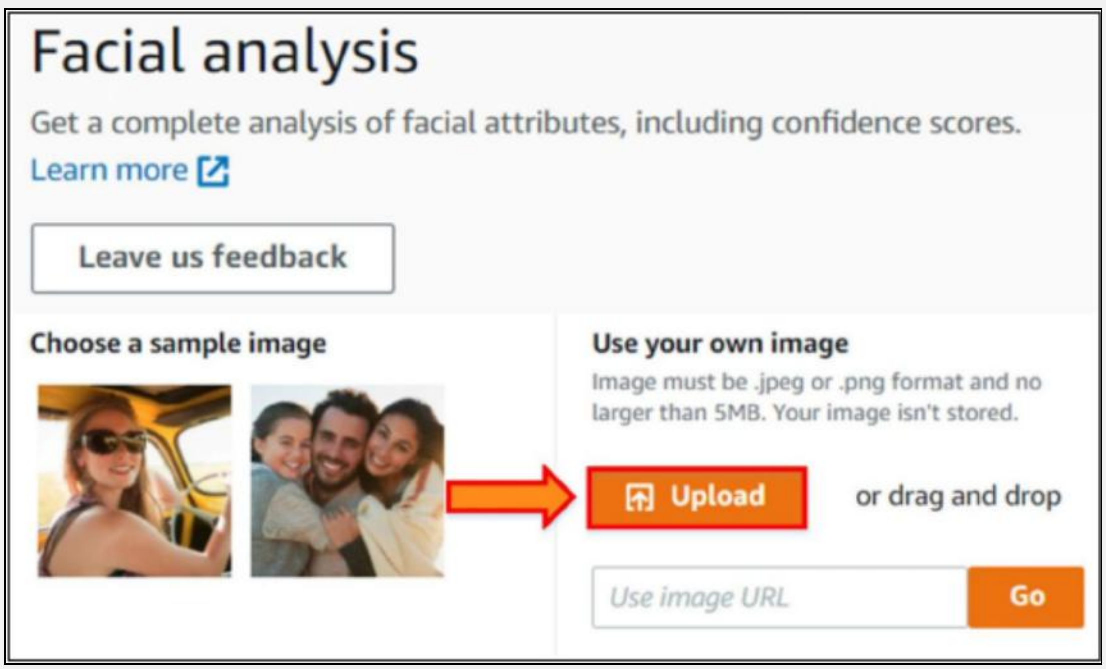
4. On the **Amazon Rekognition** page, click on the **Facial analysis**

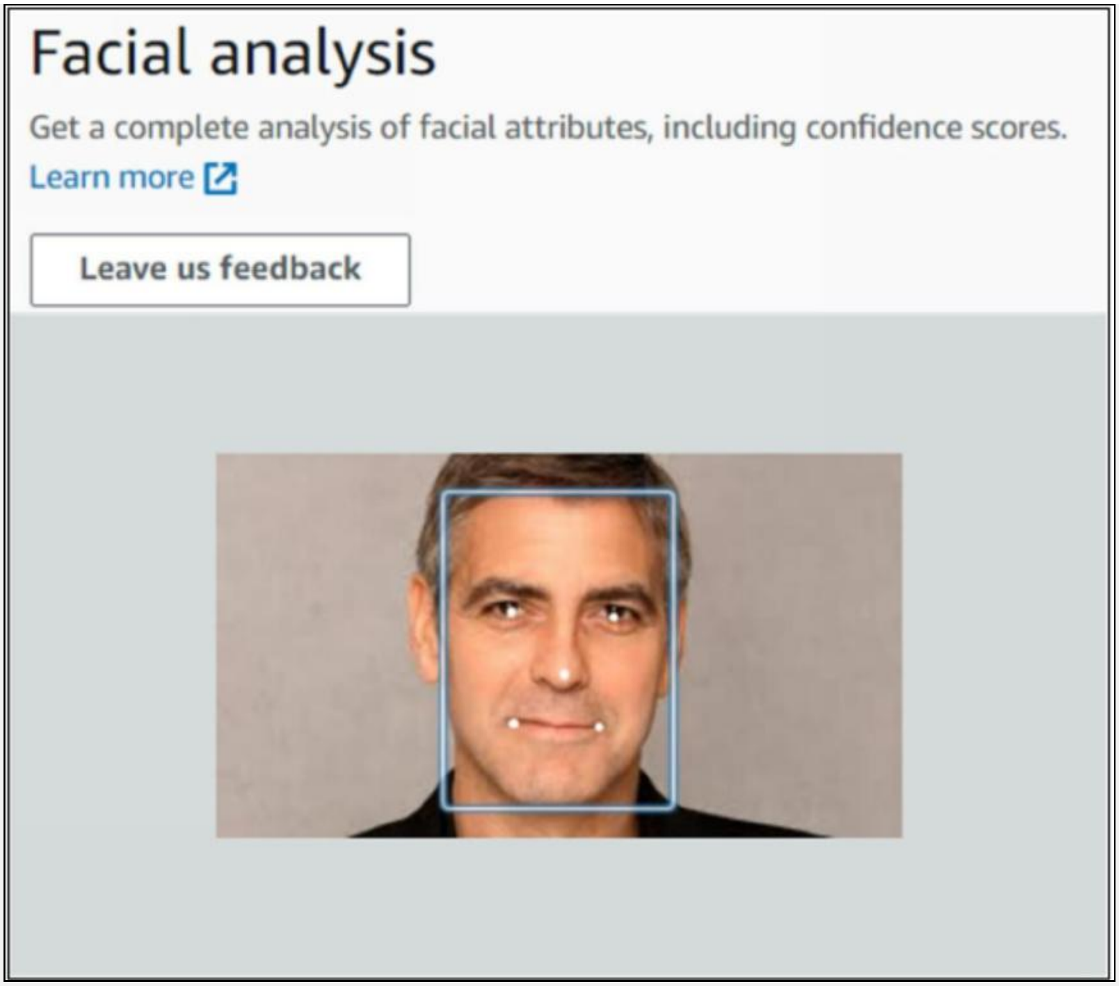
feature.



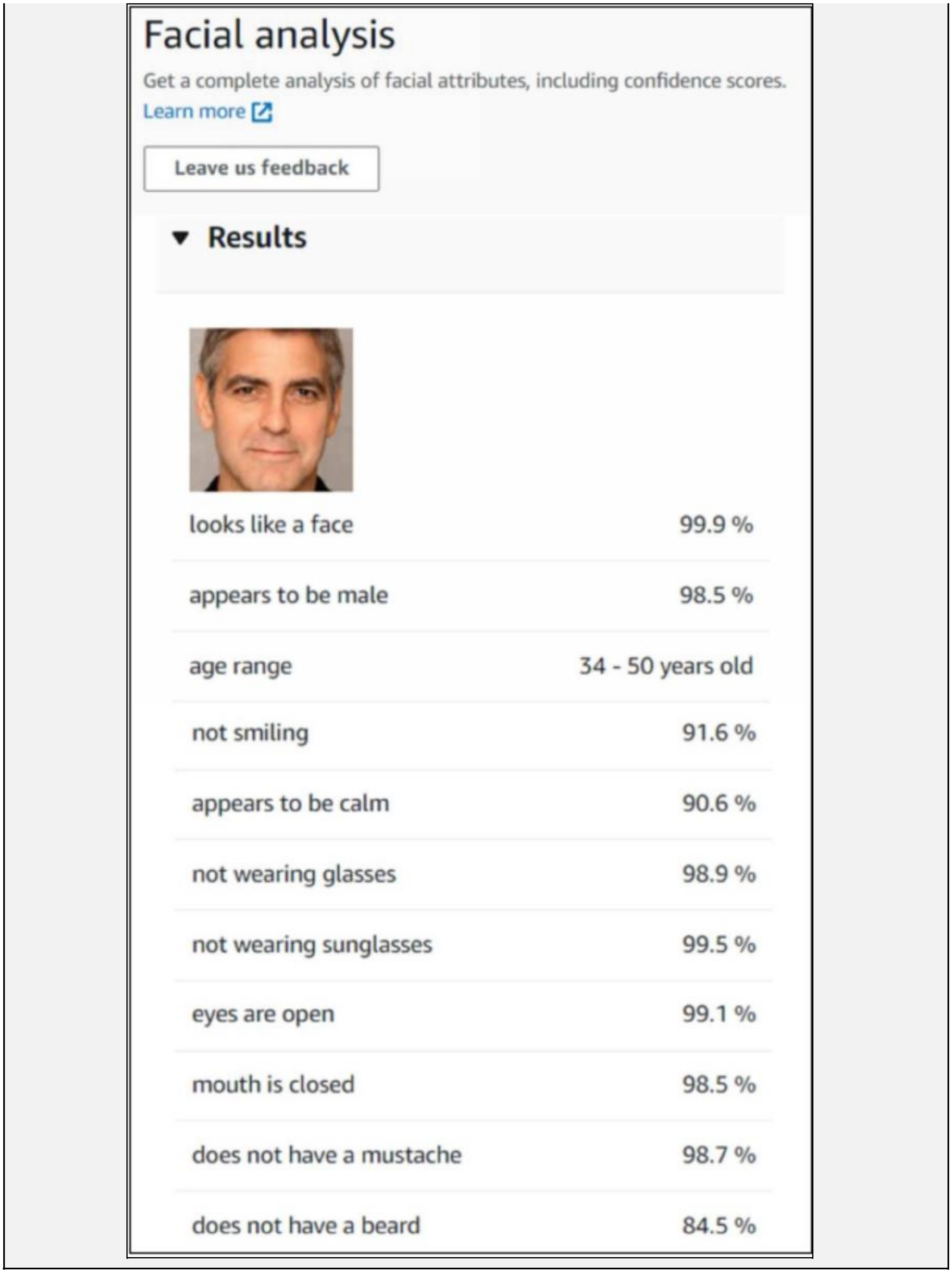
Step 2: Use Facial Analysis

1. When the **Facial analysis** page appears, scroll down and click on **Upload** to upload the patient's picture.





2. The facial analysis will show the **Result** after analyzing the **picture**.
3. The **Result** will be shown below.



Lab 02: Creating a Lambda Function

Certifications	AWS Machine Learning - Specialty
Cloud Provider	AWS
Time	

Required	1 Hr.
Tags	Lambda, AWS

Lab Prerequisites

- Familiarity with basic AWS Cloud Computing concepts and terminology
- An AWS account with an active subscription

Service Introduction

AWS Lambda is a serverless compute service that lets you run code without needing to deploy or manage servers, design workload-aware cluster scaling logic, maintain event integrations, or manage runtimes. You can run code for nearly any form of application or backend service with Lambda, and you do not have to worry about administration. Therefore, you do not need to worry about which AWS resources to launch or how well you manage them. Instead, you must upload the code to Lambda, and it will execute.

The code in AWS Lambda is performed in reaction to events in AWS services, such as adding/deleting files in an S3 bucket, making an HTTP call to the Amazon API gateway, and so on. On the other hand, Amazon Lambda can only be utilized to do background activities.

Case Study Enterprise Healthcare - Cleveland Medical Center

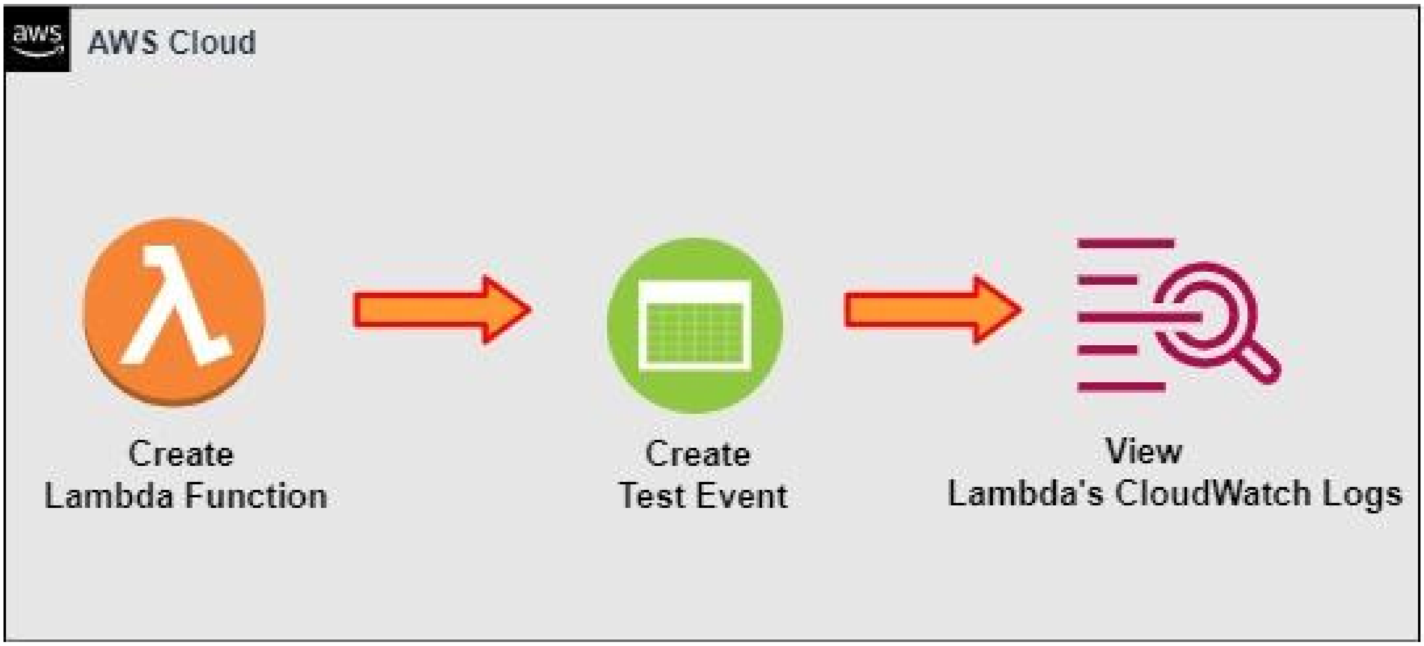
Business Challenge

Cleveland's business development team plans to shift its resources from on-premises to the cloud. One of the employees suggested using AWS cloud service providers to pursue this requirement. After migration, Cleveland wants automation for managing the underlying compute service. The management team hired a Cloud Specialist to propose a solution for the current requirement.

Proposed Solution

You are the Cloud Specialist for Cleveland medical center. You proposed a solution to this requirement by using AWS Lambda, and Cleveland can automatically manage the underlying compute resources for you with high scalability, performance, and security.

Lab Diagram



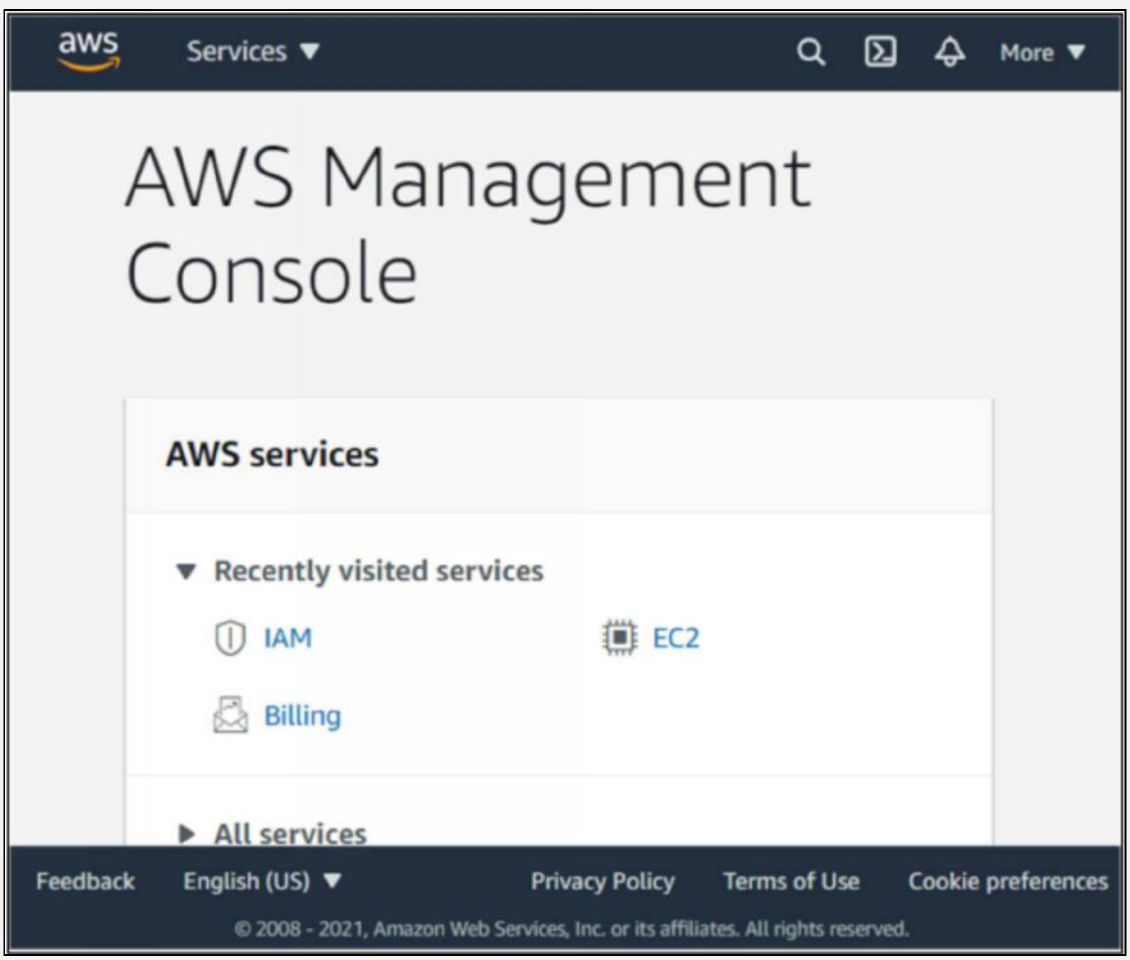
Implementation Steps

1. Author Node.js Lambda Function in AWS Console.
2. Test Function Using a Test Event.
3. View Lambda's CloudWatch Logs.

Solution

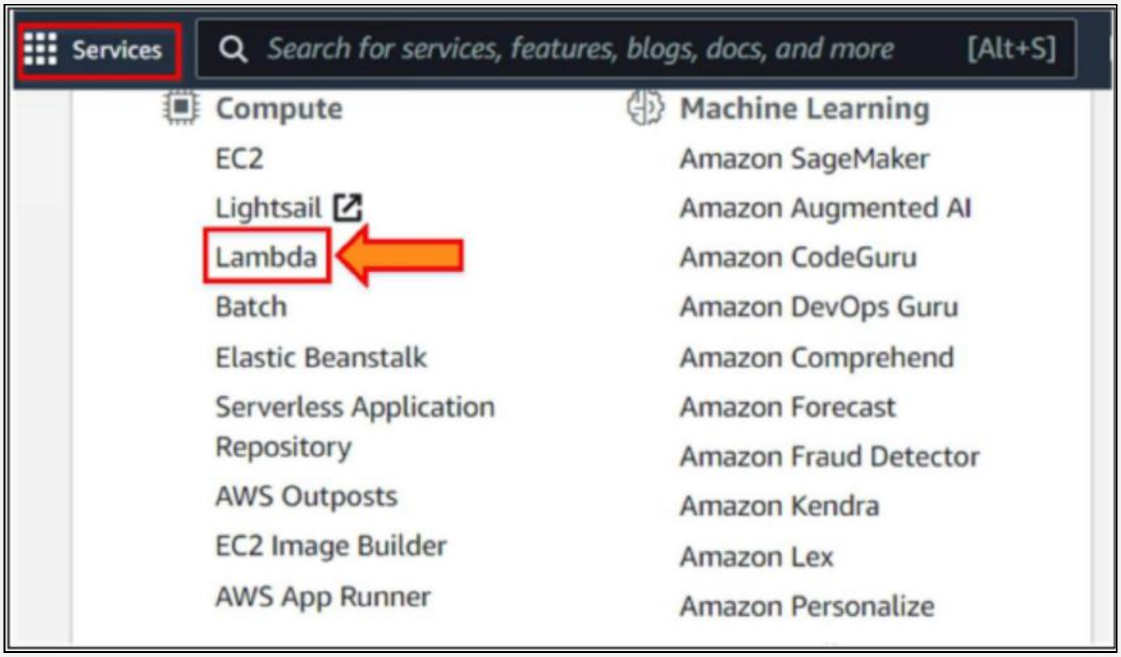
Step 1: Author Node.js Lambda Function in AWS Console

1. Log in to **AWS Management Console**.

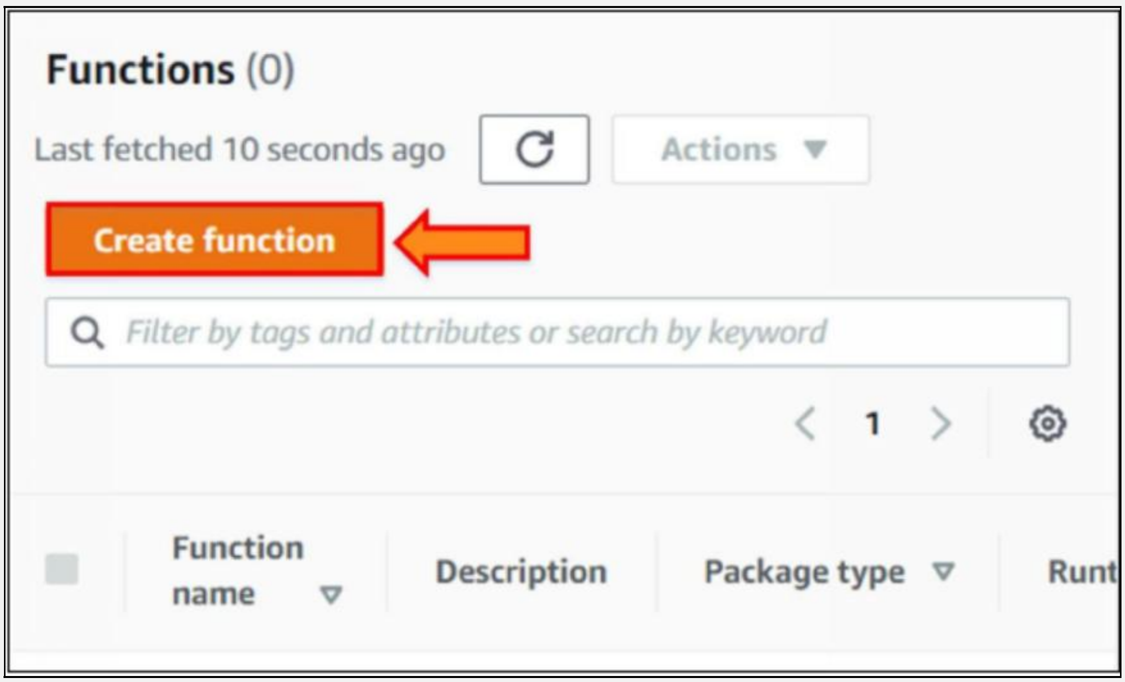


2. Click on **Services**.

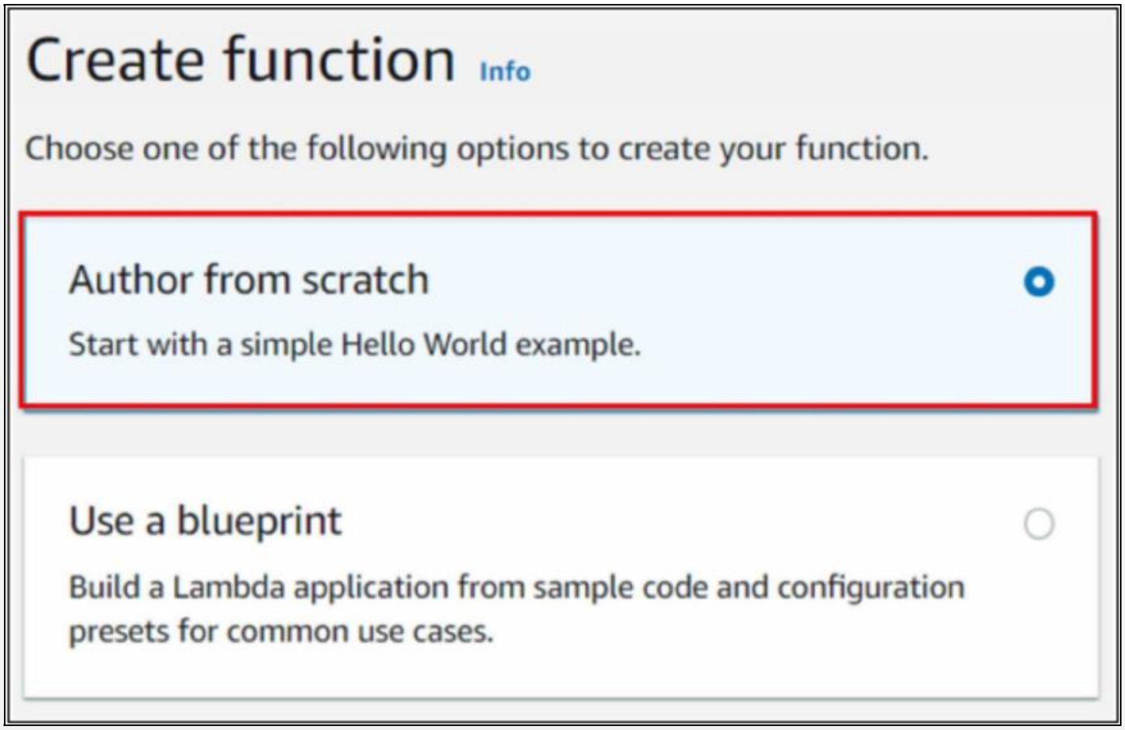
3. Click on **Lambda** under **Compute**.



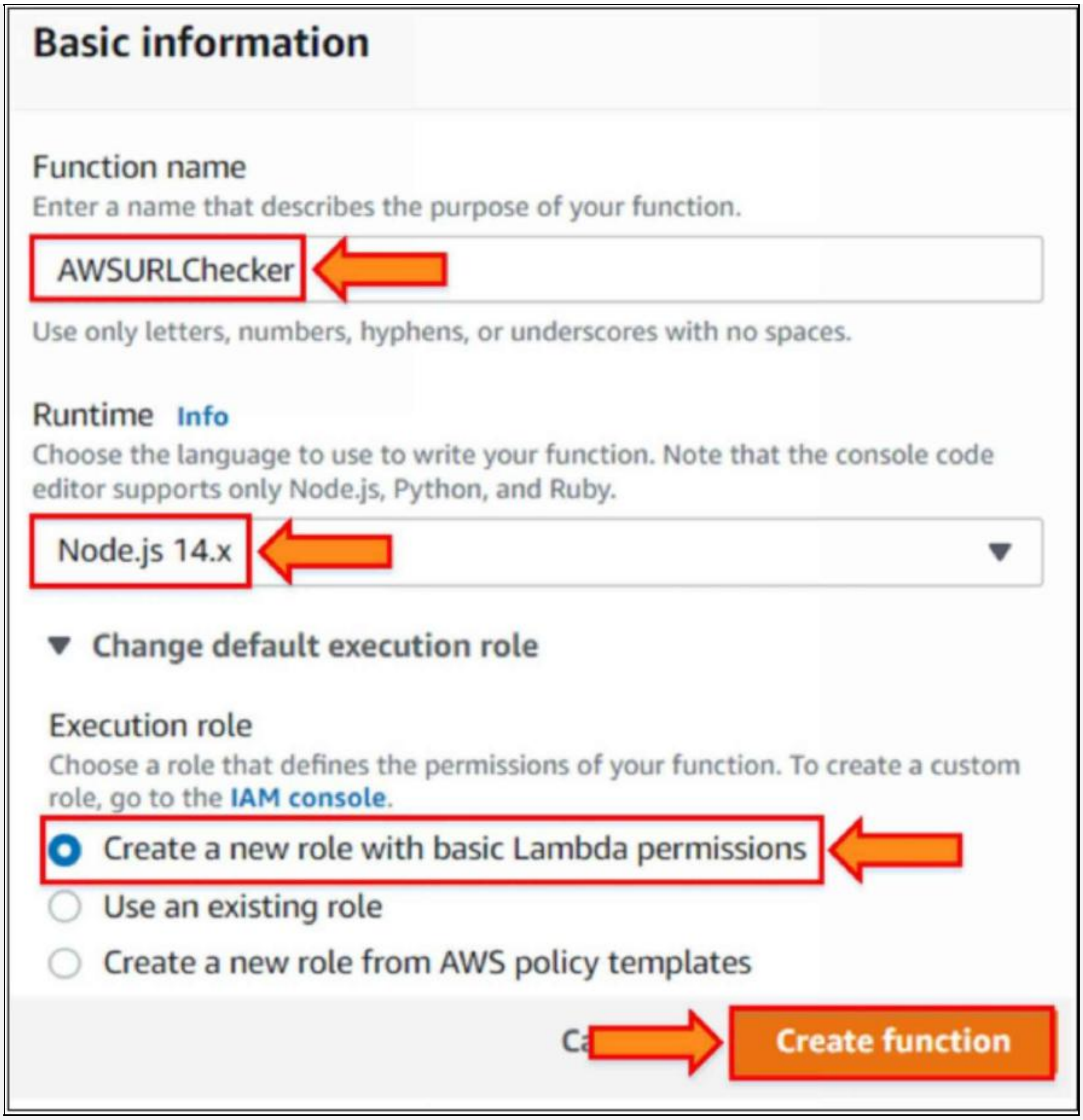
4. Click on **Create function**.



5. Select **Author from scratch**.

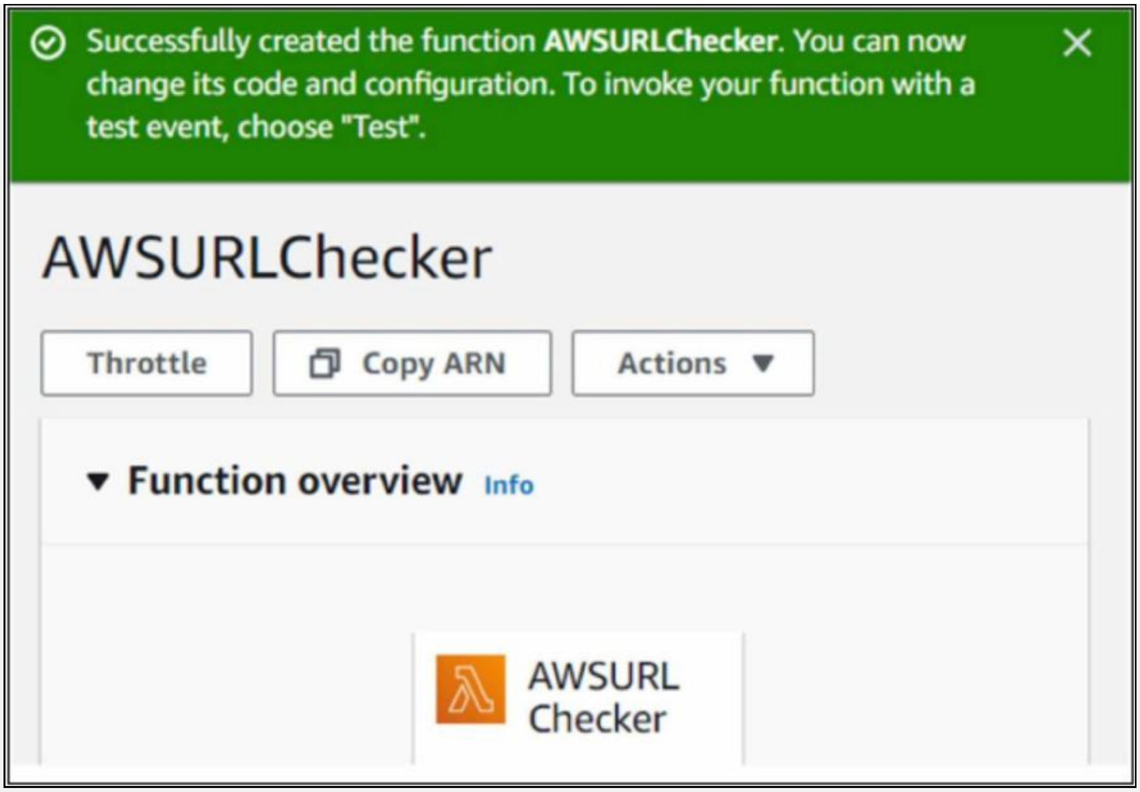


6. Set the following **values**:
   Function name: **AWSURLChecker**
   Runtime: **Node.js**
7. Expand **Change default execution role**.
8. For the **Execution role**, select **Create a new role with basic Lambda permissions**.
9. Click **Create function**.



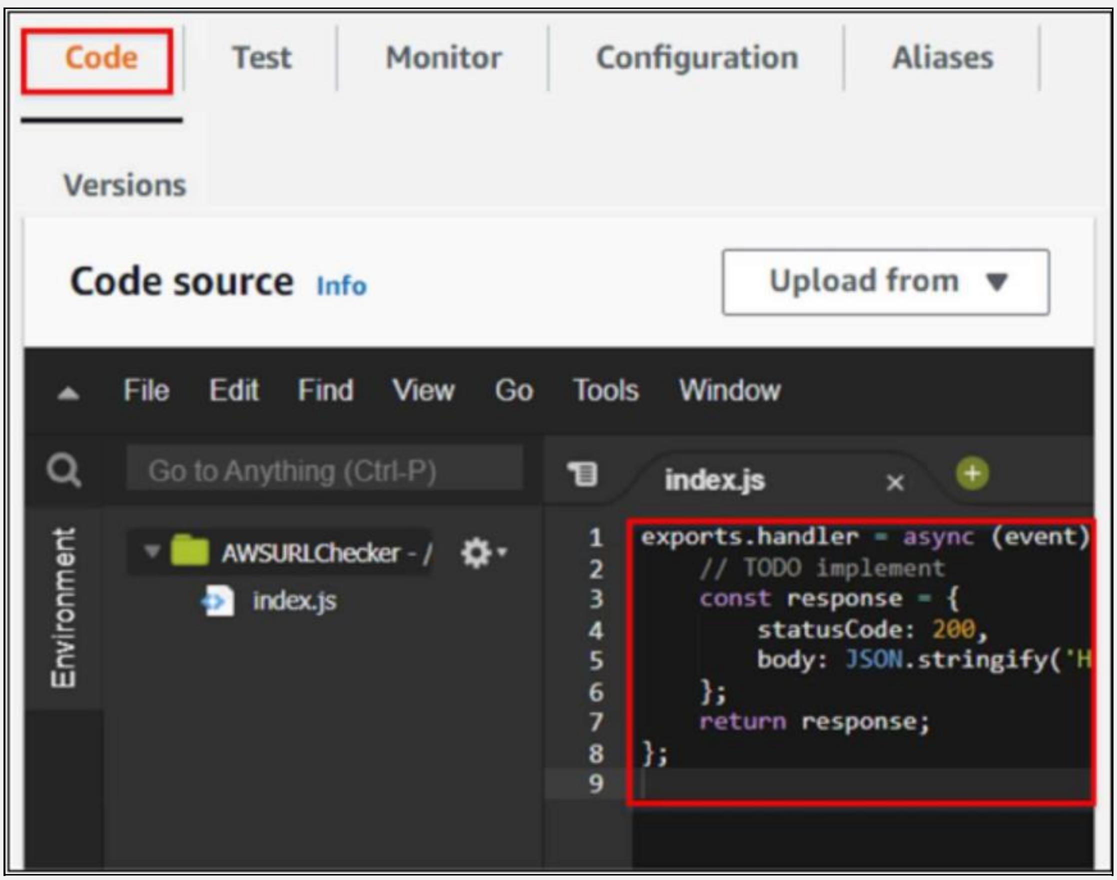
10.

The function is now successfully created.



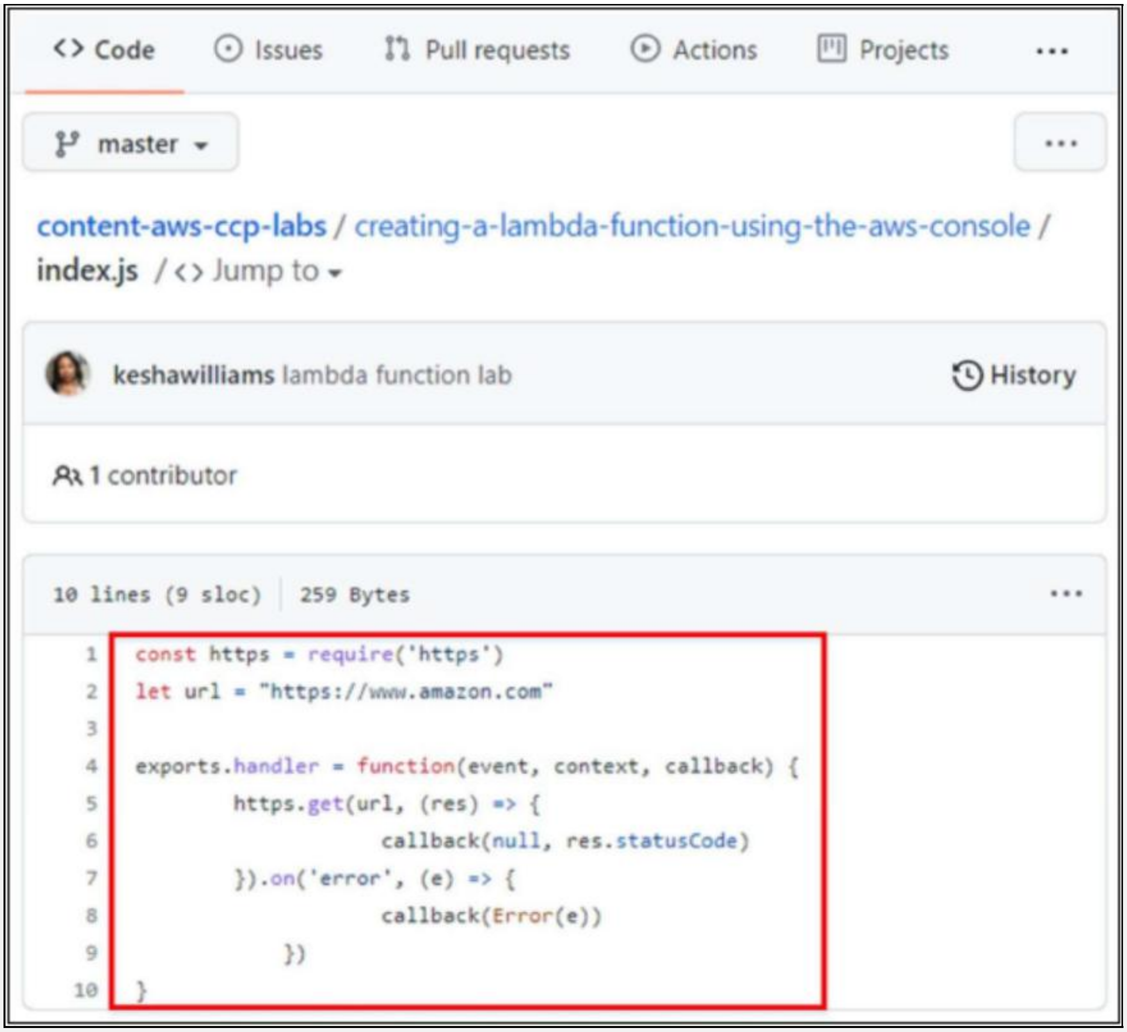
11.

Now, click on **Code**. Click on **index.js** and **delete** the code.



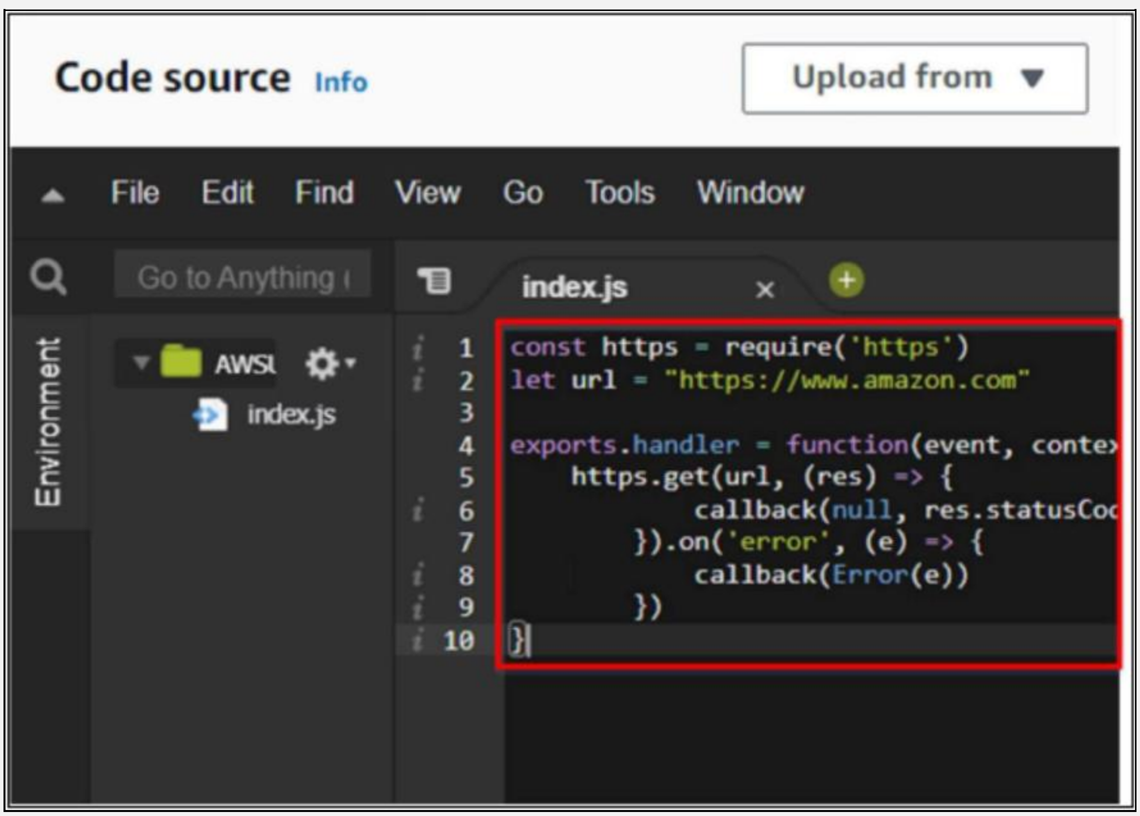
12.

Then, go to the **GitHub link** provided and copy the code.
[https://github.com/linuxacademy/content-aws-ccp-labs/blob/master/creating-a-lambda-function-usi](https://github.com/linuxacademy/content-aws-ccp-labs/blob/master/creating-a-lambda-function-using-aws-console/index.js)
[aws-console/index.js](https://github.com/linuxacademy/content-aws-ccp-labs/blob/master/creating-a-lambda-function-using-aws-console/index.js)



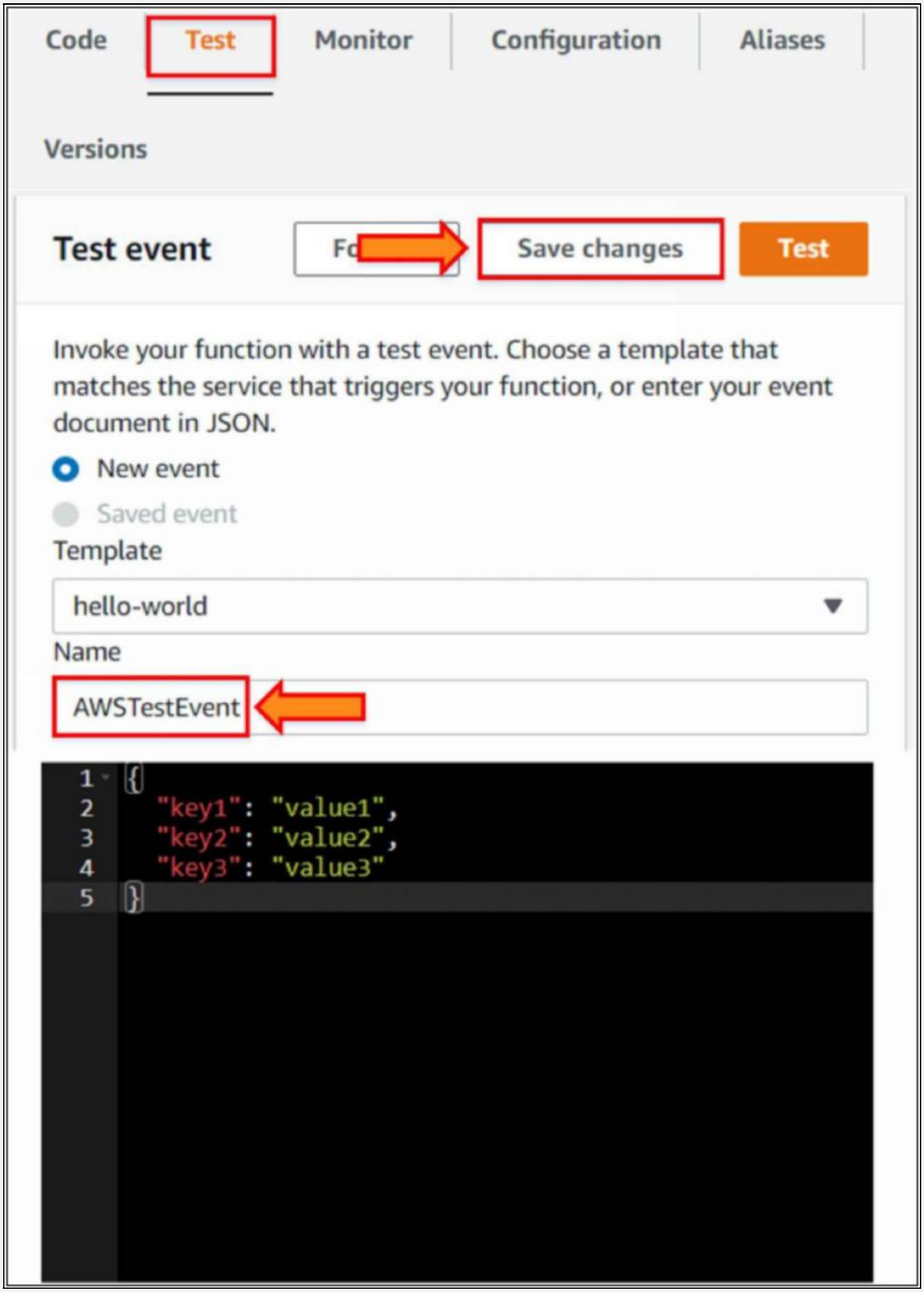
13.

Paste it into **index.js** and deploy.

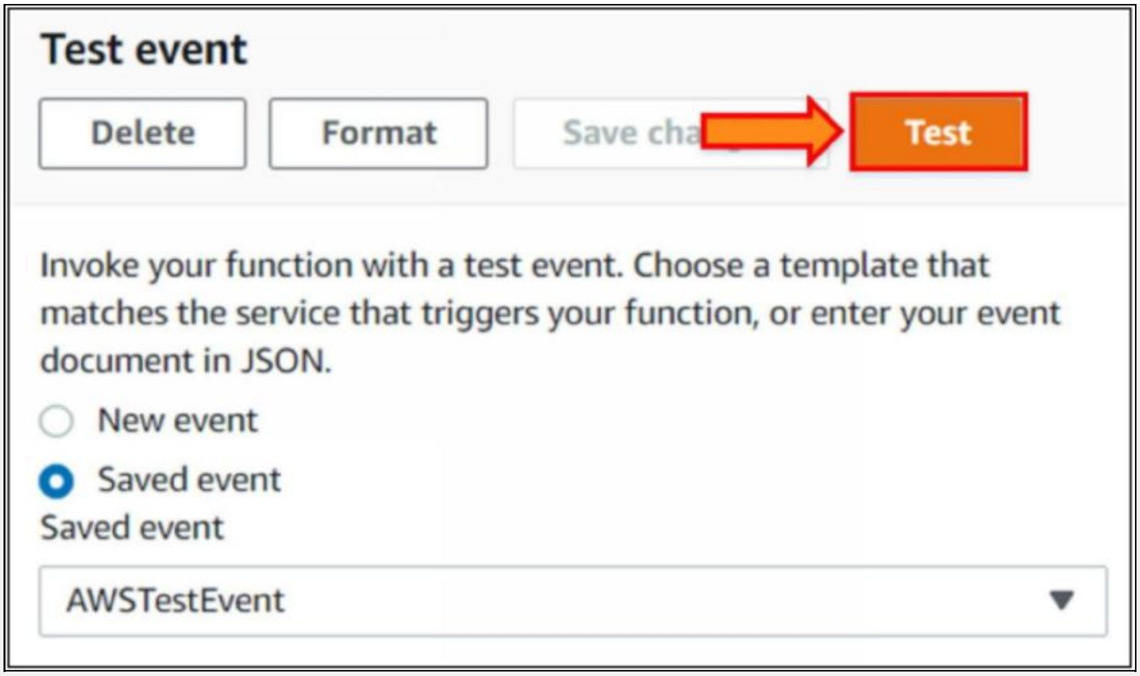


Step 2: Test Function Using a Test Event

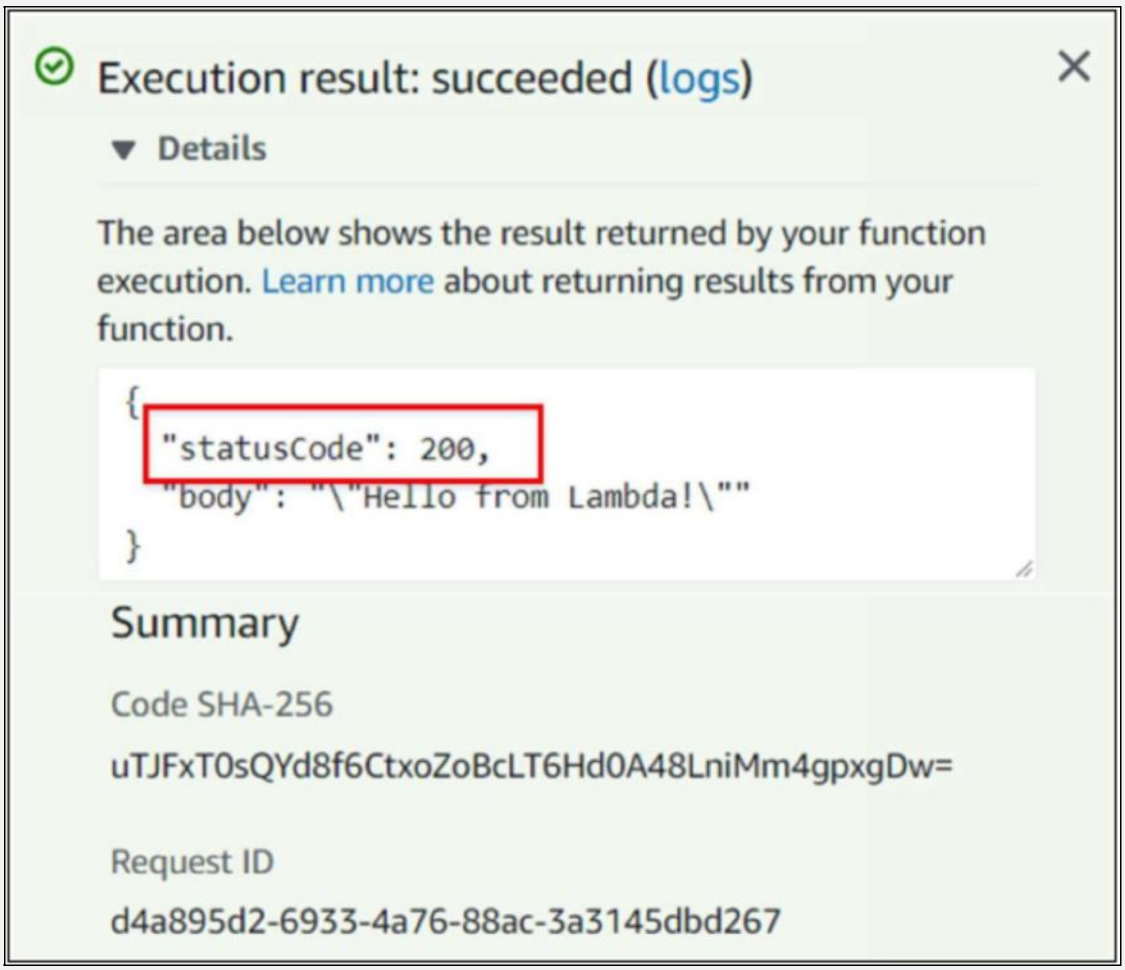
1. Click **Test** in the upper right.
2. For the **Event** template, enter **AWSTestEvent**.
3. Leave the other **default** values.
4. Click **Save changes**.



5. When the **Test event** is saved, click on **Test**.

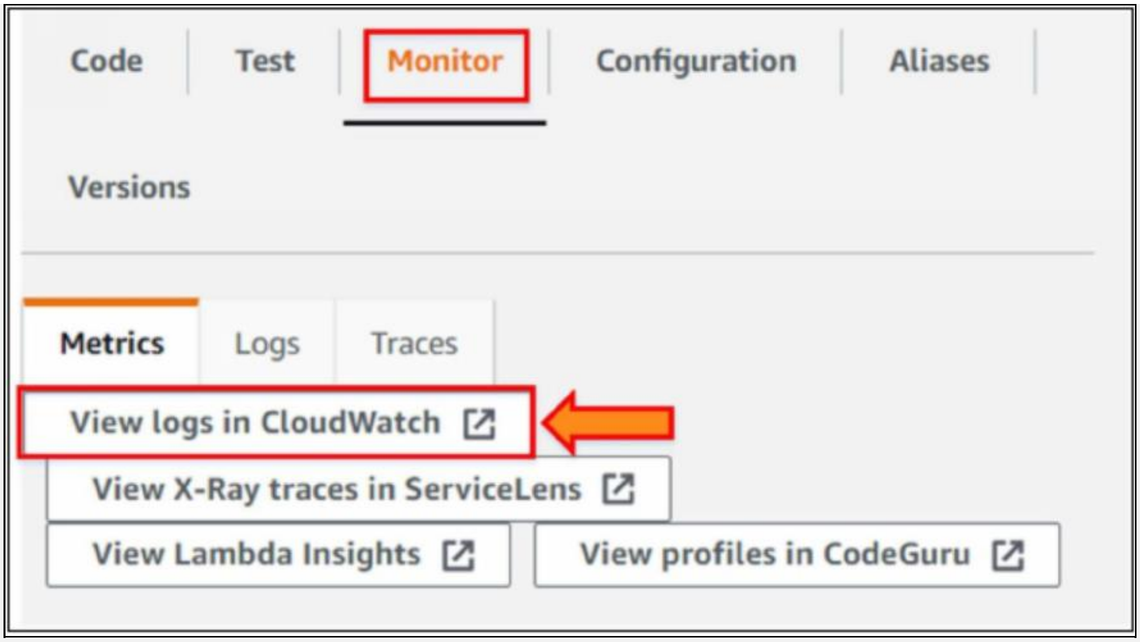


6. Review the **output response**.
7. It should be **200**, which means the website is **up** and **running**.

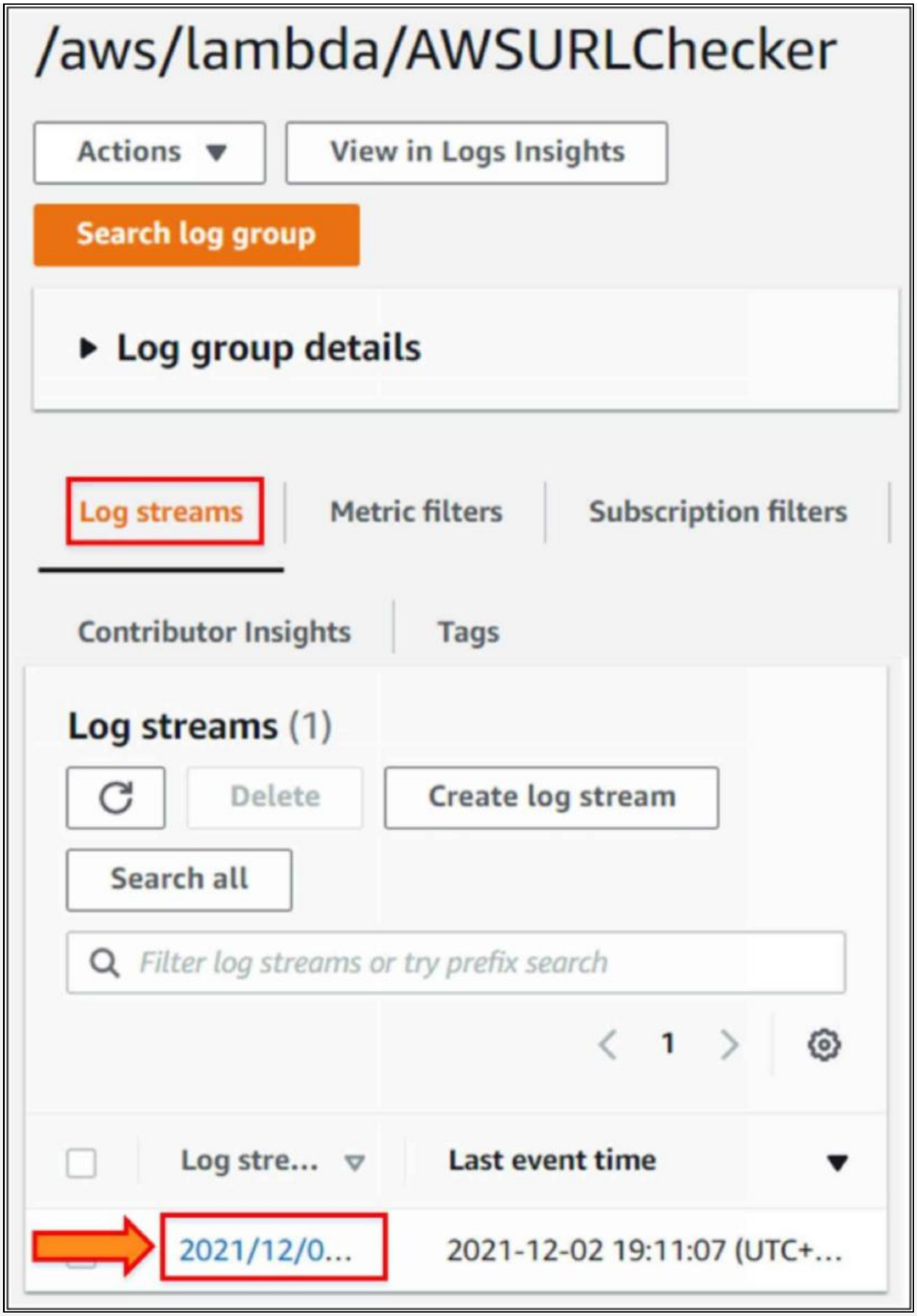


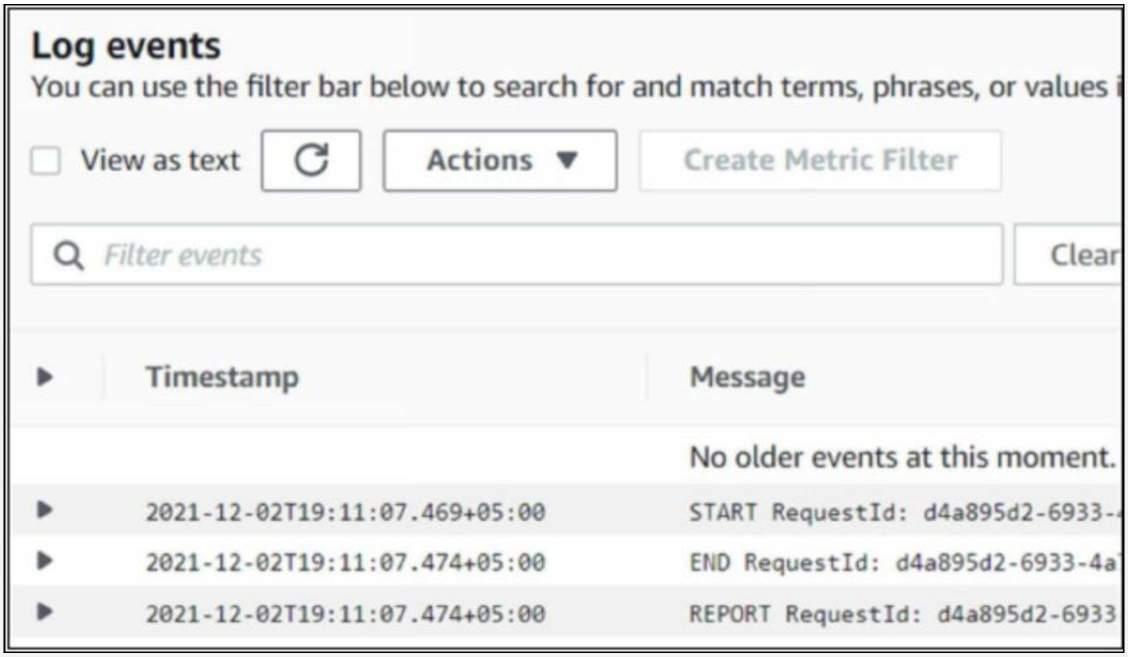
Step 3: View Lambda's CloudWatch Logs

1. Click the **Monitoring** tab.
2. Click **View logs** in **CloudWatch**.

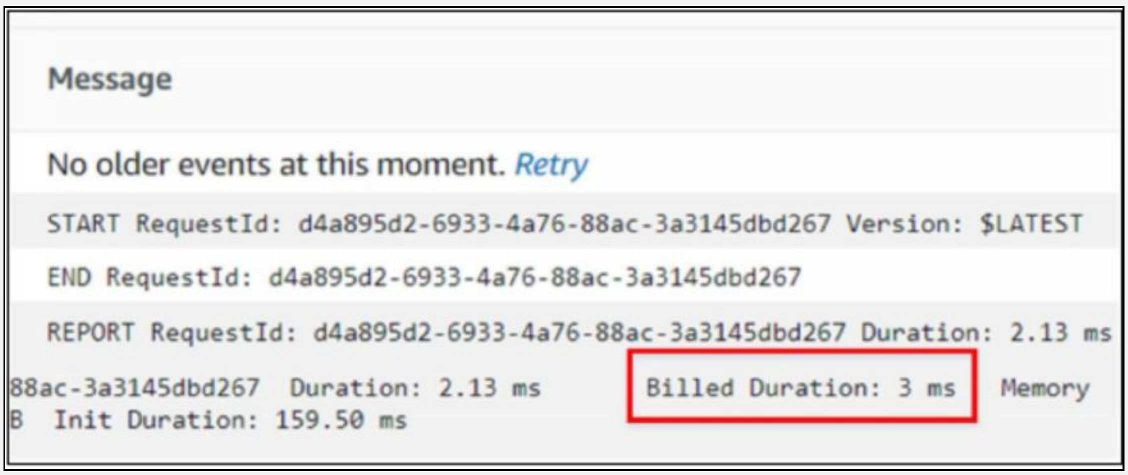


3. Click on the most recent log stream in the **Log streams** section.


/aws/lambda/AWSURLChecker
Actions
View in Logs Insights
Search log group
Log group details
Log streams
Metric filters
Subscription filters
Contributor Insights
Tags
Log streams (1)
Delete
Create log stream
Search all
Filter log streams or try prefix search
1
Log stre...
Last event time
2021/12/0...
2021-12-02 19:11:07 (UTC+...



4. Notice the **Billed duration**.



Lab 03: Build A Serverless Webpage with API Gateway and AWS Lambda

Certifications	AWS Machine Learning - Specialty
Cloud Provider	AWS
Time Required	1 Hr.
Tags	Lambda, AWS

Lab Prerequisites

- Familiarity with basic AWS Cloud Computing concepts and terminology
- An AWS account with an active subscription

Service Introduction

AWS Lambda is the ultimate abstraction layer. Lambda is a Function as a Service (FaaS) product, which means you have a function provided with the code, and AWS executes that code. In this, unlike traditional compute, you are only billed for the number of milliseconds in which your code runs. Lambda runs the code in an isolated environment, and it is stateless, but you are responsible for persistence as the environment where Lambda runs has default access to no information except incoming payload. Lambda does not maintain any result of data after the termination of a function, so it is your responsibility to put the result in a secure place.

In serverless architecture, the user sends a request to API Gateway, which then responds to AWS Lambda. AWS Lambda can communicate with databases like Dynamo DB, Amazon Aurora, or serverless Aurora. It will then send a response back to the user.

Case Study Enterprise Educational Services – Abhay Engineering College (AEC)

Background

AEC in New Zealand has more than 30,000 students and 10,000 university employees who work over seven campuses in Auckland and one campus in the Northland region of New Zealand. To keep its position as New Zealand's top university, AEC embraced digital transformation across its operations to effectively meet the current and future needs of its students and staff.

The students at the college are actively researching in the areas of machine learning, image recognition, and artificial intelligence. AEC is looking to opt for a robust strategy for the composition of these services. The demand varies based on the number of research fellows involved in a particular area at one time.

AEC is currently managing two datacenters in Auckland. The datacenters are hosting around 30 applications for students and staff. The college started its e-learning program a couple of years back, which enables remote students from all over the world to get an AEC degree online. It has put a significant load on the already stressed legacy infrastructure.

The management has restricted any further expansion in the on-premises datacenters. The CTO is working on a cloud strategy to migrate the applications to Cloud in a phased approach.

You are working as a consultant for AEC. You will support AEC in solving a number of business and technical challenges as part of this case study.

Business Challenge

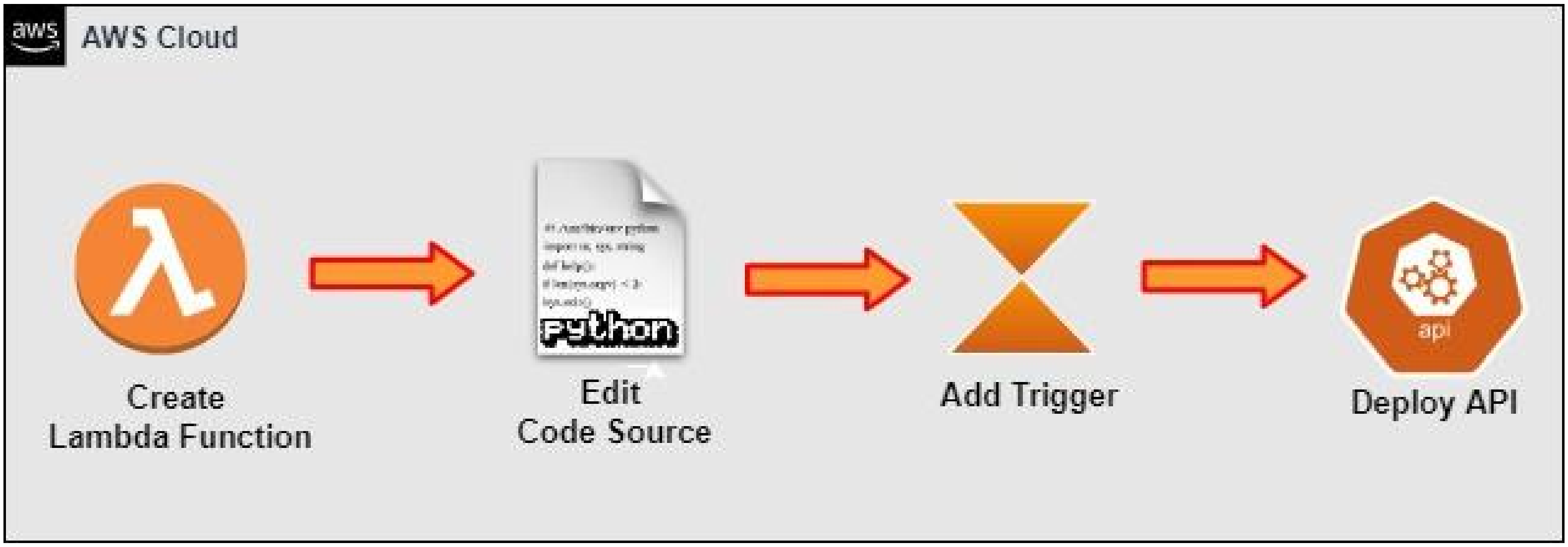
AEC is planning to launch a new student portal on AWS Cloud where a student submits their assignments, taking lectures and notes from there. The portal should be available 24/7 for all students. The management wants a solution that does not require anything to manage. They have approached you for this task.

Proposed Solution

You are the Cloud Specialist for AEC. You proposed a solution to their requirement by using AWS

Lambda. AEC can automatically manage the underlying compute resources for you with high scalability, performance, and security. This solution will configure a Serverless Webpage with API Gateway and AWS Lambda.

Lab Diagram



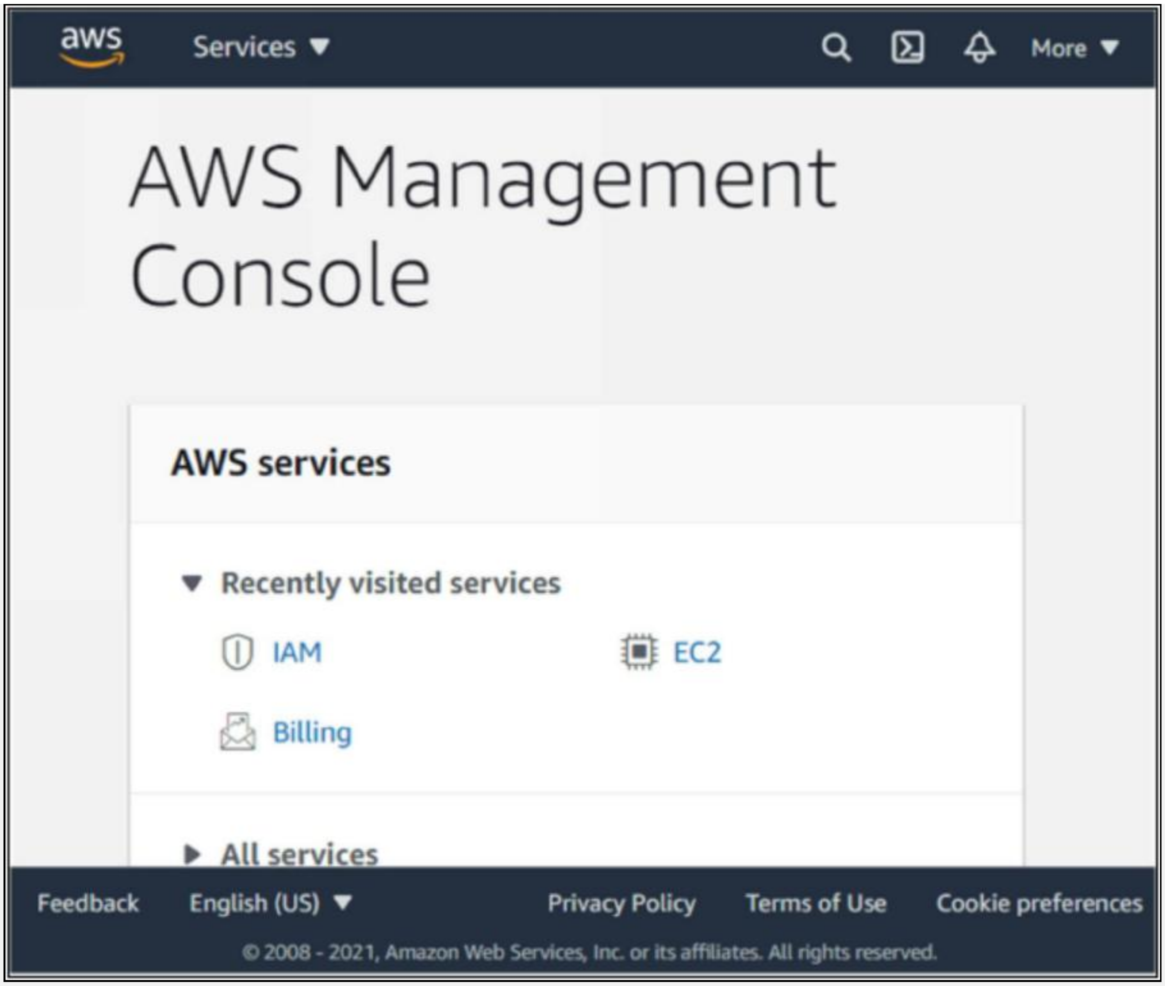
Implementation Steps

1. Create a Lambda Function
2. Edit Code Source
3. Add Trigger
4. Deploy API

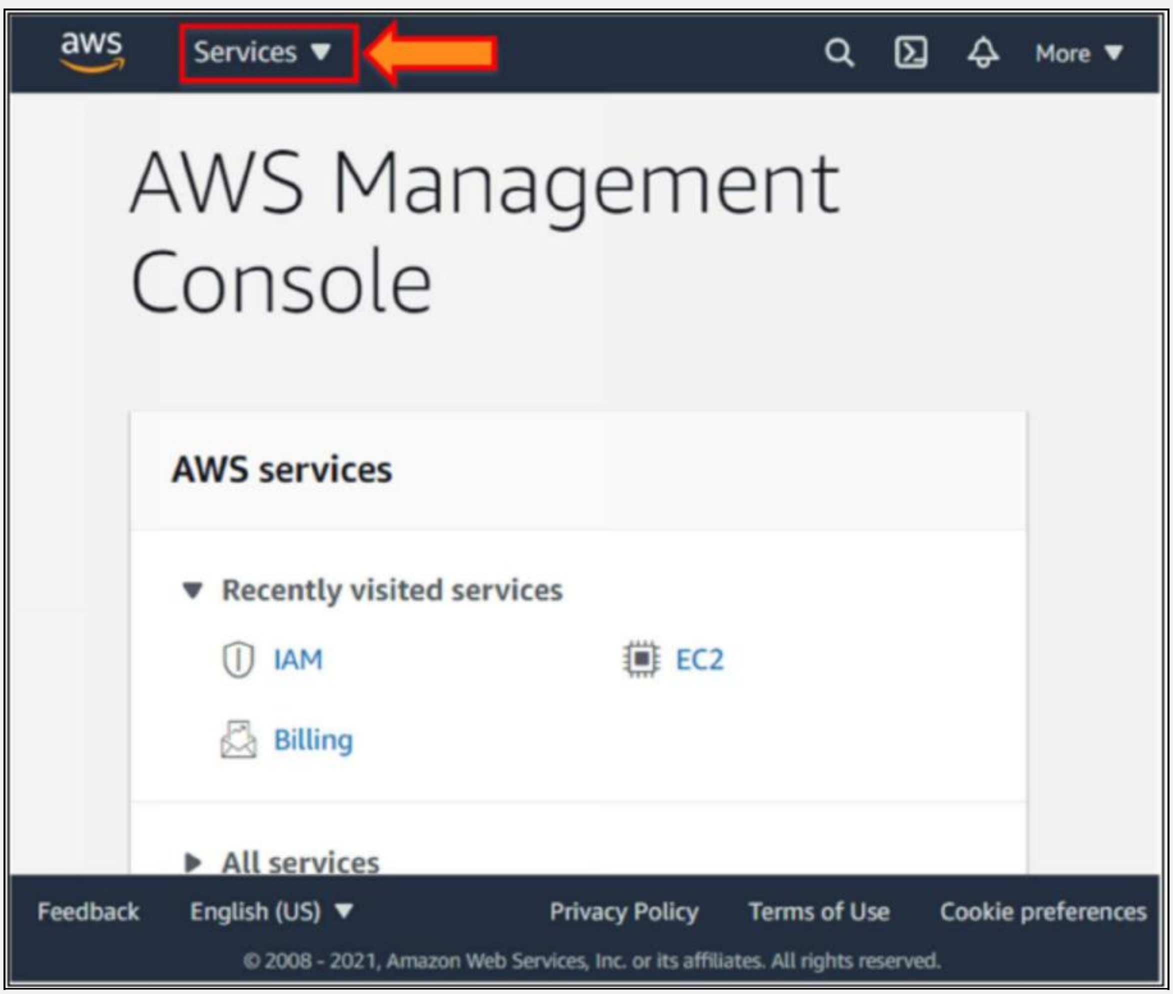
Solution

Step 1: Create a Lambda Function

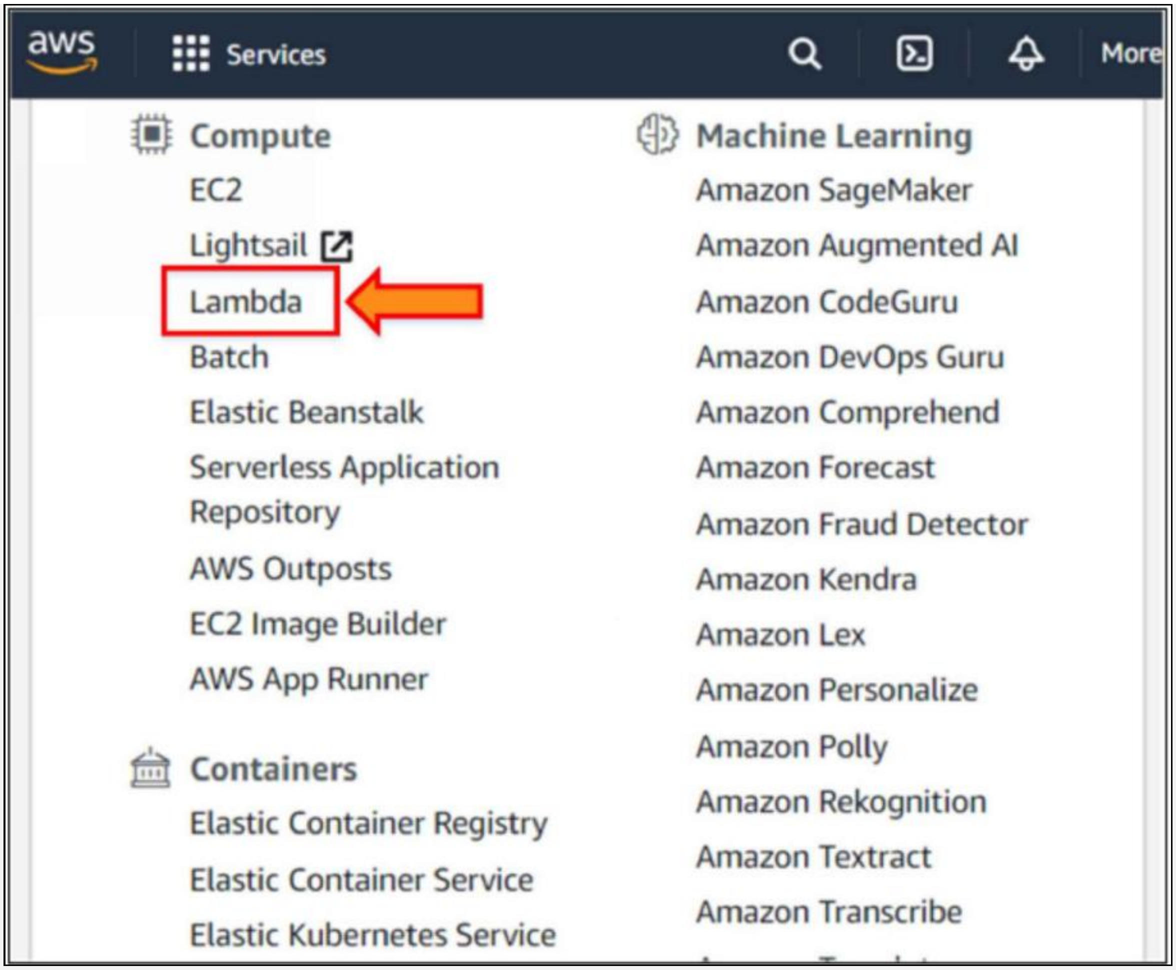
1. Log in to the **AWS Management Console**.



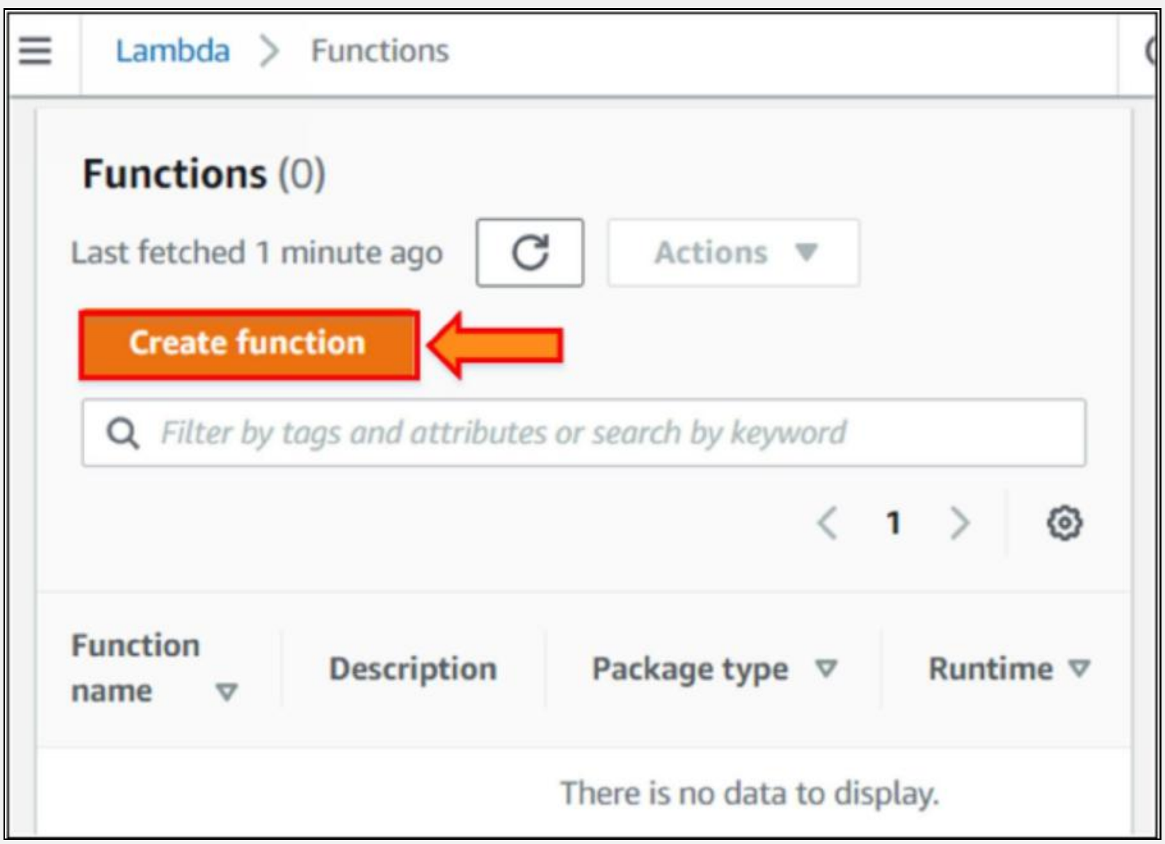
2. Go to **Services**.



3. Click on **Lambda**, which is present under **Compute**.

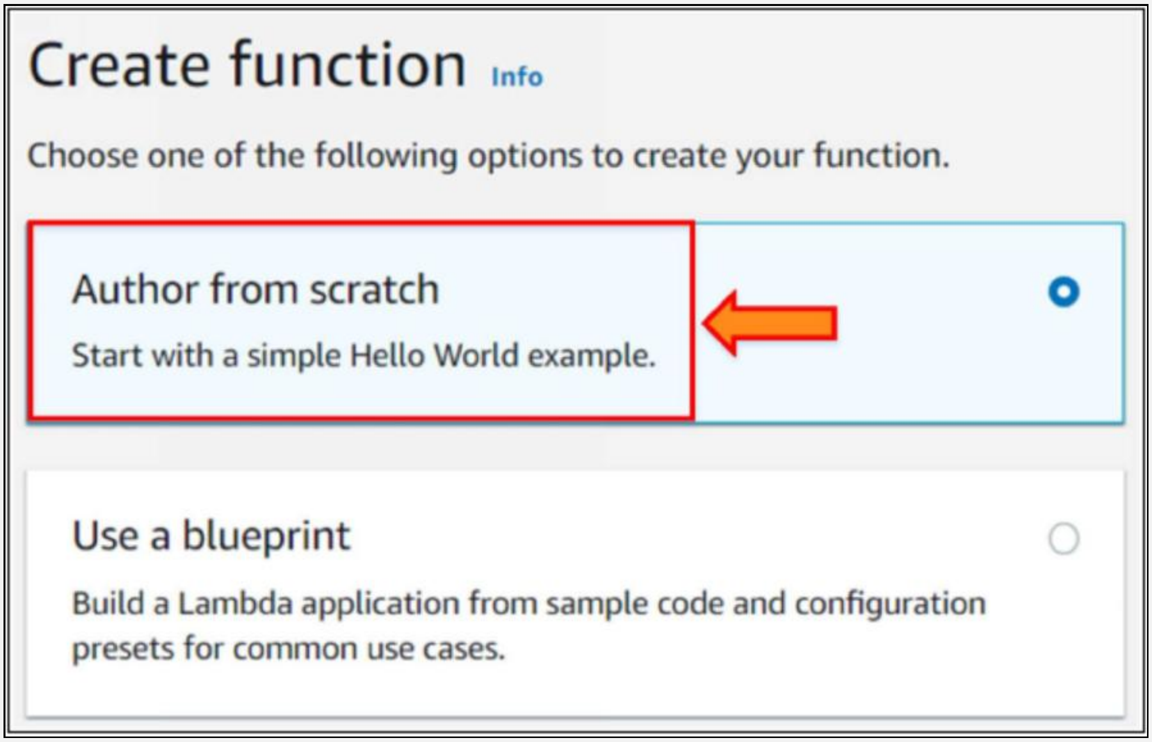


4. The screen of **AWS Lambda** will appear; click on **Create function**.

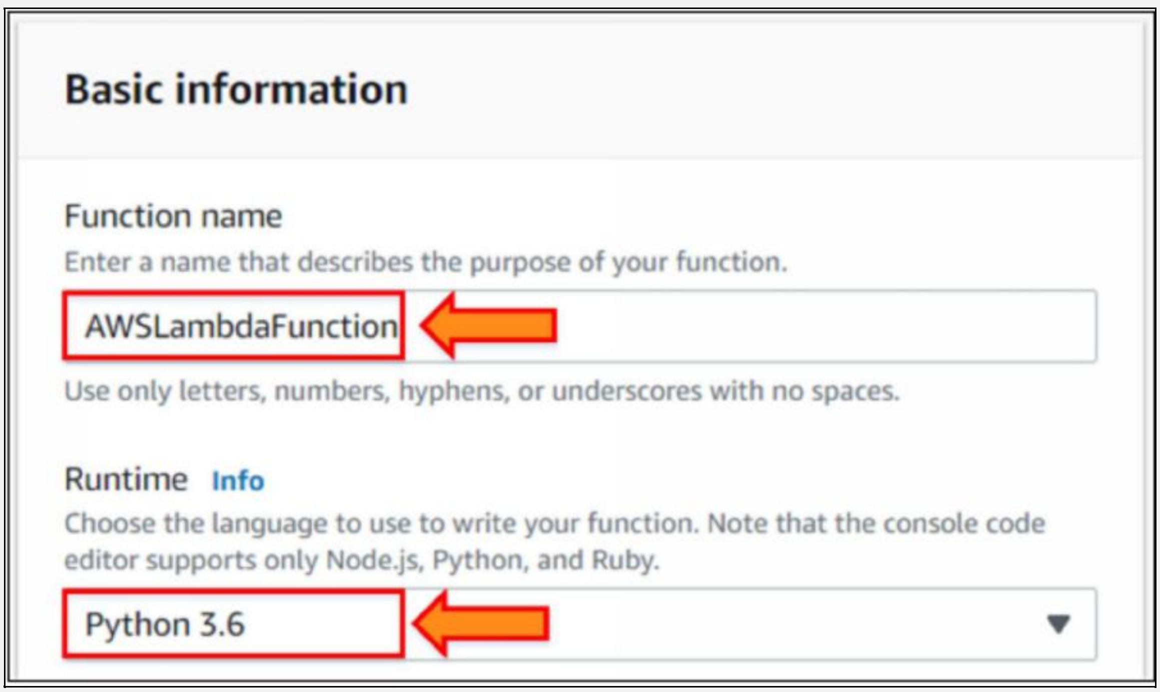


5. You can make **three** different **Lambda functions**. You can use the **author from scratch**, you can **use blueprint**, or you can use the **serverless application repository**.
6. In this solution, you will need to **Author from scratch** for creating the

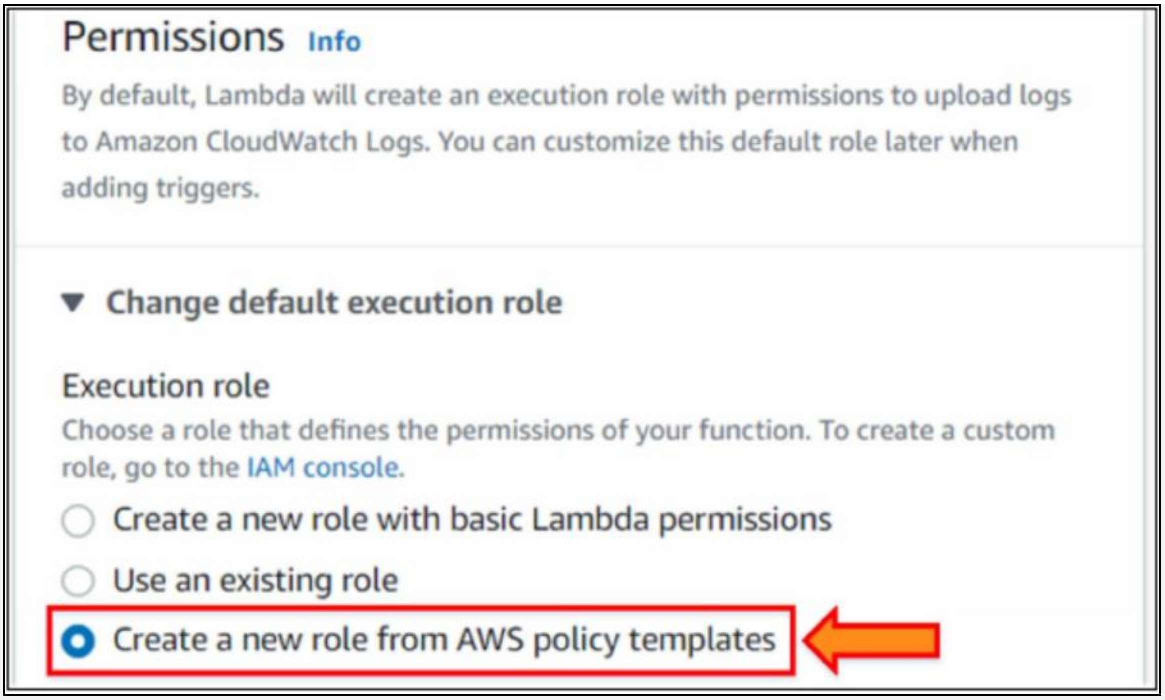
Lambda function.



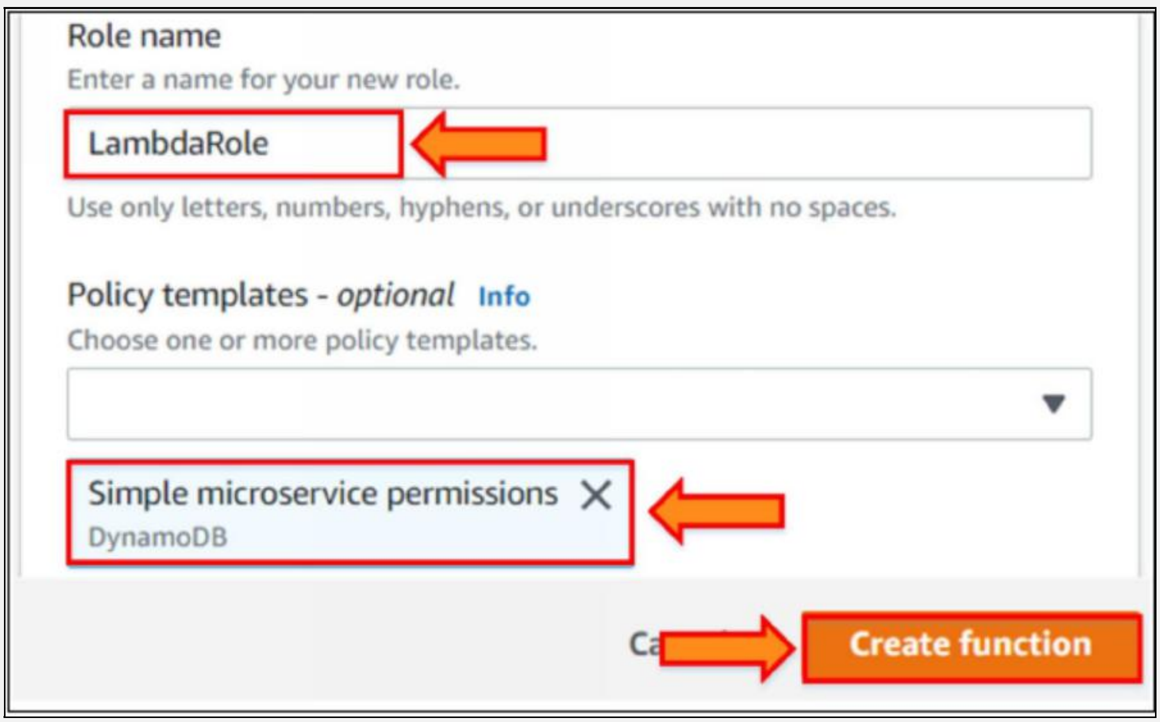
7. Write the **function name** as **AWSLambdaFunction**.
8. Select **Python 3.6** from the given list in **Runtime**.



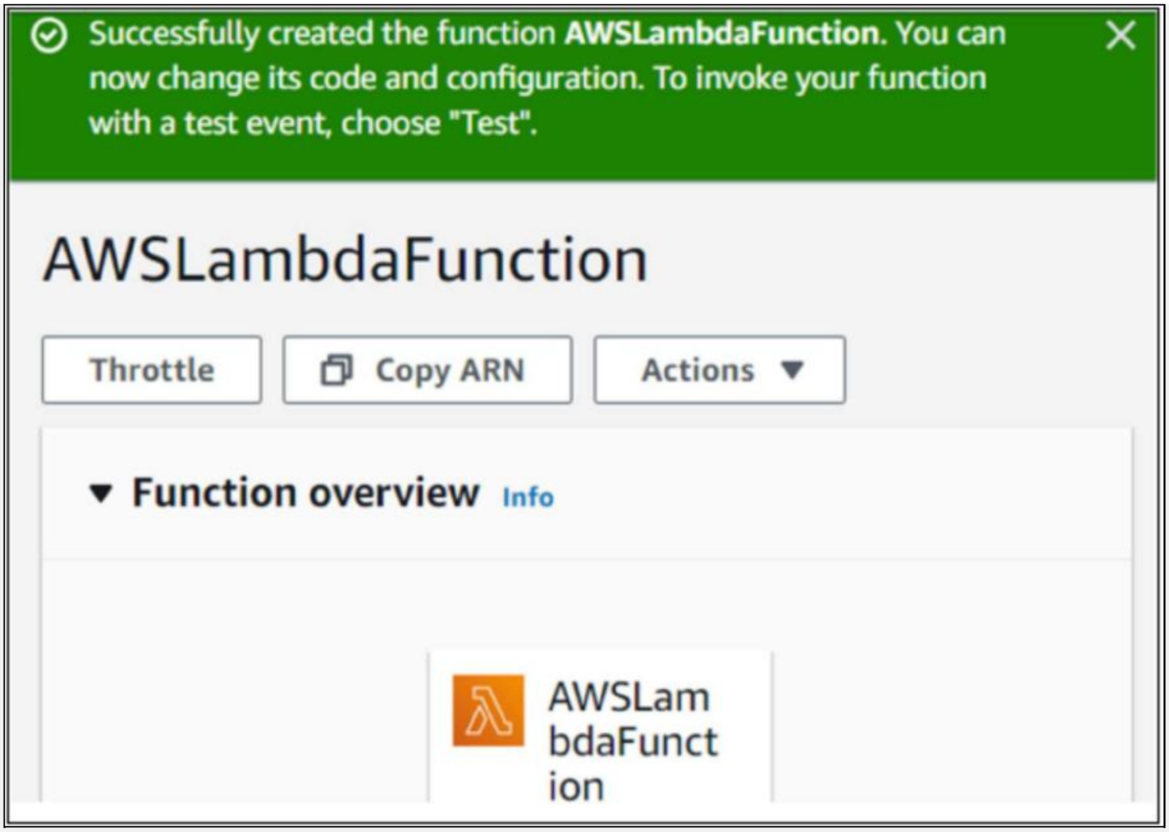
9. Select the execution role as **Create a new role from AWS policy templates**.



10. Give the name **LambdaRole** to the role.
11. Choose the policy template **Simple microservice permissions** from the given list.
12. After that, click on **Create function**.

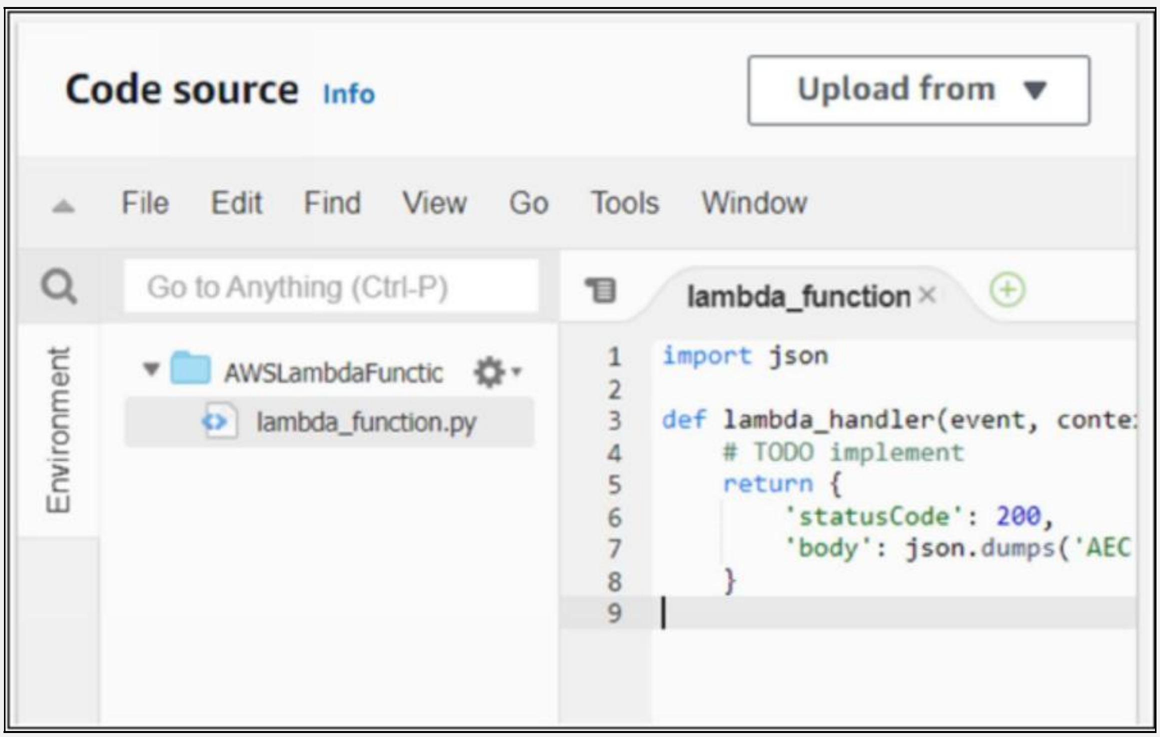


13. The **notification** of a **successful creation of Lambda function** will appear.



Step 2: Edit Code Source

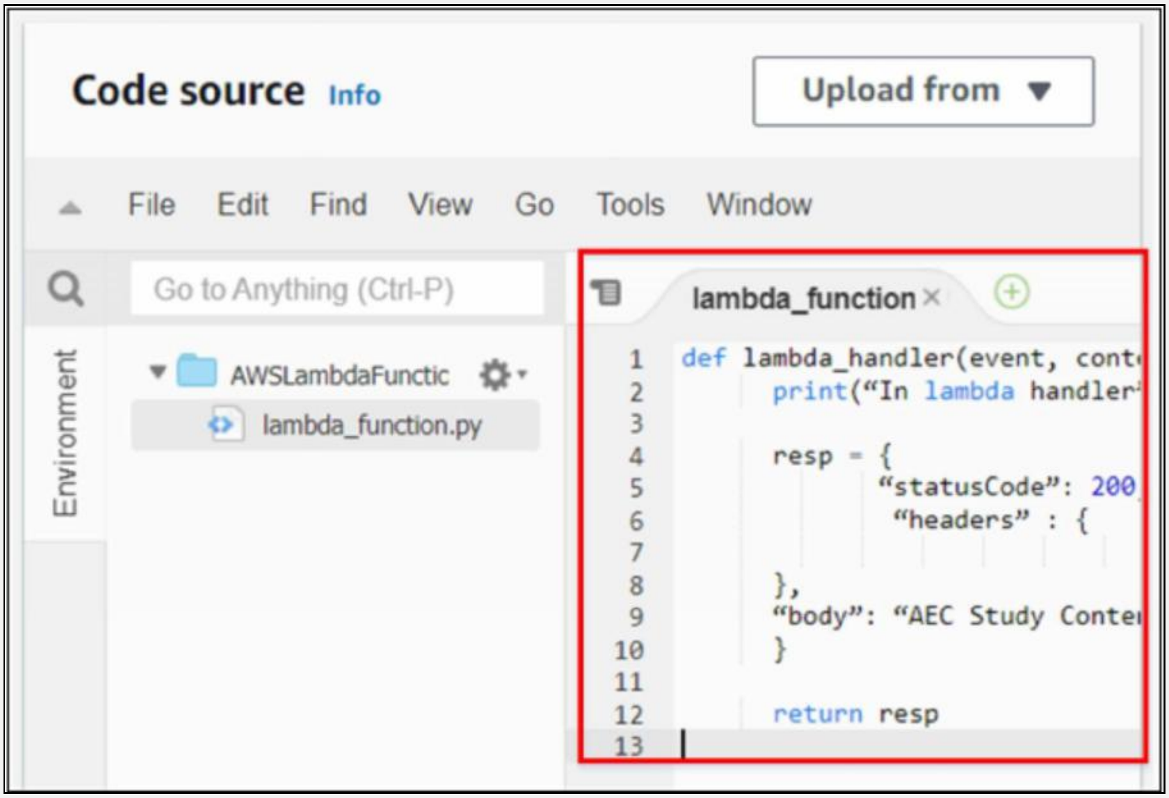
1. Go to the **Code source** section, and the default **coding** for the **Lambda function** is shown.



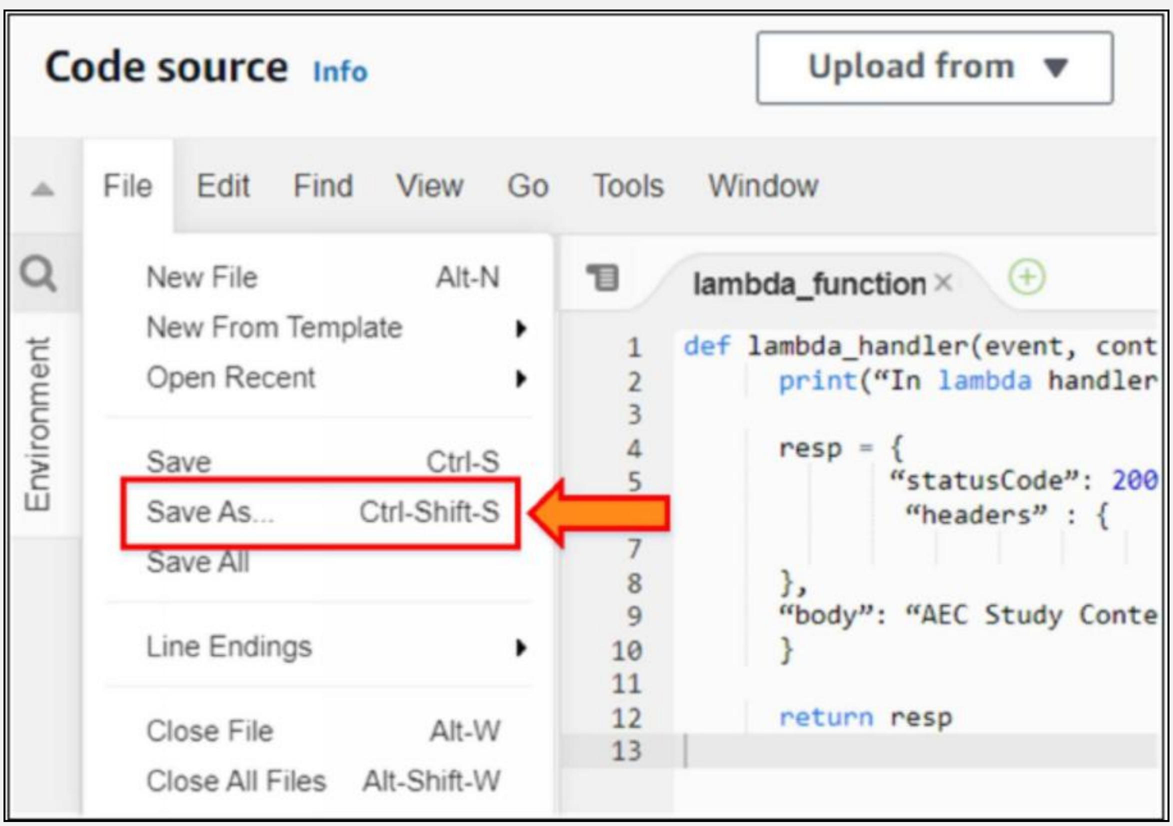
2. Go to your **resources section** and type the following **script**:

```
def lambda_handler(event, context)
    print(“In lambda handler”)

    resp = {
        “statusCode”: 200,
         “headers” : {
```

```
                    "Access-Control-Allow-Origin": "*",
    },
    "body": "IPS Study Content"
    }

    return resp
```

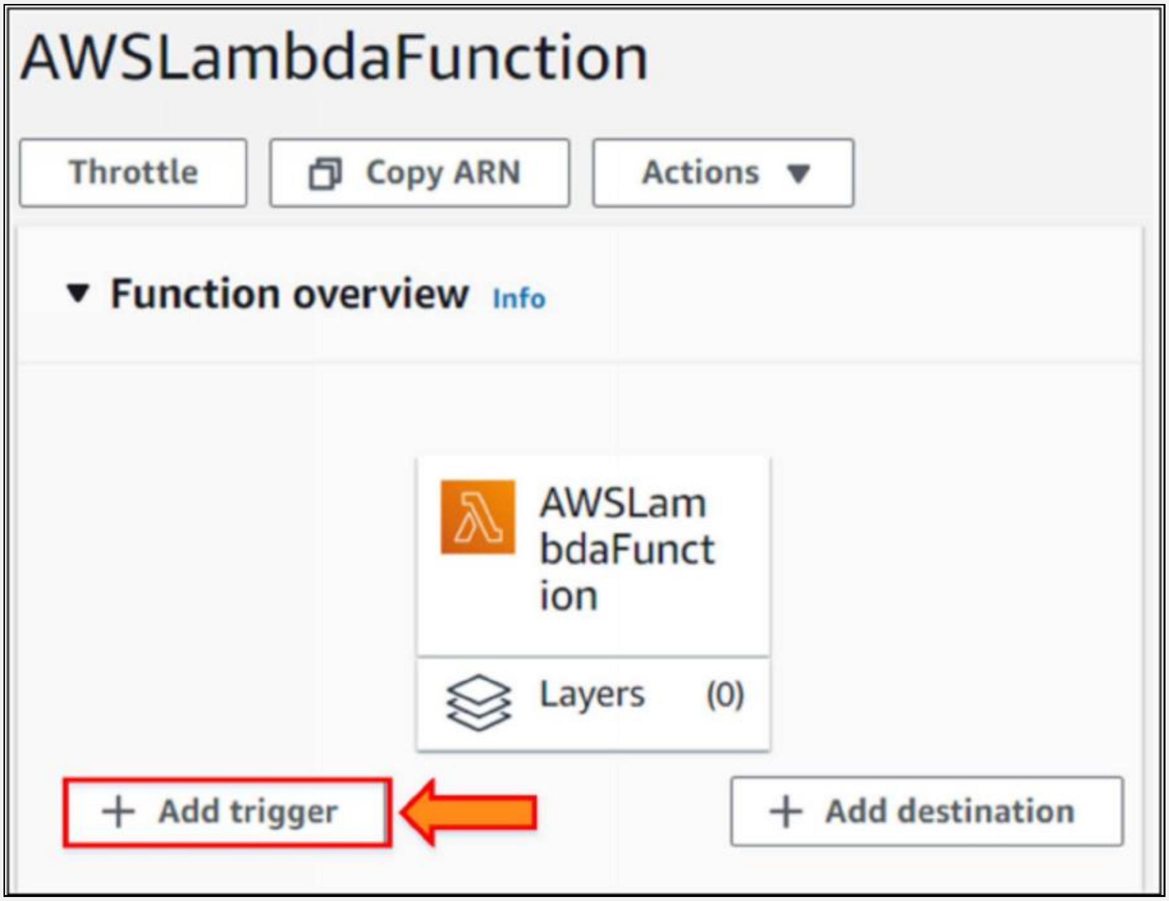


3. Go to **File**, and click on **Save As**.
4. Click on **Save**.

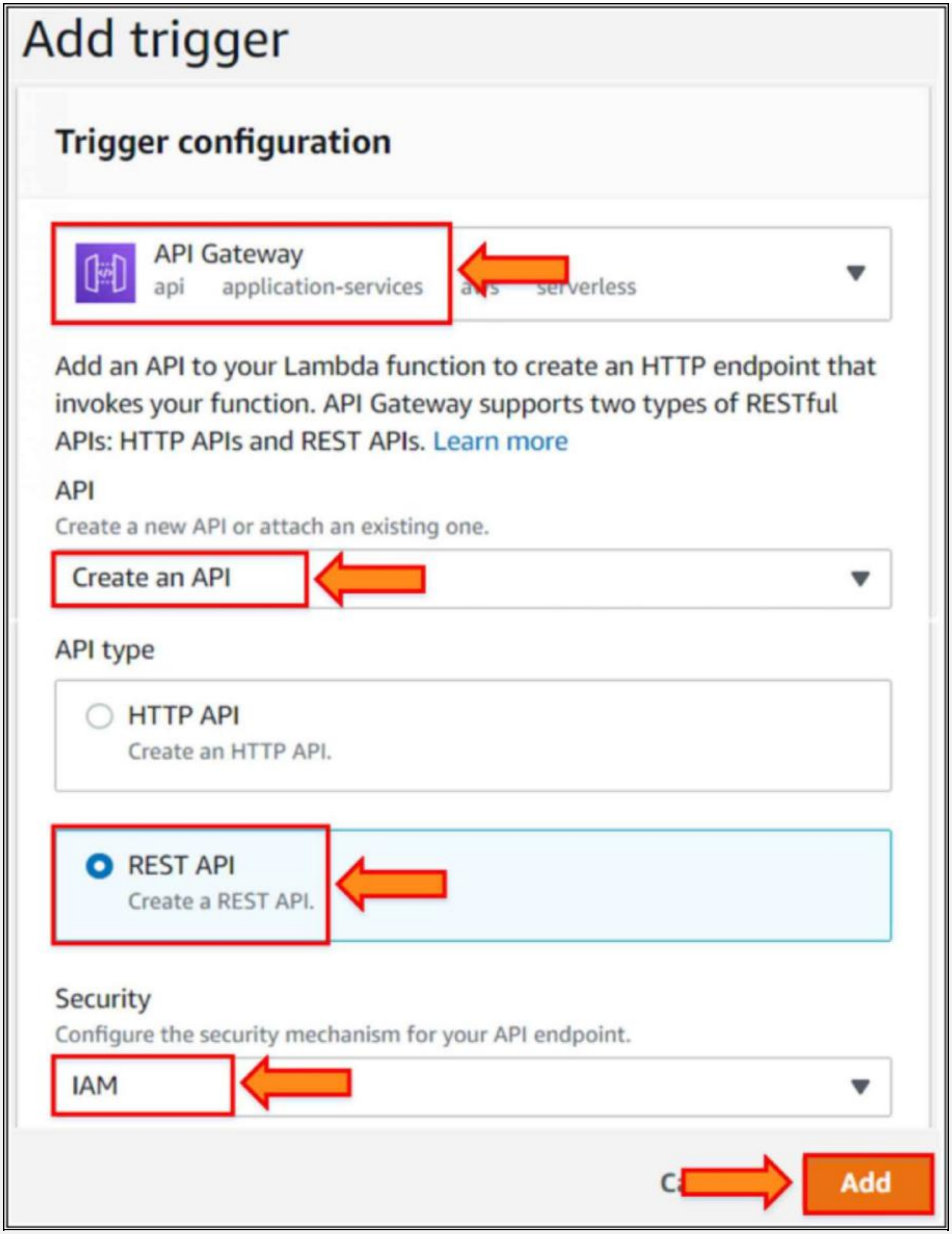


Step 3: Add Trigger

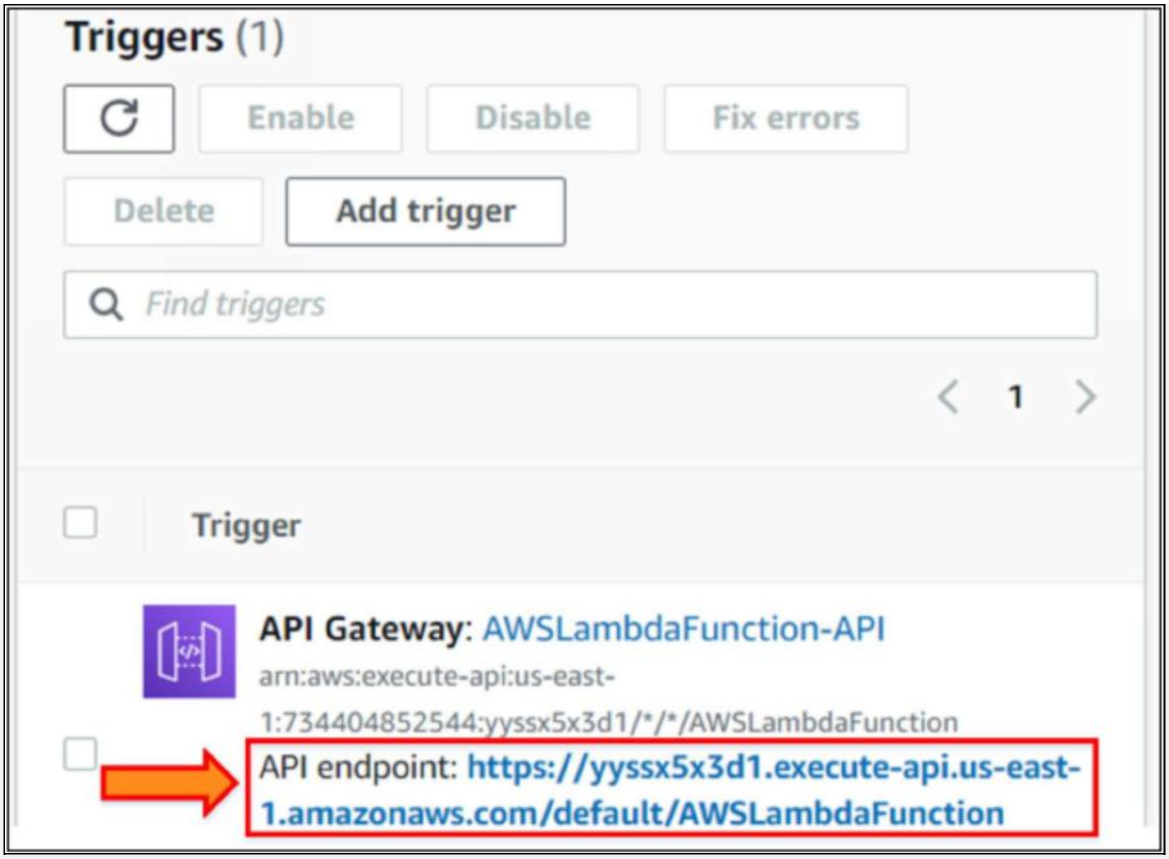
1. After that, go to **add a trigger**.
2. Click on **+ Add trigger**.



3. Select **API Gateway** from the given list.
4. Select **API type** as **REST API**.
5. Select **IAM** as a security mechanism for your API endpoint.
6. Leave the remaining options as default and click on **Add**.



7. The **API Gateway** as a **trigger** is shown. Click on the given **link**.

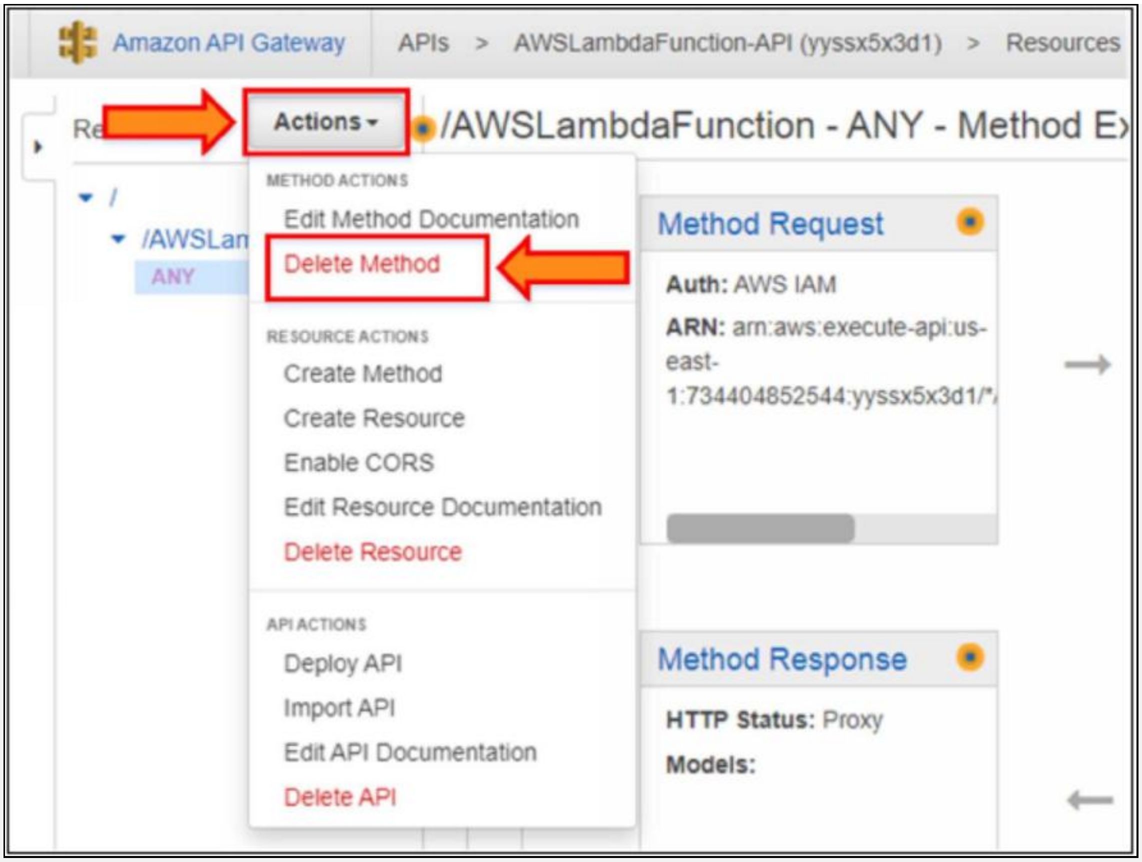


8. You will get an **error message**.

9. Now, go back and click on **AWSLambdaFunction-API**.
10. The API Gateway is shown in the form of a **User Interface**.

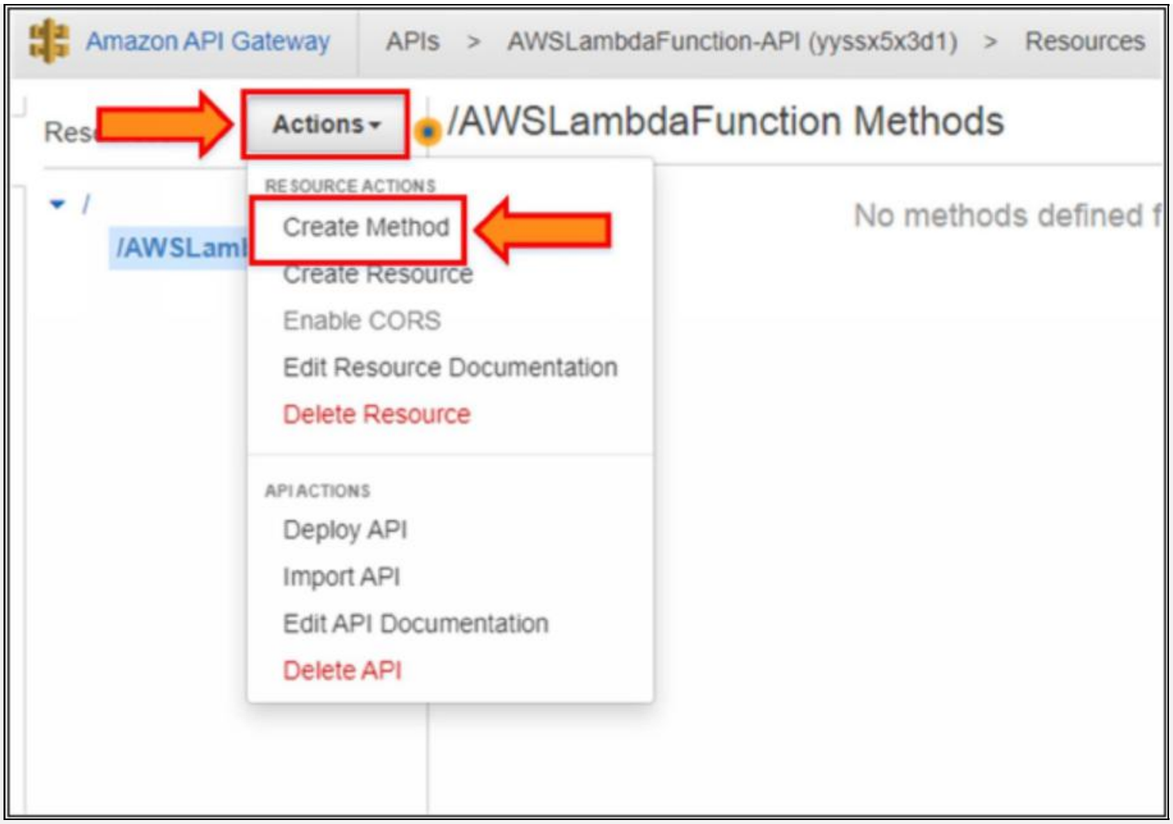
Note: The API Gateway is shown in the form of a User Interface (UI). The client is present on one side, and the Lambda function is present on another. The client sends a request of type “ANY.” The request proxies to the lambda function, and then the lambda function responds to the client.

11. Go to **Actions** and click on **Delete Method**.



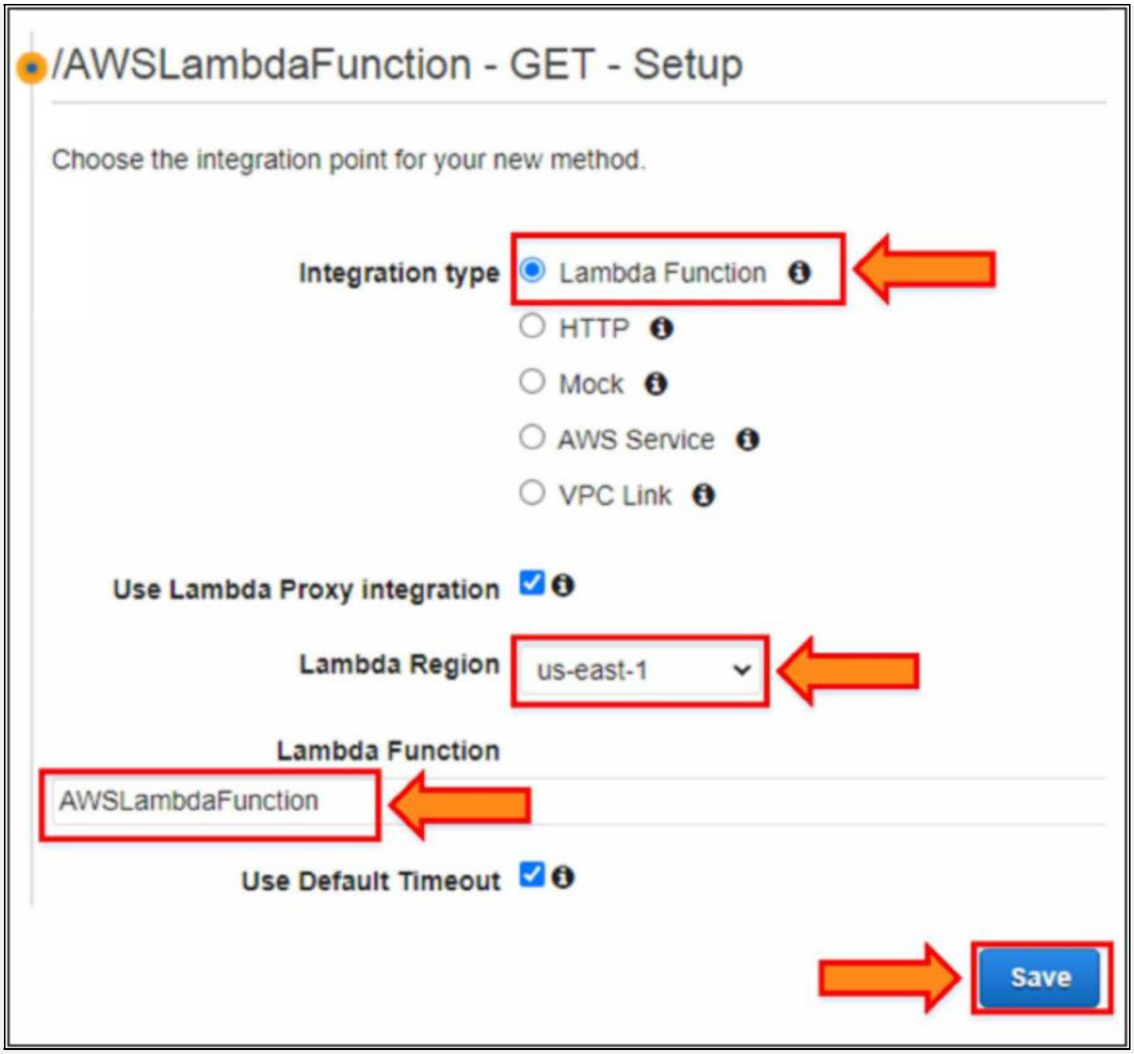
12. A dialog box will appear. Click on **Delete**.

13. Go to **Actions** and click on **Create Method** to **create a new method.**



14. Select the function type **GET** from the given list.

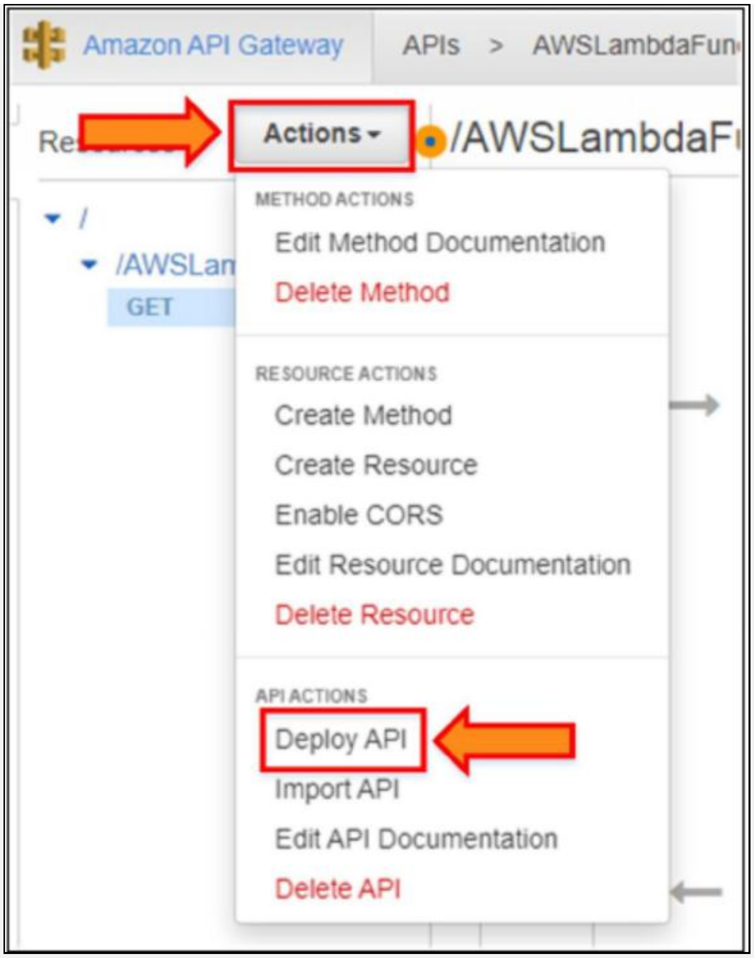
15. Select the **Integration type** as **Lambda Function**.
16. Enable **Use Lambda Proxy Integration**.
17. Select the **N.Virginia us-east-1** region.
18. Give the name of **Lambda Function** as **AWSLambdaFunction**.
19. Enable **Use Default Timeout**.
20. Click on **Save**.



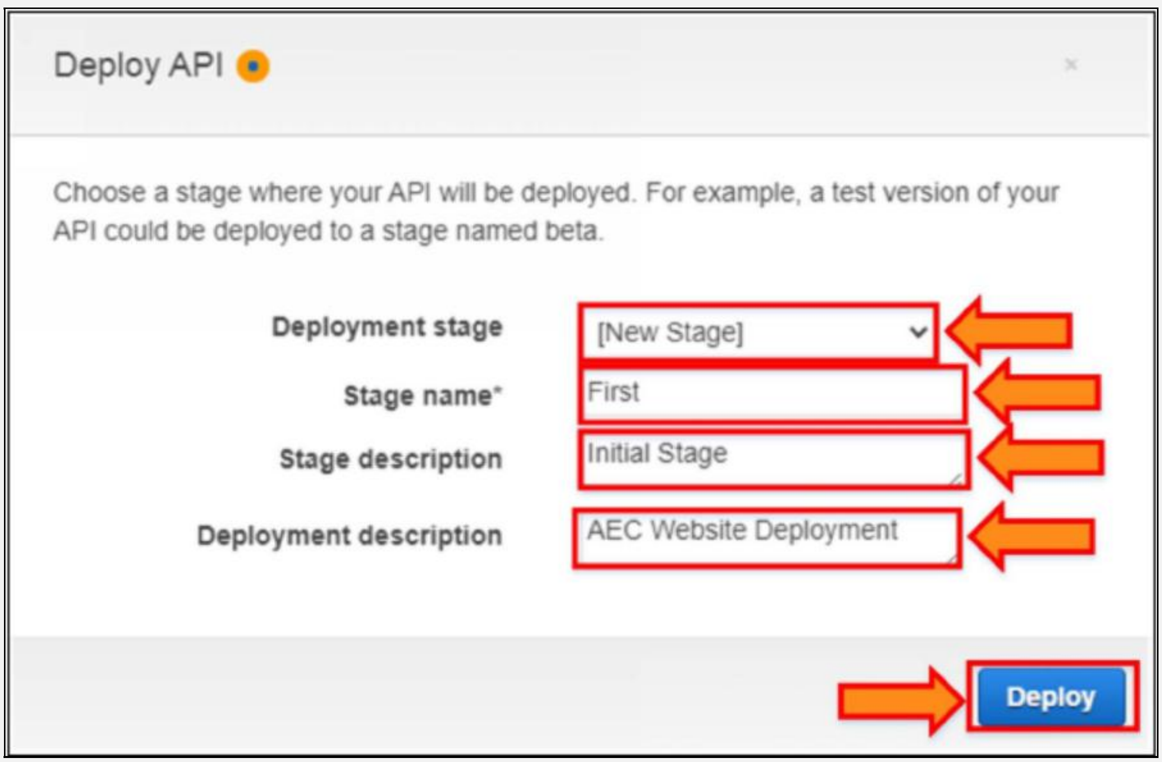
21. Click on **OK**.

Step 4: Deploy API

1. Now to **Deploy API**, go to **Actions**.
2. Click on **Deploy API**.

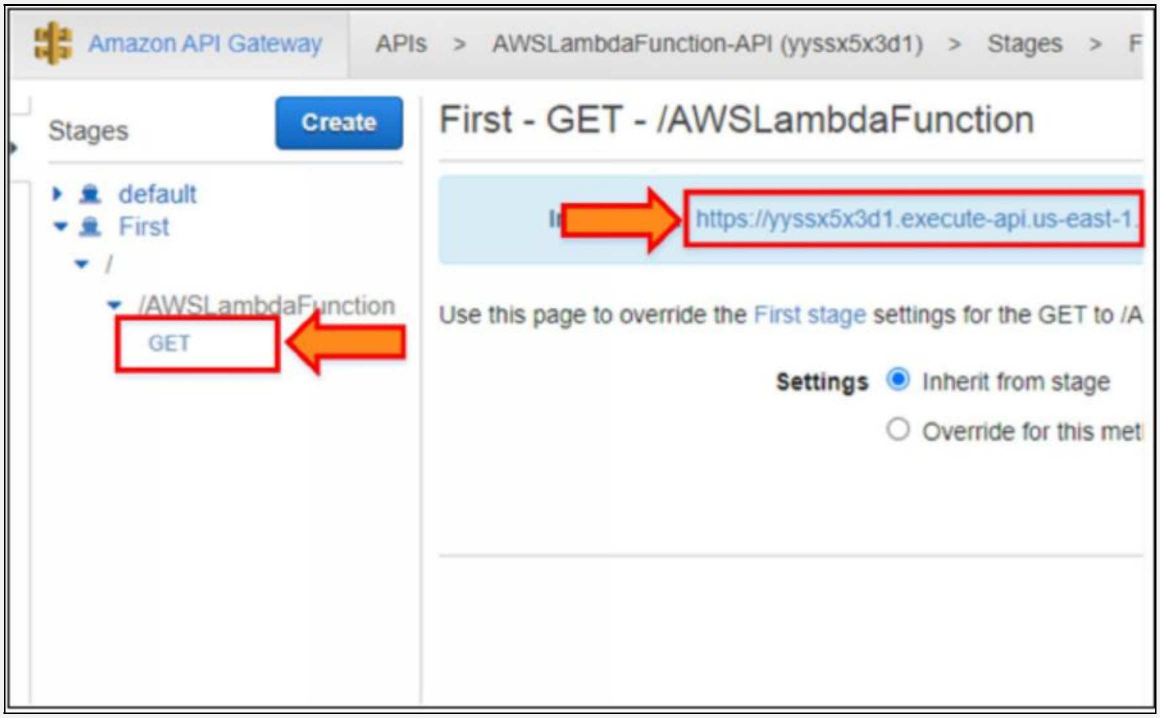


3. Select the **Deployment stage**.
4. Enter **Stage name**.
5. Enter **Stage description**.
6. Write the **Deployment description** as **AEC Website Deployment**.
7. Click on **Deploy**.

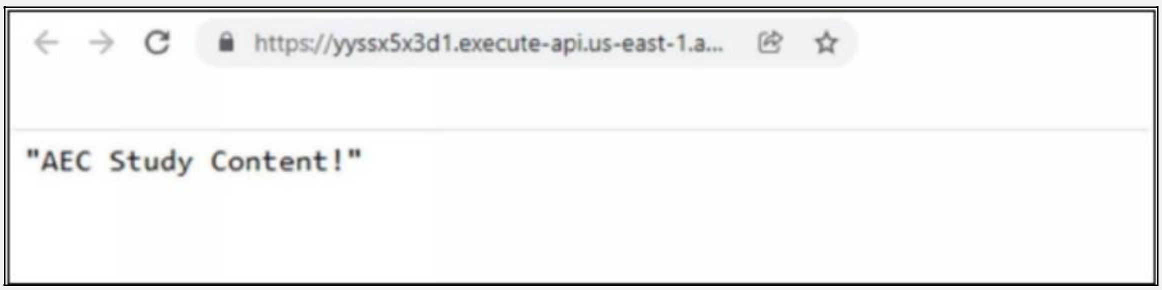


8. After **deployment of API**, click on **GET**.
9. The deployed API is shown in the figure. Click on this to **Invoke the**

URL.



10. The new tab opens with **AEC Study Content**.



Lab 04: Streaming Data Collection

Certifications	AWS Machine Learning - Specialty
Cloud Provider	AWS
Time Required	1 Hr.
Tags	Lambda, S3, AWS

Lab Prerequisites

- Familiarity with basic AWS Cloud Computing concepts and terminology
- An AWS account with an active subscription

Service Introduction

Kinesis Data Streams

You may use Amazon Kinesis Data Streams to create custom applications that process or analyze streaming data for specific purposes. To an Amazon Kinesis data stream, you may constantly add data from hundreds of thousands of sources, such as clickstreams, application logs, and social media. Within seconds, your Amazon Kinesis Applications will be able to read and analyze data from the stream.

Kinesis Data Analytics

Amazon Kinesis Data Analytics is the most straightforward method for converting and analyzing real-time streaming data using Apache Flink. Apache Flink is an open-source data stream processing framework and engine. Apache Flink applications' creation, maintenance, and integration with other AWS services are made easier using Amazon Kinesis Data Analytics.

Amazon Kinesis Data Analytics handles everything needed to constantly operate streaming applications and grows automatically to meet the amount and throughput of your incoming data. There are no servers to operate with Amazon Kinesis Data Analytics, no minimum charge or setup cost, and you pay only for the resources your streaming applications utilize.

Kinesis Data Firehose

Kinesis Data Firehose is a solution for streaming ETL (Extract, Transform, and Load). It is the most convenient method for loading streaming data into data storage and analytics tools. It can collect, transform, and load streaming data into Amazon S3, Amazon Redshift, Amazon Elasticsearch Service, and Splunk, enabling near-real-time analytics with your existing business intelligence tools and dashboards. It is a fully managed service that scales automatically to meet your data flow and does not require any ongoing management. It may also batch, compress, and encrypt data before loading it, reducing storage requirements at the destination and enhancing security.

AWS Lambda

AWS Lambda allows you to run code without creating or managing servers. There is no charge when your code is not executing; you only pay for the compute time you use. You can run code for nearly any application or backend service with Lambda, and you do not have to worry about administration. Upload your code, and Lambda will handle everything necessary to run and grow it with high availability. You may configure your code to be automatically triggered by other AWS services, or you can access it directly from any computer or smartphone app.

Amazon Simple Storage Service S3

Amazon S3 is a type of object storage that allows you to store and recover any quantity of data from any location. It is a low-cost storage solution with business resilience, reliability, efficiency, privacy, and infinite expansion.

Amazon S3 is a web service that allows you to store and retrieve an infinite quantity of data from any place and at any time. Using this service, you may quickly create projects that integrate cloud-native storage because Amazon S3 is easily customizable. You only pay for what you use; you can start small and scale up as needed without sacrificing performance or dependability.

Amazon S3 is also built to be highly adaptable. Instead of finding out how to store their data, Amazon S3 allows developers to focus on innovation. Build a simple FTP system or a complex web application like the Amazon.com retail website; read the same piece of data a million times or only for emergency disaster recovery; store whatever type and amount of data you desire.

AWS CloudFormation

AWS CloudFormation is a tool that makes it simple for developers and organizations to construct a collection of linked AWS and third-party resources and then provision and manage them logically and reasonably.

Developers may use a simple, declarative approach to deploy and change compute, database, and many other resources, abstracting away the complexities of individual resource APIs. AWS CloudFormation is meant to make resource lifecycle management repeatable, predictable, and safe, with features such as automatic rollbacks, automated state management, and resource management across accounts and regions. Multiple ways to generate resources have recently been added, including leveraging AWS CDK for writing in higher-level languages, importing existing resources, and detecting configuration drift. A new registry simplifies the creation of custom types that inherit several of CloudFormation's fundamental features.

AWS Identity and Access Management (IAM)

Individuals and groups can be granted secure access to your AWS resources by using IAM. It allows you to create and manage IAM users and provide them access to your resources. Additionally, you have the option of granting access to users outside of AWS (federated users).

- **Managed Policy:** This contains the permission required to stop an EC2 instance
- **Inline Policy:** This allows this role to be passed to another service
- **Trust Policy:** Allows System Manager and EC2 to assume the role. It enables EC2 to register with the Systems Manager and Systems Manager to stop the EC2 instance

Case Study Enterprise Biopharmaceutical – Neumora Therapeutics

Background

Neumora Therapeutics is a clinical-stage biopharmaceutical company leveraging advanced data science and artificial intelligence to develop targeted treatments for neurobehavioral disorders.

Business Challenge

Neumora Therapeutics has thousands of users interacting with the organization application. The organization gives Neumora Therapeutics a task to capture real-time user data for a marketing campaign. For users aged 21 and up, it must collect information such as name, age, gender, and location.

Proposed Solution

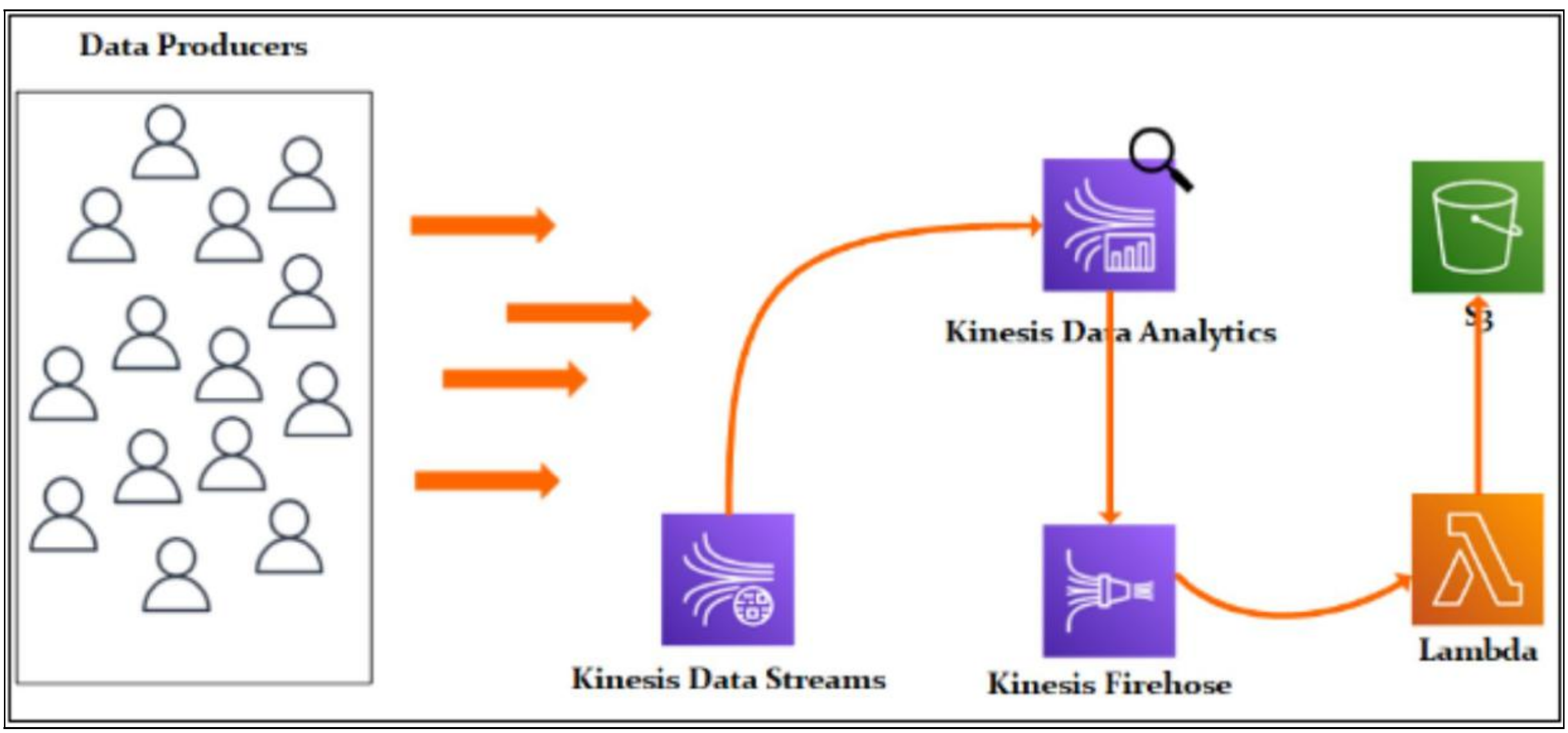
The solution is simple! You use AWS services to automate your work. You use Kinesis Data Streams to generate or collect data. Then, you use Kinesis Data Analytics to analyze data by running SQL queries. For storing the real-time analyzing data, you use Kinesis Data Firehose to store data at any storage destination. The other helping service you can use is AWS Lambda to transform data into proper JSON format. An S3 bucket is used to store that transformed data.

Before starting the lab, a Python script is used to collect the data. The Python code is provided in the following GitHub link (Chapter-3) https://github.com/12920/IPSpecialist01/blob/main/Course_AWS_Certified_Machine_Learning-master%20(1).zip

This Python script is integrated into the CloudFormation stack YAML file. In this lab, only a template is used to create a stack and collect the data.

```
import requests
import boto3
import uuid
import time
import random
import json

client = boto3.client('kinesis', region_name='<INSERT_YOUR_REGION>')
partition_key = str(uuid.uuid4())

# Added 08/2020 since randomuser.me is starting to throttle API calls
# The following code loads 500 random users into memory
number_of_results = 500
r = requests.get('https://randomuser.me/api/?exc=login&results=' + str(number_of_results))
data = r.json()["results"]

while True:
        # The following chooses a random user from the 500 random users pulled from the API in a single API call.
        random_user_index = int(random.uniform(0, (number_of_results - 1)))
        random_user = data[random_user_index]
        random_user = json.dumps(data[random_user_index])
        client.put_record(
                StreamName='<INSERT_YOUR_STREAM_NAME>',
                Data=random_user,
                PartitionKey=partition_key)
        time.sleep(random.uniform(0, 1))
```

Lab Diagram



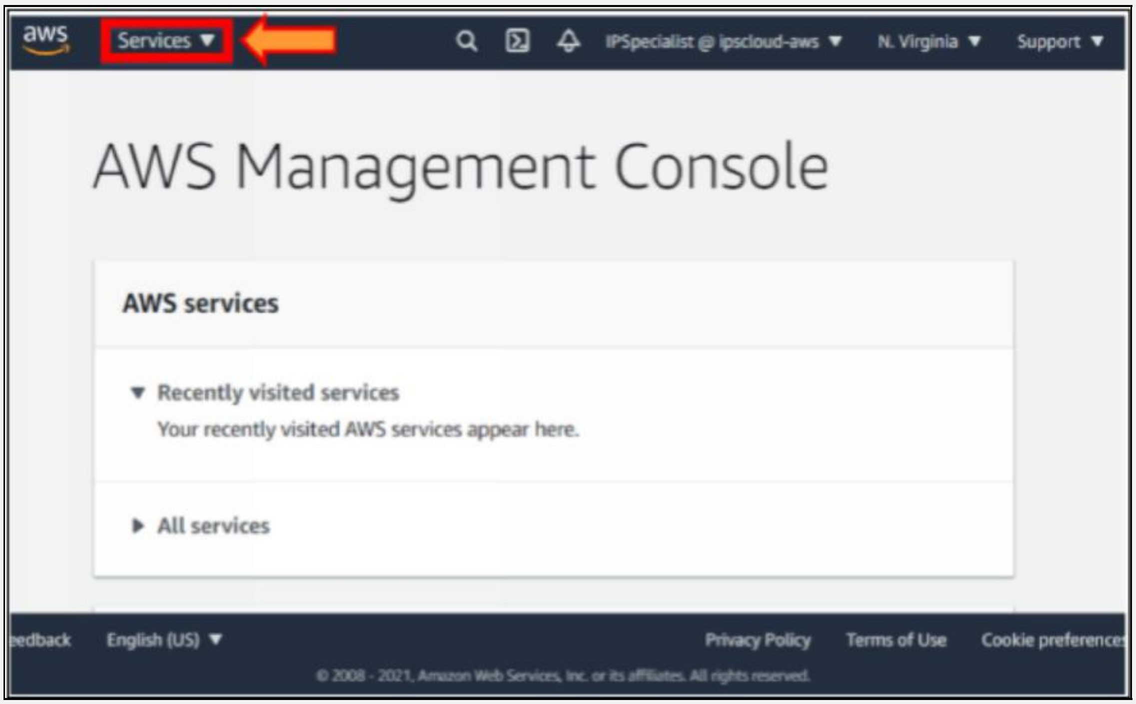
<u>Implementation Steps</u>

1. Create Kinesis Data Stream.
2. Create CloudFormation Stack.
3. Create Kinesis Data Analytics.
4. Create Kinesis Data Firehose.
5. View Collected Data.

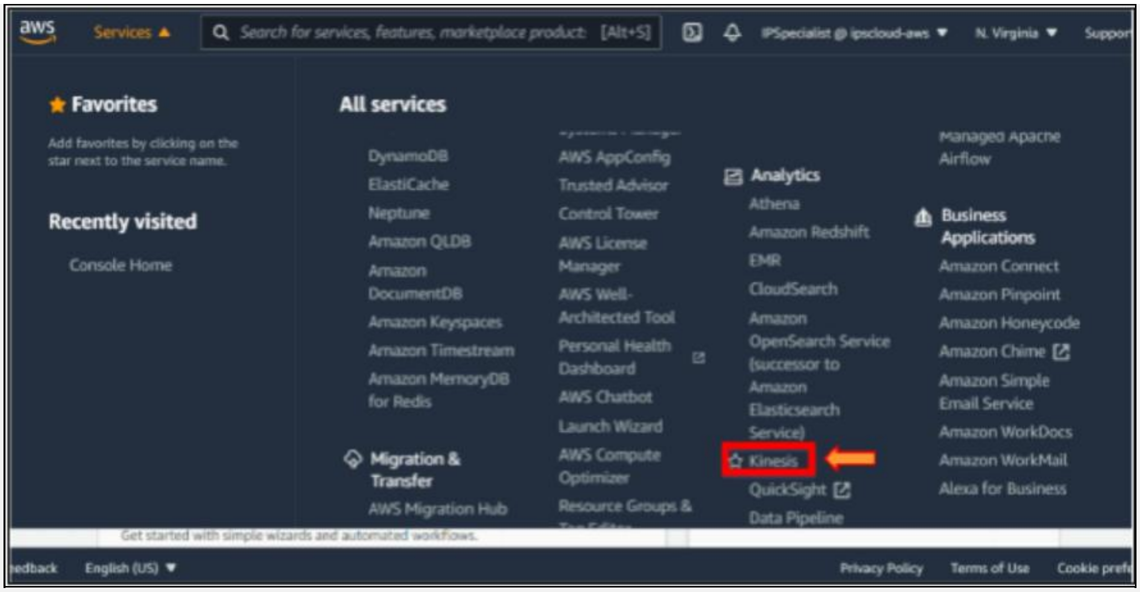
Solution

Step 1: Create Kinesis Data Stream

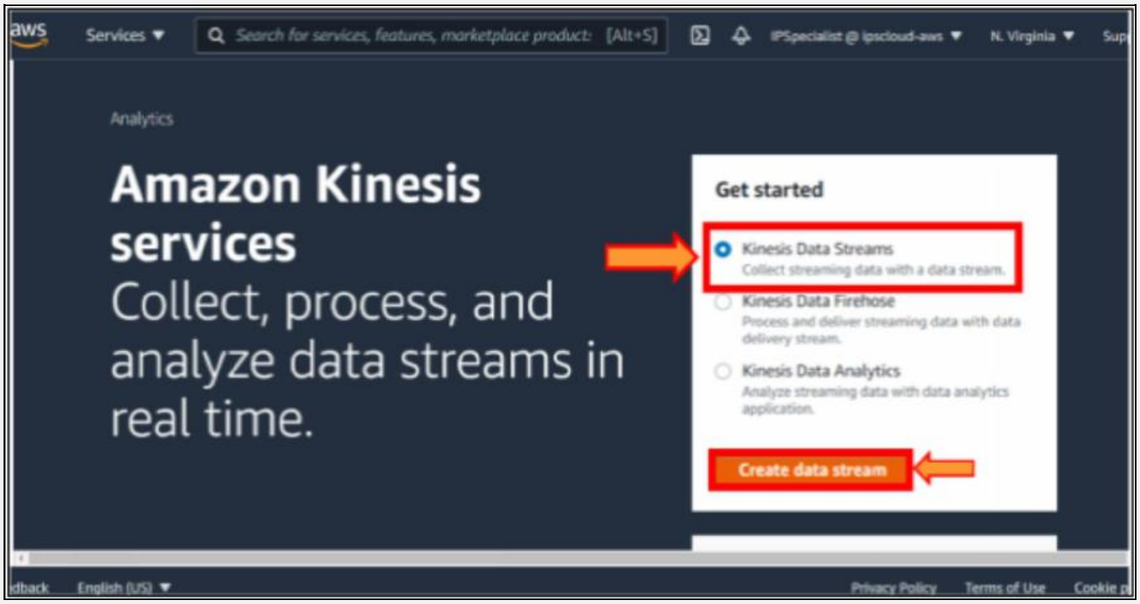
1. Log in to the **AWS Console.**
2. Click on **Services.**



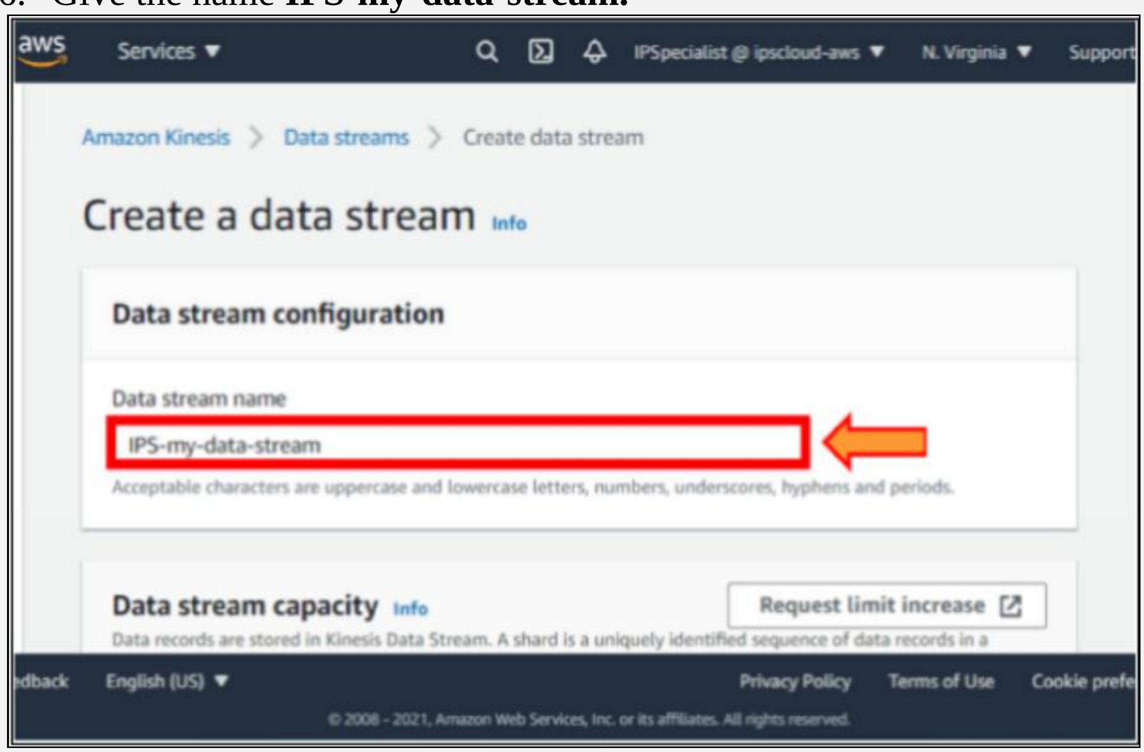
3. Select **Kinesis** from **Analytics.**



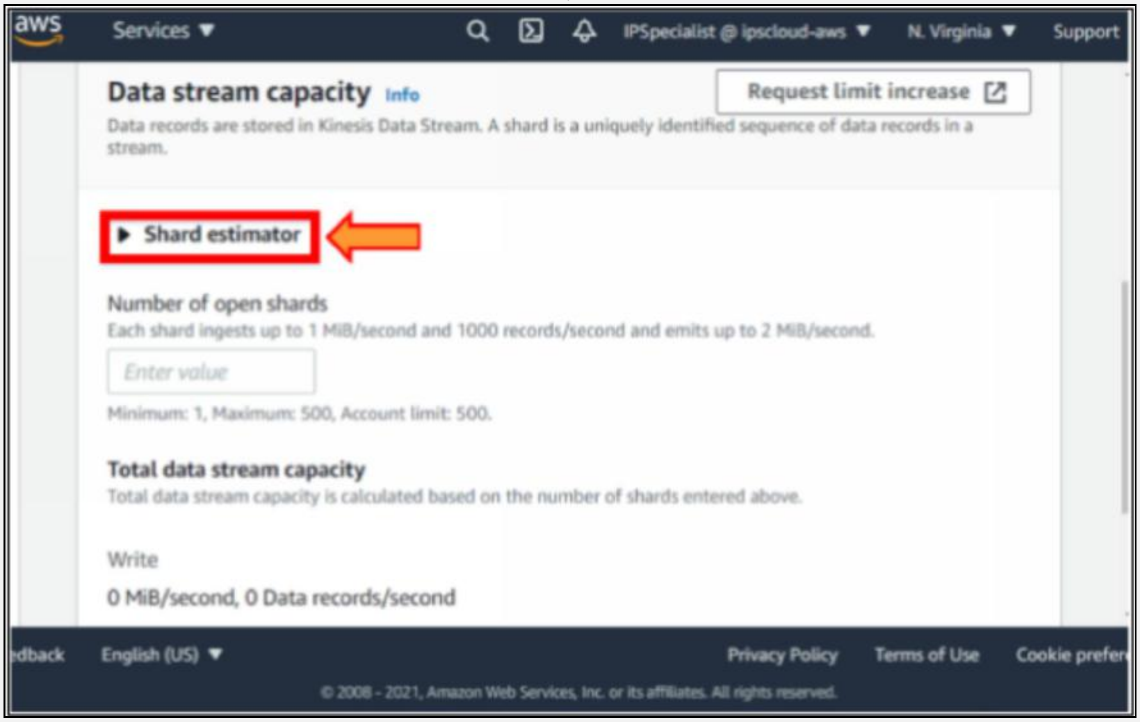
4. Select **Kinesis Data Streams**.
5. Click on the **Create Data Stream** button.



6. Give the name **IPS-my-data-stream.**

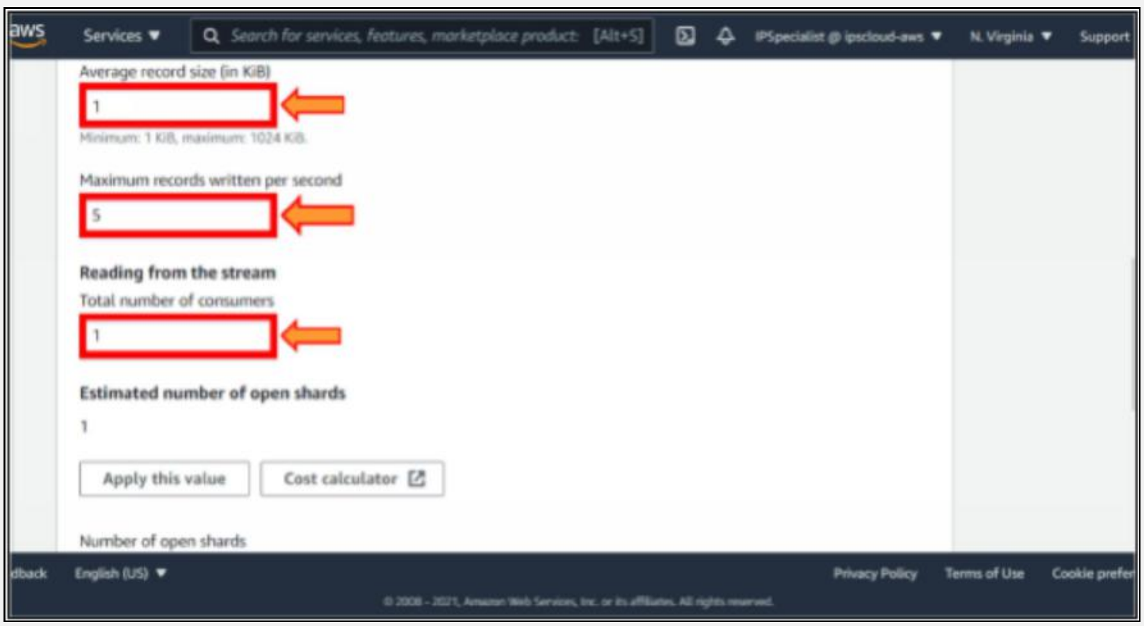


7. To calculate the number of shards, click on the **Shard Estimate.**

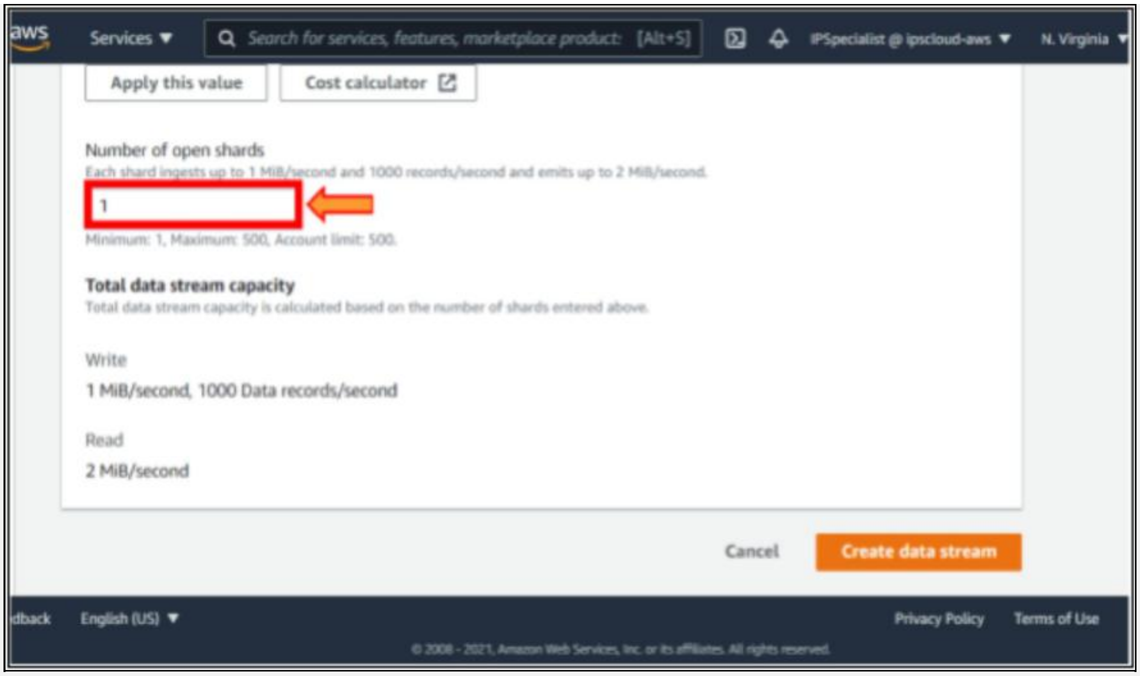


8. Give the following values in **Shard Calculator**.

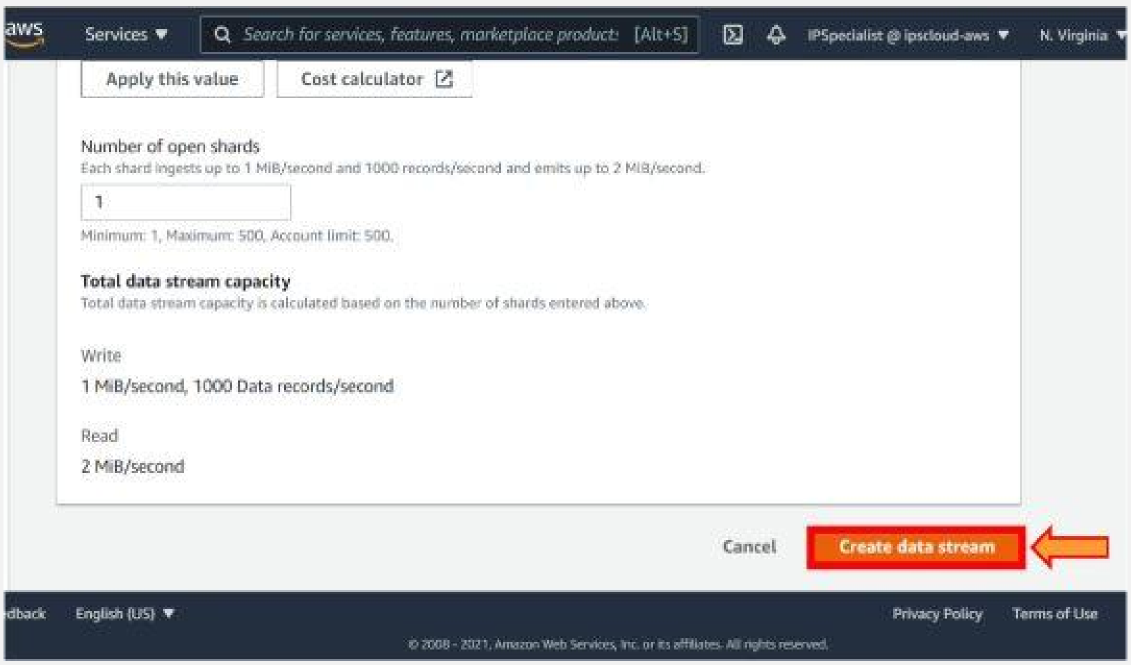
- Average record size: **1** KB
- Max records are written: **5** per second
- Number of consumer applications: **1**
- The result is Estimated shards: **1**



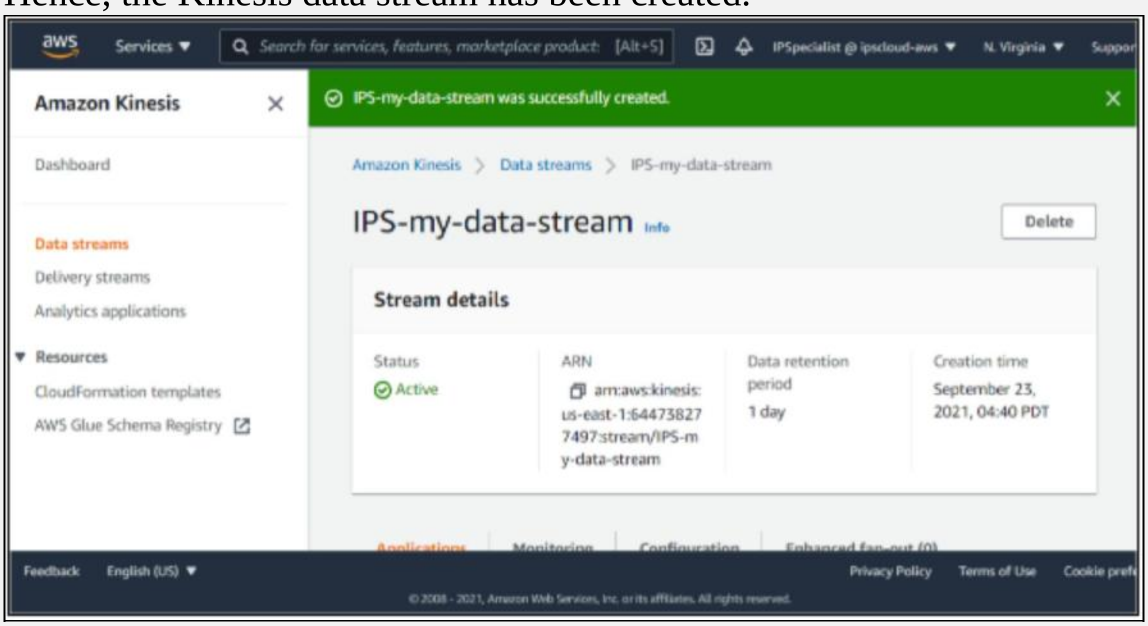
- Set number of shards: **1**



9. Click on the **Create Data Stream** button.

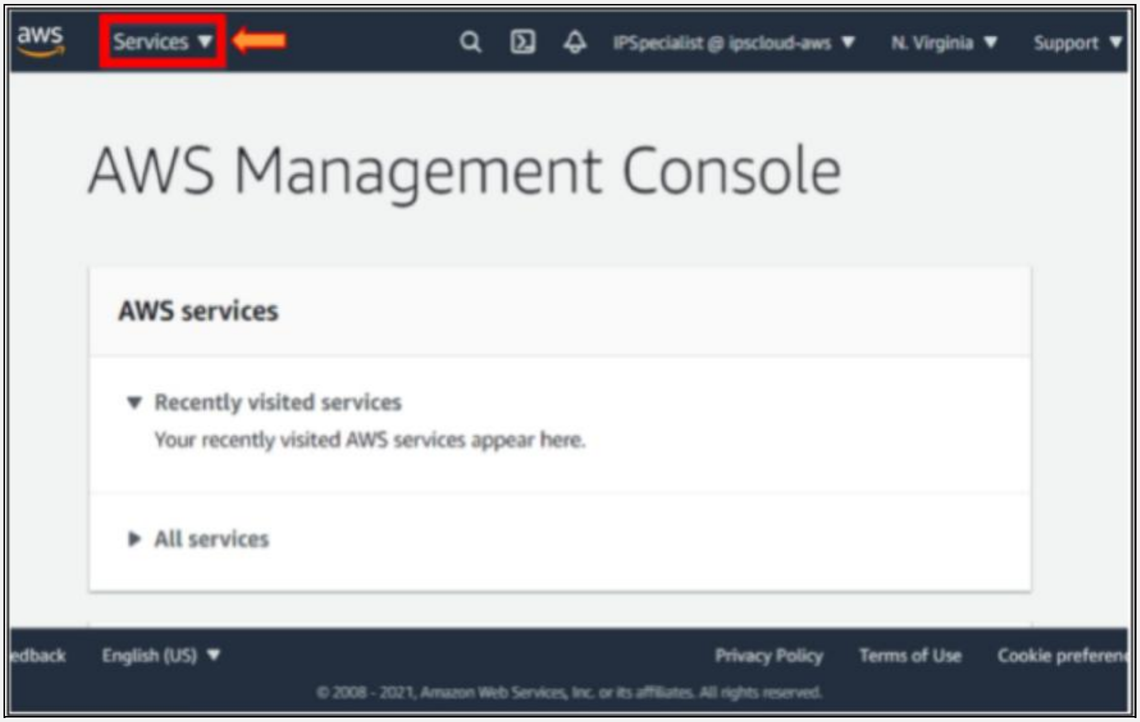


Hence, the Kinesis data stream has been created.

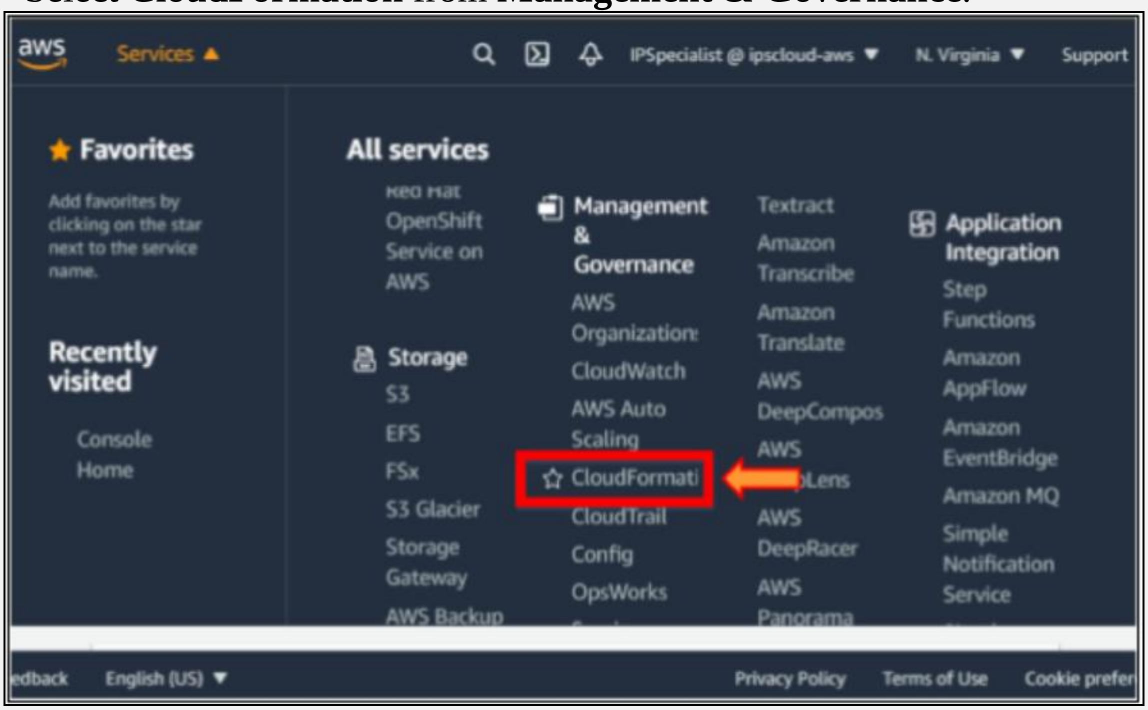


Step 2: Create CloudFormation Stack

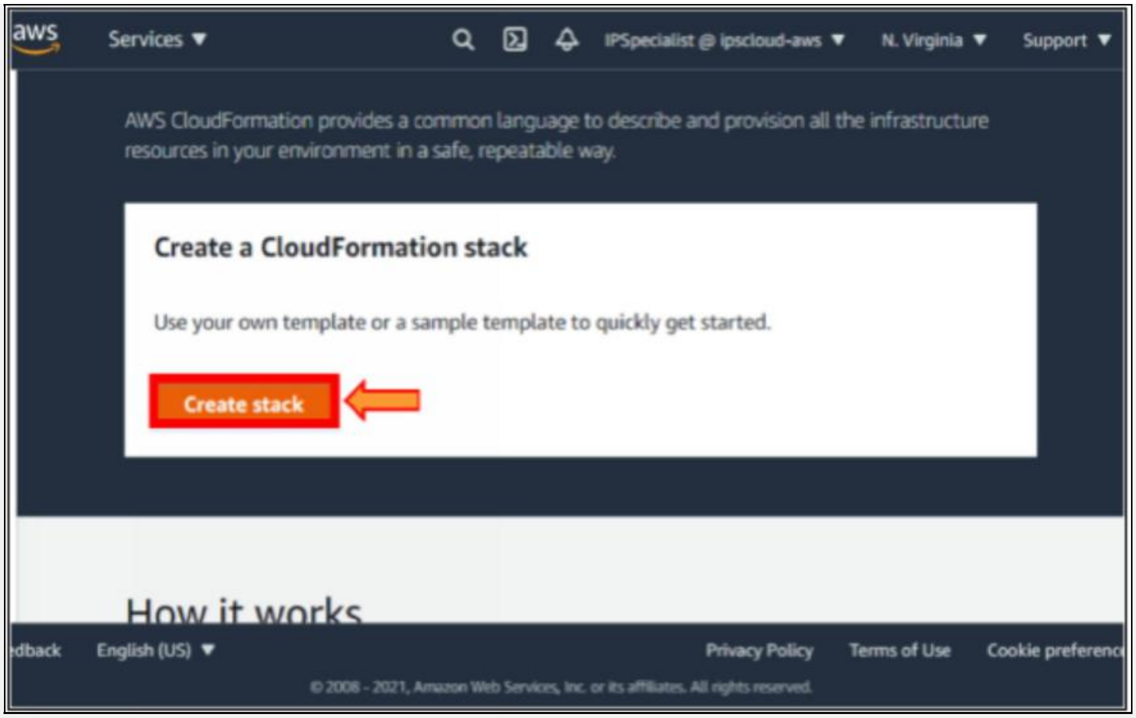
1. Click on **Services.**



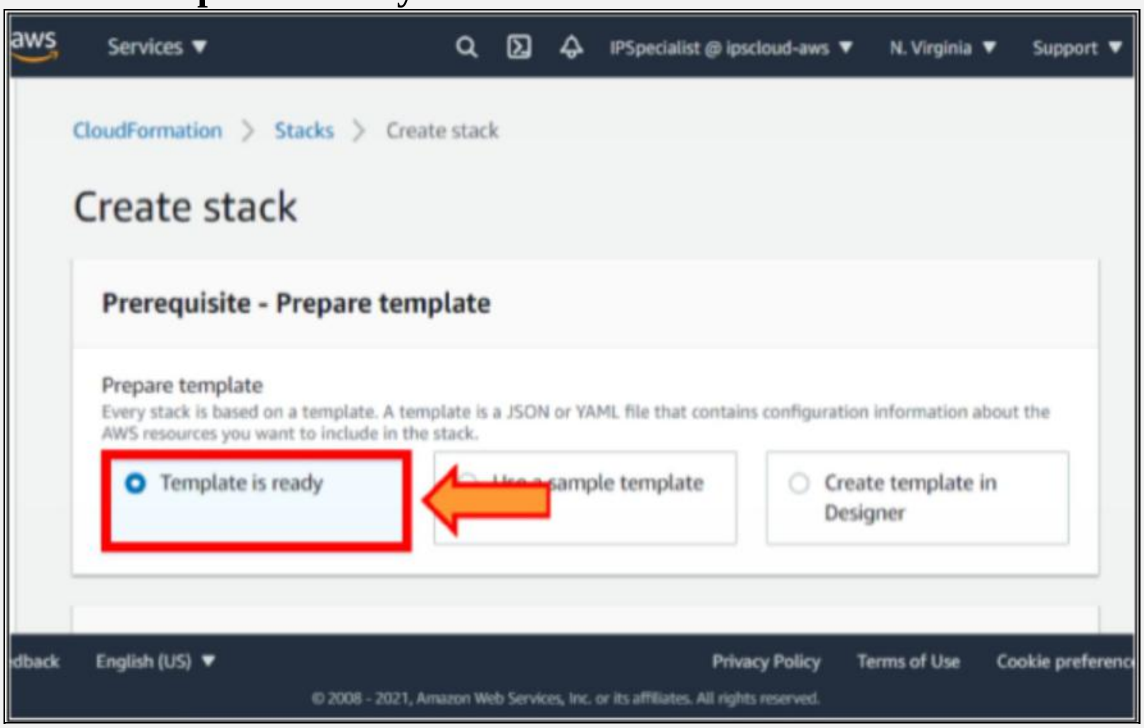
2. Select **CloudFormation** from **Management & Governance**.



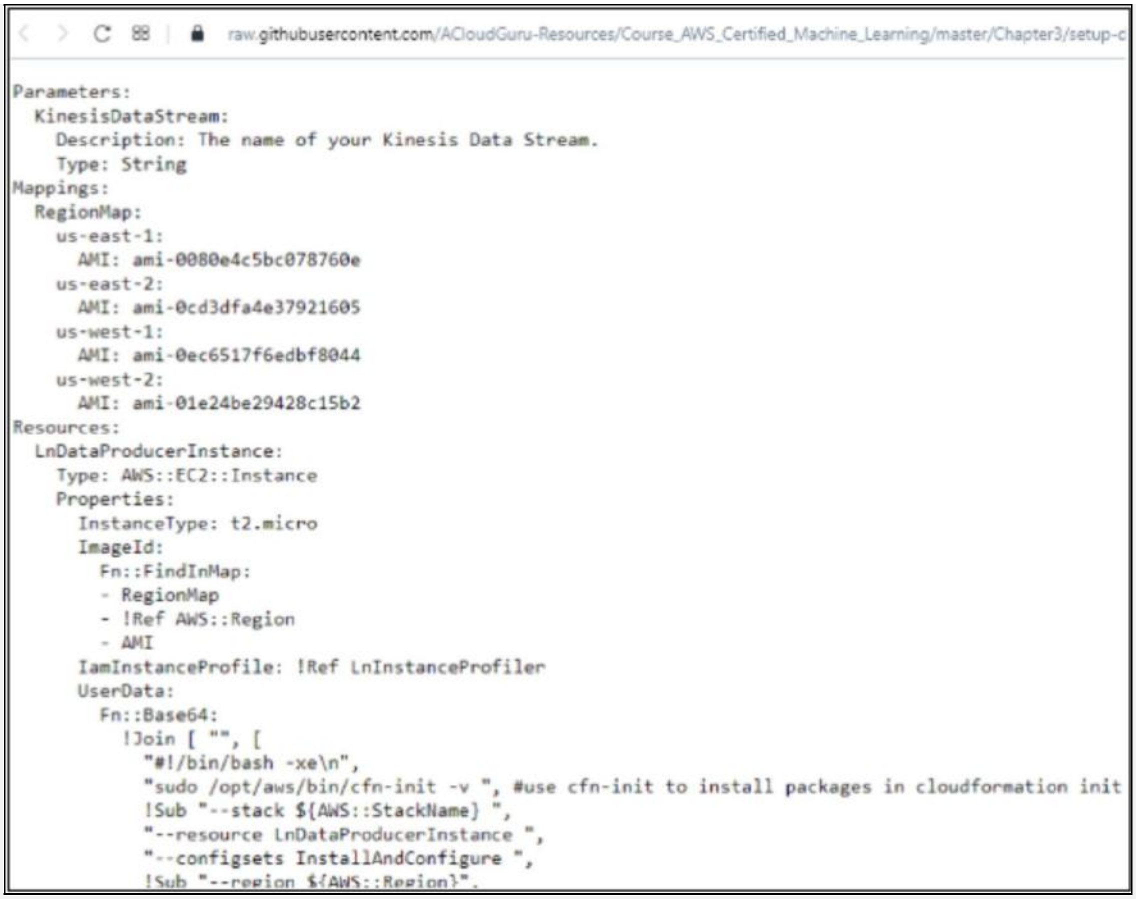
3. Click on the **Create Stack** button.



4. Select **Template is ready.**



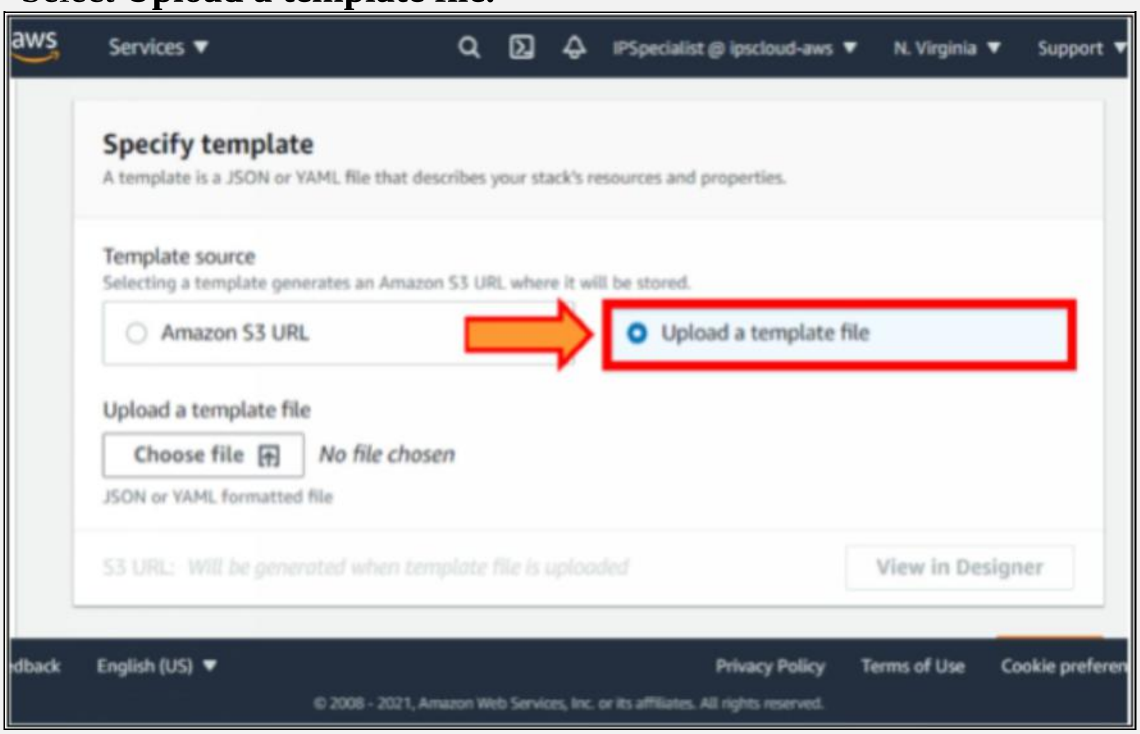
5. Download the template file provided in the following GitHub link: https://raw.githubusercontent.com/ACloudGuru-Resources/Course_AWS_Certified_Machine_Learning/master/Chapter3/setup-data-producer.yml.



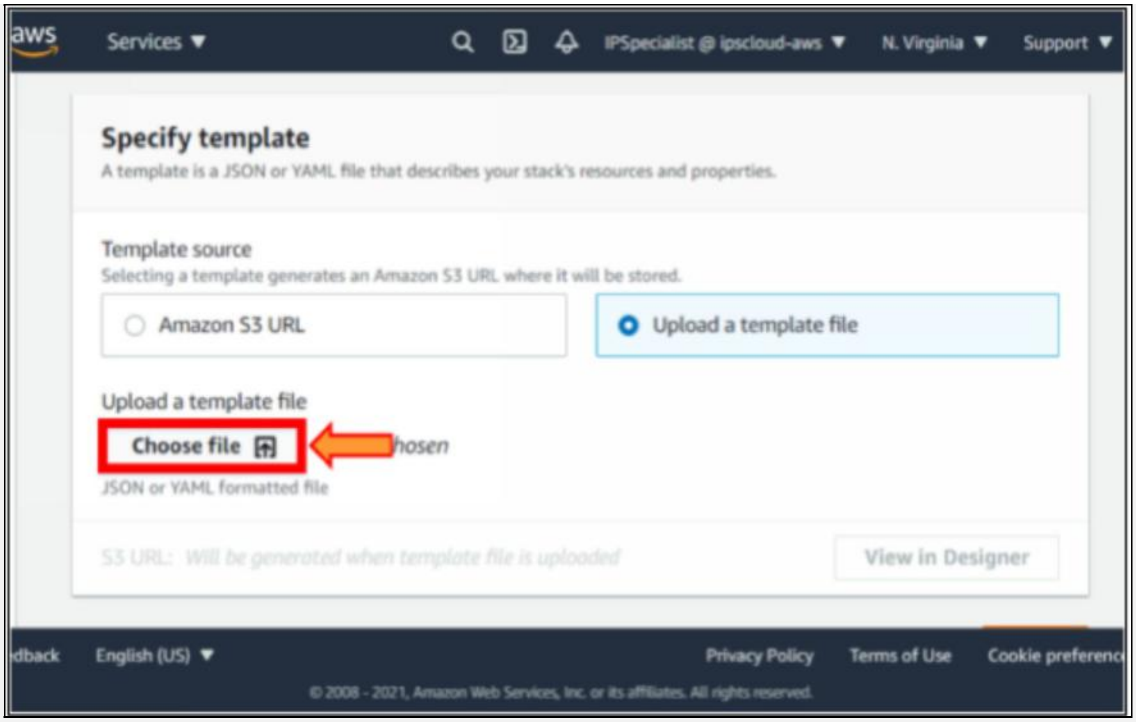
raw.githubusercontent.com/ACloudGuru-Resources/Course_AWS_Certified_Machine_Learning/master/Chapter3/setup-c

```
Parameters:
  KinesisDataStream:
    Description: The name of your Kinesis Data Stream.
    Type: String
Mappings:
  RegionMap:
    us-east-1:
      AMI: ami-0080e4c5bc078760e
    us-east-2:
      AMI: ami-0cd3dfa4e37921605
    us-west-1:
      AMI: ami-0ec6517f6edbf8044
    us-west-2:
      AMI: ami-01e24be29428c15b2
Resources:
  LnDataProducerInstance:
    Type: AWS::EC2::Instance
    Properties:
      InstanceType: t2.micro
      ImageId:
        Fn::FindInMap:
        - RegionMap
        - !Ref AWS::Region
        - AMI
      IamInstanceProfile: !Ref LnInstanceProfiler
      UserData:
        Fn::Base64:
          !Join [ "", [
            "#!/bin/bash -xe\n",
            "sudo /opt/aws/bin/cfn-init -v ", #use cfn-init to install packages in cloudformation init
            !Sub "--stack ${AWS::StackName} ",
            "--resource LnDataProducerInstance ",
            "--configsets InstallAndConfigure ",
            !Sub "--region ${AWS::Region}",
```

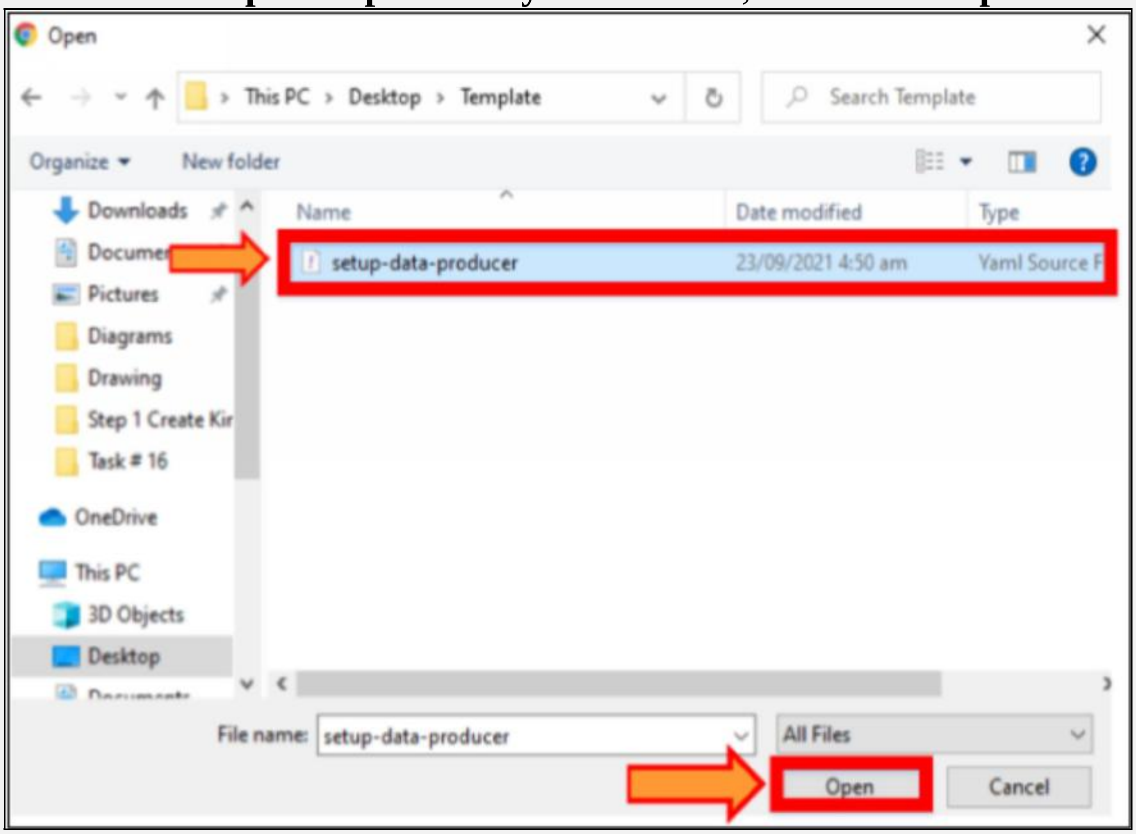
6. Select **Upload a template file.**



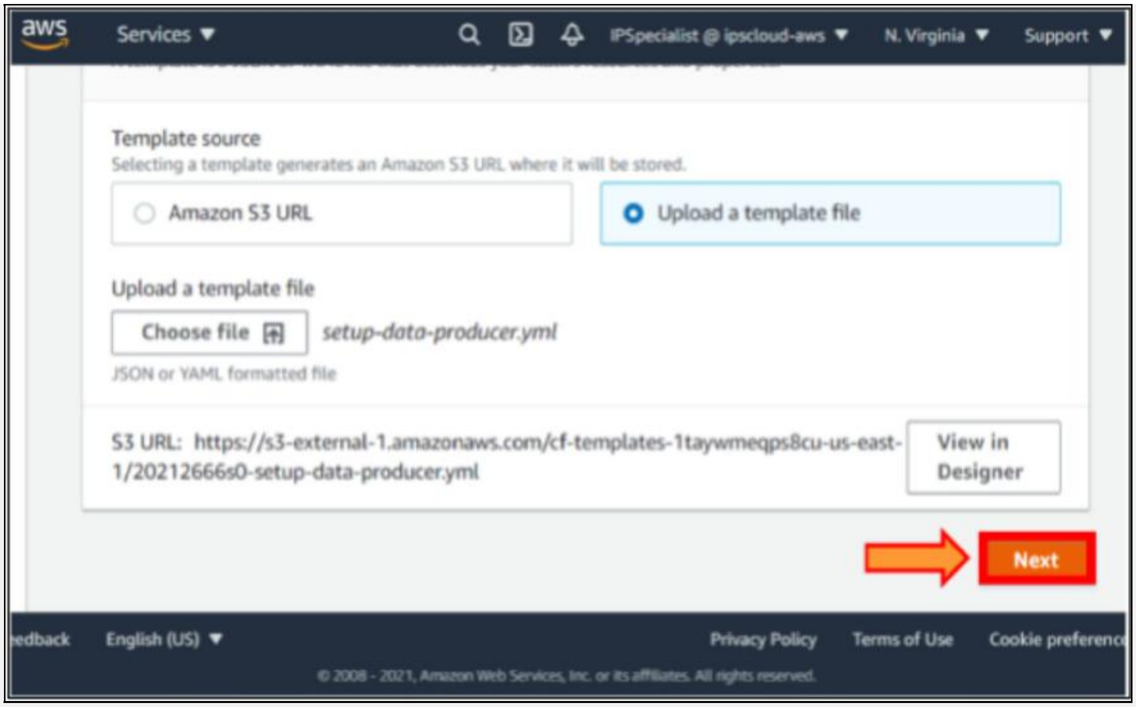
7. Click on the **Choose file** button.



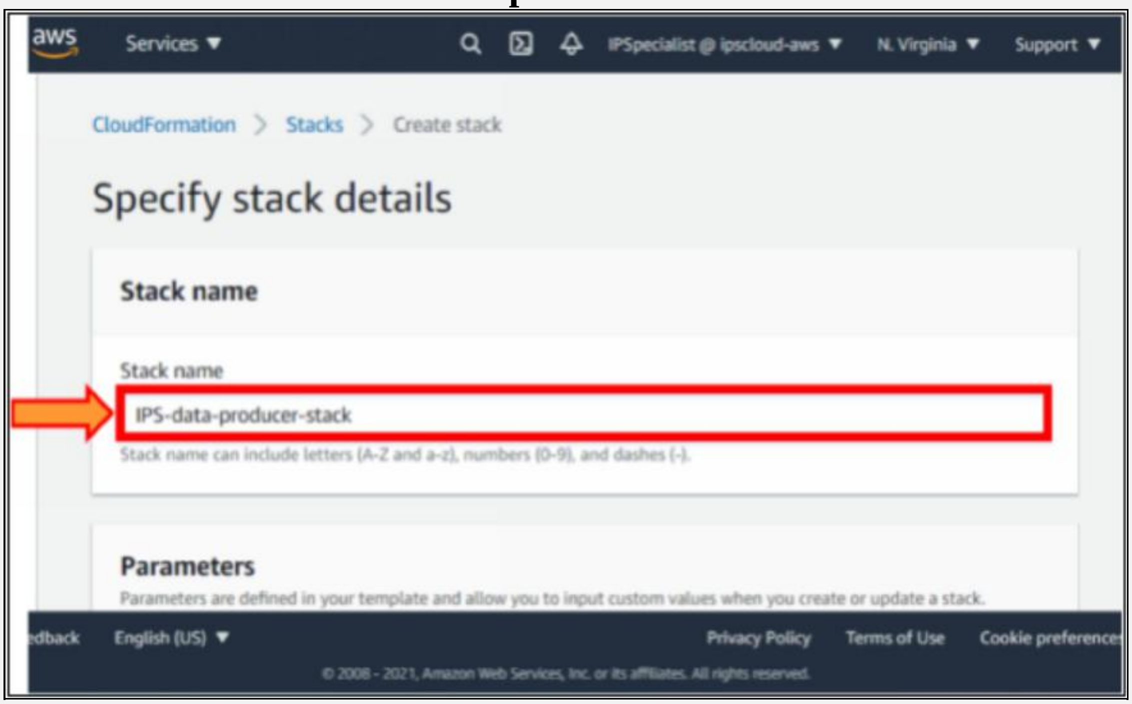
8. Select the **setup-data-producer.yml** file. Then, click on the **Open** button.



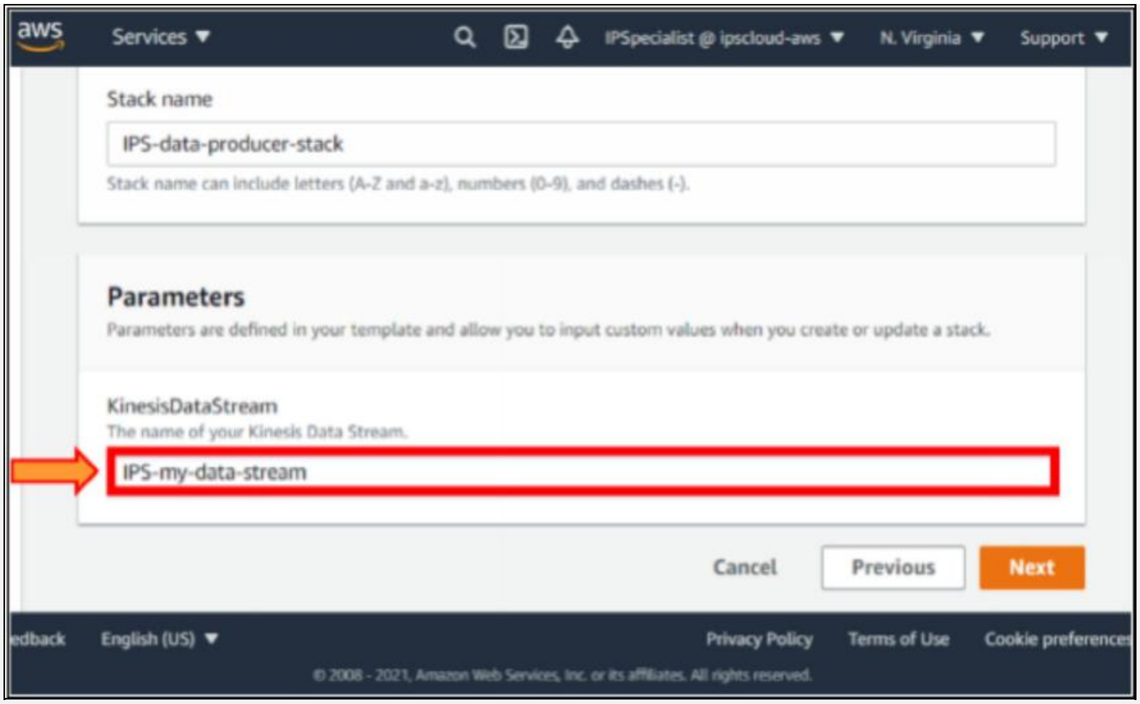
9. Click on the **Next** button.



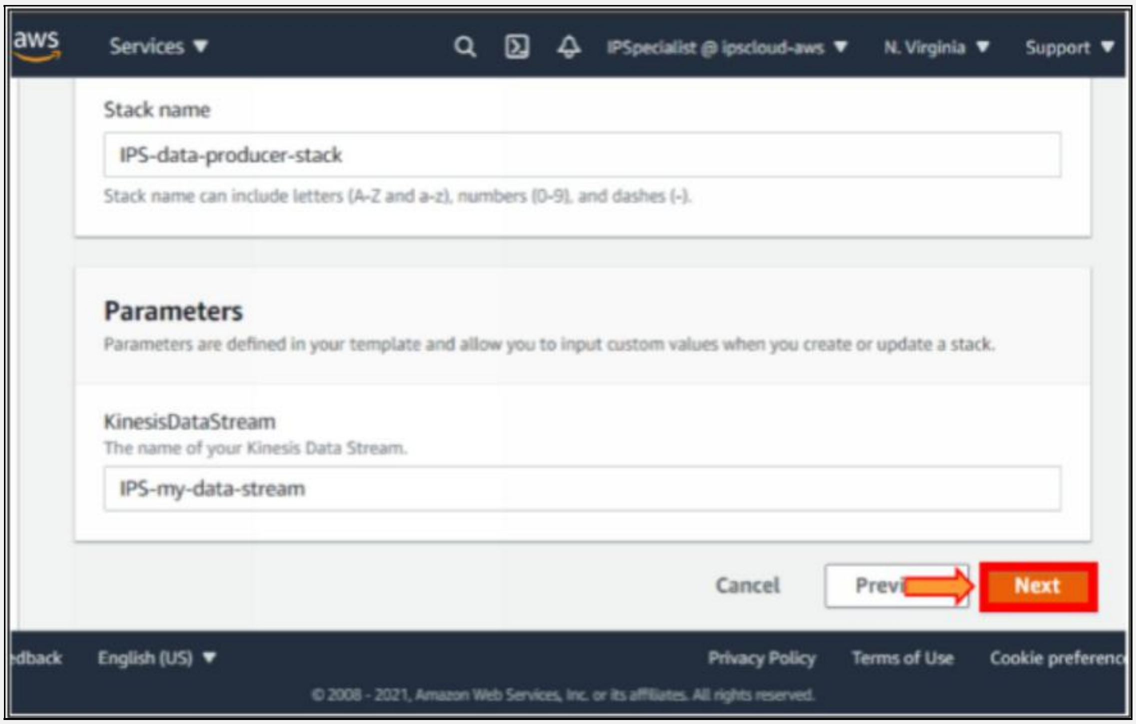
10. Name the stack as **IPS-data-producer-stack.**



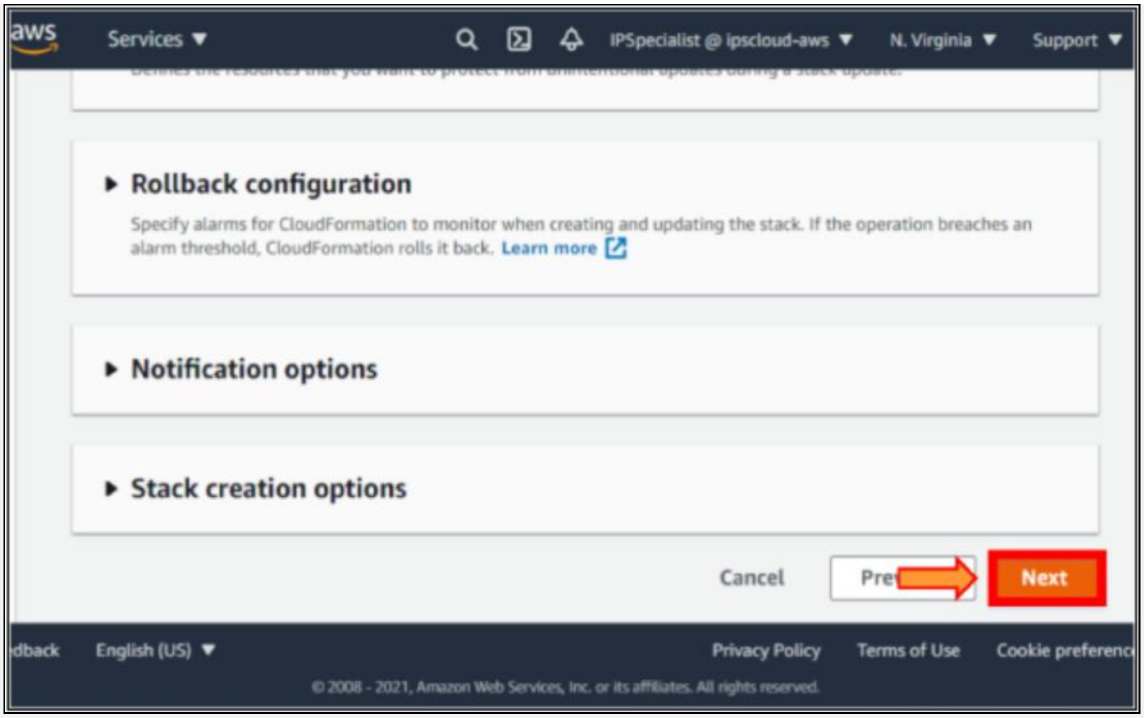
11. Then, give the Kinesis data stream name **IPS-my-data-stream.**



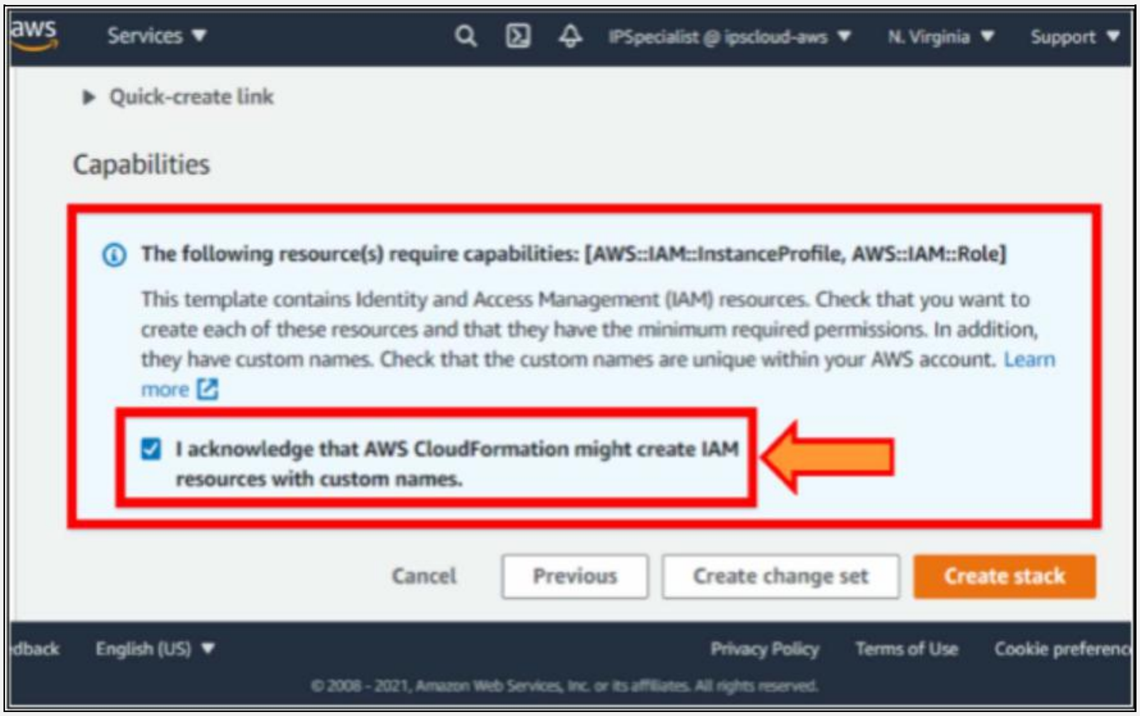
12. Click on the **Next** button.



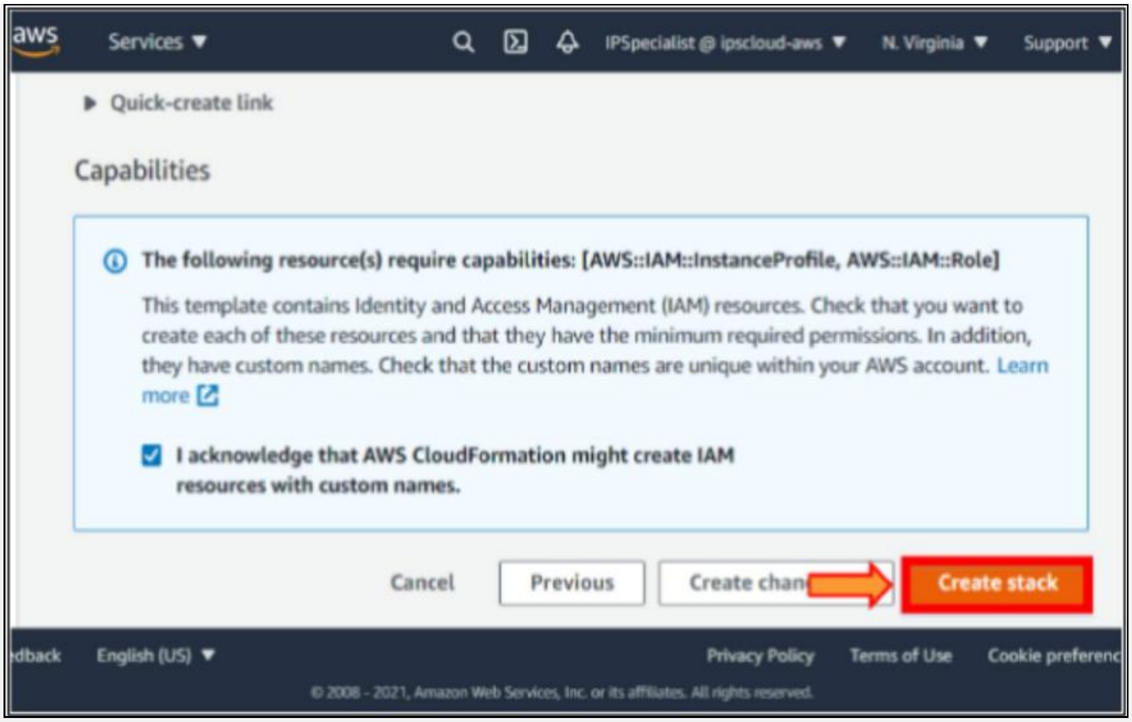
13. Scroll down. Click on the **Next** button.



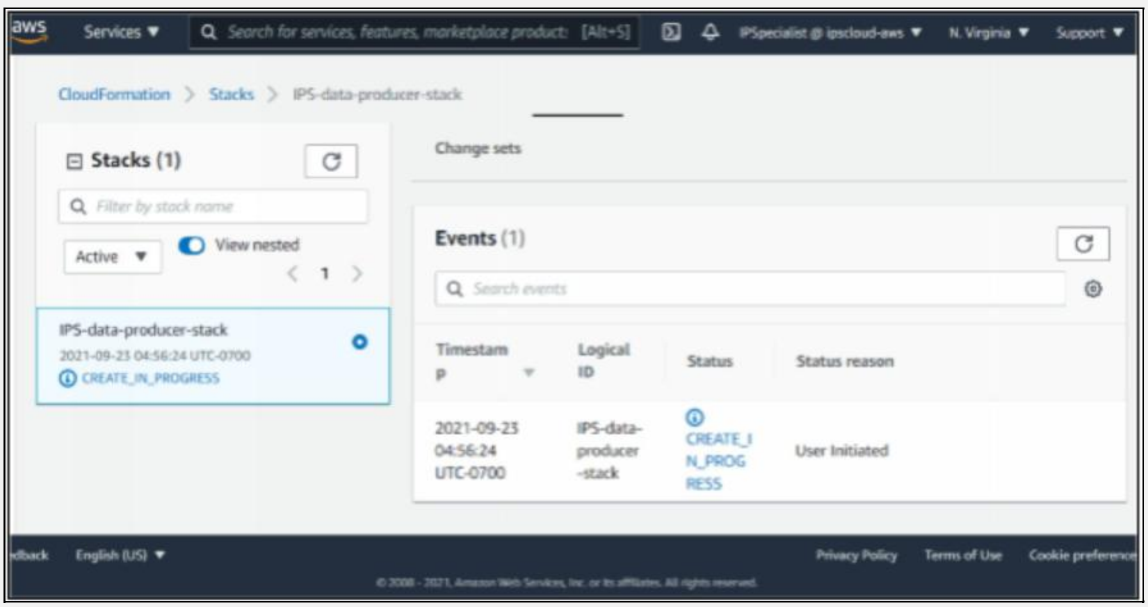
14. Scroll down. Click on the check box.



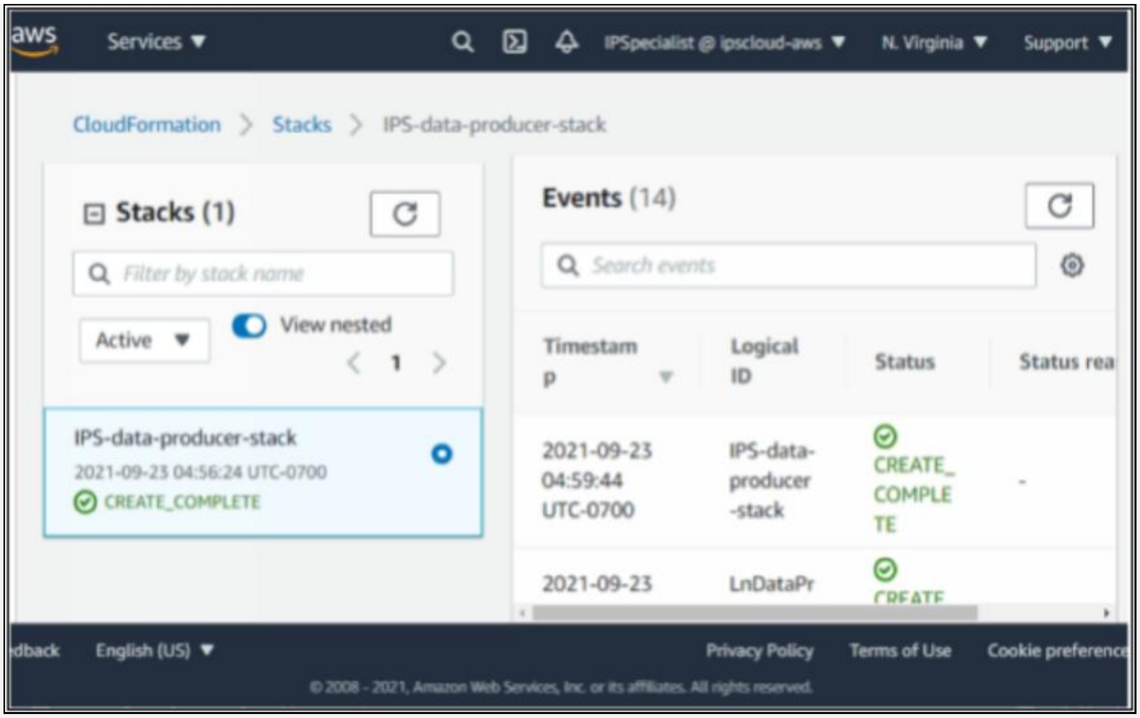
15. Then, click on the **Create Stack** button.



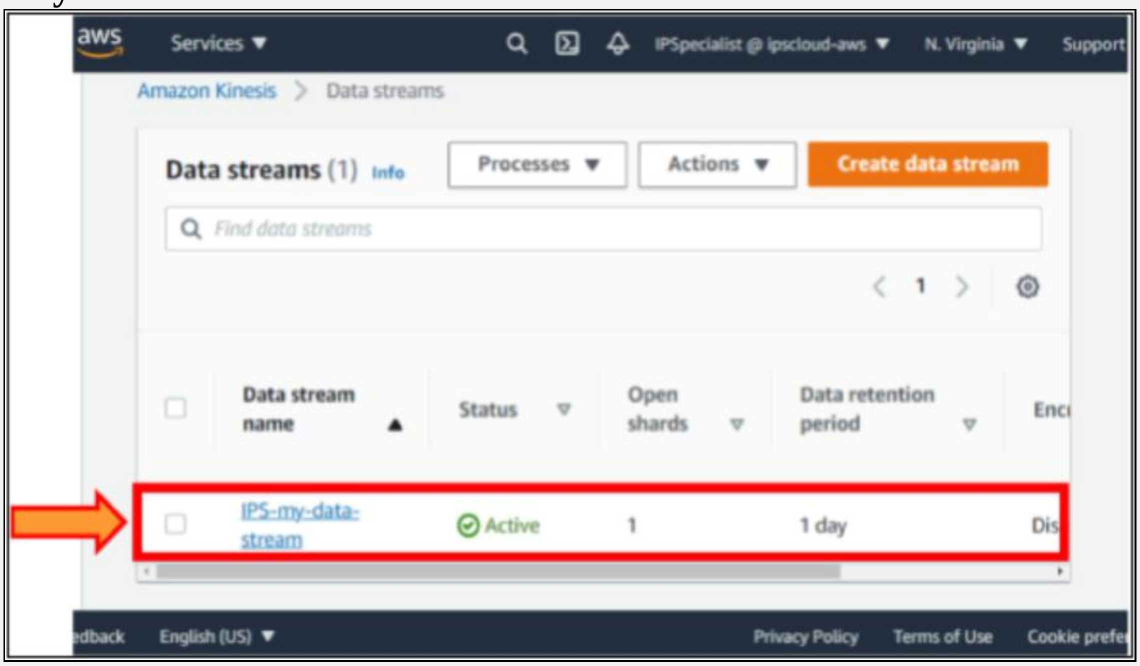
16. It will take a few minutes to create a stack. Once completed, CloudFormation will create all the services and automatically start putting that random user data into the Kinesis stream.



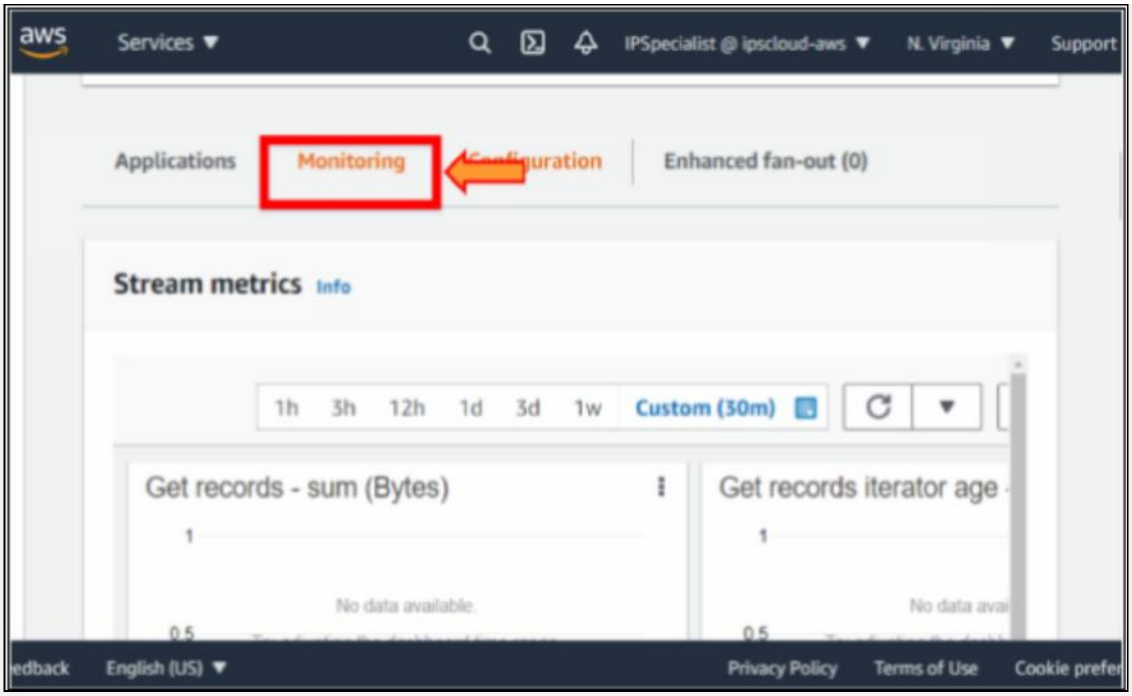
Hence, CloudFormation Stack work is completed.



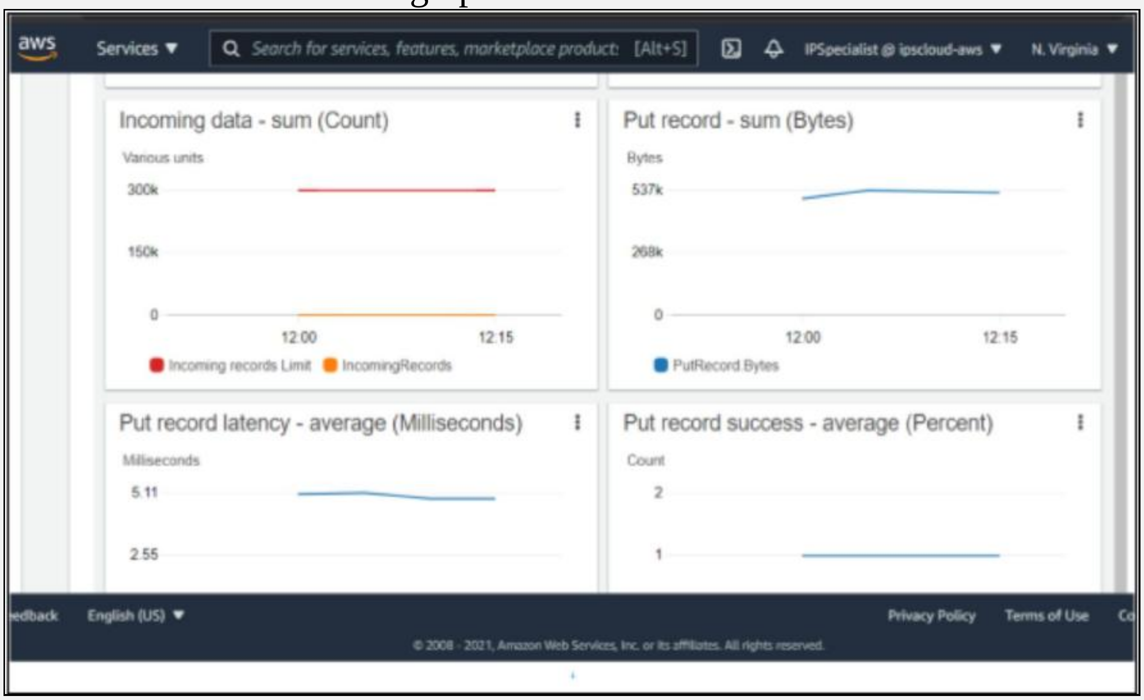
17. Go back to the **Kinesis Data Stream** dashboard, and click on the **IPS-my-data-stream.**



18. Click on the **Monitoring** tab.

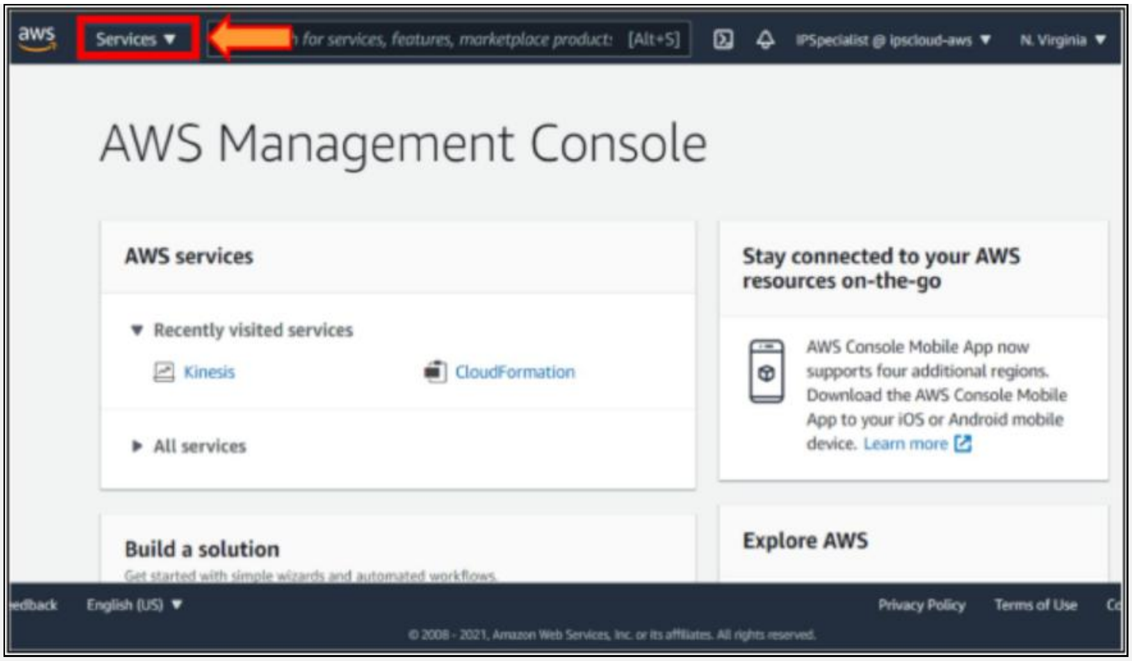


19. You will see different graphs.

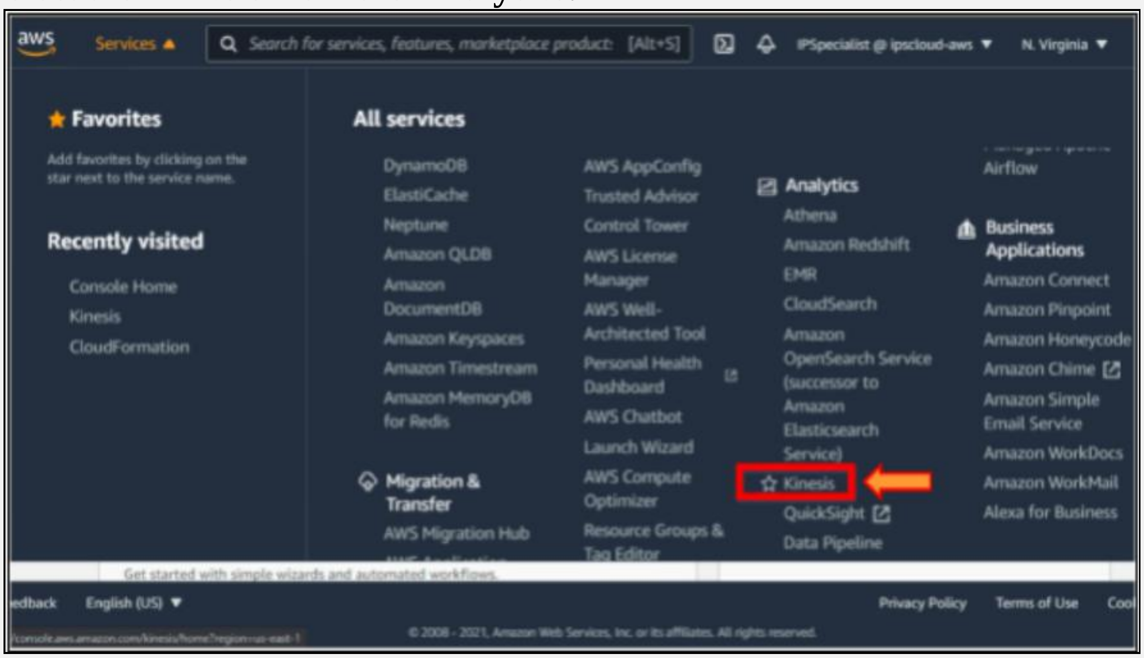


Step 3: Create Kinesis Data Analytics

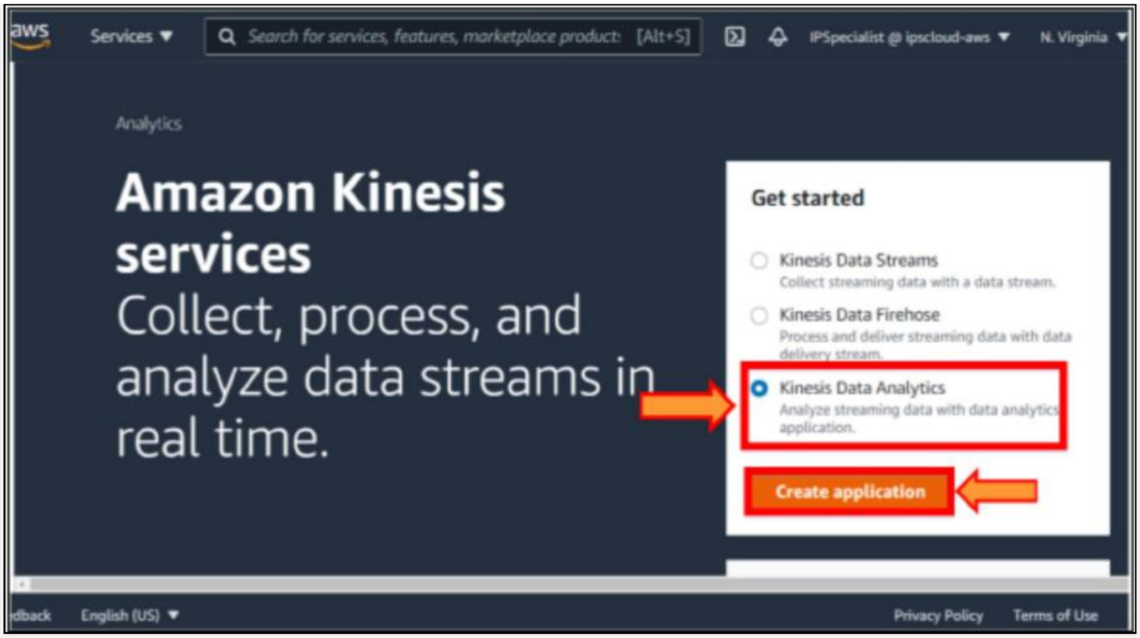
1. Click on **Services**.



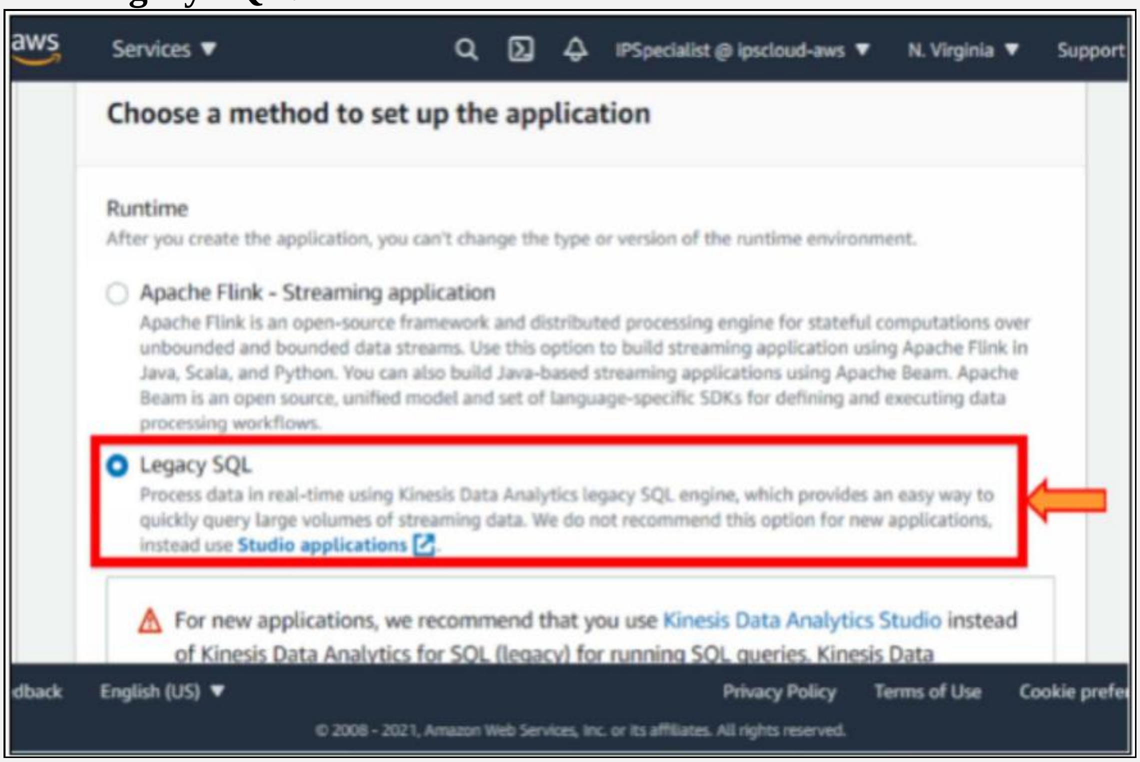
2. Select the **Kinesis** from the **Analytics.**



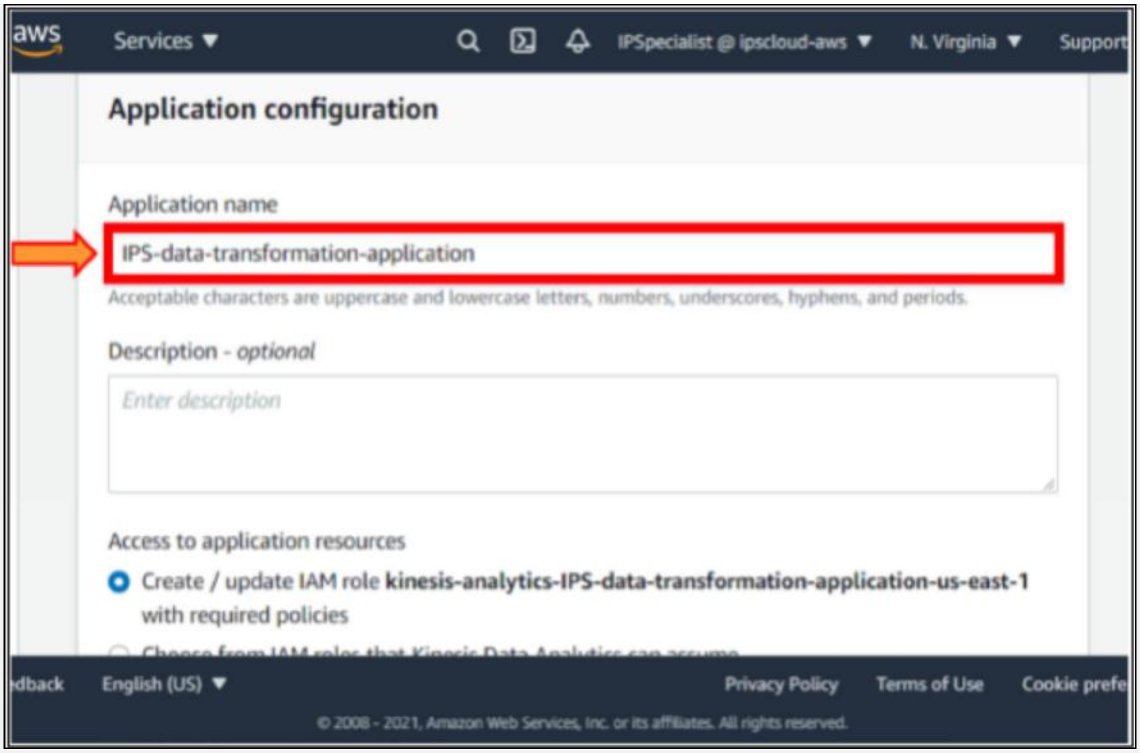
3. Select **Kinesis Data Analytics.**
4. Click on the **Create Application** button.



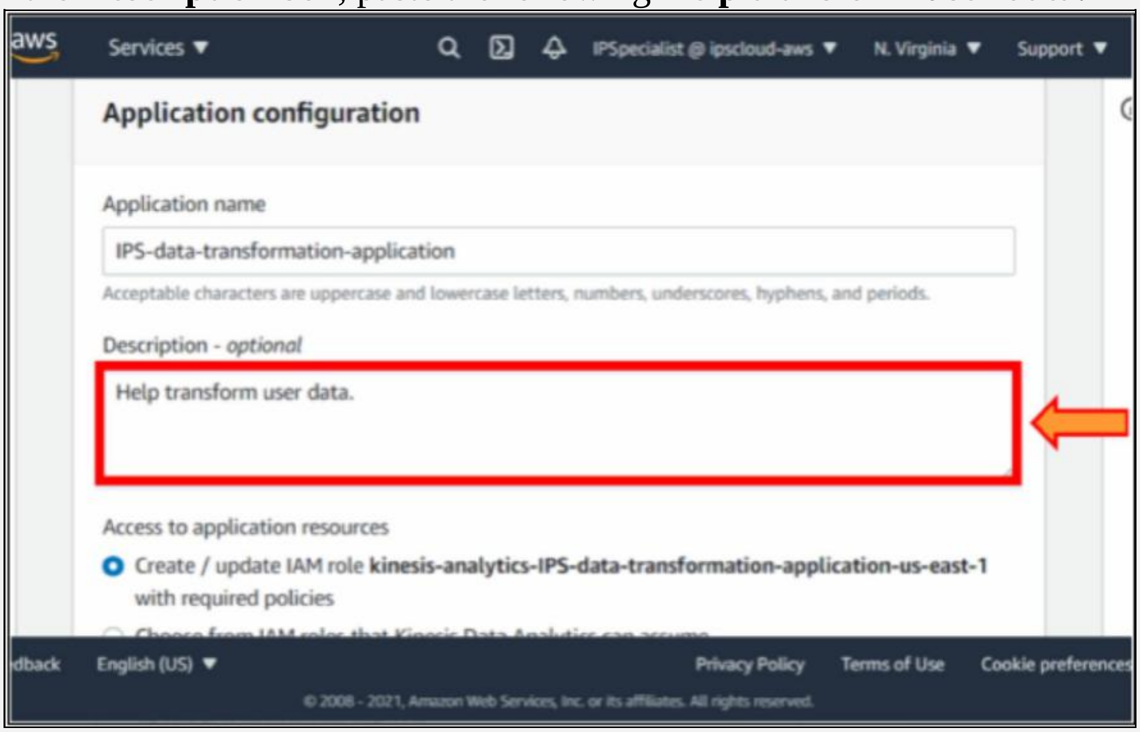
5. Select **Legacy SQL.**



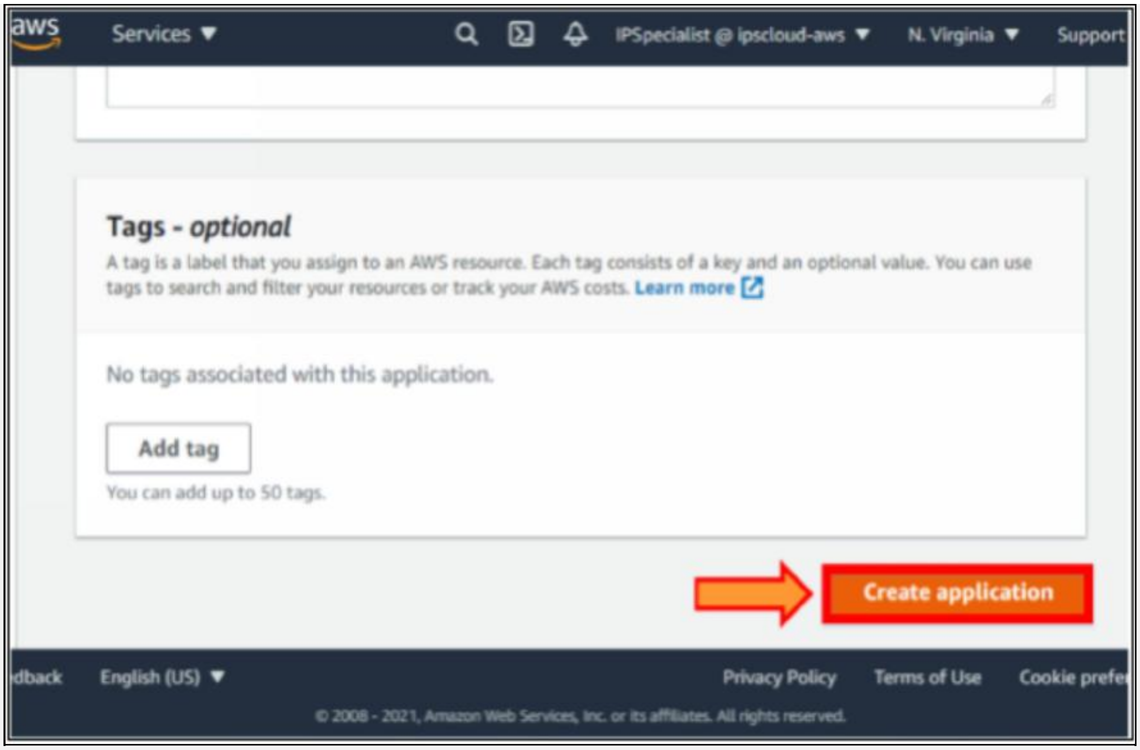
6. Give an application name **IPS-data-transformation-application.**



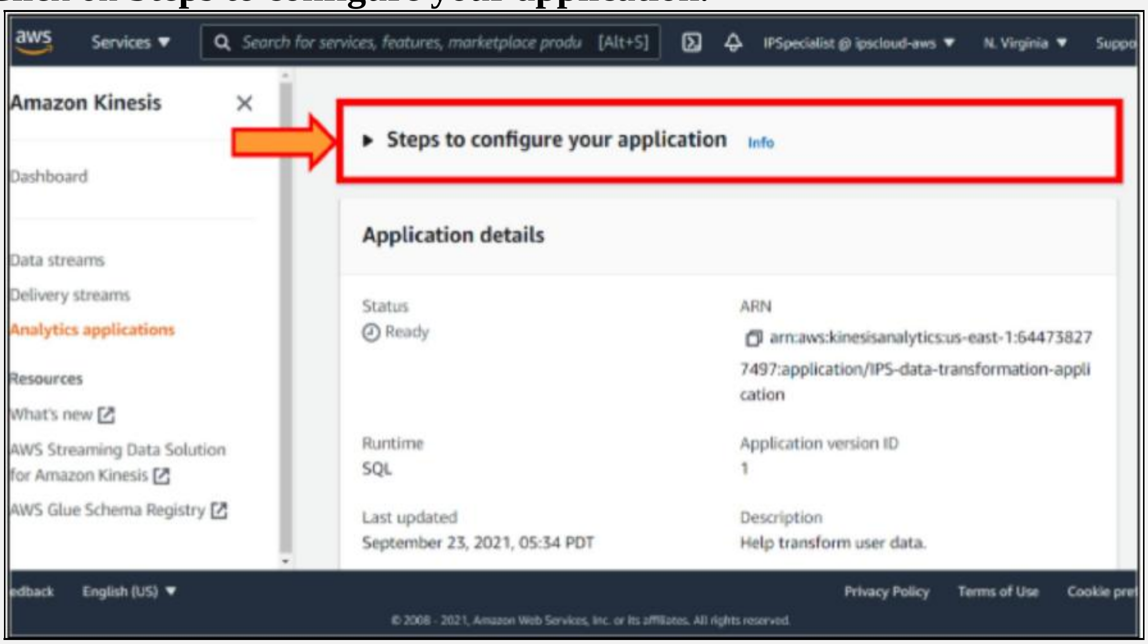
7. In the **Description** box, paste the following **Help transform user data.**



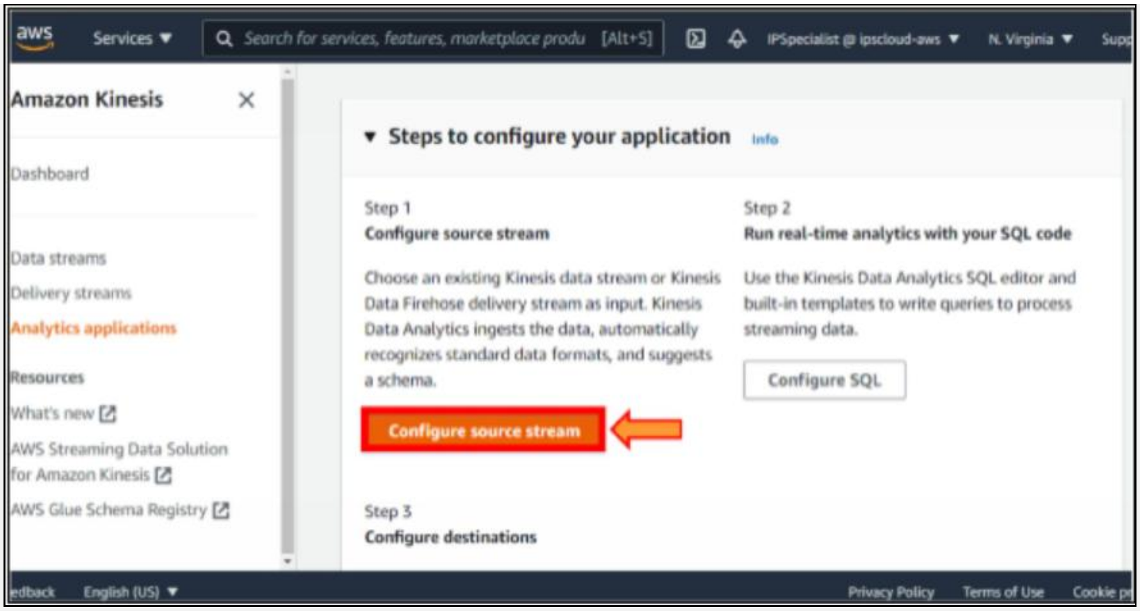
8. Click on the **Create application** button.



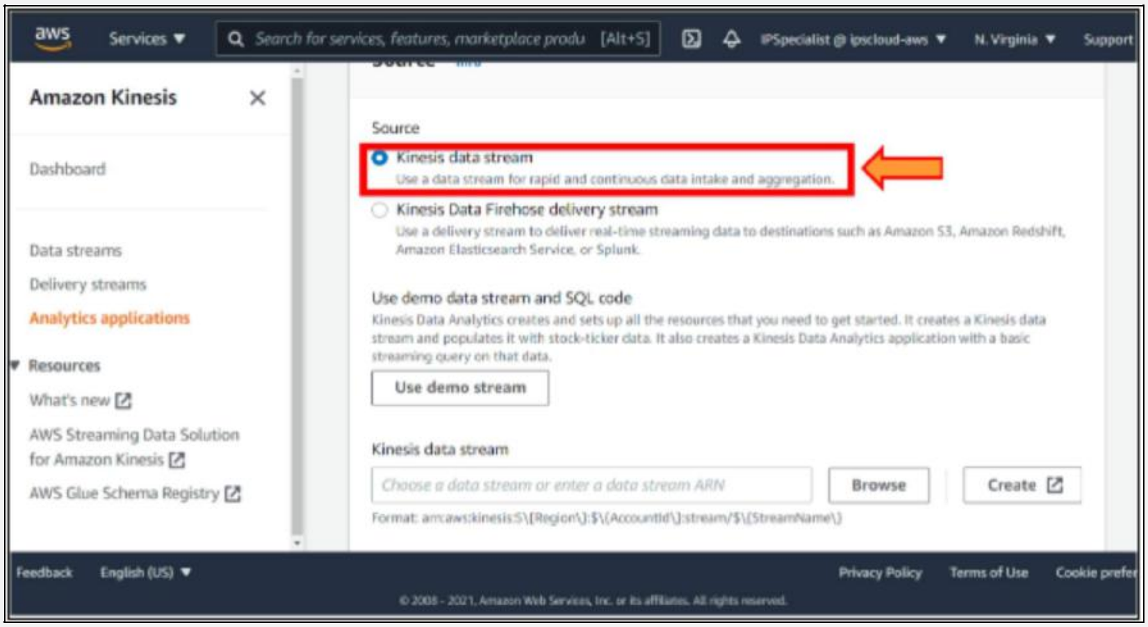
9. Click on **Steps to configure your application.**



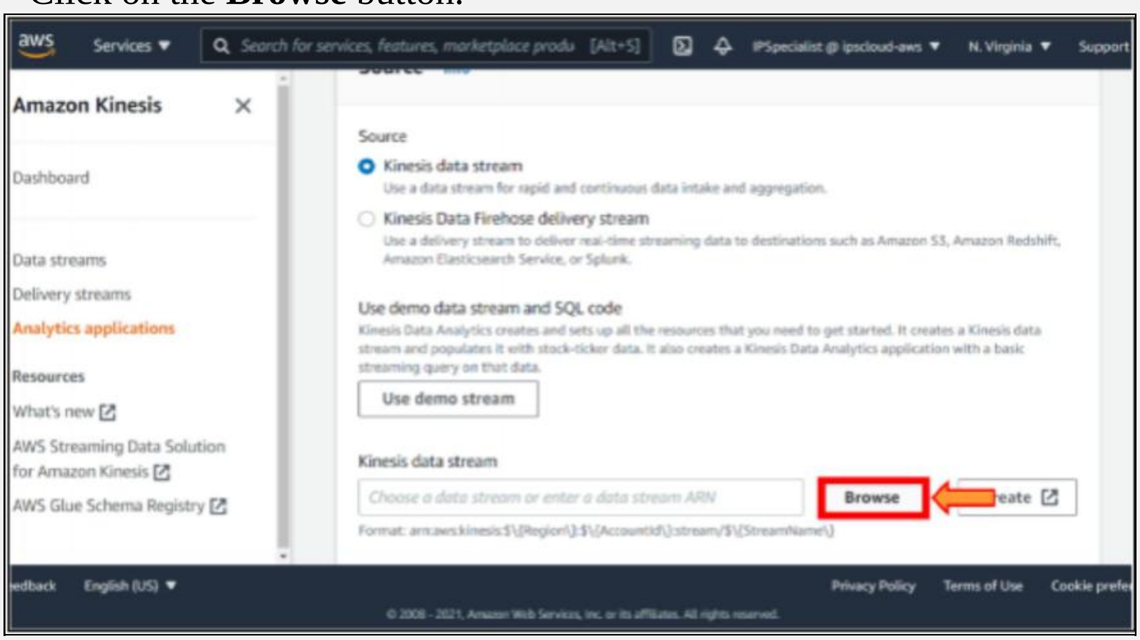
10. Click on the **Configure source stream** button.



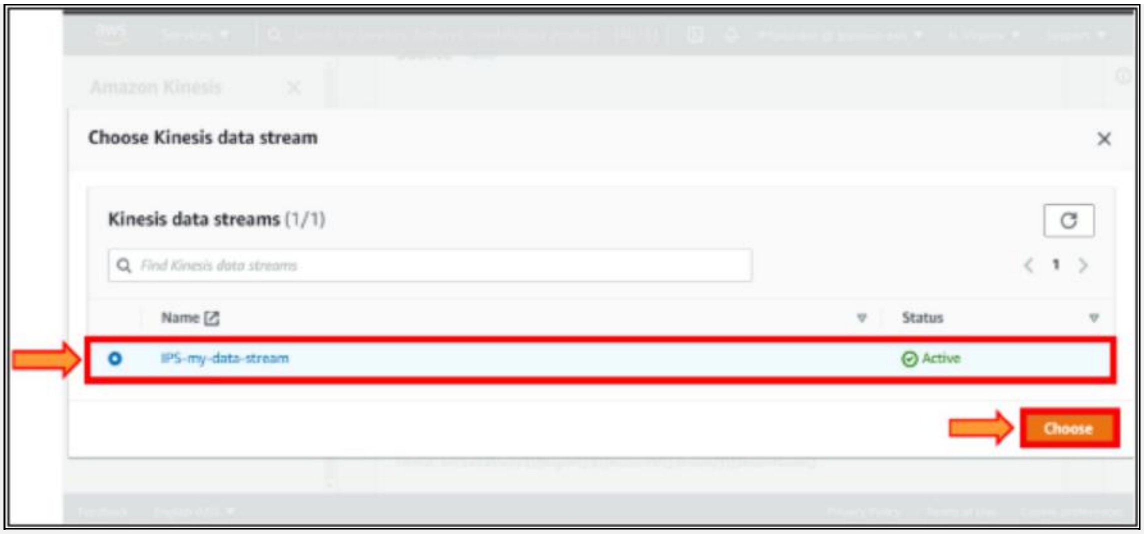
11. Select **Kinesis Data Stream.**



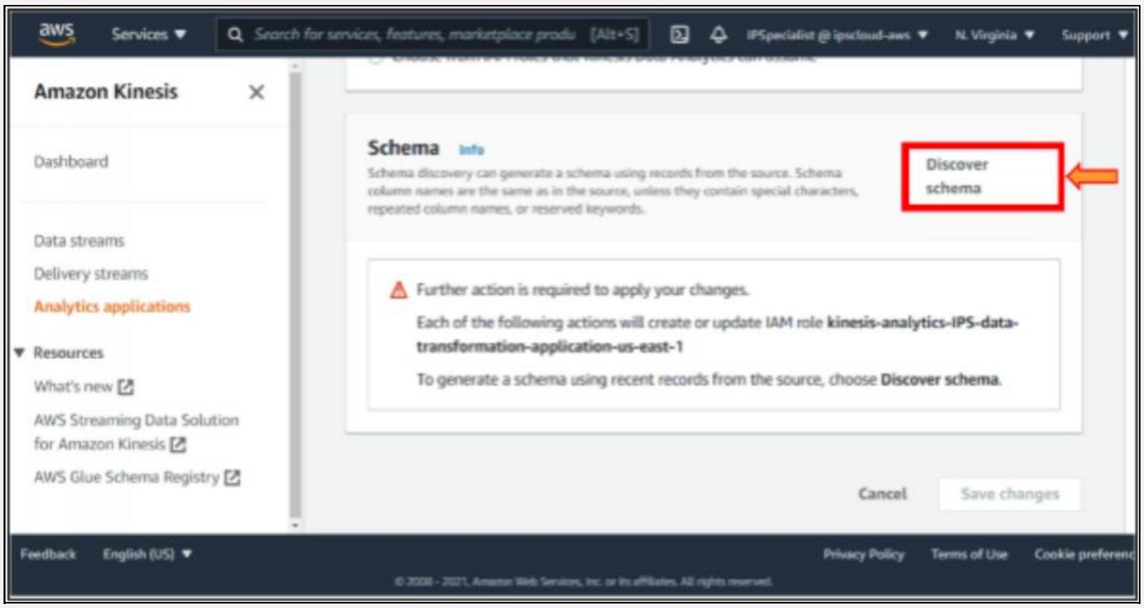
12. Click on the **Browse** button.



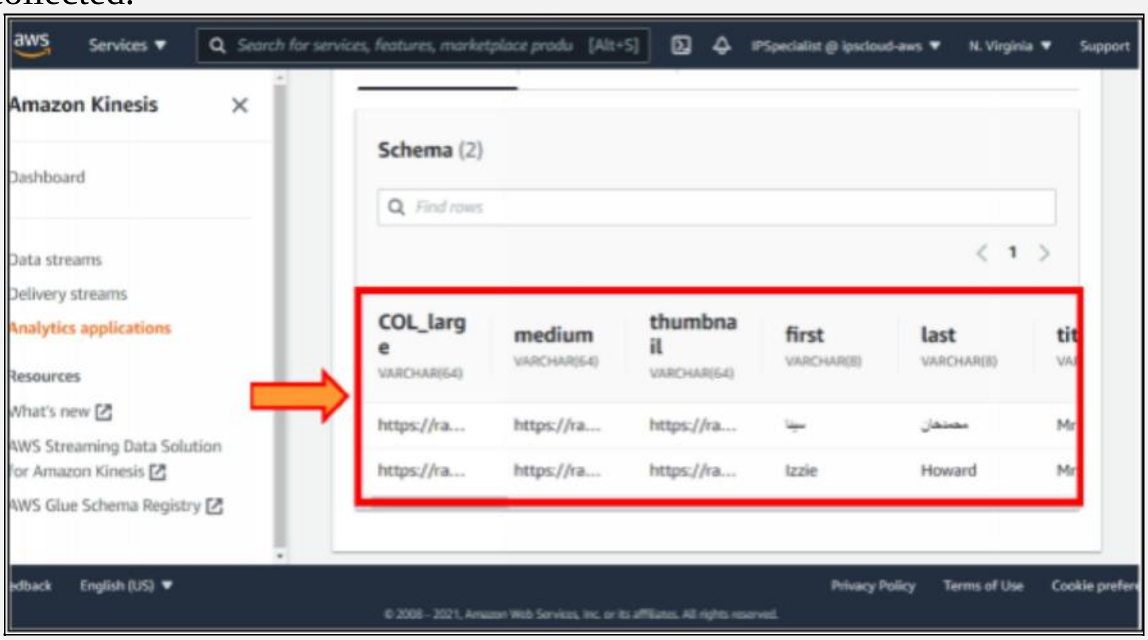
13. Select **IPS-my-data-stream**. Then, click on the **Choose** button.



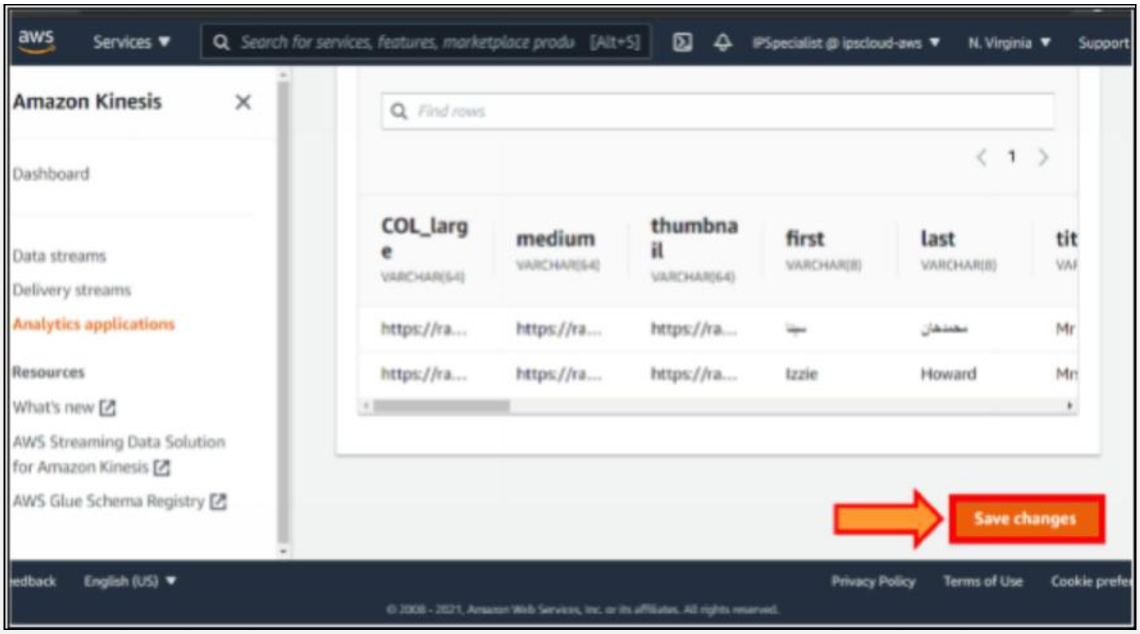
14. Scroll down. Click on the **Discover Schema** button.



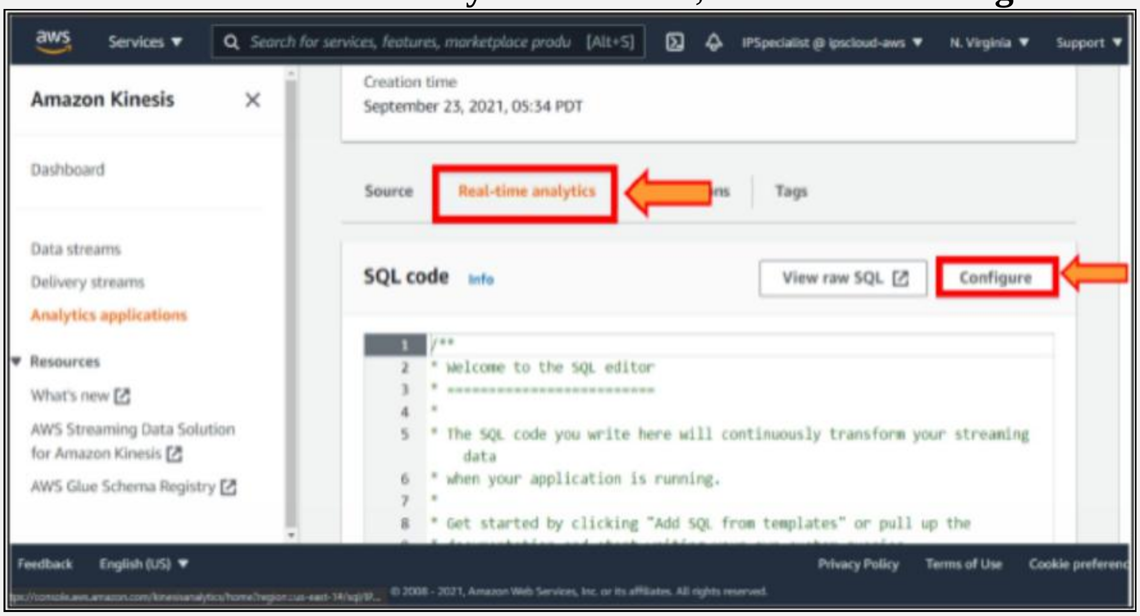
15. You will see the Raw and Formatted streaming data which has been collected.



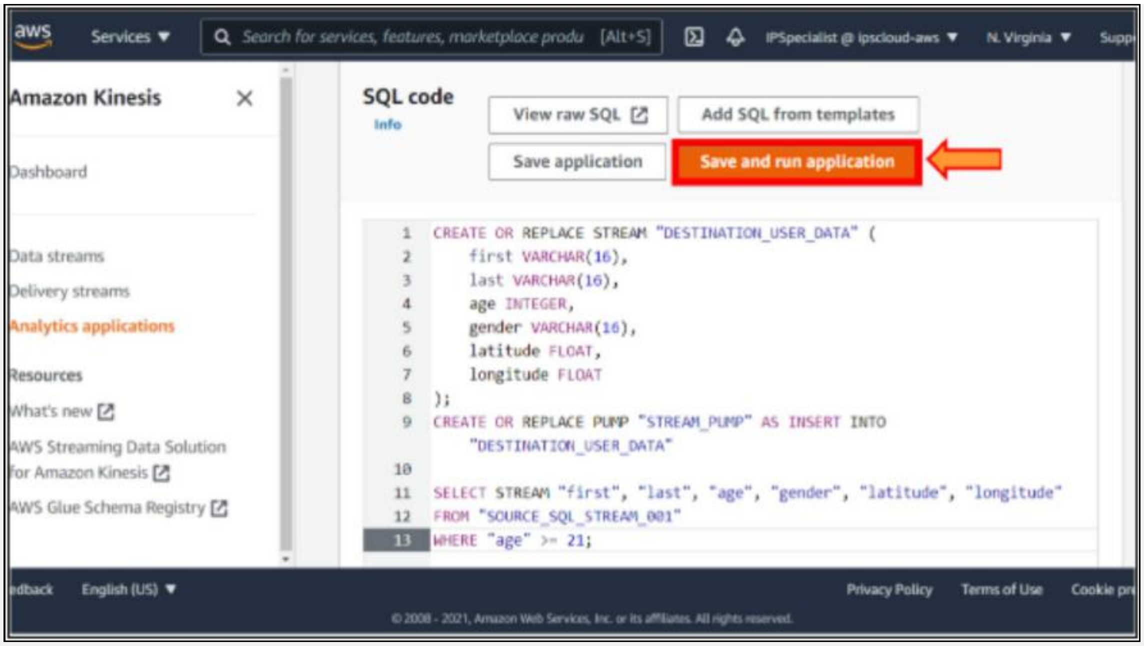
16. Click on the **Save changes** button.



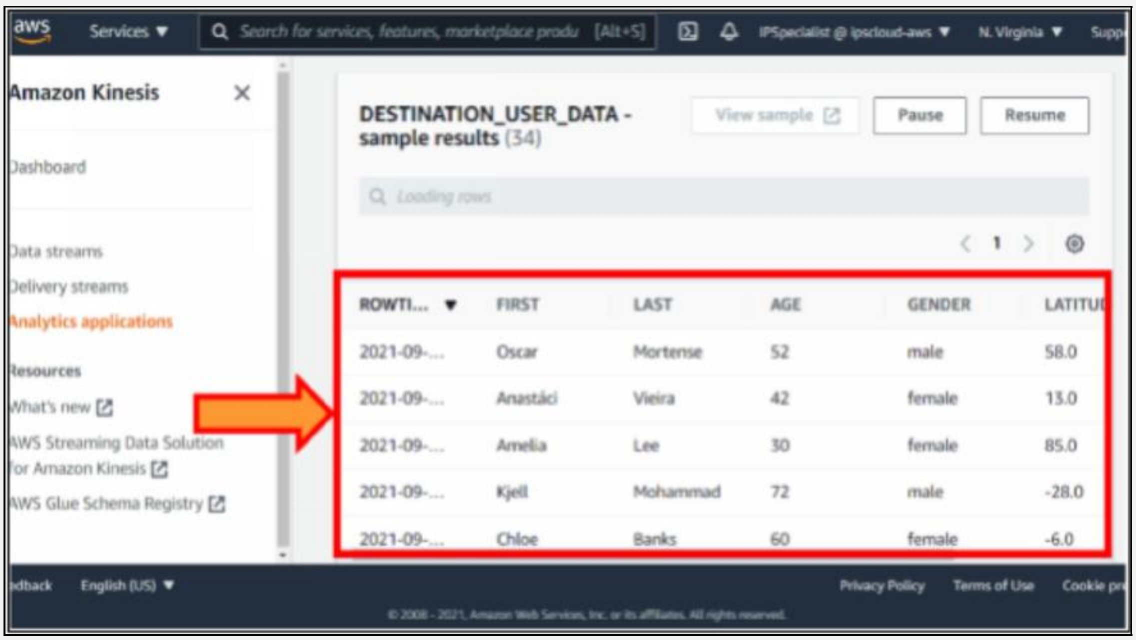
17. Click on the **Real-time analytics** tab. Then, click on the **Configure** button.



18. Now copy and paste the SQL queries in the code editor from the file provided in the following GitHub link: https://github.com/ACloudGuru-Resources/Course_AWS_Certified_Machine_Learning/blob/master/Chapter3/create-subset-transformation-query.sql
19. Click on the **Save and run application** button.

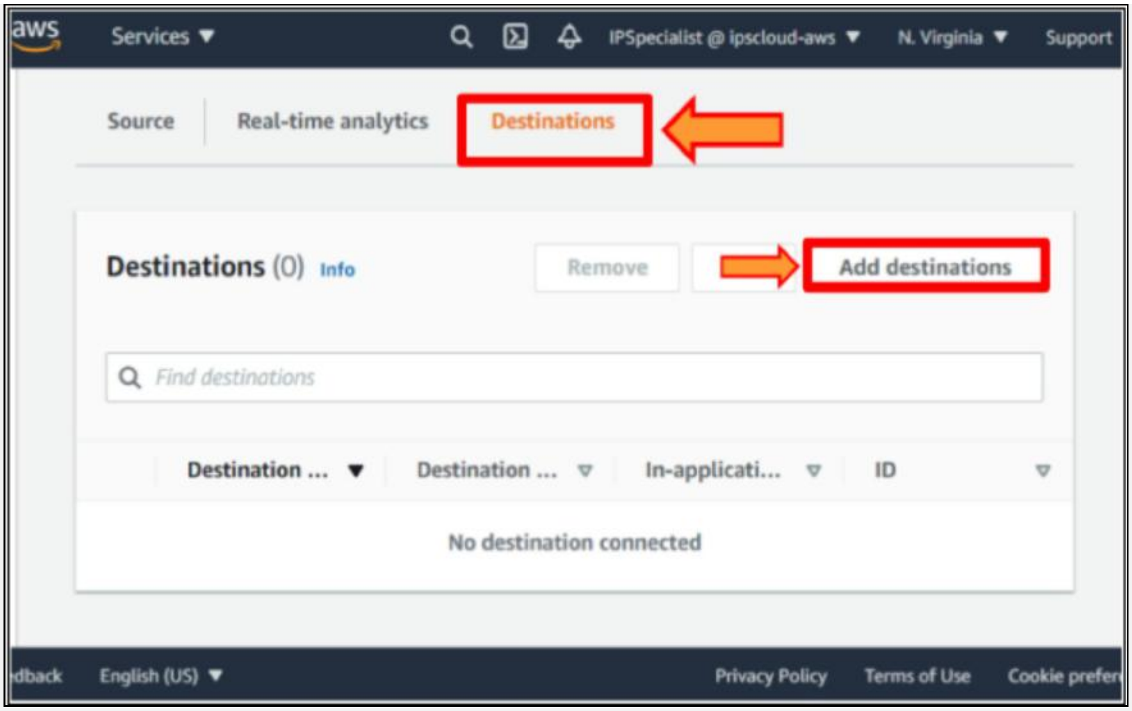


20. It will take a few minutes. Once it is completed, you will see the subset of data.

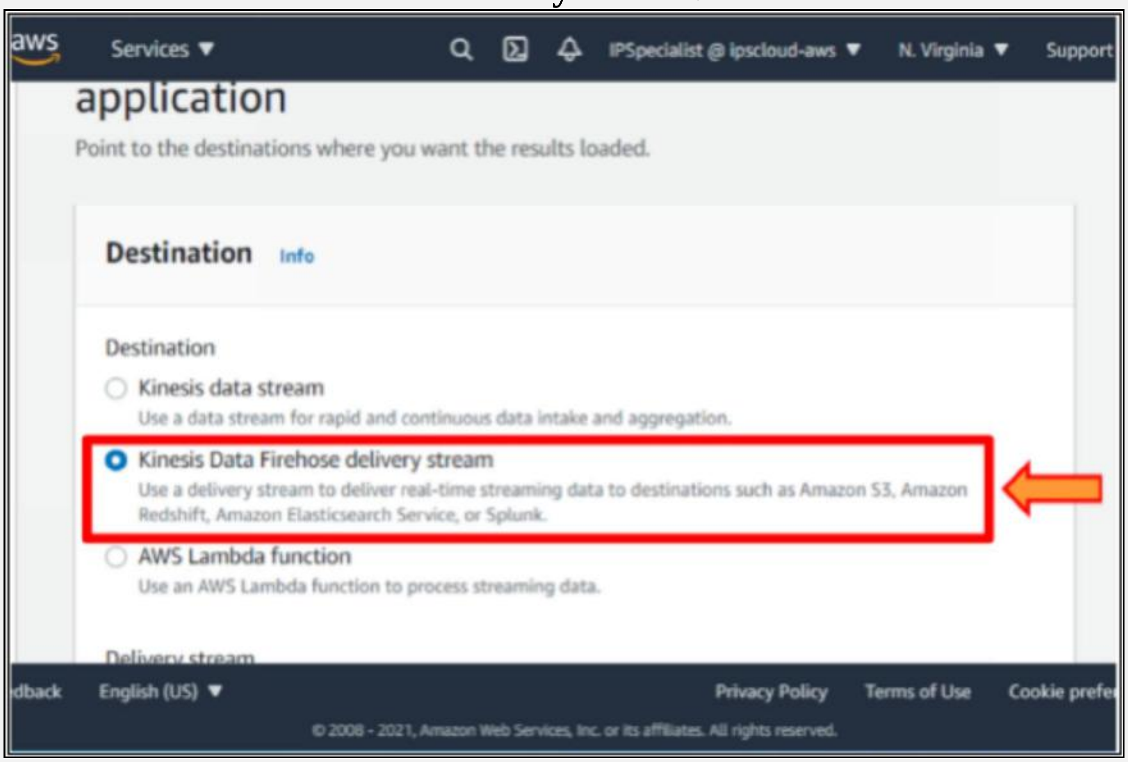


Step 4: Create Kinesis Data Firehose

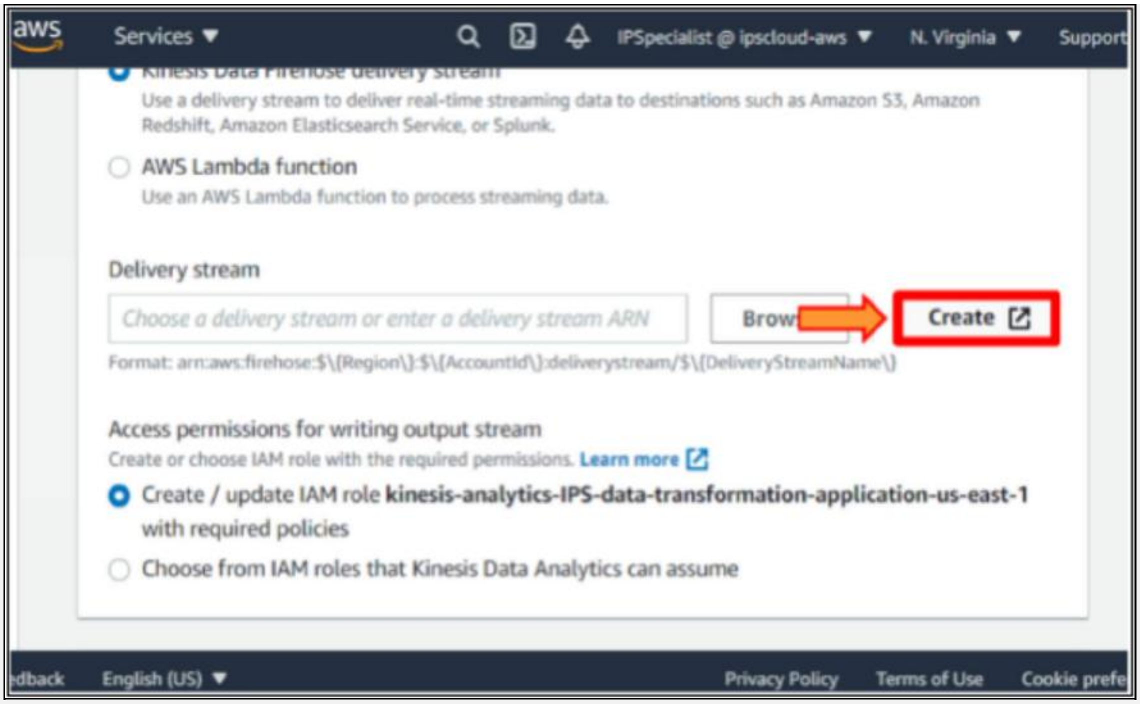
1. Click on the **Destination** tab.
2. Click on **Add destination**.



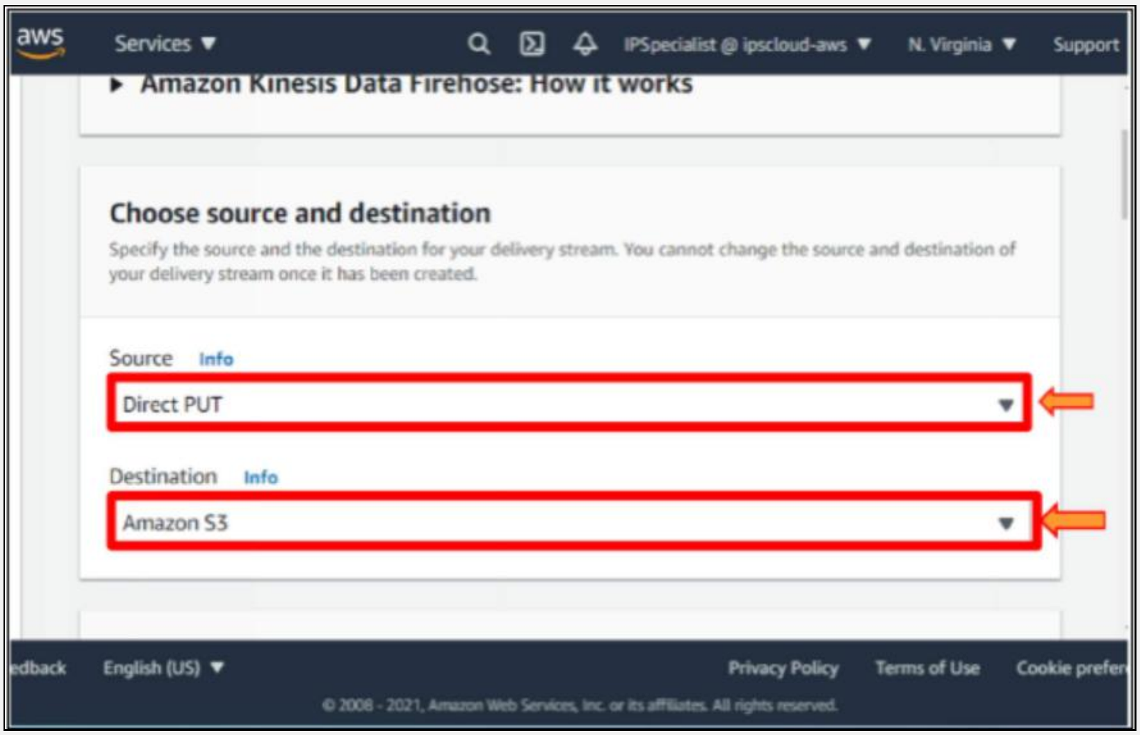
3. Select **Kinesis Data Firehose delivery stream.**



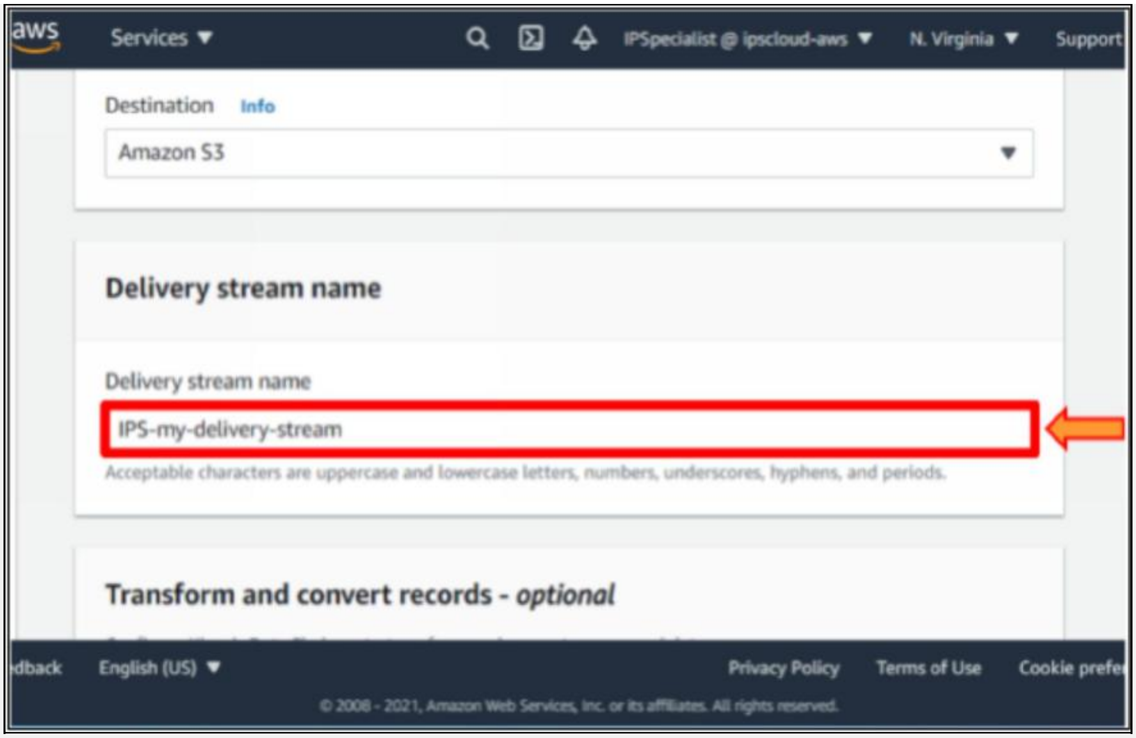
4. Click on the **Create** button.



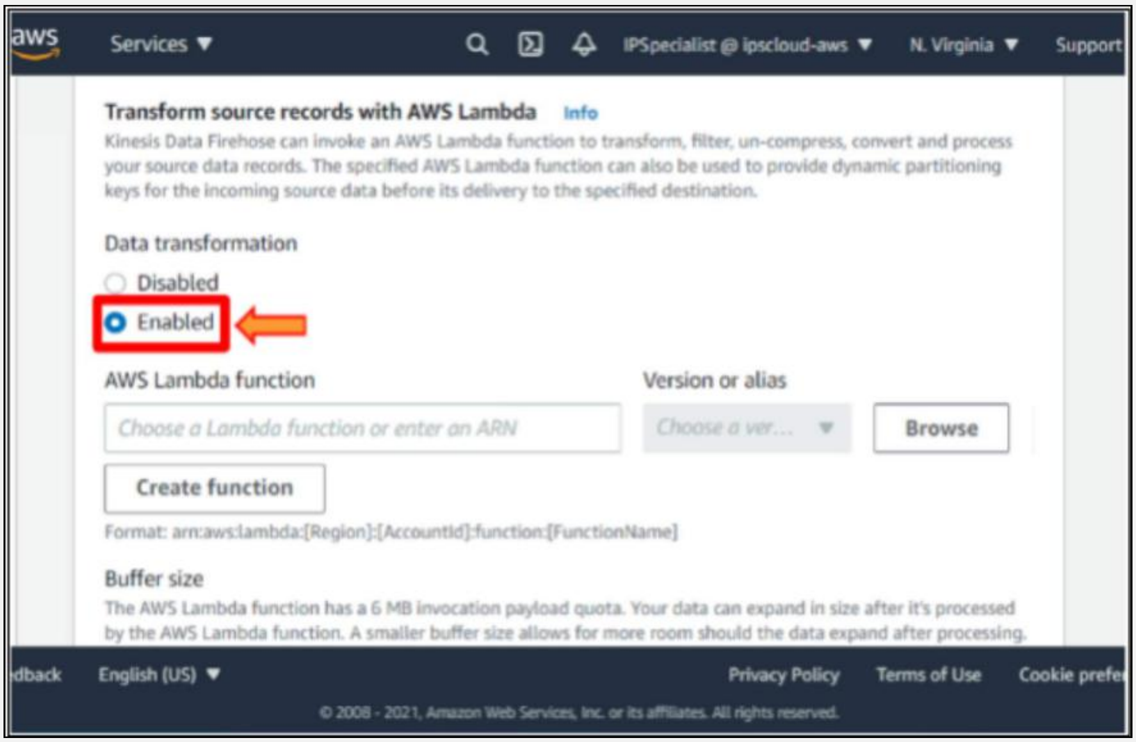
5. Select the source **Direct PUT**. Then set the destination as **Amazon S3**.



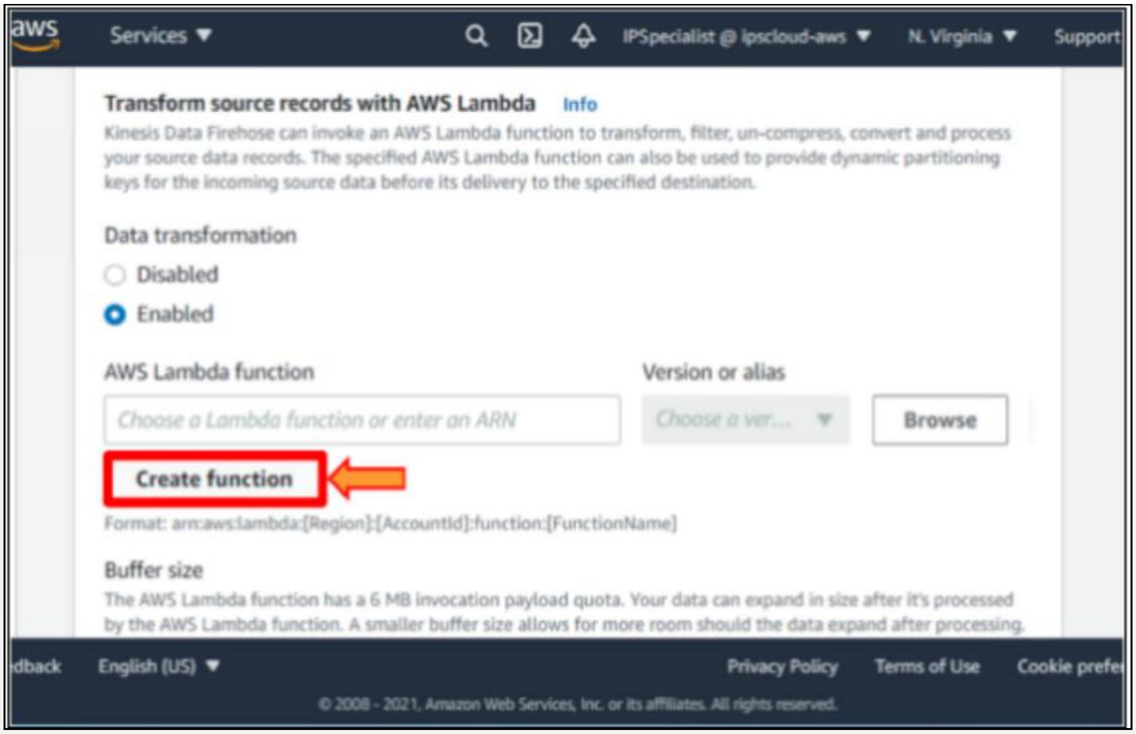
6. Give the delivery stream name **IPS-my-delivery-stream.**



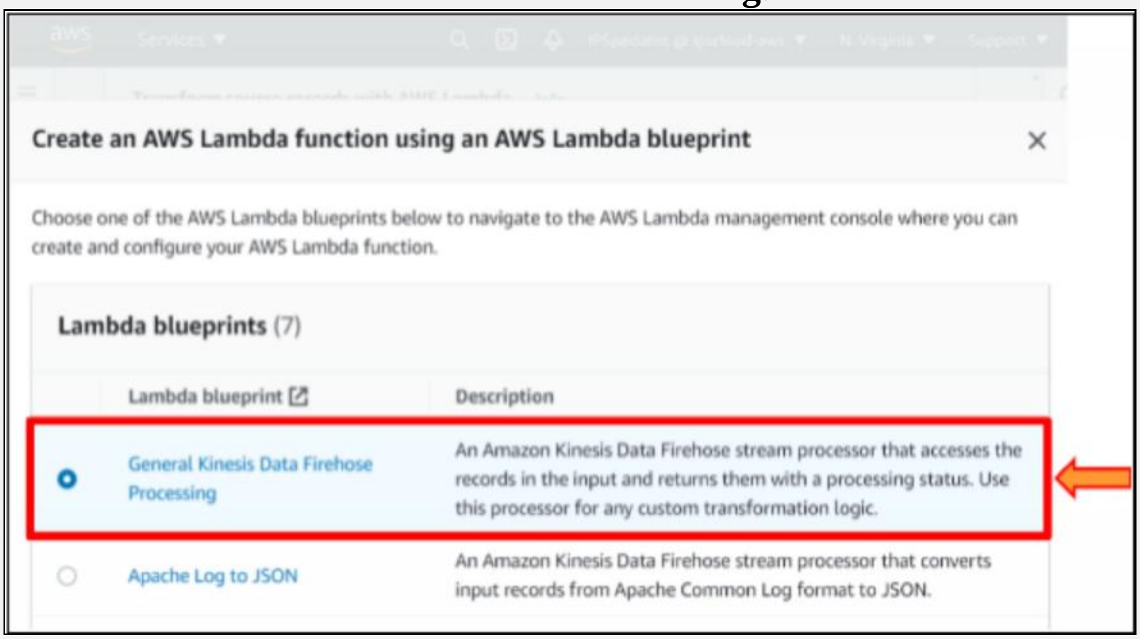
7. Scroll down. Select **Enabled**.



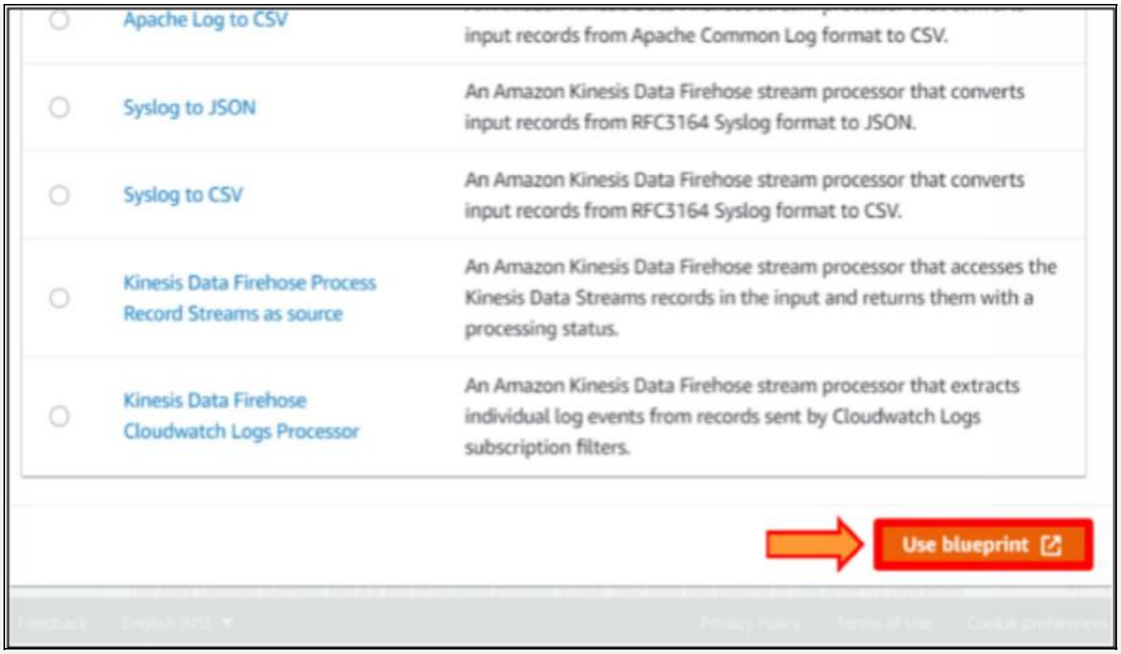
8. Click on the **Create function** button to create the Lambda function.



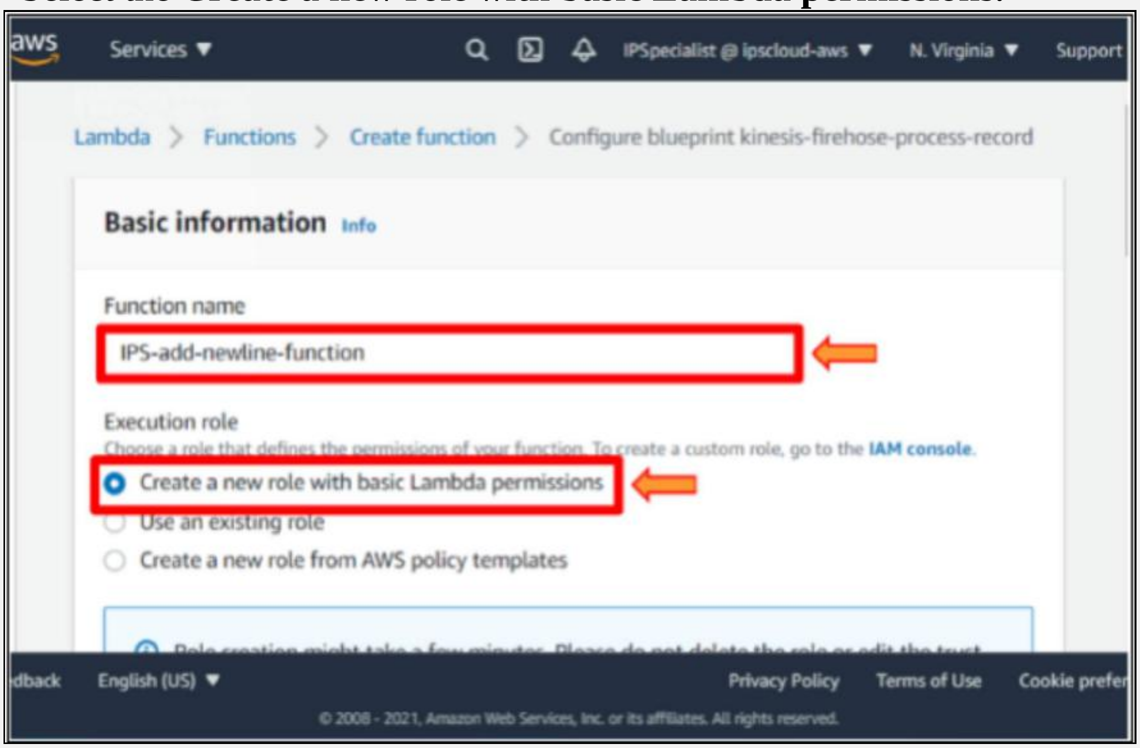
9. Select **General Kinesis Data Firehose Processing.**



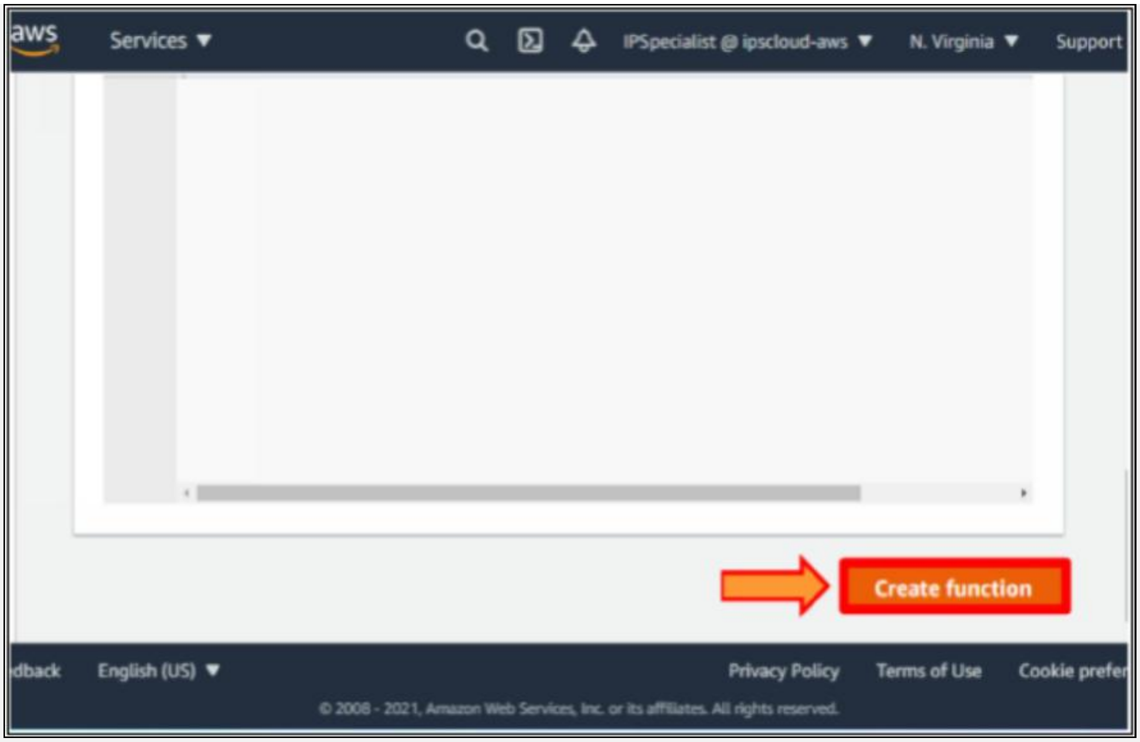
10. Scroll down. Click on the **Use blueprint** button.



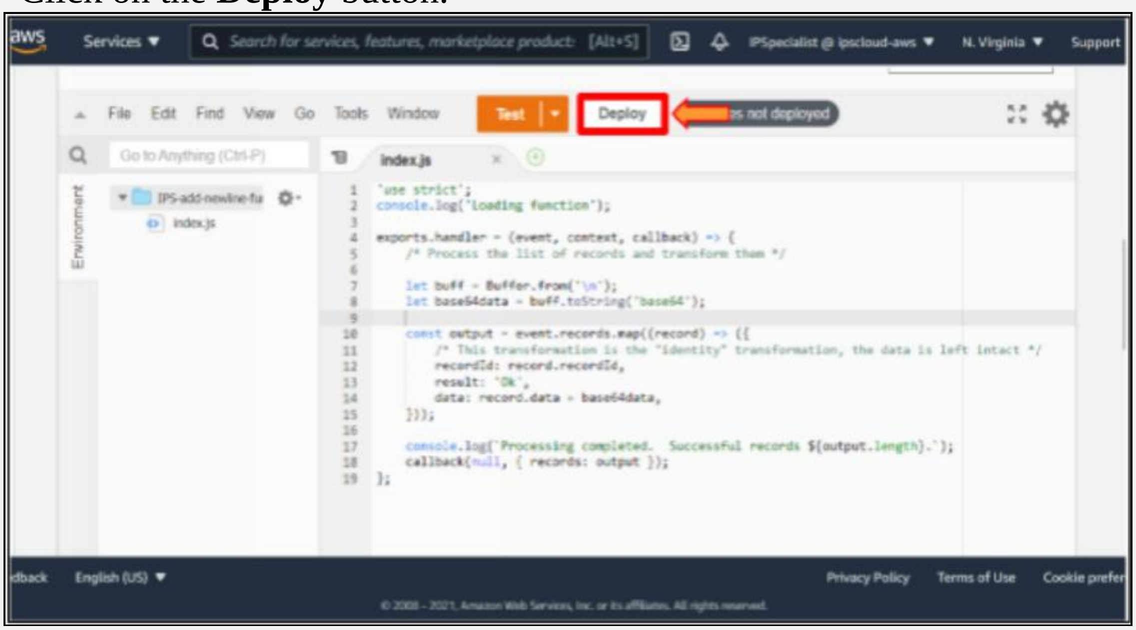
11. Give a name **IPS-add-newline-function.**
12. Select the **Create a new role with basic Lambda permissions.**



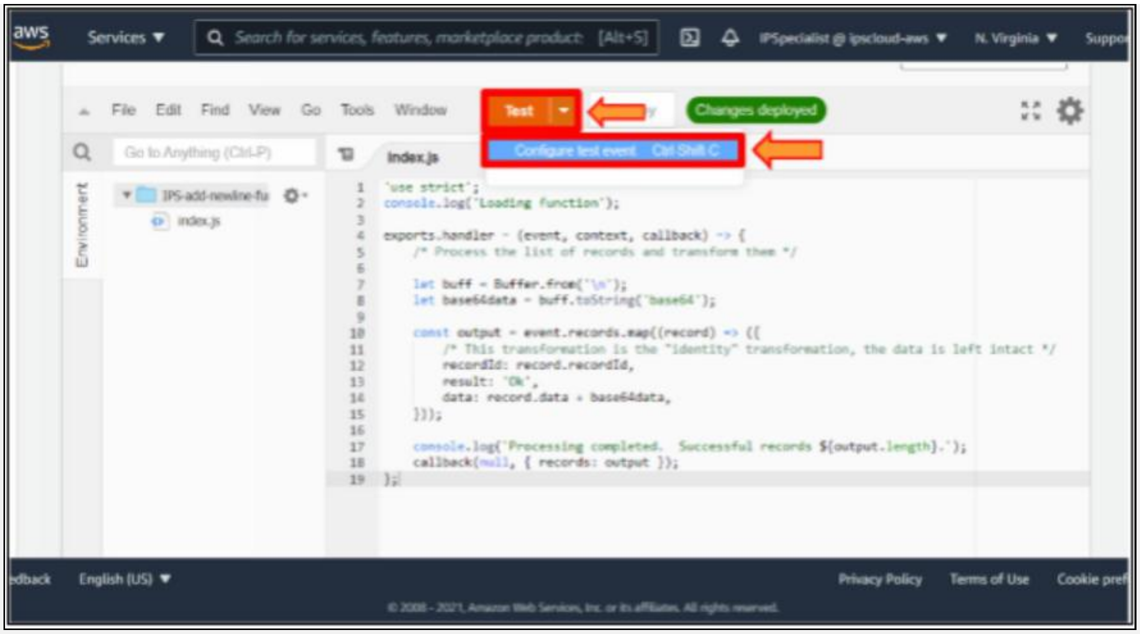
13. Scroll down. Click on the **Create function** button.



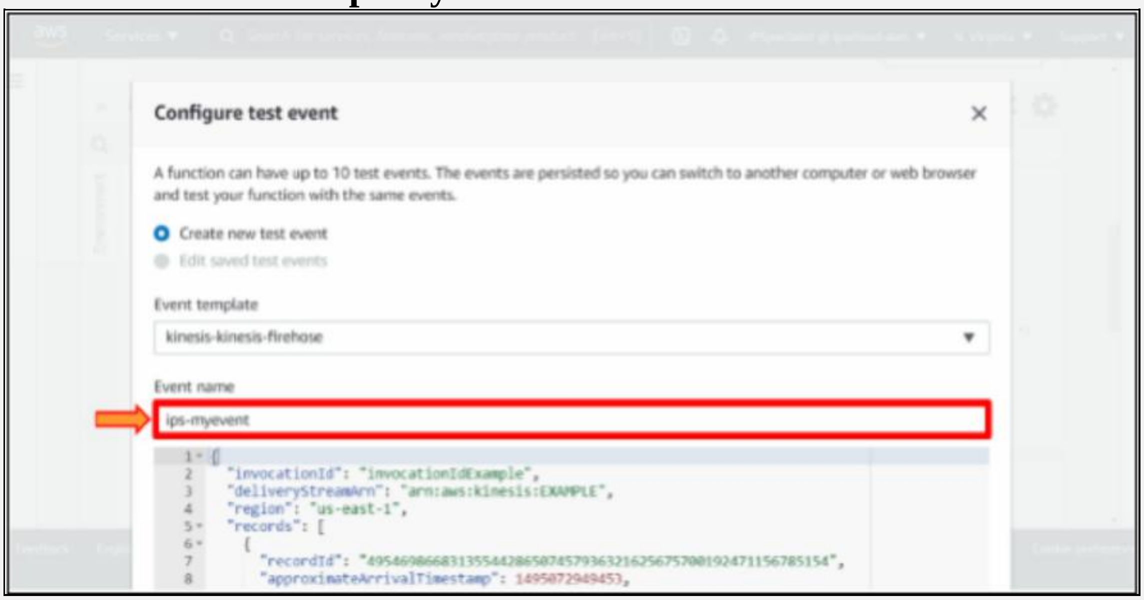
14. Now copy and paste the **index.js** code provided in the following GitHub link: https://github.com/ACloudGuru-Resources/Course_AWS_Certified_Machine_Learning/blob/master/Chapter3/index.js
15. Scroll down and paste the **index.js** code into the code editor.
16. Click on the **Deploy** button.



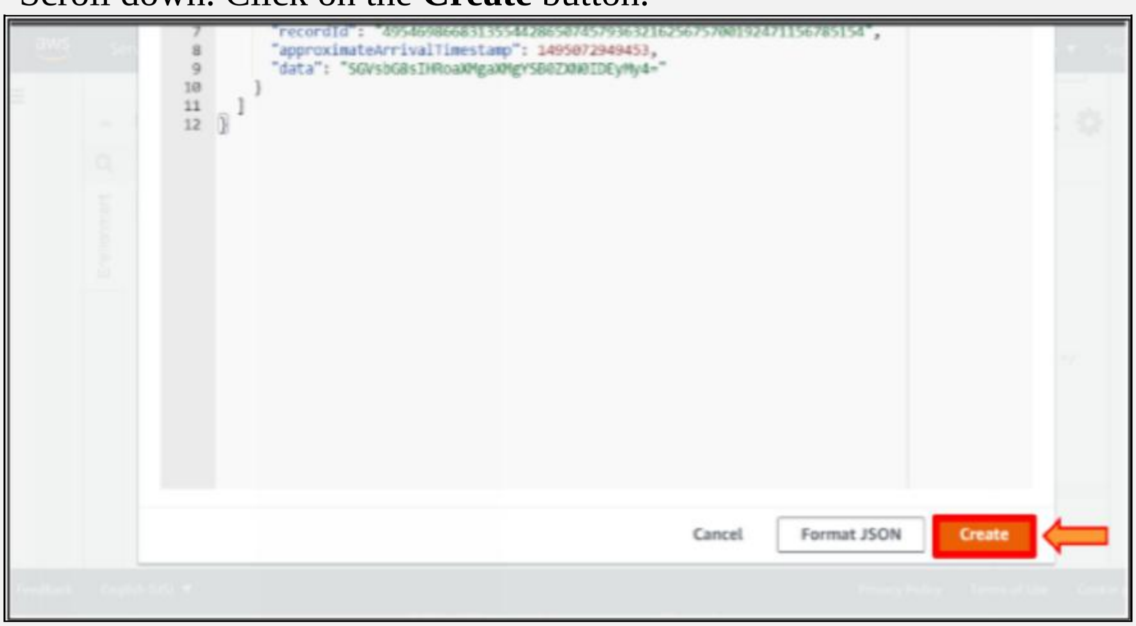
17. Click on the **Test.** Then, click on **Configure test event.**



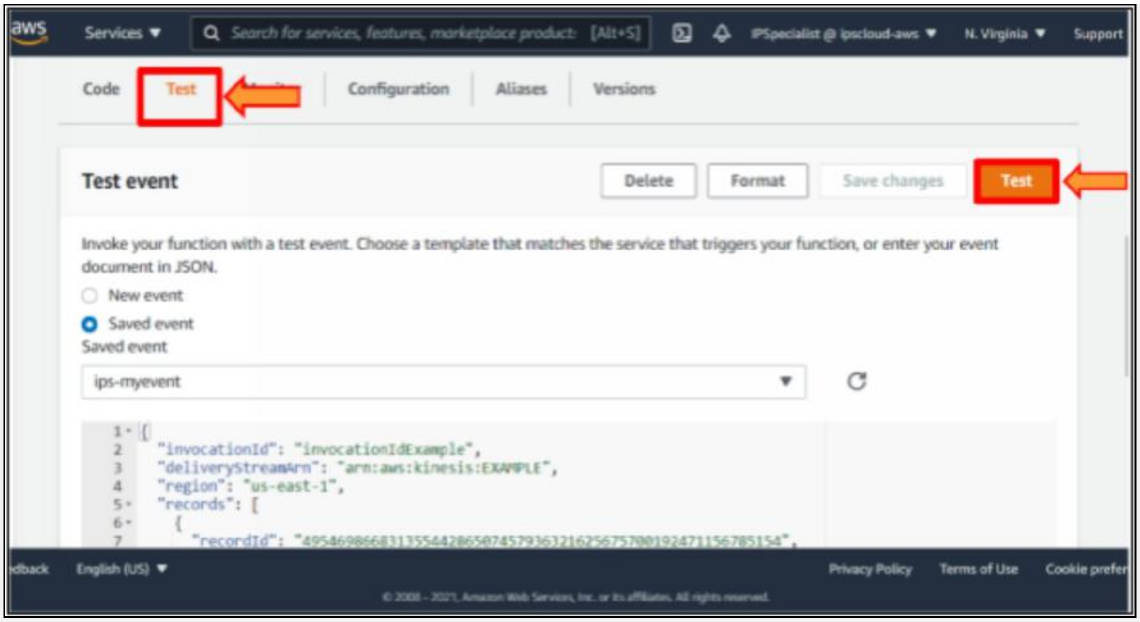
18. Give an event name **ips-myevent.**



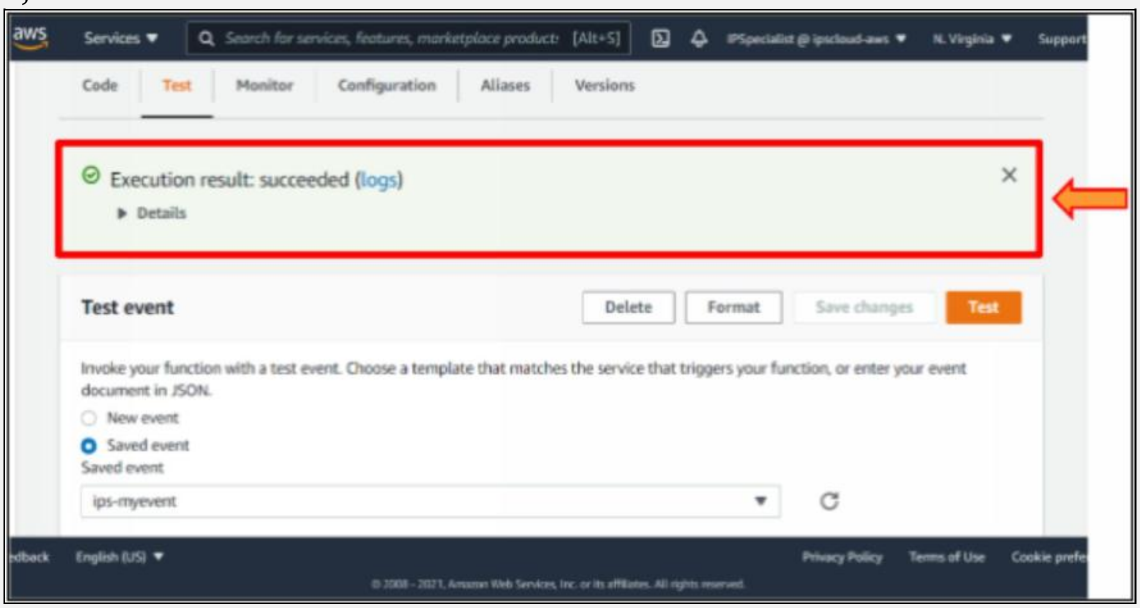
19. Scroll down. Click on the **Create** button.



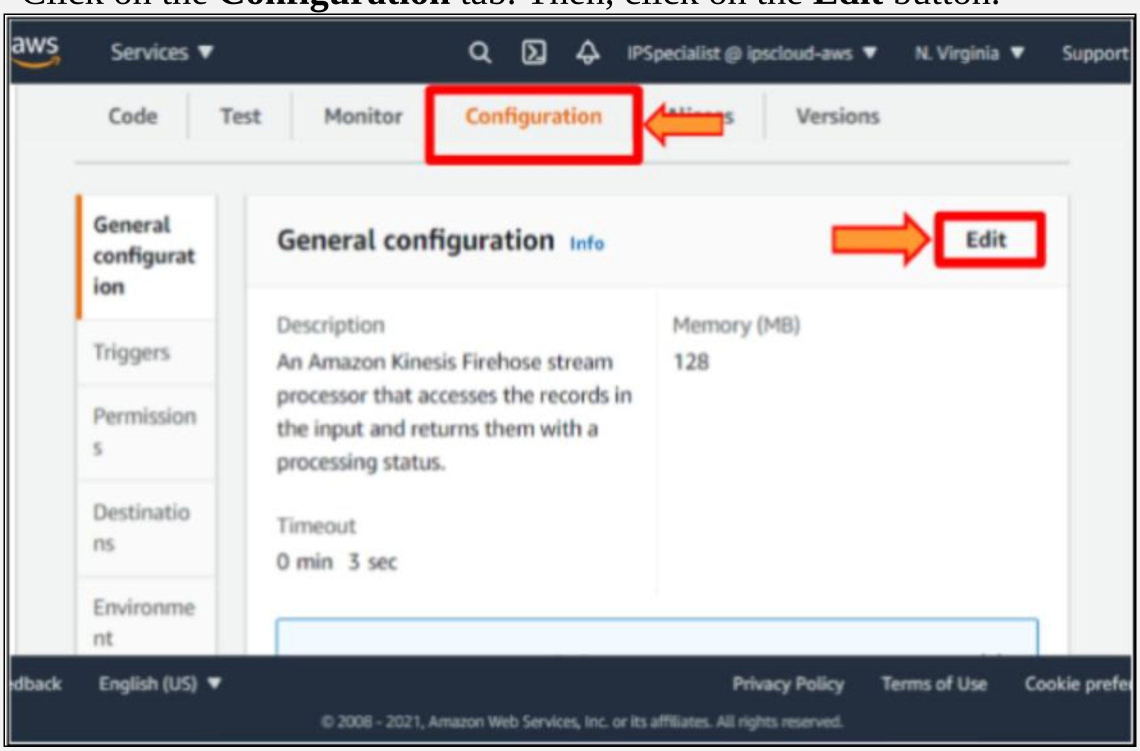
20. Click on the **Test** tab. Then, click on the **Test** button.



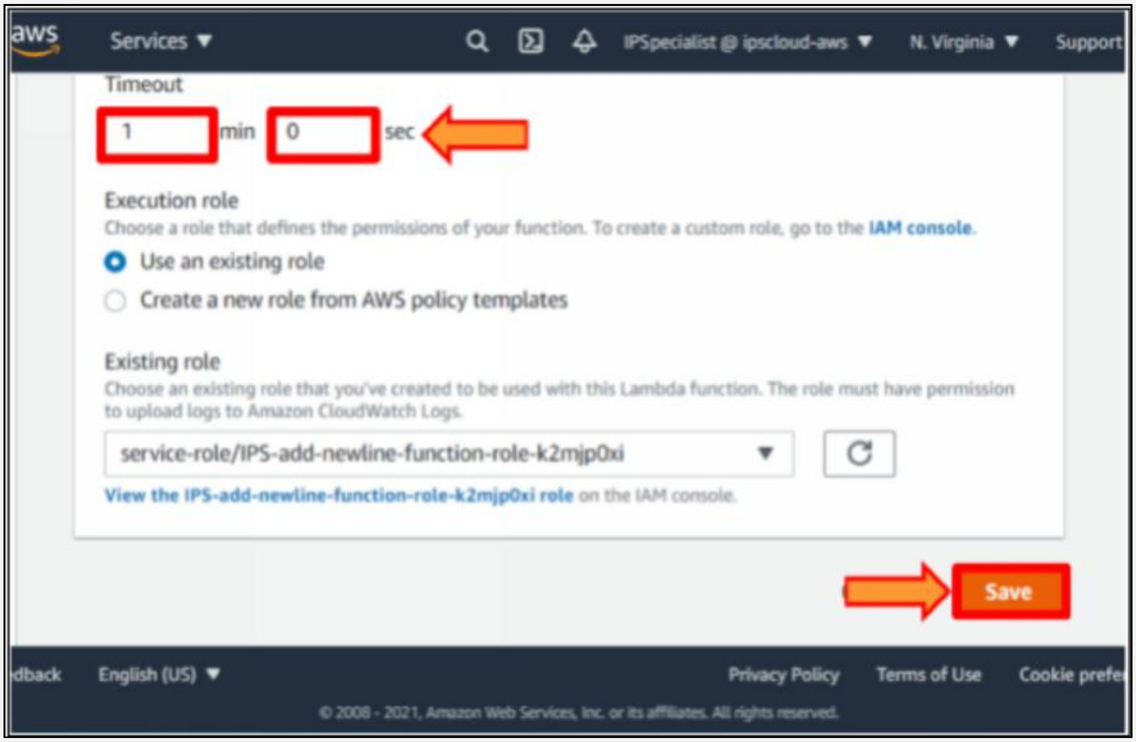
Hence, the execution result is a success.



21. Click on the **Configuration** tab. Then, click on the **Edit** button.

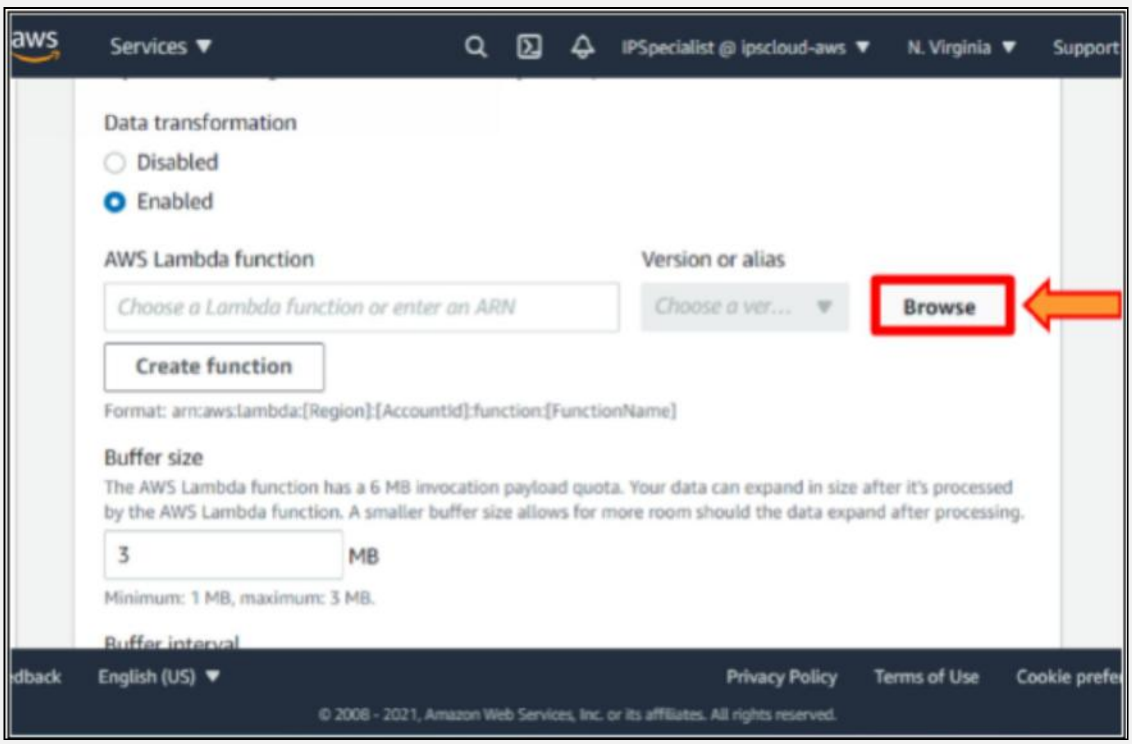


22. Scroll down and change the **time out** to **1** min. Then, click on the **Save** button.

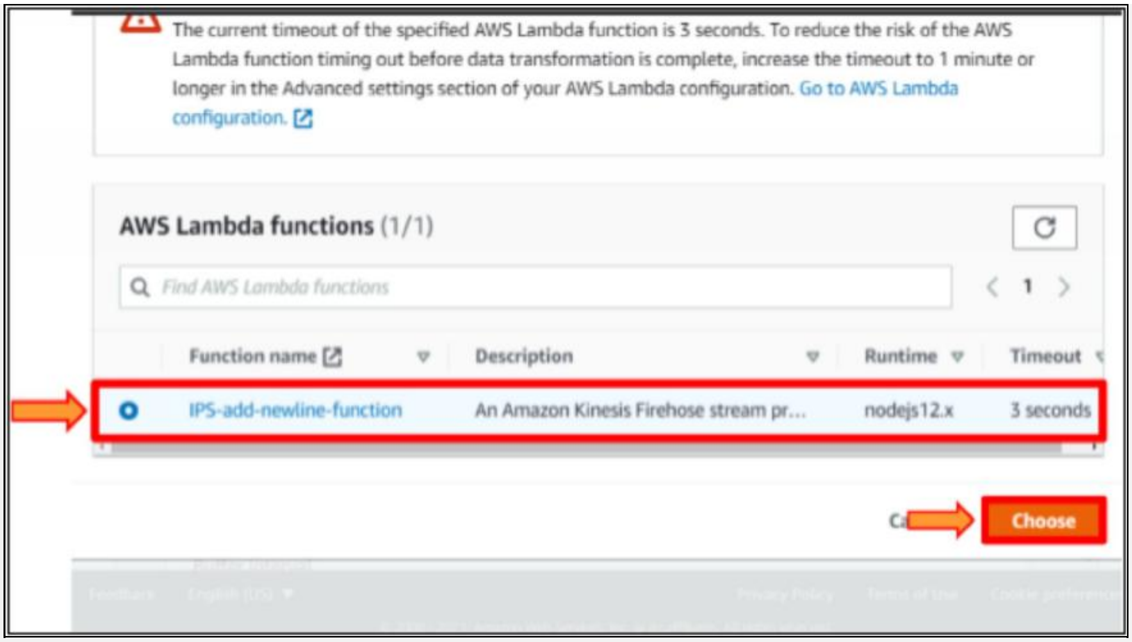


23. Now, go back to the **Create a delivery stream** page.

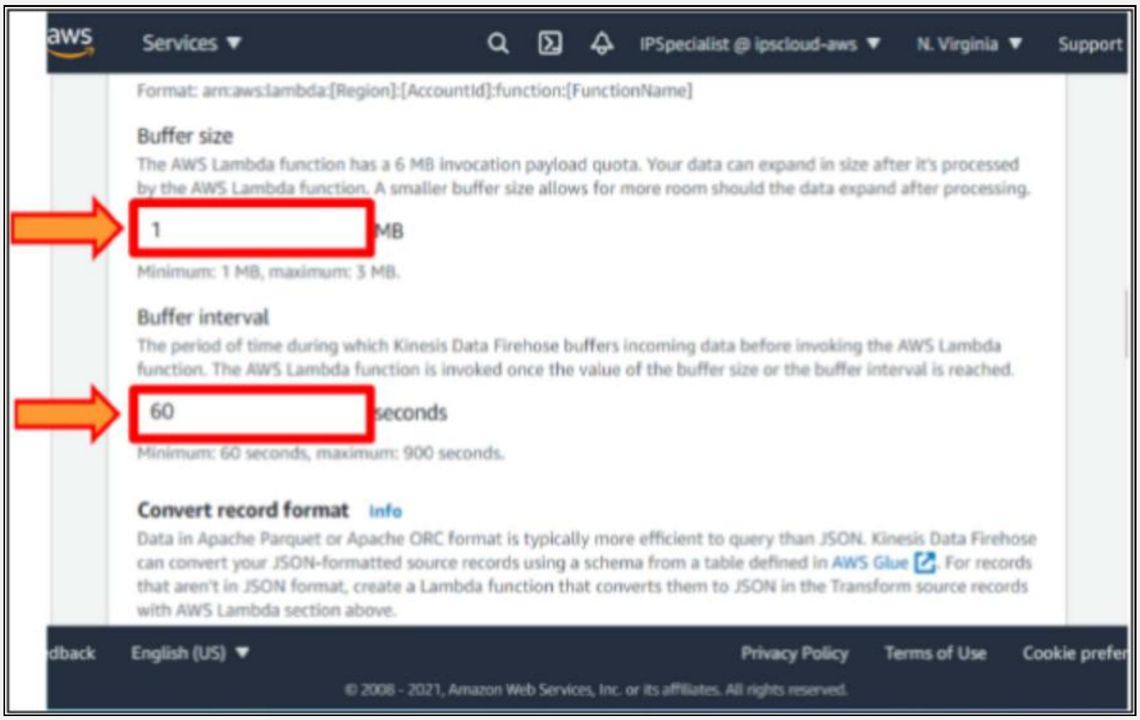
24. Click on the **Browse** button to select the Lambda function.



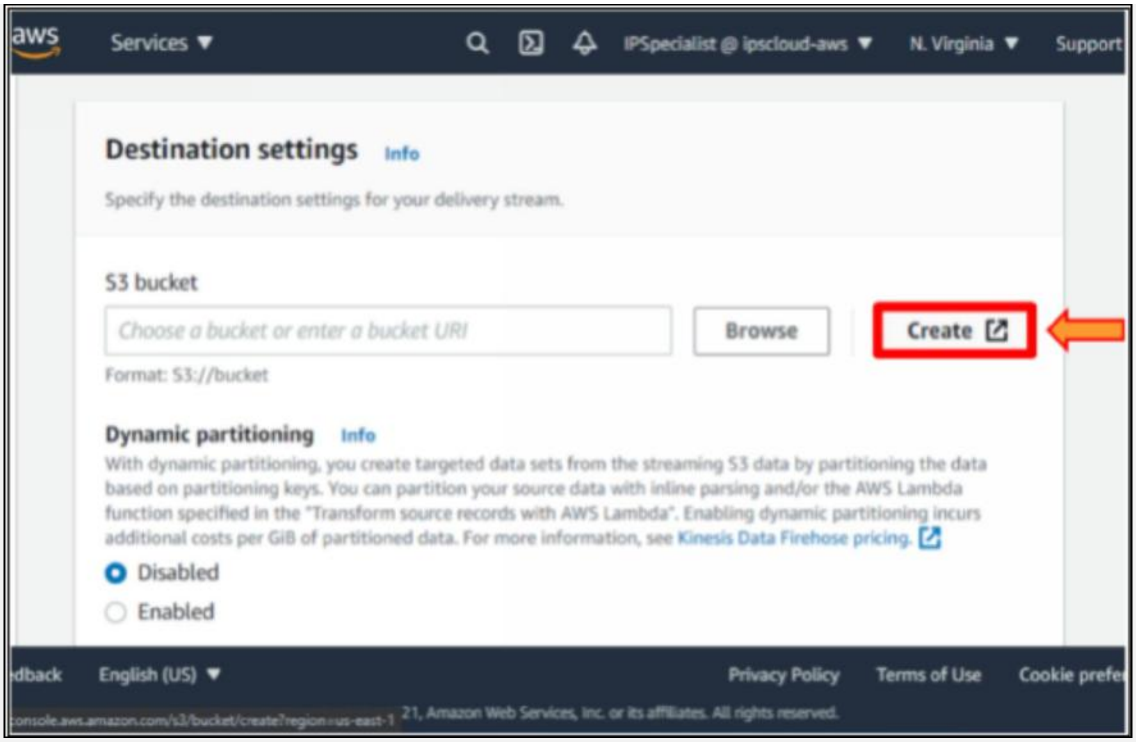
25. Select the **IPS-add-newline-function.** Then, click on the **Choose** button.



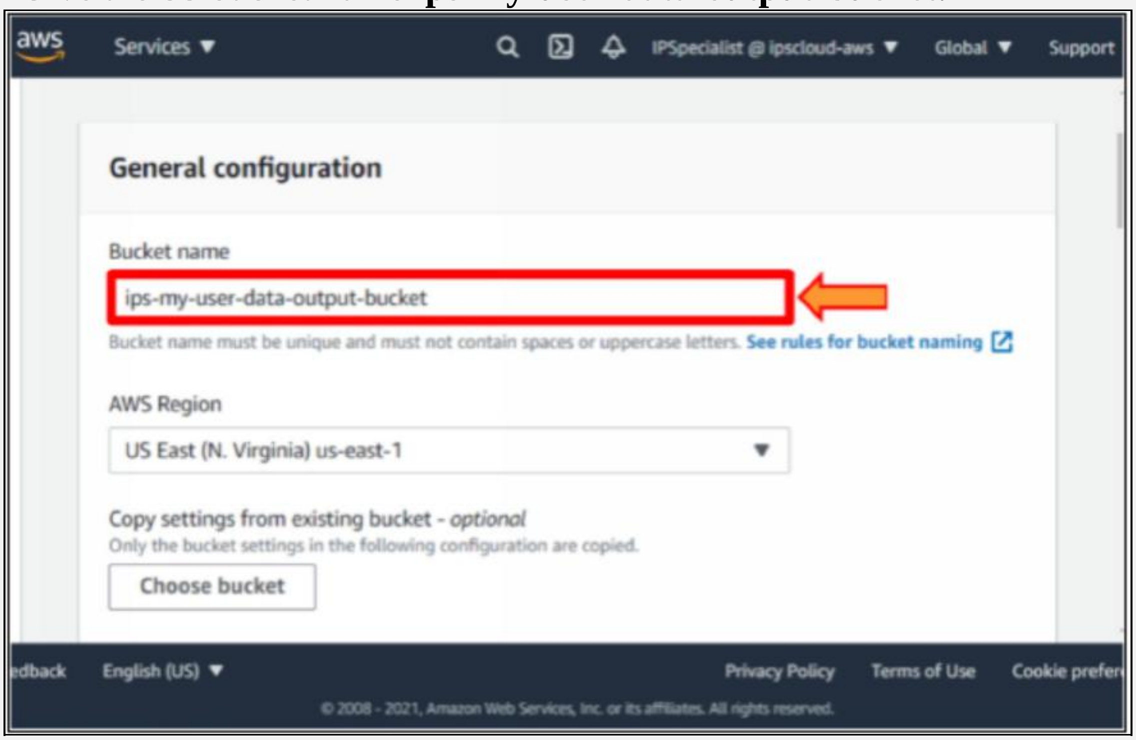
26. Set the Buffer size to **1** MB.
27. Set the Buffer interval to **60** sec.



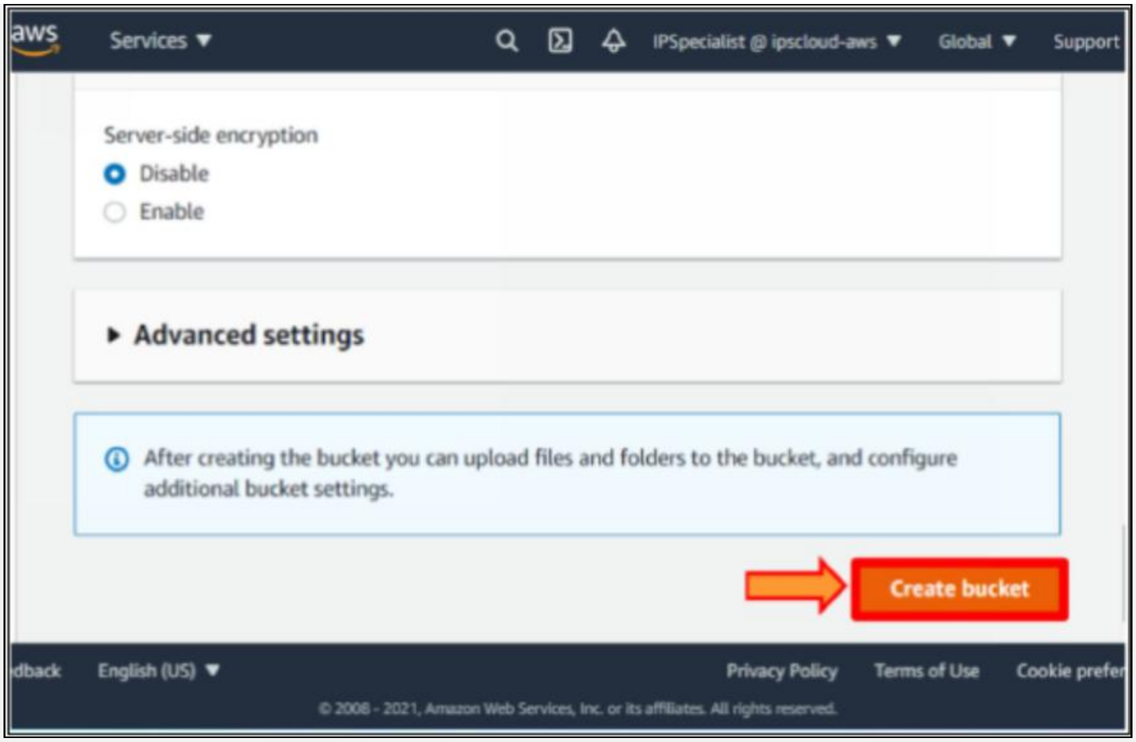
28. Click on the **Create** button to create an S3 bucket.



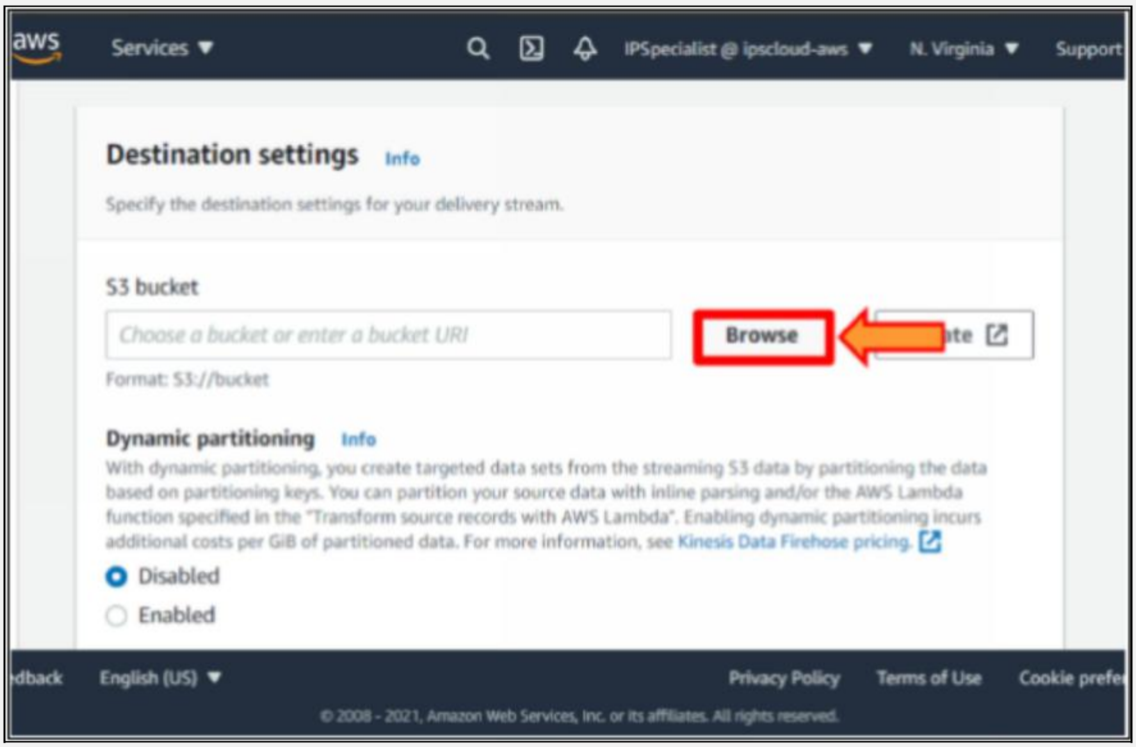
29. Give the S3 bucket name **ips-my-user-data-output-bucket.**



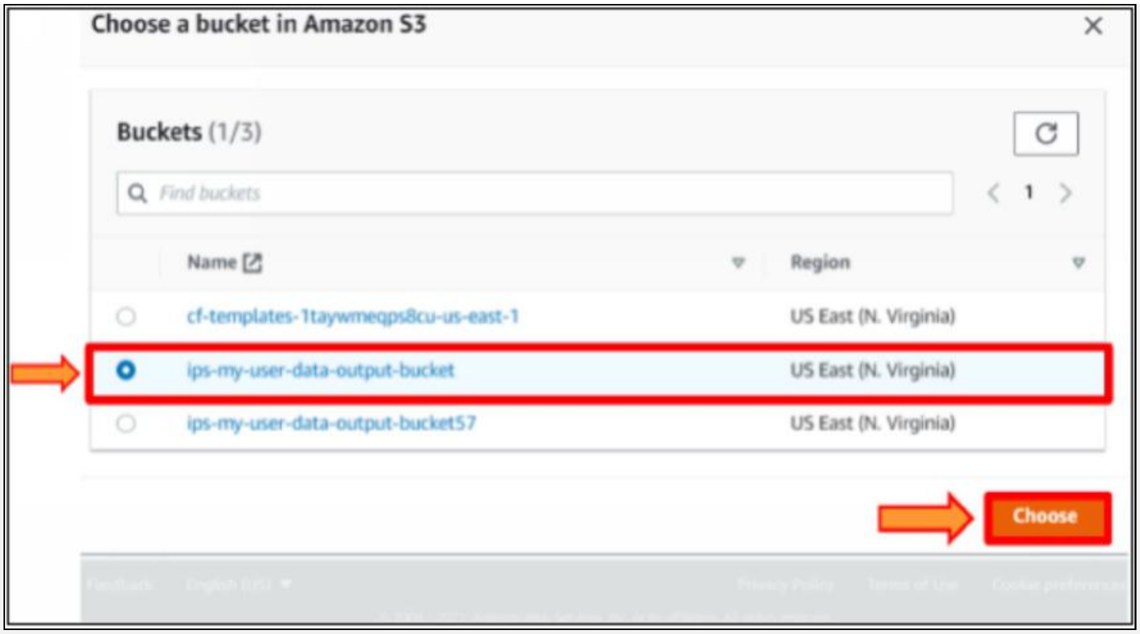
30. Click on **Create bucket.**



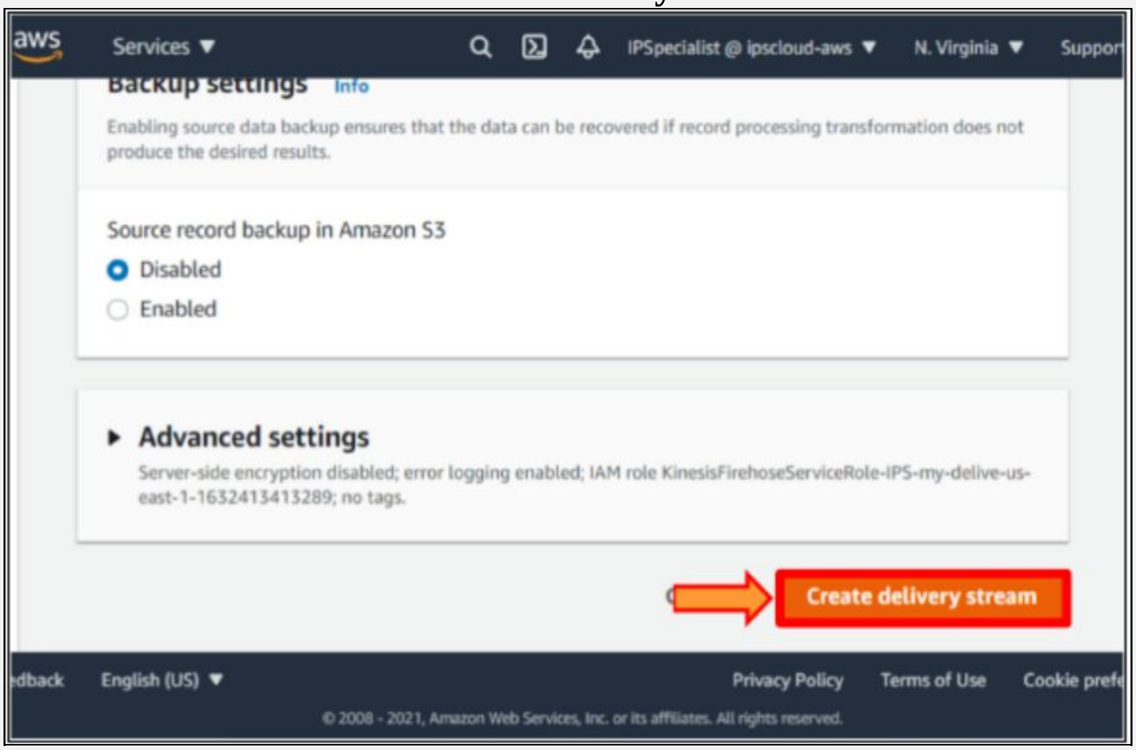
31. Click on the **Browse** button.



32. Select the **ips-my-user-data-output-bucket.** Then, click on the **Choose** button.

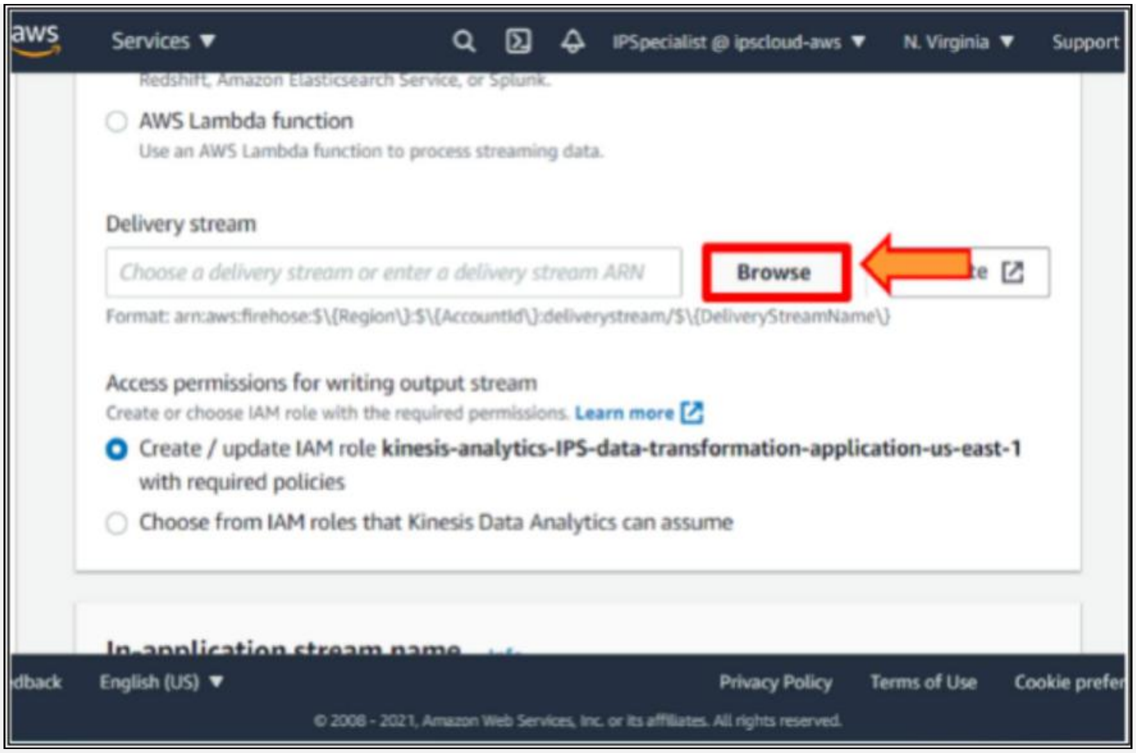


33. Scroll down. Click on the **Create delivery stream** button.

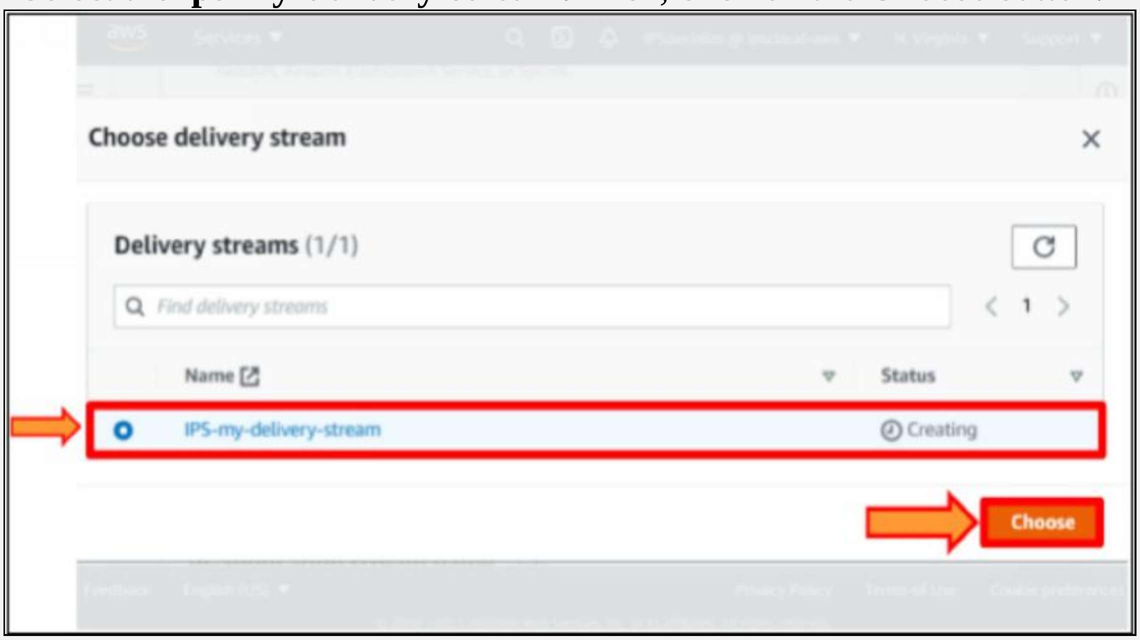


Hence, the delivery stream has been created.

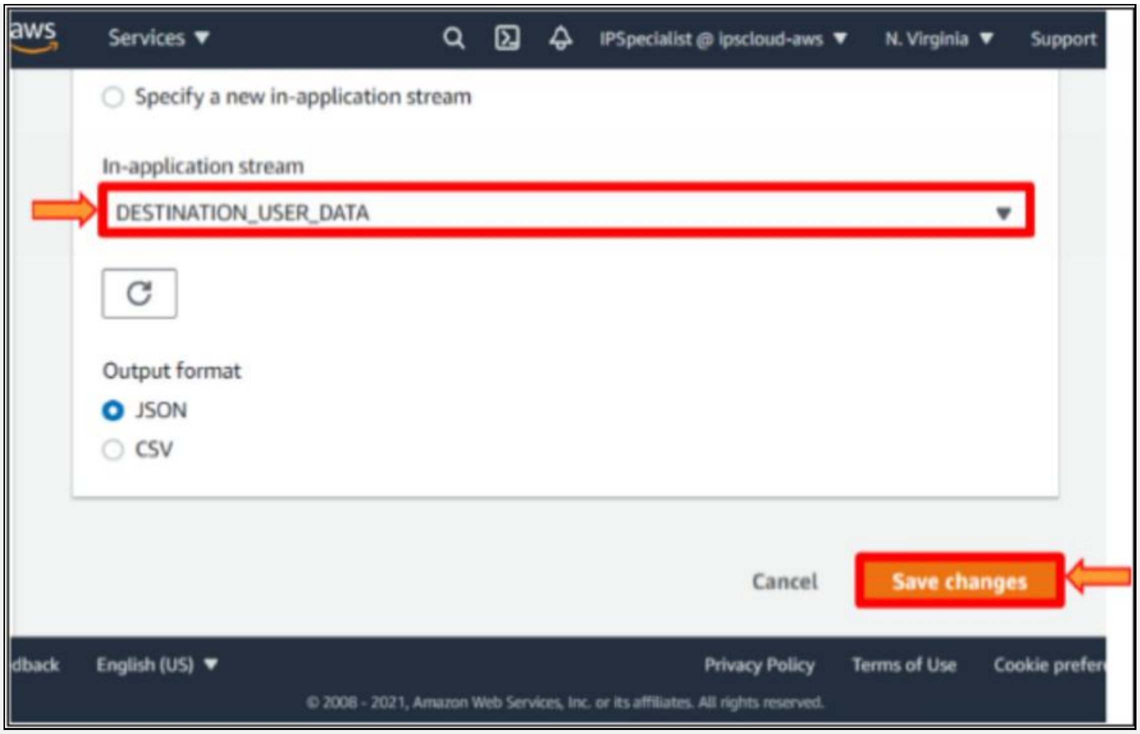
34. Now go to the **Configure destination for IPS-data-transformation-application**.
35. Click on the **Browse** button to select the delivery stream.



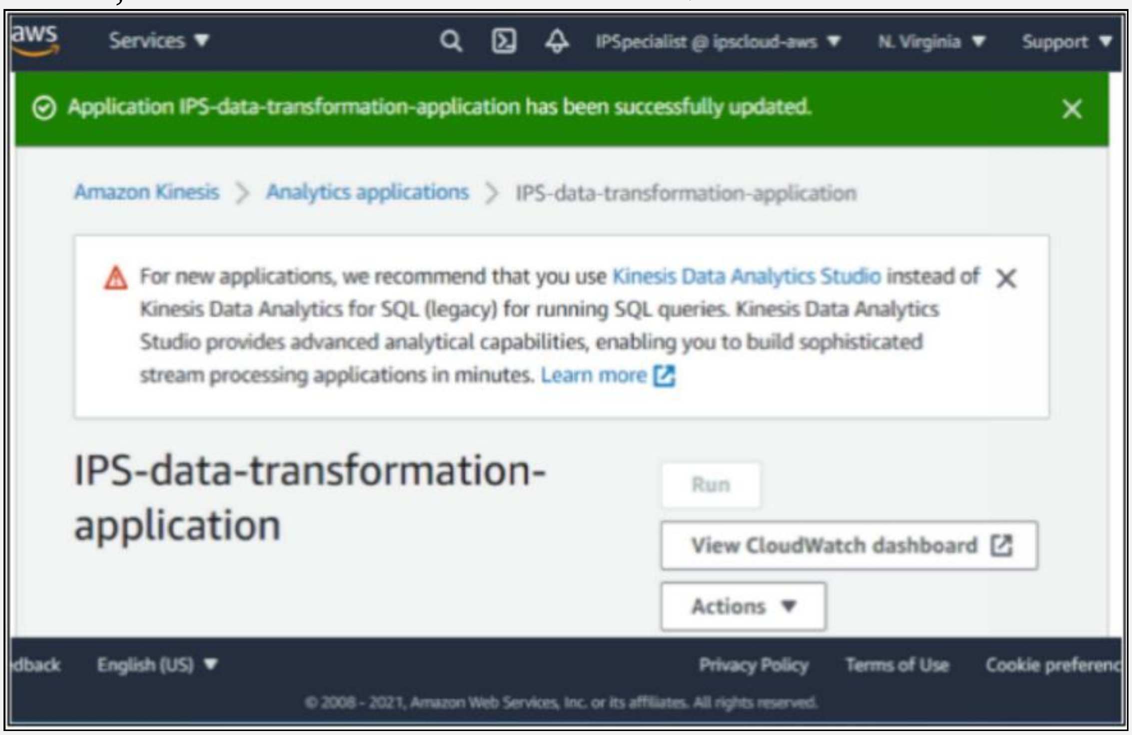
36. Select the **ips-my-delivery-stream.** Then, click on the **Choose** button.



37. Scroll down. Select the **DESTINATION_USER_DATA.**
38. Then, click on the **Save changes** button.

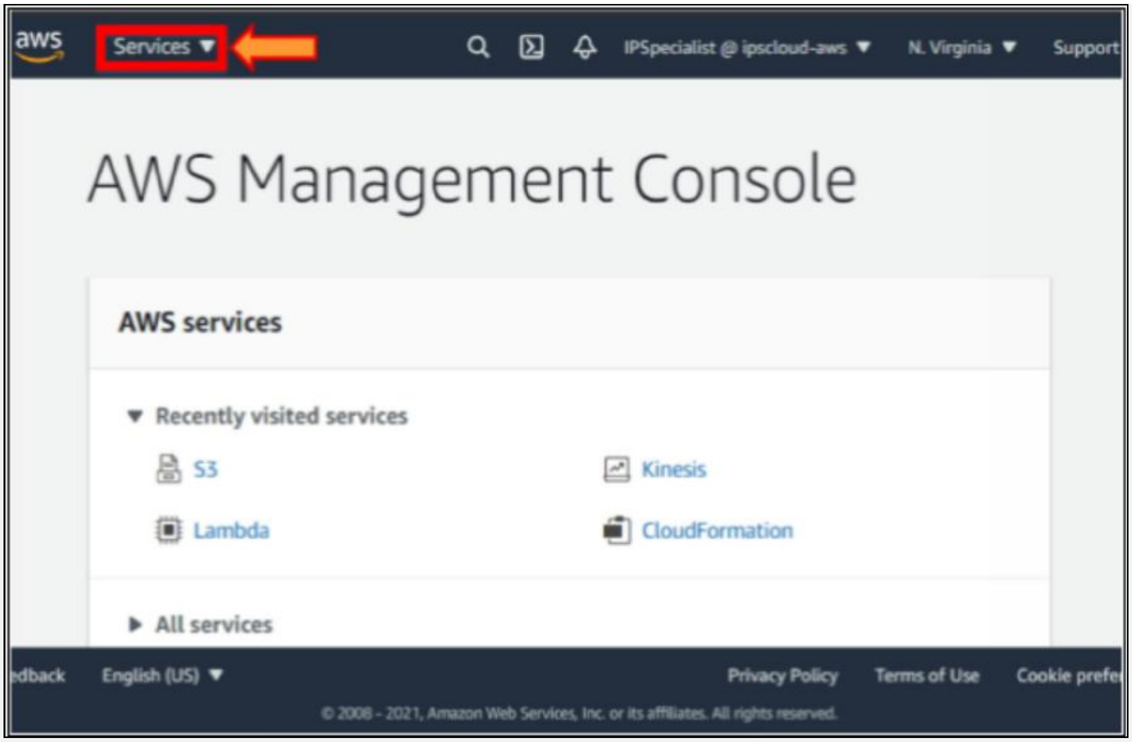


39. Hence, the Kinesis Data Firehose is created.

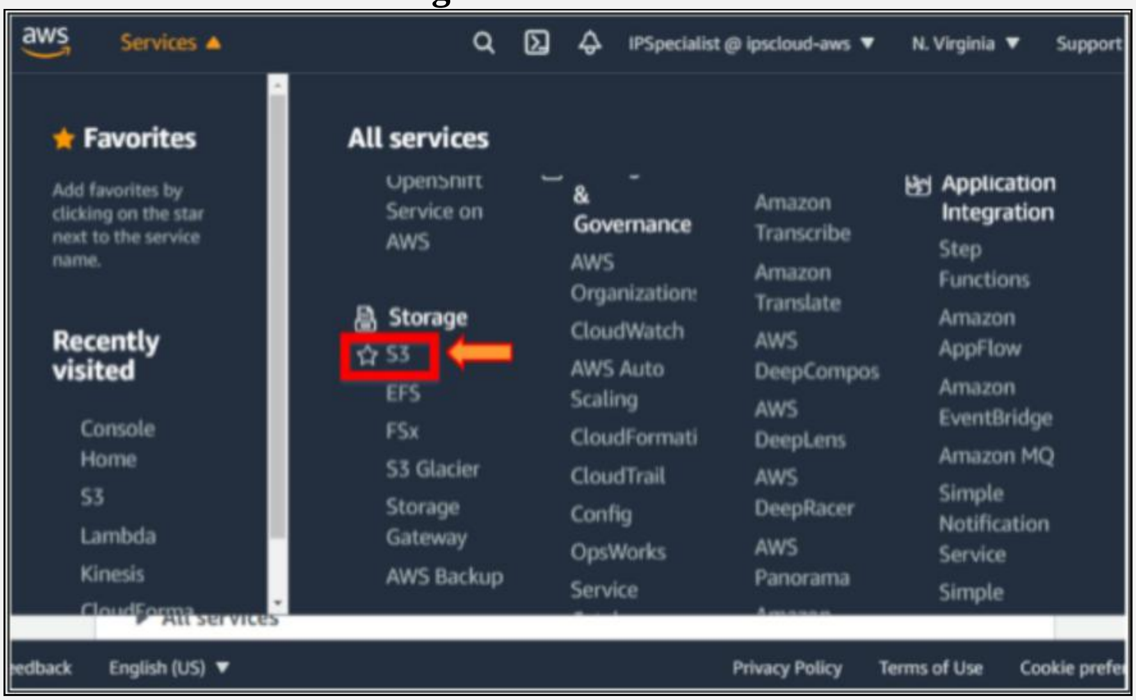


Step 5: View Collected Data

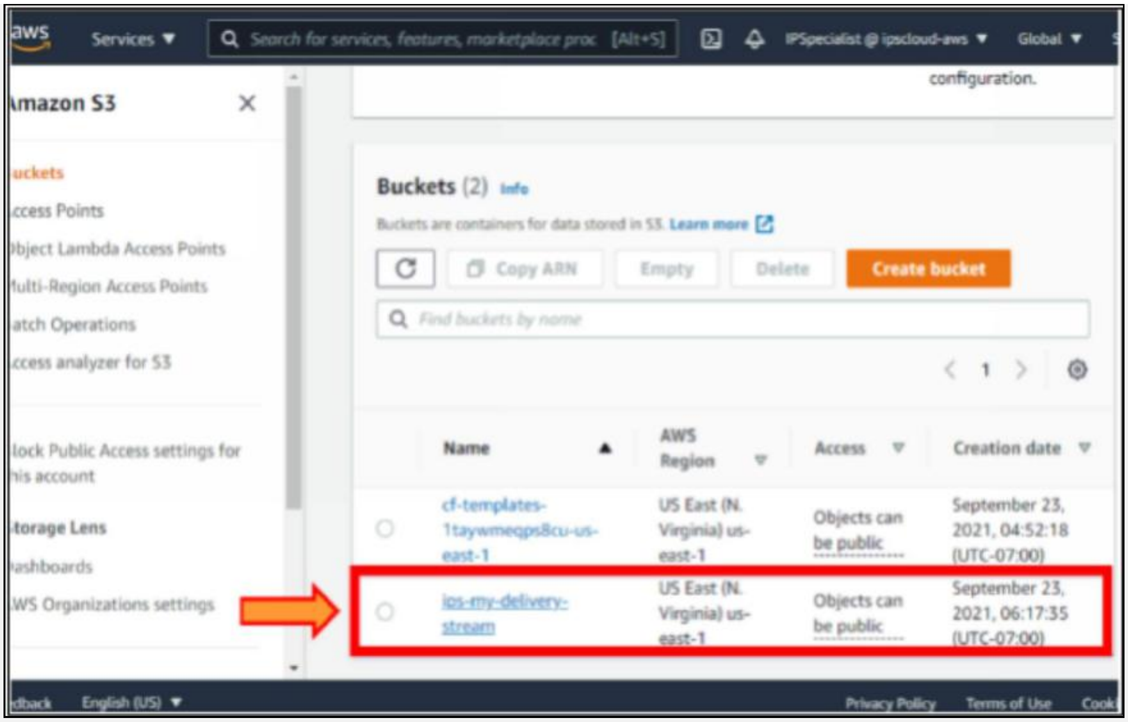
1. Click on **Services.**



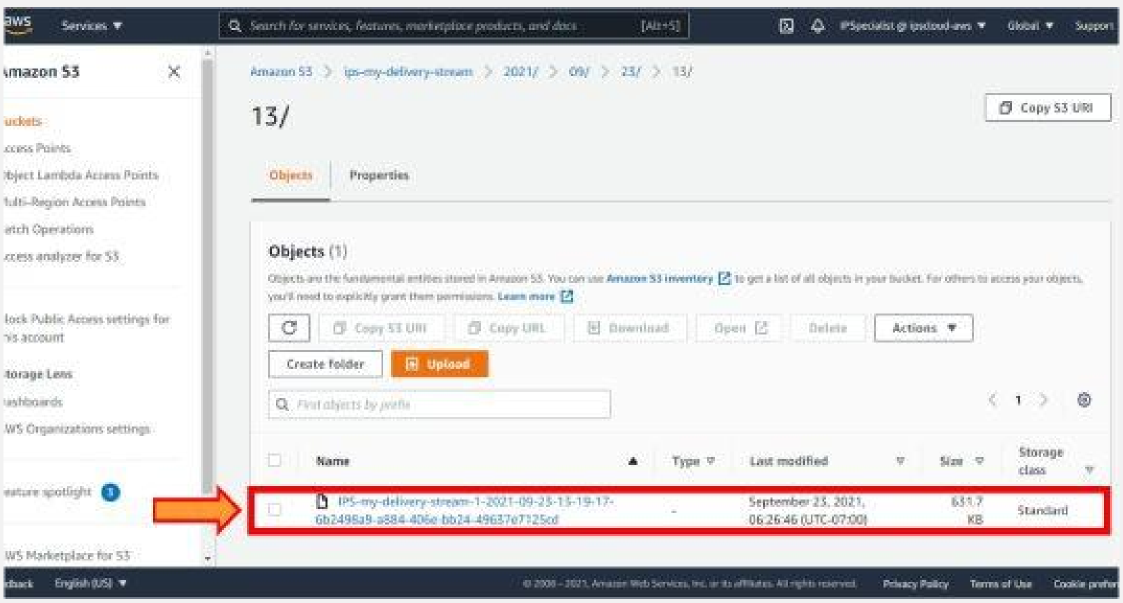
2. Select the **S3** from **Storage.**



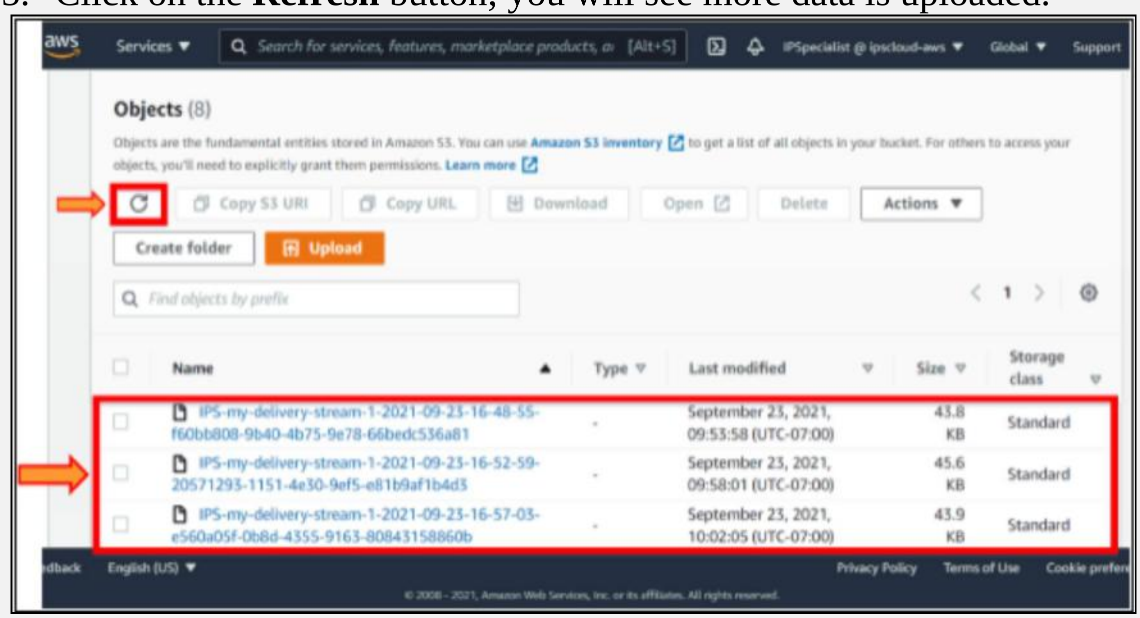
3. Click on **IPS-my-user-data-output-bucket.**



4. Click on the **Folder**. Finally, you can see a file uploaded onto the S3 bucket.



5. Click on the **Refresh** button; you will see more data is uploaded.



Hence, you have successfully streamed data into Kinesis. You have also queried and transformed it with Kinesis Data Analytics. Then, you have to set up a destination using a Kinesis Data Firehose to output the data onto the S3 bucket.

Lab 05: Performing Real-Time Data Analysis with Kinesis

Certifications	AWS Machine Learning - Specialty
Cloud Provider	AWS
Time Required	1 Hr.
Tags	Kinesis, S3, AWS

Lab Prerequisites

- Familiarity with basic AWS Cloud Computing concepts and terminology
- An AWS account with an active subscription

Service Introduction

Kinesis Data Firehose

Kinesis Data Firehose is a solution for streaming ETL. It is the most convenient method for loading streaming data into data storage and analytics tools. It can gather, convert, and load streaming data into Amazon S3, Amazon Redshift, Amazon Elasticsearch Service, and Splunk, allowing you to do near-real-time analytics with your existing business intelligence tools and dashboards. It is a fully managed service that scales automatically to meet your data flow and does not require any ongoing management. It may also batch, compress, and encrypt data before loading it, reducing storage requirements at the destination and enhancing security.

Kinesis Data Analytics

Amazon Kinesis Data Analytics is the most straightforward method for converting and analyzing real-time streaming data using Apache Flink. Apache Flink is an open-source data stream processing framework and engine. Amazon Kinesis Data Analytics makes it easier to create, administer, and integrate Apache Flink applications with other AWS services.

Amazon Kinesis Data Analytics covers everything required to run streaming applications continuously and scales automatically to meet the volume and throughput of your incoming data. There are no servers to operate with Amazon Kinesis Data Analytics, no minimum charge or setup cost, and you pay for the resources your streaming applications utilize.

Amazon Simple Storage Service S3

Amazon S3 is a type of object storage that allows you to store and recover any quantity of data from any location. It is a low-cost storage solution with business resilience, reliability, efficiency, privacy, and infinite expansion.

Amazon S3 provides a simple web service interface for storing and retrieving any quantity of data from any place at any time. Using this service, you may quickly create projects that integrate cloud-native storage because Amazon S3 is easily customizable. You only pay for what you use; you can start small and scale up as needed without sacrificing performance or dependability.

Amazon S3 is also built to be highly adaptable. Instead of finding out how to store their data, Amazon S3 allows developers to focus on innovation. Build a simple FTP system or a complex web application like the Amazon.com retail website; read the same piece of data a million times or only for emergency disaster recovery; store whatever type and amount of data you desire.

Amazon Elastic Compute Cloud (EC2)

Amazon Elastic Compute Cloud (Amazon EC2) is a cloud computing service that gives the durability of computing power. It is intended to make web-scale computing more accessible to IT engineers.

Amazon EC2 offers “compute” in the cloud, much as Amazon Simple Storage Service (Amazon S3) enables “storage” in the cloud. The easy web service interface of Amazon EC2 allows you to obtain and configure capacity quickly. It lets you control your computer resources entirely and enable you to run on Amazon’s tried-and-true computing infrastructure. The time to buy and boot new Amazon EC2 server instances is reduced to minutes, allowing you to scale up quickly and down capacity as your computing needs change. Amazon EC2 revolutionizes computing economics by enabling you to pay only for the resources you utilize.

Case Study Enterprise Biopharmaceutical – Neumora Therapeutics

Business Challenge

Neumora Therapeutics has thousands of users interacting with the organization application. The organization gives Neumora Therapeutics a task of analyzing the data in real-time. The organization also wants you to load real-time data into a cloud storage destination as a backup copy.

Proposed Solution

The solution is you use AWS services to automate your task and analyze the stored data in real-time within minutes. For storing the data in real-time, you use Amazon Simple Storage S3. You use Amazon Kinesis Data Analytics to run SQL queries to analyze the stored data in real-time. To store real-time analyzed data at any storage destination, you use Amazon Kinesis Firehose.

Before starting the lab, launch an EC2 instance and install the related python packages used in the Python script provided in the following GitHub link:

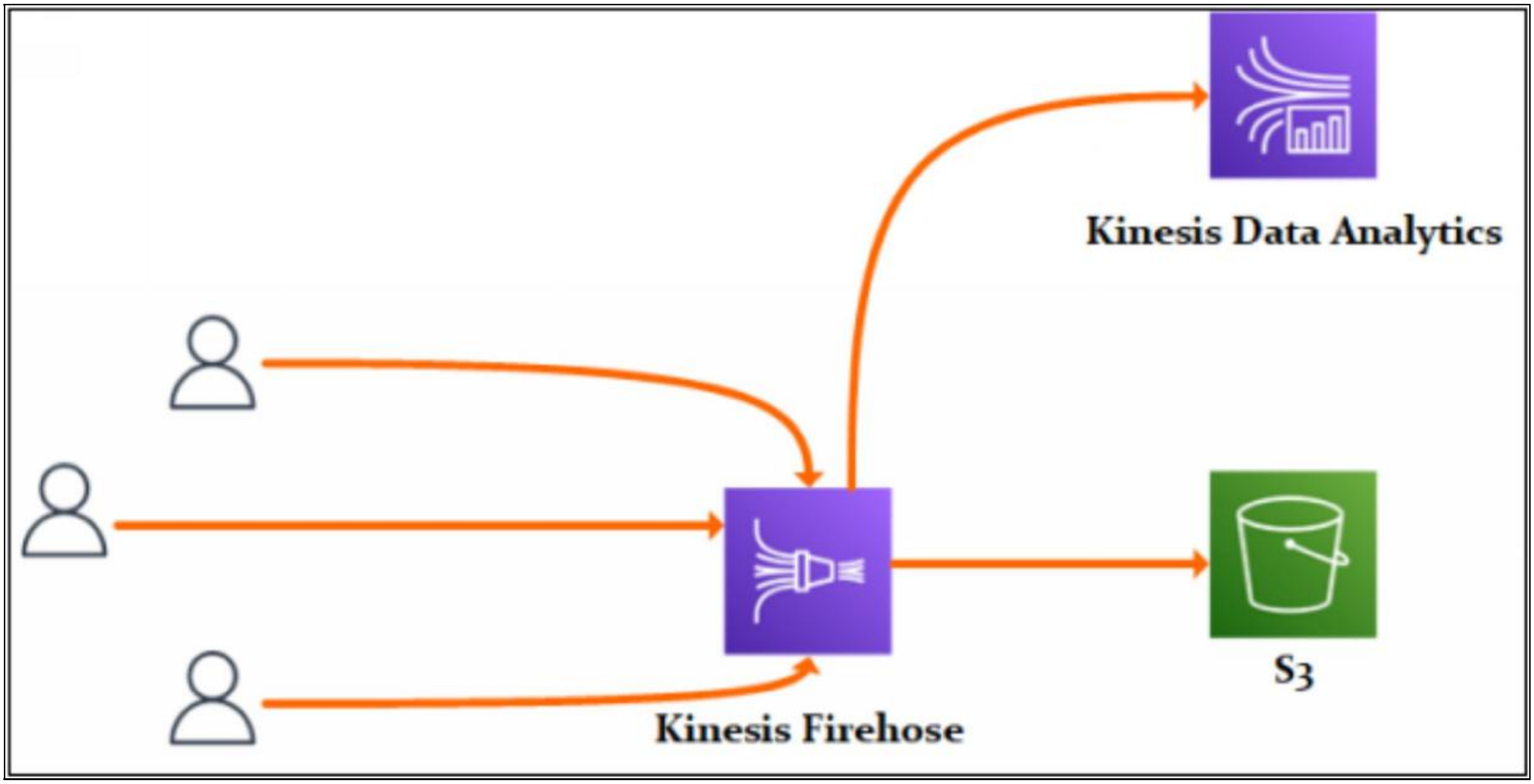
https://github.com/12920/IPSpecialist01/blob/main/Course_AWS_Certified_Machine_Learning-master%20(1).zip

The following Python script is used to generate the dummy data in this lab (Lab 03-02).

raw.githubusercontent.com/linuxacademy/content-aws-mls-c0

```
#!/usr/bin/env python
import boto3
from faker import Faker
import random
import time
import json

DeliveryStreamName = 'captains-kfh'
client = boto3.client('firehose')
fake = Faker()

captains = [
    "Jean-Luc Picard",
    "James T. Kirk",
    "Han Solo",
    "Kathryn Janeway",
    "Malcolm Reynolds",
    "William Adama",
    "Turanga Leela",
    "Jacob Keyes",
    "Wilhuff Tarkin",
    "Christopher Pike",
    "David Bowman",
    "The Doctor",
    "John Robinson",
    "Khan Noonien Singh"
];

record = {}
while True:

    record['user'] = fake.name();
    if random.randint(1,100) < 5:
        record['favoritecaptain'] = "Neil Armstrong";
```

Lab Diagram



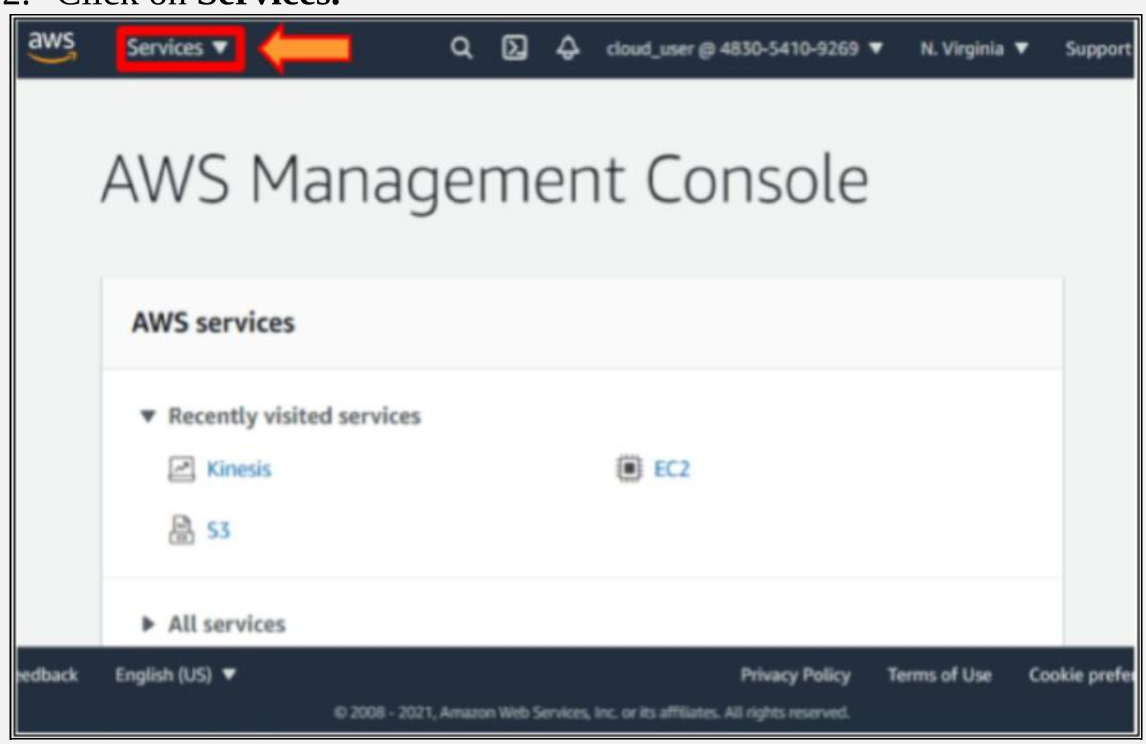
Implementation Steps

1. Create Kinesis Data Firehose.
2. Generate Data to the Stream.
3. Create Kinesis Data Analytics.

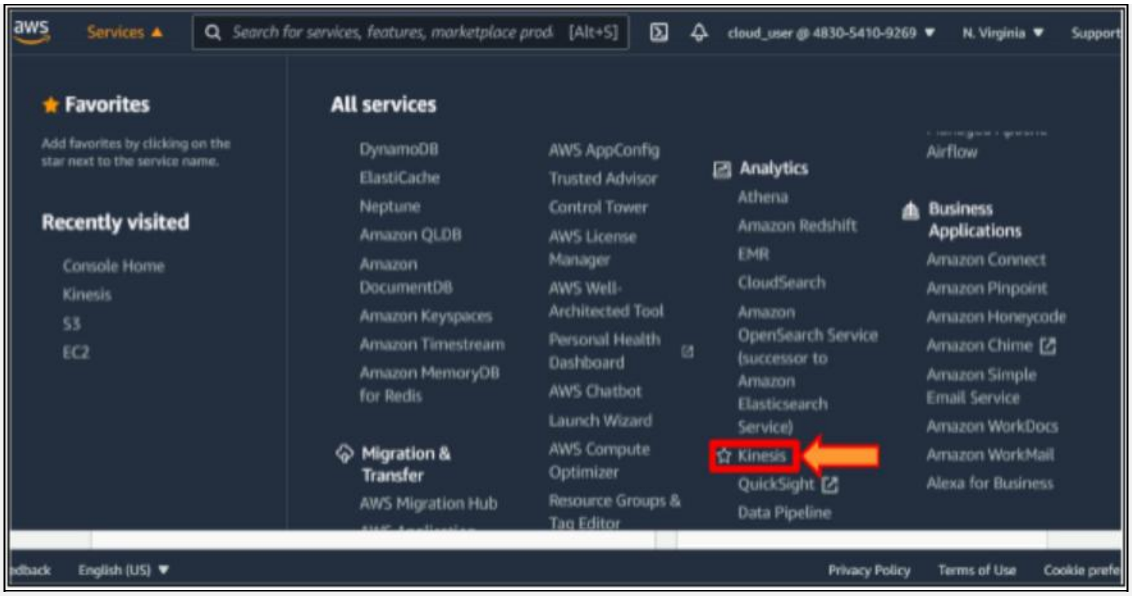
Solution

Step 1: Create Kinesis Data Firehose

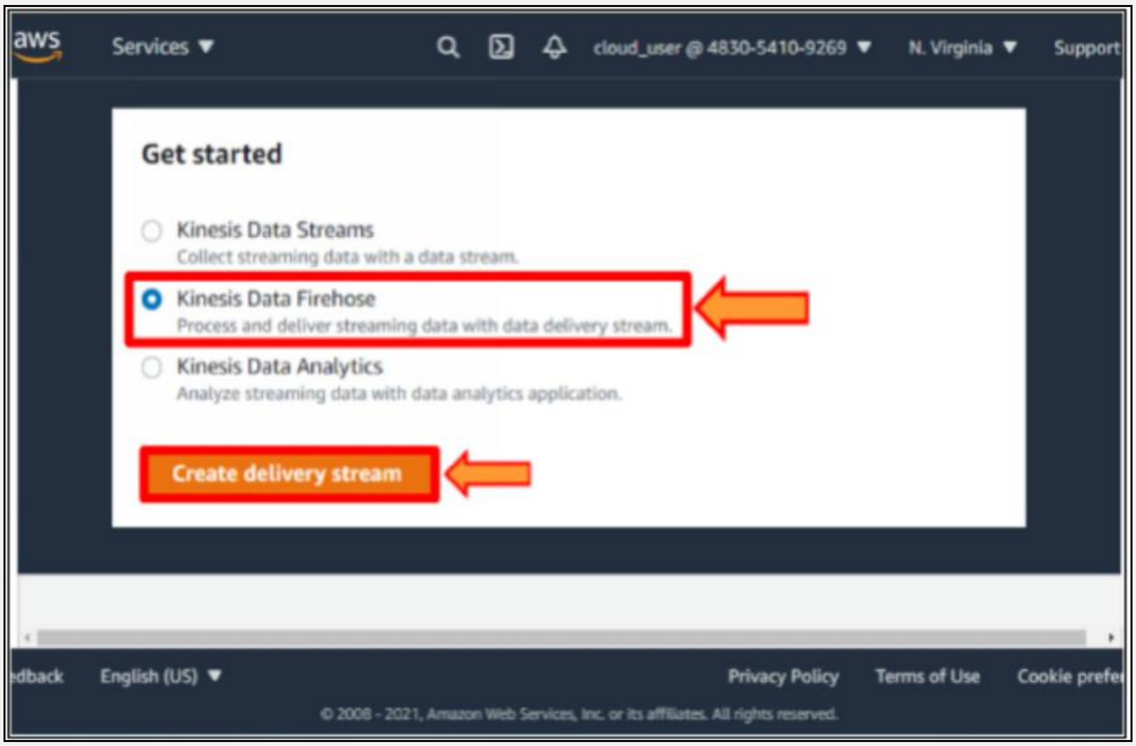
1. Log in to the **AWS Console.**
2. Click on **Services.**



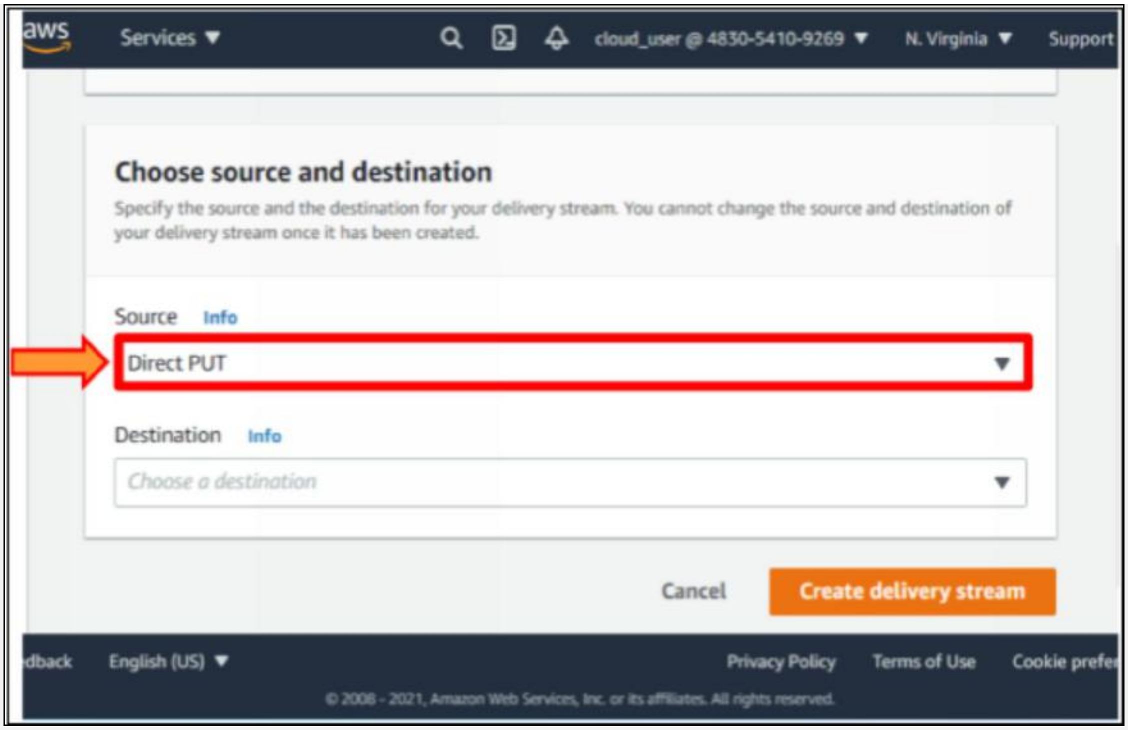
3. Select **Kinesis** from **Analytics**.



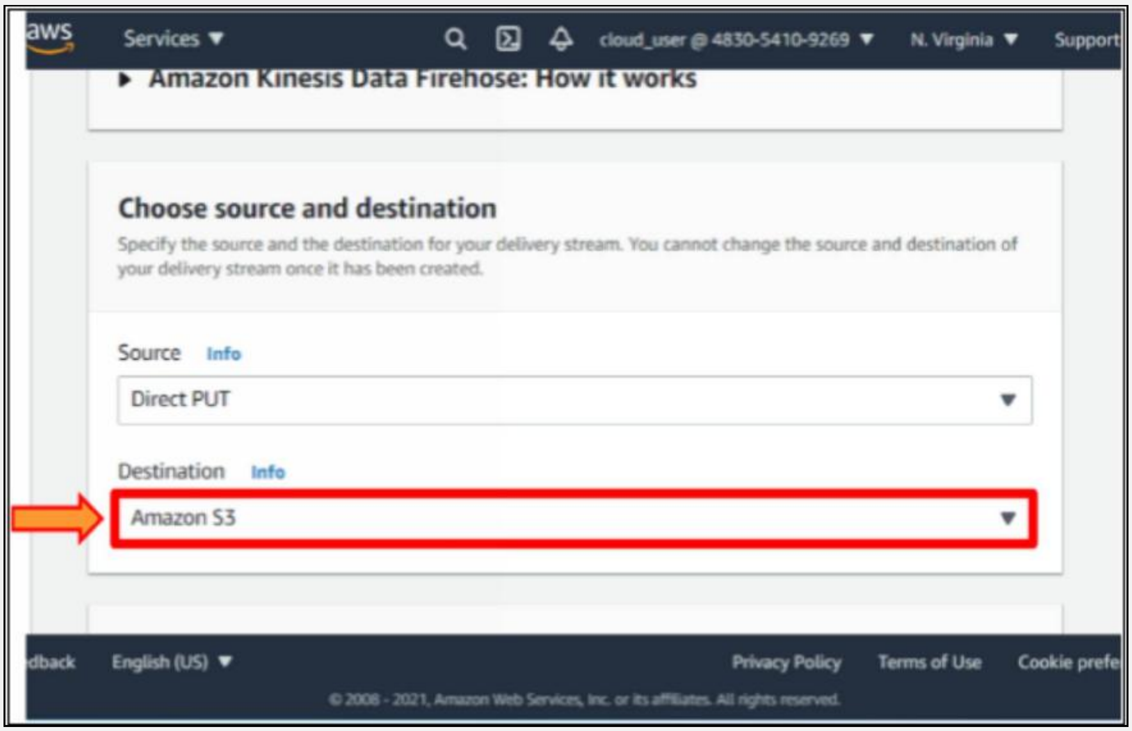
4. Select **Kinesis Data Firehose.** Then, click on the **Create delivery stream** button.



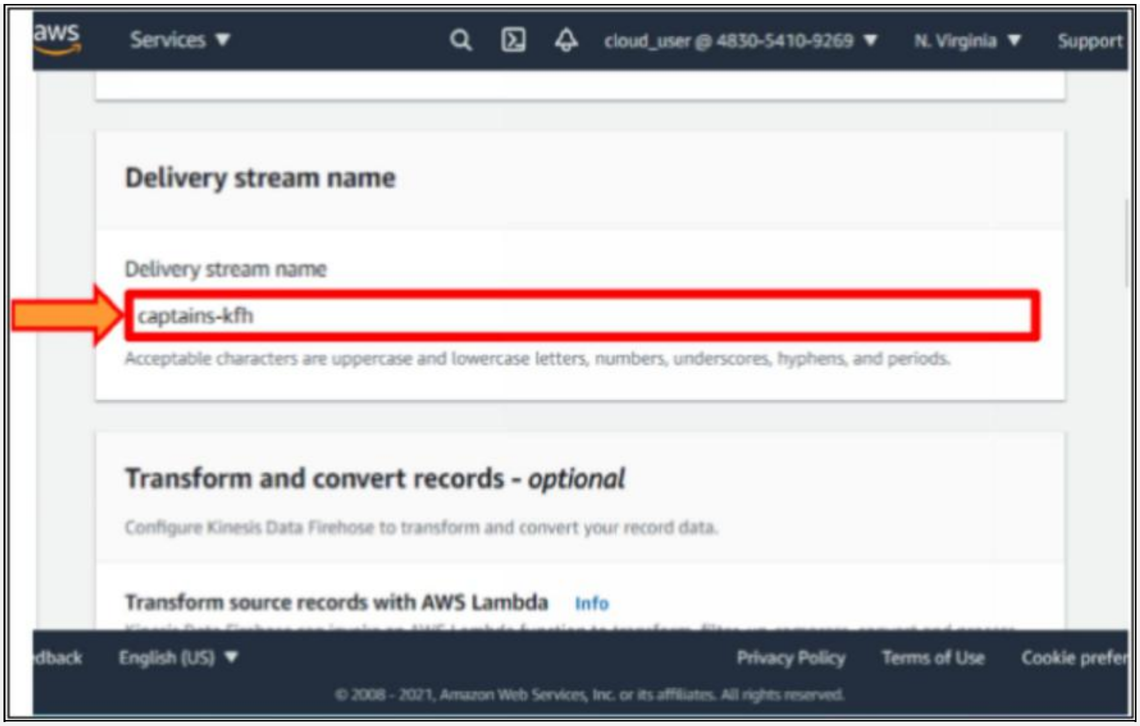
5. Select **Direct PUT** in **Source.**



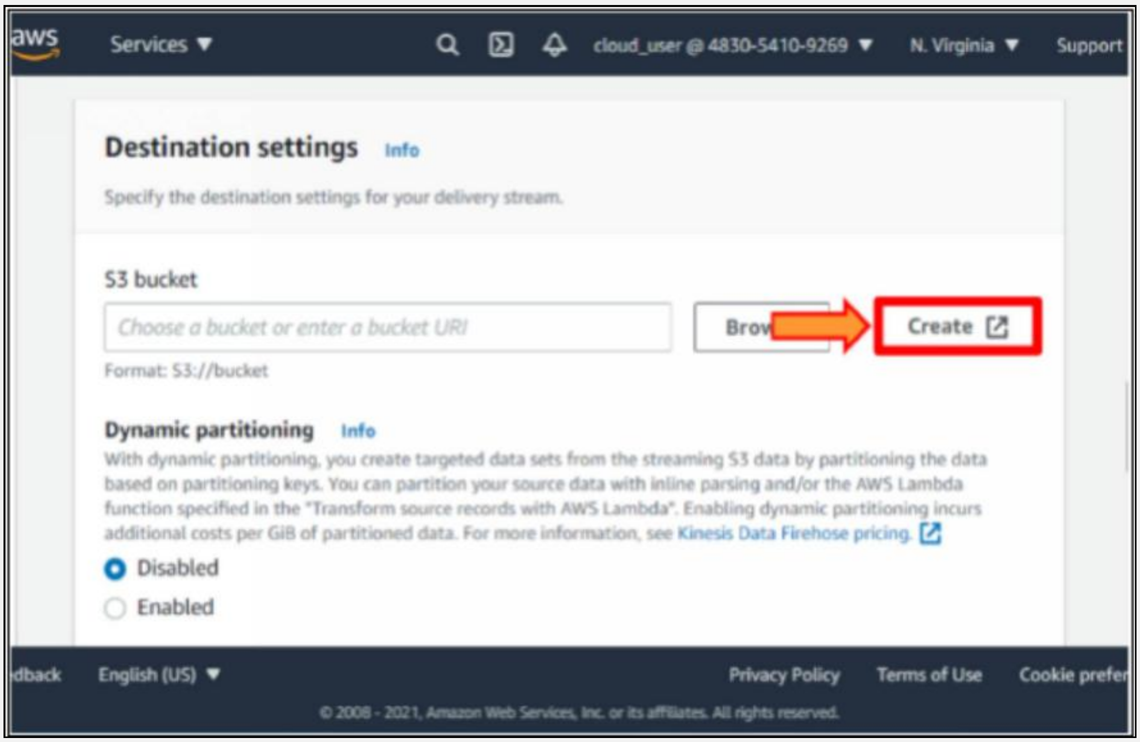
6. Select **Amazon S3** in **Destination.**



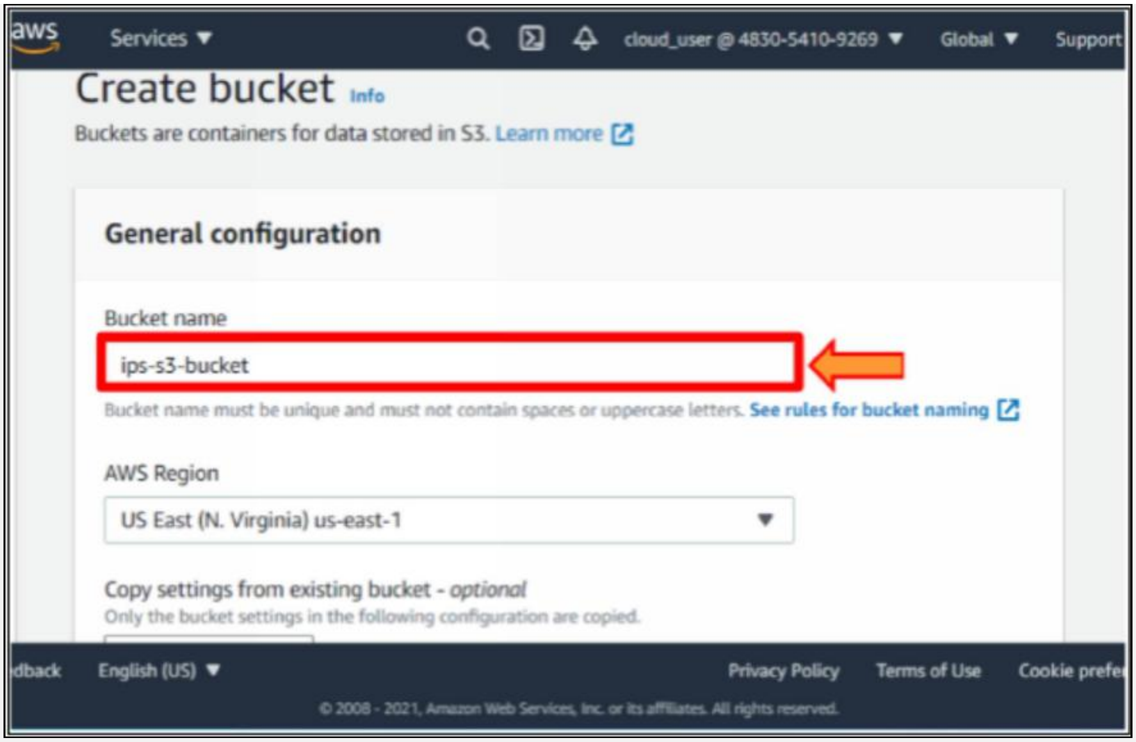
7. Name the delivery stream **captains-kfh.**



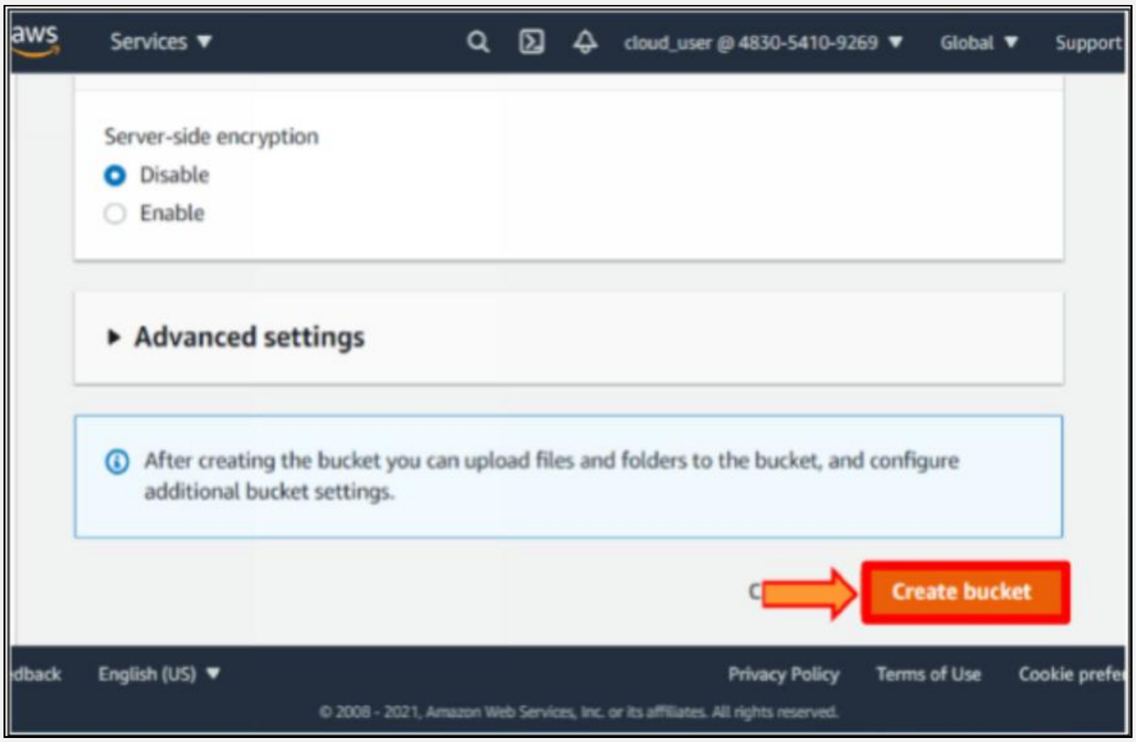
8. Scroll down. Click on the C**reate** button to create an S3 bucket.



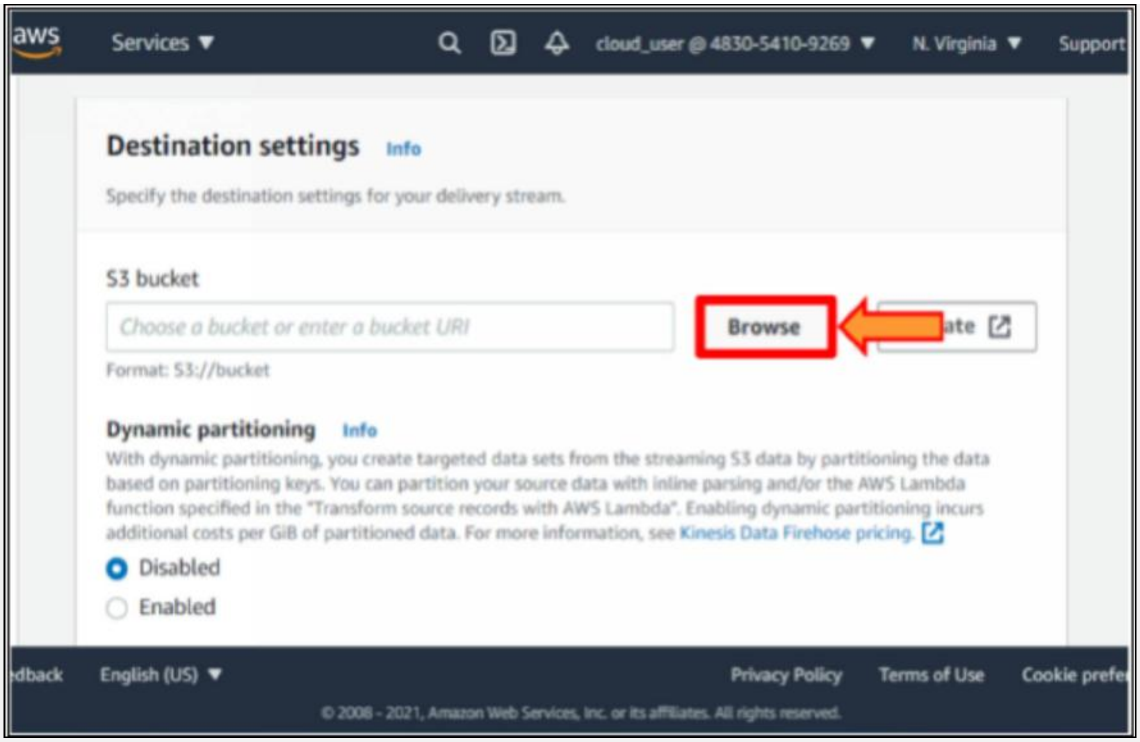
9. Give a bucket name **ips-s3-bucket.**



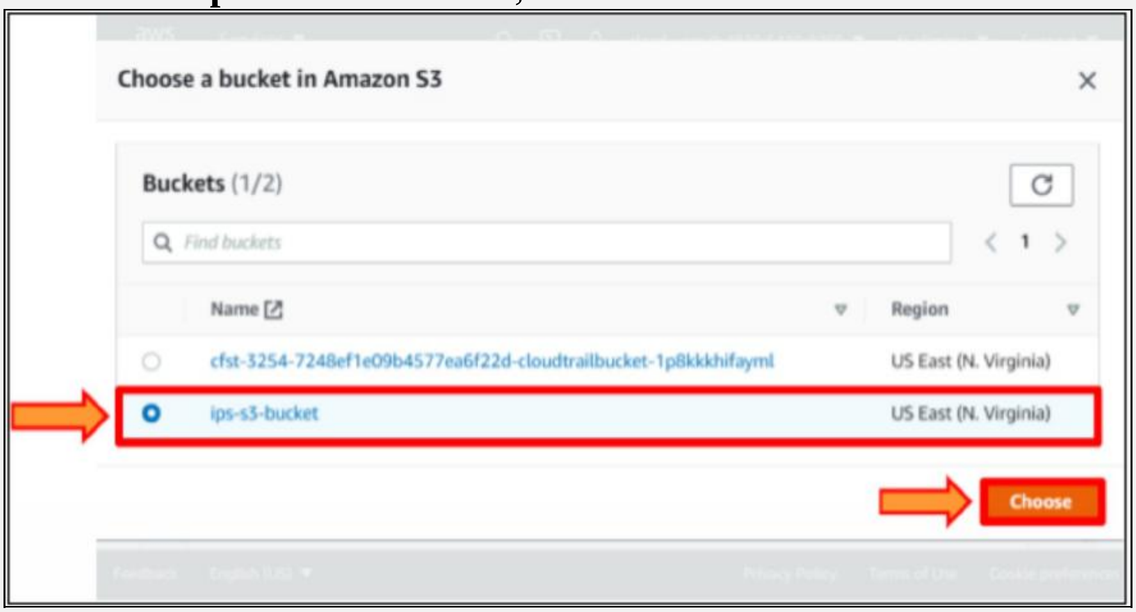
10. Scroll down. Click on the **Create bucket** button.



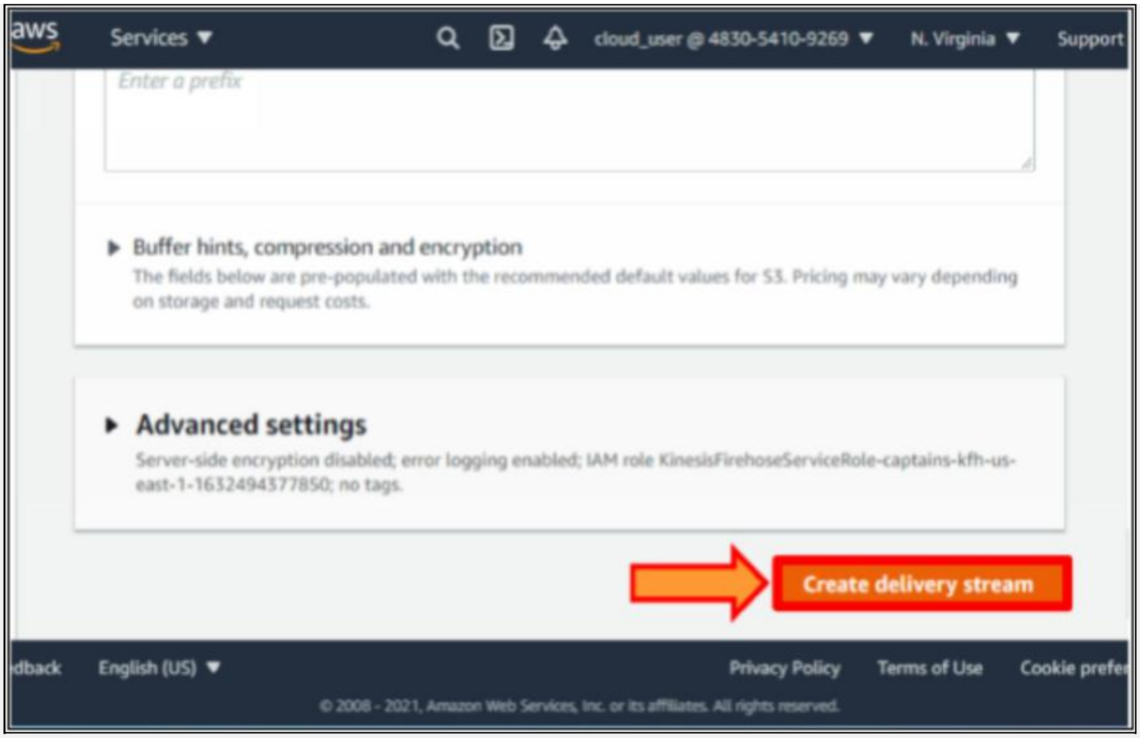
11. Go back to the **Create Delivery Stream** page.
12. Click on the **Browse** button.



13. Select **ips-s3-bucket.** Then, click on the **Choose** button.

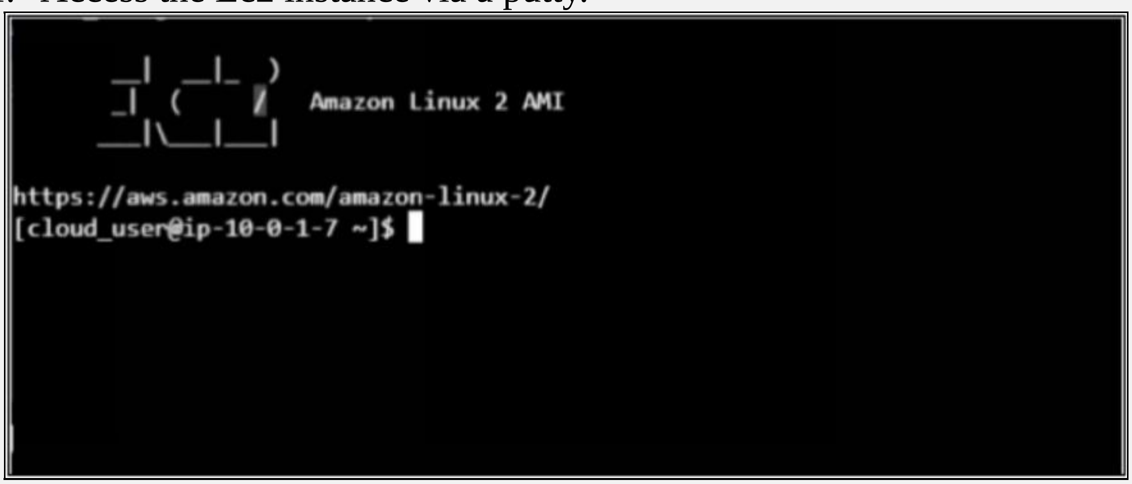


14. Scroll down. Click on the **Create delivery stream** button.

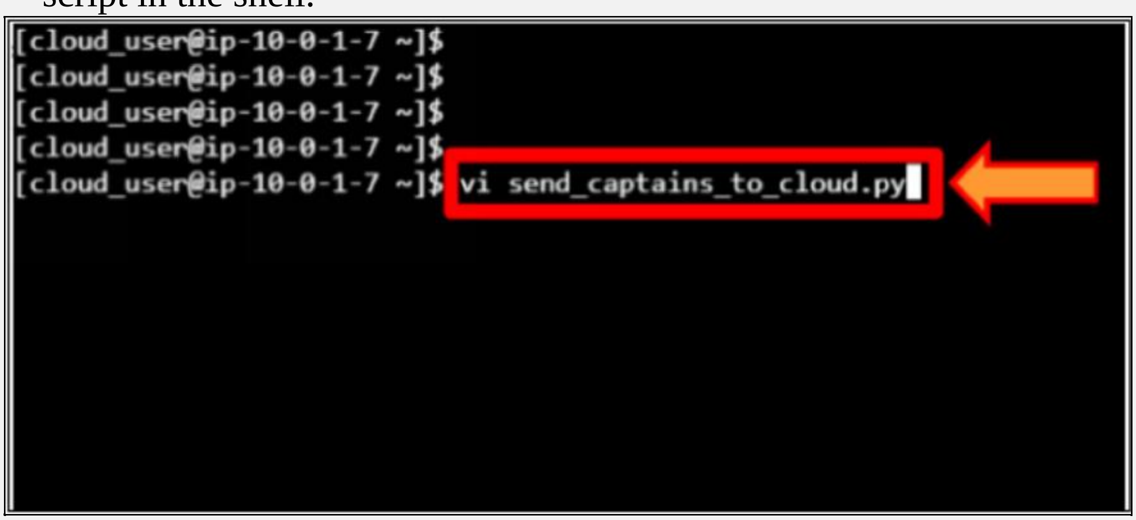


Step 2: Generate Data to the Stream

1. Access the Ec2 instance via a putty.



2. Type the **vi send_captains_to_clous.py** command to view the Python script in the shell.



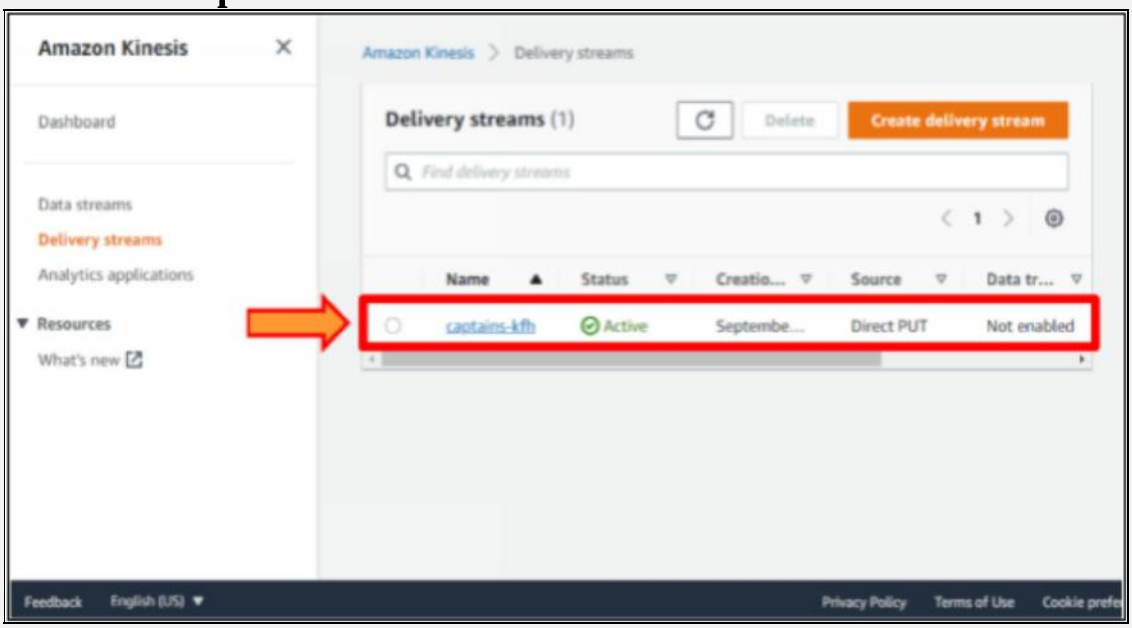
3. Type **:q,** then press **Enter** to quit from the vi editor.

```
#!/usr/bin/env python
import boto3
from faker import Faker
import random
import time
import json

DeliveryStreamName = 'captains-kfh'
client = boto3.client('firehose')
fake = Faker()

captains = [
    "Jean-Luc Picard",
    "James T. Kirk",
    "Han Solo",
    "Kathryn Janeway",
    "Malcolm Reynolds",
    "William Adama",
    "Turanga Leela",
    "Jacob Keyes",
    "Wilhuff Tarkin",
    "Christopher Pike",
    "David Bowman",
    "The Doctor",
    "John Robinson",
    "Khan Noonien Singh"
];

record = {}
:q
```

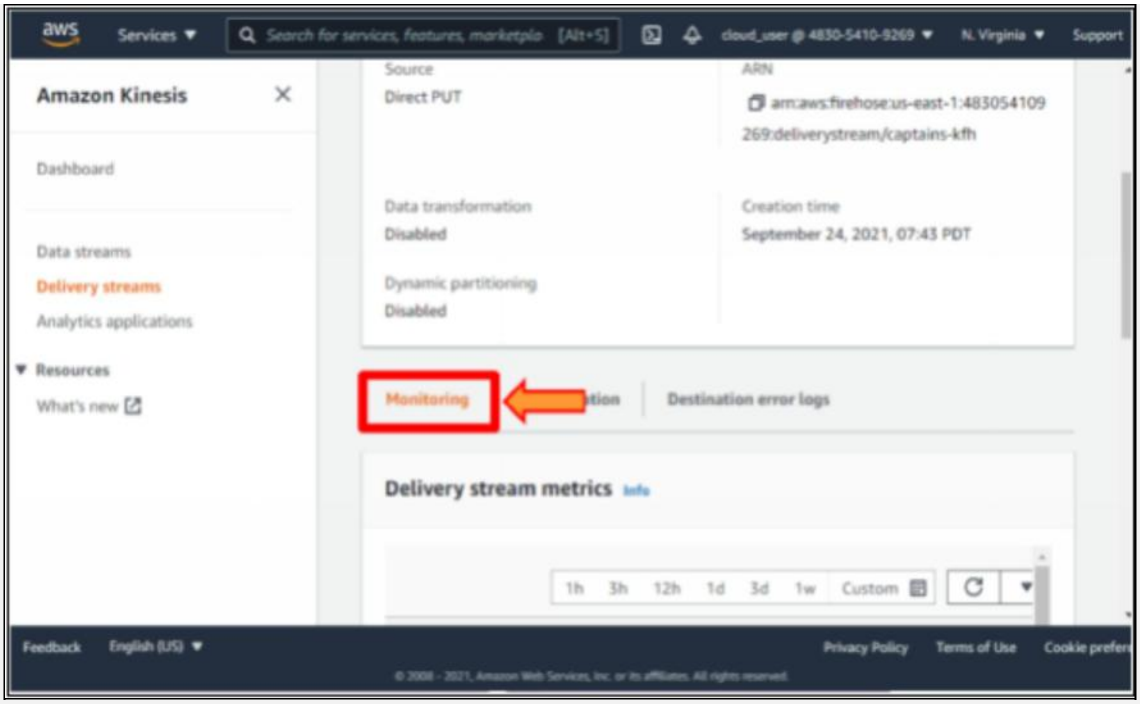
4. Type the **python3 send_captains_to_cloud.py** command to run the python script.

```
[cloud_user@ip-10-0-1-7 ~]$
[cloud_user@ip-10-0-1-7 ~]$
[cloud_user@ip-10-0-1-7 ~]$
[cloud_user@ip-10-0-1-7 ~]$
[cloud_user@ip-10-0-1-7 ~]$
[cloud_user@ip-10-0-1-7 ~]$ python3 send_captains_to_cloud.py
```

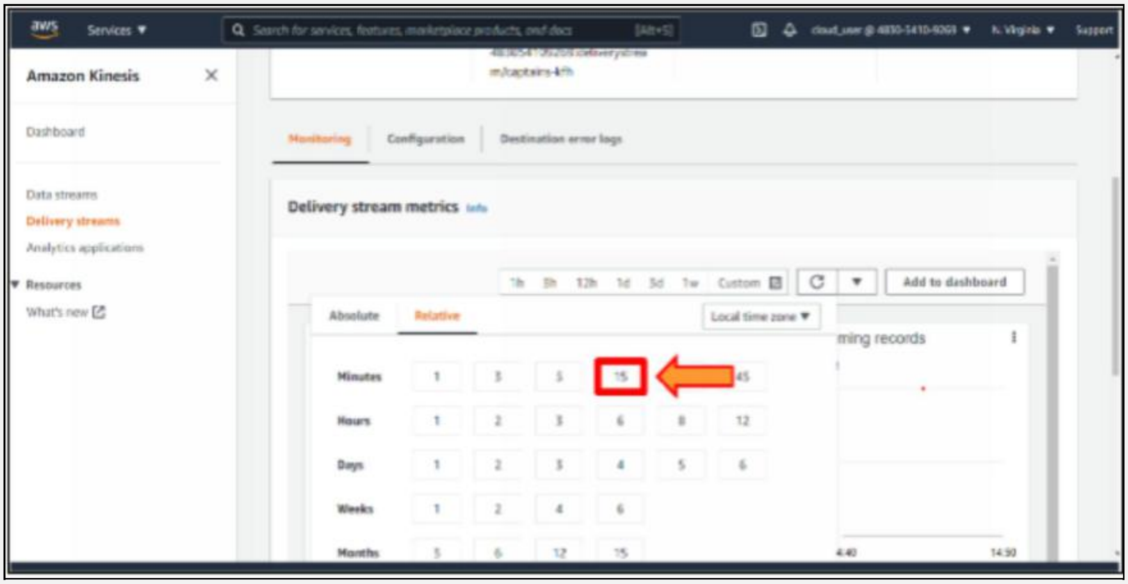
5. Go back to the **Kinesis Data Firehose** dashboard.
6. Click on **captain-kfh.**



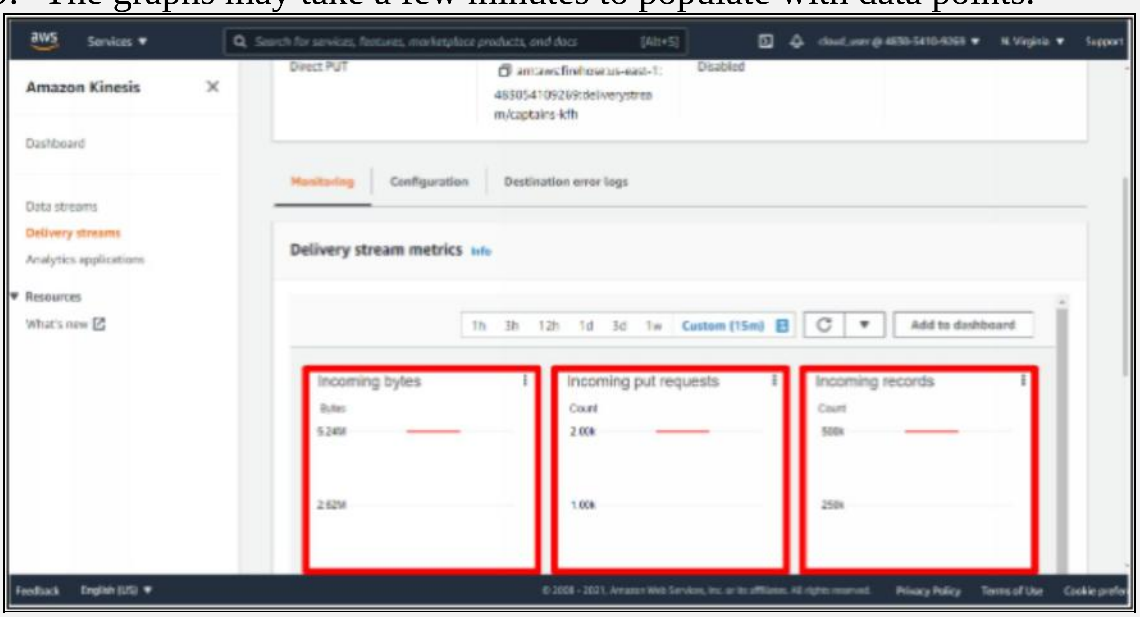
7. Click on the **Monitoring** tab.



8. Click on **15 min.**

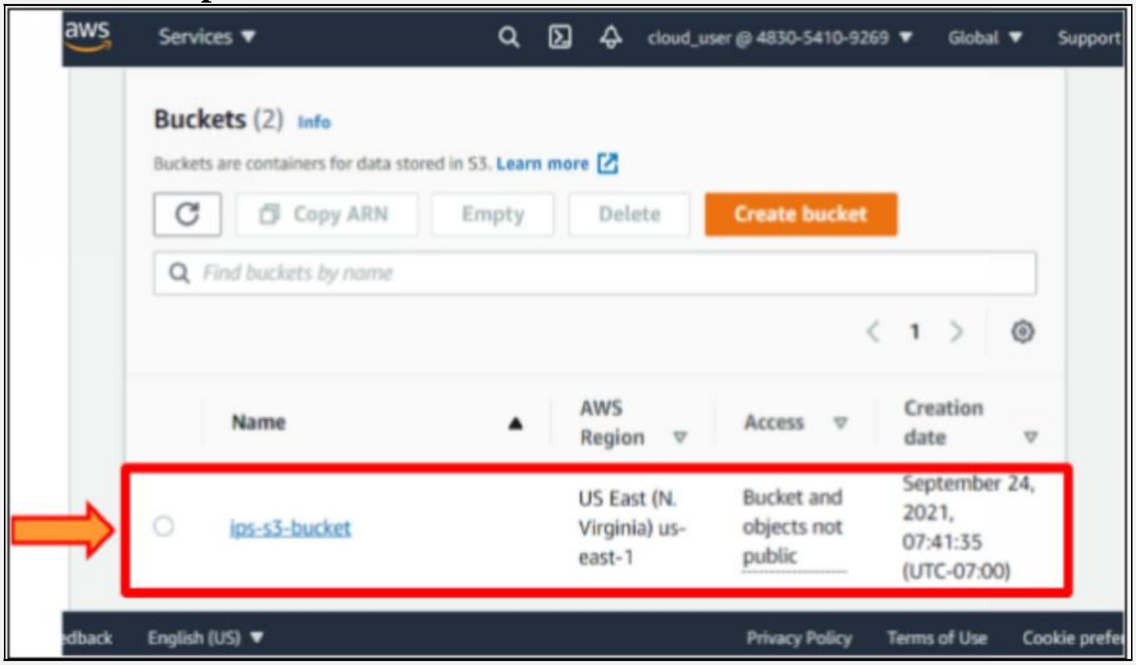


9. The graphs may take a few minutes to populate with data points.

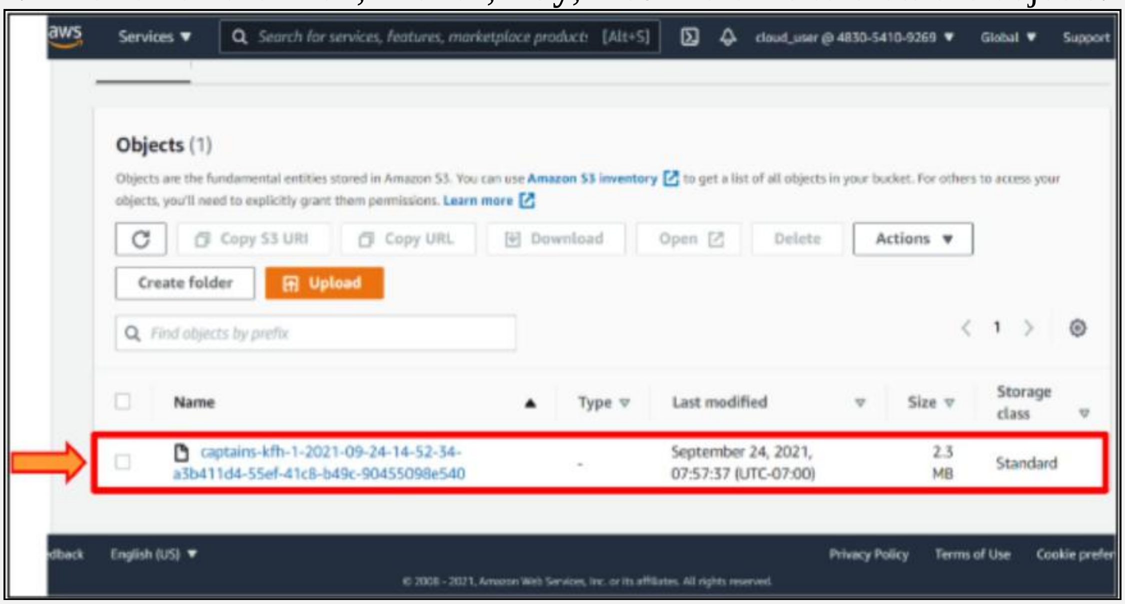


10. Go to the **Amazon S3** dashboard.

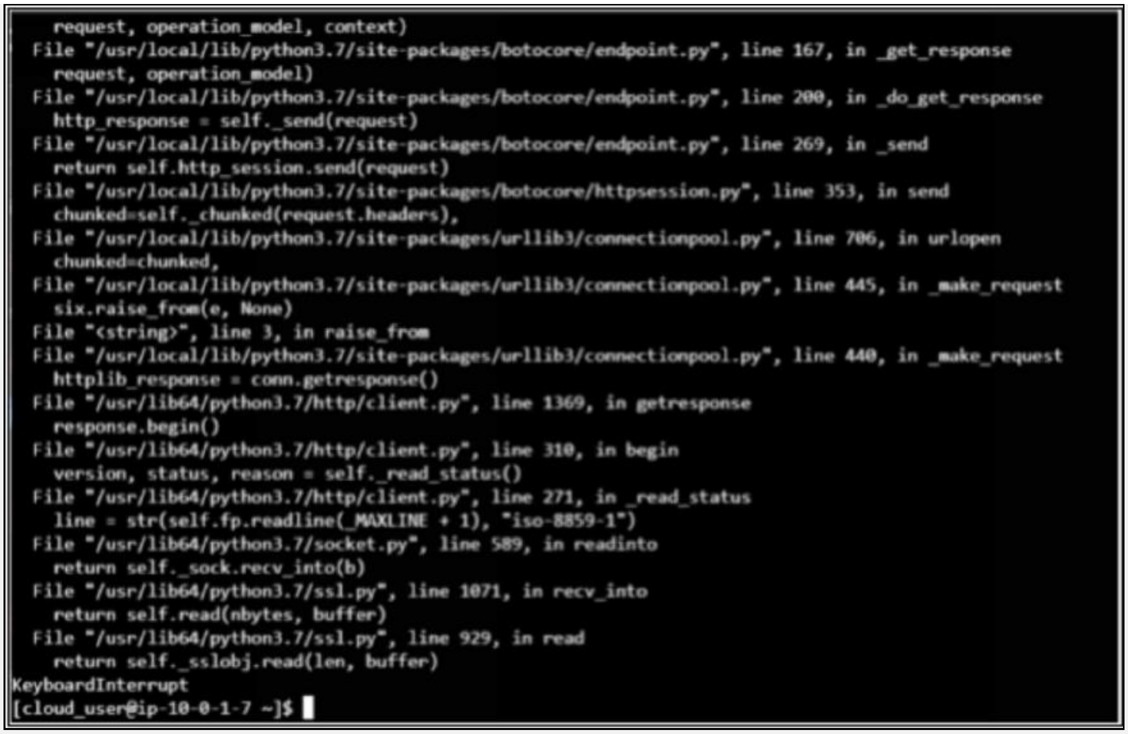
11. Select **ips-s3-bucket.**



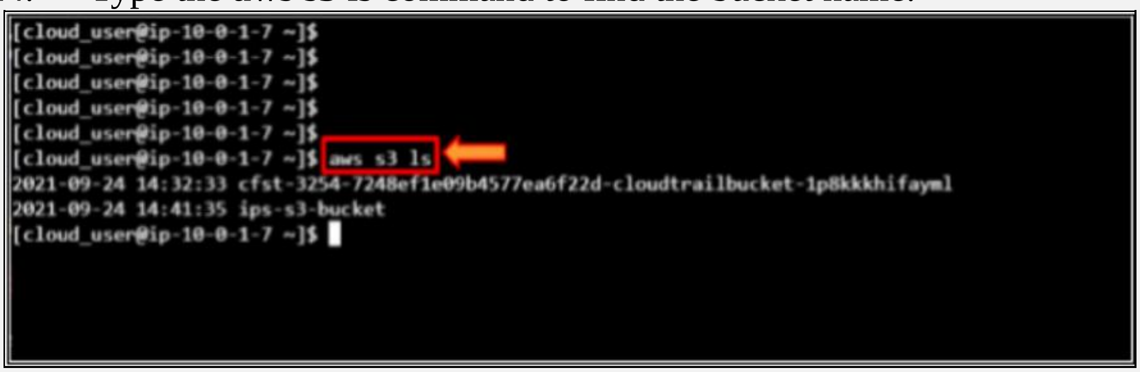
12. Click on the **Year**, **Month**, **Day**, **Hour folders** to view the objects.



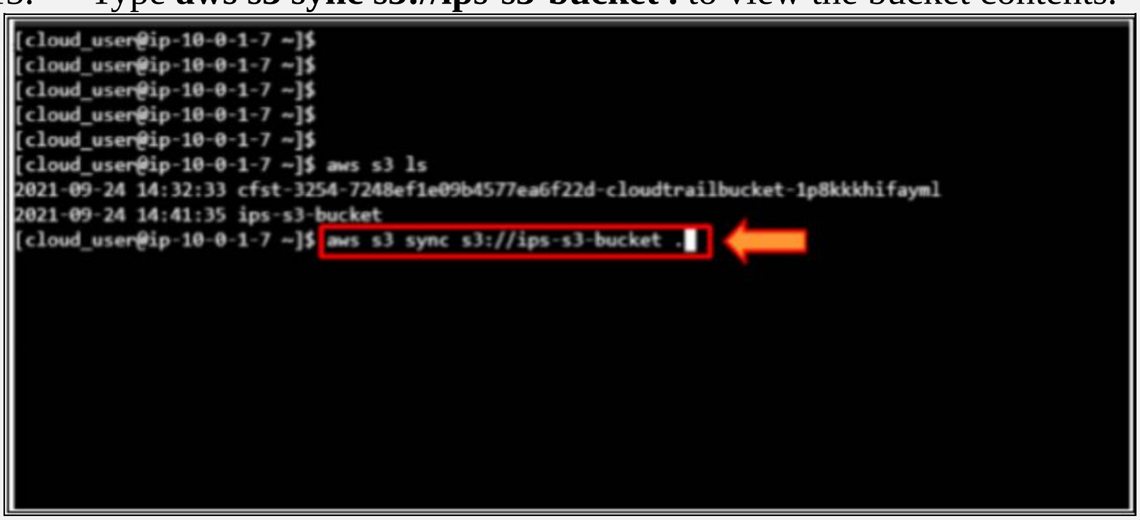
13. Go back to the terminal. Press **Ctrl+C** to stop sending data to the stream.



14. Type the **aws s3 ls** command to find the bucket name.

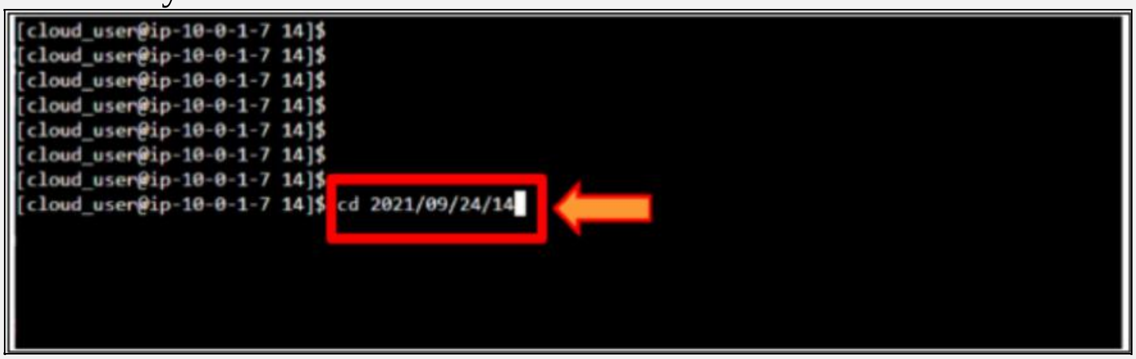


15. Type **aws s3 sync s3://ips-s3-bucket .** to view the bucket contents.

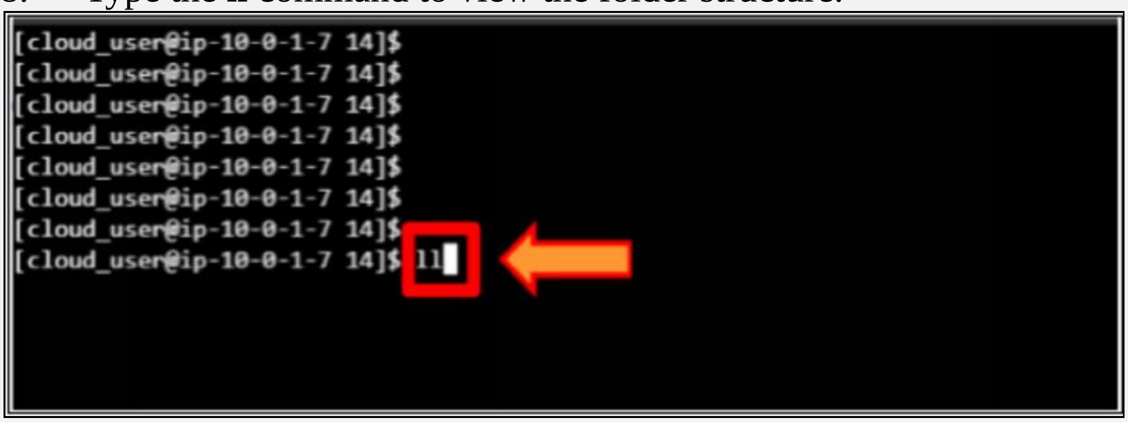


16. Type the **ll** command to view the folder structure.

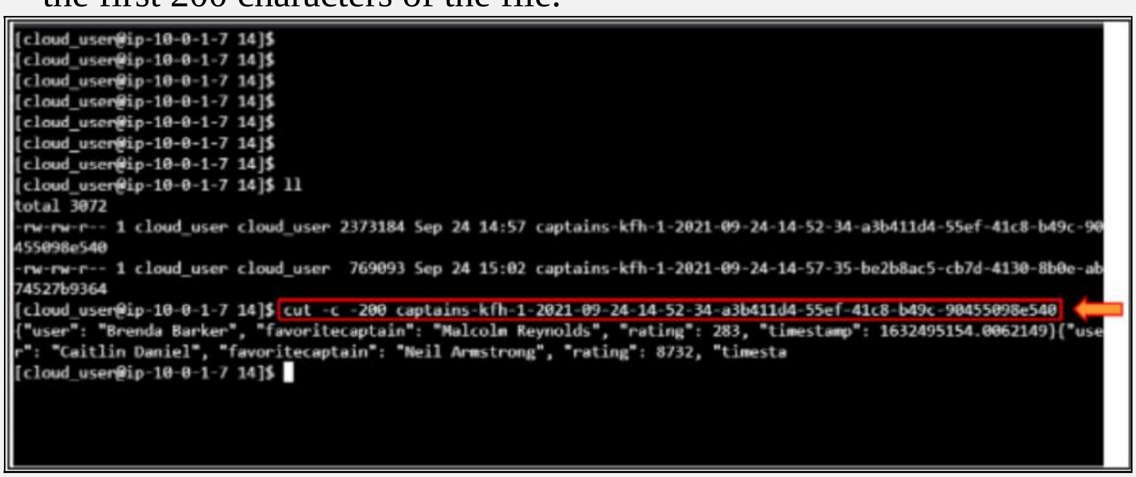
17. Type **cd <YEAR>/<MONTH>/<DAY>/<HOUR>/** to change the directory.



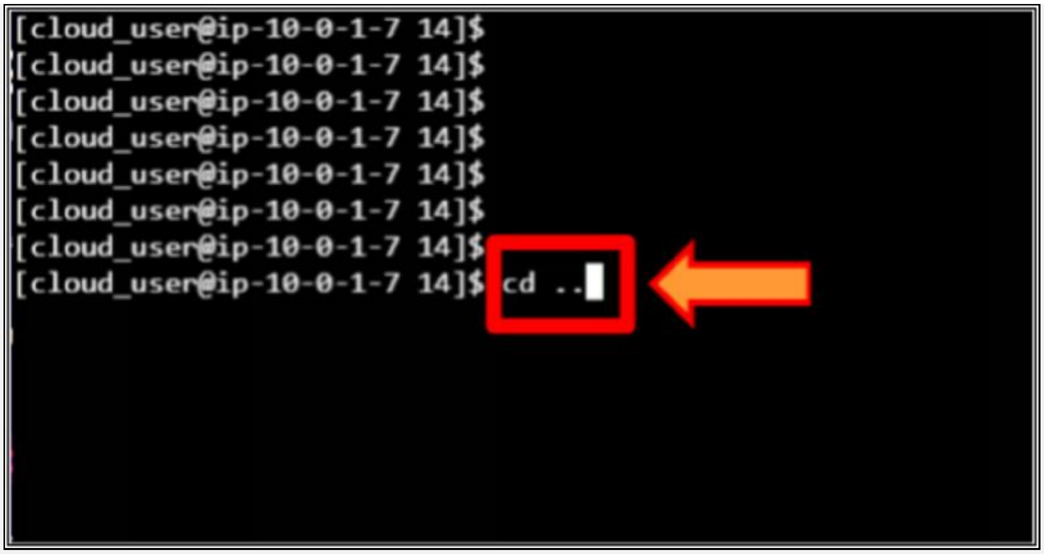
18. Type the **ll** command to view the folder structure.



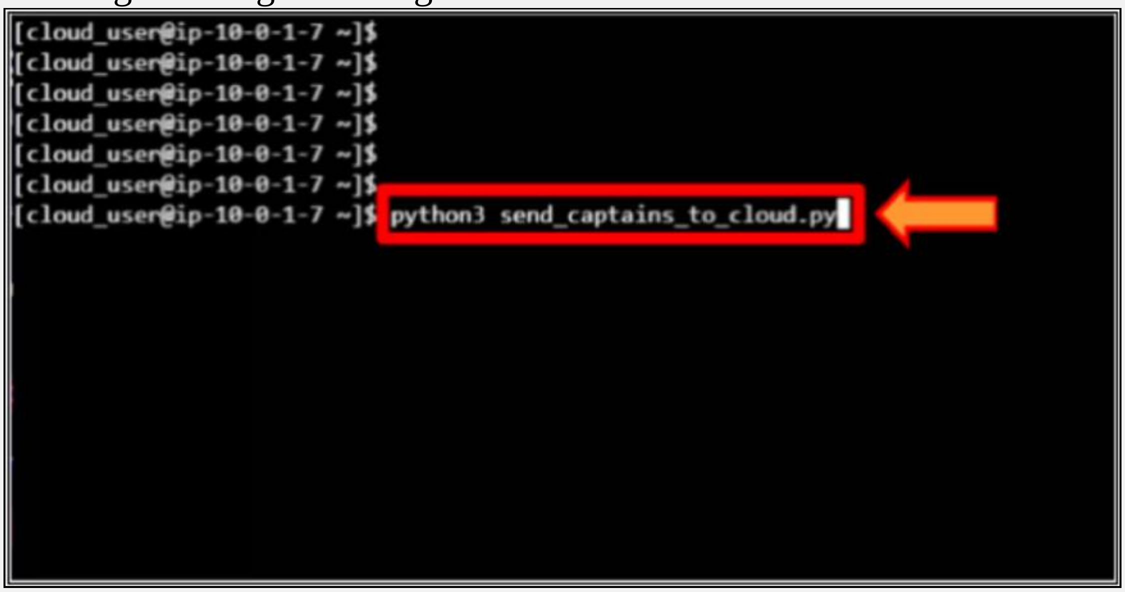
19. Copy one of the file names.
20. Type the **cut –c -200 <INSERT_FILE_NAME>** command to view the first 200 characters of the file.



21. Type the **cd** .. command to return to the home directory.

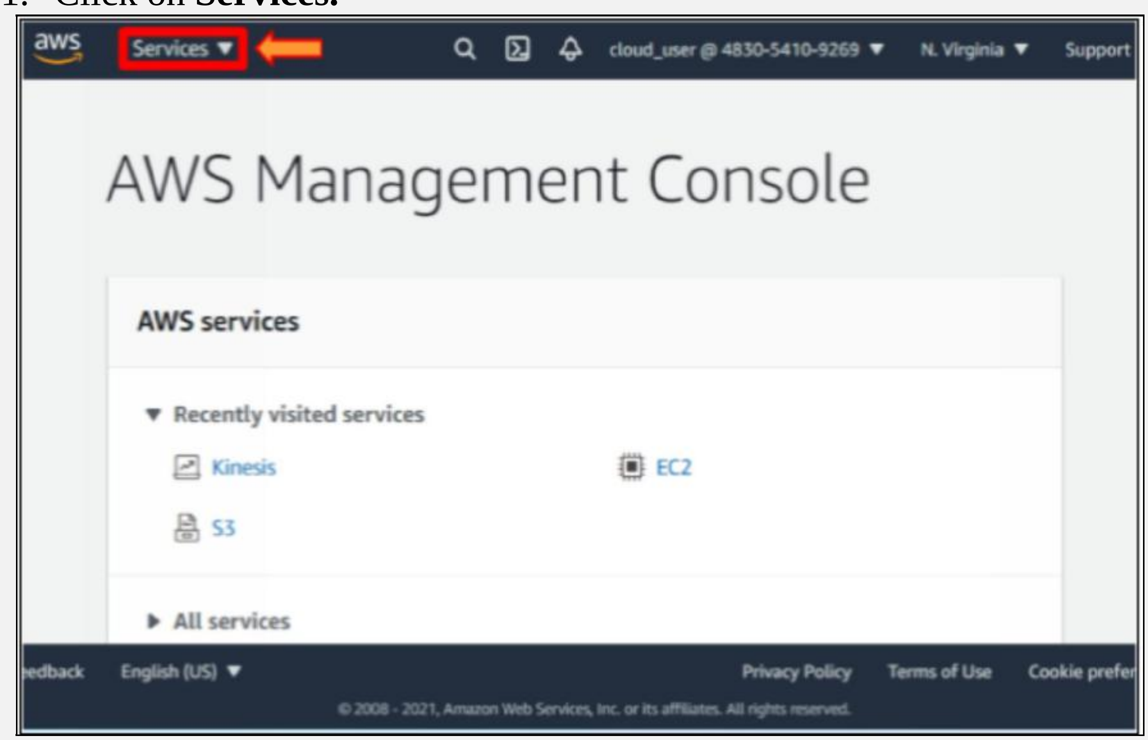


22. Type the **python3 send_captains_to_clous.py** command to start the data-generating stream again.

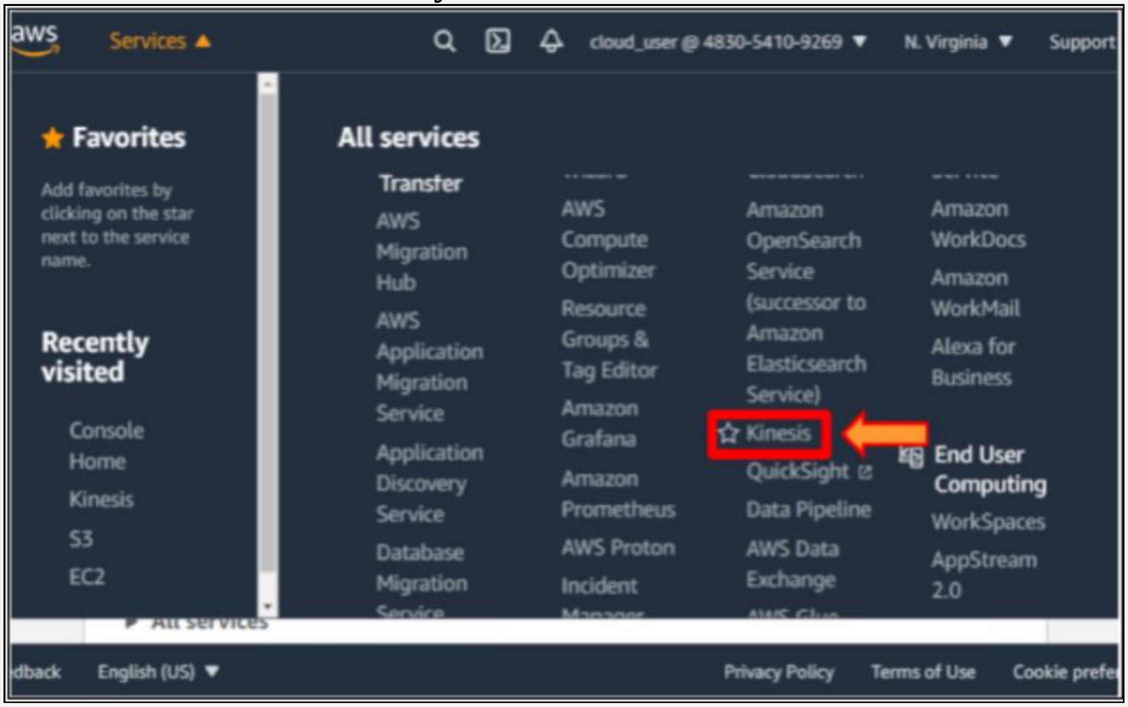


Step 3: Create Kinesis Data Analytics

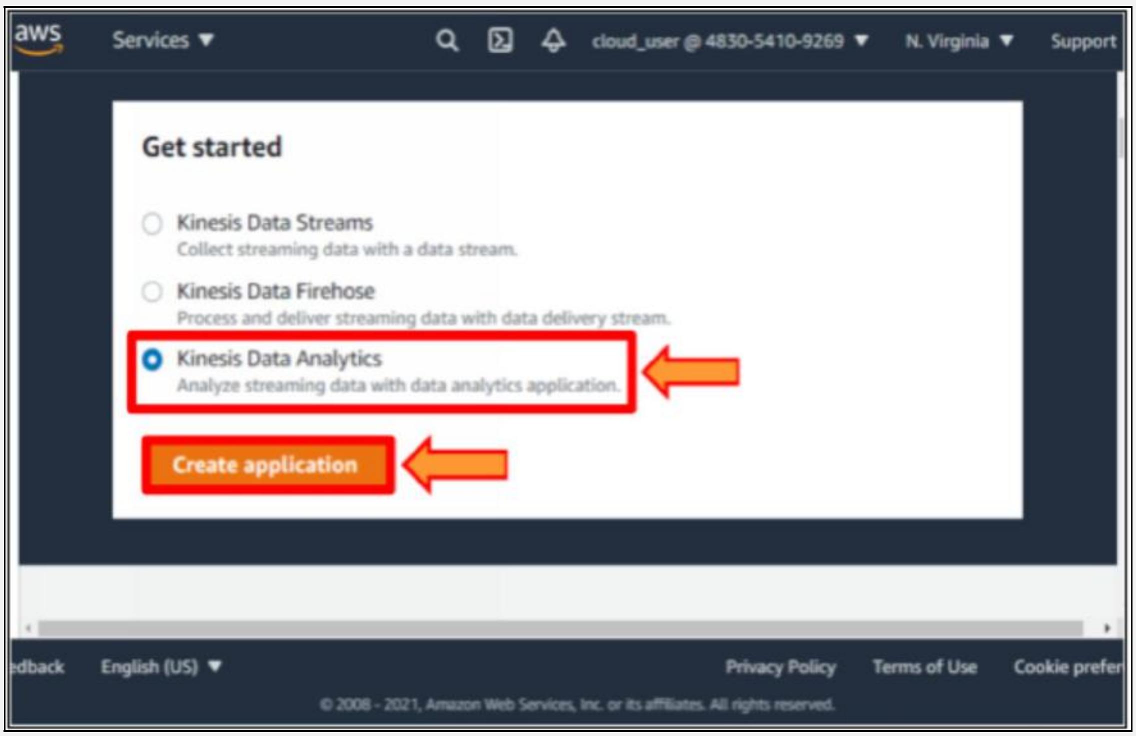
1. Click on **Services.**



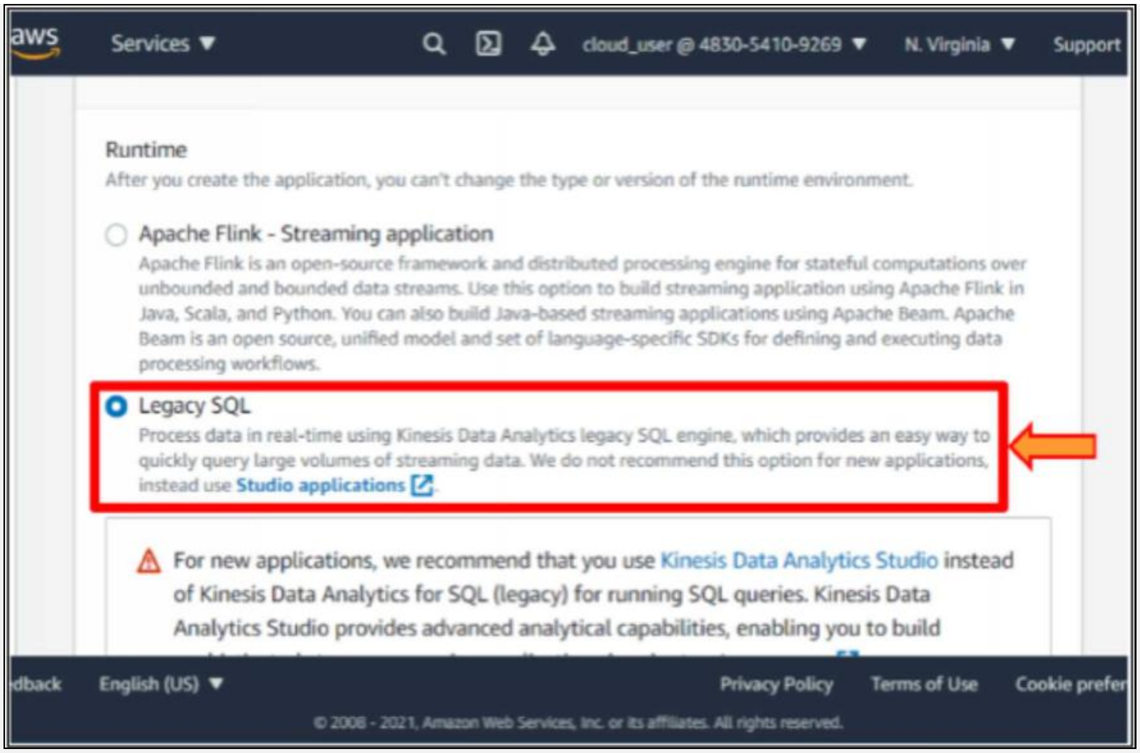
2. Select **Kinesis** from **Analytics.**



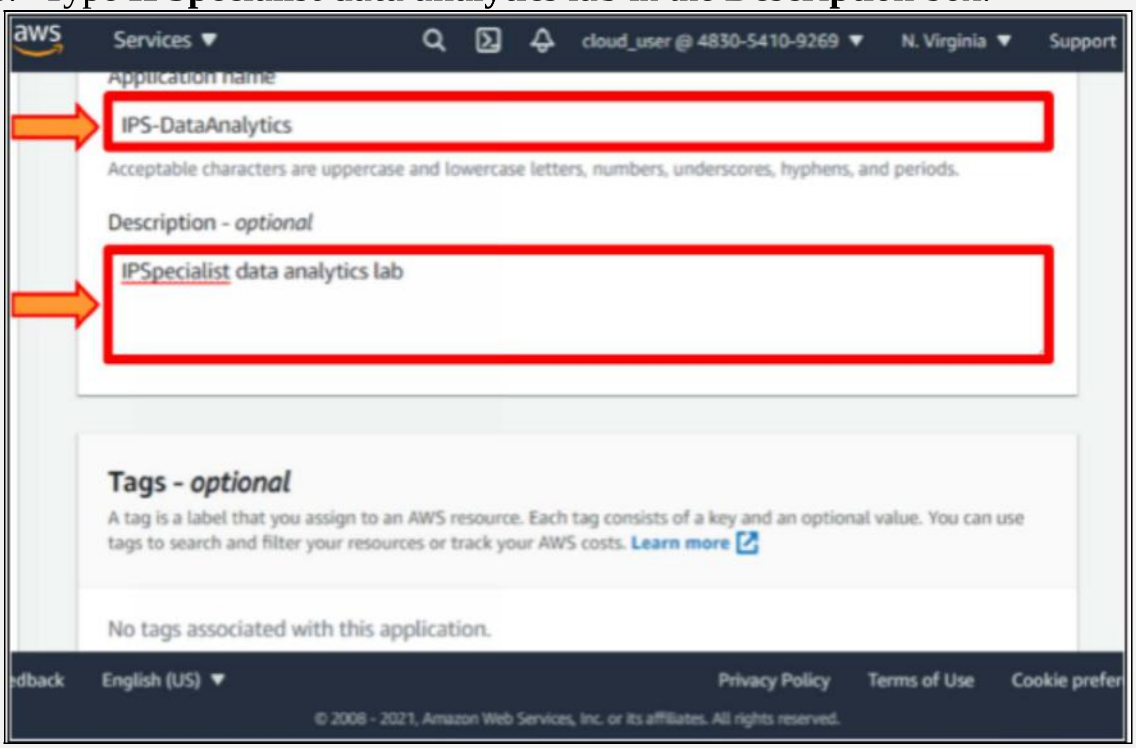
3. Select **Kinesis Data Analytics.** Then, click on the **Create application** button.



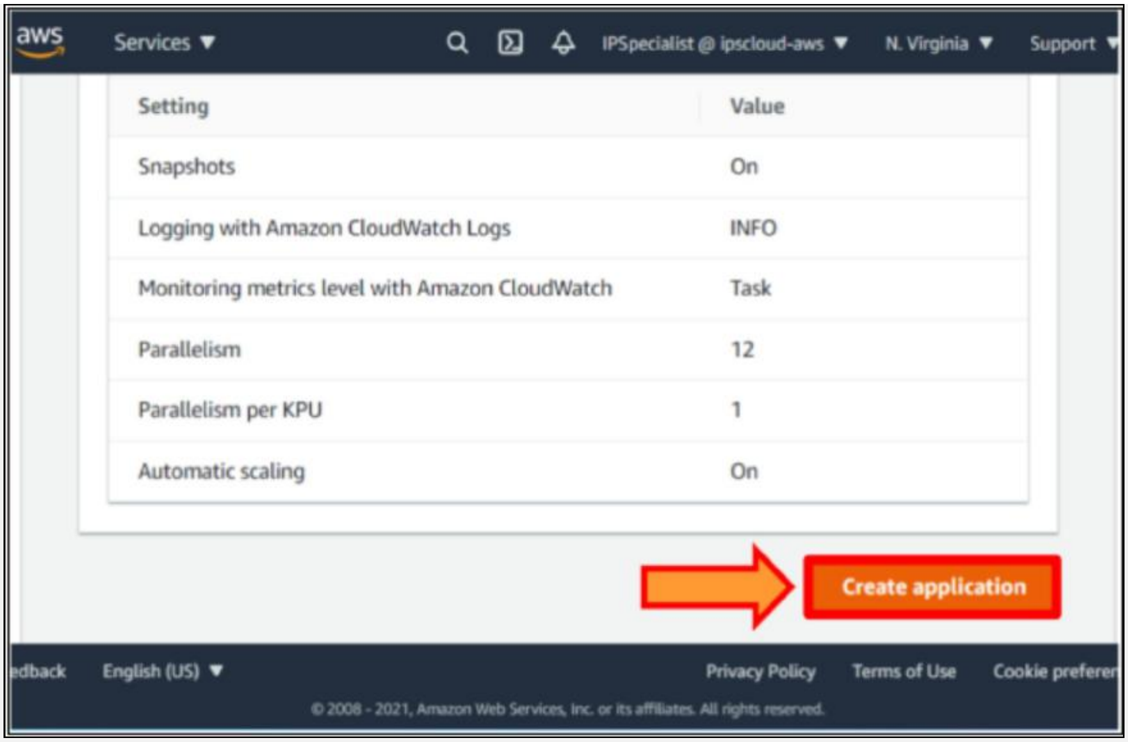
4. Select **Legacy SQL.**



5. Give an application name **IPS-DataAnalytics**.
6. Type **IPSpecialist data analytics lab** in the **Description** box.

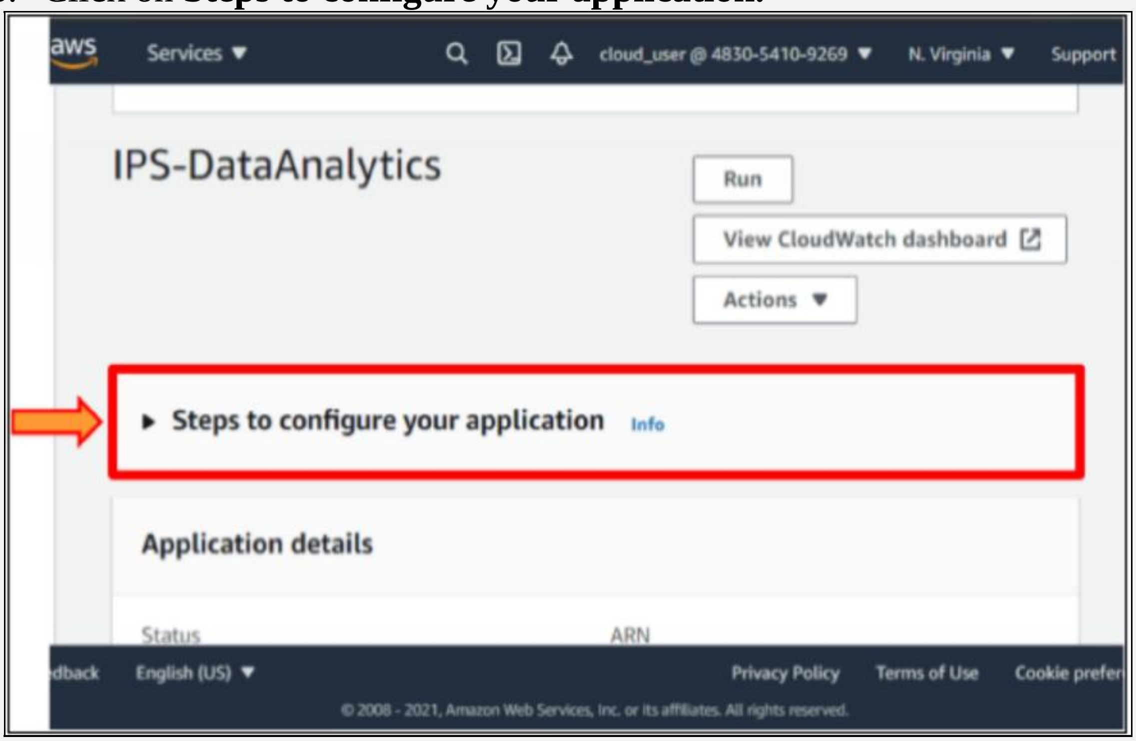


7. Click on the **Create application** button.

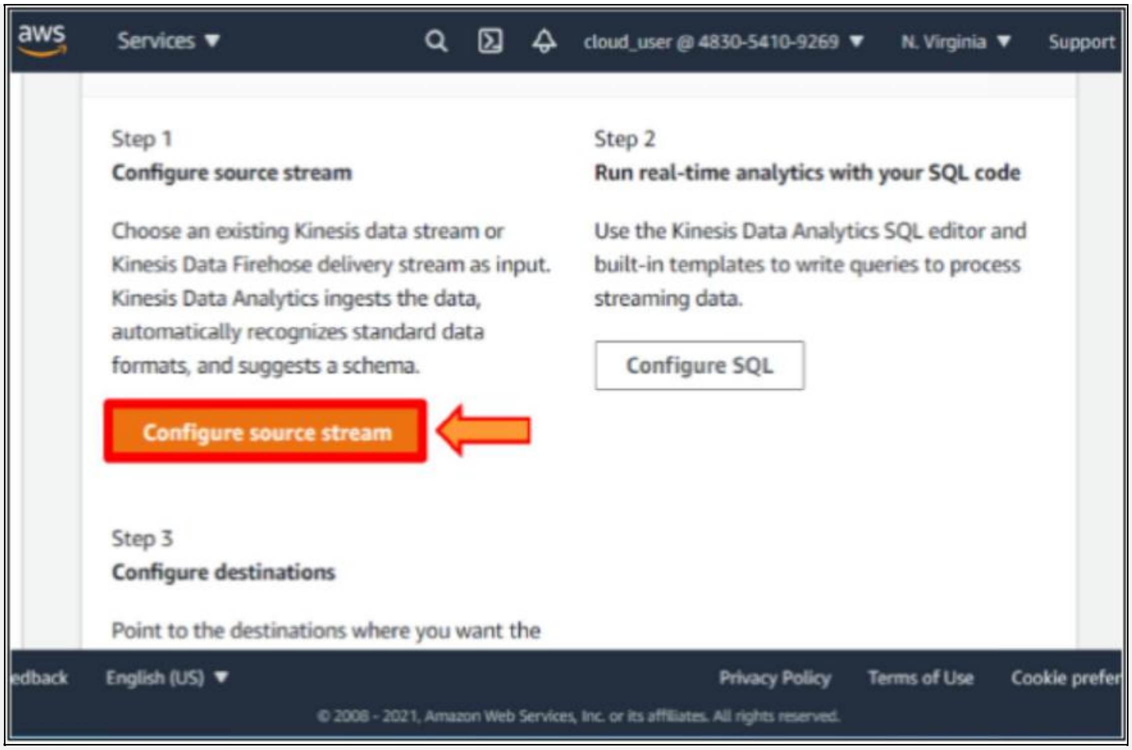


Hence, the Kinesis Data Analytics application has been created.

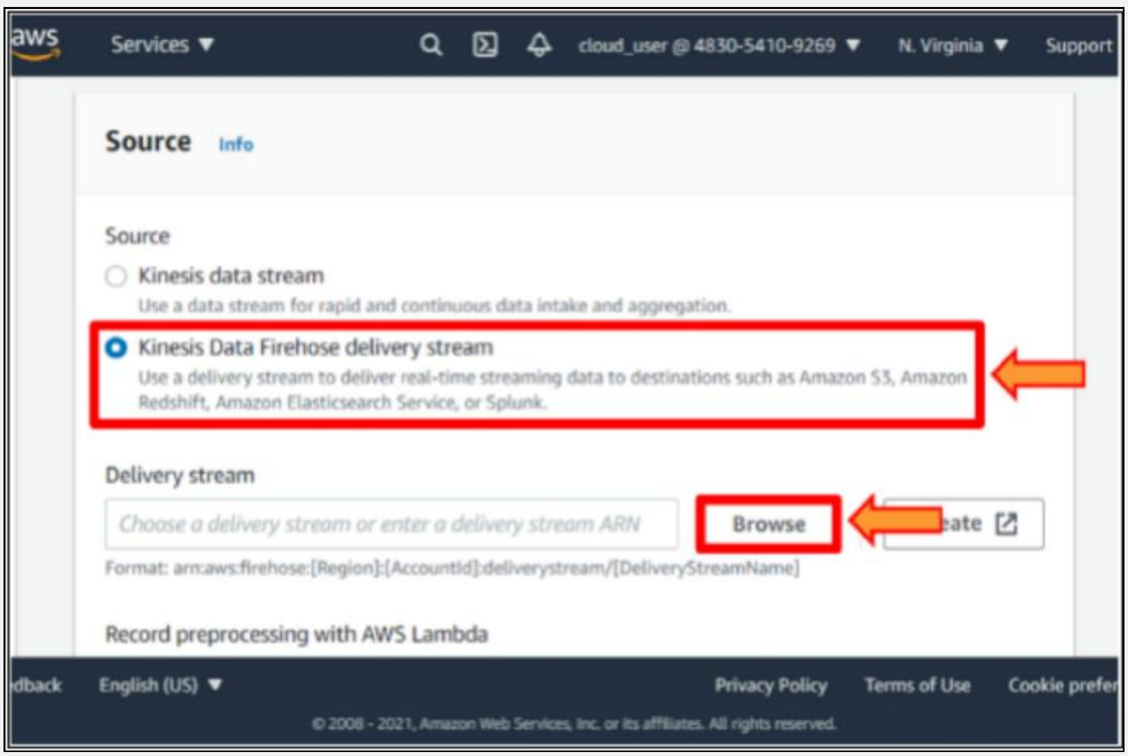
8. Click on **Steps to configure your application.**



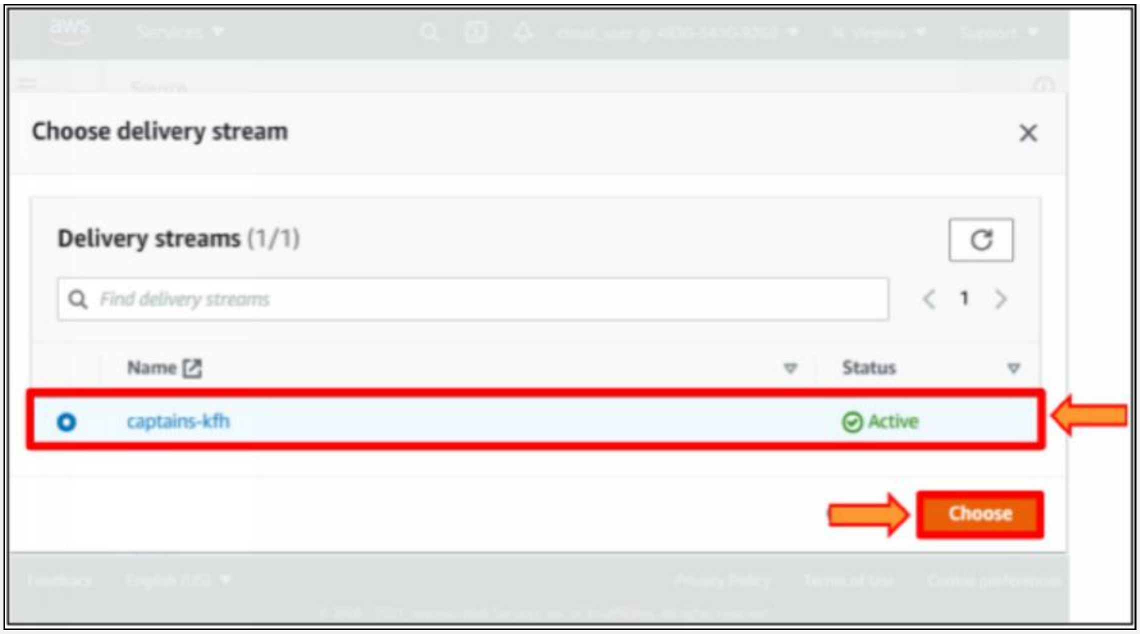
9. Click on the **Configure source stream** button.



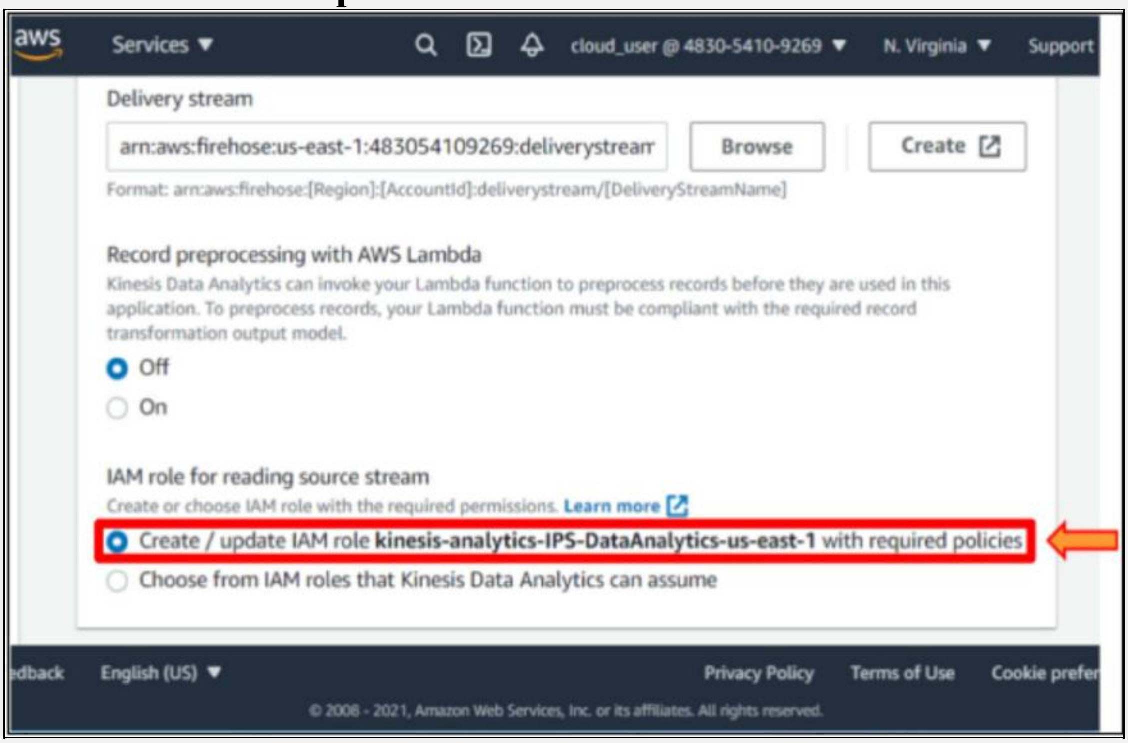
10. Select **Kinesis Data Firehose delivery stream.**
11. Click on the **Browse** button.



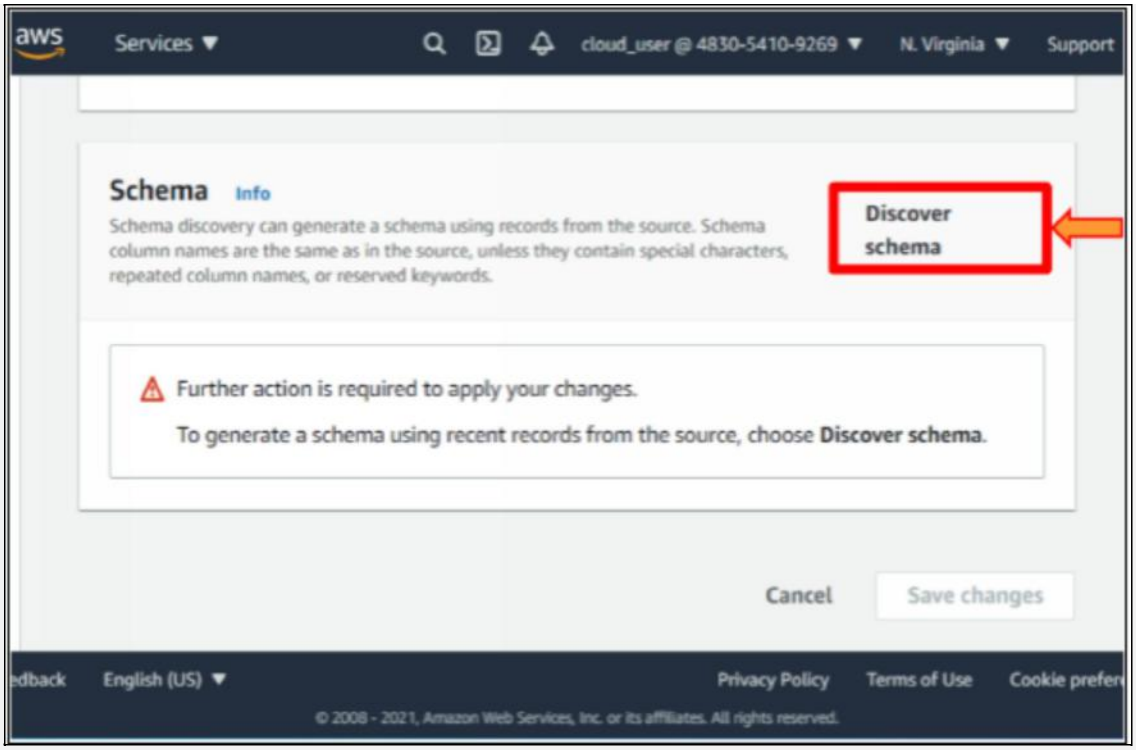
12. Select **captains-kfh**. Then, click on the **Choose** button.



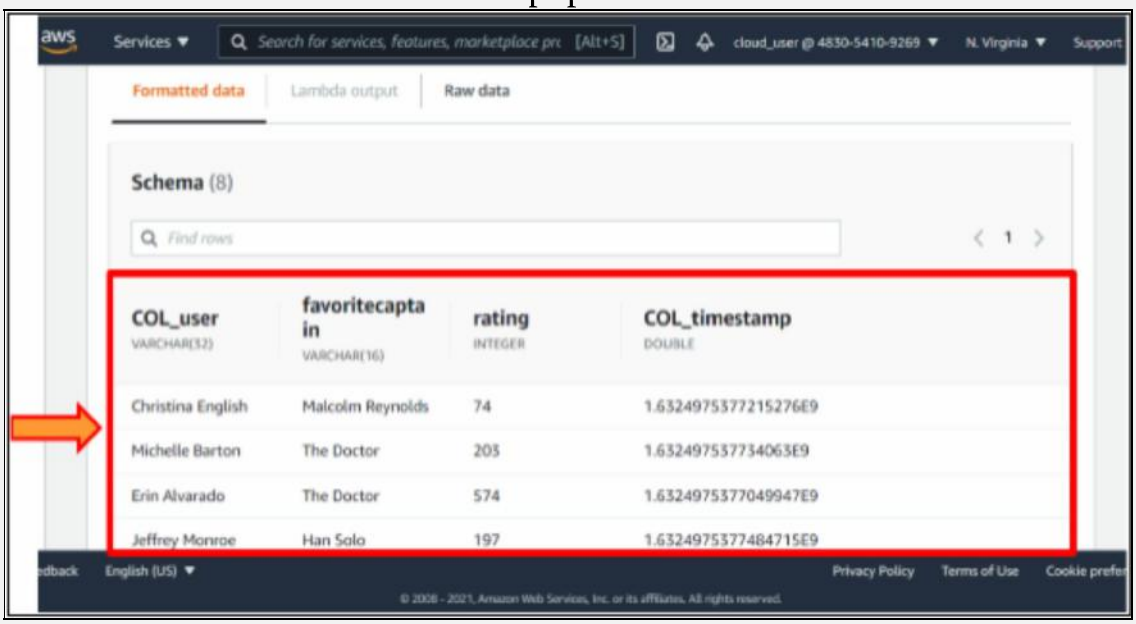
13. Select **Create/Update IAM Role.**



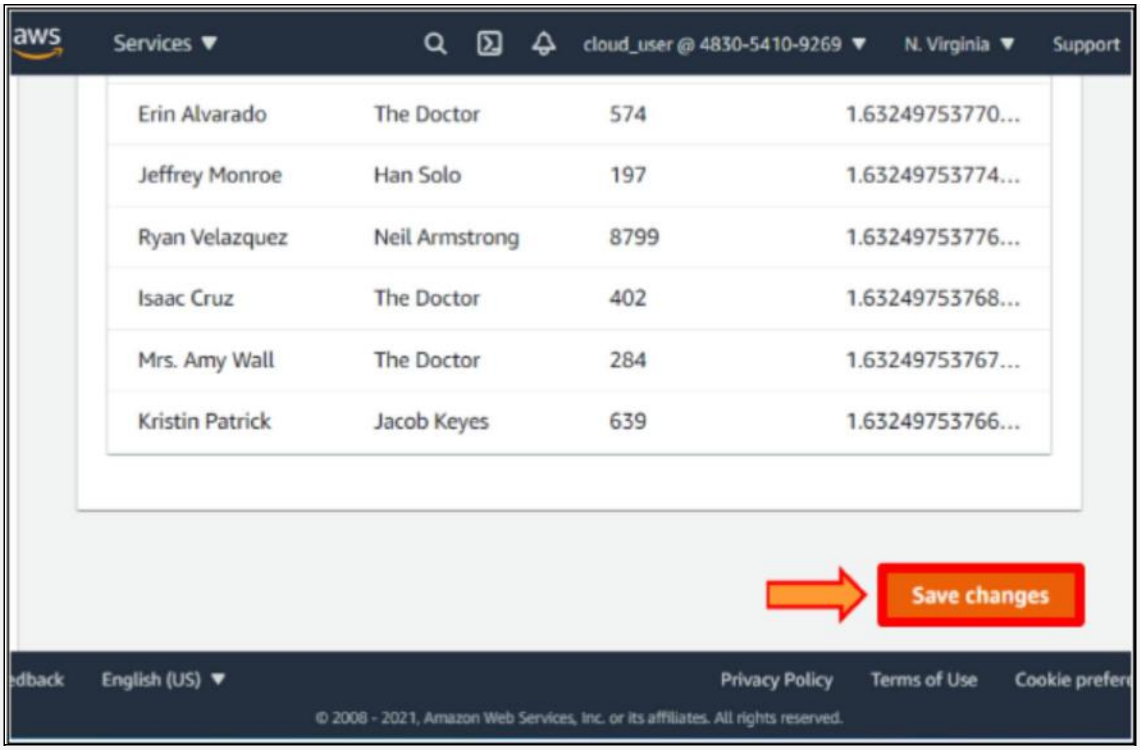
14. Scroll down. Click on the **Discover Schema** button,



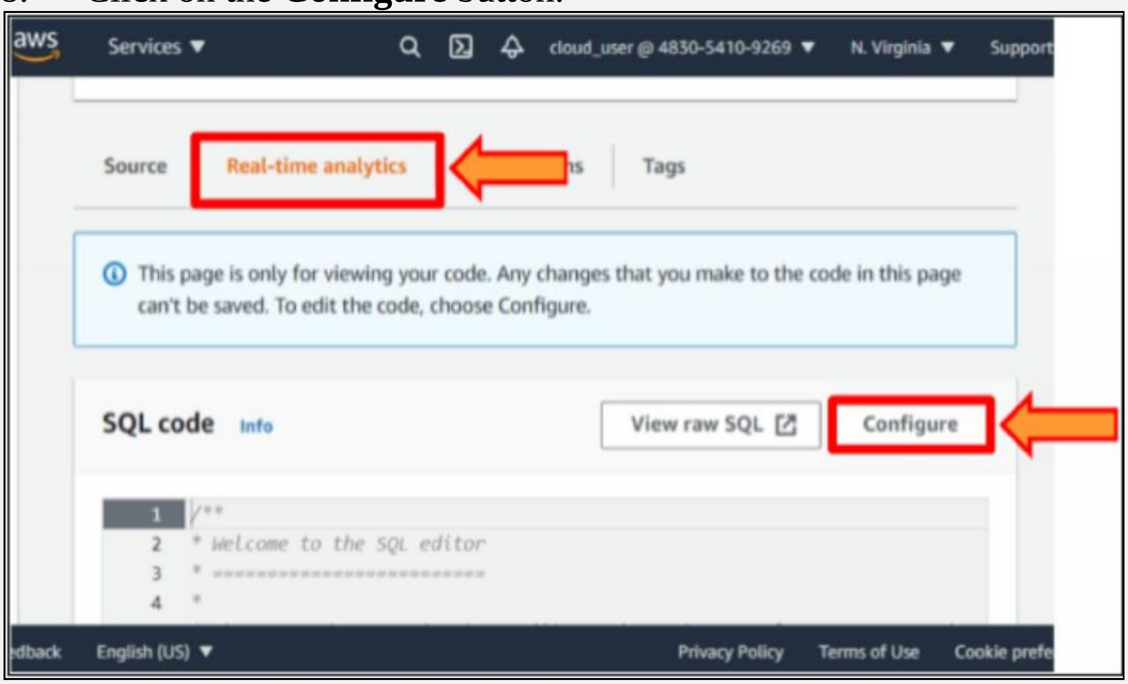
15. It will take a few minutes to populate the table.



16. Scroll down. Click on the **Save changes** button.



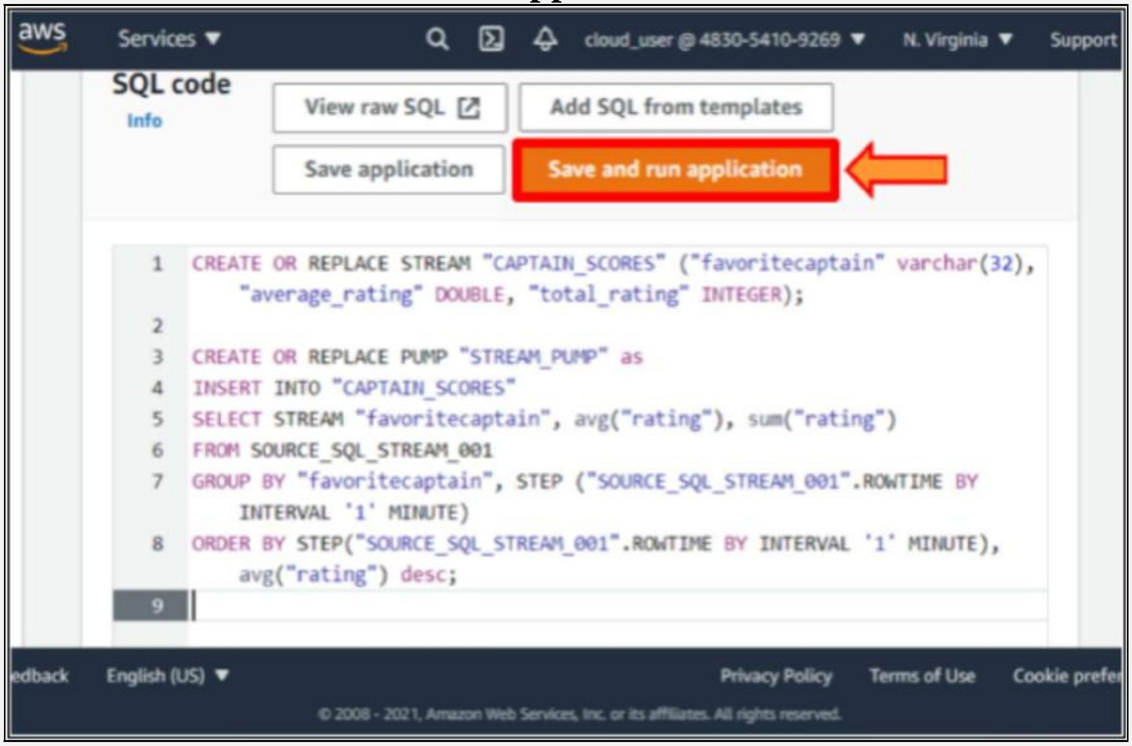
17. Scroll down. Click on the **Real-time analytics** tab.
18. Click on the **Configure** button.



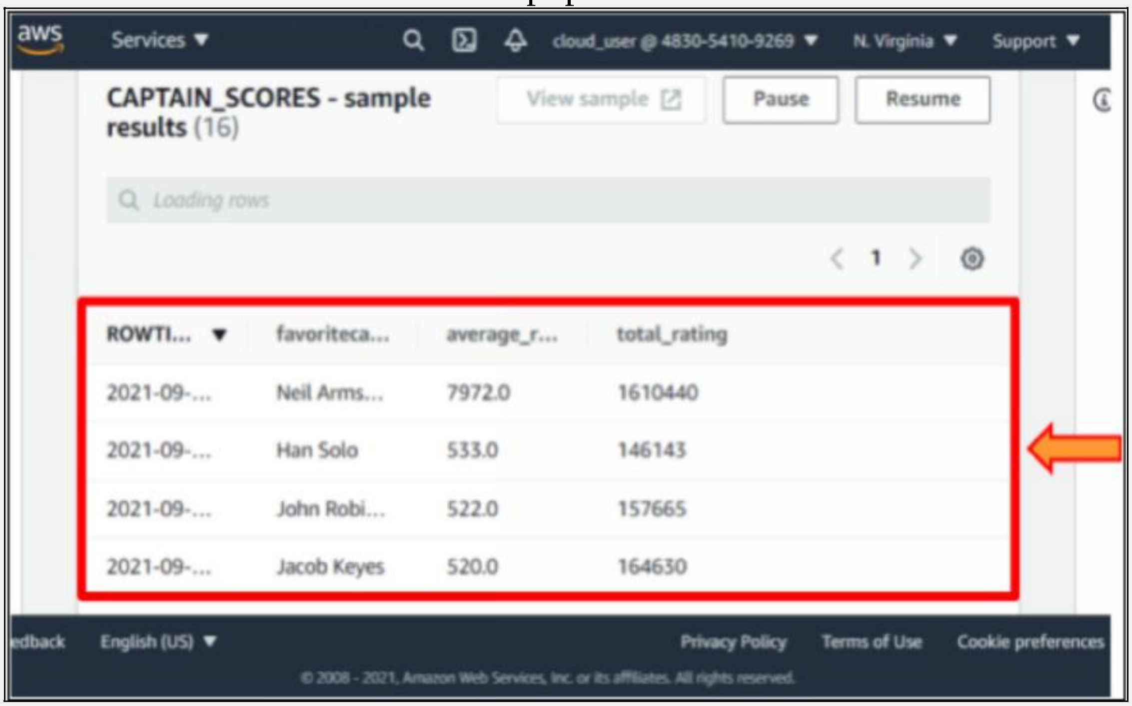
19. Copy and paste the below SQL code.

```
CREATE OR REPLACE STREAM "CAPTAIN_SCORES" ("favoritecaptain"
varchar(32),    "average_rating"    DOUBLE,    "total_rating"
INTEGER);

CREATE OR REPLACE PUMP "STREAM_PUMP" as
INSERT INTO "CAPTAIN_SCORES"
SELECT     STREAM     "favoritecaptain",     avg("rating"),
sum("rating")
FROM SOURCE_SQL_STREAM_001
GROUP          BY          "favoritecaptain",          STEP
("SOURCE_SQL_STREAM_001".ROWTIME BY INTERVAL '1' MINUTE)
ORDER BY STEP("SOURCE_SQL_STREAM_001".ROWTIME BY INTERVAL '1'
MINUTE), avg("rating") desc;
```

20. Click on the **Save and run application** button.



21. It will take a few minutes to populate the table.

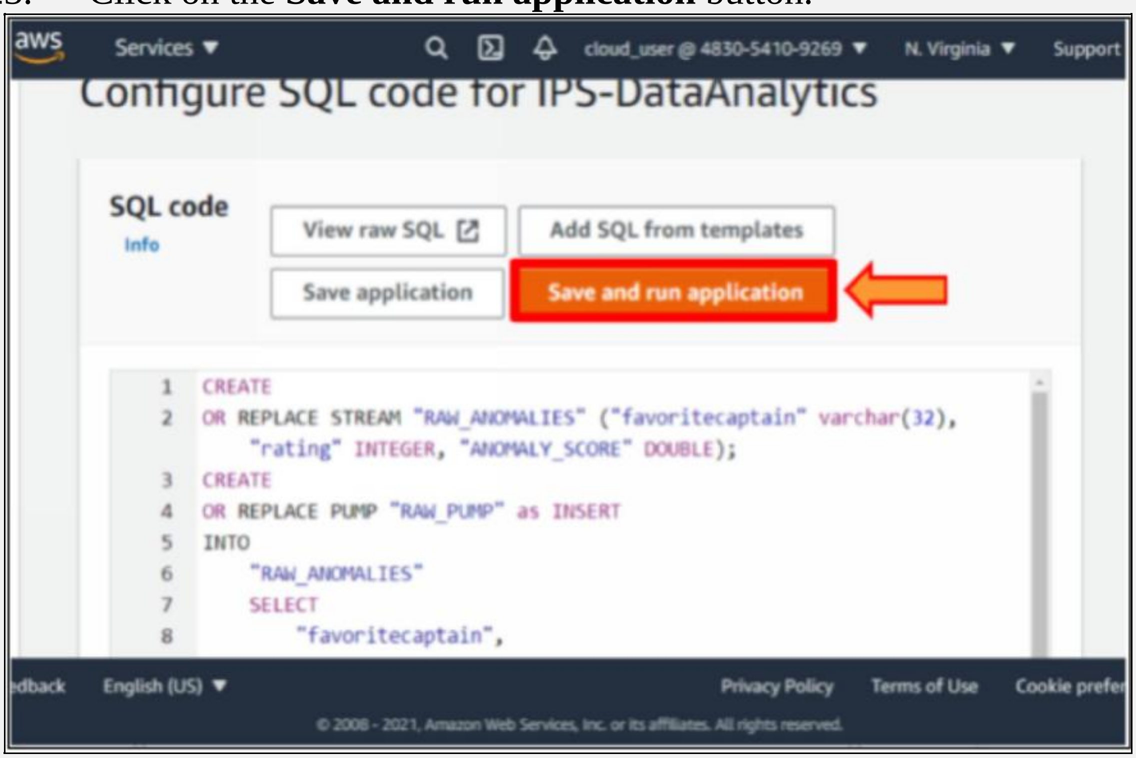


22. Copy and paste the below SQL code to find the anomalous ratings.

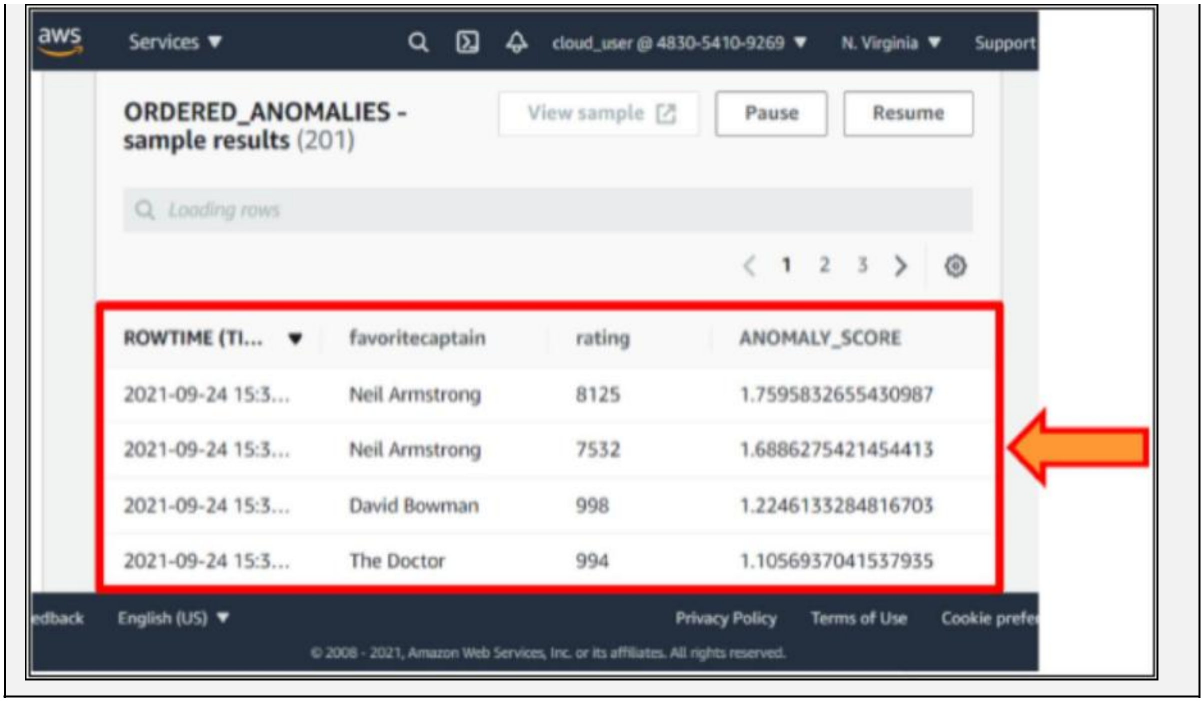
```
CREATE
OR REPLACE STREAM "RAW_ANOMALIES" ("favoritecaptain"
varchar(32), "rating" INTEGER, "ANOMALY_SCORE" DOUBLE);
CREATE
OR REPLACE PUMP "RAW_PUMP" as INSERT
INTO
   "RAW_ANOMALIES"
   SELECT
      "favoritecaptain,"
      "rating",
```

```
        "ANOMALY_SCORE"
    FROM
        TABLE(    RANDOM_CUT_FOREST(CURSOR(SELECT
            STREAM "favoritecaptain",
            "rating"
        FROM
            "SOURCE_SQL_STREAM_001")) );
CREATE
OR REPLACE STREAM "ORDERED_ANOMALIES"
("favoritecaptain" varchar(32),
"rating" INTEGER,
"ANOMALY_SCORE" DOUBLE);
CREATE
OR REPLACE PUMP "ORDERED_PUMP" as INSERT
INTO
    "ORDERED_ANOMALIES"
    SELECT
        STREAM *
    FROM
        RAW_ANOMALIES
    ORDER BY
        FLOOR("RAW_ANOMALIES".ROWTIME TO SECOND),
        "ANOMALY_SCORE" desc;
```

23. Click on the **Save and run application** button.



24. It will take a few minutes to populate the table.



Lab 06: Data Analysis & Visualization

Certifications	AWS Machine Learning - Specialty
Cloud Provider	AWS
Time Required	1 Hr.
Tags	QuickSight, S3, SageMaker, AWS

Lab Prerequisites

- Familiarity with basic AWS Cloud Computing concepts and terminology
- An AWS account with an active subscription

Service Introduction

AWS QuickSight

Amazon QuickSight is a cloud-based business analytics solution that enables all employees to create visualizations, do ad-hoc analysis, and quickly get business insights from their data, anytime and on any device. Upload CSV and Excel files; connect to SaaS apps like Salesforce; connect to on-premises databases like SQL Server, MySQL, and PostgreSQL; and locate your AWS data sources in real-time, including Amazon Redshift, Amazon RDS, Amazon Aurora, Amazon Athena, and Amazon S3. QuickSight allows businesses to extend their business analytics capabilities to thousands of users while still providing quick and responsive query processing. QuickSight enables businesses to expand their business analytics capabilities to thousands of users. It uses a powerful in-memory engine to offer quick and responsive query performance (SPICE).

AWS SageMaker

Amazon SageMaker is a fully managed service that allows any developer or data scientist to quickly create, train, and deploy Machine Learning (ML) models. SageMaker makes it easy to create high-quality models by removing the heavy lifting from each phase of the machine learning process.

AWS Simple Storage Service S3

Amazon S3 is a type of object storage that allows you to store and recover any quantity of data from any location. It is a low-cost storage solution with business resilience, reliability, efficiency, privacy, and infinite expansion.

Amazon S3 is a web service that allows you to store and retrieve an infinite quantity of data from any place and at any time. You may quickly create projects that integrate cloud-native storage using this service because Amazon S3 is easily customizable, and you only pay for what you use. You can start small and scale up as needed without sacrificing performance or dependability.

Amazon S3 is also built to be highly adaptable. Instead of finding out how to store their data, Amazon S3 allows developers to focus on innovation. Build a simple FTP system or a complex web application like the Amazon.com retail website; read the same piece of data a million times or only for emergency disaster recovery; store whatever type and amount of data you desire.

Case Study Enterprise Public Service – IPS

Background

The Social Security Institute, known as IPS is decentralized public service with its own legal team and resources that was established in accordance with Article 53 of Law 20.255, which implemented a Social Security System overhaul in 2008. Using the Ministry of Labor and Social Welfare as a conduit, as well as the Undersecretary of Social Welfare, it operates under the authority of the President of the Republic. This service, which is a public utility overseen by Chile's Senior Public Management System, is responsible for managing the solidarity pension system and pension plans formerly managed by the former INP. Additionally, it runs the ChileAtiende network, which informs citizens on policies and government benefits.

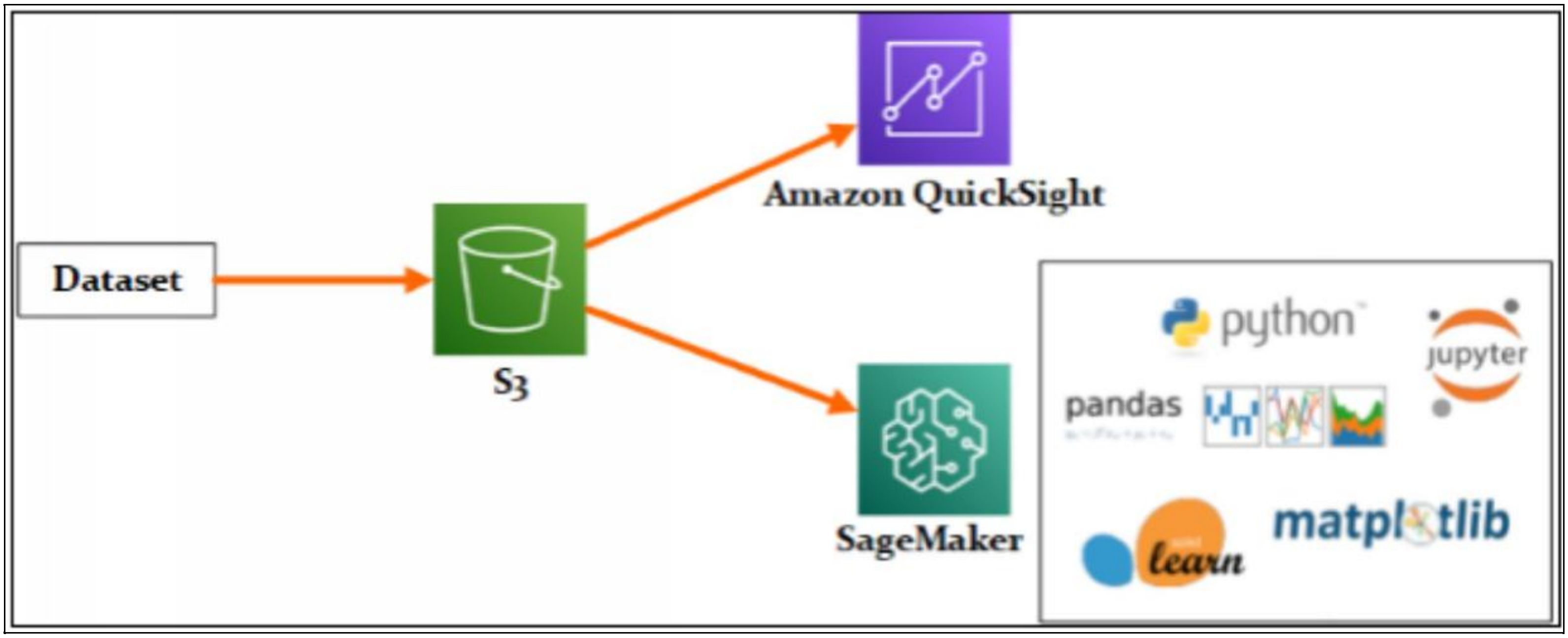
Business Challenge

You work as a Data Analytics in IPS. IPS gives you the task of analyzing and visualizing the sales data of the car. To see the sales figures using the visualization of data. Which month generates the most sales? Which dealership sold the most cars? How can you automate visualizing and analyzing the data with drag and drop?

Proposed Solution

The solution is to use AWS services to visualize and analyze the data. You use AWS QuickSight service to drag and drop the visualizing and analyzing of data. Using Python libraries, you can also use AWS SageMaker Jupyter notebook to visualize the data. In this lab, you use AWS S3 to upload your data. Then, you will use AWS QuickSight to visualize data using a pie chart and vertical bar chart. For the box plot, you will use AWS SageMaker Jupyter Notebook.

Lab Diagram



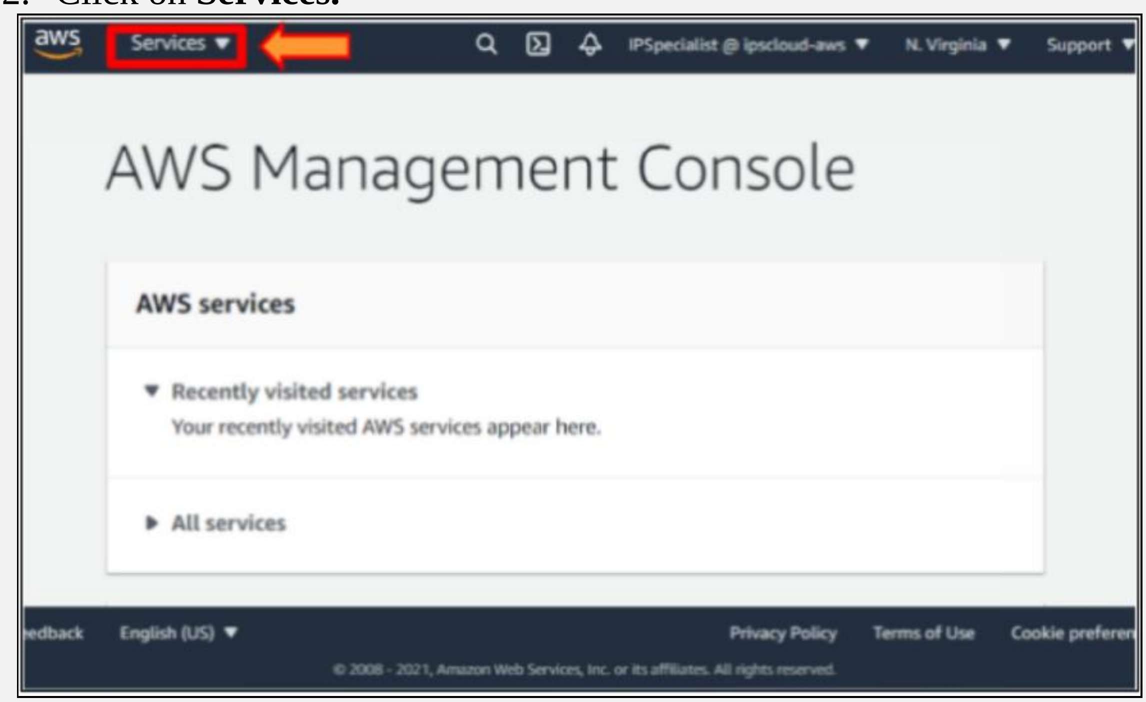
Implementation Steps

1. Create an S3 Bucket.
2. Visualize Data on AWS QuickSight.
3. Visualize Data on AWS SageMaker.

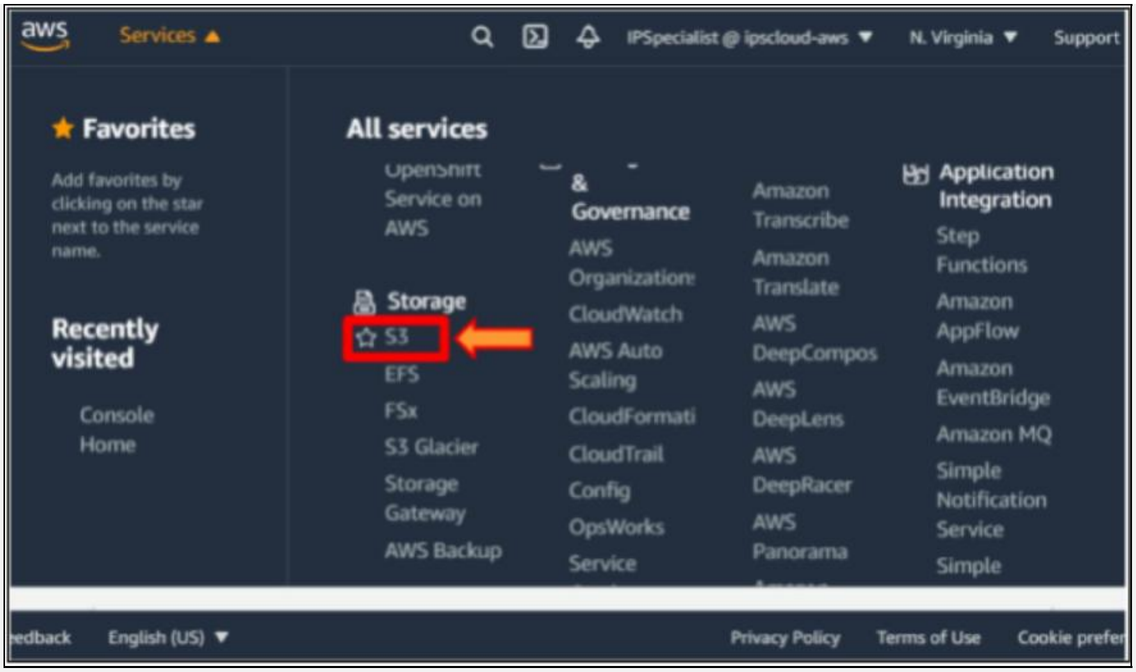
Solution

Step 1: Create an S3 Bucket

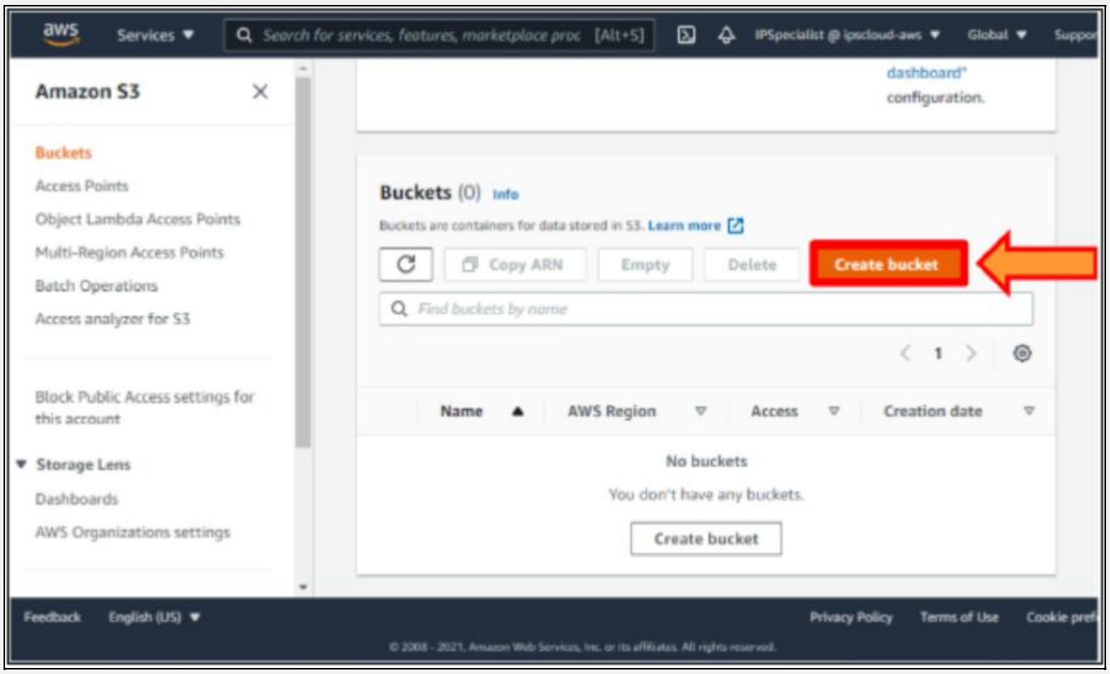
1. Log in to the **AWS Console**.
2. Click on **Services.**



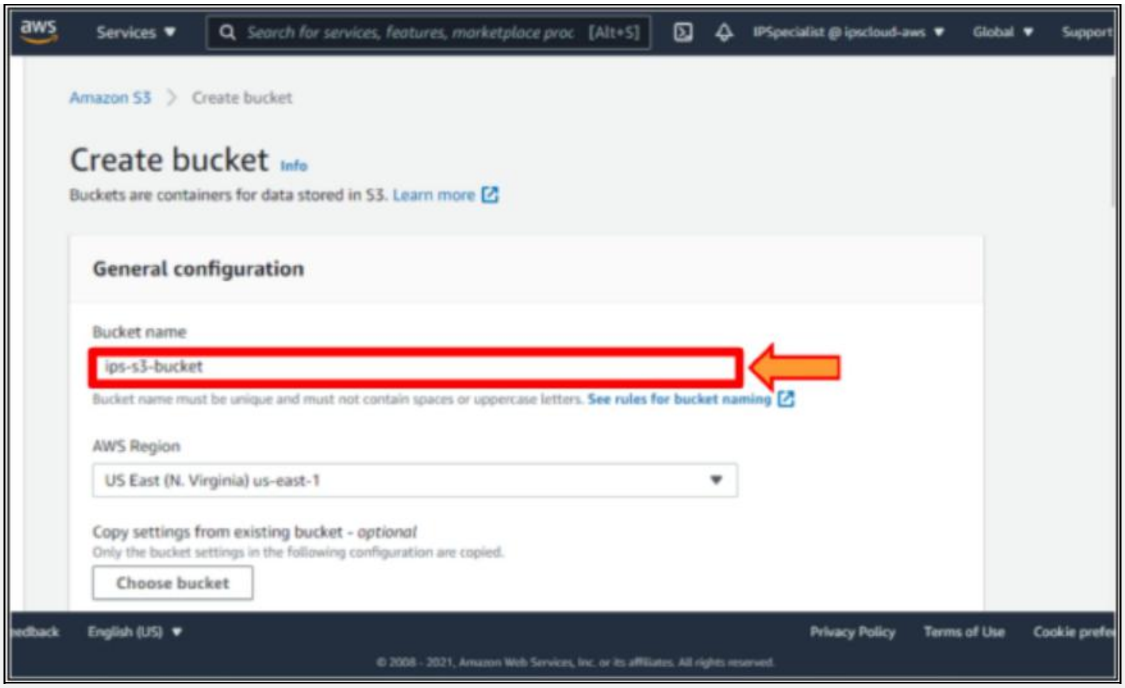
3. Select **S3** from the **Storage**.



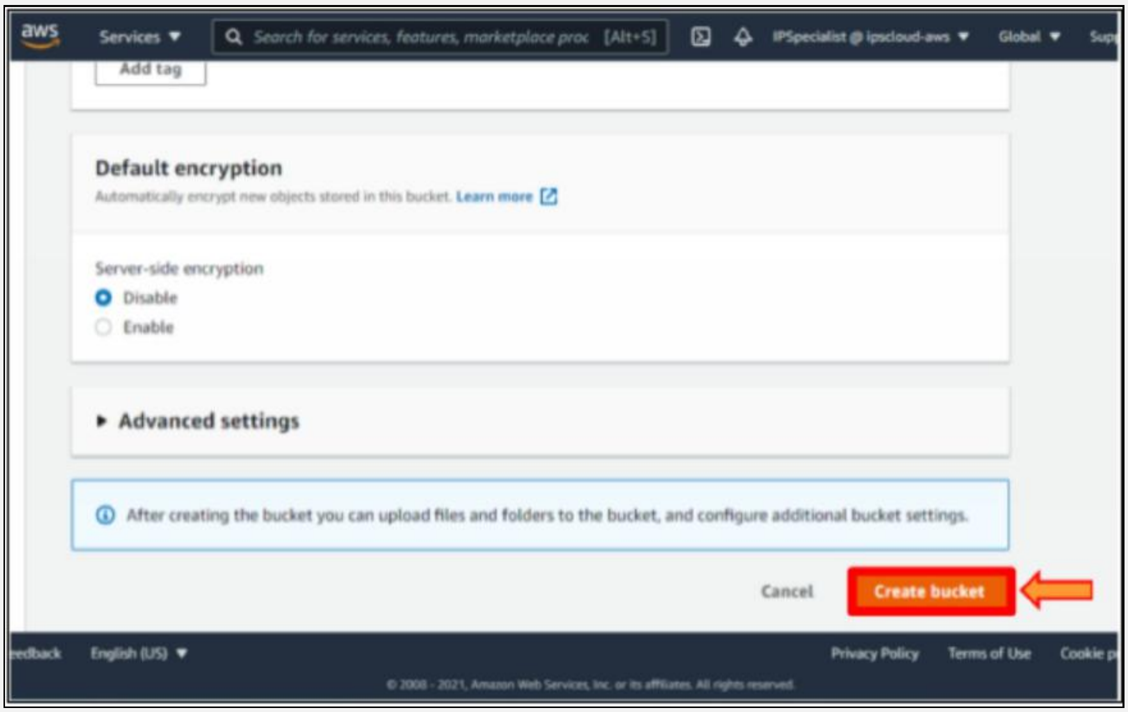
4. Click on the **Create bucket** button.



5. Give a bucket name **ips-s3-bucket.**



6. Scroll down. Click on the **Create bucket** button.

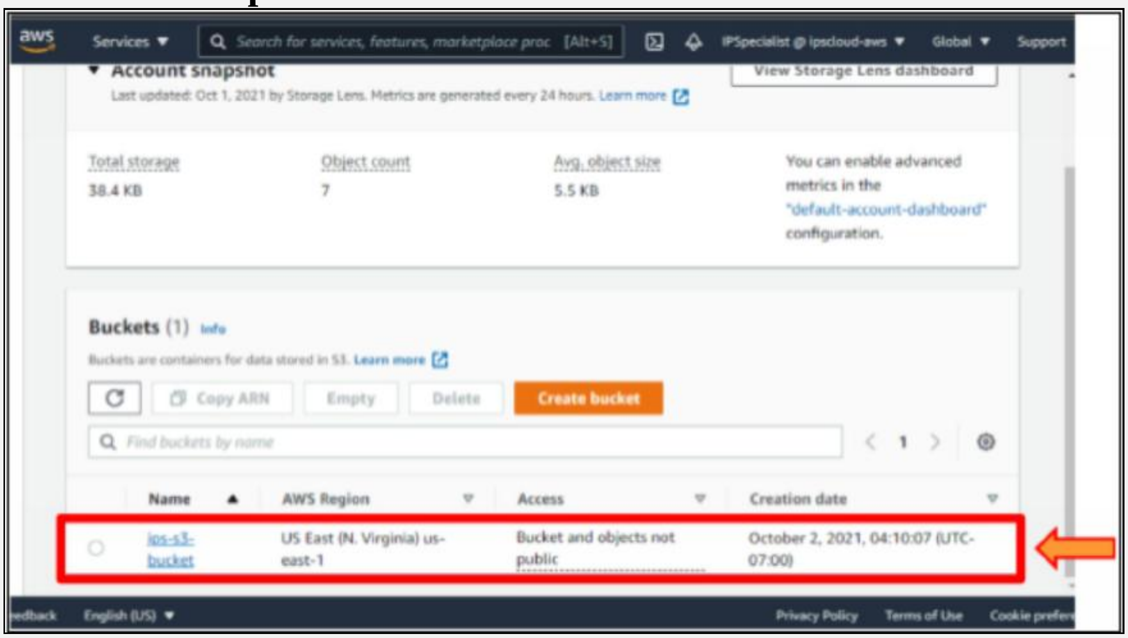


7. Download the car dataset provided in the following GitHub link: https://github.com/ACloudGuru-Resources/Course_AWS_Certified_Machine_Learning/blob/master/Chapter5/car_data.csv

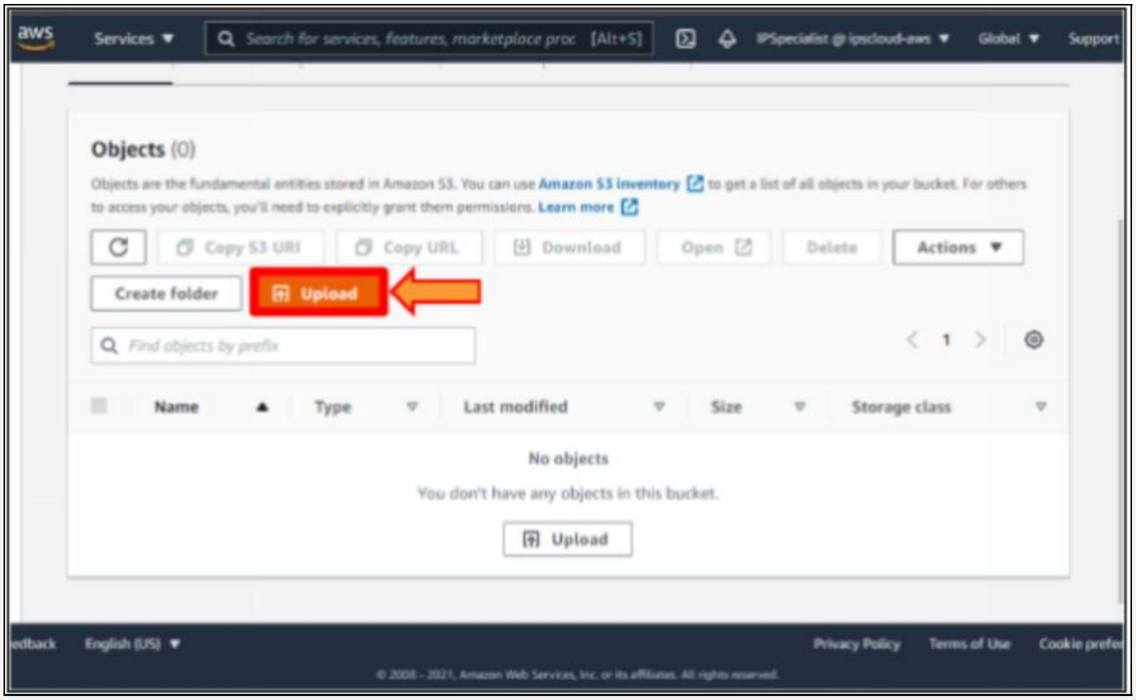
Search this file...

	car	year	engine_hp	avg_mpg	price	salesman	dealership	sold_date	sold_month
1	car	year	engine_hp	avg_mpg	price	salesman	dealership	sold_date	sold_month
2	Corvette	2011	335.0	22.5	46135	2	Big Bobs	2012-05-06	5
3	Corvette	2011	300.0	23.5	40650	2	Uptown Cars	2011-05-16	5
4	Corvette	2011	300.0	24.0	36350	2	Uptown Cars	2013-07-31	7
5	Corvette	2011	230.0	23.0	29450	2	Uptown Cars	2014-07-05	7
6	Corvette	2011	230.0	23.0	34500	2	Uptown Cars	2013-05-20	5
7	Corvette	2012	230.0	23.0	31200	3	Car Town	2013-04-10	4
8	Corvette	2012	300.0	21.5	44100	3	Uptown Cars	2015-12-13	12
9	Corvette	2012	300.0	24.0	39300	2	Big Bobs	2013-05-19	5
10	Corvette	2012	230.0	23.0	36900	3	Uptown Cars	2015-04-27	4
11	Corvette	2013	230.0	22.5	37200	2	Car Town	2013-12-15	12
12	Corvette	2013	300.0	24.0	39600	3	Big Bobs	2015-05-30	5
13	Corvette	2013	230.0	23.5	31500	2	Uptown Cars	2014-03-21	3
14	Corvette	2013	300.0	23.5	44400	3	Uptown Cars	2016-03-18	3
15	Corvette	2013	230.0	23.5	37200	3	Uptown Cars	2013-02-19	2
16	Corvette	2013	230.0	23.5	31500	3	Car Town	2013-10-27	10
17	Corvette	2013	320.0	21.5	48250	2	Car Town	2016-01-27	1
18	Corvette	2013	320.0	24.0	43550	3	Big Bobs	2014-03-25	3

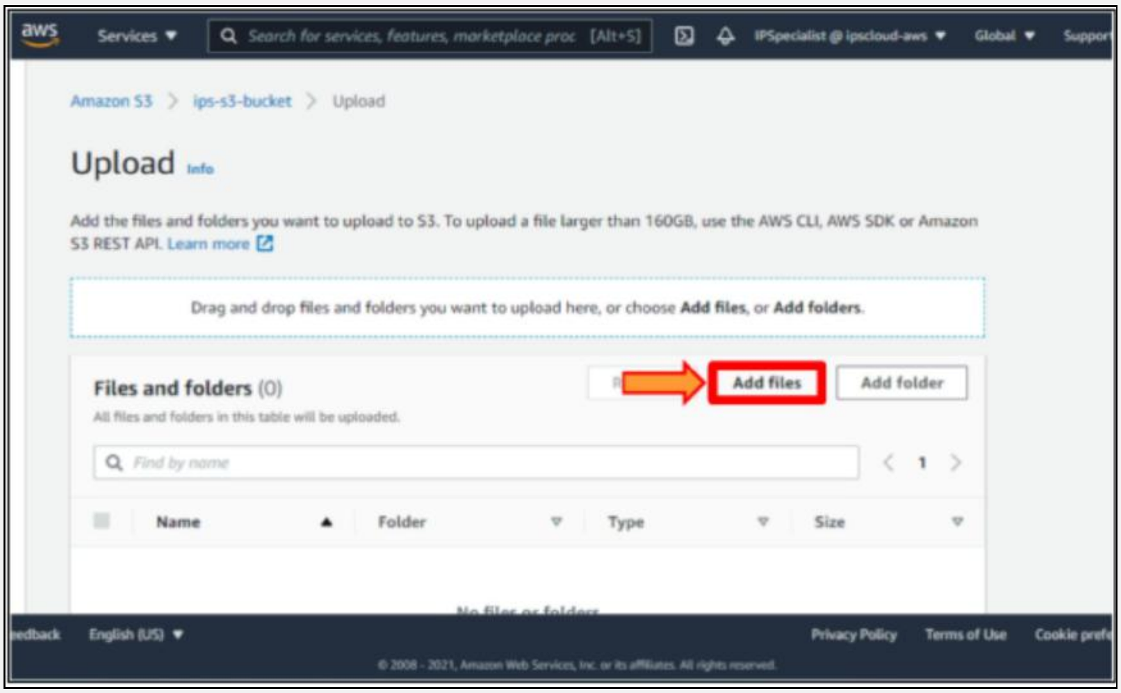
8. Click on the **ips-s3-bucket**.



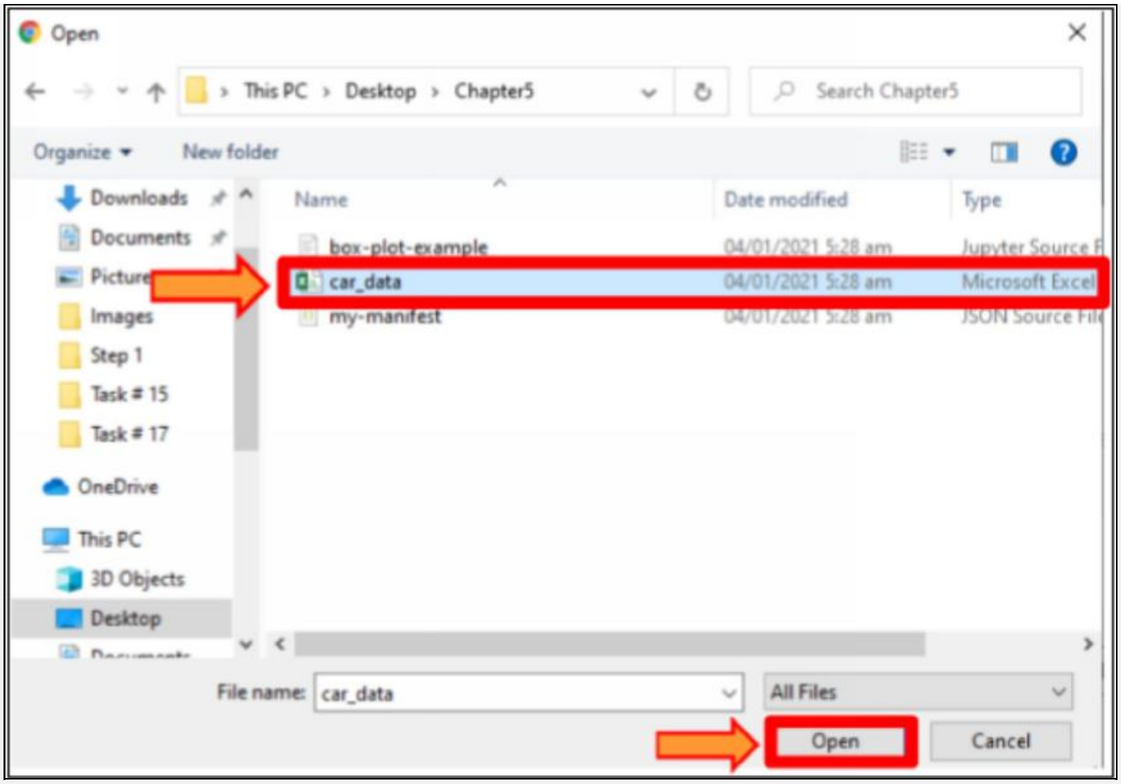
9. Click on the **Upload** button.



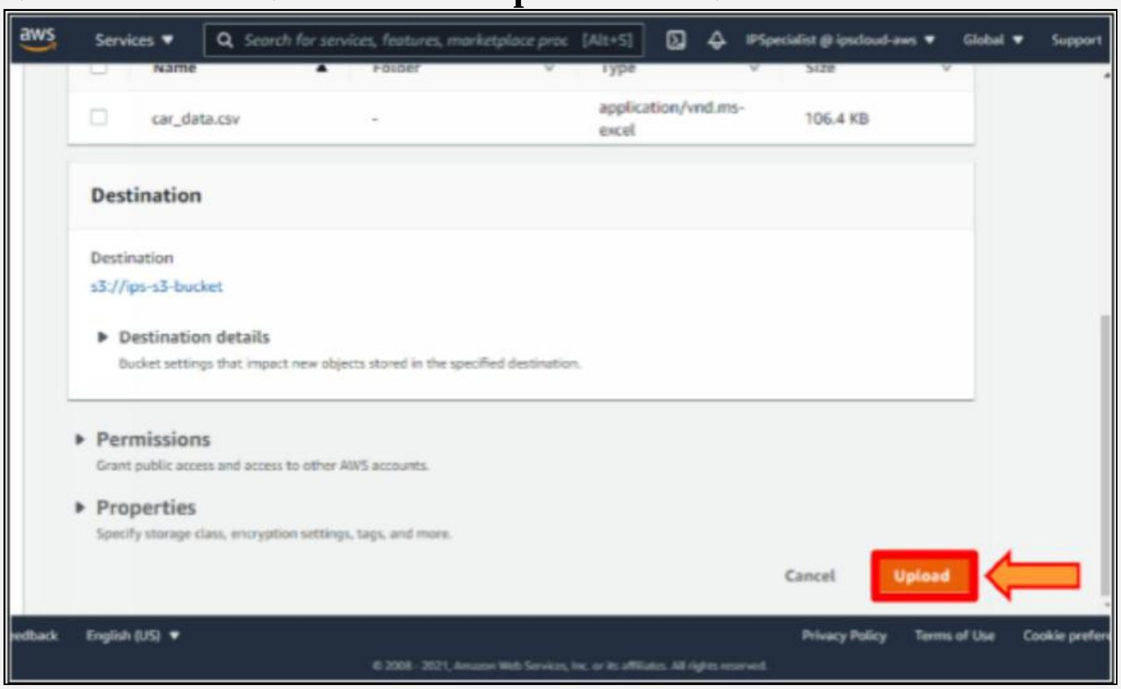
10. Click on the **Add files** button.



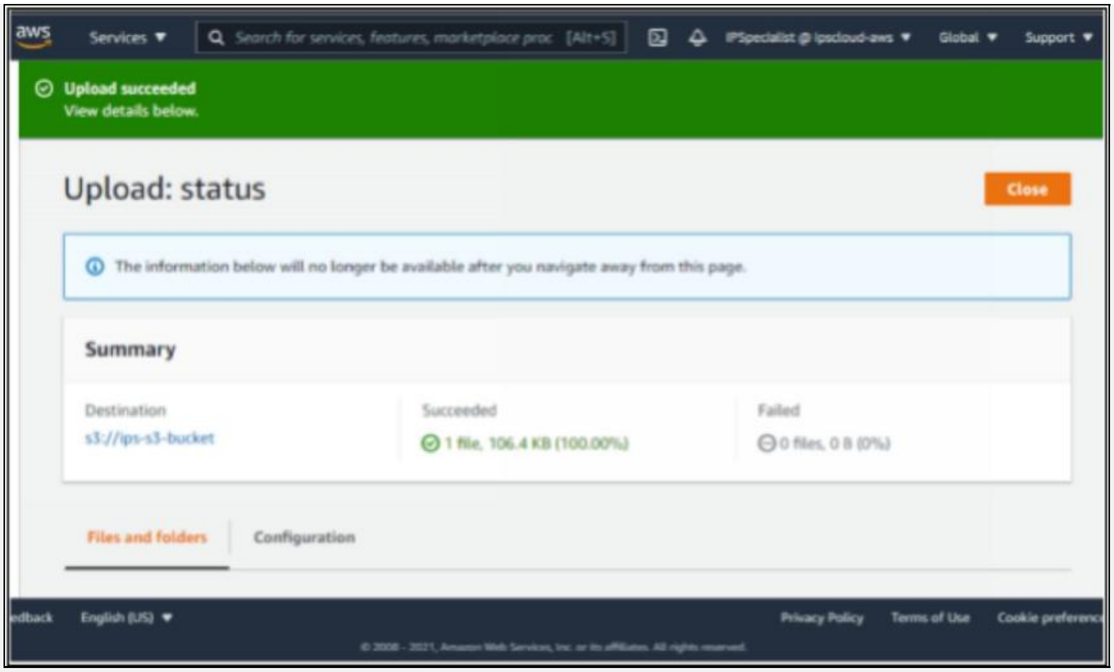
11. Select the **car_data.csv.** Then click on the **Open** button.



12. Scroll down. Click on the **Upload** button.

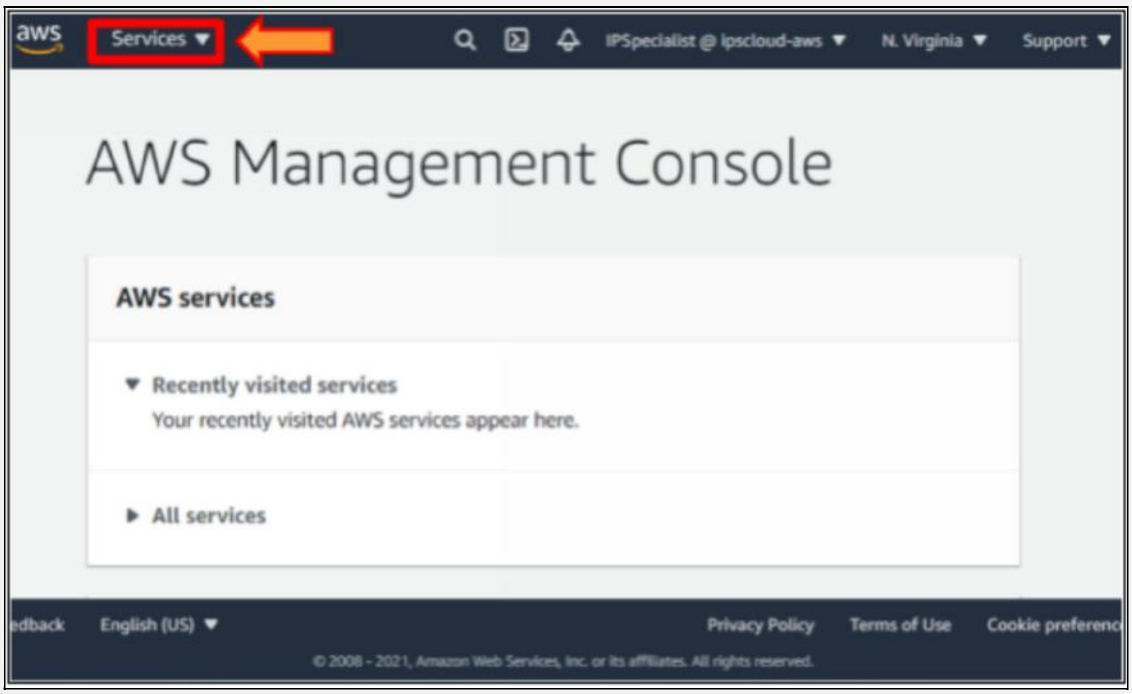


13. So now, the file has been uploaded.

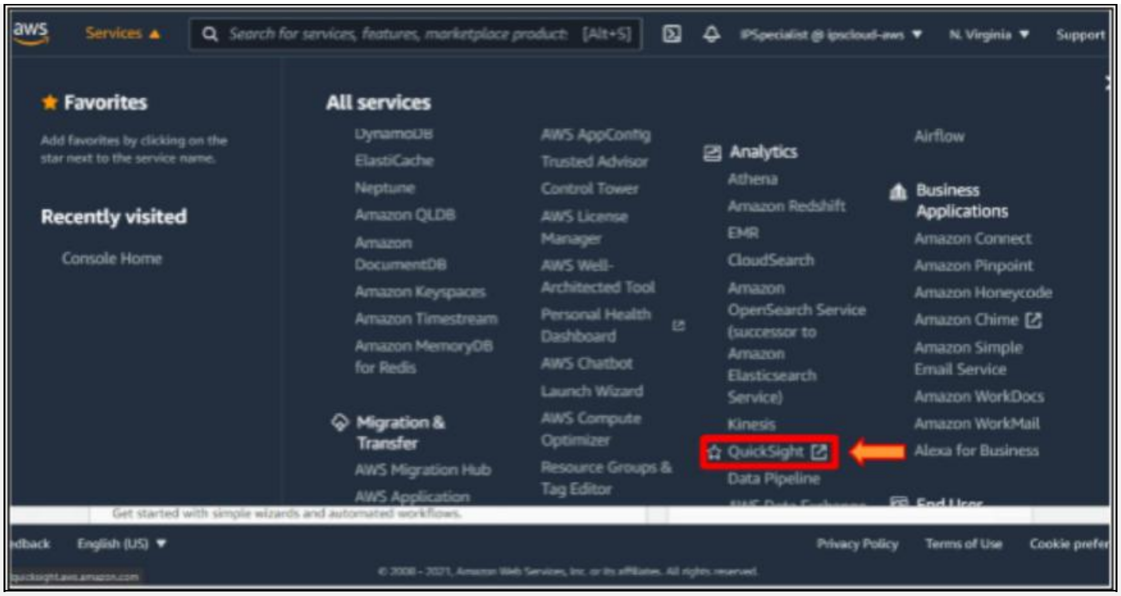


Step 2: Visualize Data on AWS QuickSight

1. Click on **Services.**

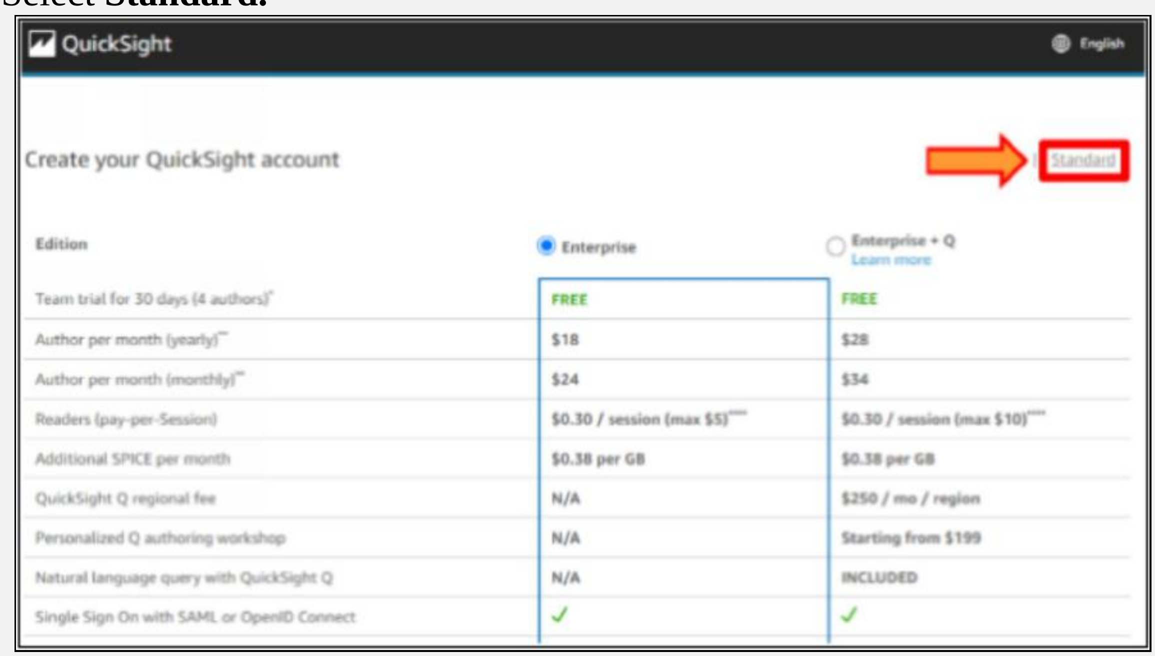


2. Select the **QuickSight** from the **Analytics**.



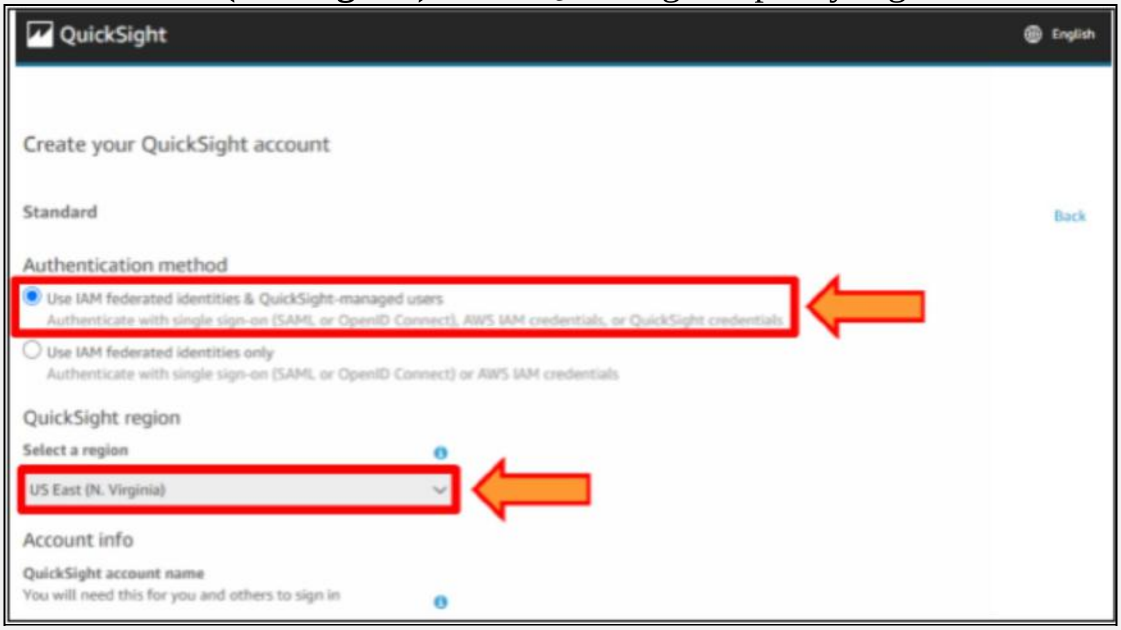
3. Click on the **Sign up for QuickSight** button.

4. Select **Standard.**

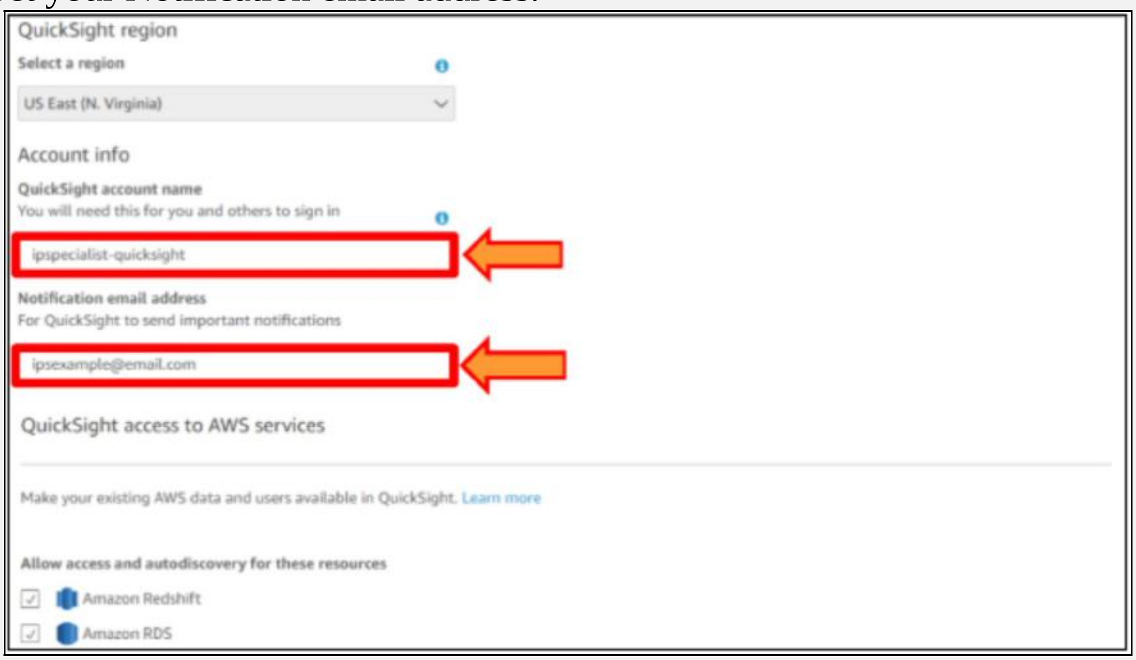


5. Select **Use IAM federated identities & QuickSight-managed users**.

6. Select **US East (N. Virginia)** in the QuickSight capacity region.

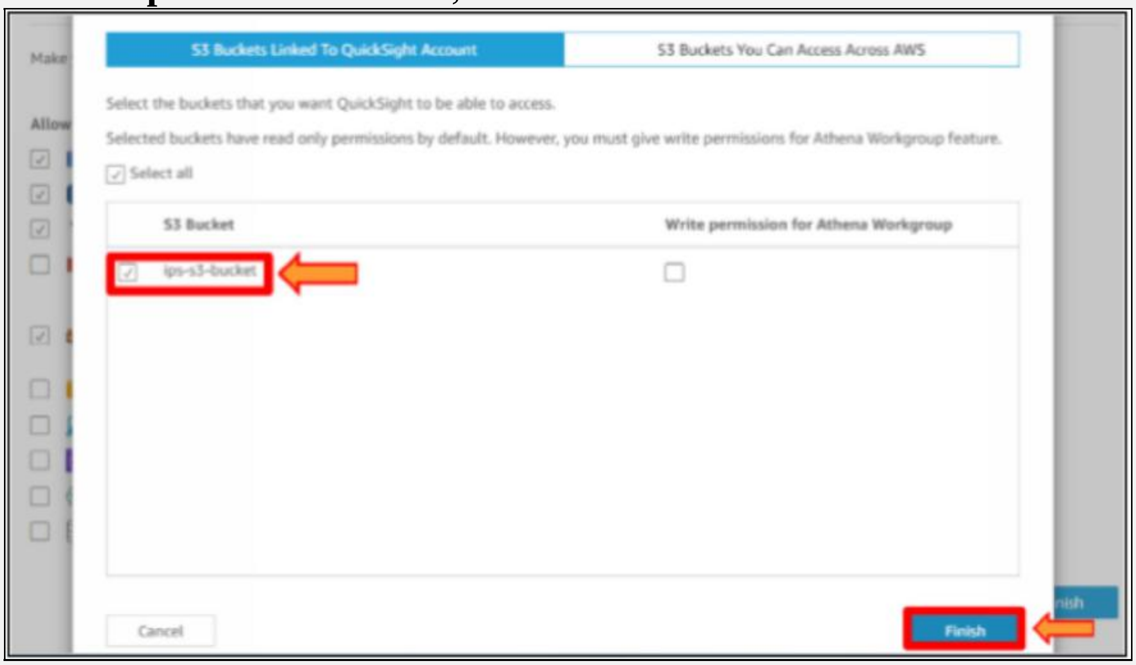


7. Name your QuickSight account.
8. Set your Notification email address.



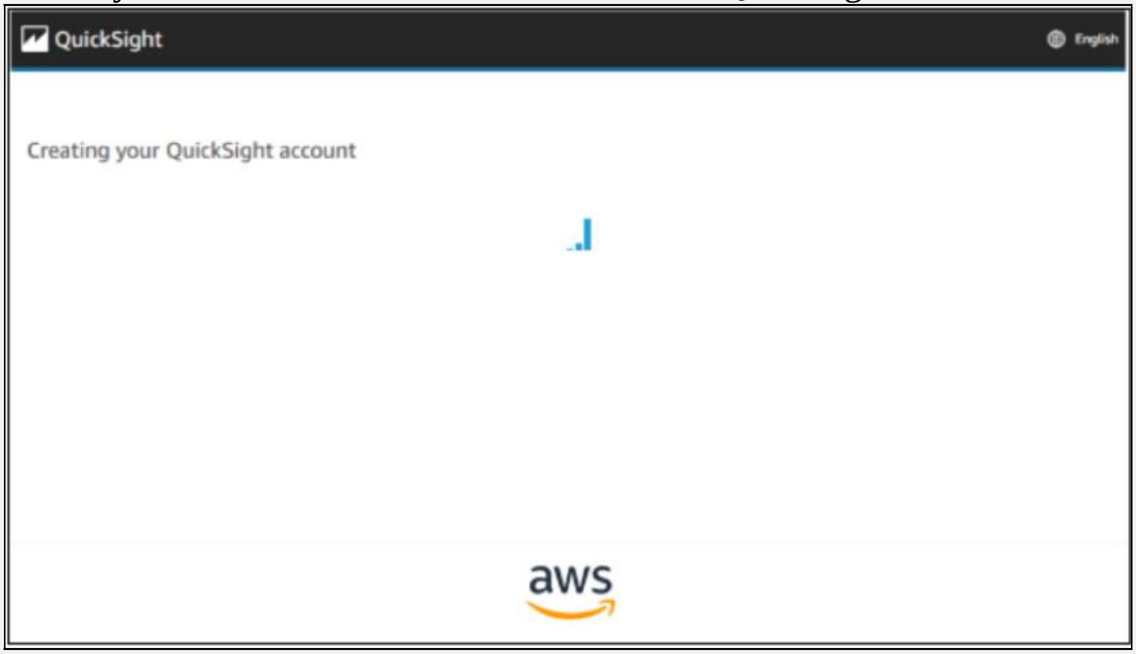
9. Click on the **Amazon S3** Check box.

10. Select **ips-s3-bucket.** Then, click on the **Finish** button.

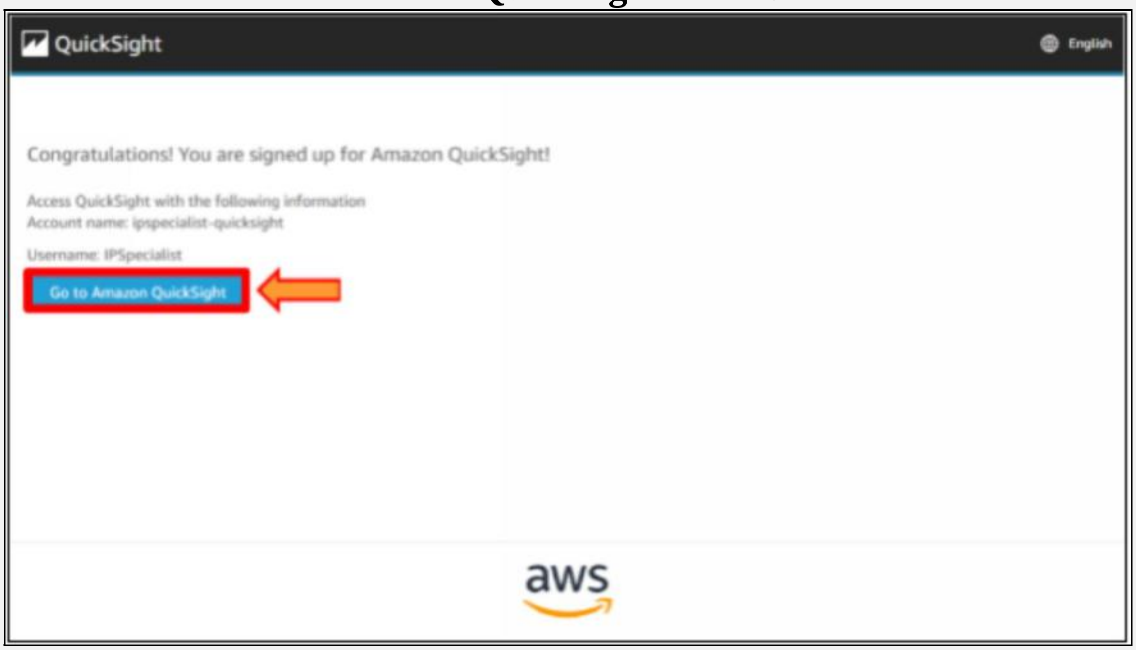


11. Click on the **Finish** button.

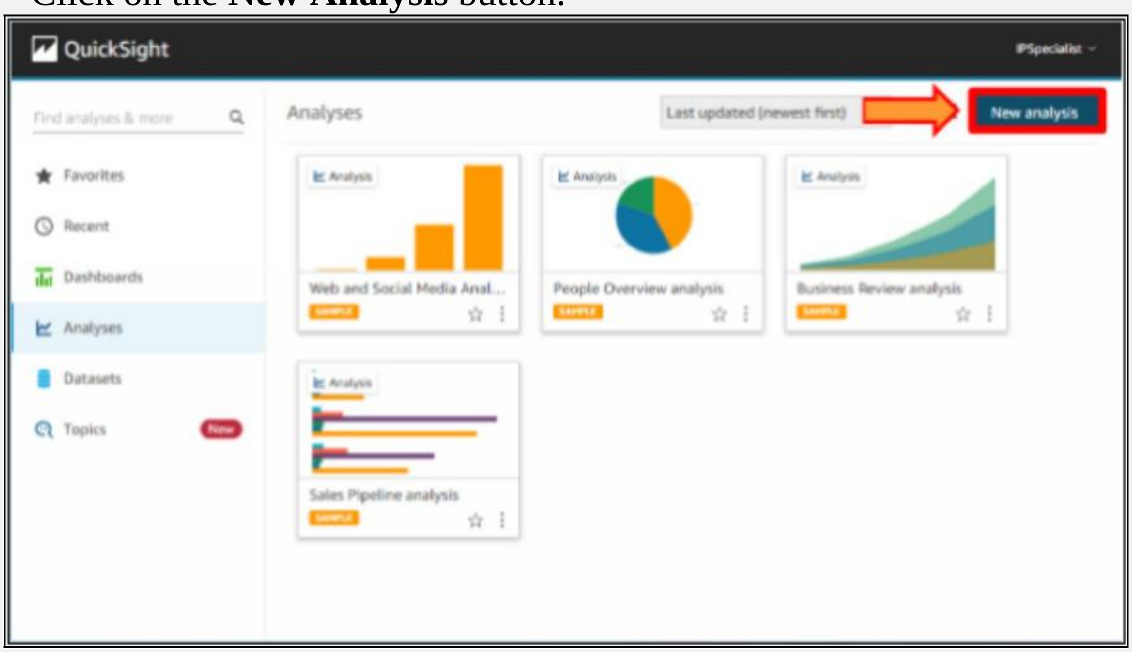
12. It may take a few minutes to create an AWS QuickSight account.



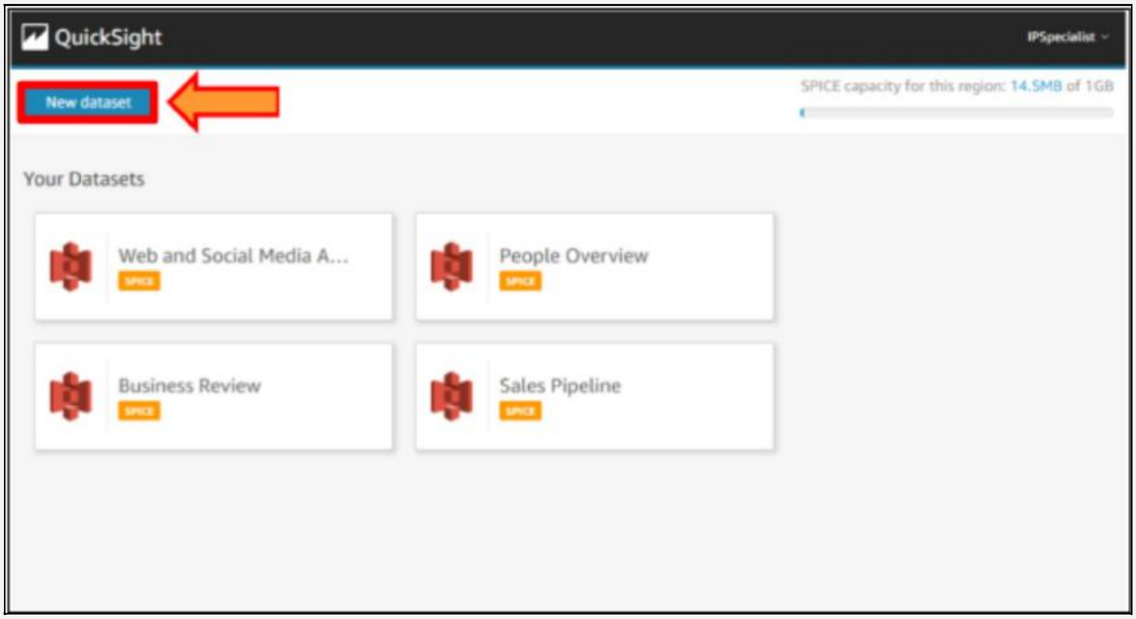
13. Click on the **Go to Amazon QuickSight** button.



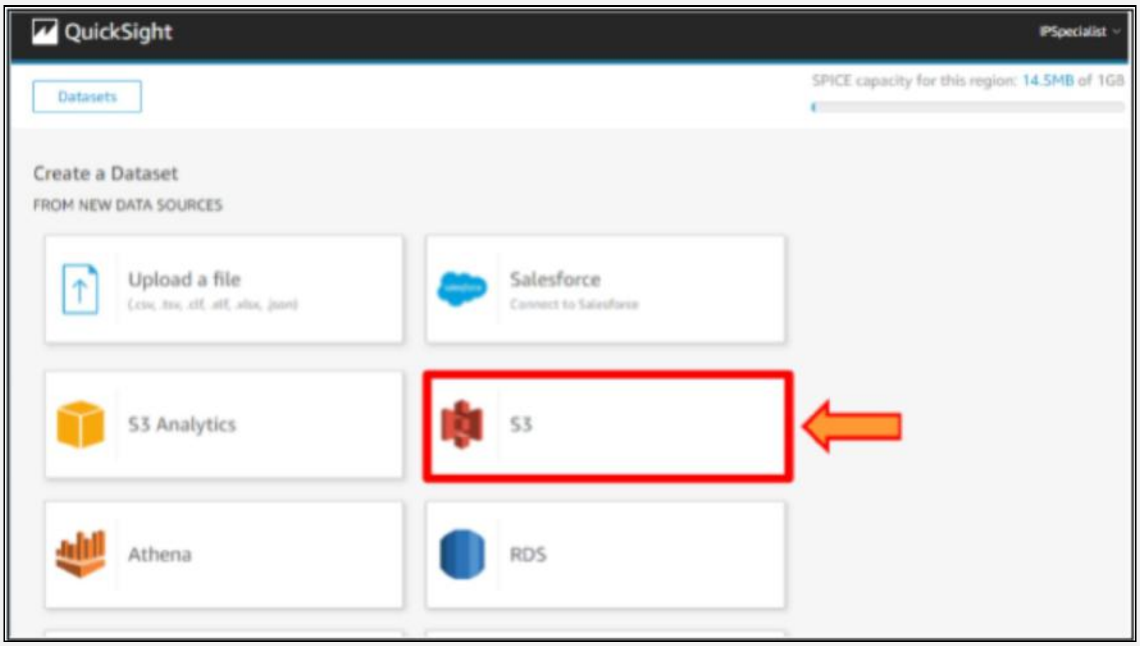
14. Click on the **New Analysis** button.



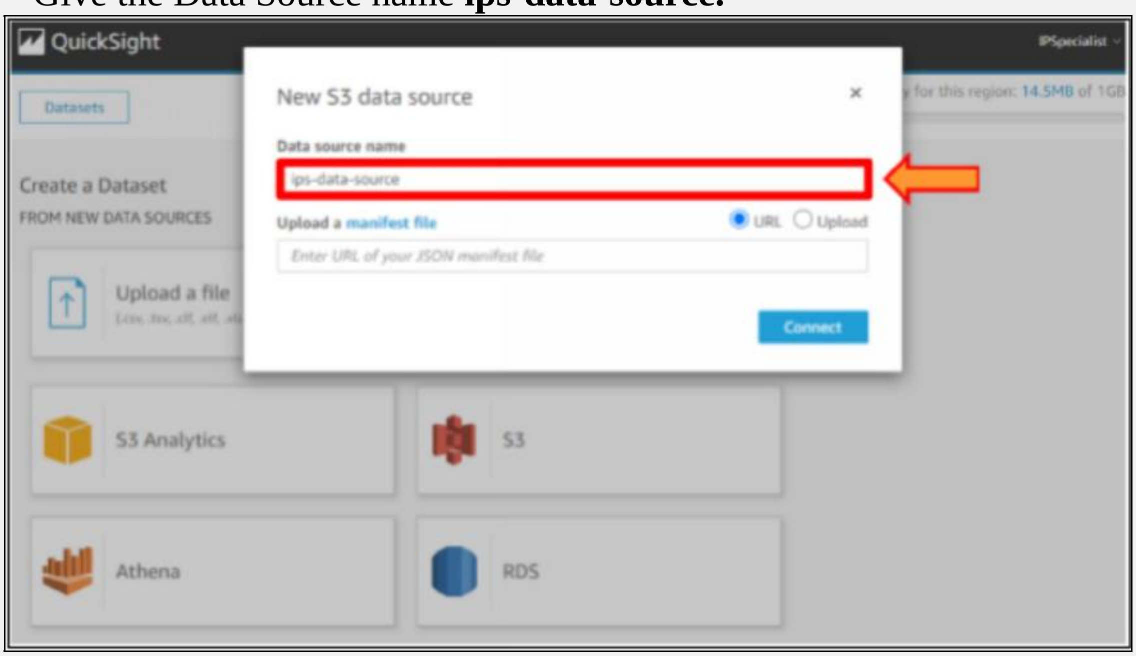
15. Click on the **New dataset** button.



16. Click on the **S3.**

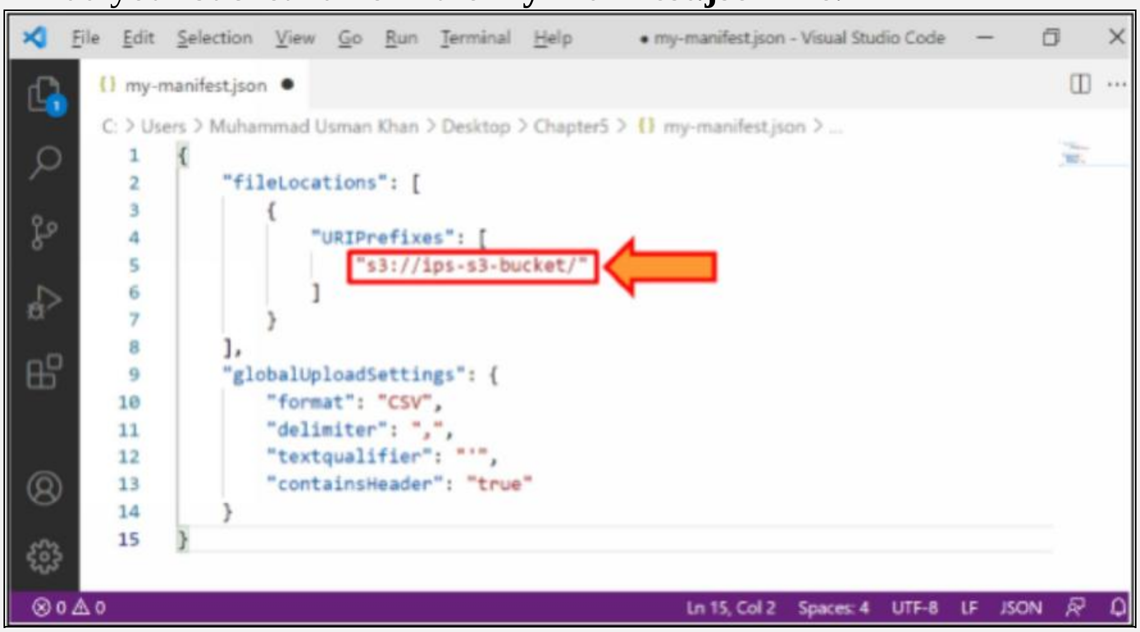


17. Give the Data Source name **ips-data-source.**

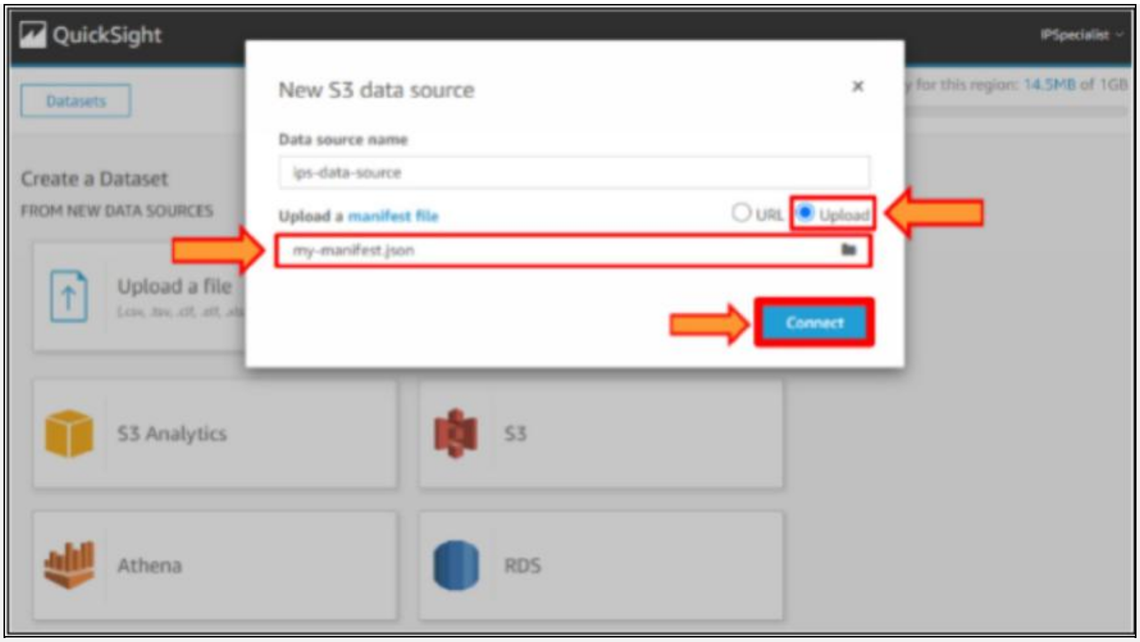


18. Download the my-manifest.json file from the following GitHub link: https://github.com/ACloudGuru-Resources/Course_AWS_Certified_Machine_Learning/blob/master/Chapter5/my-manifest.json

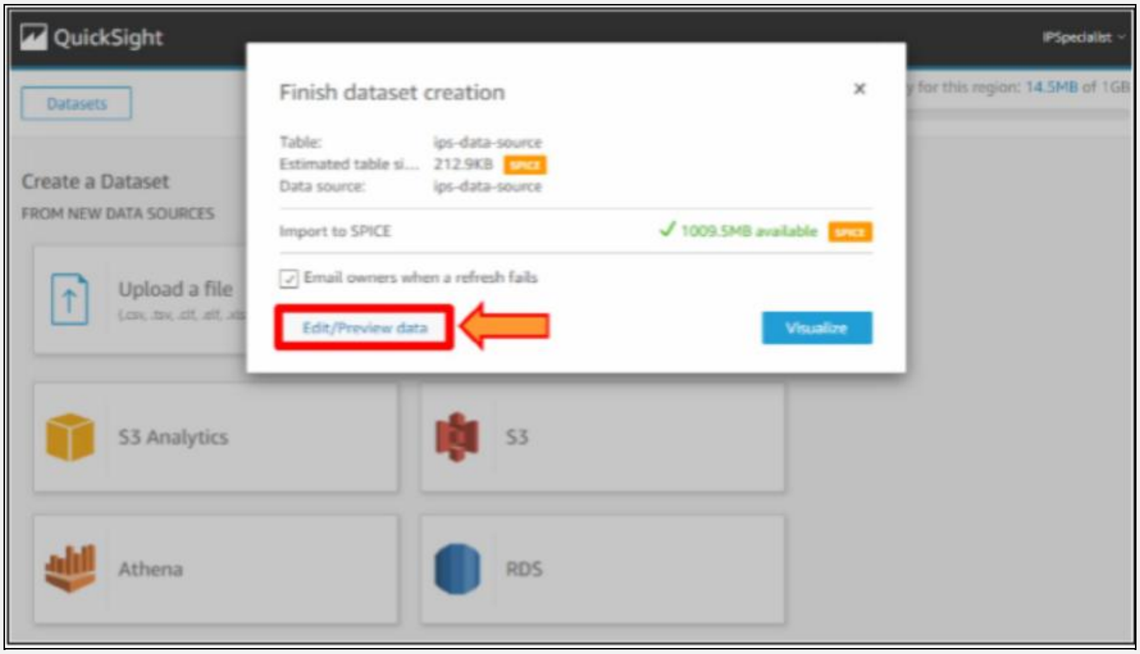
19. Add your bucket name in the **my-manifest.json** file.



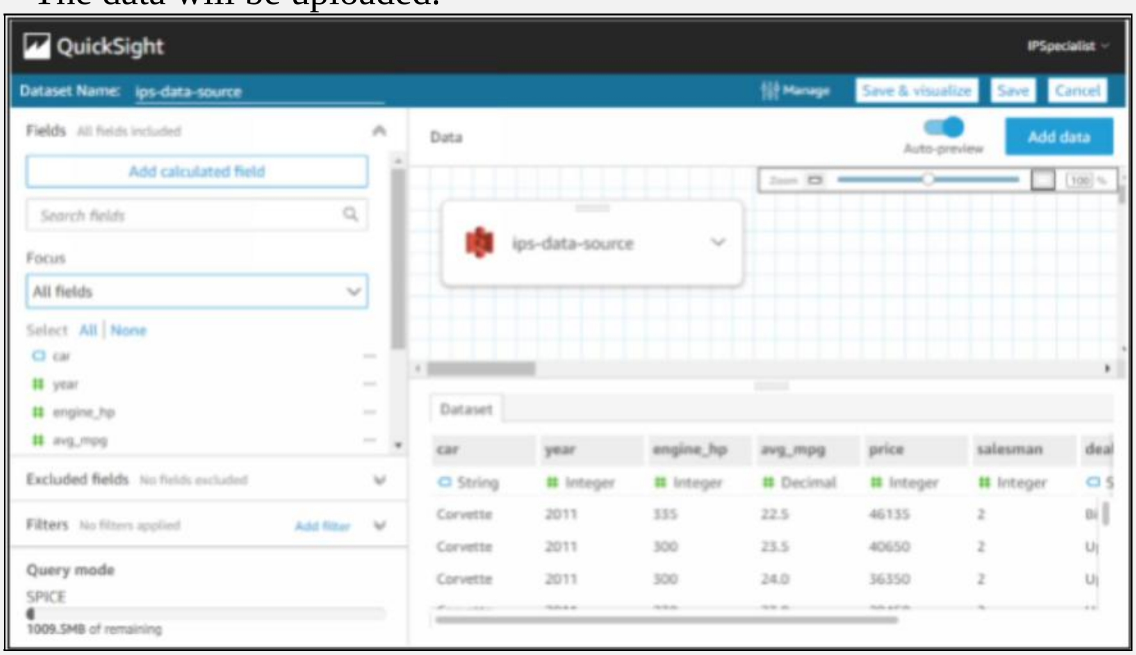
20. Select **Upload**. Then, upload the **my-manifest.json** file.
21. Click on the **Connect** button.



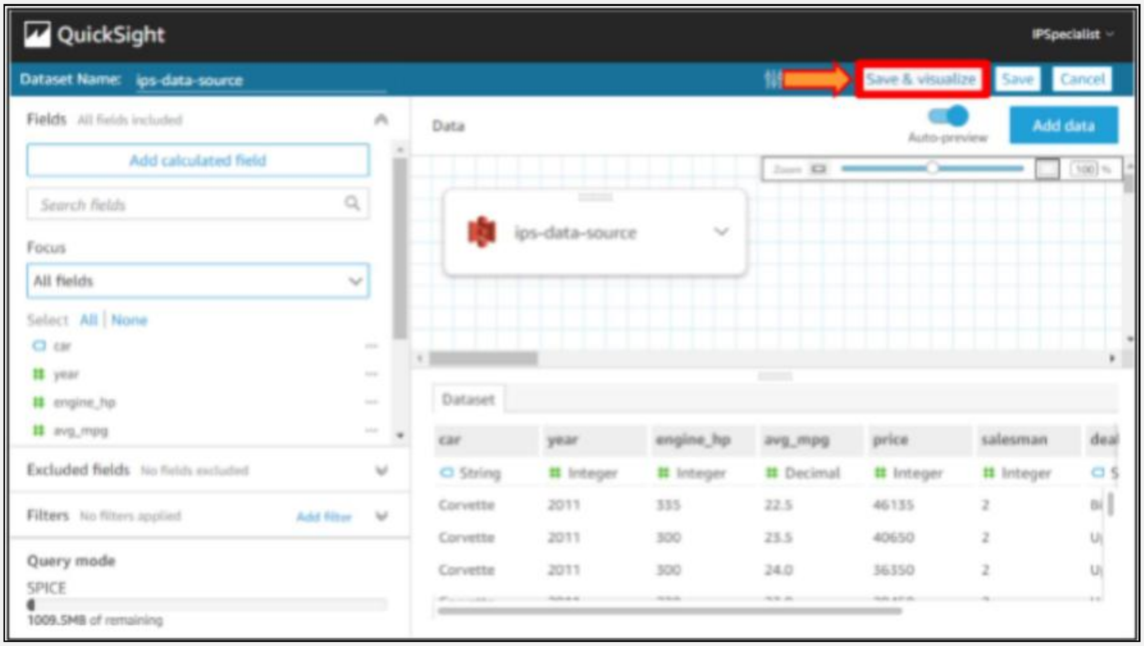
22. Click on the **Edit and Preview** button.



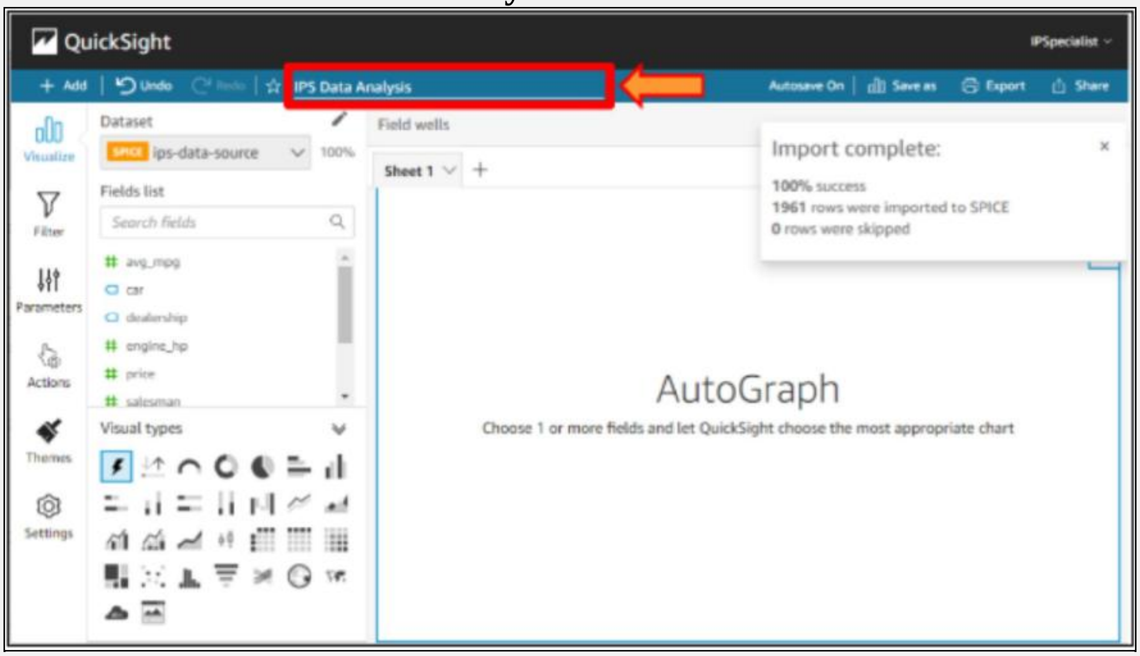
23. The data will be uploaded.



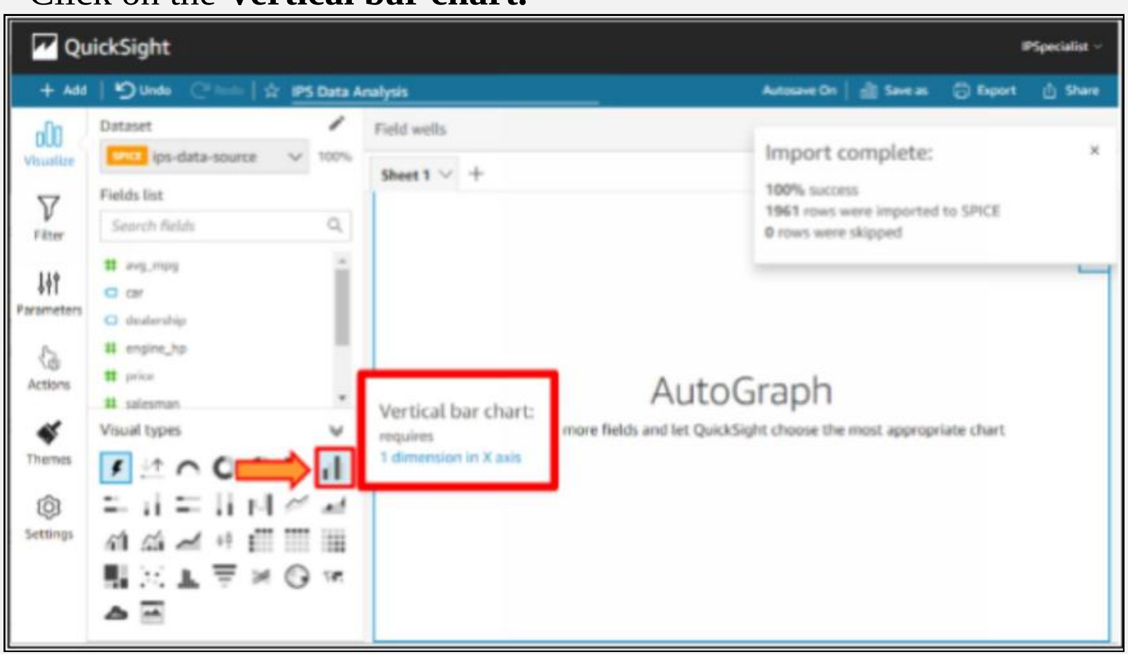
24. Click on the **Save & Visualize** button.



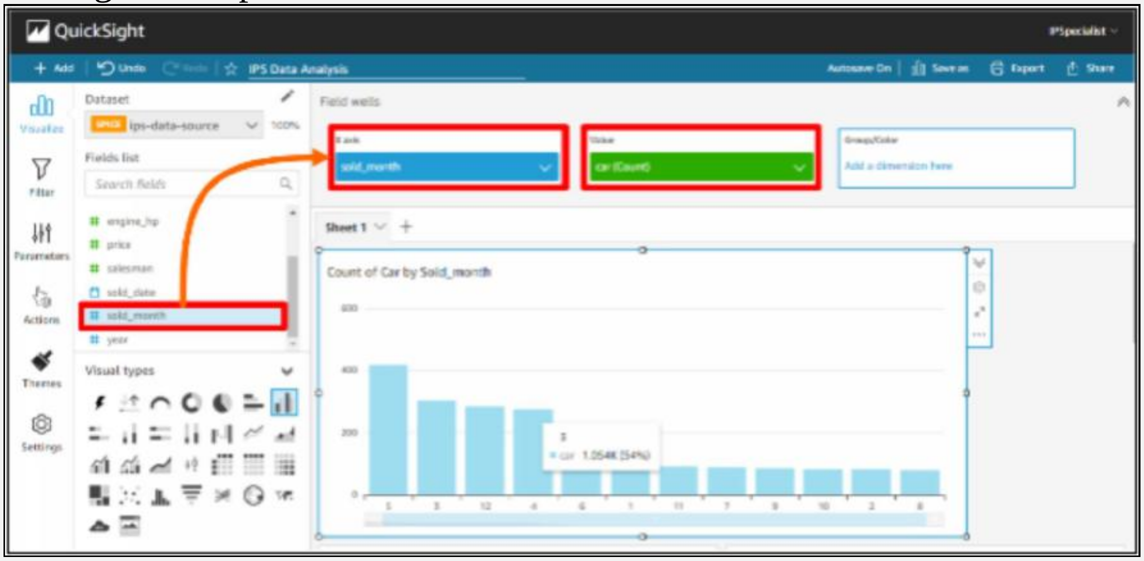
25. Give the name **IPS Data Analysis.**



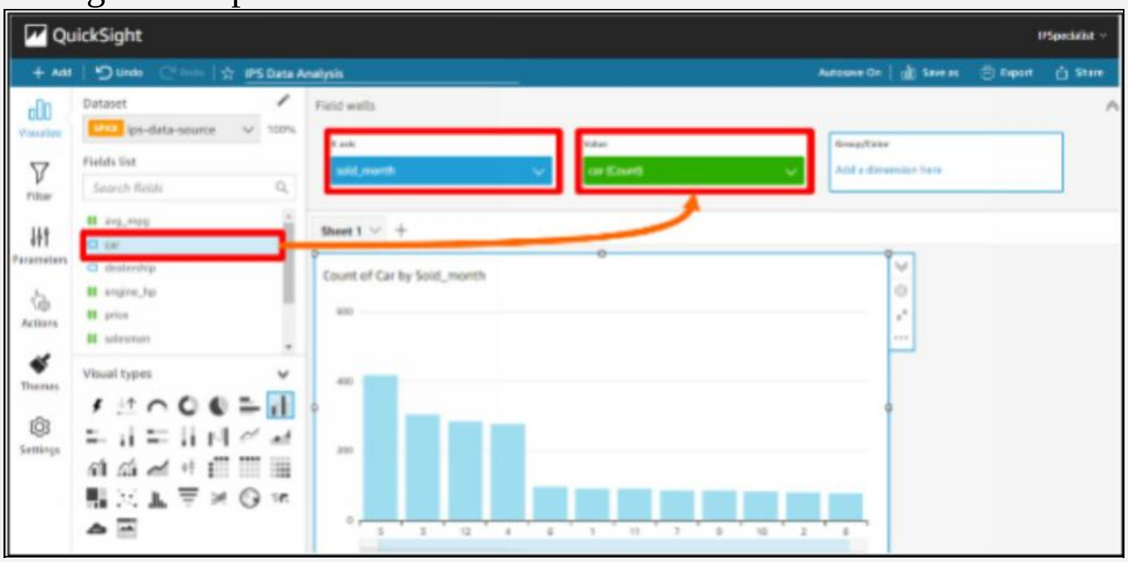
26. Click on the **Vertical bar chart.**



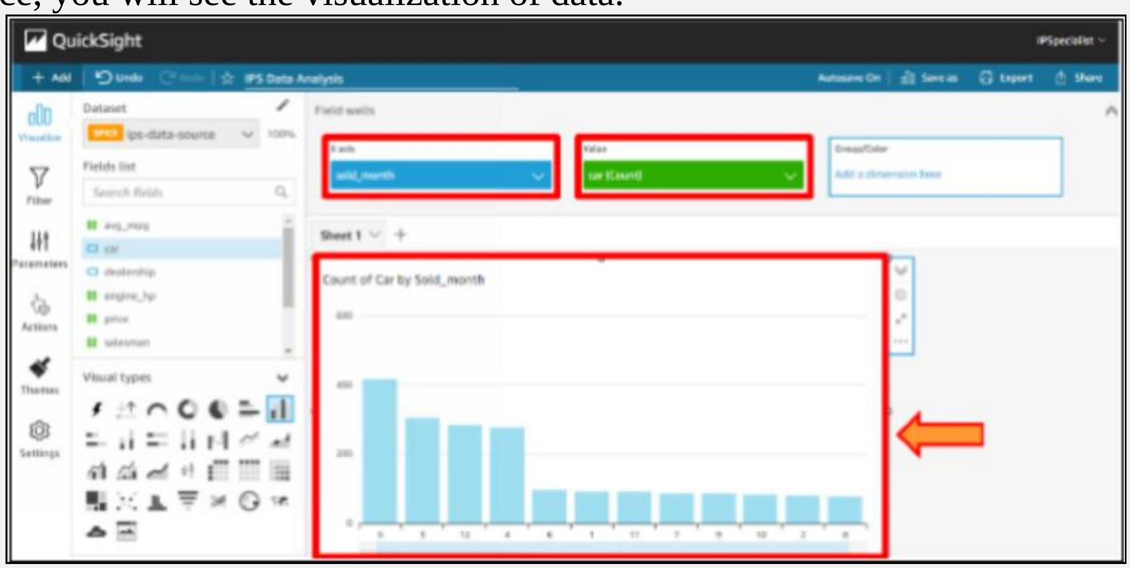
27. Drag and drop the **sold_month** feature onto the **X-Axis** box.



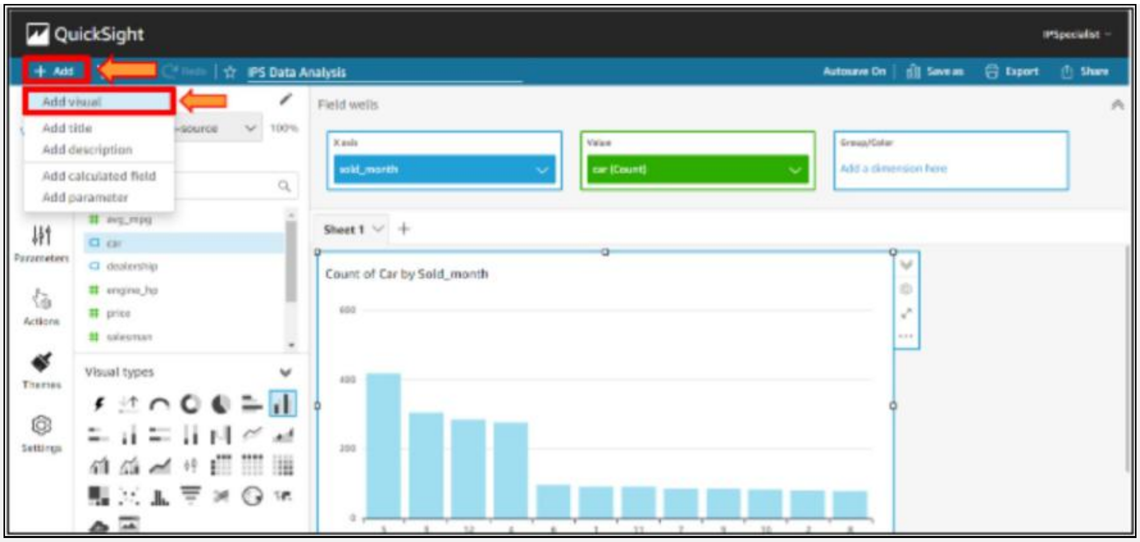
28. Drag and drop the **Car** feature onto the **Value** box.



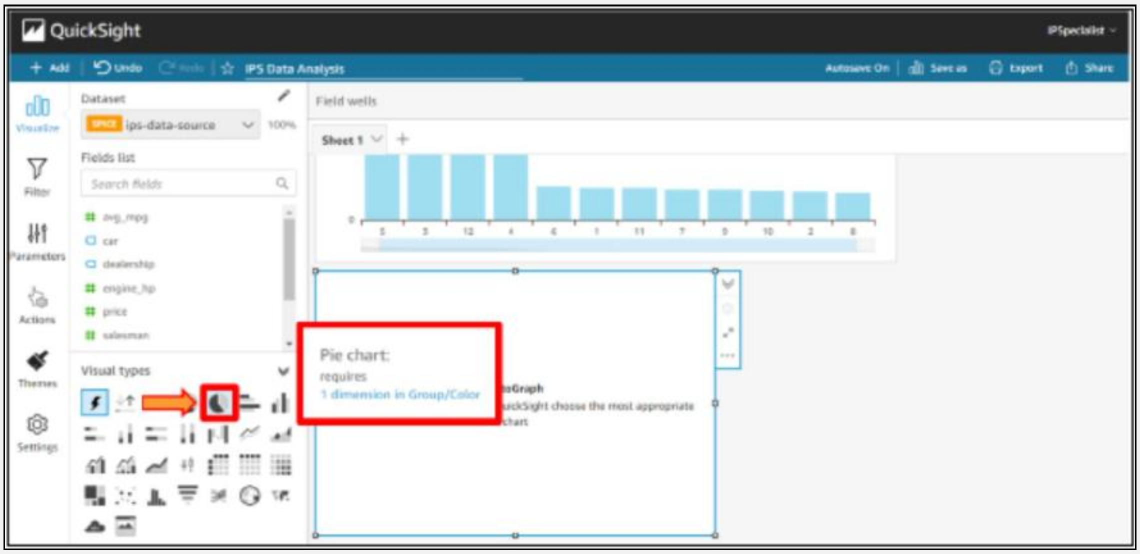
Hence, you will see the visualization of data.



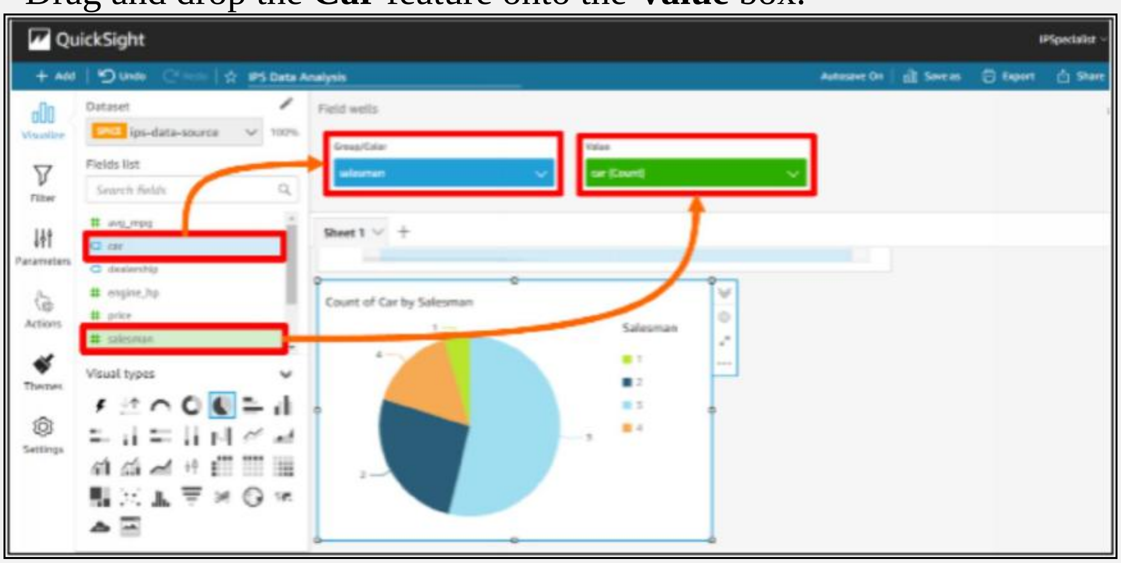
29. Click on the **Add** button. Then, click on the **Add Visual**.



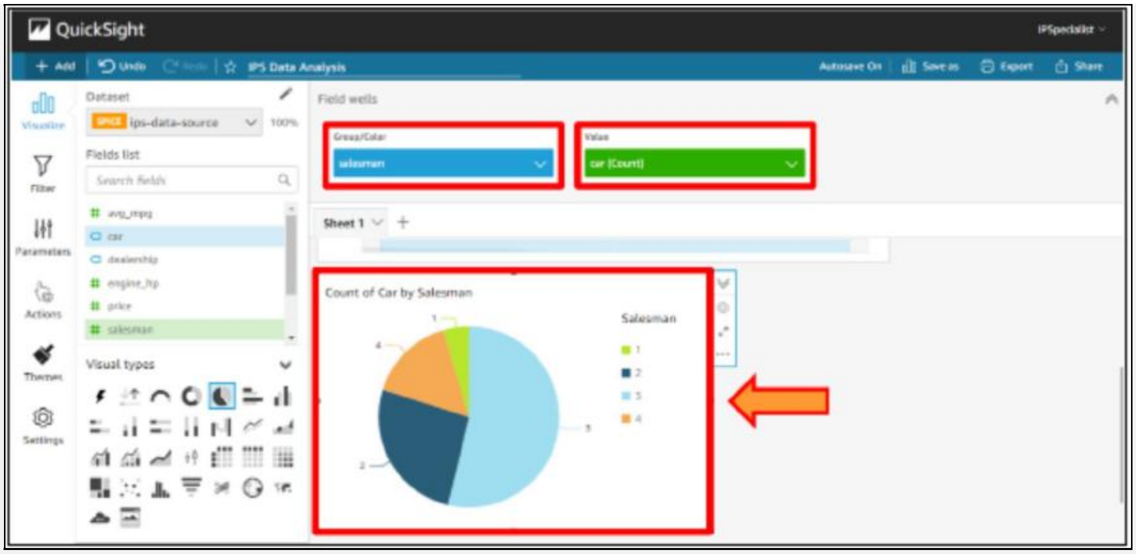
30. Click on the **Pie Chart.**



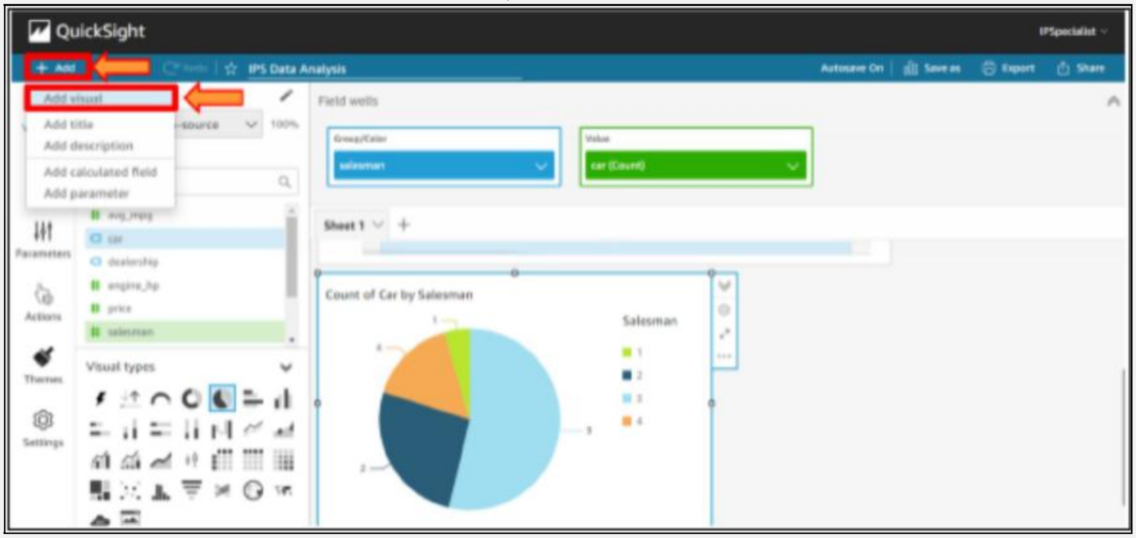
31. Drag and drop the **Salesman** feature onto the **Group/Colour** box.
32. Drag and drop the **Car** feature onto the **Value** box.



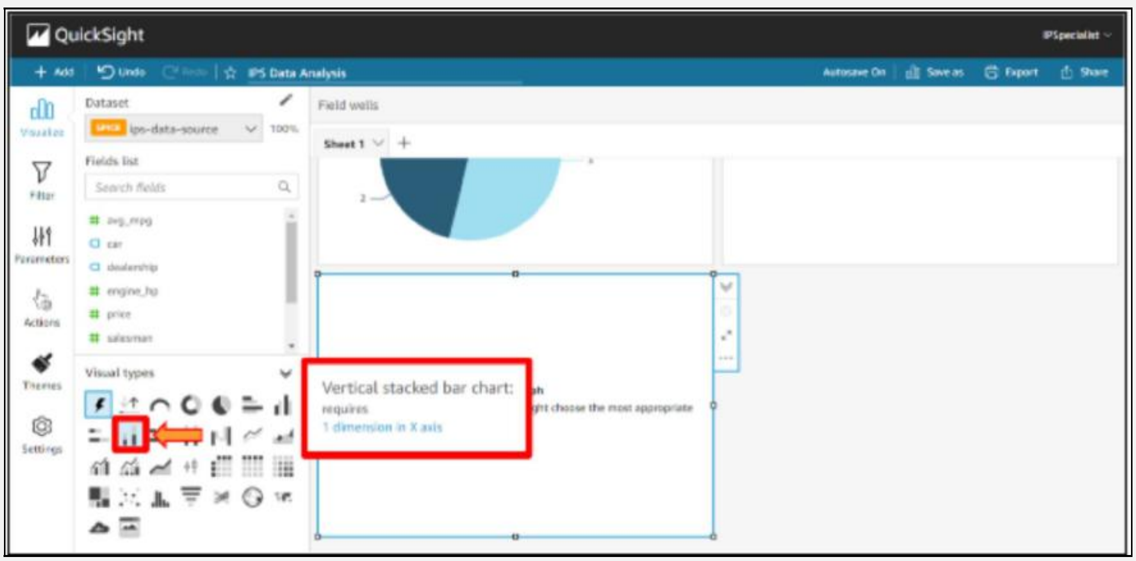
Hence, you will see the visualization of the Pie Chart.



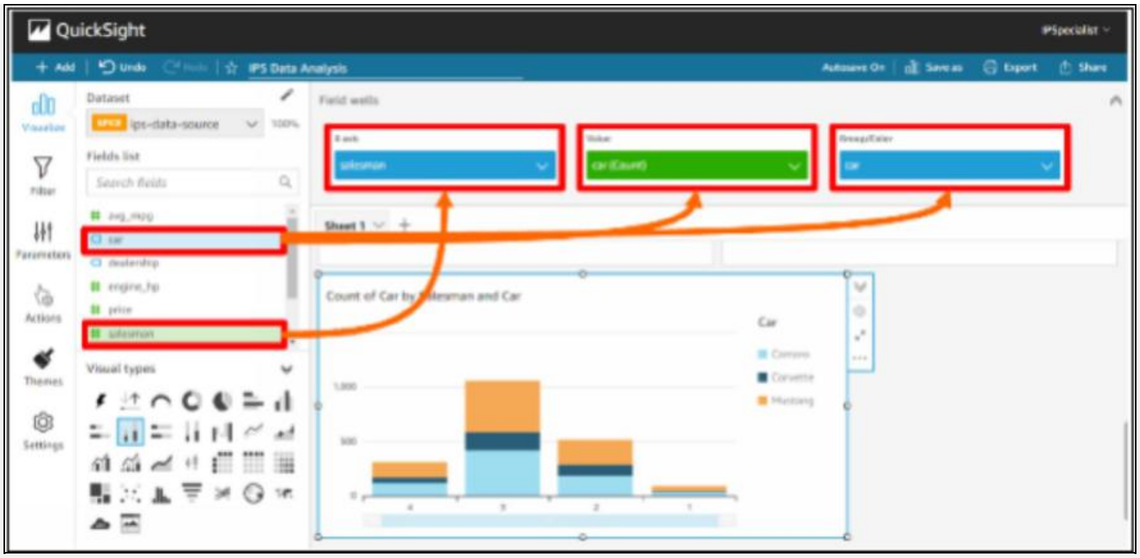
33. Click on the **Add** button. Then, click on the **Add Visual.**



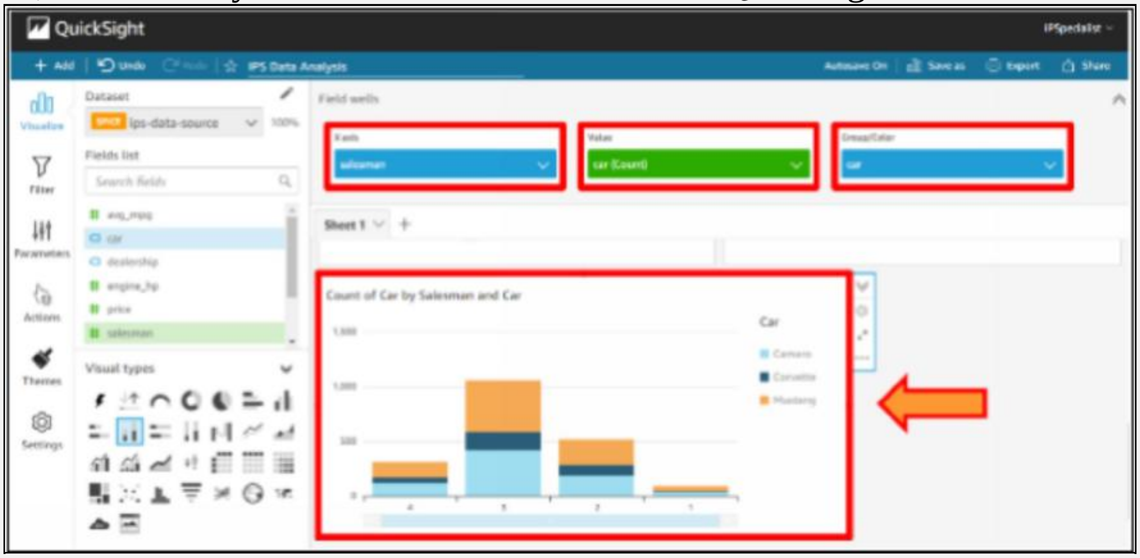
34. Click on the **Vertical bar chart**.



35. Drag and drop the **Salesman** feature onto the **X-Axis** box.
36. Drag and drop the **Car** feature onto the **Value** box.
37. Drag and drop the **Car** feature onto the **Group/Colour** box.



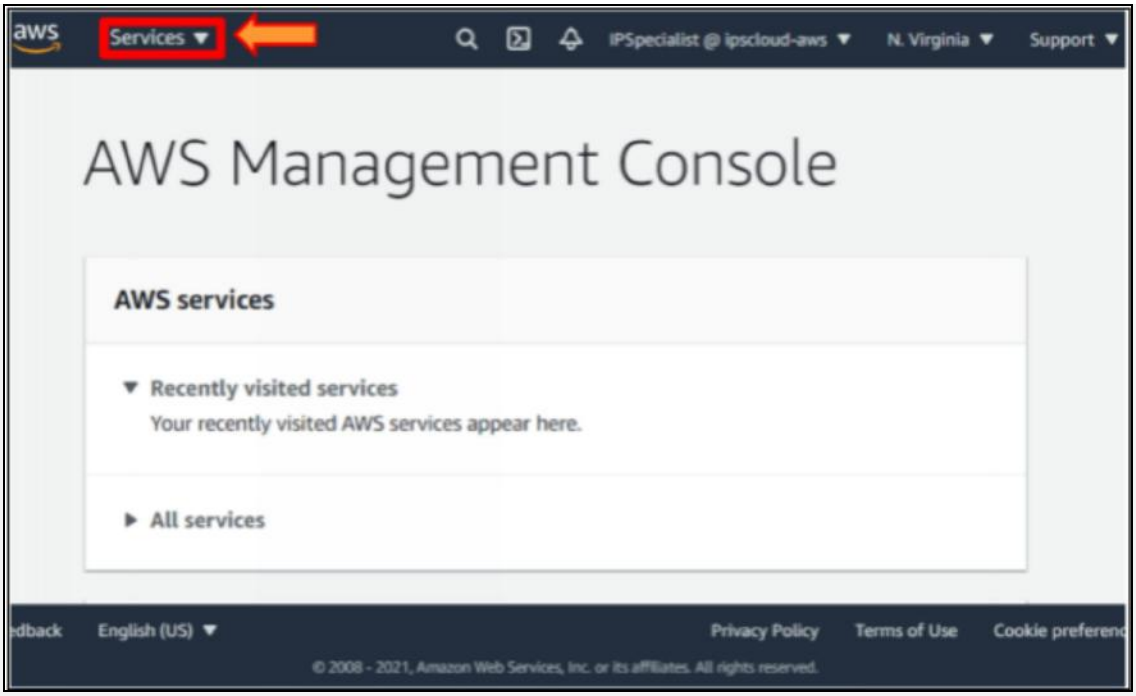
Hence, this is how you can visualize data on AWS QuickSight.



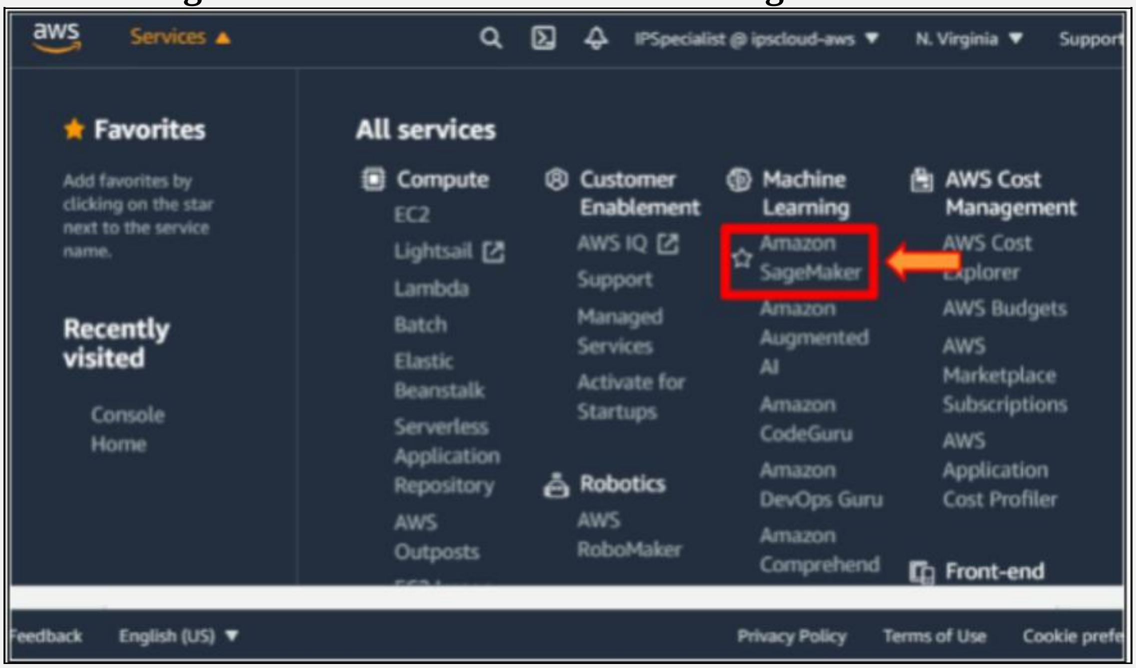
38. Unsubscribe your AWS QuickSight account so that you do not get charged.

Step 3: Visualize Data on AWS SageMaker

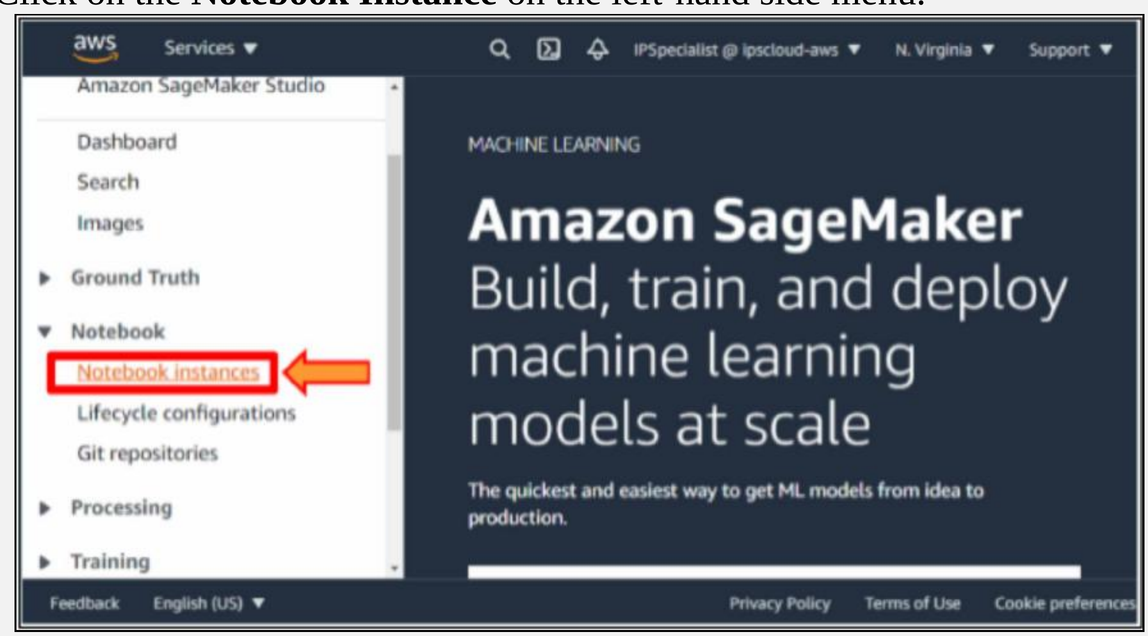
1. Click on **Services.**



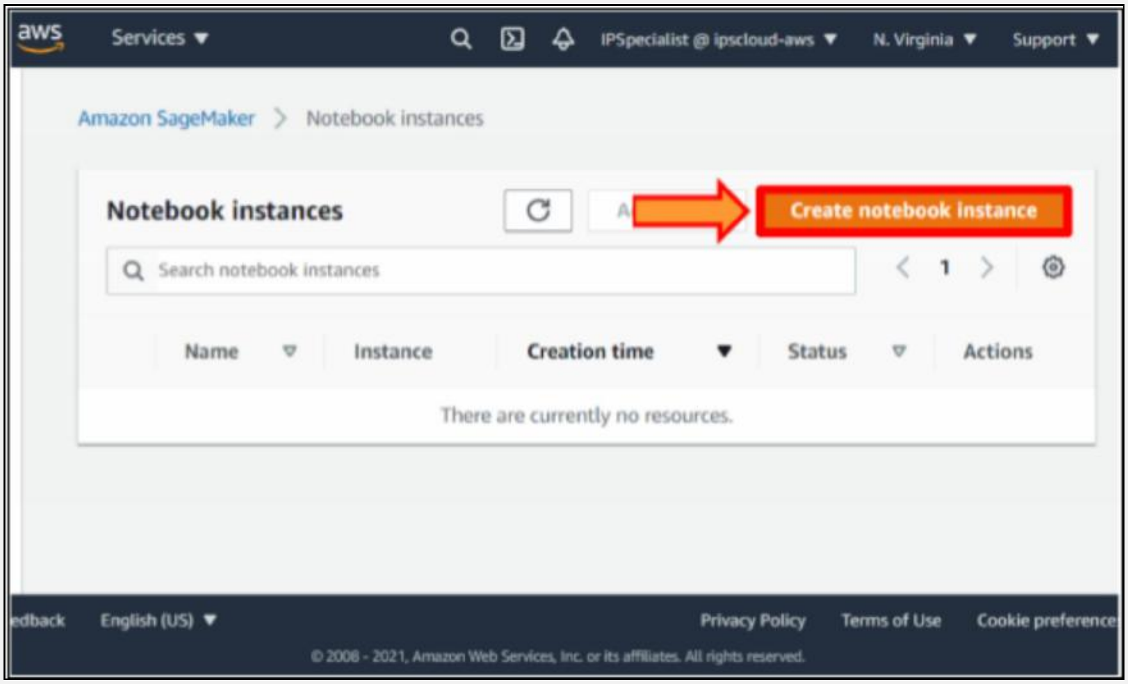
2. Select the **SageMaker** from the **Machine Learning.**



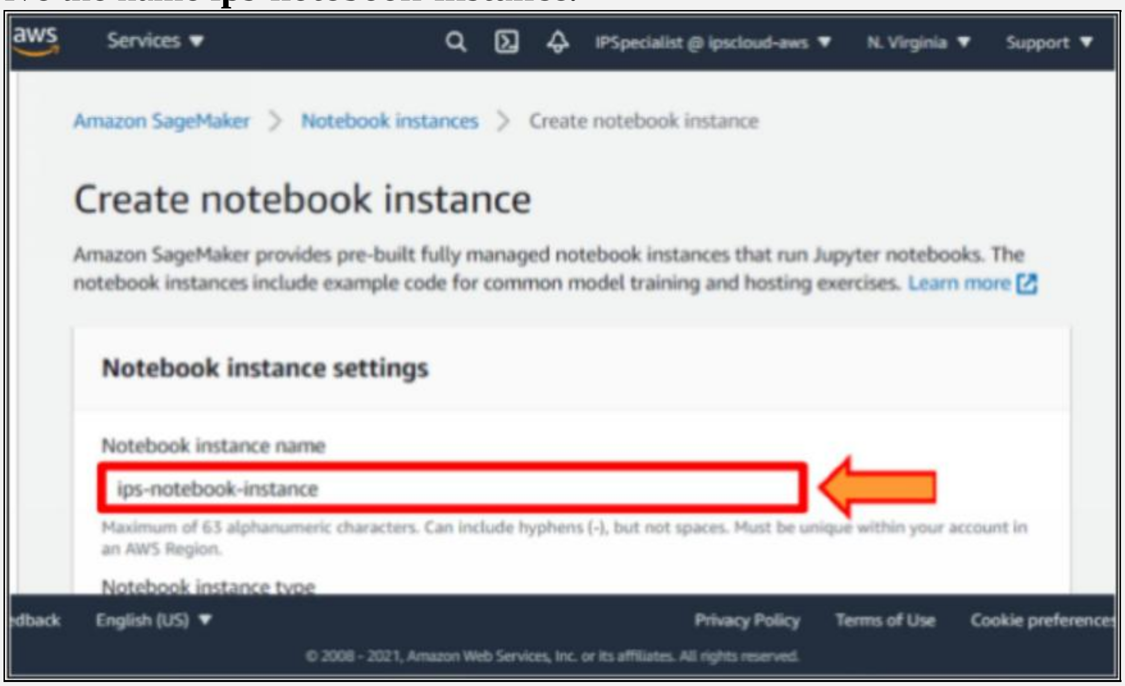
3. Click on the **Notebook Instance** on the left-hand side menu.



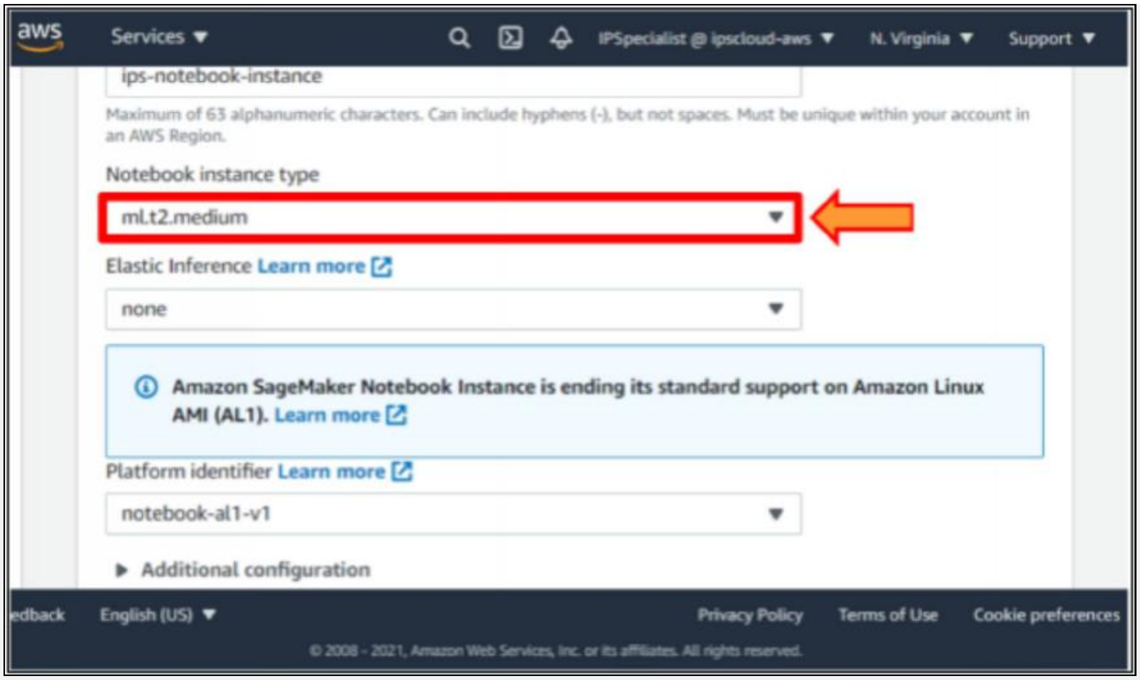
4. Click on the **Create notebook instance** button.



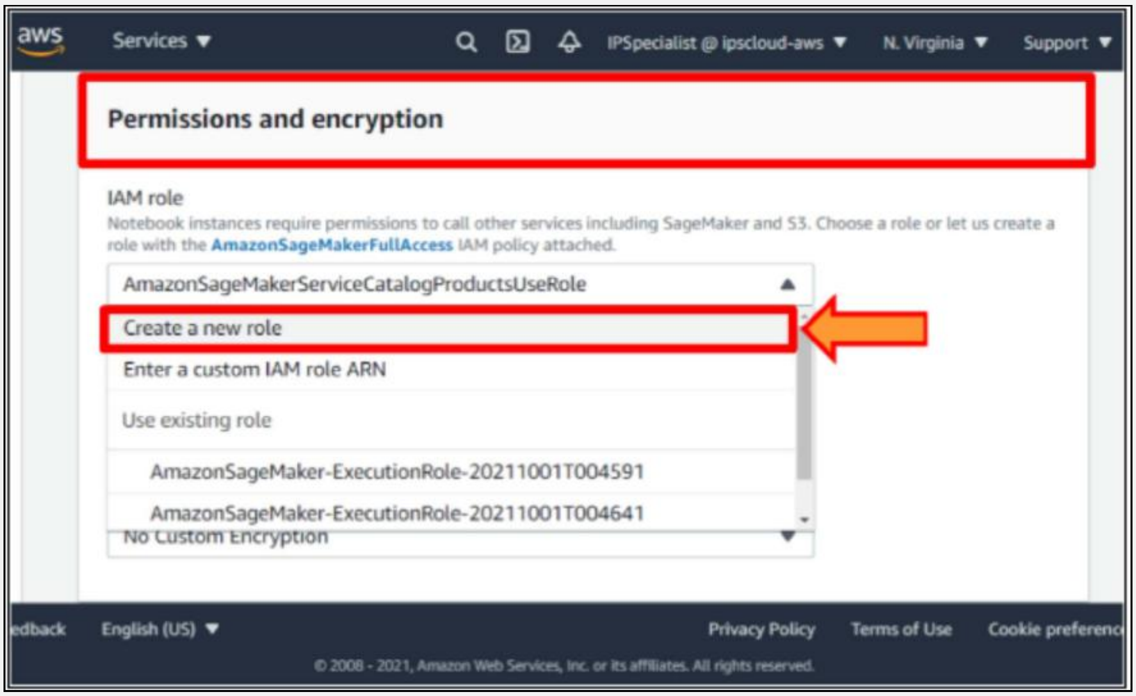
5. Give the name **ips-notebook-instance.**



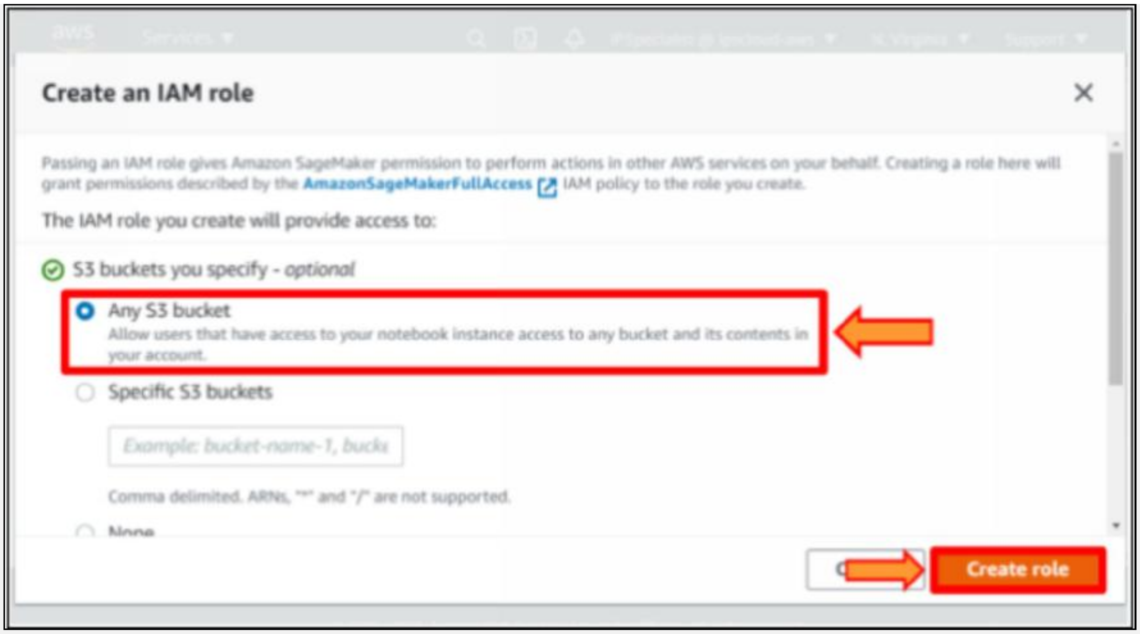
6. Select the **ml.t2.medium** Notebook instance type.



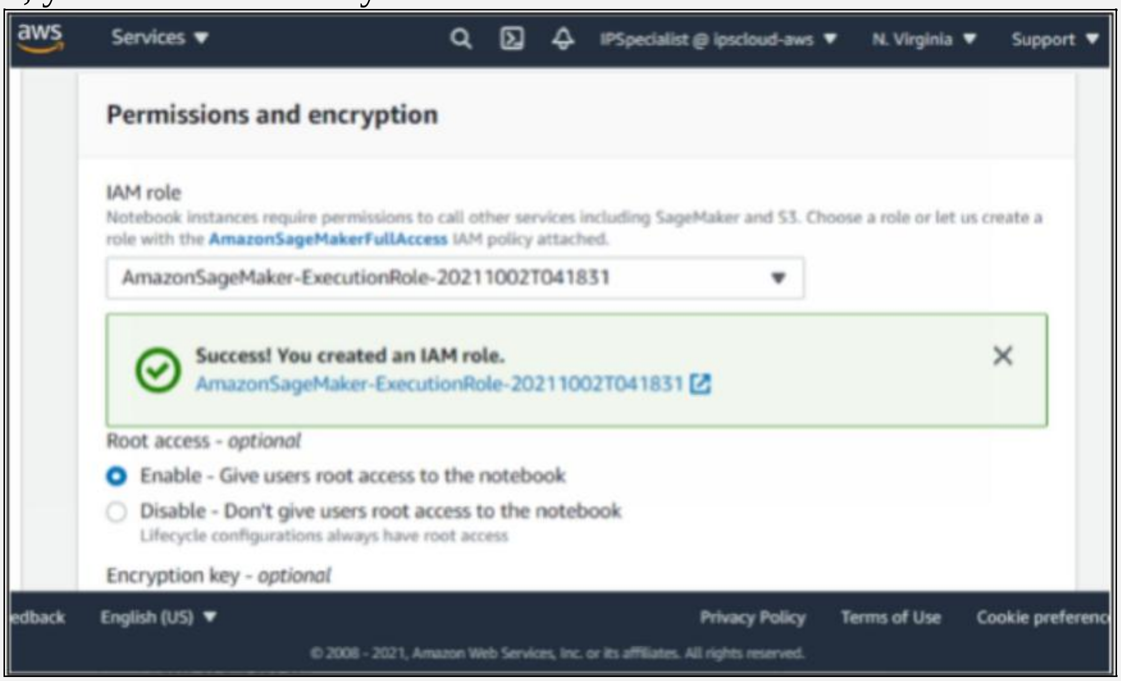
7. Scroll down under permission and encryption.
8. Click on the **Create a new role.**



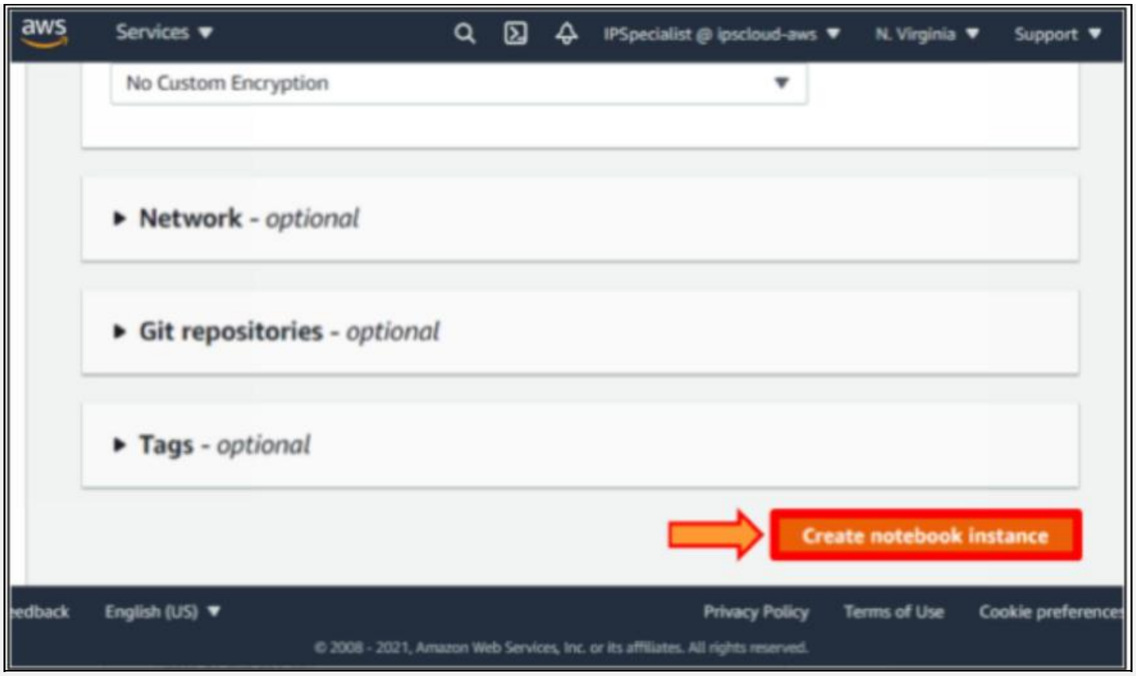
9. Select **Any S3 Bucket.**
10. Click on the **Create role** button.



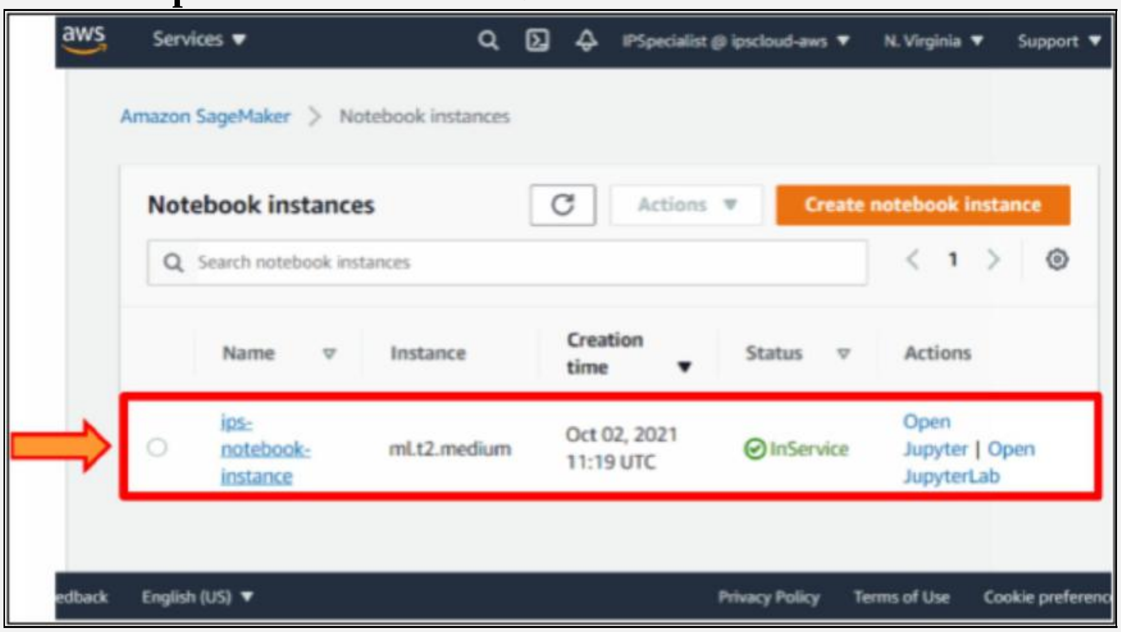
Hence, you have successfully created an IAM role.



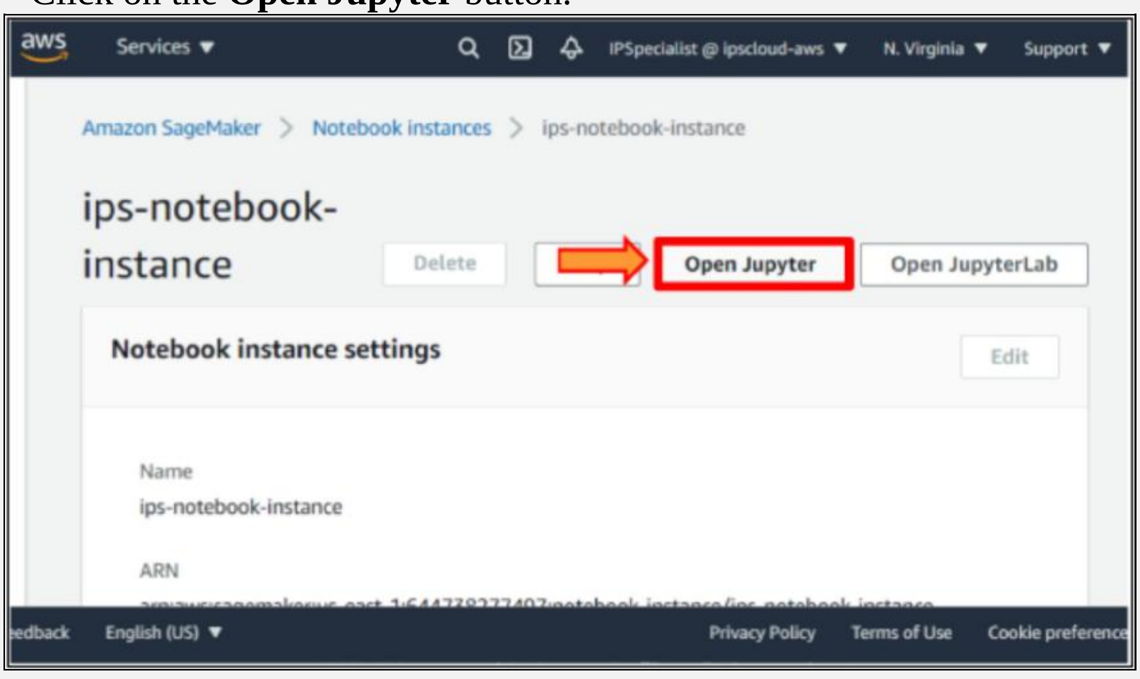
11. Scroll down. Click on the **Create notebook instance** button. It will take a few minutes.



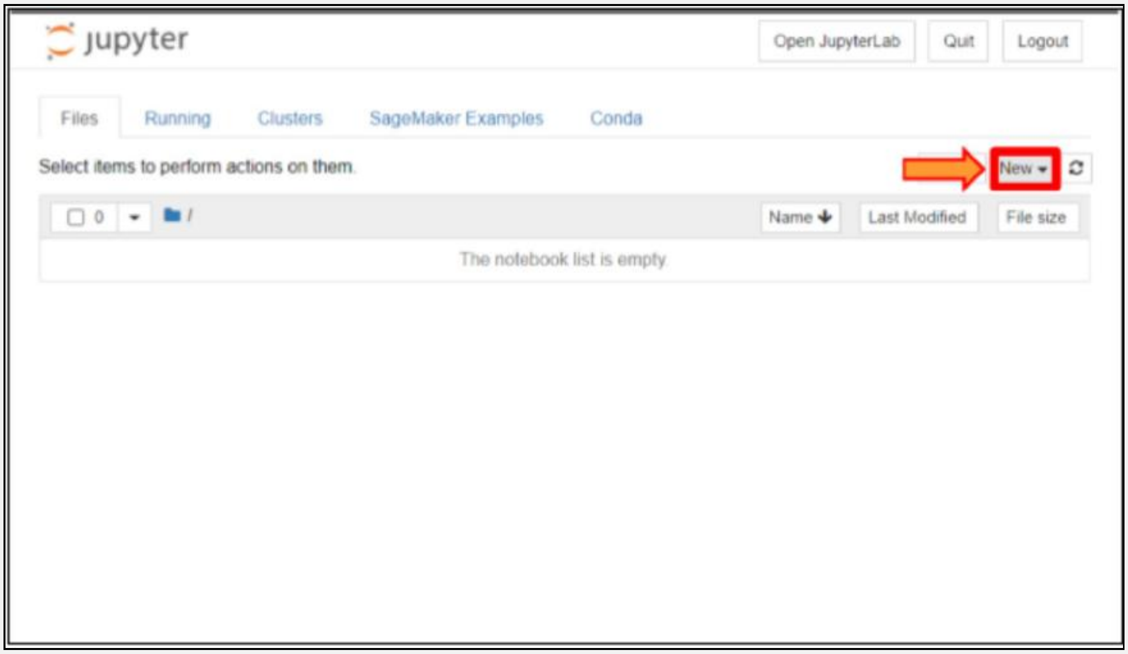
12. Click on **ips-notebook-instance.**



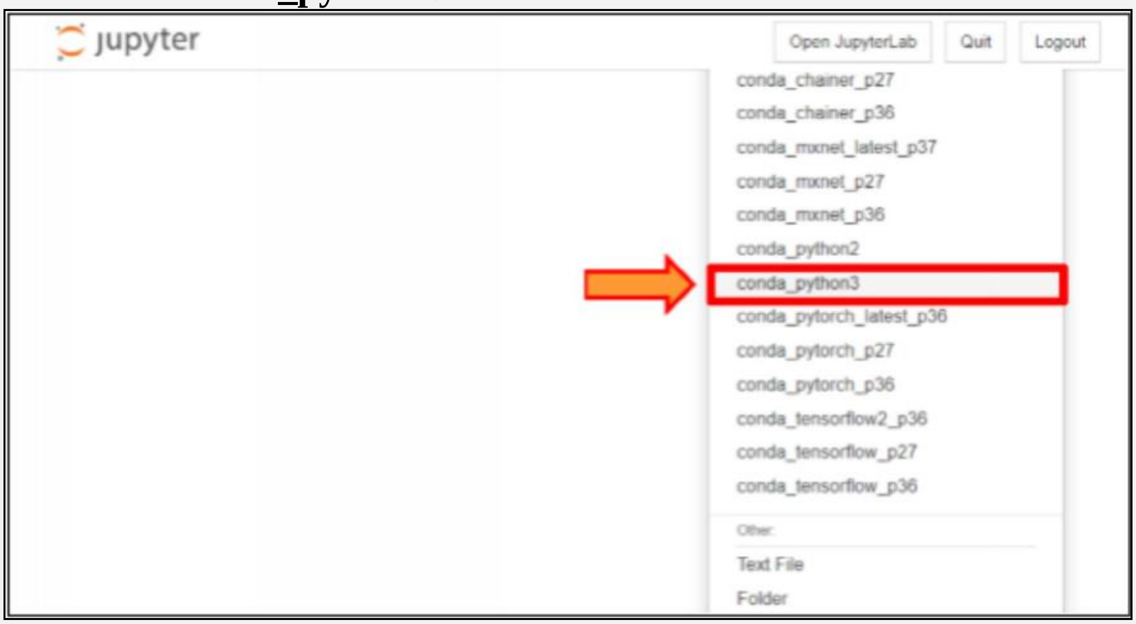
13. Click on the **Open Jupyter** button.



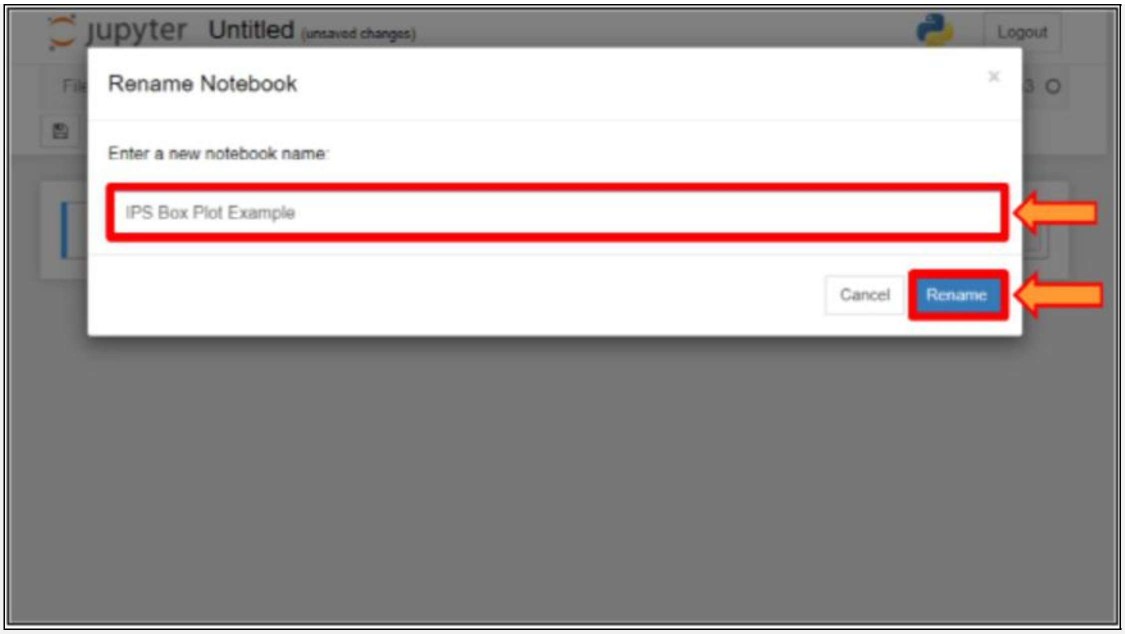
14. Click on **New.**



15. Click on **Conda_python3.**

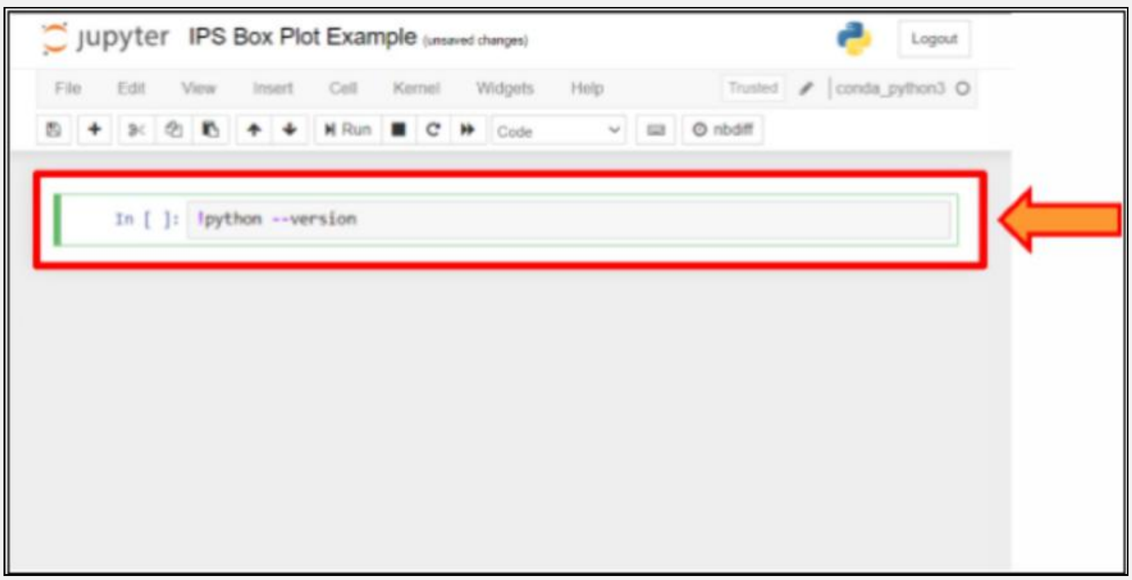


16. Give the Jupyter notebook name **IPS Box Plot Example**. Then, click on the **Rename** button.

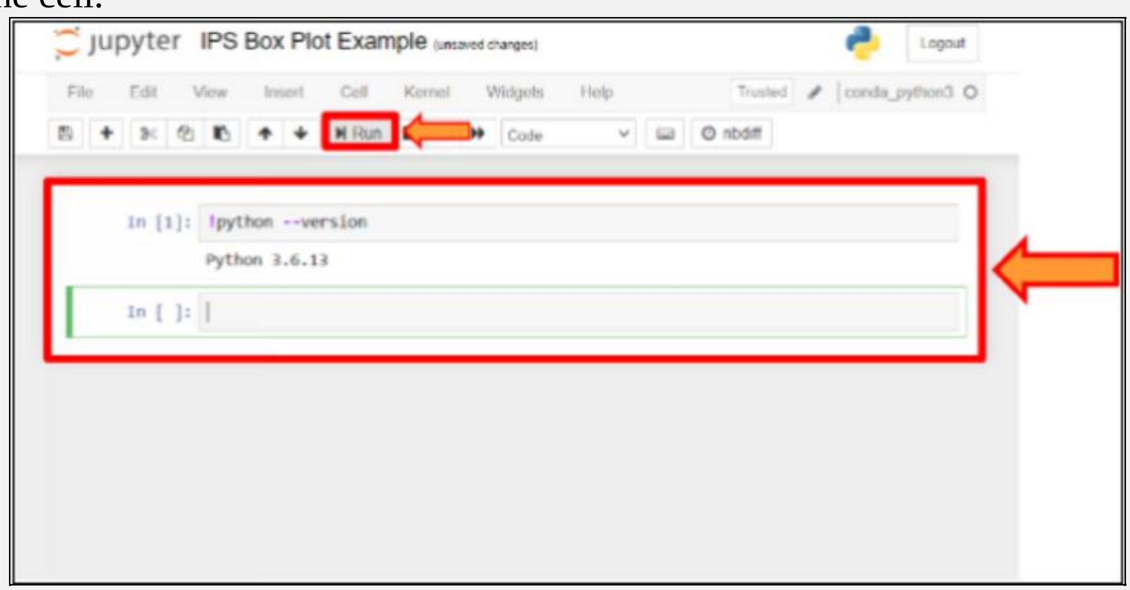


17. Copy and paste the below code on the cell to check the Python version.

```
! Python --version
```

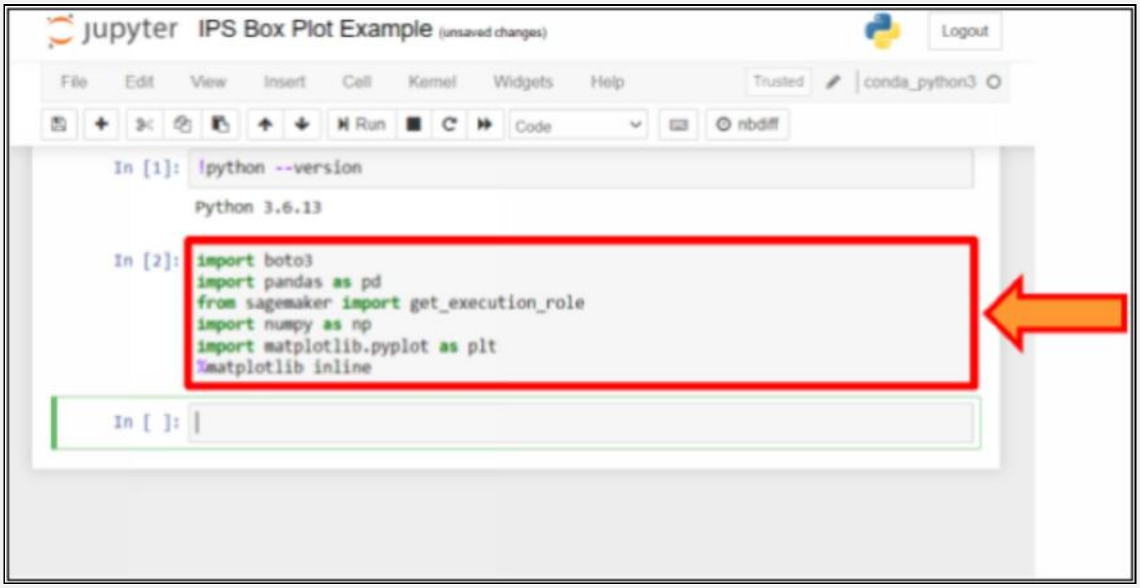


18. Click on the **Run** button to execute the cell or press **Shift + Enter** to run the cell.



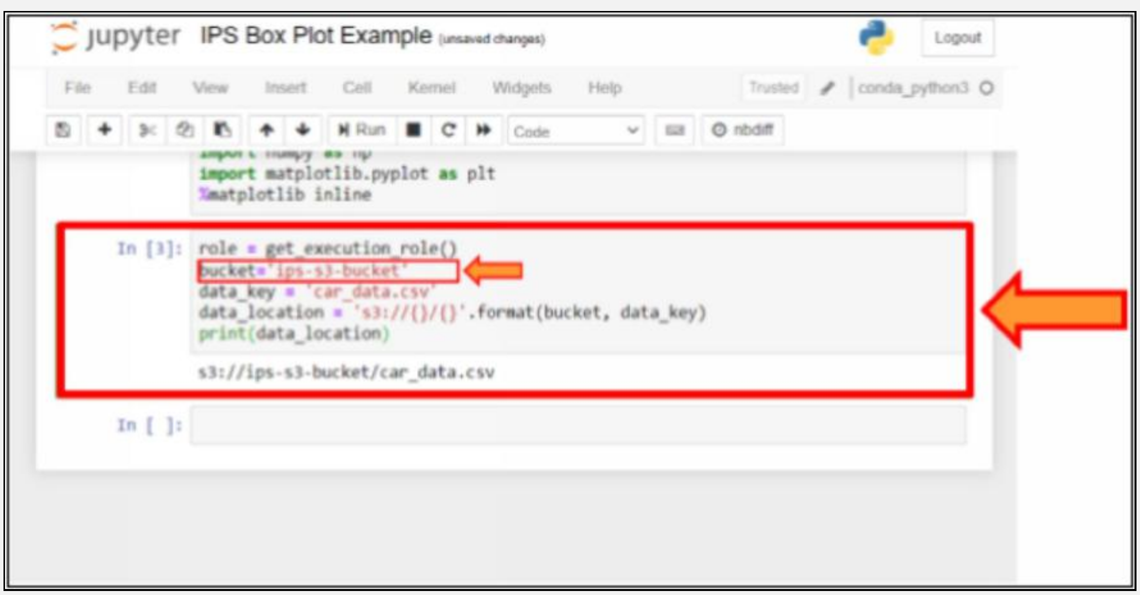
19. Copy and paste the below code to **Import the python libraries**. Then, press **Shift + Enter** to run the cell.

```
import boto3
import pandas as pd
from sagemaker import get_execution_role
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
```



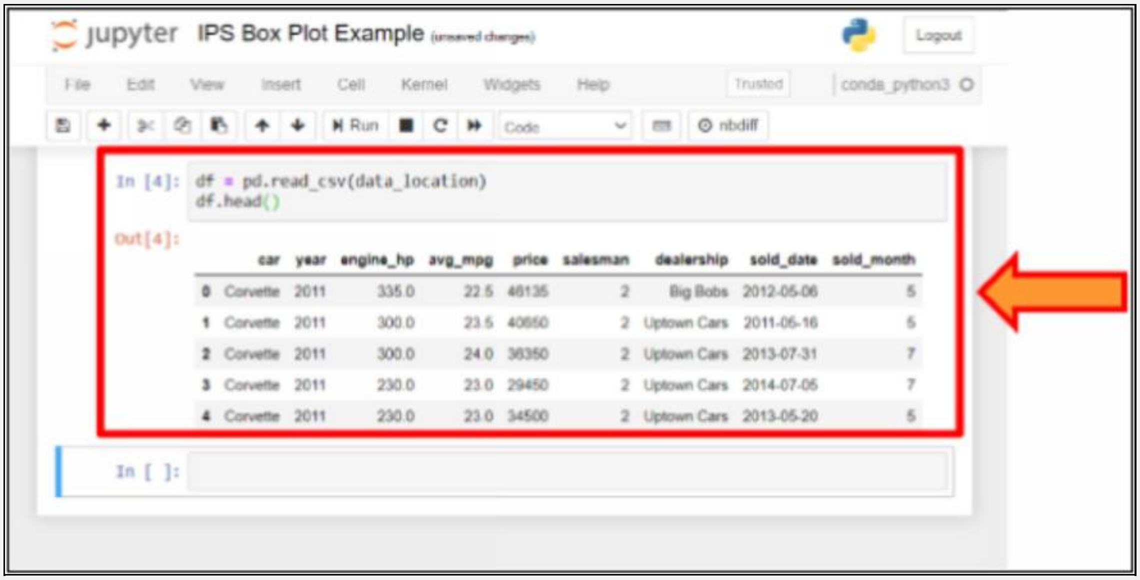
20. Copy and paste the below code to **Get the car data from S3**. Also, add the **ips-s3-bucket name**. Then, press **Shift + Enter** to run the cell.

```
role = get_execution_role()
bucket='<YOUR_BUCKET_NAME_HERE>'
data_key = 'car_data.csv'
data_location = 's3://{}/{}'.format(bucket, data_key)
print(data_location)
```



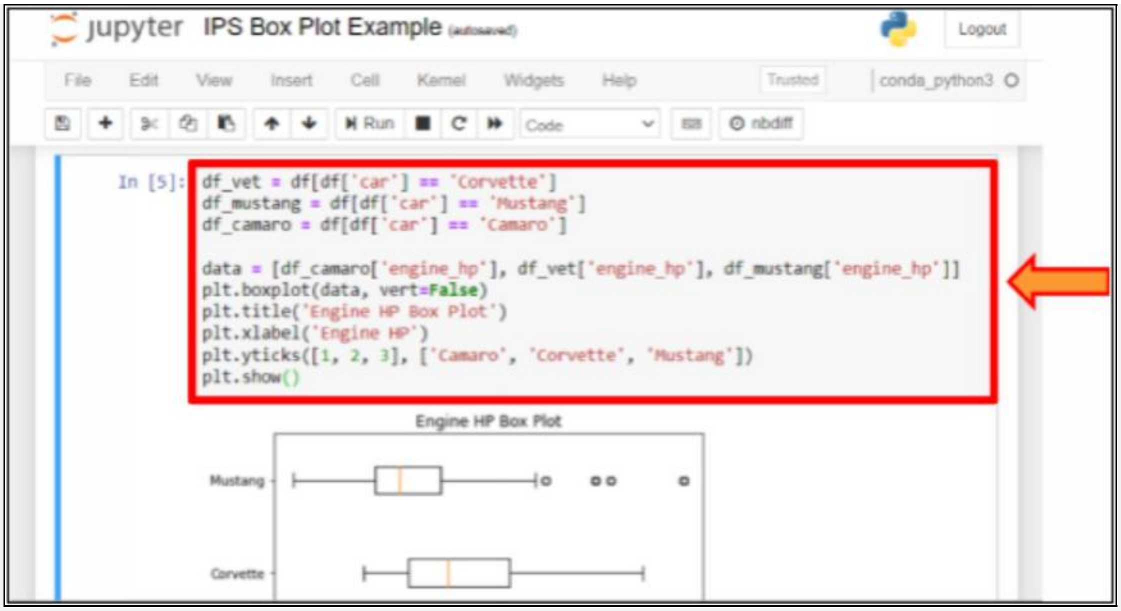
21. Copy and paste the code below to **read a CSV file into a data frame and see the first five rows of the data frame**. Then, press **Shift + Enter** to run the cell.

```
df = pd.read_csv(data_location)
df.head()
```

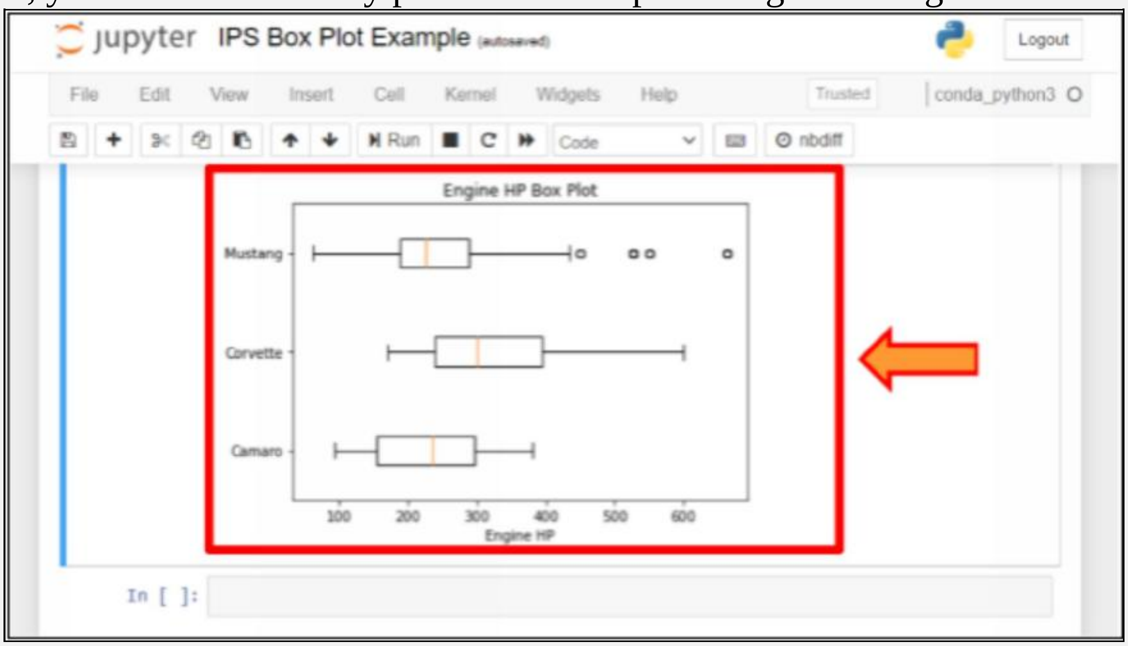


22. Copy and paste the below code to the **plot box plot**. Then, press **Shift + Enter** to run the cell.

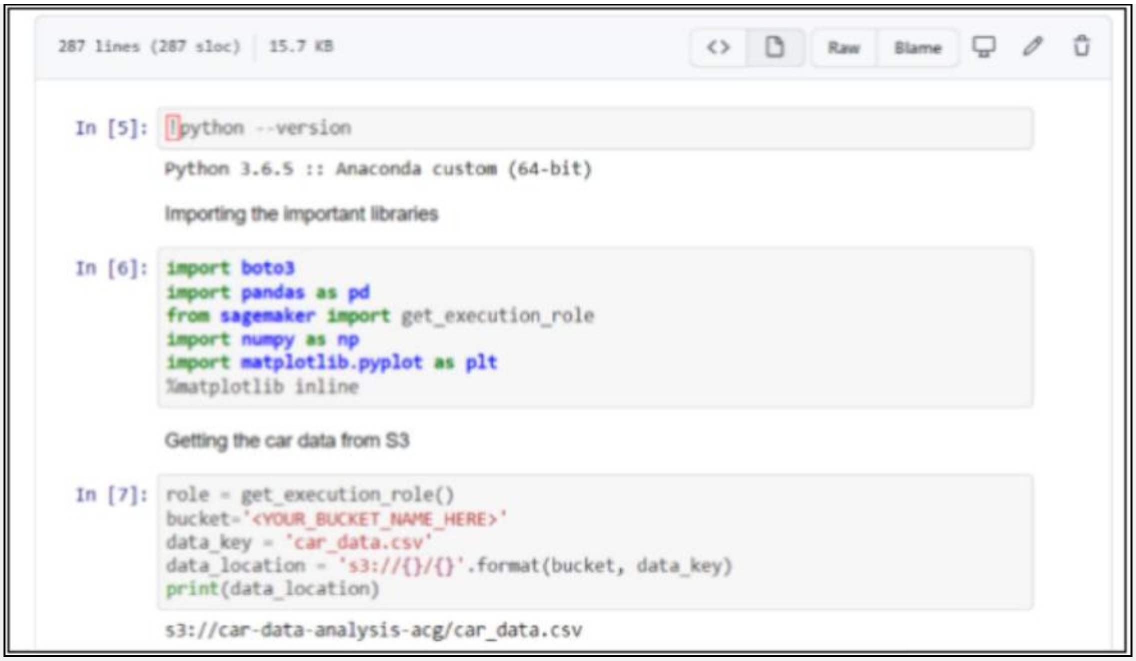
```
df_vet = df[df['car'] == 'Corvette']
df_mustang = df[df['car'] == 'Mustang']
df_camaro = df[df['car'] == 'Camaro']

data = [df_camaro['engine_hp'], df_vet['engine_hp'], df_mustang['engine_hp']]
plt.boxplot(data, vert=False)
plt.title('Engine HP Box Plot')
plt.xlabel('Engine HP')
plt.yticks([1, 2, 3], ['Camaro', 'Corvette', 'Mustang'])
plt.show()
```



Hence, you have successfully plotted the box plot using AWS SageMaker.



23. The Jupyter Notebook is provided in the following Github link: https://github.com/ACloudGuru-Resources/Course_AWS_Certified_Machine_Learning/blob/master/Chapter5/box-plot-example.ipynb



So this is how you can visualize the data using AWS SageMaker.

24. After completing the lab, delete all the AWS services; Therefore, you do not get charged.

Lab 07: Modeling

Certifications	AWS Machine Learning - Specialty
Cloud Provider	AWS
Time Required	1 Hr.
Tags	QuickSight, S3, SageMaker, AWS

Lab Prerequisites

- Familiarity with basic AWS Cloud Computing concepts and terminology
- An AWS account with an active subscription

Service Introduction

AWS QuickSight

Amazon QuickSight is a cloud-based business analytics solution that enables all employees to create visualizations, do ad-hoc analysis, and quickly get business insights from their data, anytime and on any device. Upload CSV and Excel files; connect to SaaS apps like Salesforce; connect to on-premises databases like SQL Server, MySQL, and PostgreSQL; and locate your AWS data sources in real-time, including Amazon Redshift, Amazon RDS, Amazon Aurora, Amazon Athena, and Amazon S3. QuickSight allows businesses to extend their business analytics capabilities to thousands of users while still providing quick and responsive query processing. QuickSight enables businesses to expand their business analytics capabilities to thousands of users. It uses a powerful in-memory engine to offer quick and responsive query performance (SPICE).

AWS SageMaker

Amazon SageMaker is a fully managed service that allows any developer or data scientist to quickly create, train, and deploy Machine Learning (ML) models. SageMaker makes it easy to create high-quality models by removing the heavy lifting from each phase of the machine learning process.

AWS Simple Storage Service S3

Amazon S3 is a type of object storage that allows you to store and recover any quantity of data from any location. It is a low-cost storage solution with business resilience, reliability, efficiency, privacy, and infinite expansion.

Amazon S3 is a web service that allows you to store and retrieve an infinite quantity of data from any place and at any time. You may quickly create projects that integrate cloud-native storage using this service because Amazon S3 is easily customizable, and you only pay for what you use. You can start small and scale up as needed without sacrificing performance or dependability.

Amazon S3 is also built to be highly adaptable. Instead of finding out how to store their data, Amazon S3 allows developers to focus on innovation. Build a simple FTP system or a complex web application like the Amazon.com retail website; read the same piece of data a million times or only for emergency disaster recovery; store whatever type and amount of data you desire.

Case Study Enterprise EMRs – Savana

Background

Savana offers processing and analytic services for Electronic Medical Records (EMRs). With more than 180 hospitals spread over 15 nations, Savana, a Madrid-based company, runs one of the world's largest Artificial Intelligence (AI), multicentric, and multilingual research networks.

Savana, a company, established in Madrid, assists healthcare providers in maximizing the value of their EMRs for research. To produce pertinent results for healthcare and life science providers looking into disease prediction and therapy, it blends research-grade methods with Natural Language Processing (NLP) and predictive analytics. It can process the massive amounts of data needed to run machine learning algorithms using Amazon Web Services (AWS). Along with meeting regional regulatory requirements, it can scale its infrastructure to support its rapid growth across international markets, including the US, Latin America, and Western Europe. Compared to the prior technology, it has shortened study processing times by 25% and IT expenditures by up to 90%.

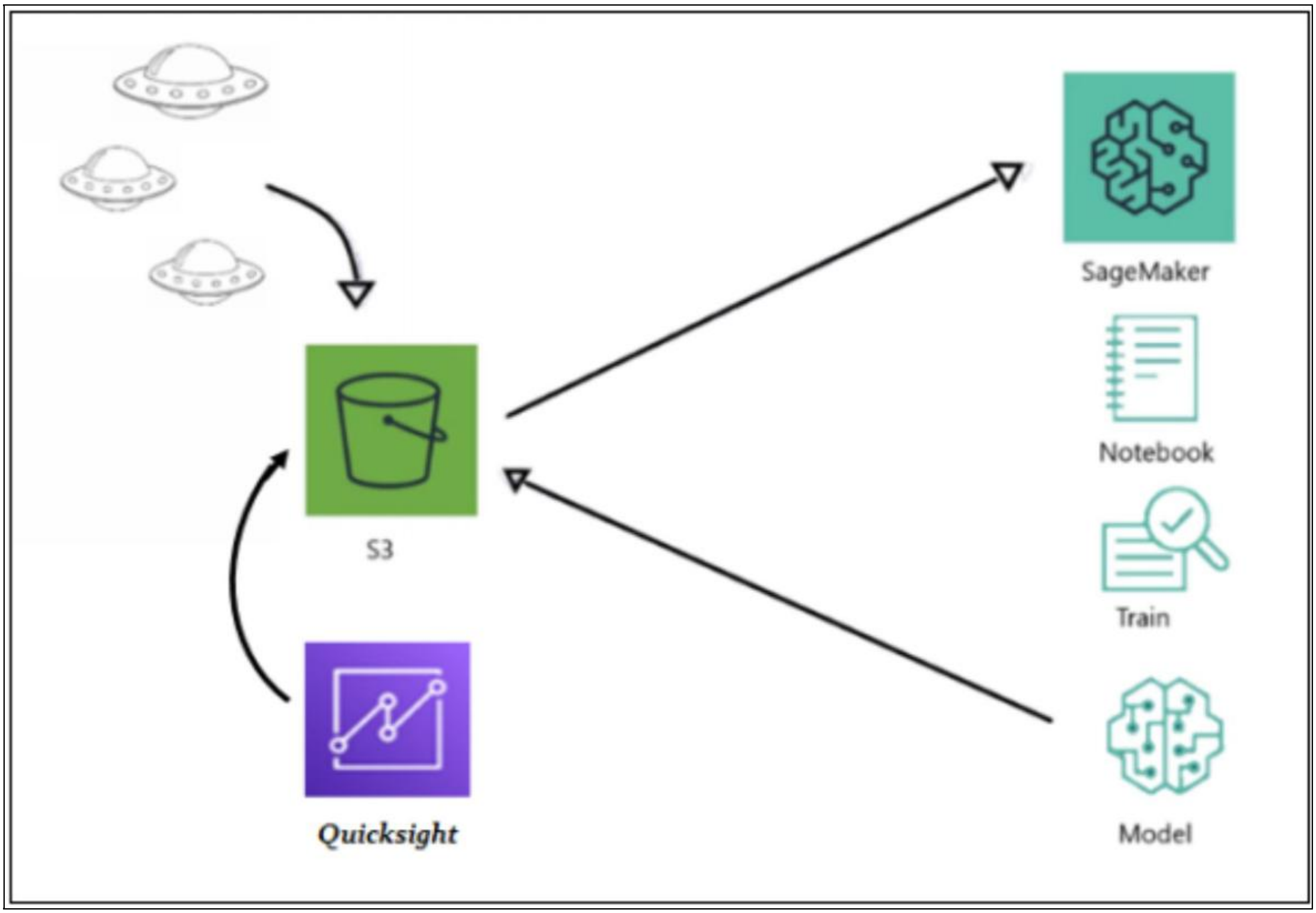
Business Challenge

Through an automated data pipeline fed by hospitals looking into disease prediction and therapy, Savana processes billions of EMRs and data points. To properly complete this, it is required to have a reliable architecture, handle the volume of data, and have the computing power required to run sophisticated algorithms while maintaining the greatest levels of security. The company's infrastructure must scale as it expands to meet regulatory requirements for data management, storage, and transmission worldwide.

Proposed Solution

The solution is for you to use AWS services to automate all the tasks. You create an AWS SageMaker hyperparameter tuning job with different ranges of values for the hyperparameter to find the best configuration, which minimizes the validation: objective_loss metric. The reason that you use this metric is that it is used in multi-classification problems. This metric measures the performance of the classification model, and what it does is repeatably calculate the difference between the values that the model predicts and the actual values of a label. Hence, every time it passes over data and makes predictions, it recalculates the objective loss and tries to minimize this value overall. Hence, that is the task here. AWS recommends that you minimize this value when using it as our objective metric.

Lab Diagram



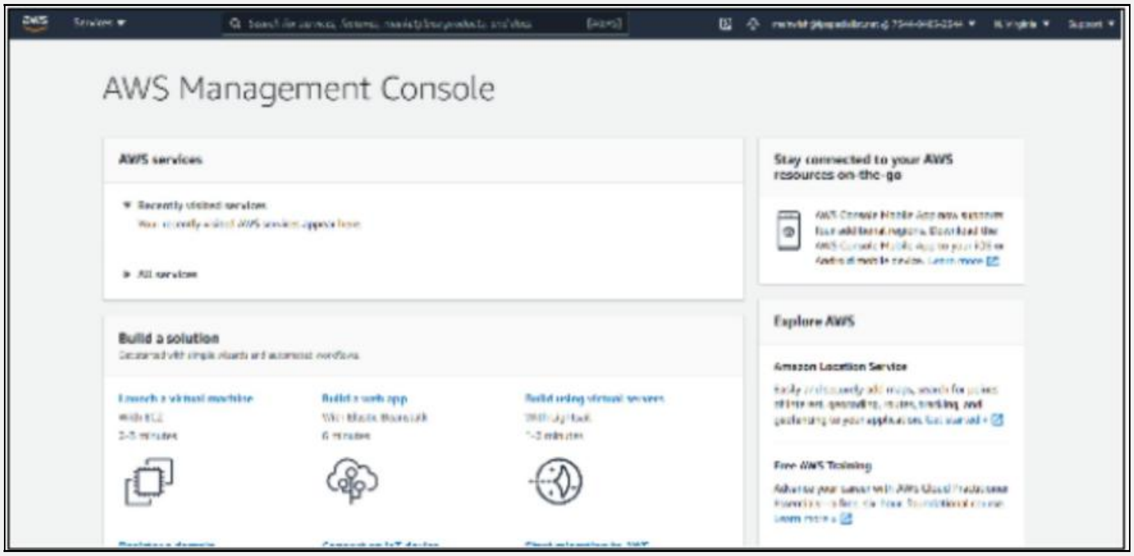
Implementation Steps

1. Create an S3 bucket.
2. Create Notebook instances.
3. Create a QuickSight account.

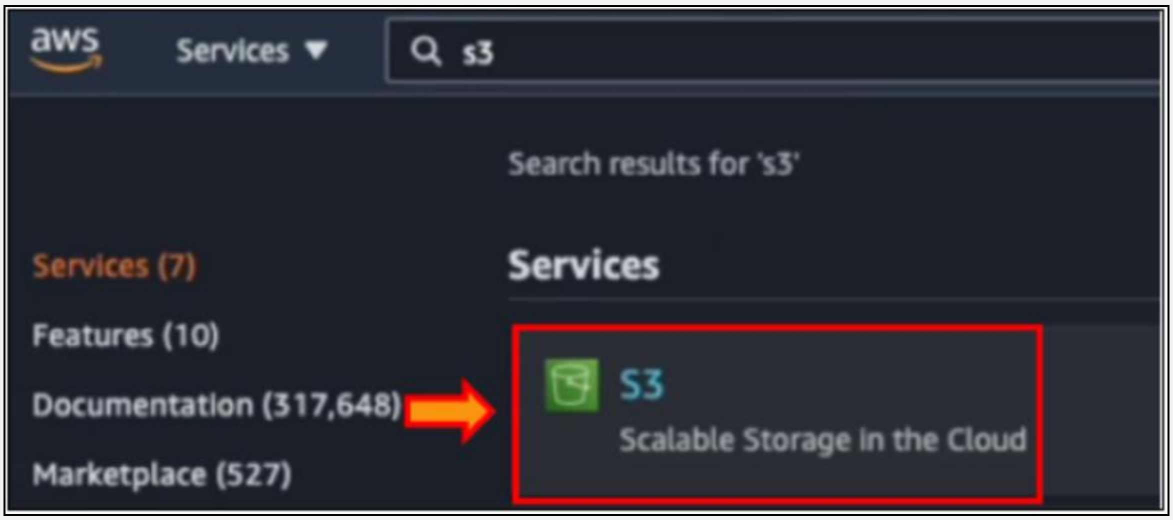
Solution

Step 1: Create an S3 Bucket

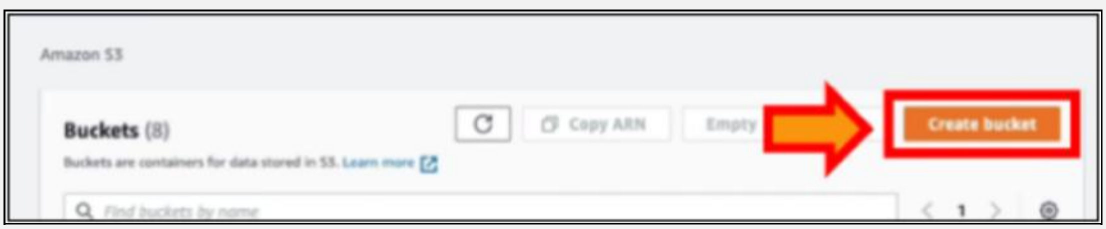
1. Log in to **AWS Management Console**.



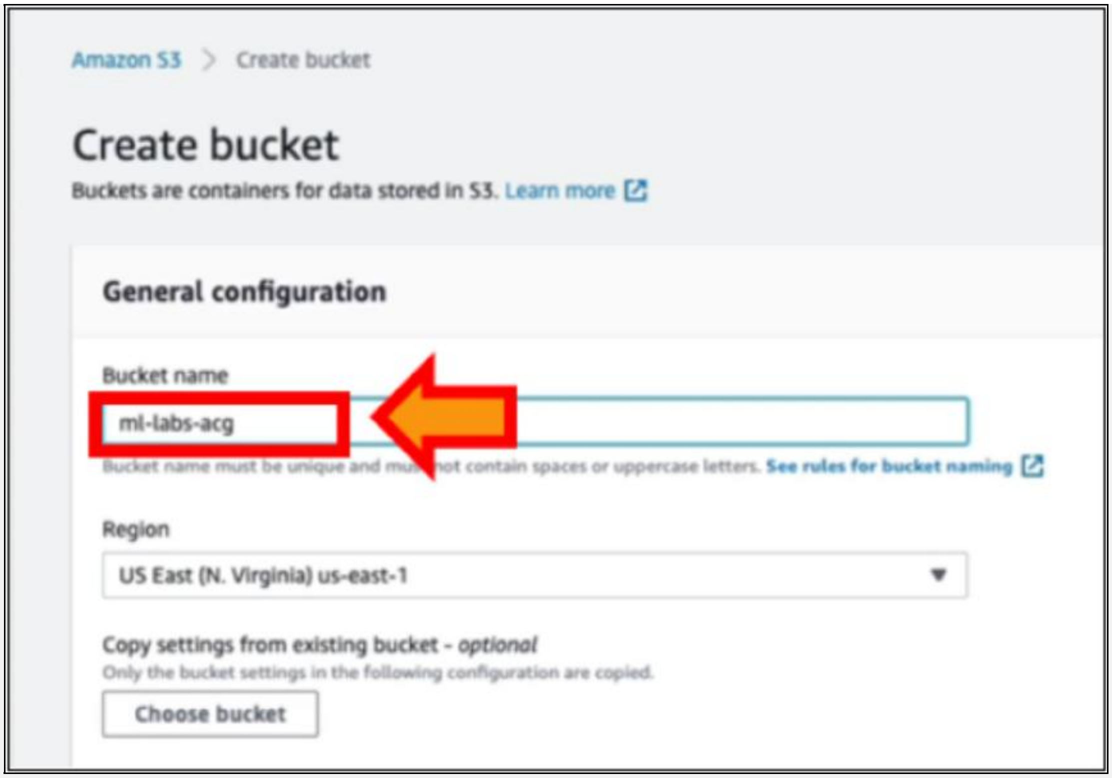
2. Search for **S3** in the search bar.



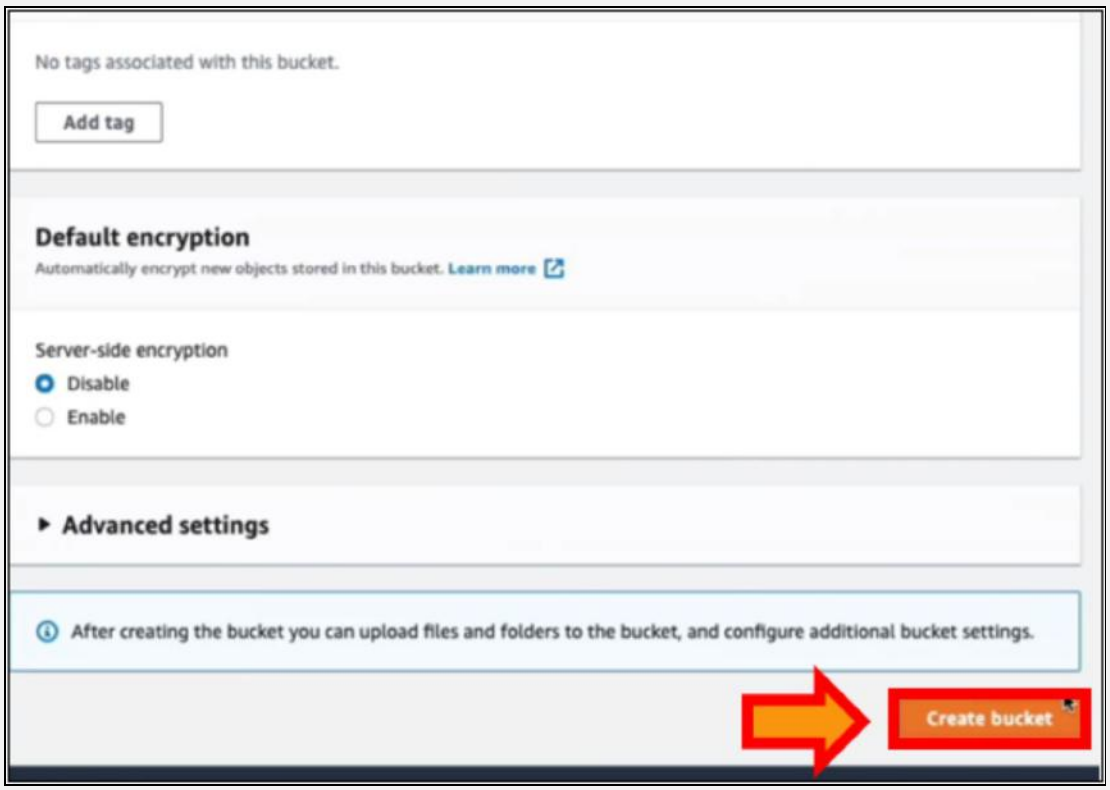
3. Click on **Create bucket**.



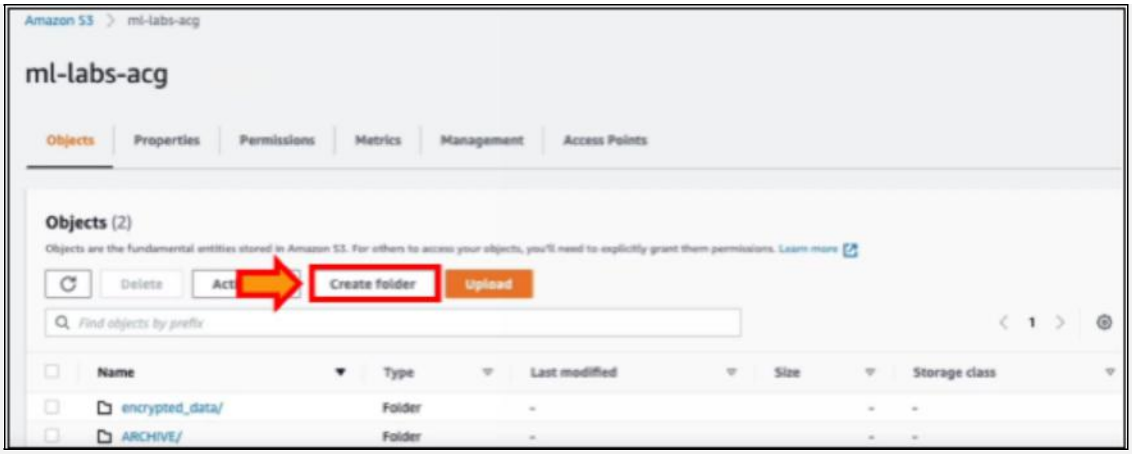
4. Define bucket name.



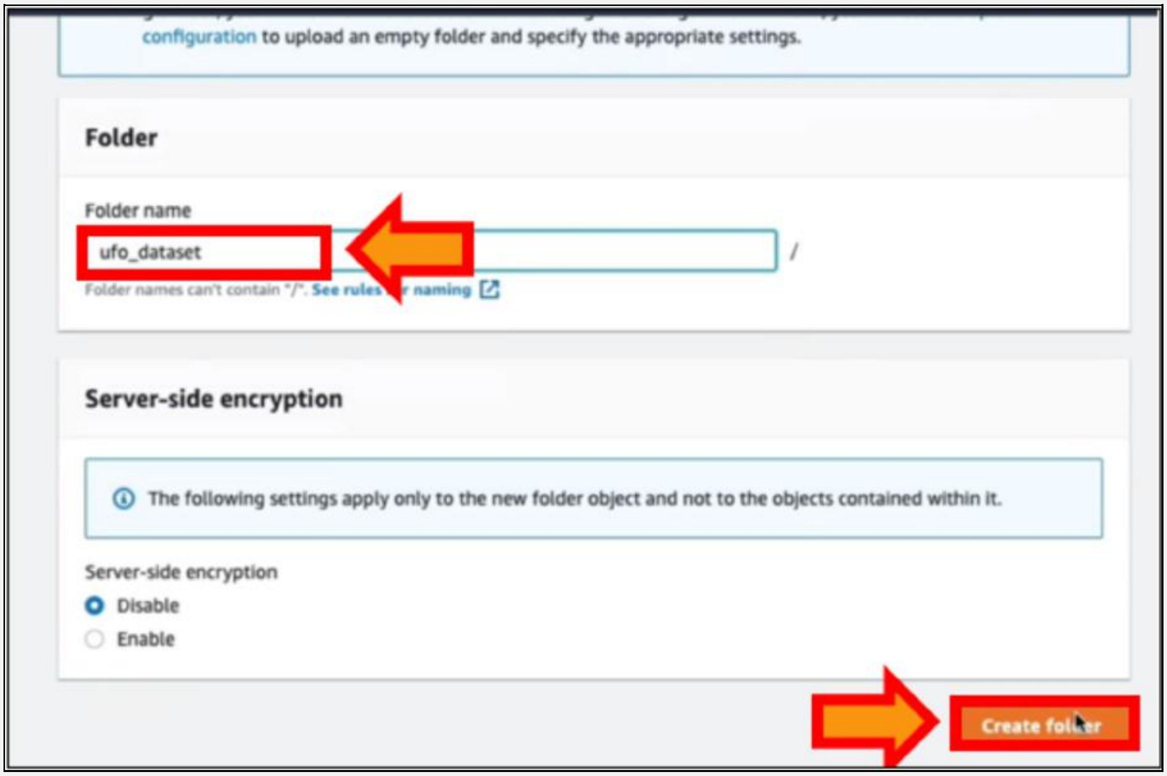
5. Click on **Create bucket**.



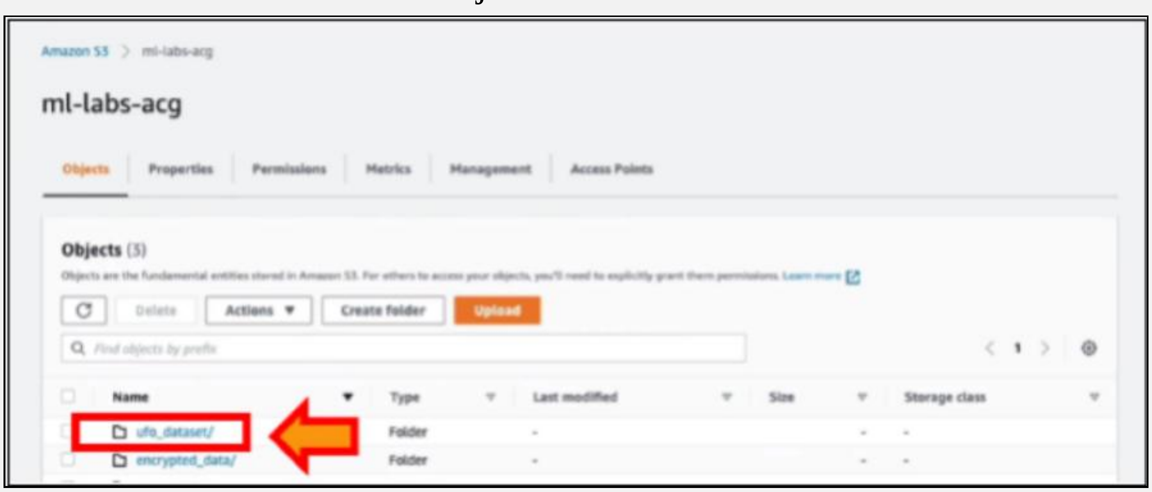
6. Click on **Create folder**.



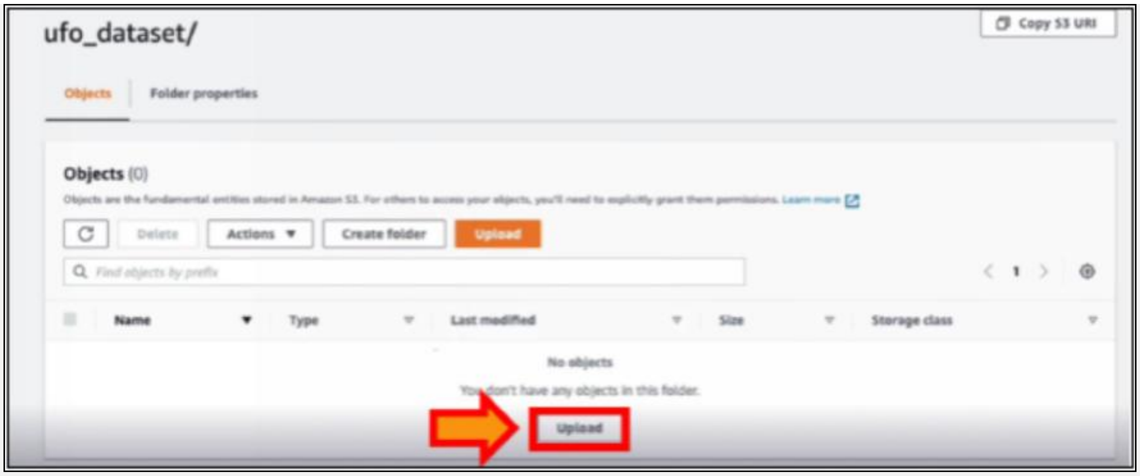
7. Define folder name.
8. Click on **Create folder.**



9. Click on the name of the object.



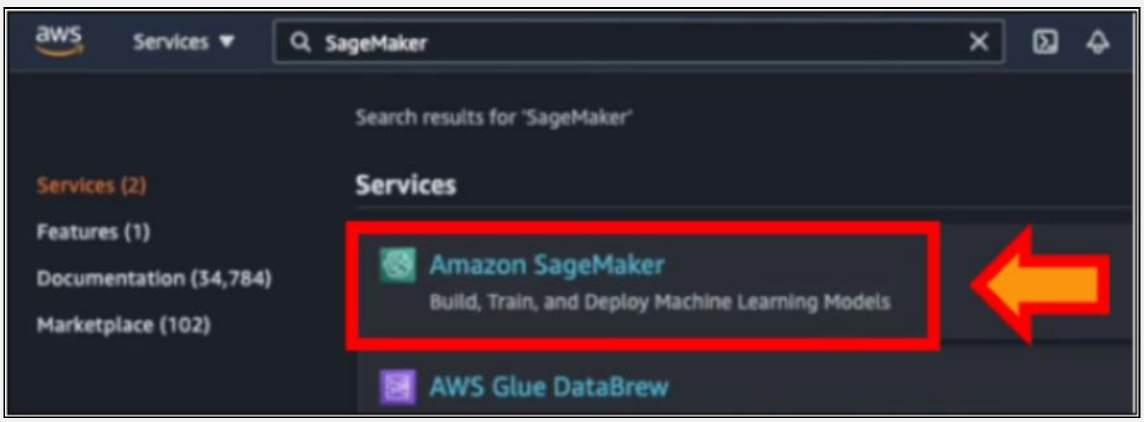
10. Click on **Upload**.



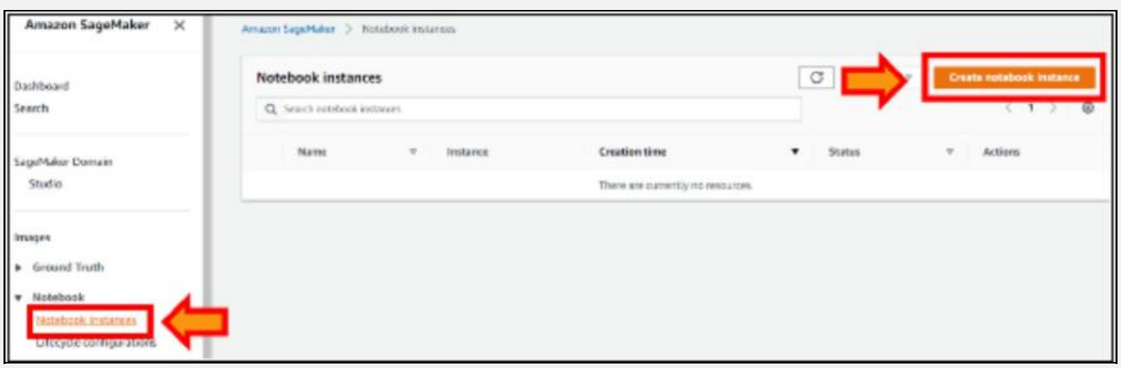
11. Click on **Upload**.

Step 2: Create Notebook Instances

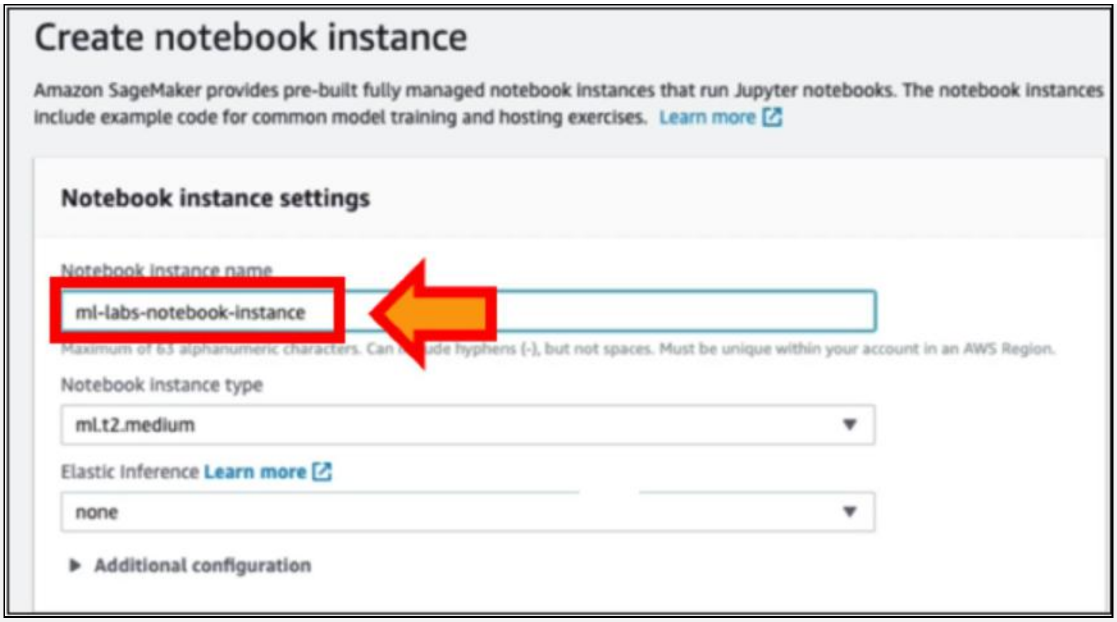
12. Search **Amazon SageMaker** in the search bar.



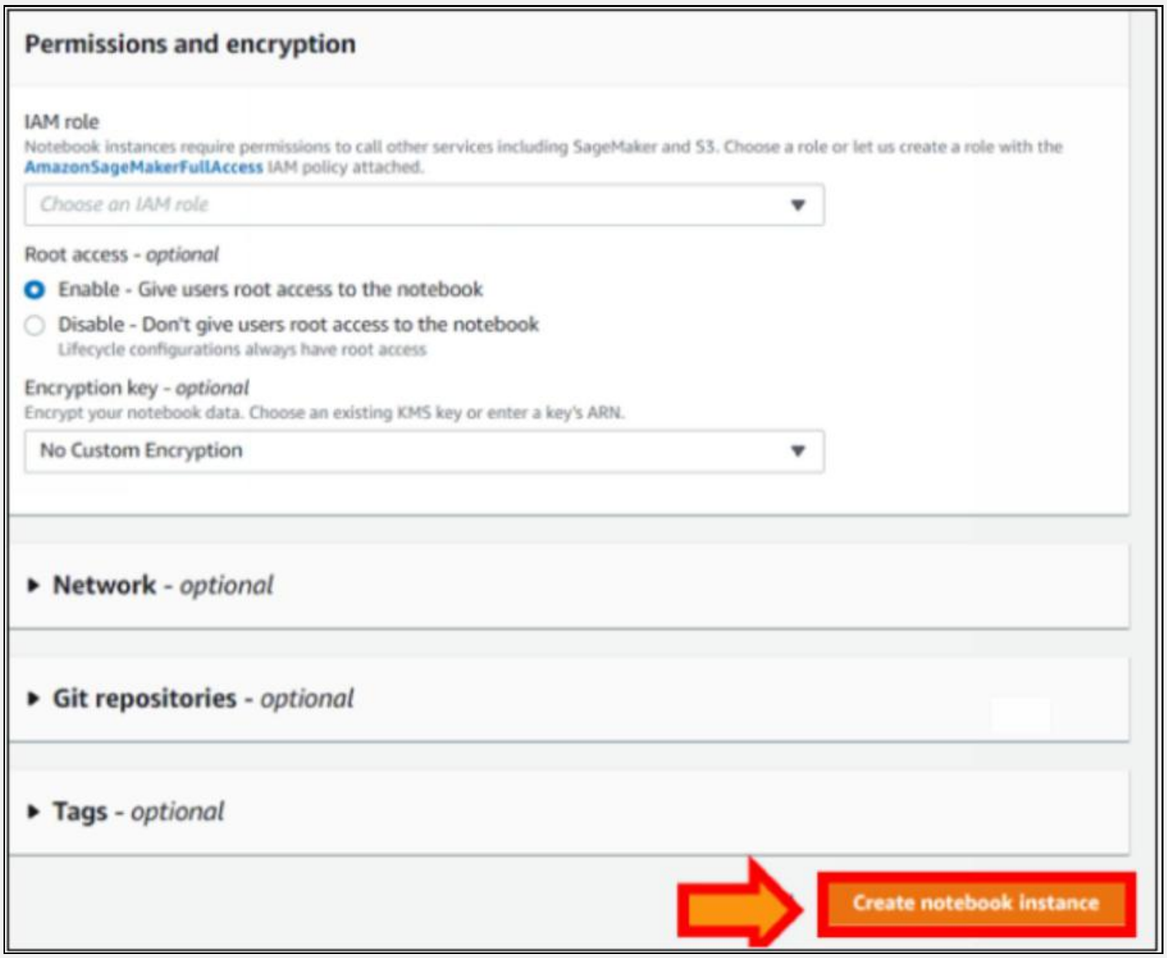
13. Click on **Notebook instances** under dashboards.
14. Click on **Create Notebook instances.**



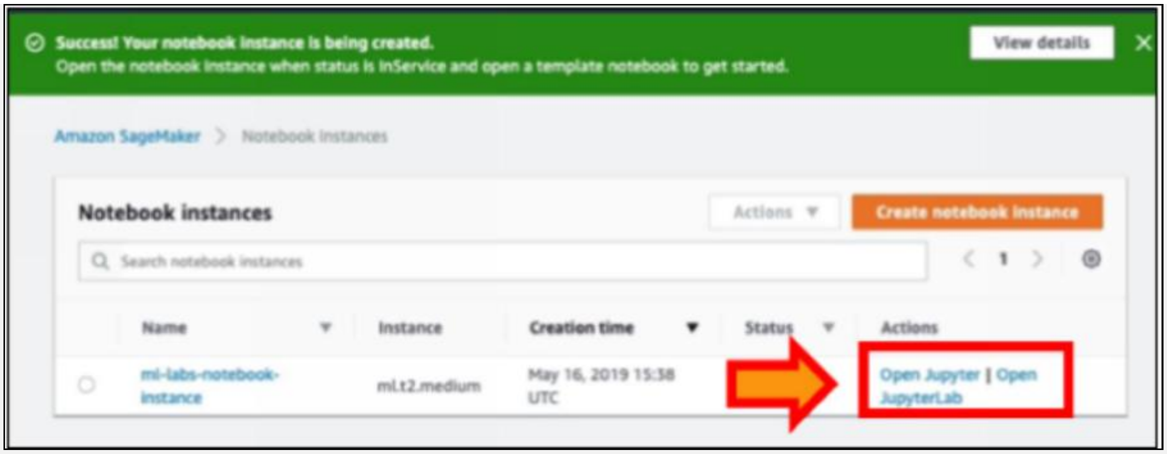
15. Define Notebook instance name.



16. Click on **Create Notebook Instance**.

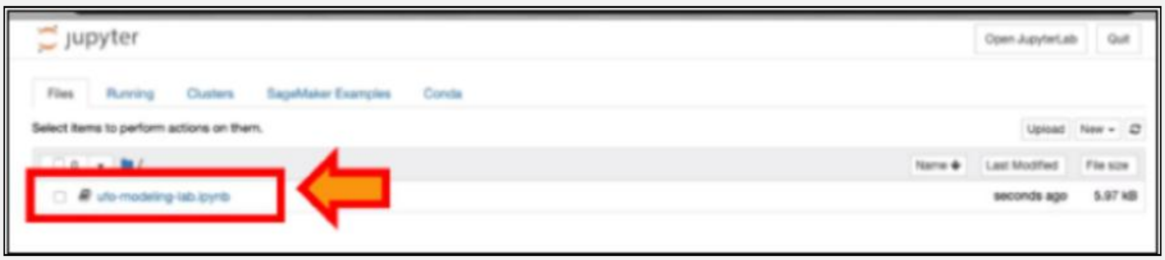


17. Click on **Open Jupyter**.

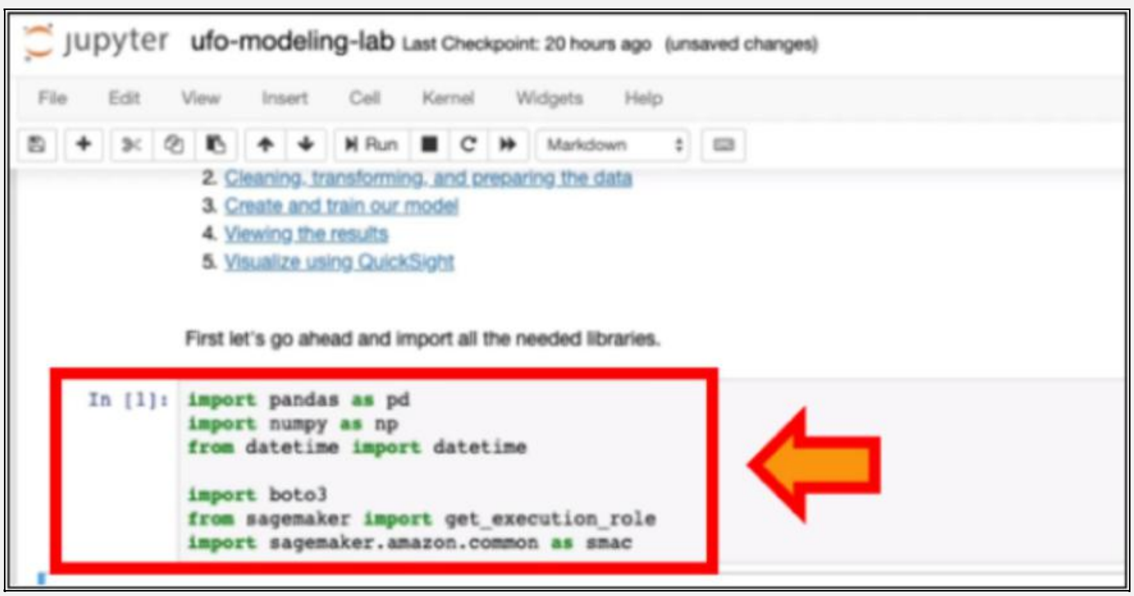


18. Click on **Upload**.

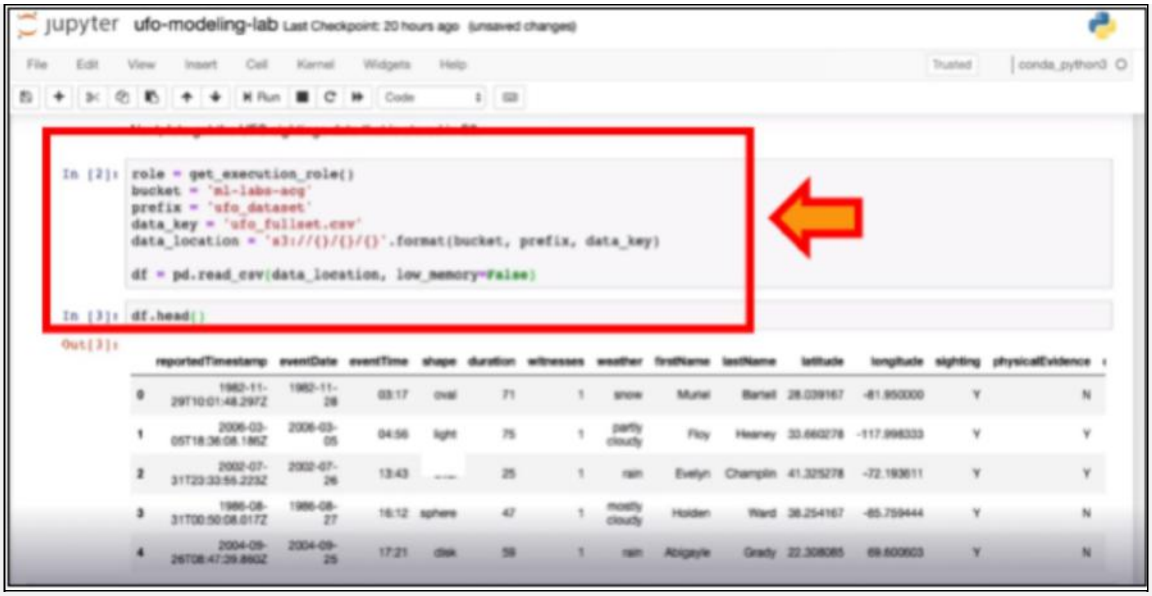
19. Select an item to perform actions.



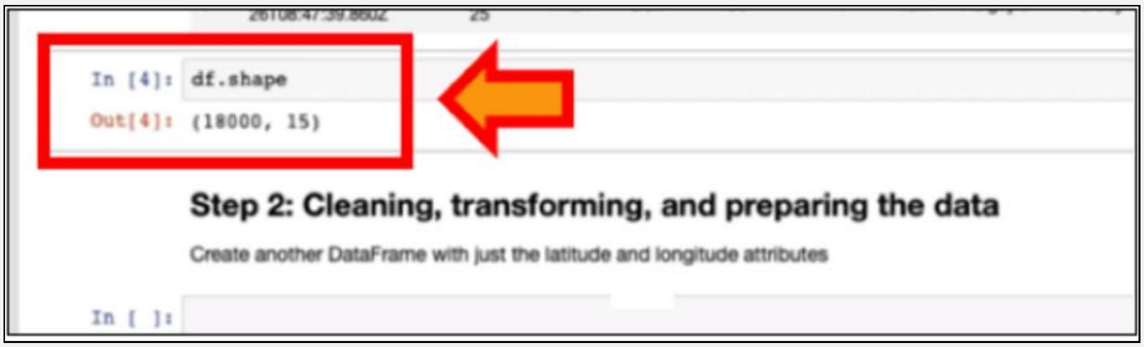
20. Import library.



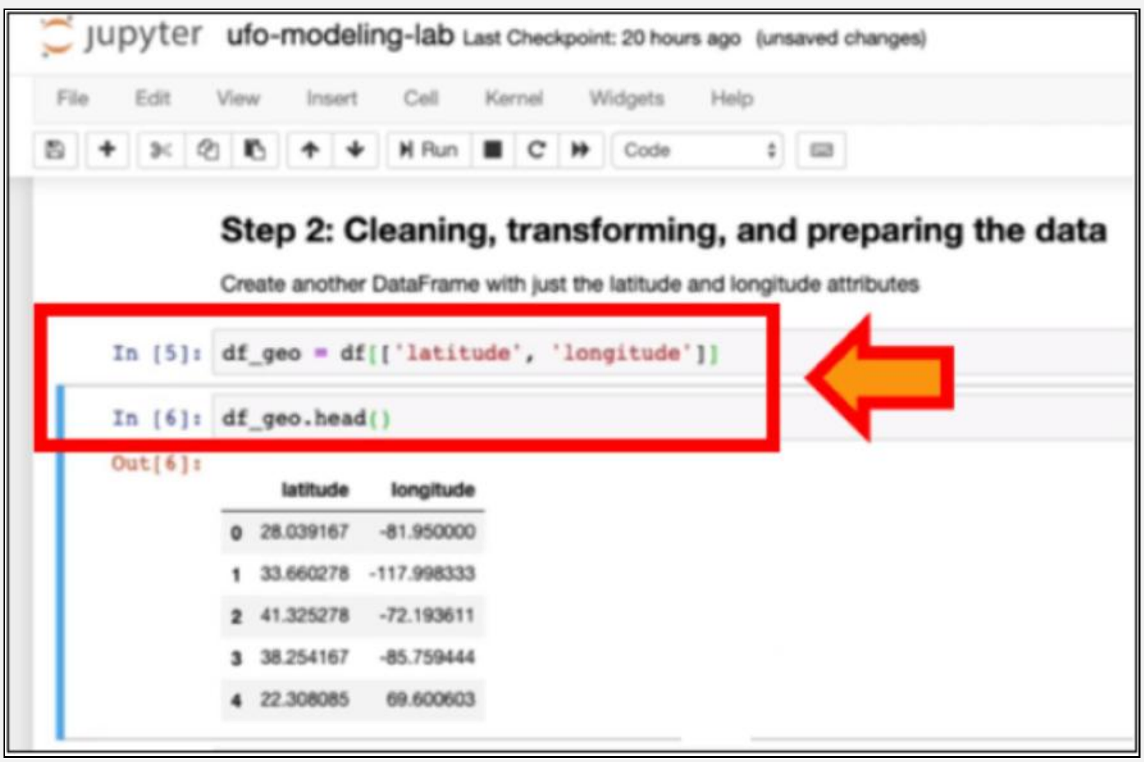
21. Import library.



22. Import library.



23. Import library.

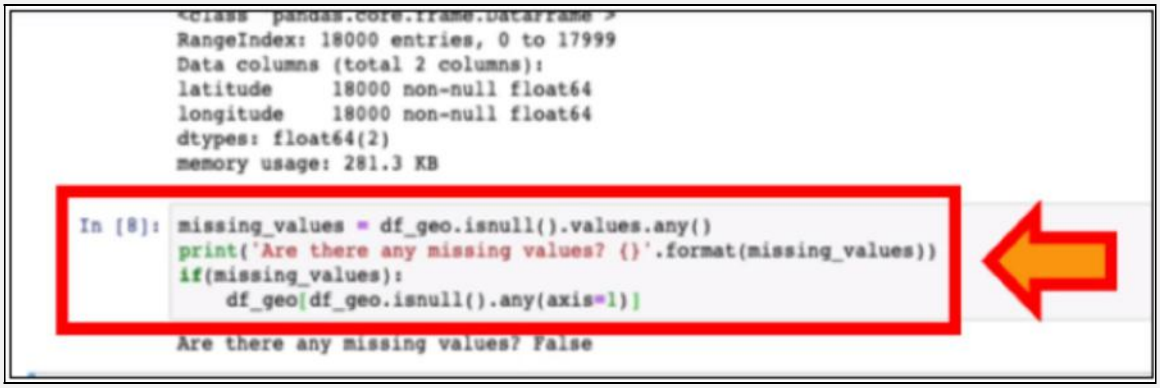


24. Import library.

```
In [7]: df_geo.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 18000 entries, 0 to 17999
Data columns (total 2 columns):
latitude      18000 non-null float64
longitude     18000 non-null float64
dtypes: float64(2)
memory usage: 281.3 KB

In [ ]:
```

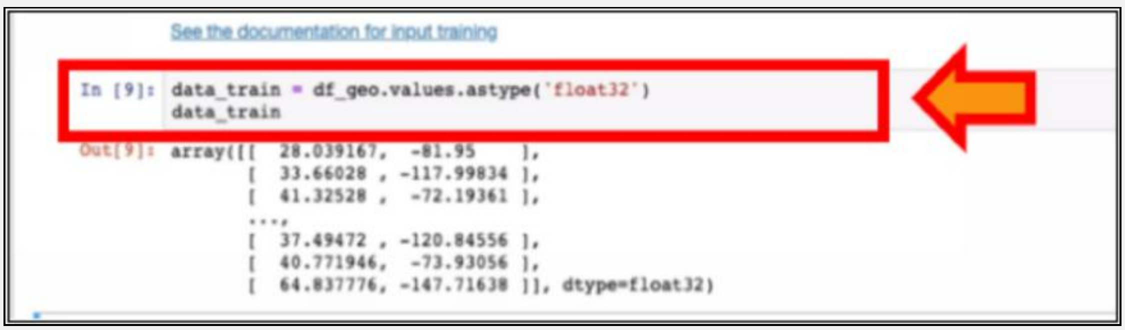
Next, let's go ahead and transform the pandas DataFrame (our dataset) into a numpy.ndarray. When we do this each row is converted to a Record object. According to the documentation, this is what the K-Means algorithm expects as training data. This is what we will use as training data for our model.

25. Import library.



```
RangeIndex: 18000 entries, 0 to 17999
Data columns (total 2 columns):
latitude      18000 non-null float64
longitude     18000 non-null float64
dtypes: float64(2)
memory usage: 281.3 KB

In [8]: missing_values = df_geo.isnull().values.any()
        print('Are there any missing values? {}'.format(missing_values))
        if(missing_values):
            df_geo[df_geo.isnull().any(axis=1)]

Are there any missing values? False
```

26. Import library.



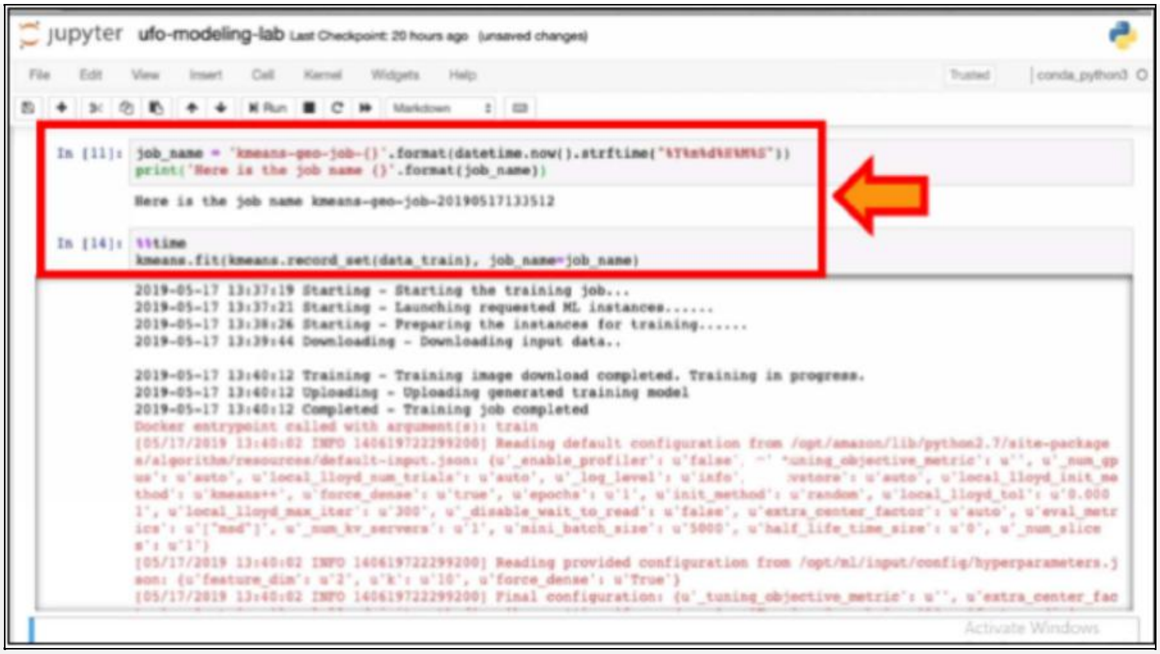
See the documentation for input training

```
In [9]: data_train = df_geo.values.astype('float32')
        data_train

Out[9]: array([[  28.039167,  -81.95    ],
               [  33.66028 , -117.99834 ],
               [  41.32528 ,  -72.19361 ],
               ...,
               [  37.49472 , -120.84556 ],
               [  40.771946,  -73.93056 ],
               [  64.837776, -147.71638 ]], dtype=float32)
```

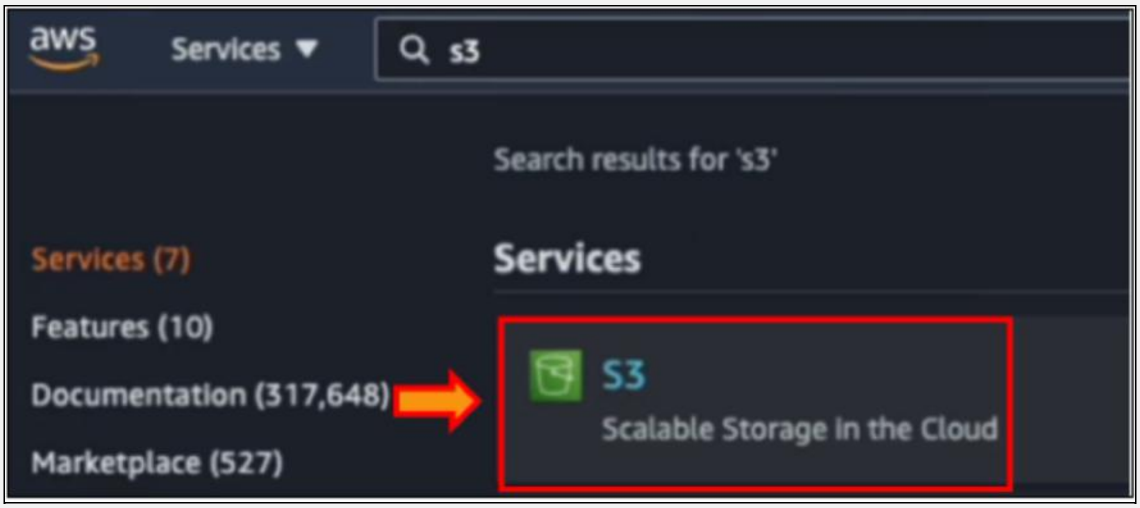
27. Import library.

```
In [10]: from sagemaker import KMeans

         num_clusters = 10
         output_location = 's3://' + bucket + '/model-artifacts'

         kmeans = KMeans(role=role,
                         train_instance_count=1,
                         train_instance_type='ml.c4.xlarge',
                         output_path=output_location,
                         k=num_clusters)

In [11]: job_name = 'kmeans-geo-job-{}'.format(datetime.now().strftime("%Y%m%d%H%M%S"))
         print('Here is the job name {}'.format(job_name))
```

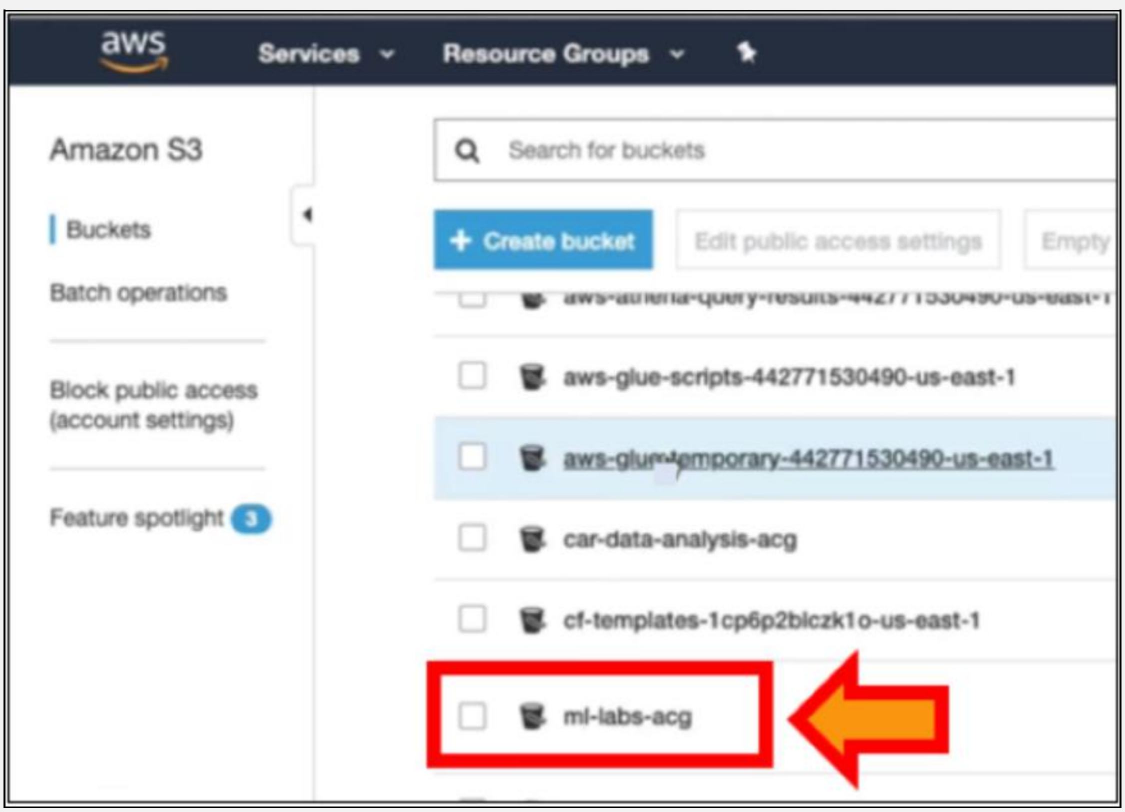
28. Import library.



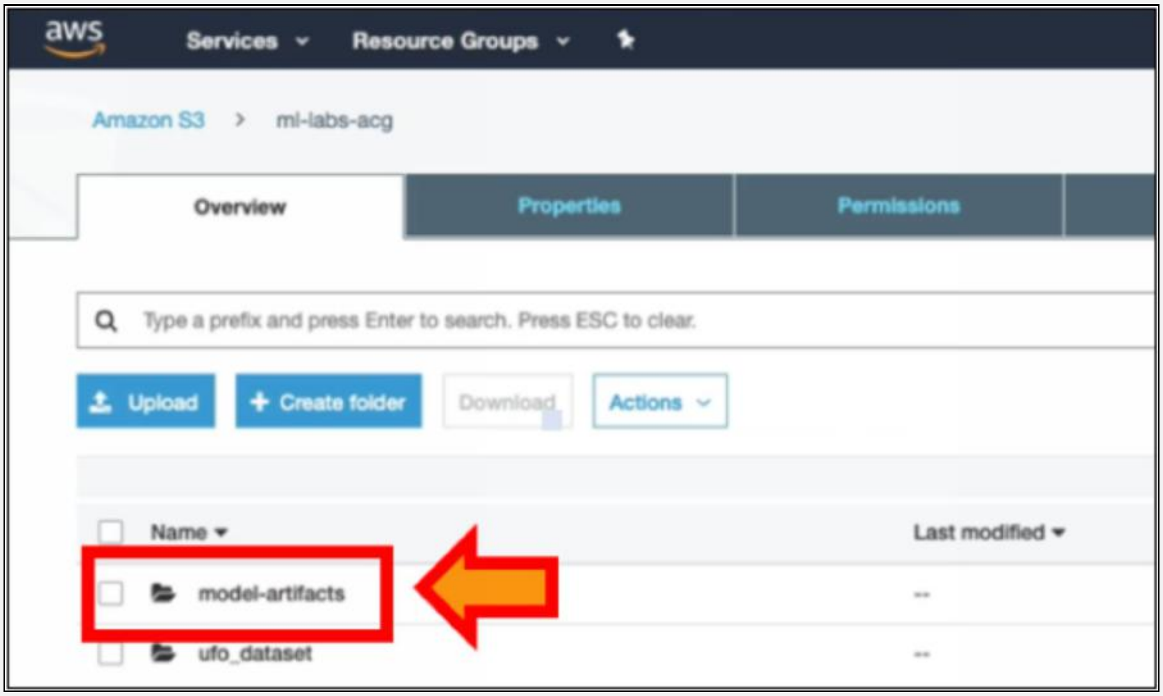
29. Search for **S3** in the search bar.



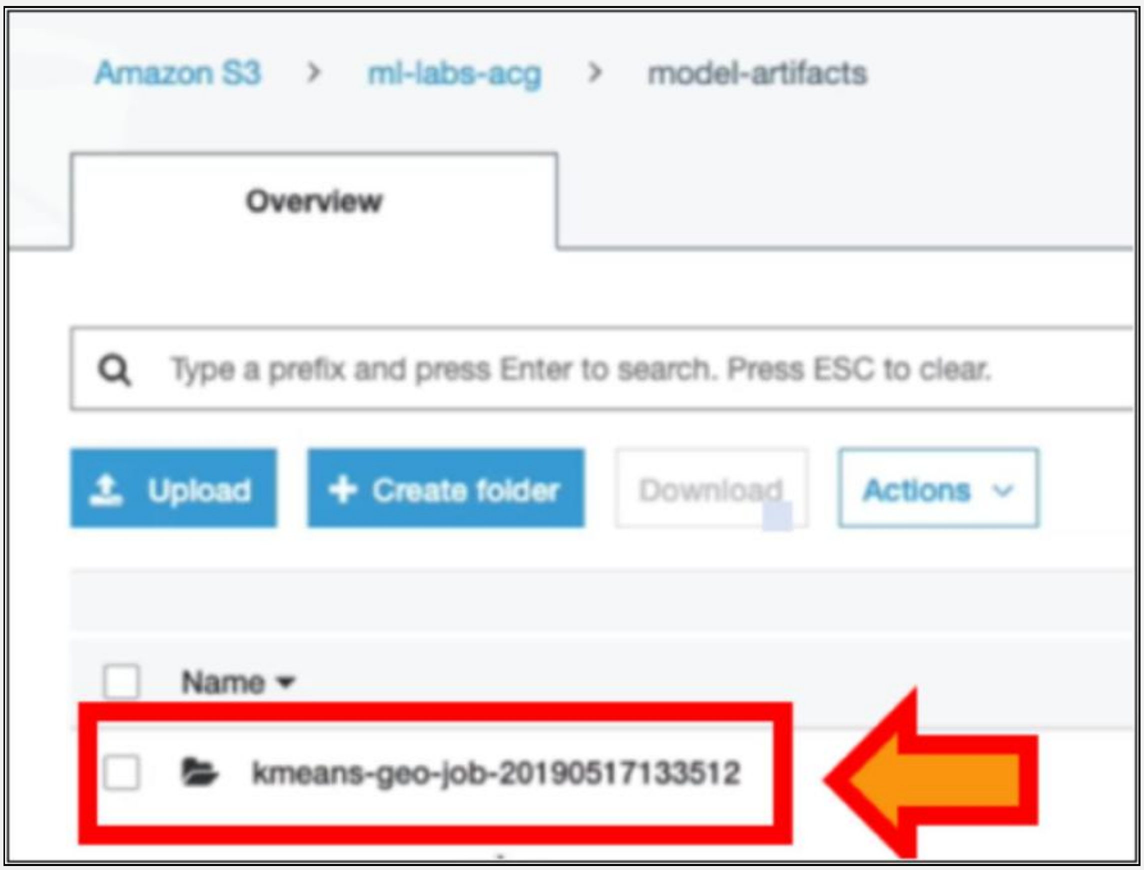
30. Select the bucket.



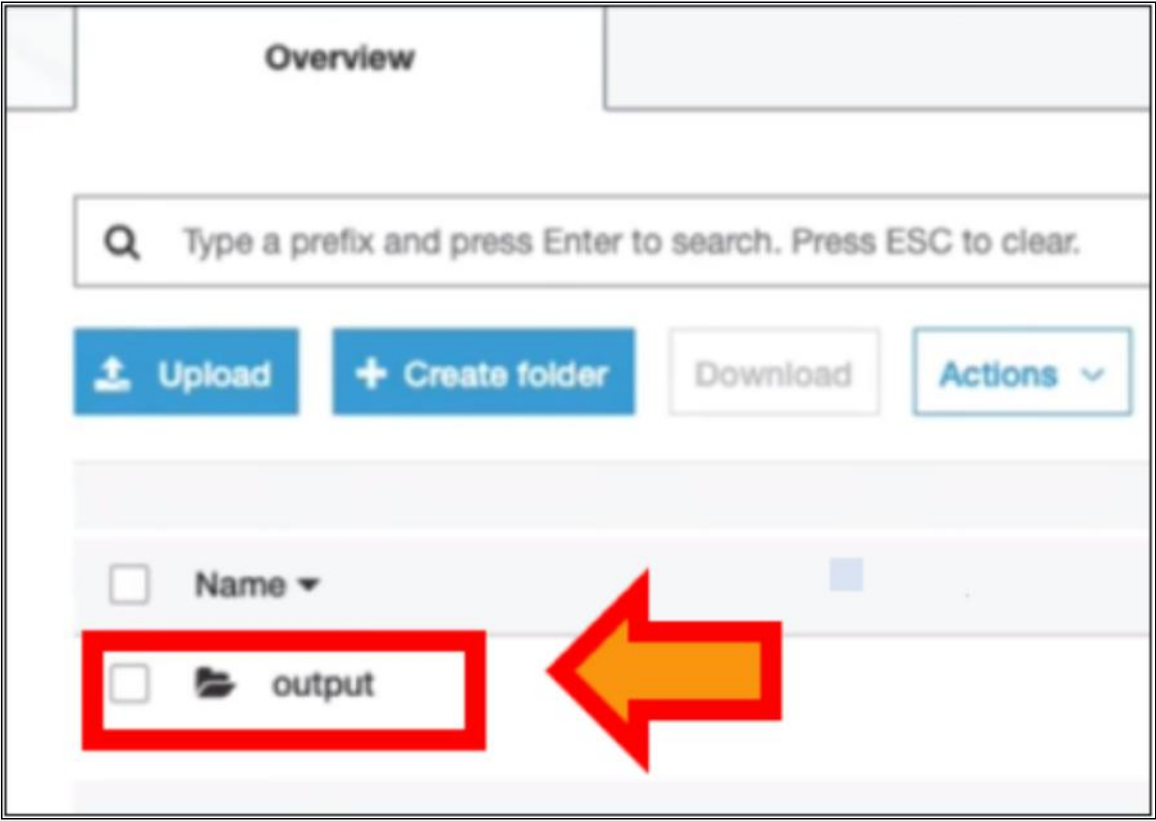
31. Select **model-artifacts**.



32. Click on **Kmeans-geo-jobs**.

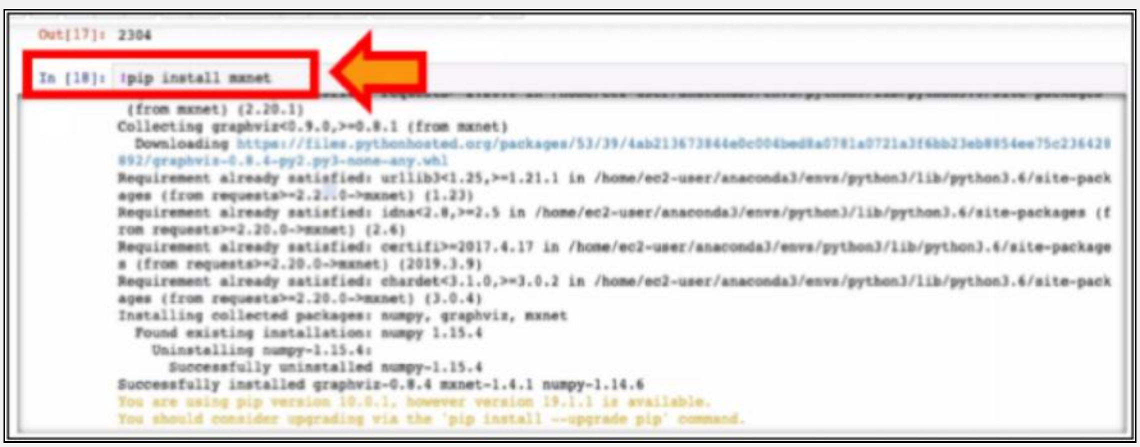


33. Click on **output**.

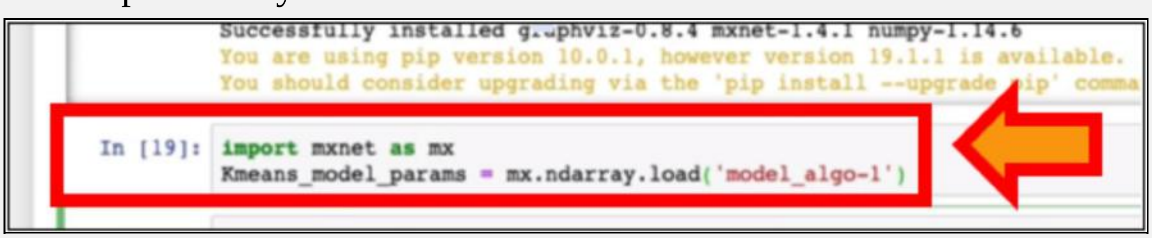


34. Import library.

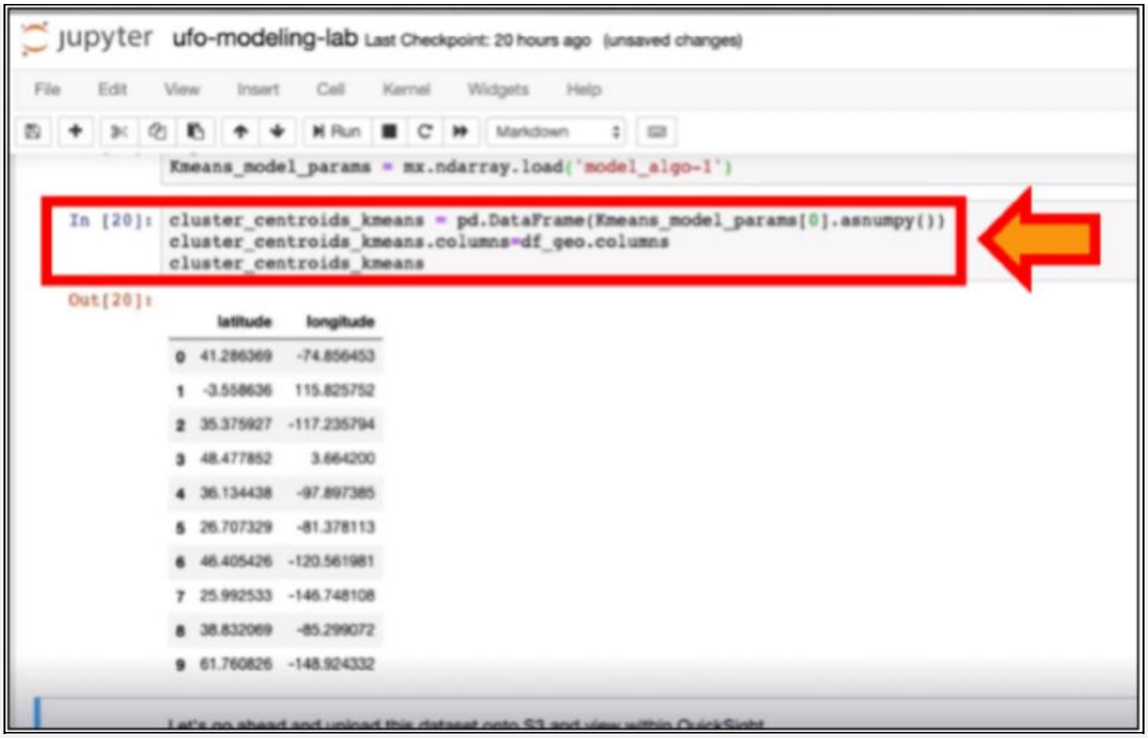
35. Import library.



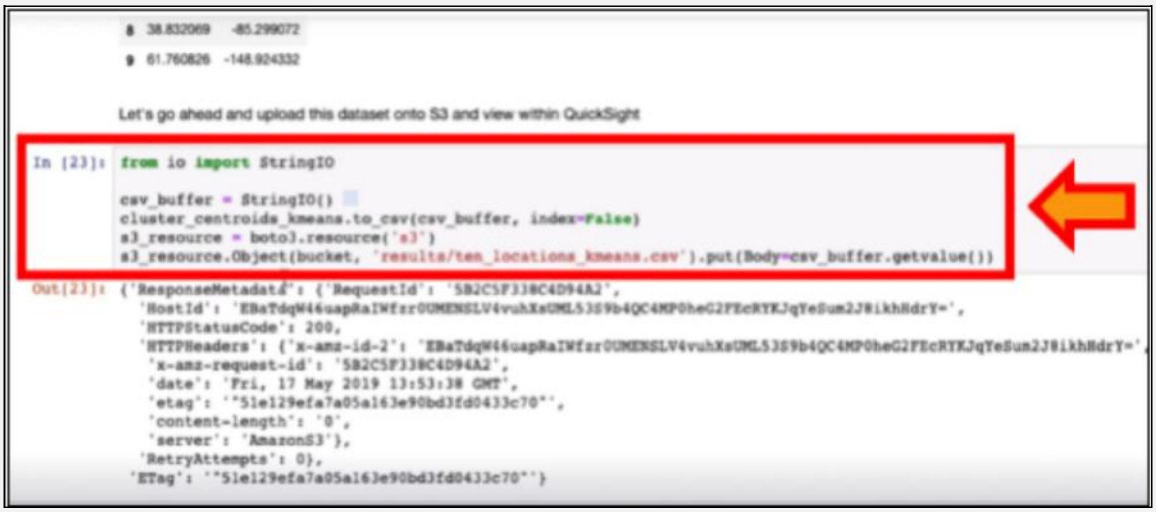
36. Import library.



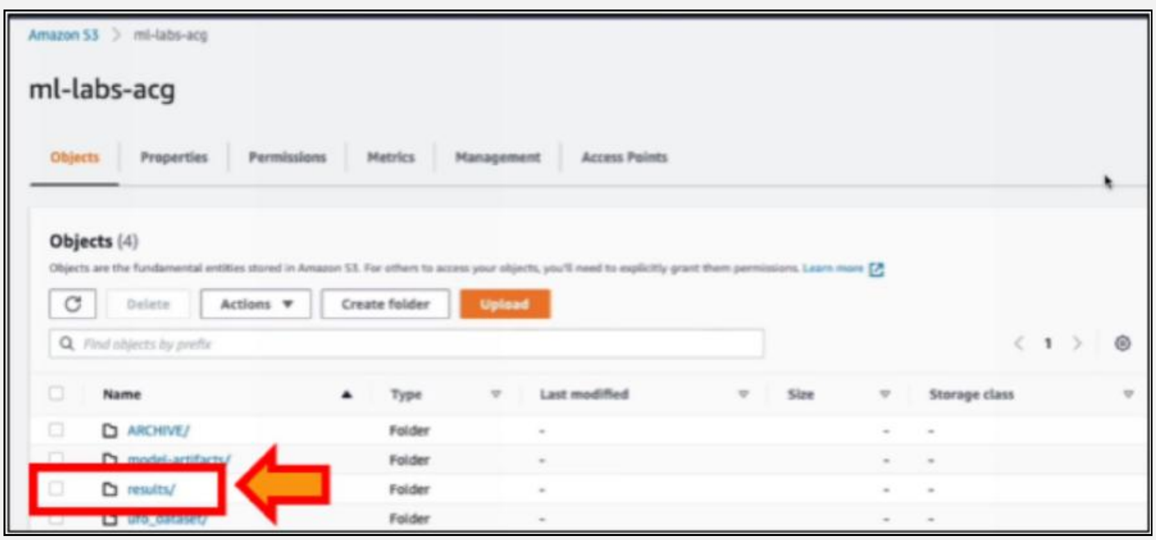
37. Import library.



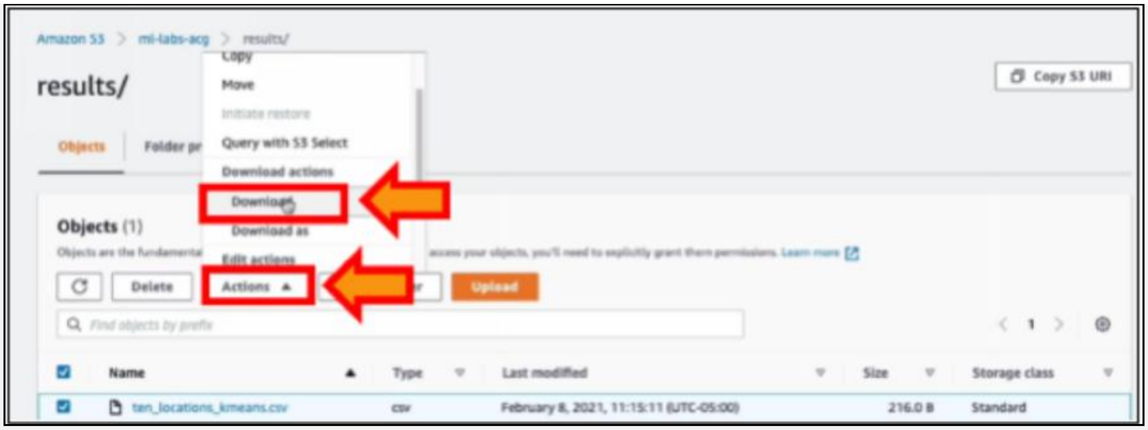
38. Import library.



39. Click on **results**.

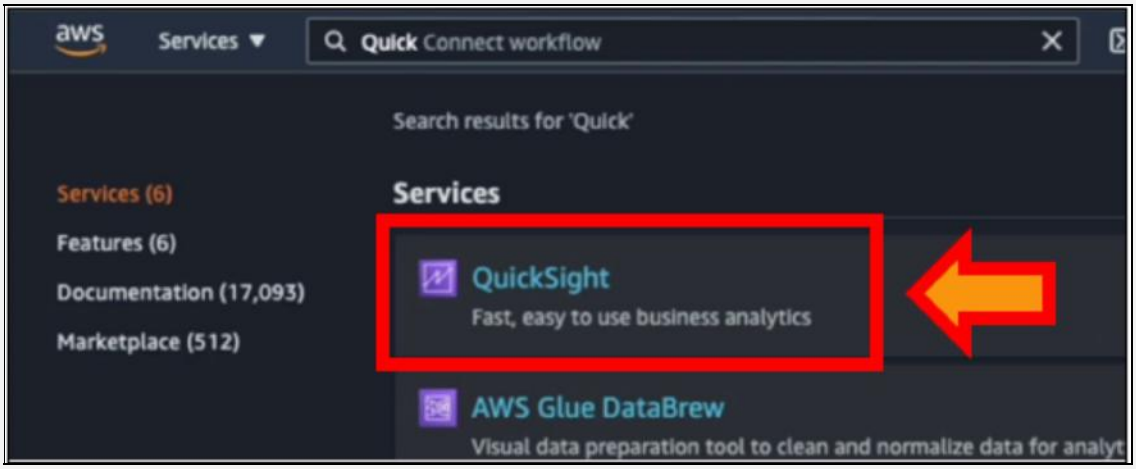


40. Select object.
41. Click on **Actions**.
42. Click on **Download**.

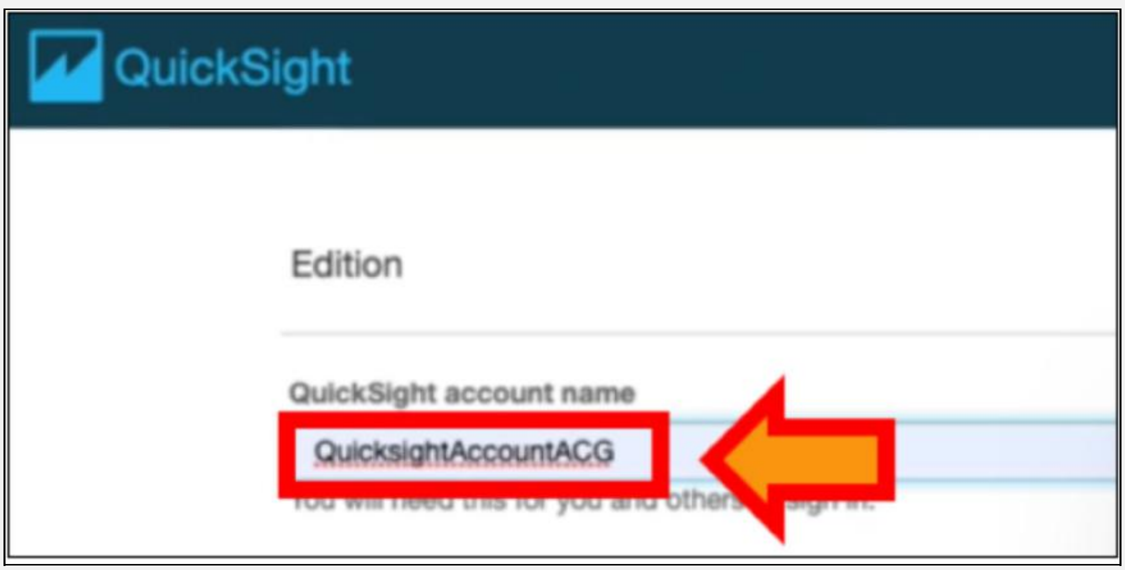


Step 3: Create a QuickSight Account

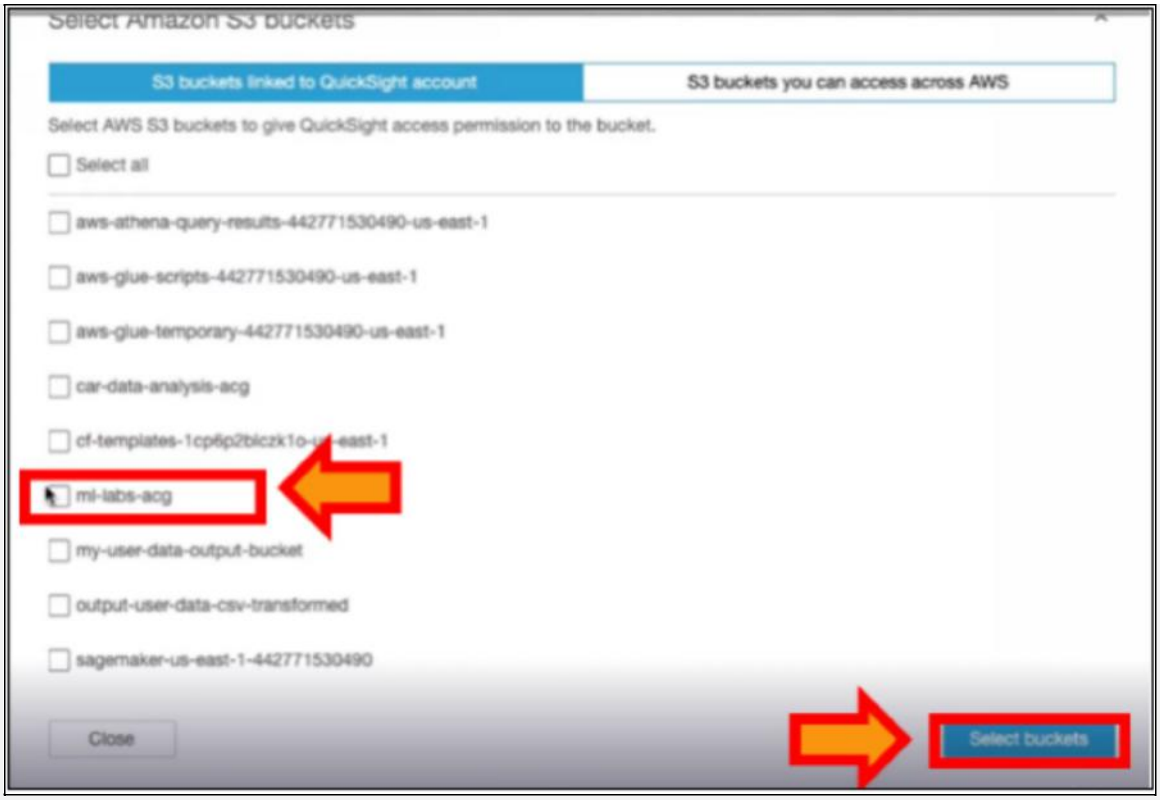
43. Search for **QuickSight** in the search bar.



44. Define the QuickSight account name.



45. Select **Amazon S3 Bucket.**
46. Click **Select bucket**.

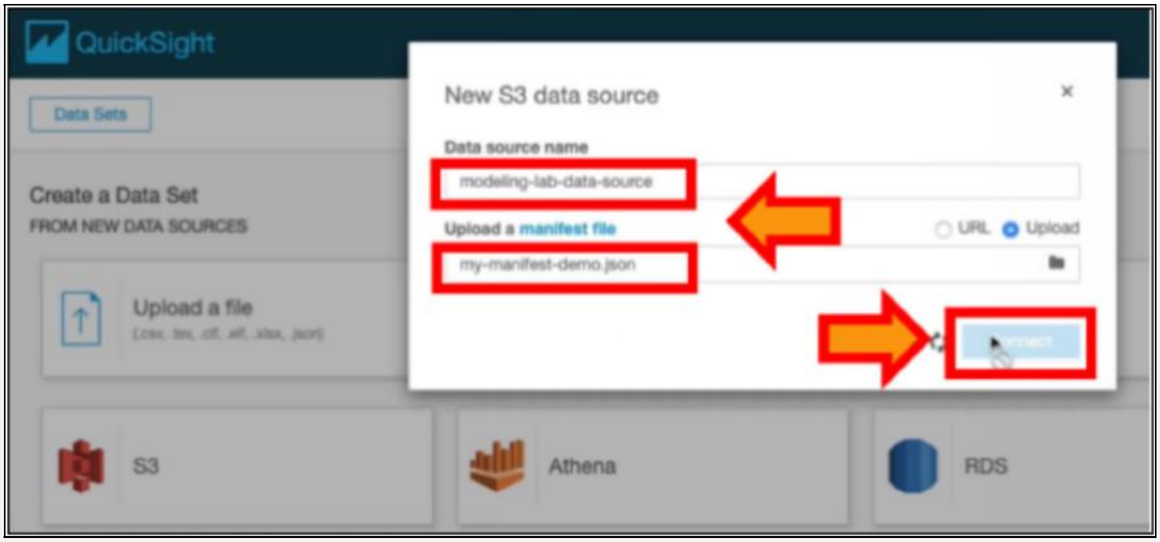


47. Click on **Finish**.

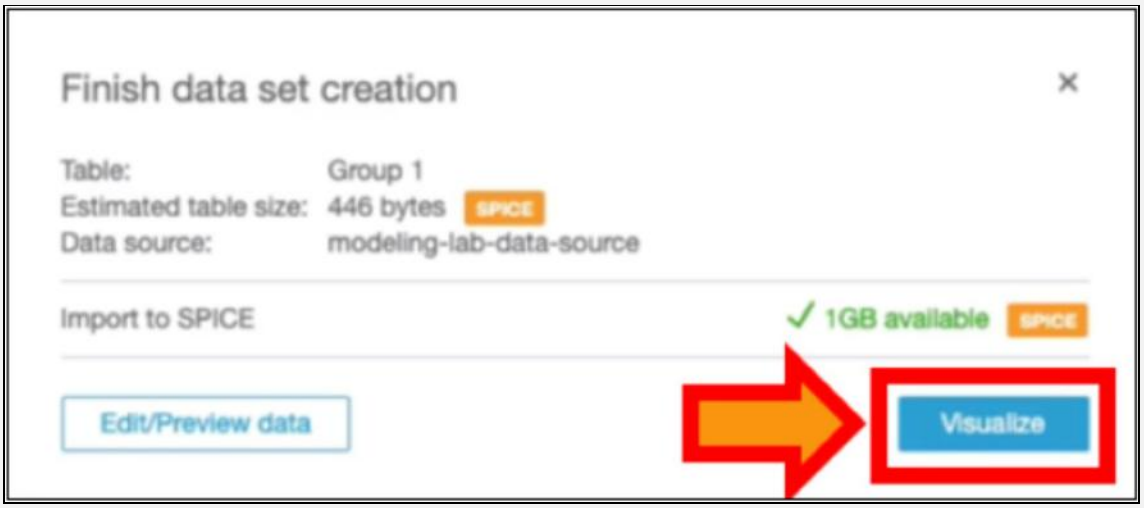
48. Click on **Go to Amazon QuickSight**.

49. Click on “Manage data.”

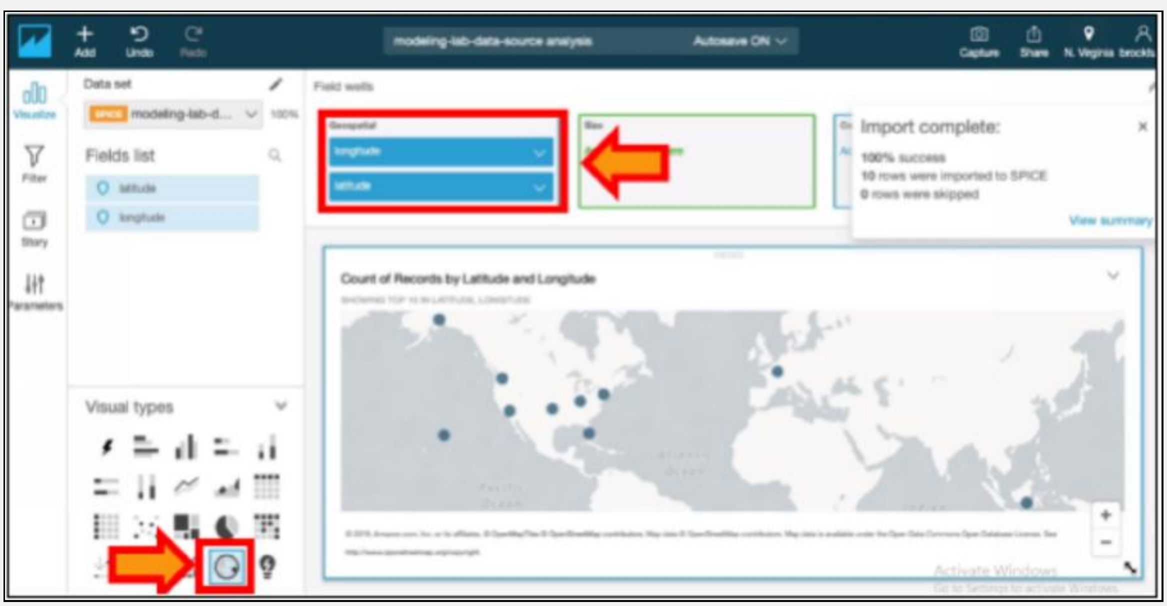
50. Click on **S3**.
51. Define the data source name.
52. Upload a manifest file.
53. Click on **Connect**.



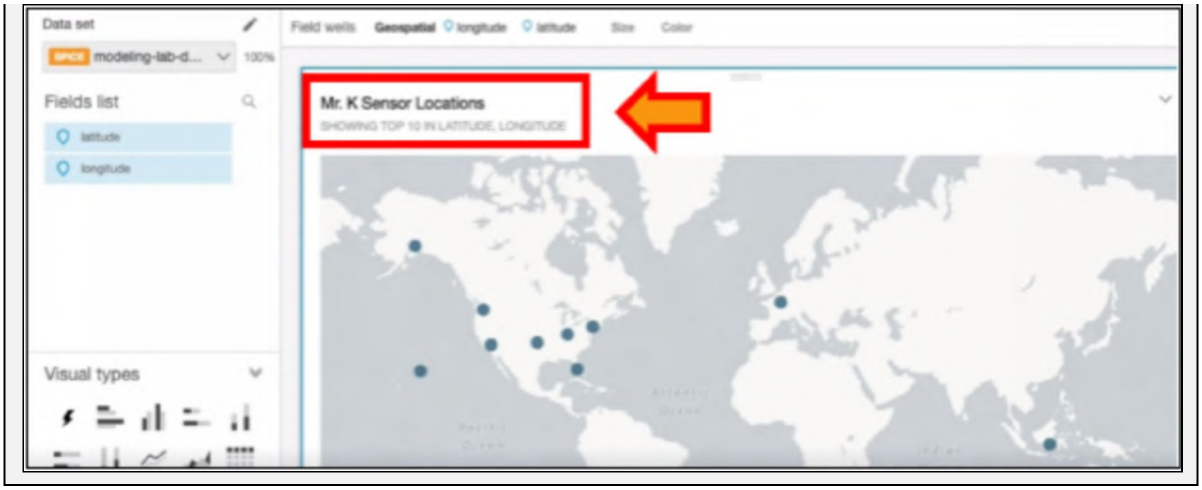
54. Click on **Visualize.**



55. Select the Visual type.
56. Drag the Field list into Geospatial.



57. Rename Geospatial if you want.



Lab 08: Introducing Jupyter Notebooks (AWS SageMaker)

Certifications	AWS Machine Learning - Specialty
Cloud Provider	AWS
Time Required	1 Hr.
Tags	SageMaker, AWS

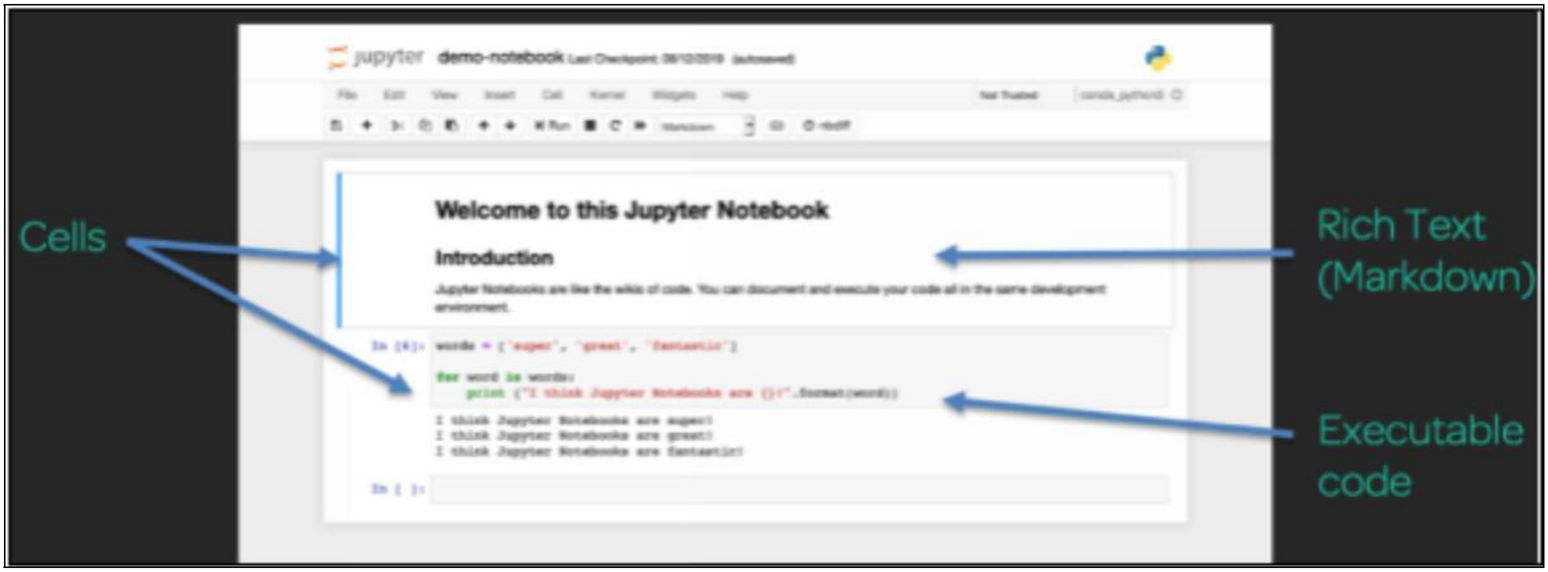
Lab Prerequisites

- Familiarity with basic AWS Cloud Computing concepts and terminology
- An AWS account with an active subscription

Service Introduction

You may create and share documents with live code, mathematics, visuals, and narrative text using the open-source web application Jupyter Notebook.

You can build and operate Jupyter notebooks from your DLAMI instance using a Jupyter notebook server. You can use Jupyter notebooks to run Machine Learning (ML) experiments for inference and training while utilizing the AWS infrastructure and DLAMI package libraries.



Case Study Enterprise EMRs – Savana

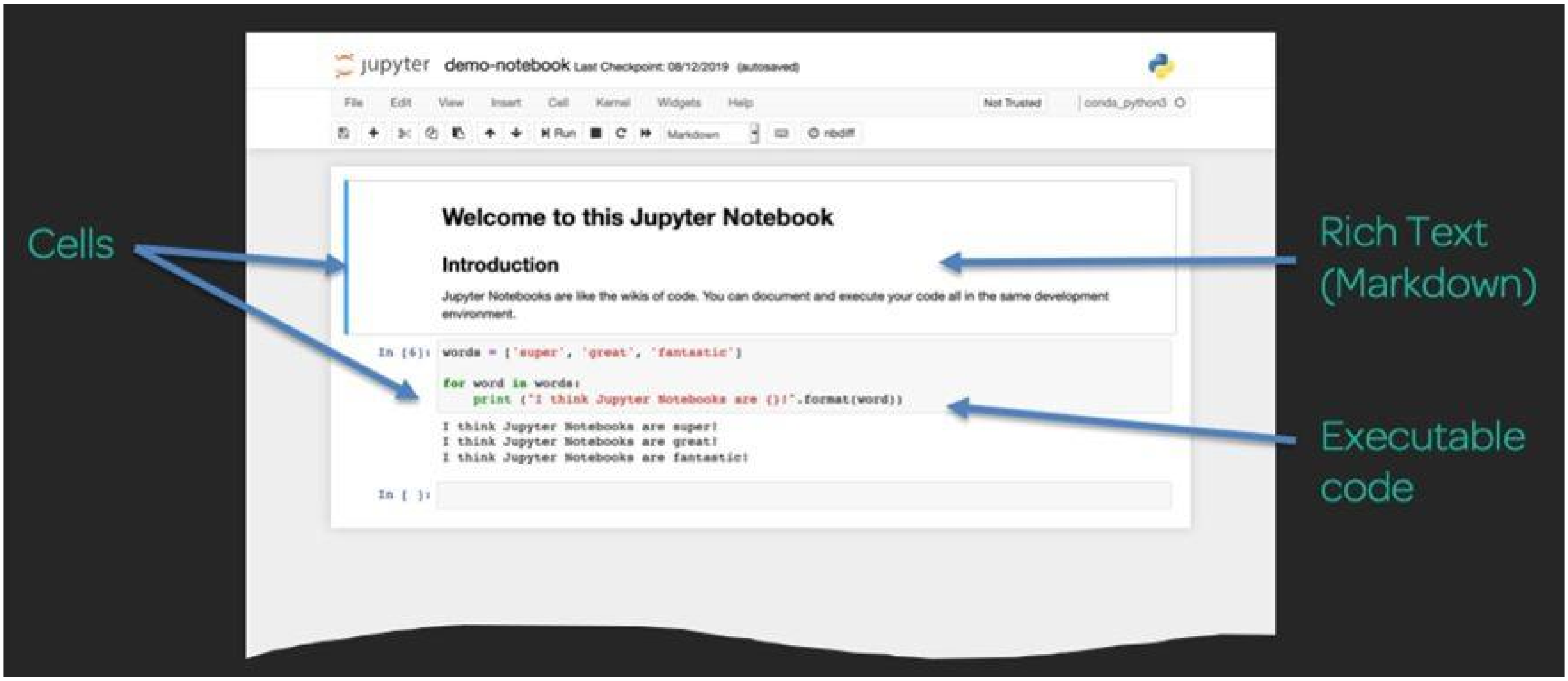
Business Challenge

You are working in an organization. You are provided some sample heights and weights of individual penguins. You have to use the data (stored in a csv file) to train a data model and create inferences so that you can guess what a penguin weighs if you know its height. Hence, you need to build a machine learning model that tells you approximately what a penguin should weigh based on height.

Proposed Solution

The solution is for you to use AWS services to automate all the tasks. You create an AWS SageMaker hyperparameter tuning job with different ranges of values for the hyperparameter to find the best configuration, which minimizes the validation: objective_loss metric. The reason that you use this metric is that it is used in multi-classification problems. This metric measures the performance of the classification model, and what it does is repeatably calculate the difference between the values that the model predicts and the actual values of a label. Hence, every time it passes over data and makes predictions, it recalculates the objective loss and tries to minimize this value overall. Hence, that is the task here. AWS recommends that you minimize this value when using it as our objective metric.

Lab Diagram



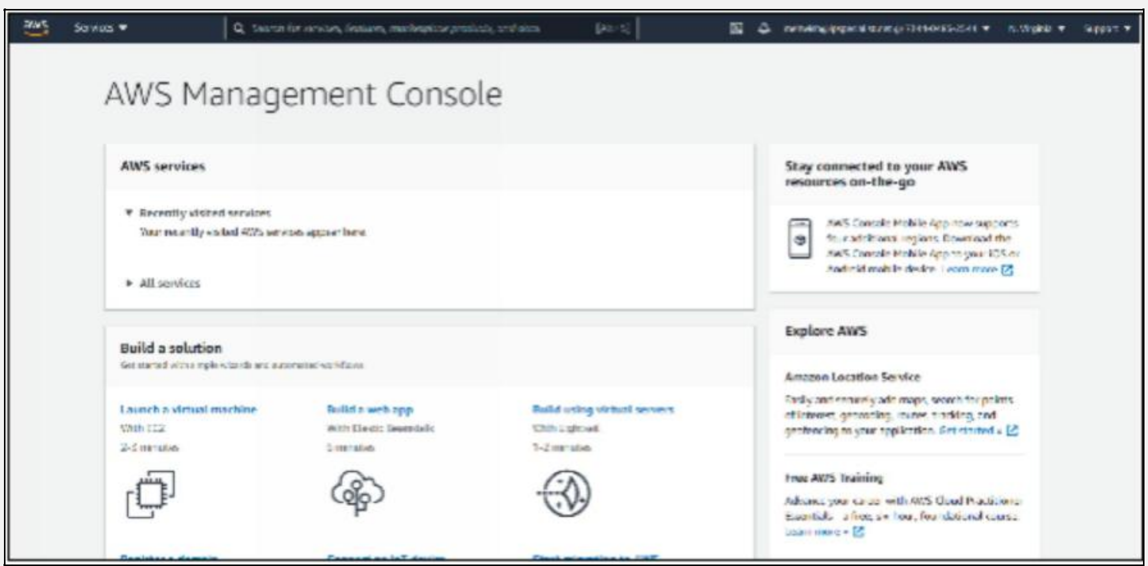
<u>Implementation Steps</u>

1. Navigate to Jupyter Notebook.
2. Browse Jupyter Notebooks.
3. Create New Folders and Files
4. Build Your Own Notebook
5. Use Markdown to Add Richly Formatted Text to a Notebook

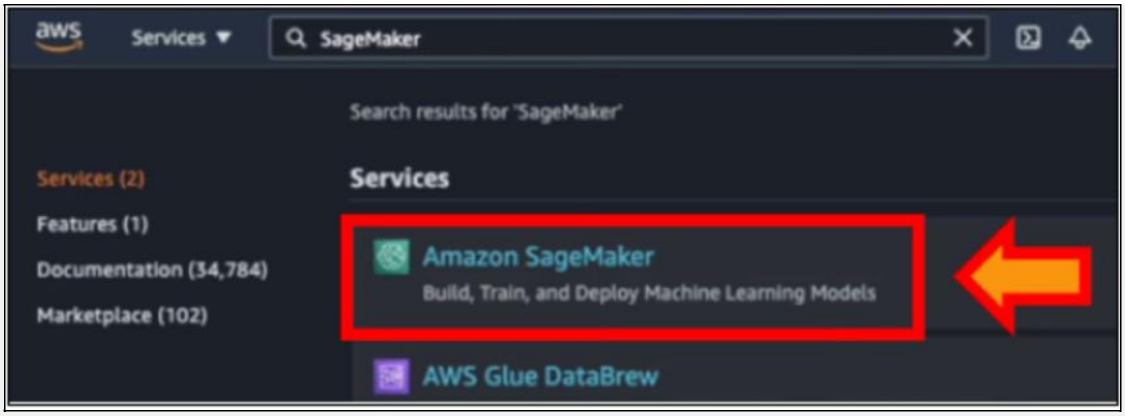
Solution

Step 1: Navigate to Jupyter Notebook

1. Login to **AWS Management Console**.



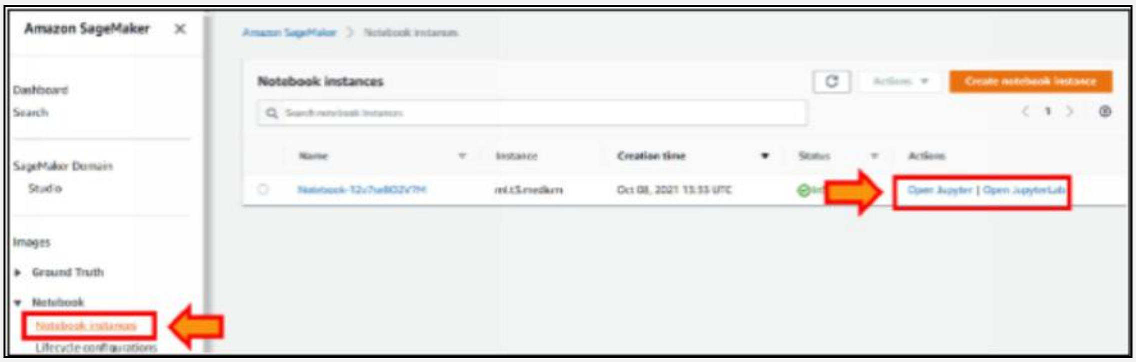
2. Search for **SageMaker** in the search bar.



3. Click on **Notebook Instances** under the dashboard.

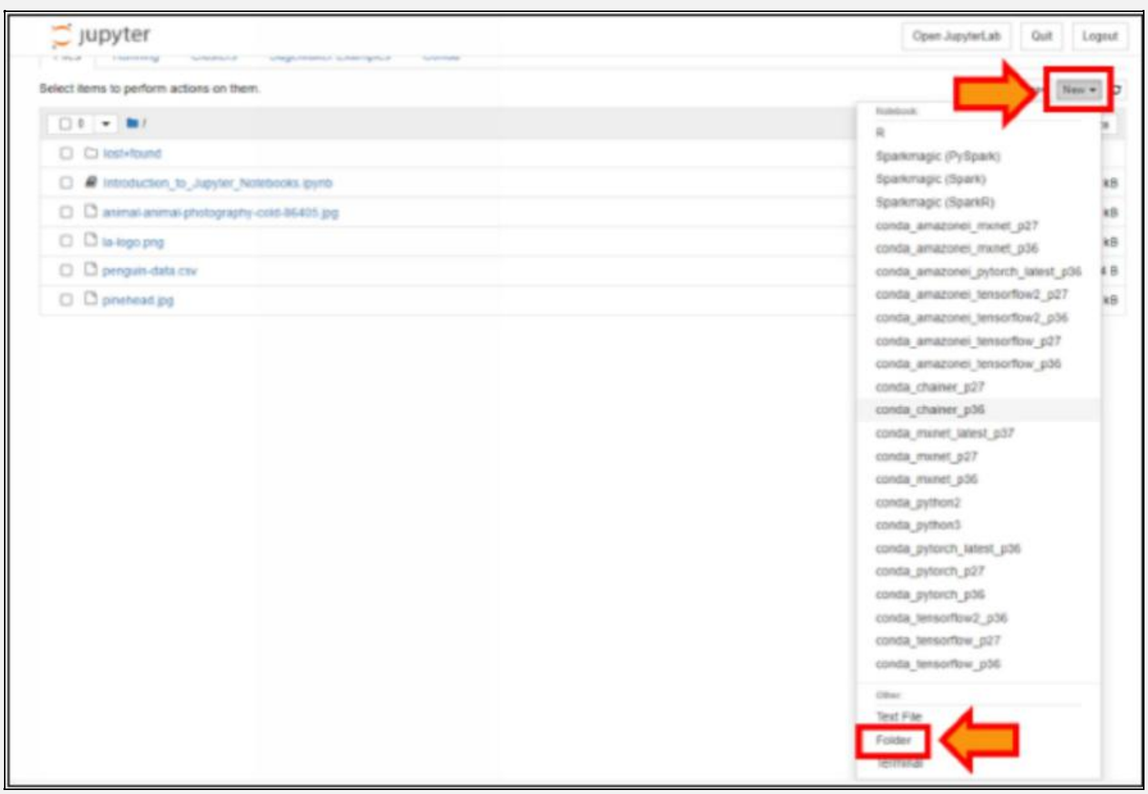
Step 2: Browse Jupyter Notebooks

4. Click on **Open Jupyter**.

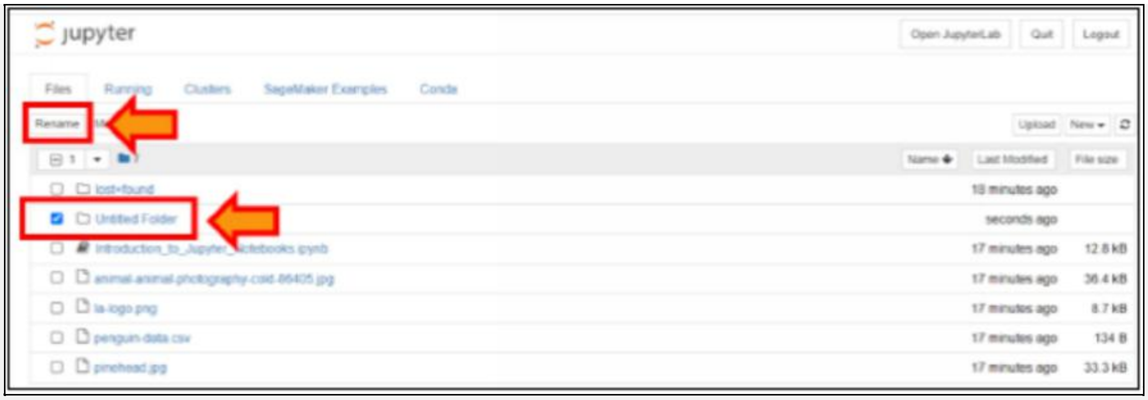


Step 3: Create New Folders and Files

5. Click on **New**.
6. Click on **Folder**.



7. Select Folder.
8. Click on **Rename**.

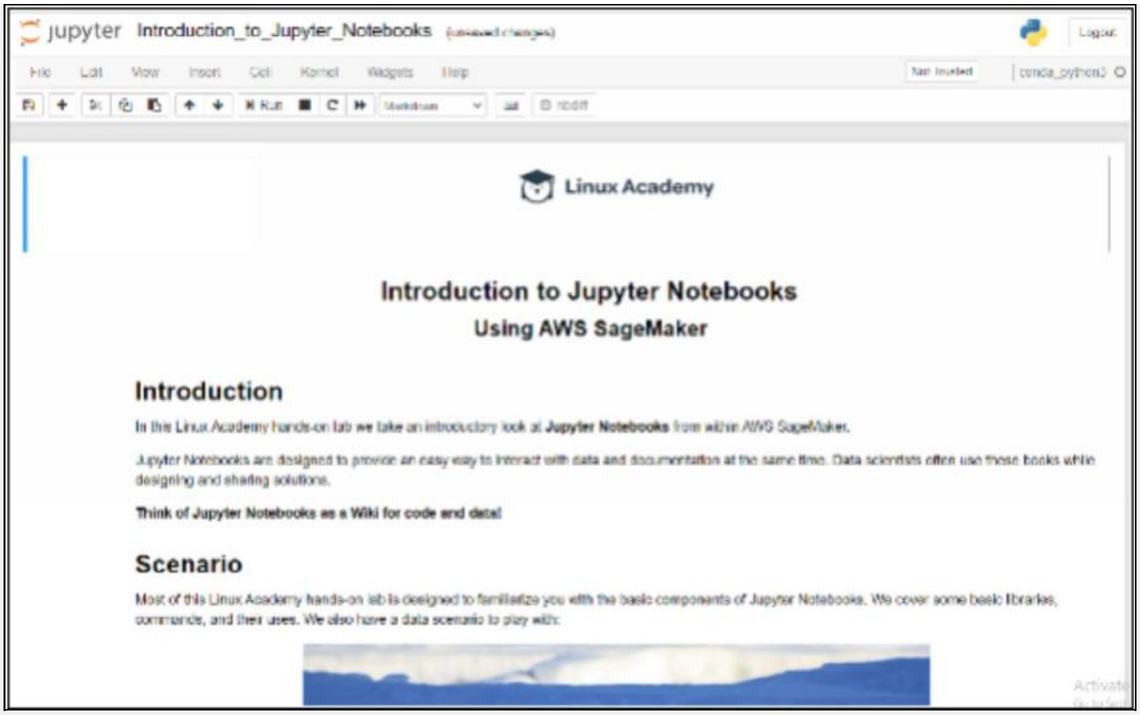


9. Define Name.
10. Click on **Rename**.

11. Click on the **new folder**.

12. Select the item to perform the action.

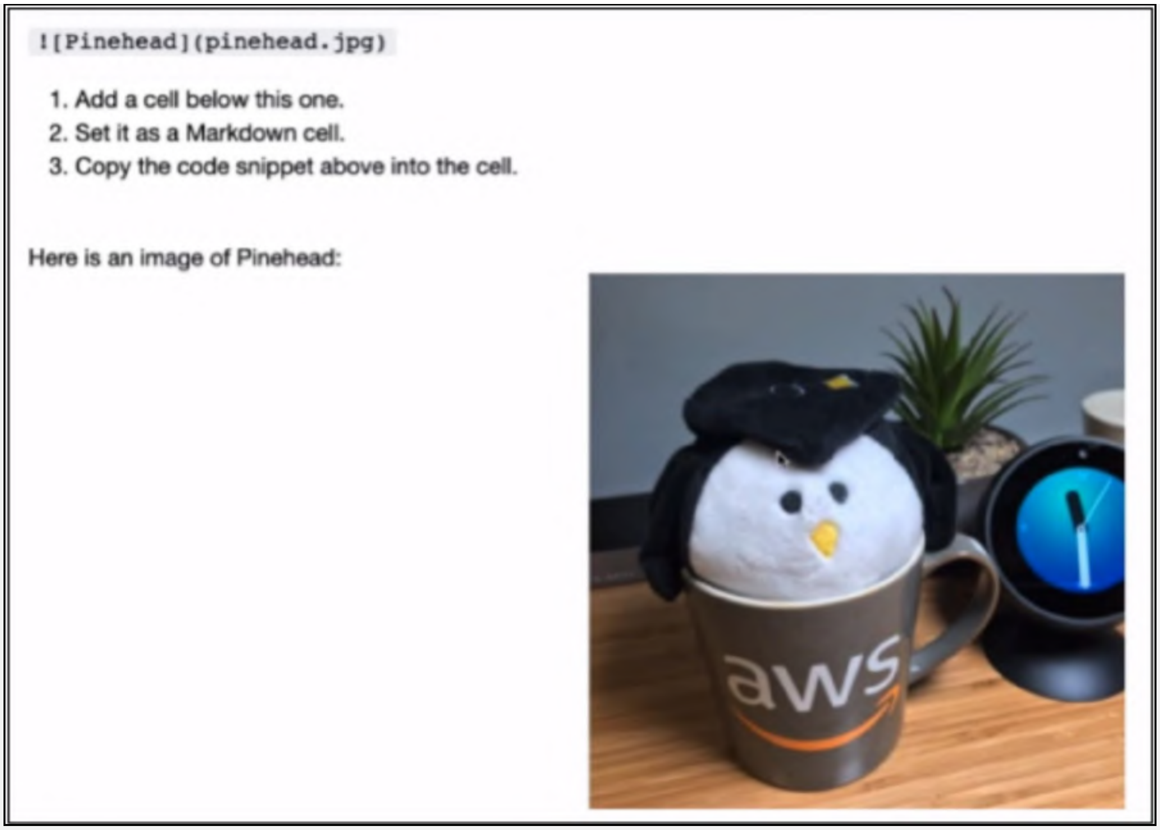
13. Go through to the **Introduction to Jupyter Notebooks**.



14. Click on **Insert.**
15. Click on **Insert Cell Below.**

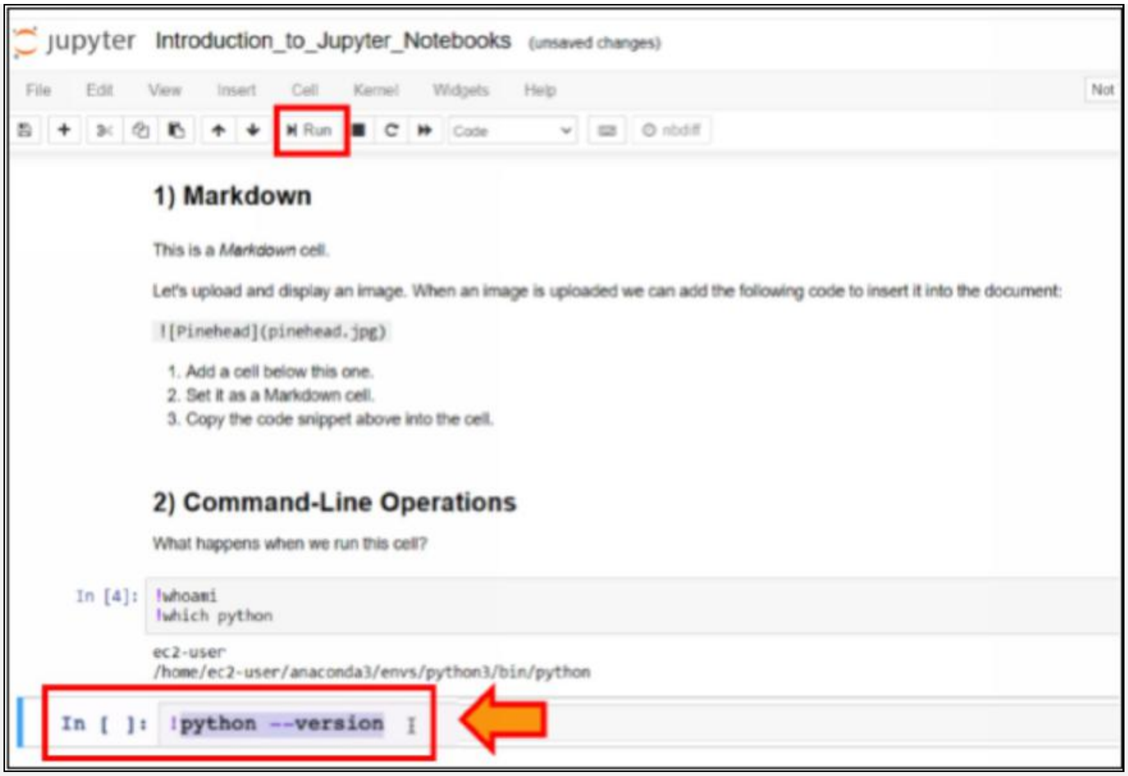
16. Select the cell. Click on **Run.**

17. Go through the output result.

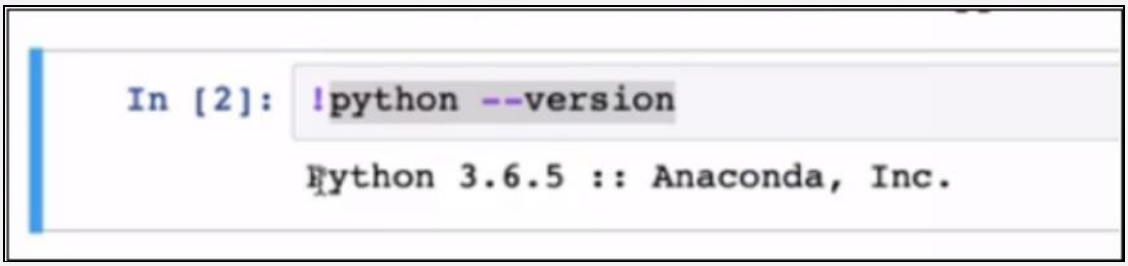


18. Select the cell. Click on **Run**.

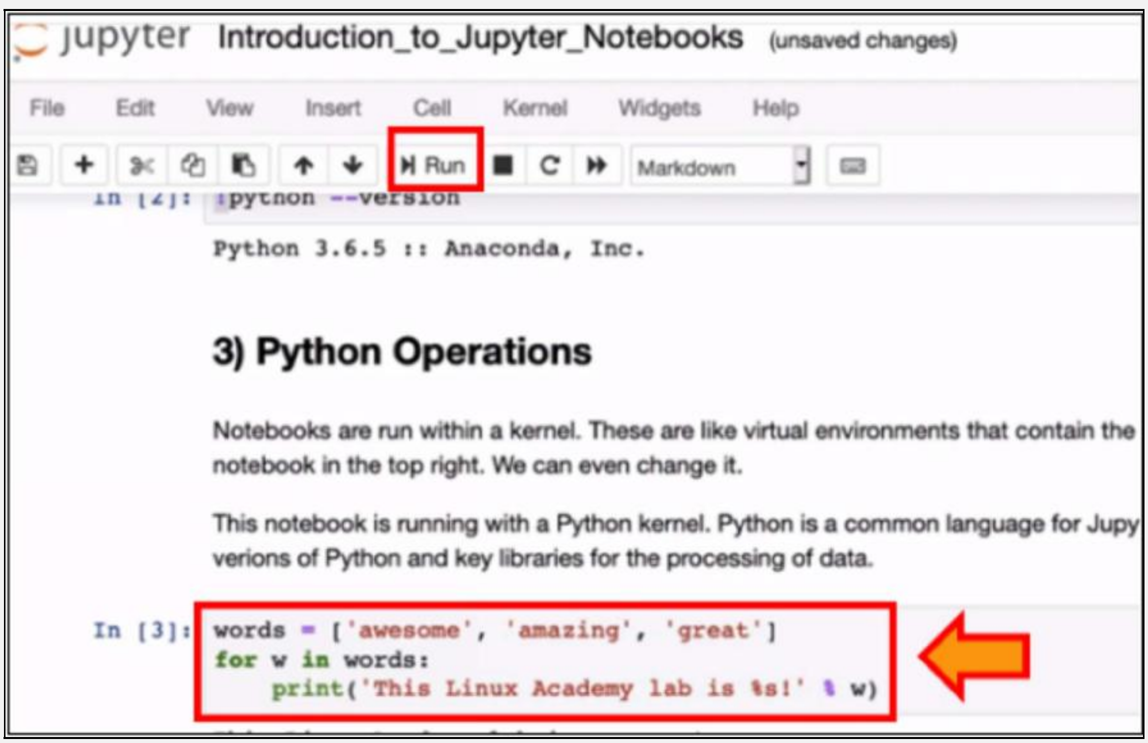
19. Select the cell. Click on **Run**.



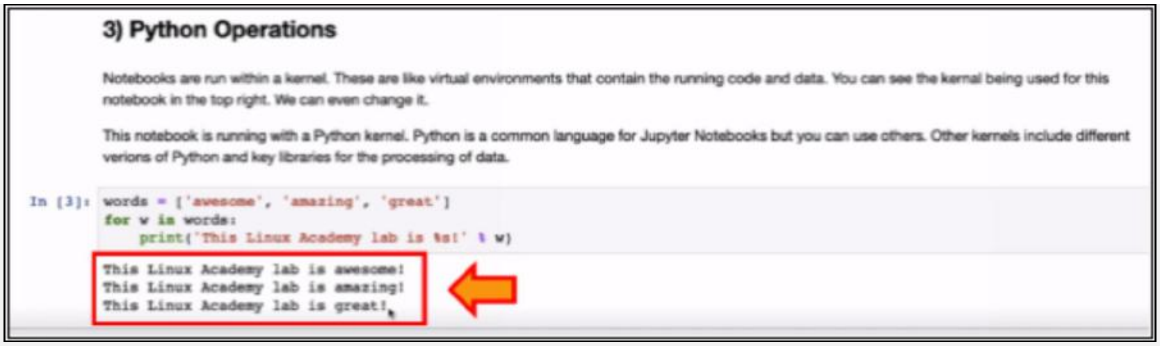
20. Go through the output result.



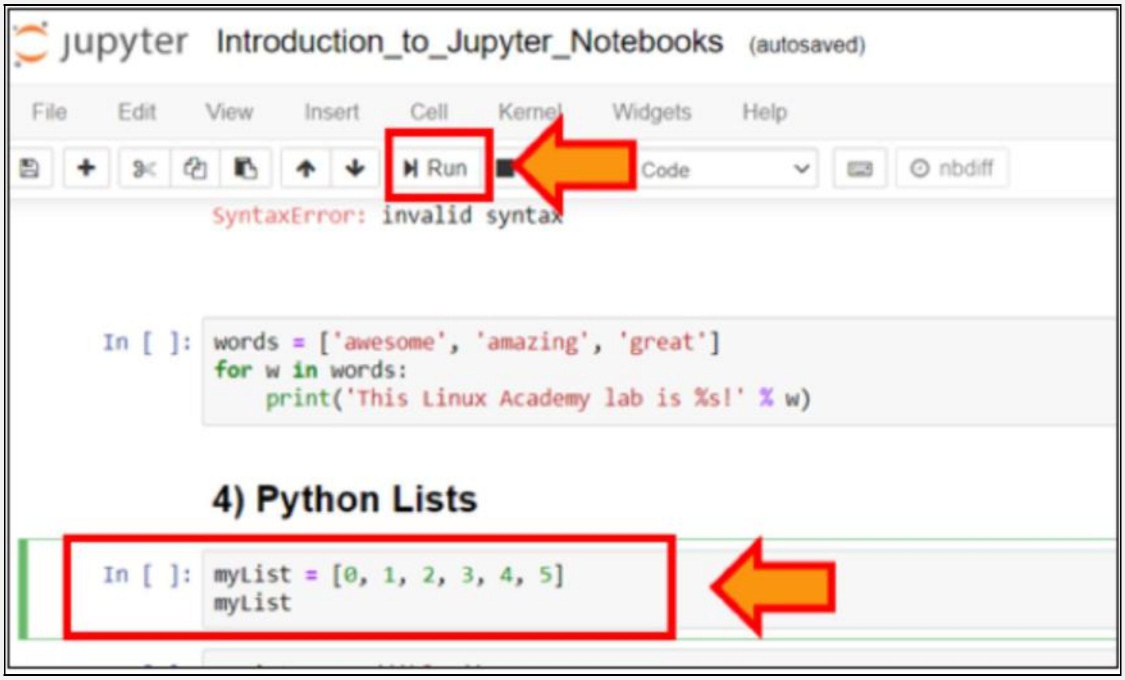
21. Select the cell. Click on **Run**.



22. Go through the output result.



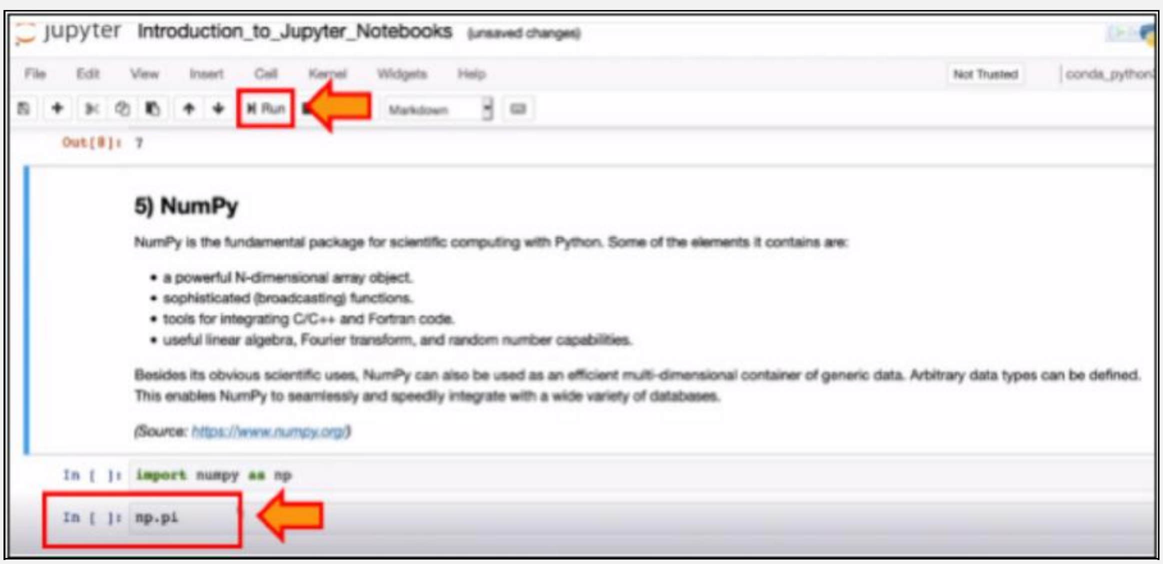
23. Select the cell. Click on **Run**.



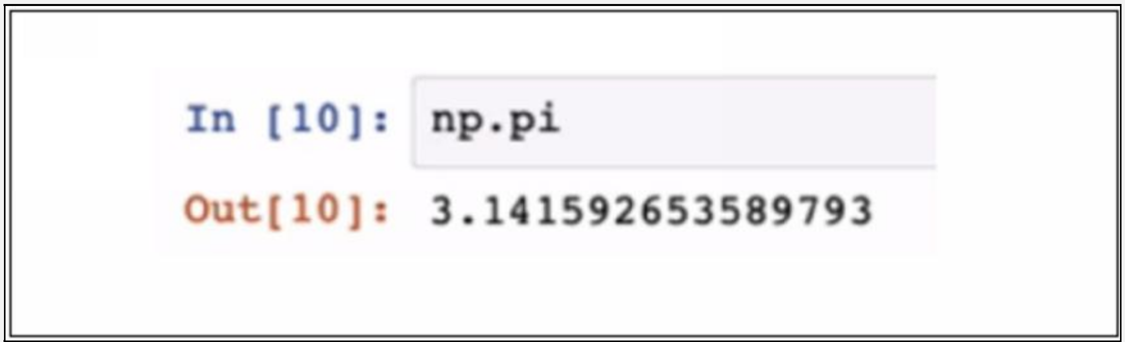
24. Go through the output result.

25. Similarly, select each cell and click on **Run** and see the output.

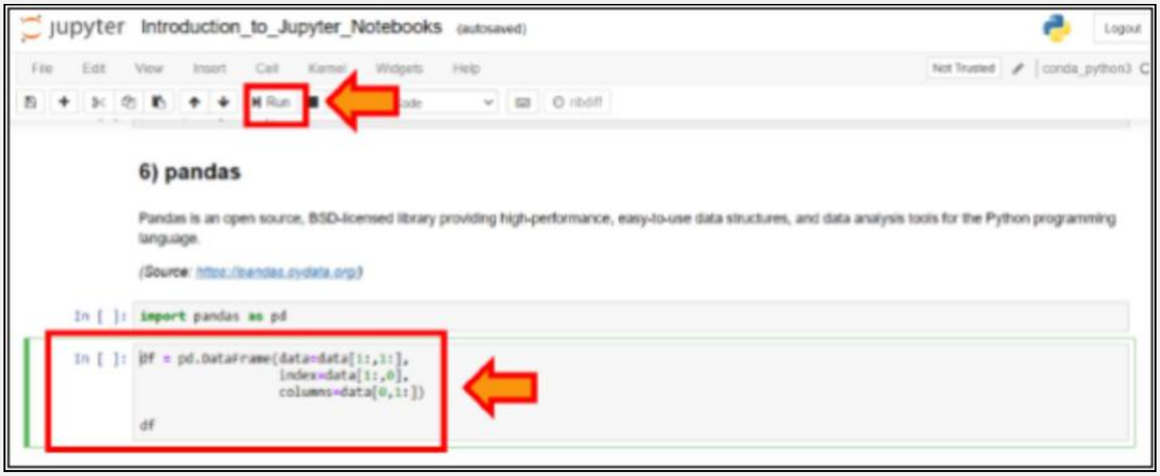
26. Select the cell. Click on **Run**.



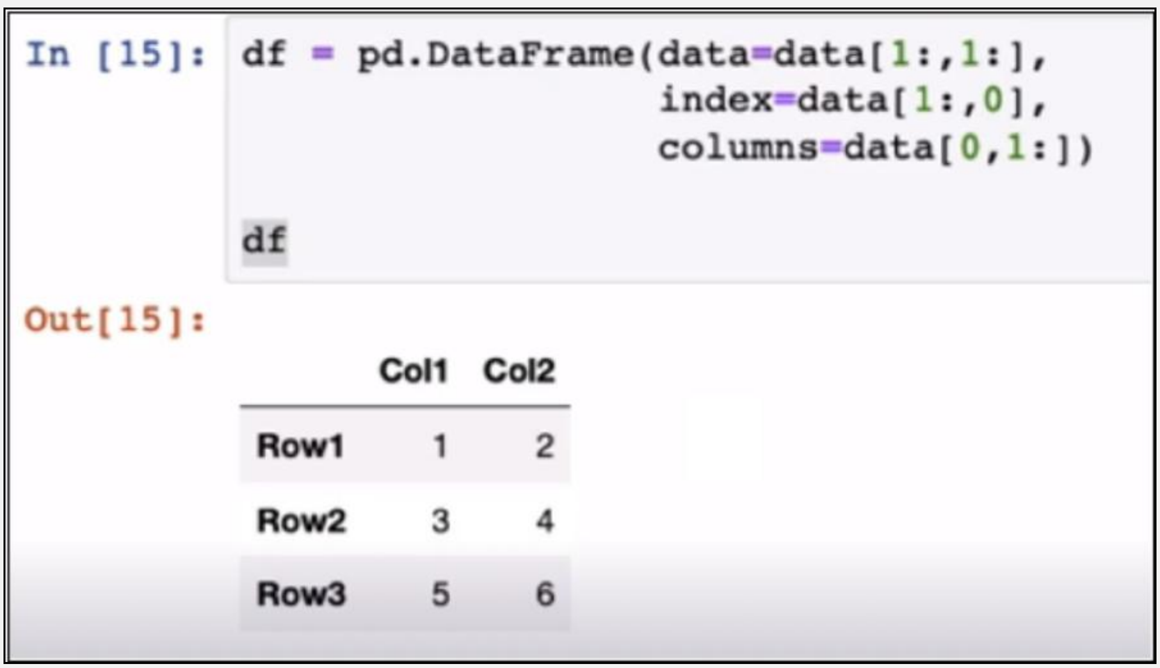
27. Go through the output result.



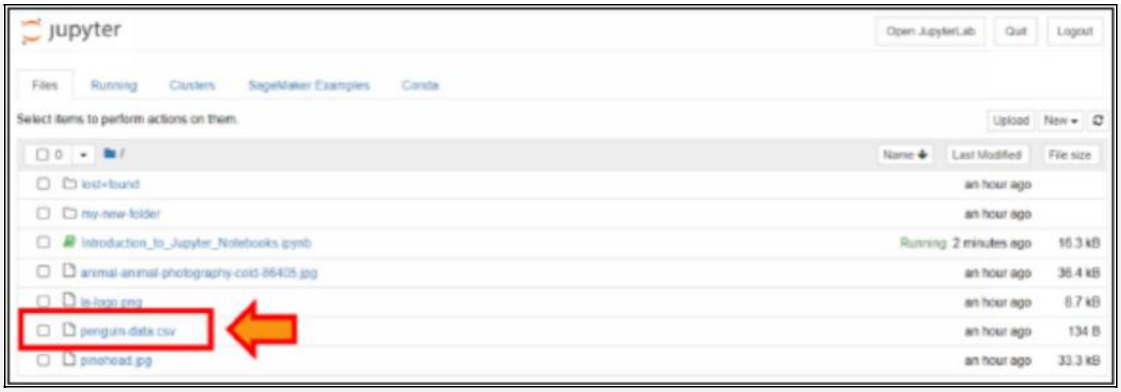
28. Select the cell. Click on **Run.**

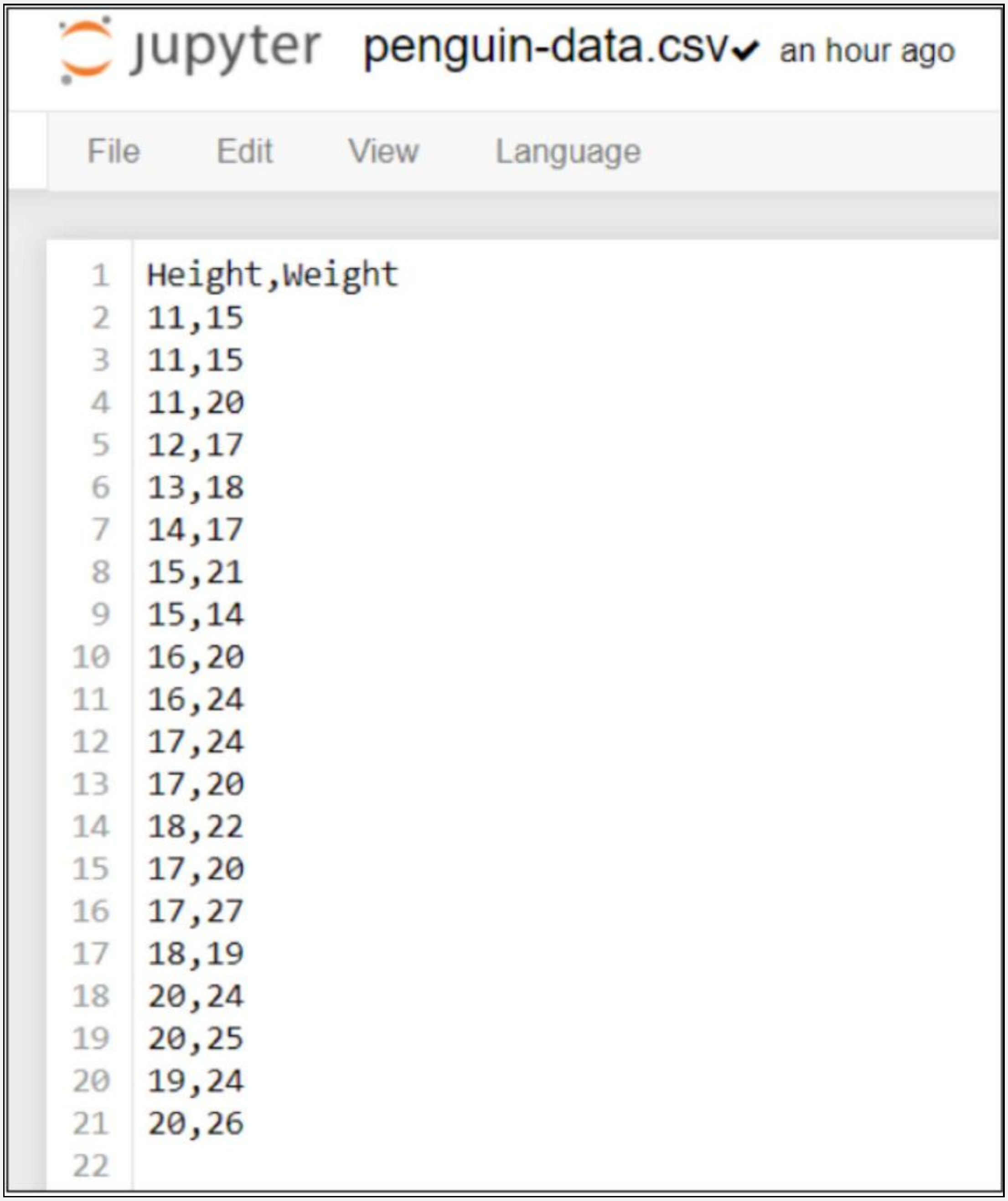


29. Go through the output result.

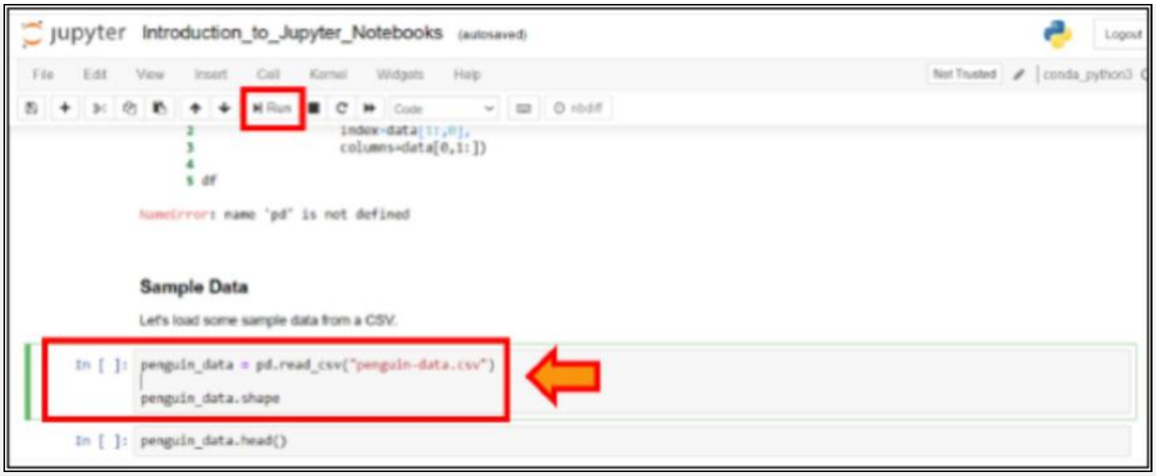


30. Go through penguin-data.csv

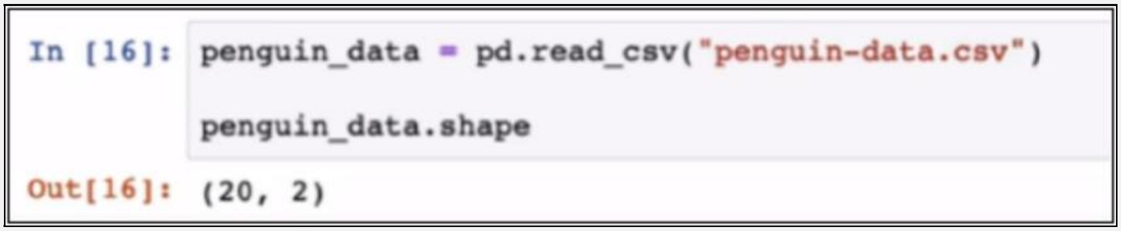




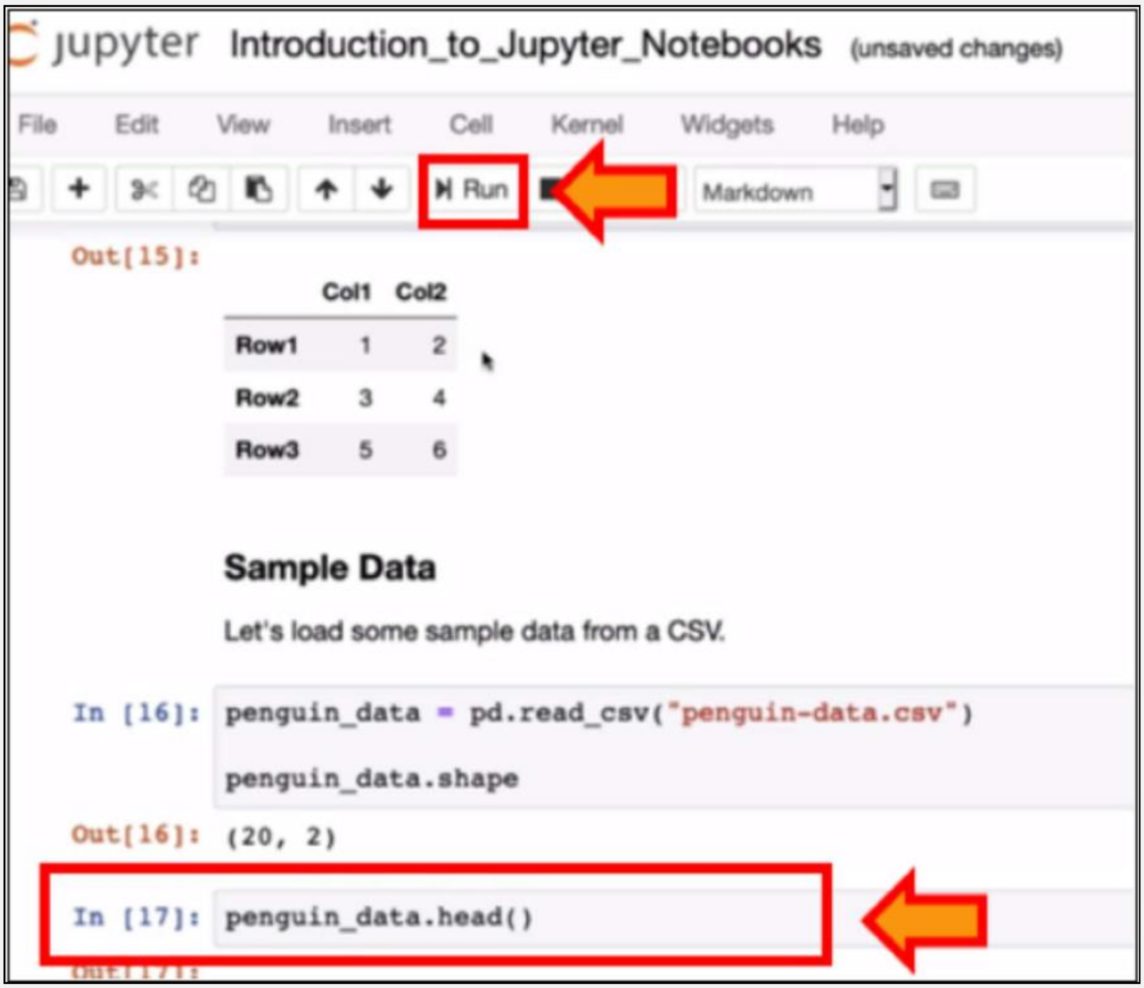
31. Select the cell. Click on **Run.**



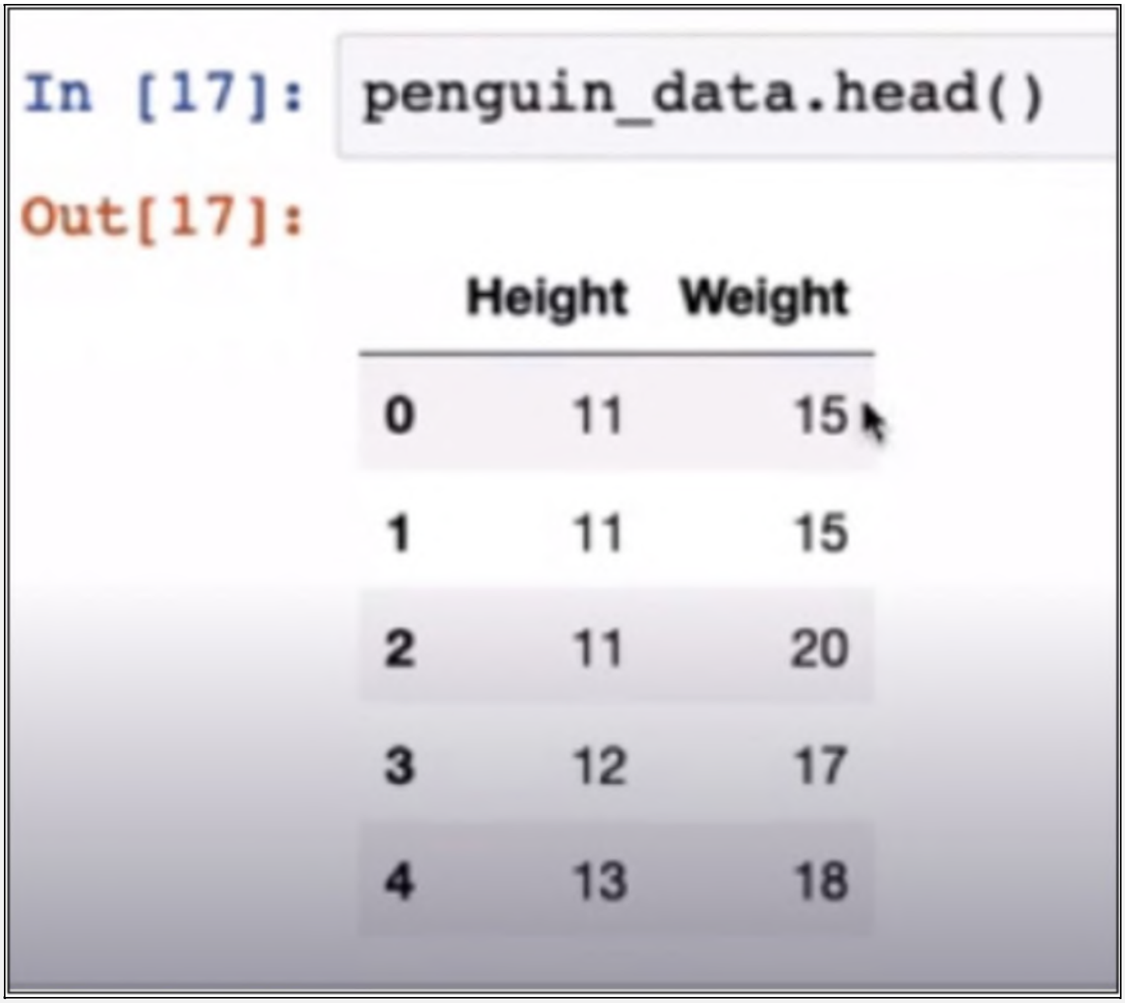
32. Go through the output result.



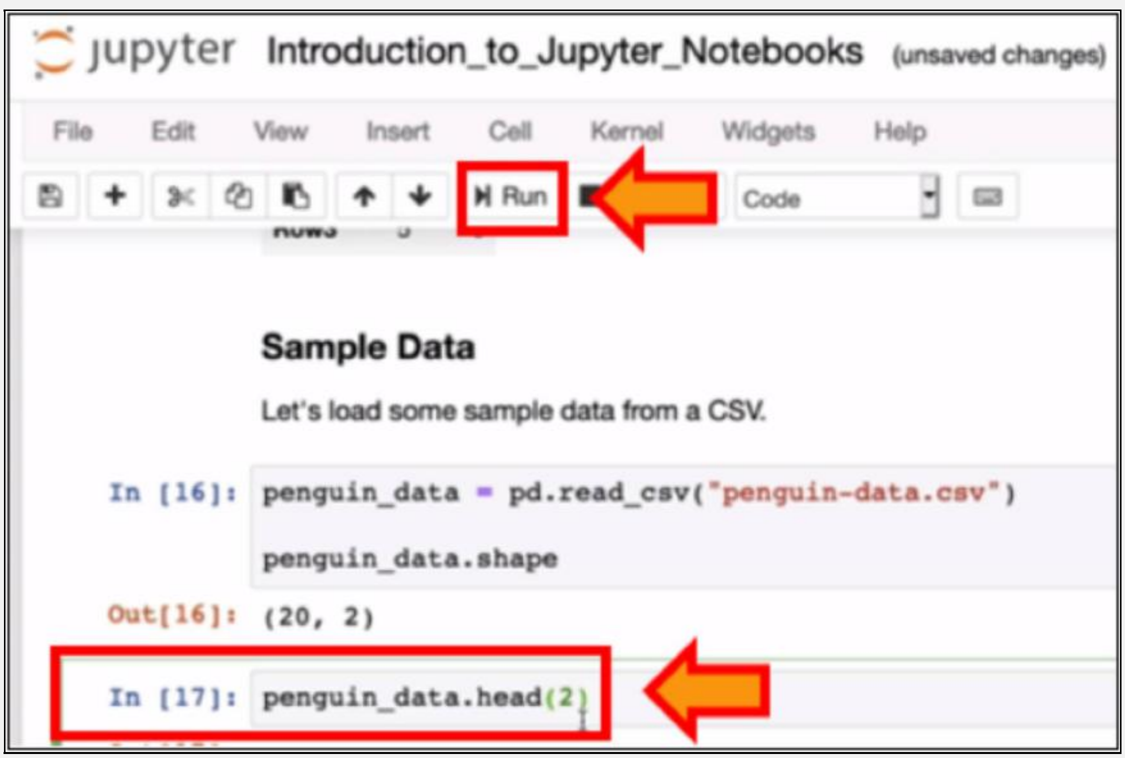
33. Select the cell. Click on **Run.**



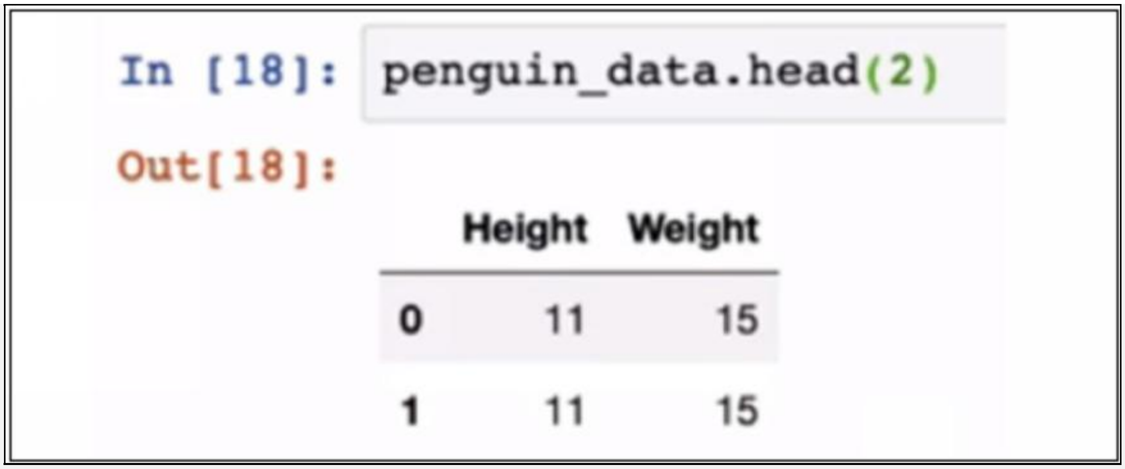
34. Go through the output result.



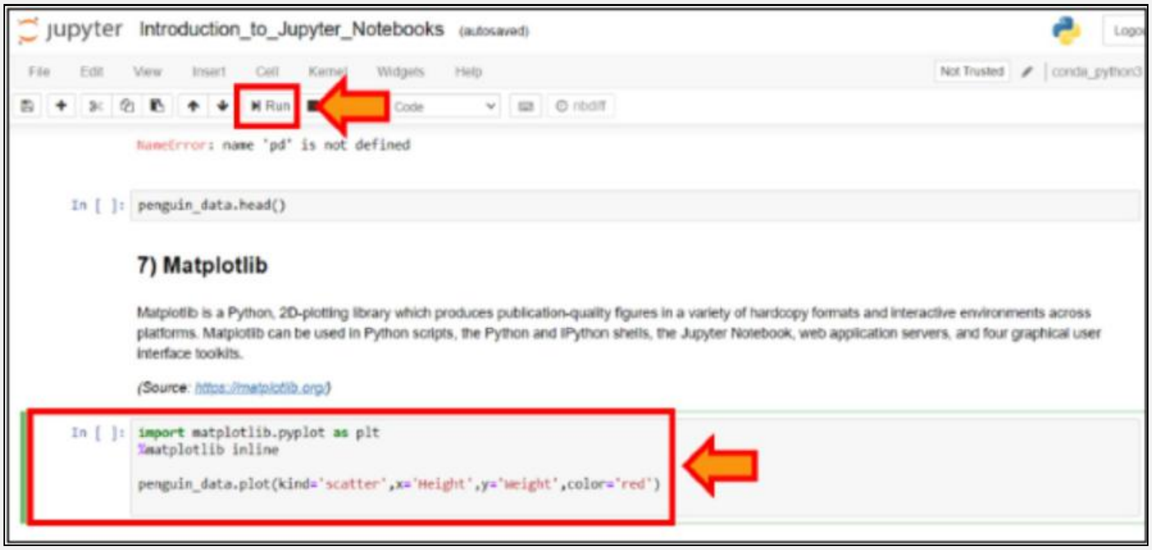
35. Select the cell. Click on **Run.**



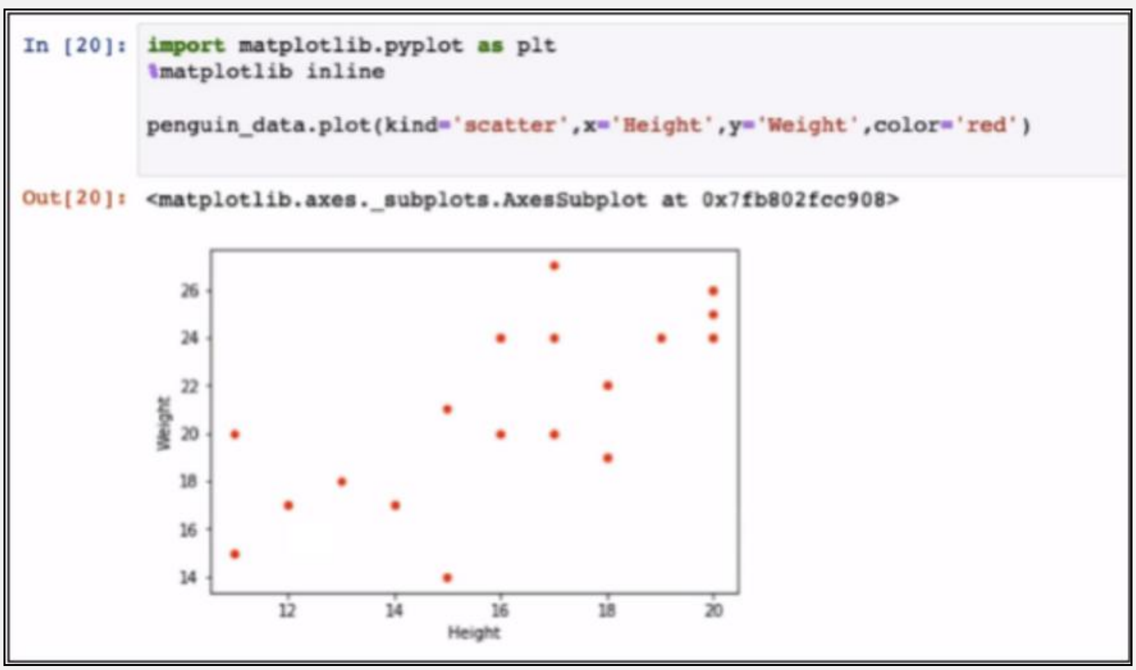
36. Go through the output result.



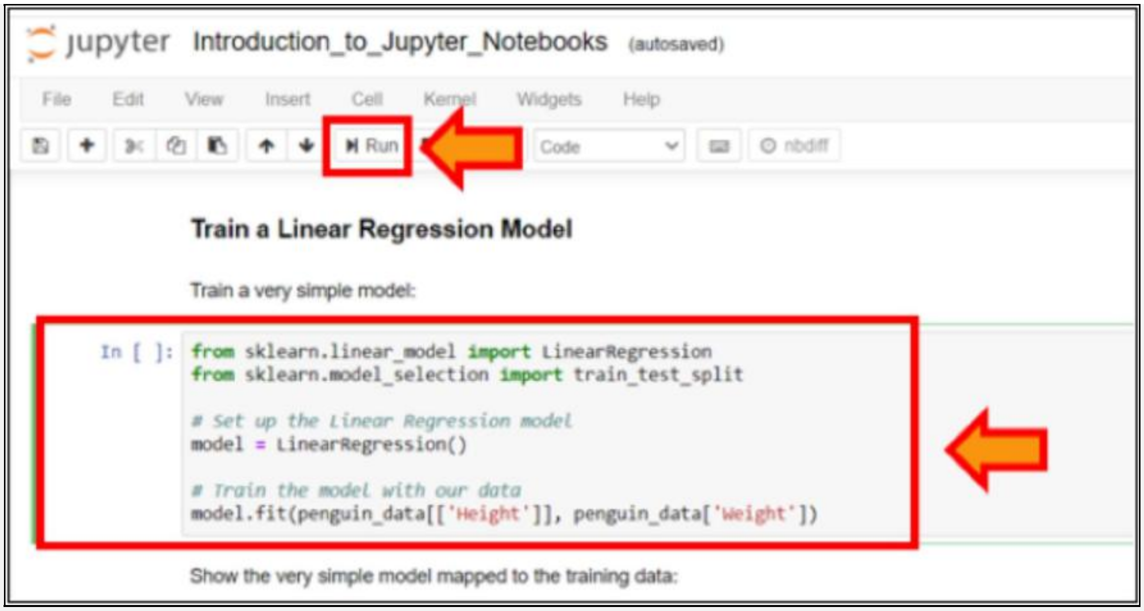
37. Select the cell. Click on **Run.**



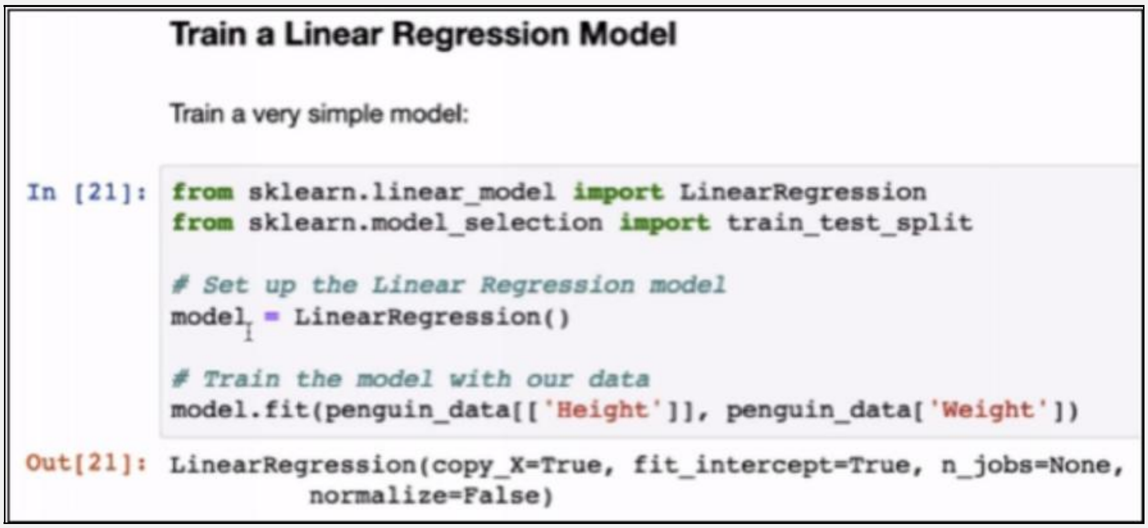
38. Go through the output result.



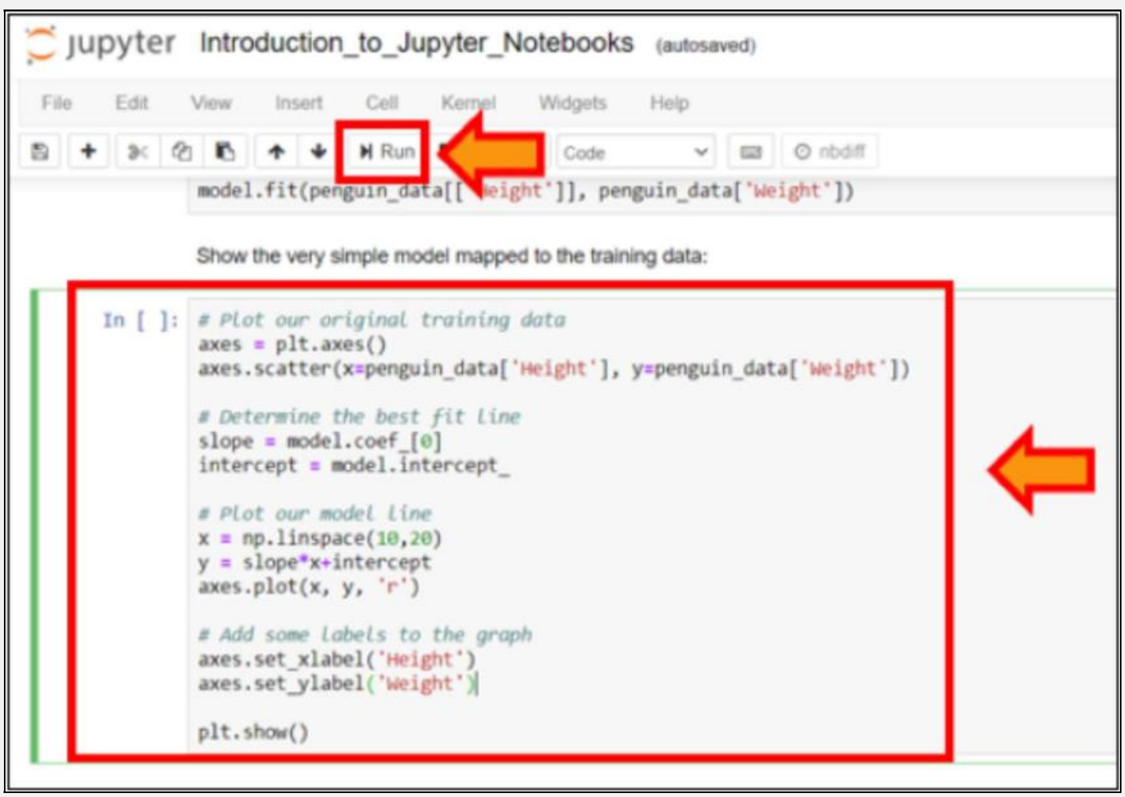
39. Select the cell. Click on **Run.**



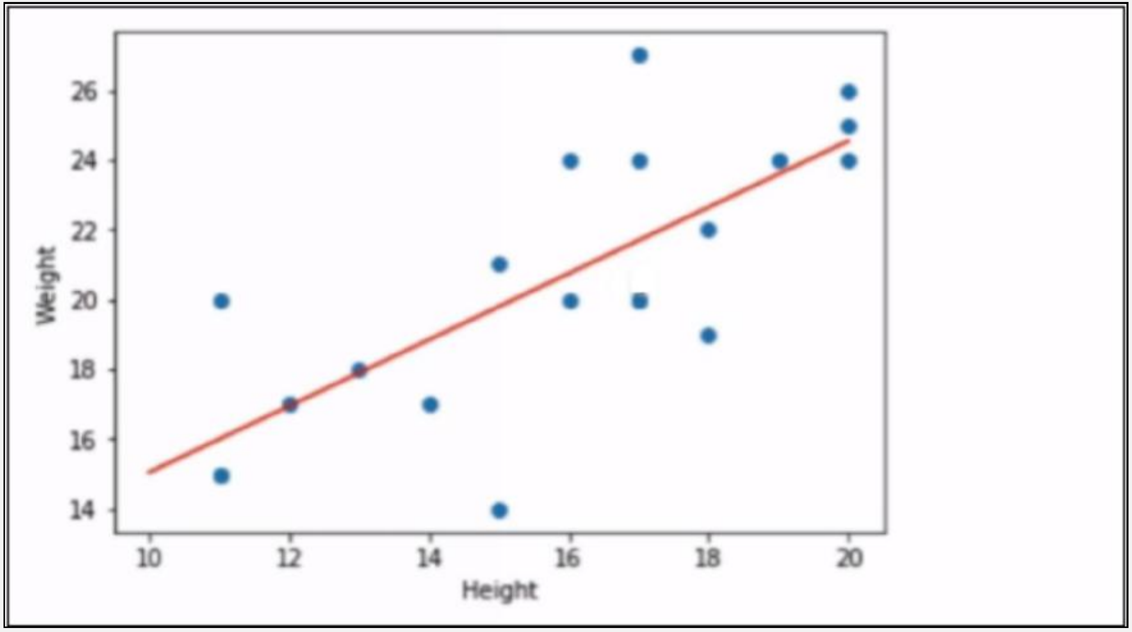
40. Go through the output result.



41. Select the cell. Click on **Run.**

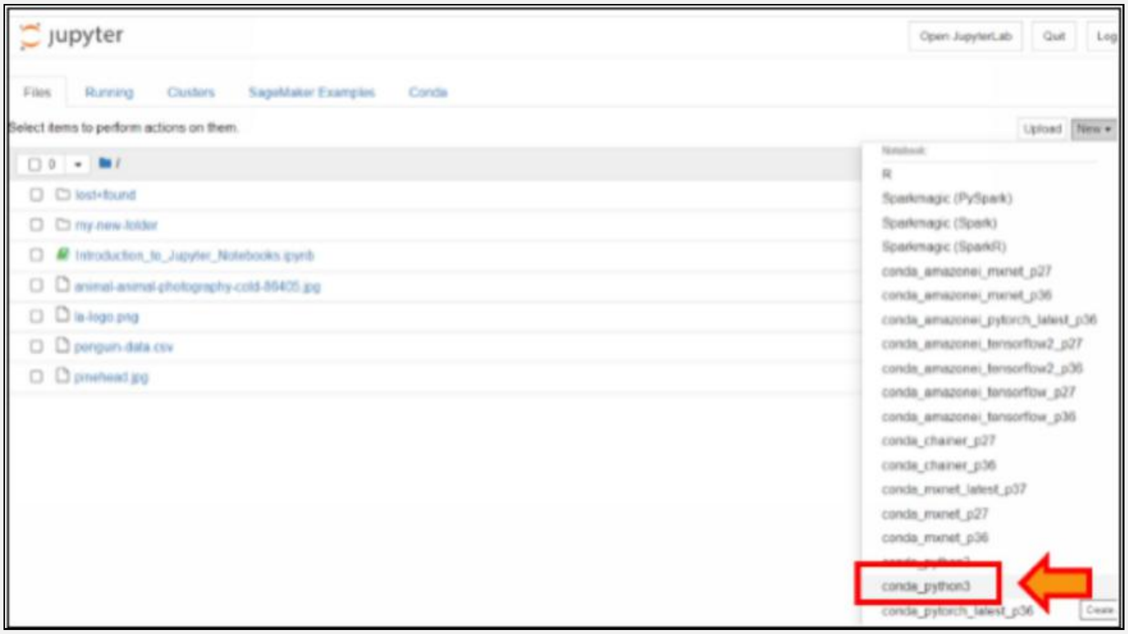


42. Go through the output result.



Step 4: Build Your Own Notebook

43. Click on **conda_python3.**



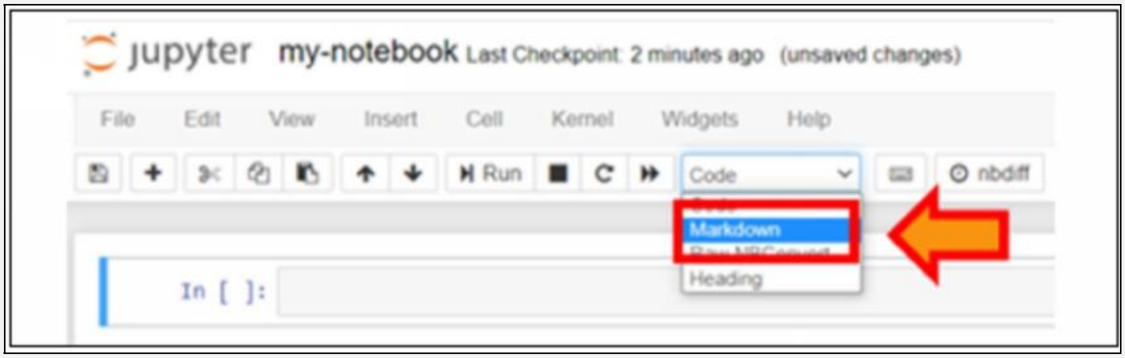
44. Click on **Untitled**.

45. Define notebook name.

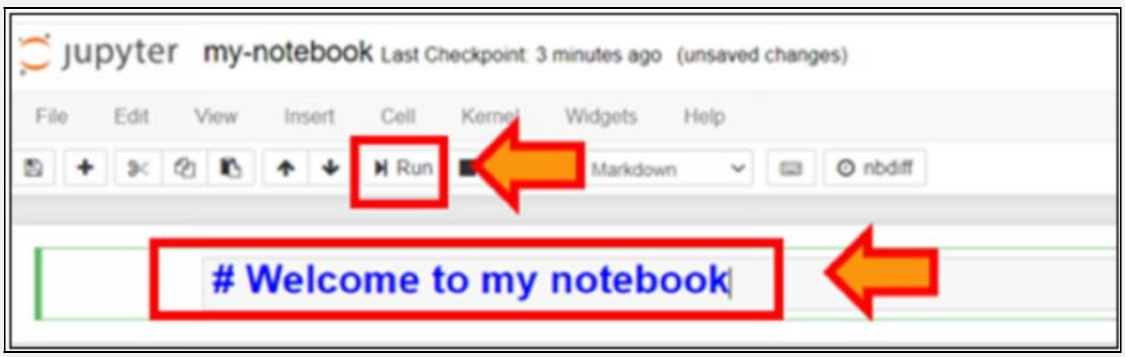
46. Click on **Rename**.

Step 5: Use Markdown to Add Richly Formatted Text to a Notebook

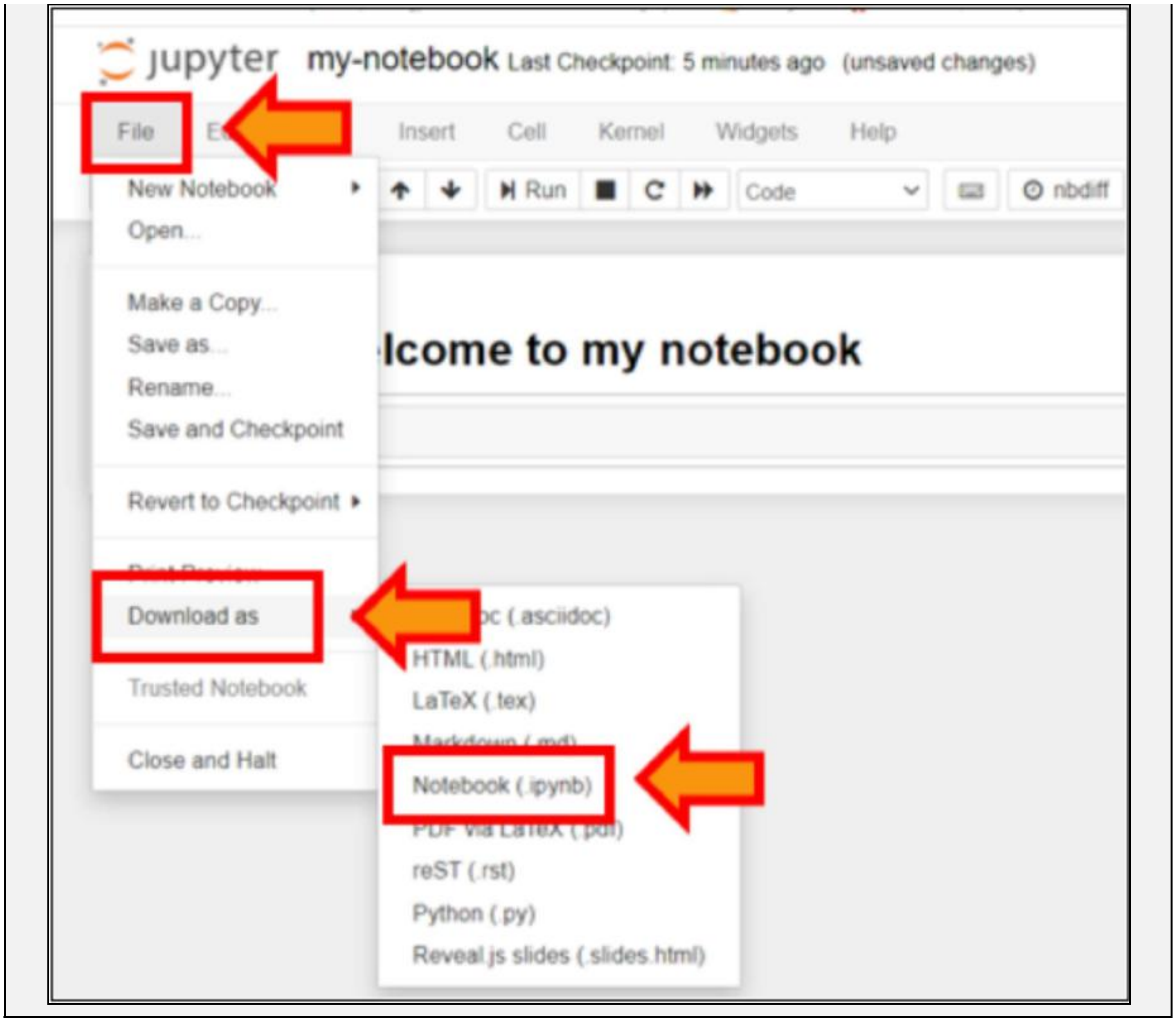
47. Click on **Markdown.**



48. Add **#Welcome to my notebook** in the cell.



49. Click on **File**.
50. Click on **Download as**.
51. Click on **Notebook (.ipynb).**



Lab 09: Algorithms

Certifications	AWS Machine Learning - Specialty
Cloud Provider	AWS
Time Required	1 Hr.
Tags	S3, SageMaker, AWS

Lab Prerequisites

- Familiarity with basic AWS Cloud Computing concepts and terminology
- An AWS account with an active subscription

Service Introduction

XGBoost

The XGBoost (eXtreme Gradient Boosting) version of the gradient boosted trees method is a popular and efficient open-source implementation. Gradient boosting is a supervised learning approach for effectively predicting a target variable by combining an ensemble of estimates from a series of simpler and weaker models. Because of its strong handling of a range of data kinds, relationships, and distributions, as well as the diversity of hyperparameters that can be fine-tuned, the XGBoost method does well in machine learning contests. XGBoost may be used to solve issues involving regression, classification (binary and multi-class), and ranking.

Linear Learner Algorithm

Linear models have supervised learning algorithms for regression, binary classification, or multi-class classification problems. You give the model labels (x,y) with x being a high dimensional vector and y numeric. The algorithm learns a linear function, or, for classification problems, a linear threshold function and maps a vector x to an approximation of label y. Both classification and regression problems may be solved with Amazon SageMaker's Linear Learner algorithm.

Case Study Enterprise Financial Service– Kasasa

Background

Community banks and credit unions use Kasasa's decades-old, cutting-edge financial technology and marketing services to help people feel proud of their money. The business's main office is in Austin, Texas.

Business Challenge

You work as an AWS architect in Kasasa. Kasasa wants to build a model that can predict the legitimacy of a UFO sighting based on information supplied by the submitter.

Proposed Solution

We can use the ground truth dataset that has been collected by the organization previously. The organization has previously determined whether a sighting could be explained (as a hoax or other natural explanation), unexplained or probable.

We will build a model to determine whether newly reported UFO sightings are legitimate or not (explained, unexplained or probable). The organization requires the model to be at least 90% accurate. Hence, we will choose an algorithm, prepare and transform our data and determine how accurate our model is.

Our final result is that we need to have a model artifact stored in S3. We also want to present different model validation metrics, like accuracy, recall, precision, and the F1 score. And we would also present the model error rate to show that our model is getting smarter and smarter, and the error rate decreases as we train our model.

We will build a model using the XGBoost algorithm as a multi-classification problem with researchOutcome, which is an attribute in our dataset as the target attribute. Our goal when using this algorithm is to minimize the training and the validation error.

The reason to choose XGBoost is that it is simple to implement and only needs two hyperparameters with 35 optional hyperparameters. We can use XGBoost for different problems, like ranking problems or regression problems, but here we will use it as a classification problem, specifically, a multi-classification problem.

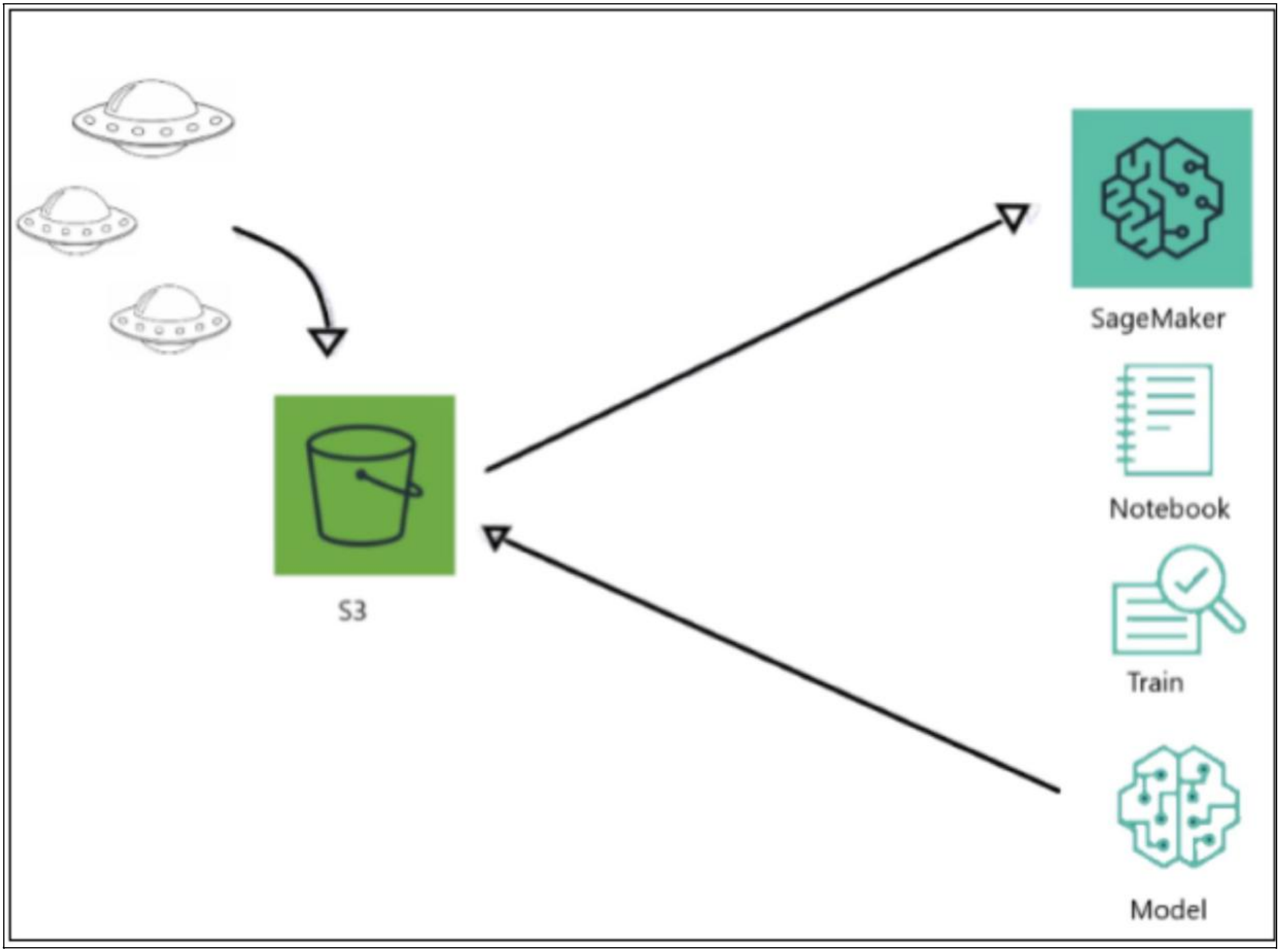
Since we are not exactly sure which attributes are most important, we need to use a classification algorithm over a clustering algorithm because if we choose a clustering algorithm, we need to know which attributes within our dataset will be the most determining factor.

We will employ a different algorithm to categorize the veracity of a UFO sighting. We will build a model using the Linear Learner algorithm as another multi-classification problem with the same attribute as our target attribute, researchOutcome. The goal is to maximize the training accuracy along with other metrics, like precision, F1 score, and recall. Linear Learner is a very flexible algorithm, and we can train different models and determine which attributes from our data are the most determining factors. Another feature of Linear Learner is it also has built-in hyperparameter tuning.

The same dataset that the organization previously gathered will be used. We are going to use this dataset and create a notebook instance. Within this notebook, we will analyze, visualize, prepare, transform, and get our data ready for our machine learning algorithm. In our case, XGBoost and Linear Learner both expect all of our values to be in numeric format, so we will need to encode those values or transform those values into numeric attributes.

Once we do the data transformation, we will divide our dataset into a training set, a validation set, and a testing set. Once we have our training and validation dataset ready, we will then run it through our algorithm, which will create a model and automatically upload that model onto S3. The logs from each of these algorithms from both the XGBoost and the Linear Learner will show us our model's error rate and accuracy.

Lab Diagram



Implementation Steps

1. Upload Data into S3.
2. Import all the necessary libraries.
3. Load the data from Amazon S3.
4. Clean, transform, analyze, and prepare the dataset.
5. Create and Train our model (XGBoost).
6. Create and train our model (Linear Learner).

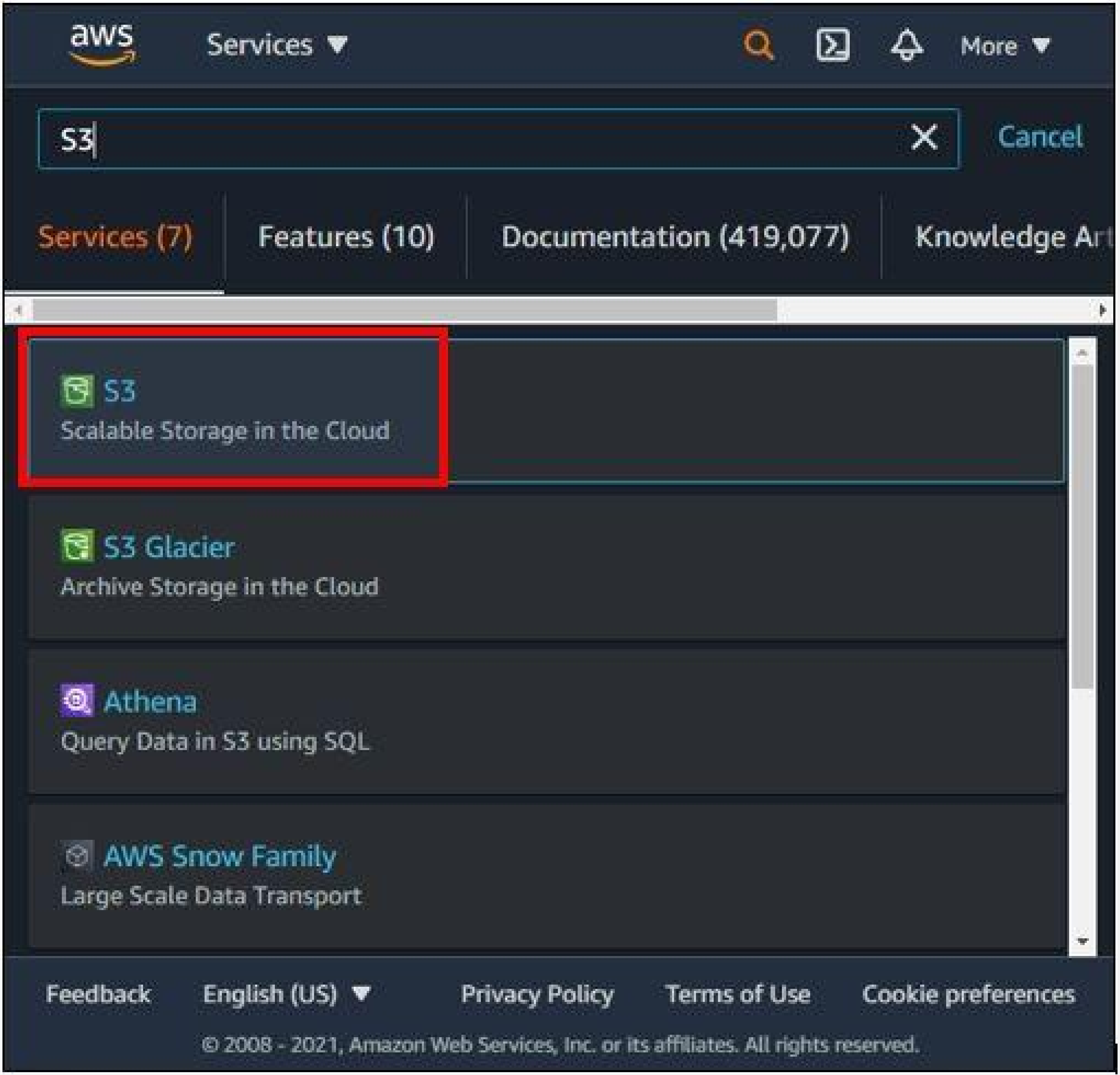
Solution

Note: A lot of trial and error happens during the machine learning process. This is just part of the machine learning process. You will continuously create jobs that fail, and you will figure out the reasons why. You will realize that you do not have enough data, so you will have to get more. You will realize your data may not be in the right format. You might have accidentally left out a hyperparameter or used the wrong value for a hyperparameter. You will realize that your model is extremely inaccurate, so you must change something. You will realize that one algorithm works better than the other. But you need to remember that the machine learning process is a trial-and-error.

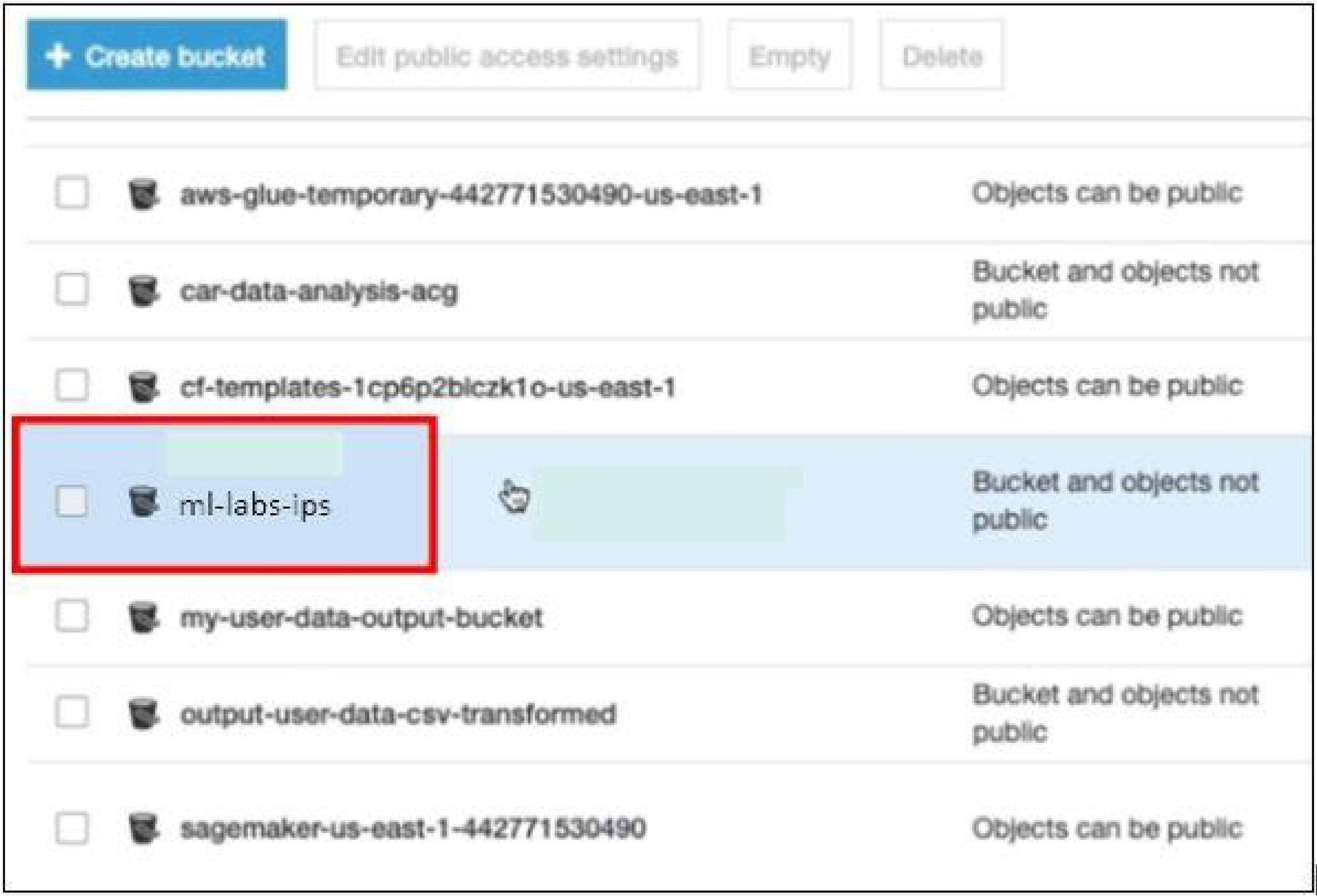
Step 1: Upload Data into S3

1. Log in to the **AWS Console**.

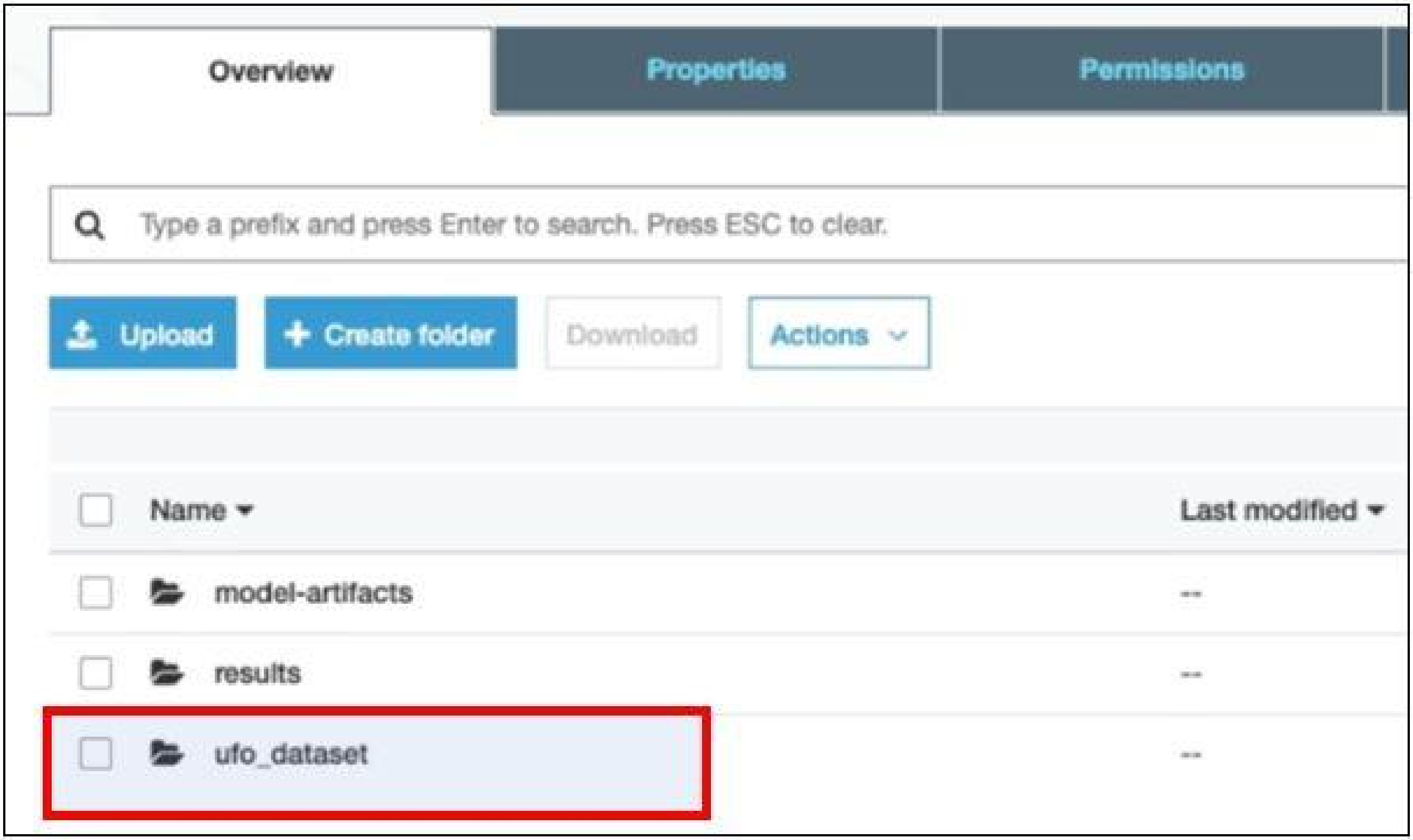
2. Go to **Services** and click on **S3**.



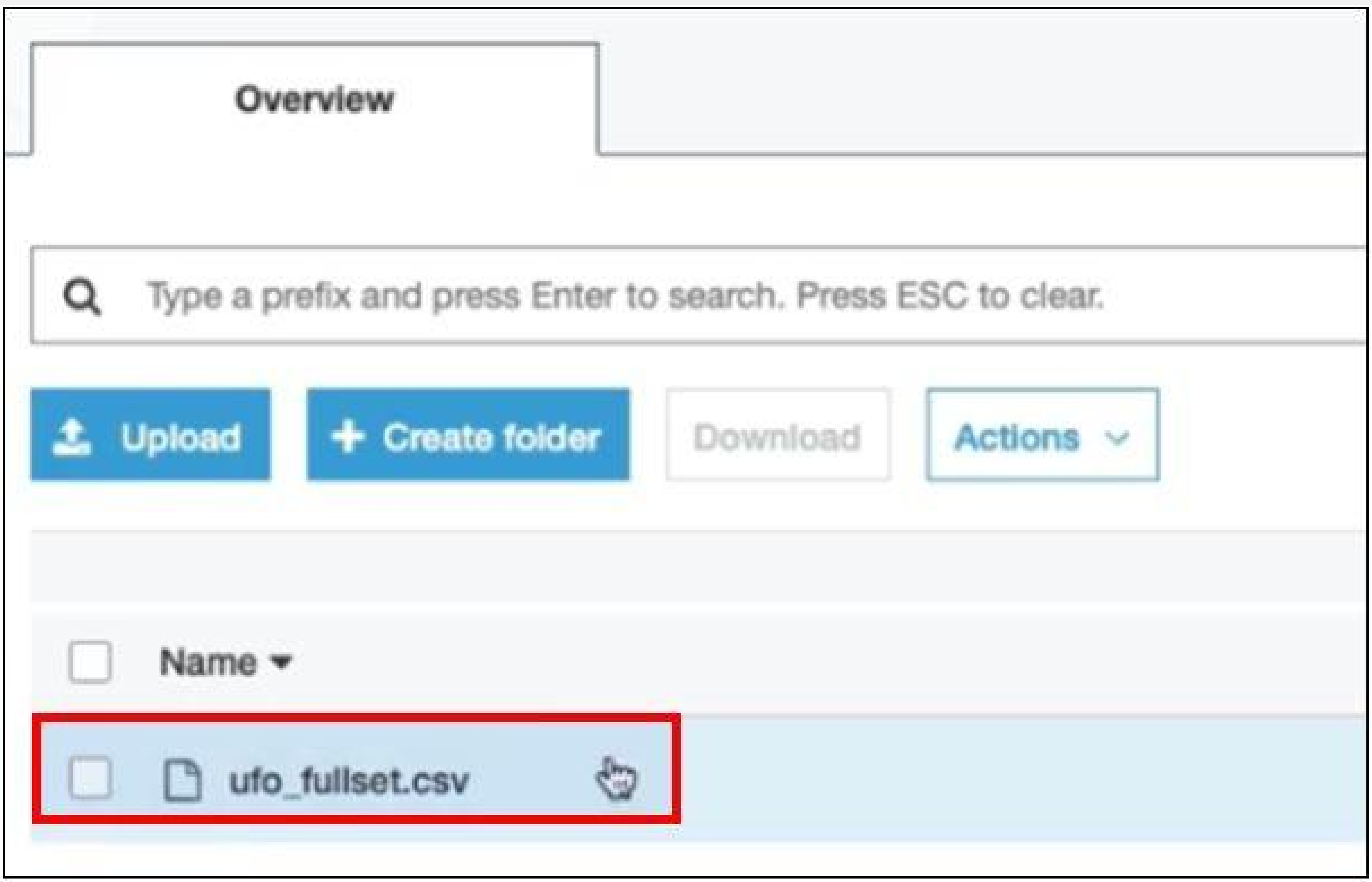
3. Click on the bucket **ml-labs-ips**.



4. Click on the subfolder **ufo_dataset.**



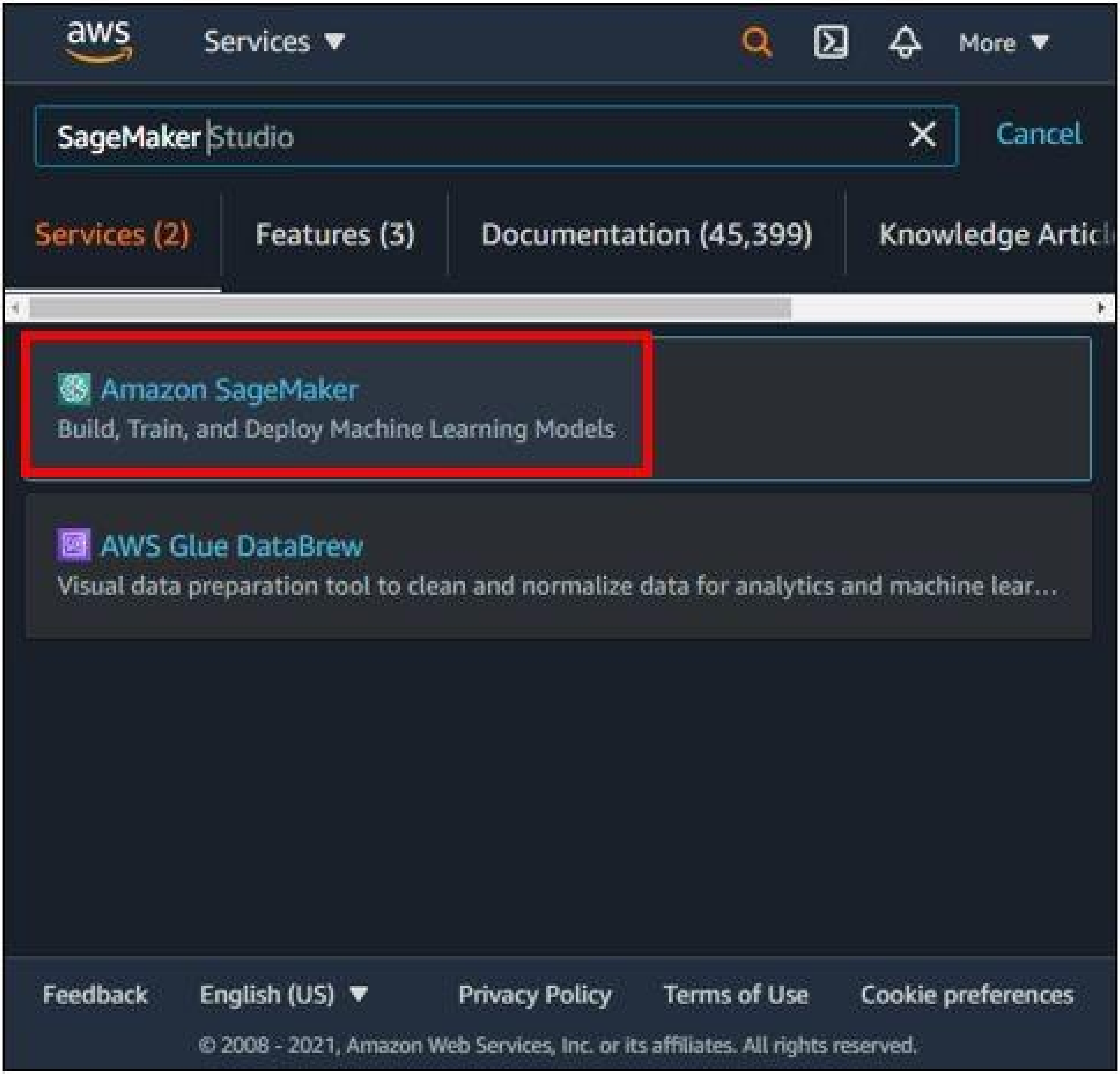
5. Here, you can see the UFO dataset named **uf0_fullset.csv.**



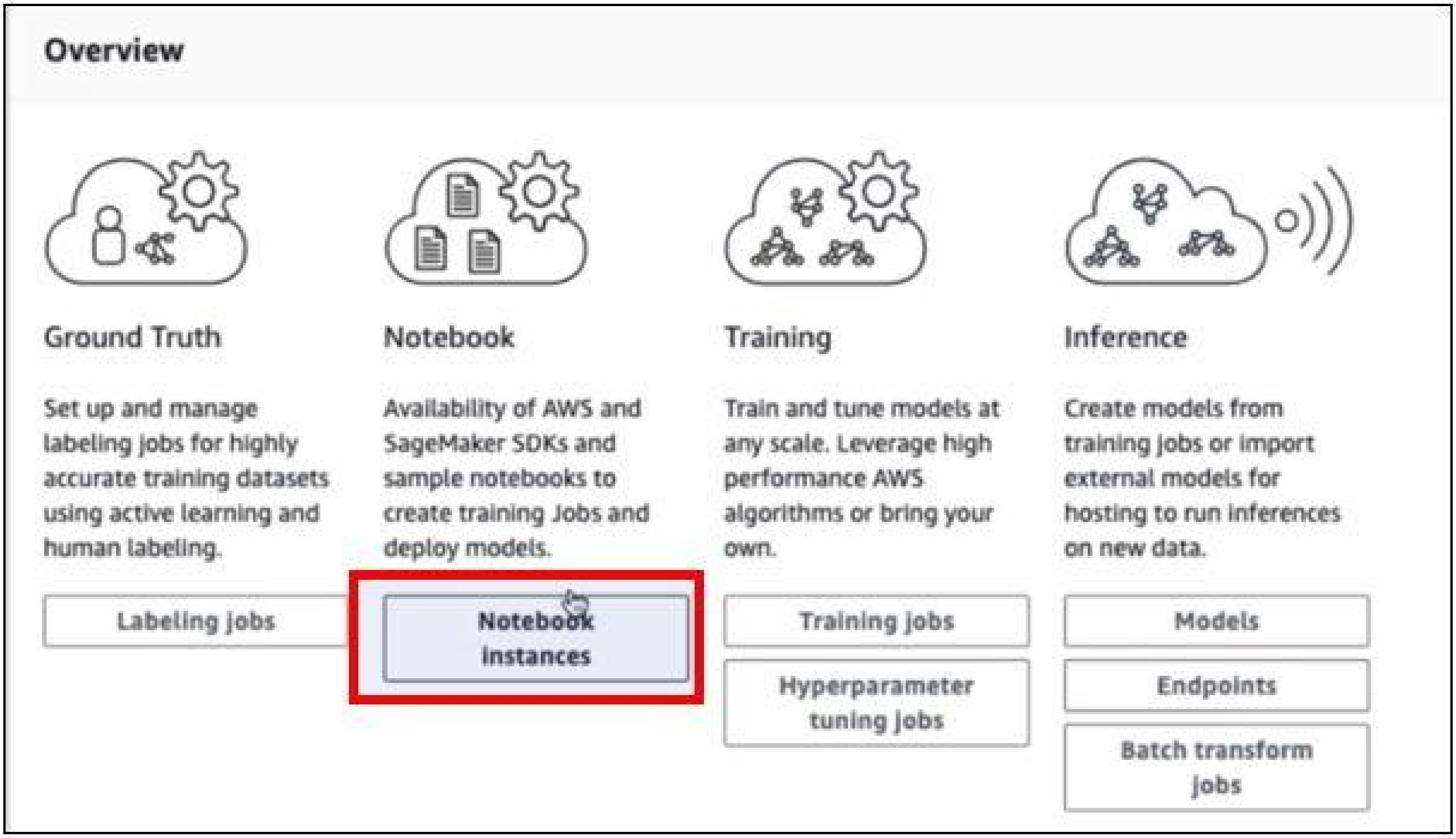
If you do not have this dataset, you can get it from the GitHub link below:

https://github.com/ACloudGuru-Resources/Course_AWS_Certified_Machine_Learning/blob/master/Chapter7/ufo_fullset.csv

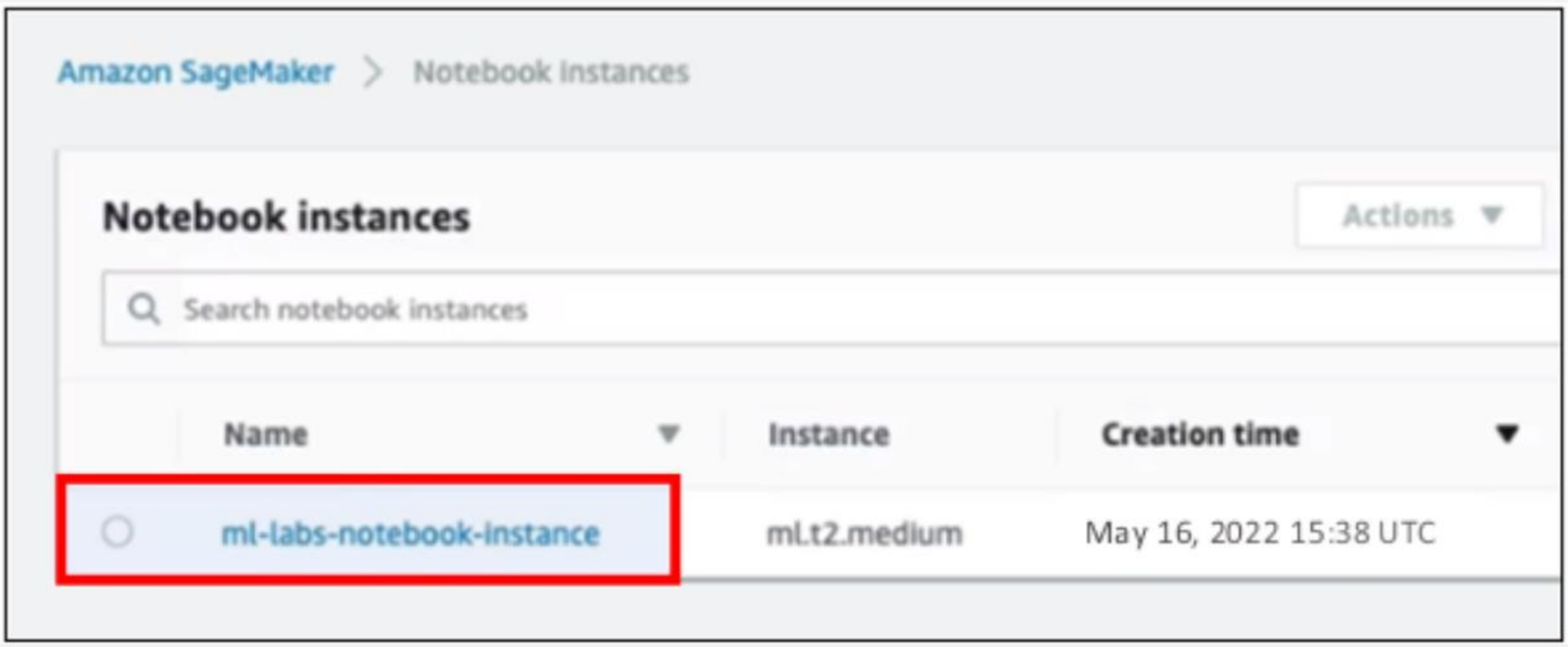
6. Navigate to **SageMaker.**



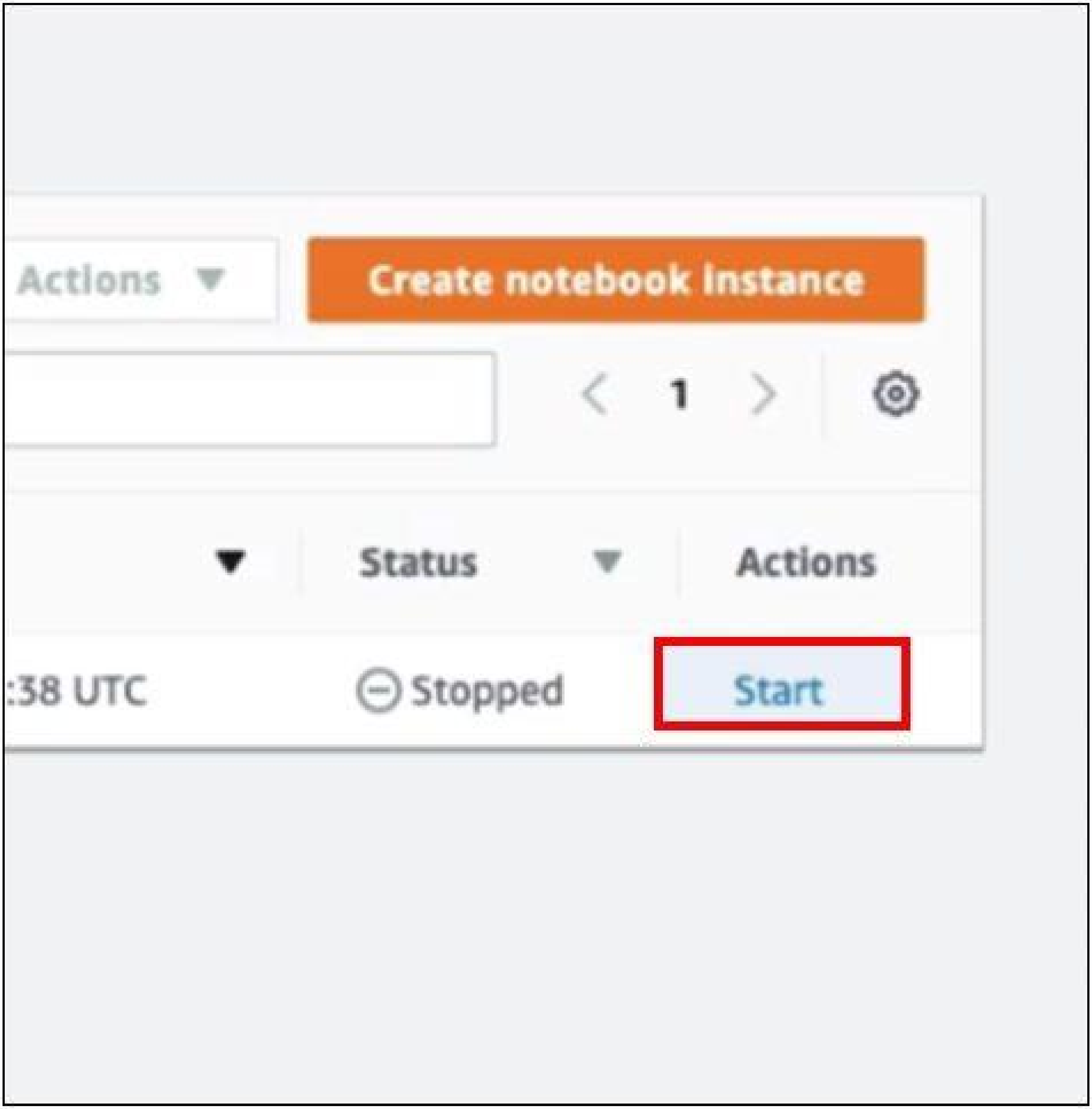
7. Click on **Notebook Instances**.



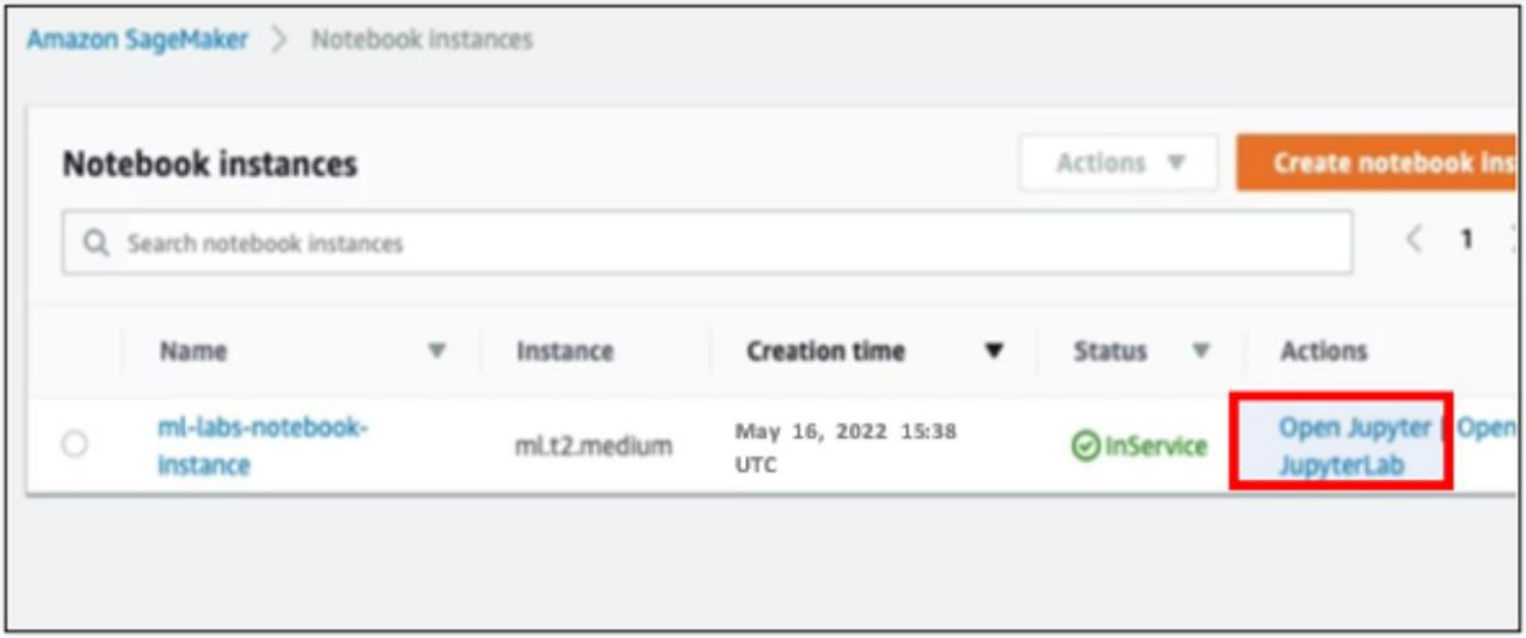
8. Click on **ml-labs-notebook-instance.**

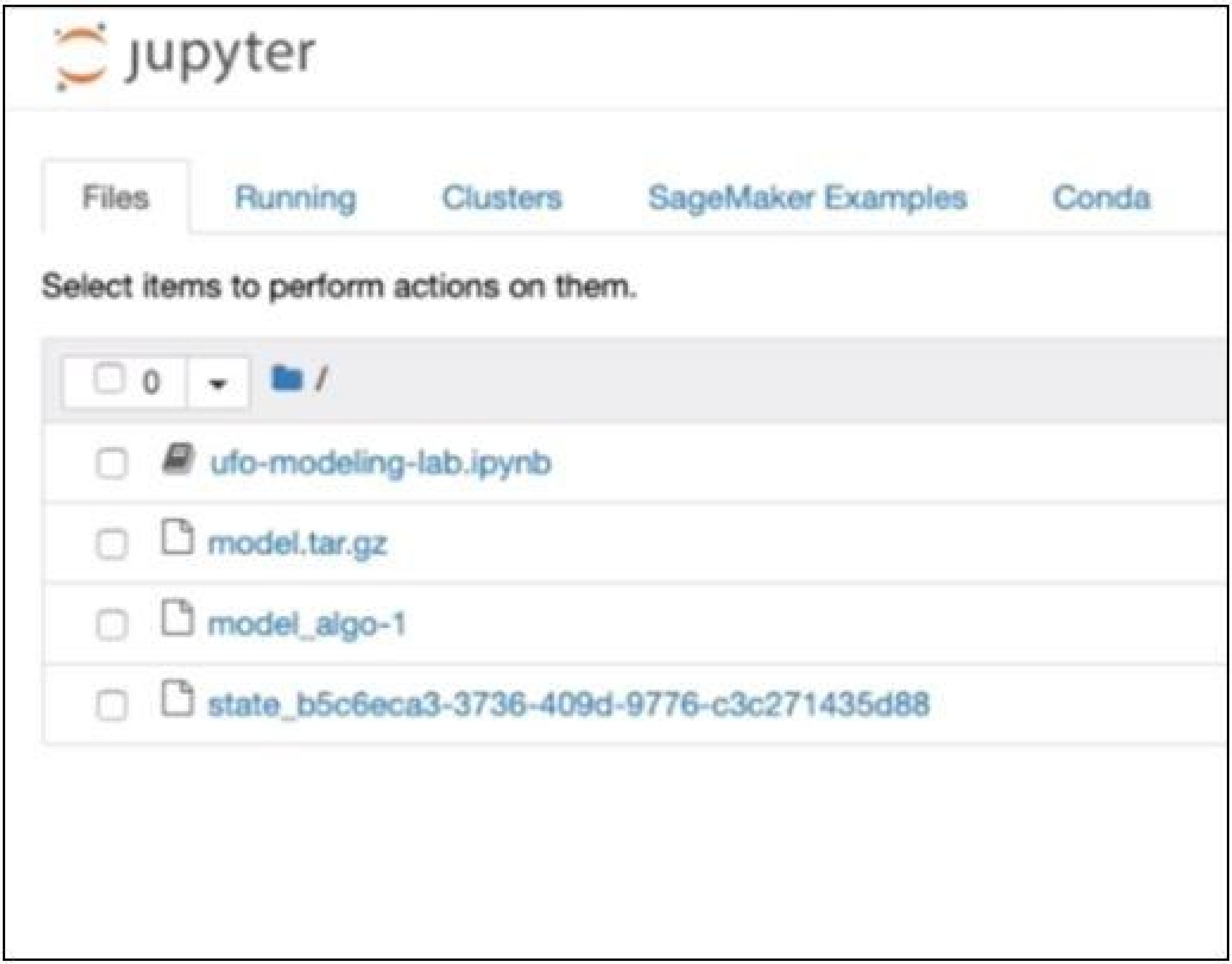


9. Click on **Start**.

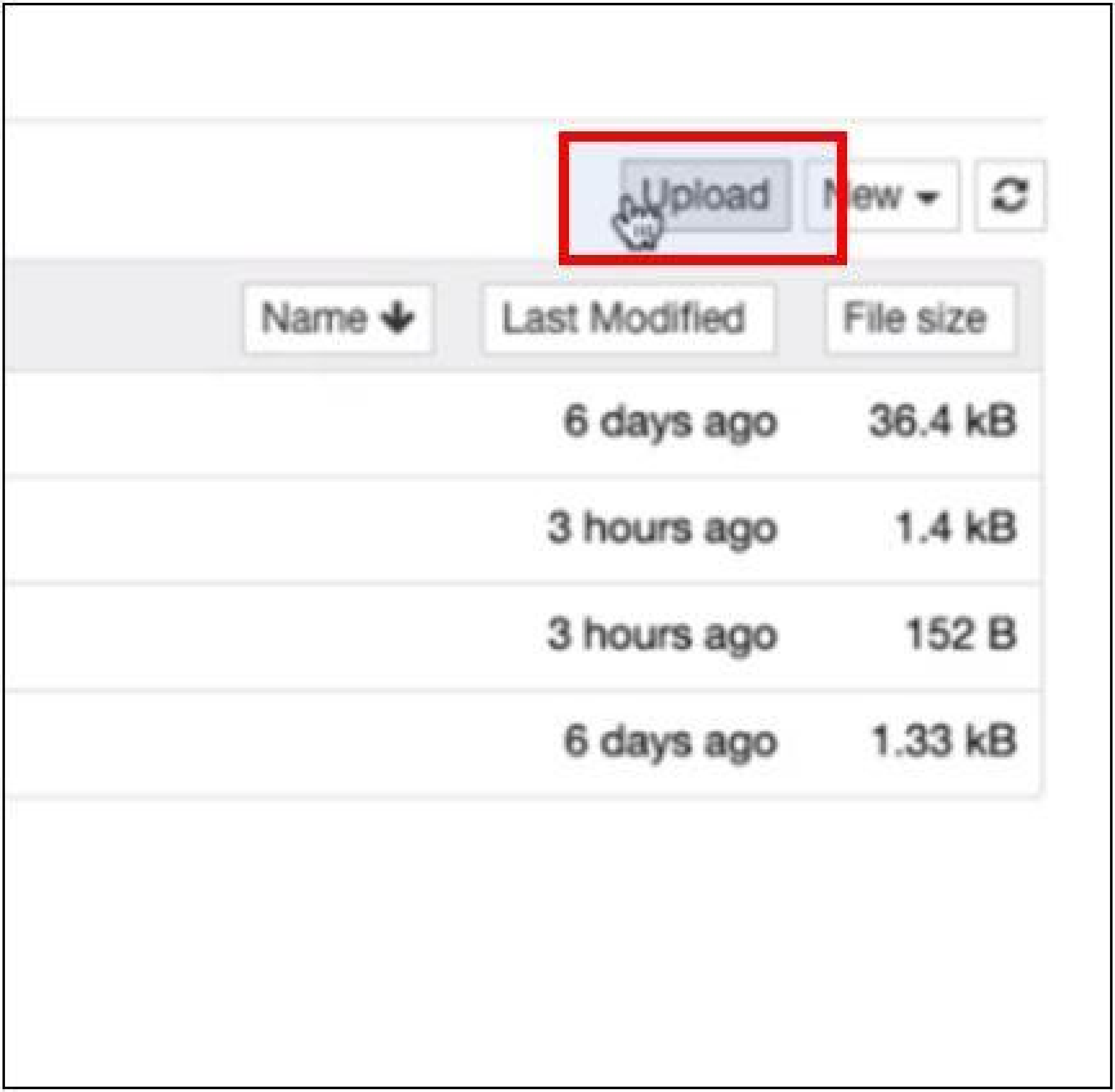


10. Click on **Open Jupyter.**

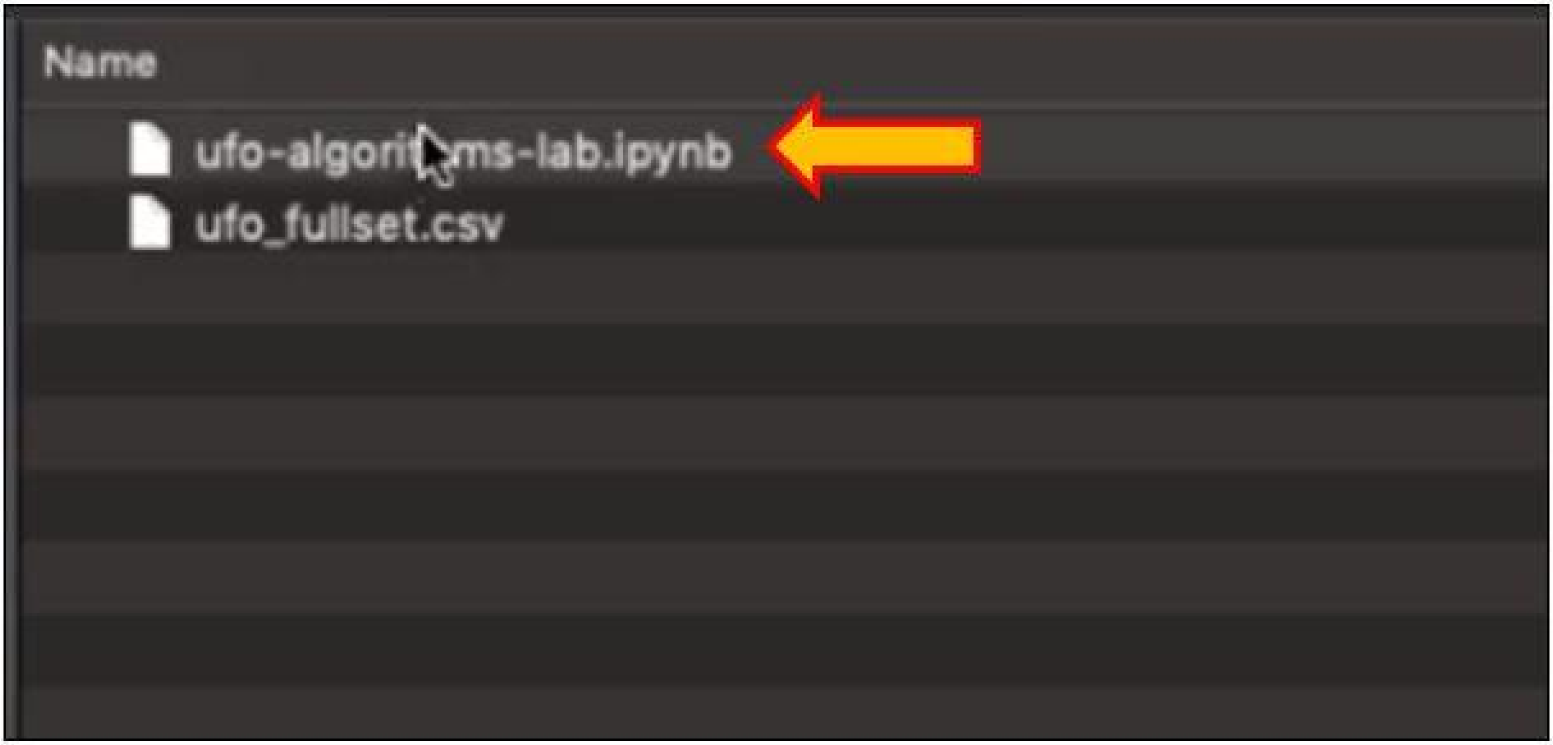




11. Click on **Upload.**



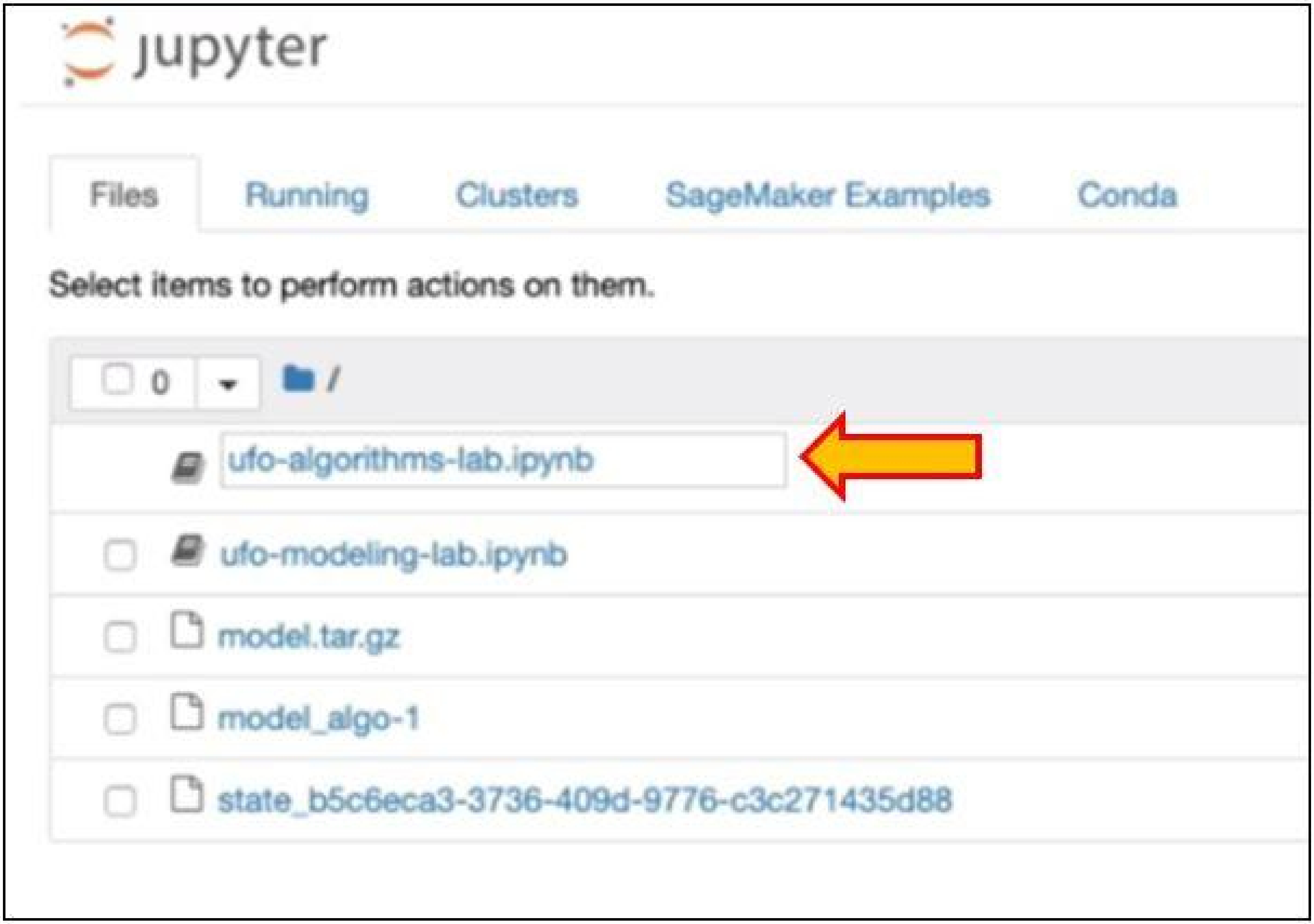
12. Select **ufo-algorithms-lab.ipynb.**



You can also download it from Github from the link given below:

https://github.com/ACloudGuru-Resources/Course_AWS_Certified_Machine_Learning/blob/master/Chapter7/ufo-algorithms-lab.ipynb

13. Navigate into the **Jupyter notebook**.



Step 2: Import all the necessary libraries

1. Use the code given below to import all the necessary libraries.

```
In [ ]: import pandas as pd
        import numpy as np
        from datetime import datetime
        import io
        import sagemaker.amazon.common as smac

        import boto3
        from sagemaker import get_execution_role
        import sagemaker

        import matplotlib.pyplot as plt
        import seaborn as sns
```

We are going to use NumPy and Pandas along with some other libraries. We are also going to use the Python SageMaker libraries that AWS offers. We are also going to use Matplotlib and Seaborn. These Python libraries allow us to build visualizations and graphs to help us analyze our data.

Step 3: Load the data from Amazon S3

We will load our data from S3 into the memory of our notebook instance.

1. Insert your bucket name in the code.

```
In [ ]: role = get_execution_role()
        bucket='<INSERT_BUCKET_NAME_HERE>'
        sub_folder = 'ufo_dataset'
        data_key = 'ufo_fullset.csv'
        data_location = 's3://{}/{}/{}'.format(bucket, sub_folder, data_key)

        df = pd.read_csv(data_location, low_memory=False)
        df.head()
```

2. Name your bucket, such as **ml-labs-ips**.

```
In [ ]: role = get_execution_role()
        bucket='ml-labs-acg'
        sub_folder = 'ufo_dataset'
        data_key = 'ufo_fullset.csv'
        data_location = 's3://{}/{}/{}'.format(bucket, sub_folder, data_key)

        df = pd.read_csv(data_location, low_memory=False)
        df.head()
```

3. Once we run this, it will download the dataset from S3 and load it into the memory of our notebook instance. We will store that into a data frame and look at what that data frame or our dataset looks like.

Out[2]:

	reportedTimestamp	eventDate	eventTime	shape	duration	witnesses	weather	firstName	lastName	latitude	longitude	sighting	physicalEvidence
0	1977-04-04T04:02:23.340Z	1977-03-31	23:46	circle	4	1	rain	Ila	Bashirian	47.329444	-122.578889	Y	N
1	1982-11-22T02:06:32.019Z	1982-11-15	22:04	disk	4	1	partly cloudy	Eriberto	Runolfsson	52.664913	-1.034894	Y	Y
2	1992-12-07T19:06:52.482Z	1992-12-07	19:01	circle	49	1	clear	Miller	Watsica	38.951667	-92.333889	Y	N
3	2011-02-24T21:06:34.898Z	2011-02-21	20:56	disk	13	1	partly cloudy	Clifton	Bechtelar	41.496944	-71.367778	Y	N
4	1991-03-09T16:18:45.501Z	1991-03-09	11:42	circle	17	1	mostly cloudy	Jayda	Ebert	47.606389	-122.330833	Y	N

4. We are trying to solve the problem of determining what the researchOutcome is.

Note: All of this is explained, but there are explained, unexplained, and probable options available.

Out[2]:

ventDate	eventTime	shape	duration	witnesses	weather	firstName	lastName	latitude	longitude	sighting	physicalEvidence	contact	researchOutcome
1977-03-31	23:46	circle	4	1	rain	Ila	Bashirian	47.329444	-122.578889	Y	N	N	explained
1982-11-15	22:04	disk	4	1	partly cloudy	Eriberto	Runolfsson	52.664913	-1.034894	Y	Y	N	explained
1992-12-07	19:01	circle	49	1	clear	Miller	Watsica	38.951667	-92.333889	Y	N	N	explained
2011-02-21	20:56	disk	13	1	partly cloudy	Clifton	Bechtelar	41.496944	-71.367778	Y	N	N	explained
1991-03-09	11:42	circle	17	1	mostly cloudy	Jayda	Ebert	47.606389	-122.330833	Y	N	N	explained

Step 4: Clean, transform, analyze, and prepare the dataset

1. We need to clean, transform, analyze, and start preparing our dataset. We need to check to see any missing values within our dataset. If there are missing values, then it will display those, and we can look at what we might need to do to fix our missing values.

```
In [ ]: # Let's check to see if there are any missing values
        missing_values = df.isnull().values.any()
        if(missing_values):
            display(df[df.isnull().any(axis=1)])

In [ ]: df['shape'].value_counts()

In [ ]: # Replace the missing values with the most common shape
        df['shape'] = df['shape'].fillna(df['shape'].value_counts().index[0])
```

2. When we run the code, we get some missing values. If we look, the shape for a couple of these observations is null or not a number.

```
In [3]: # Let's check to see if there are any missing values
        missing_values = df.isnull().values.any()
        if(missing_values):
            display(df[df.isnull().any(axis=1)])
```

	reportedTimestamp	eventDate	eventTime	shape	duration	witnesses	weather	firstName	lastName	latitude	longitude	sighting	ph
1024	2011-03-23T18:32:20.473Z	2011-03-22	21:12	NaN	3	1	rain	Deon	Feil	37.681944	-121.766944	Y	
2048	1998-04-23T18:47:16.029Z	1998-04-23	10:07	NaN	40	2	partly cloudy	Vincenzo	Rohan	38.254167	-85.759444	Y	

3. We will replace those missing values with the most common shape or the most common value. We could do this by getting the count for each shape; hence, we can see that a circle is the most common shape.

```
In [4]: df['shape'].value_counts()

Out[4]: circle      6047
        disk        5920
        light       1699
        square      1662
        triangle    1062
        sphere      1020
        box          200
        oval         199
        pyramid      189
        Name: shape, dtype: int64
```

4. We will replace the missing values with the most common shape; in this case, it will replace those two missing values with the circle shape.

```
In [5]: # Replace the missing values with the most common shape
        df['shape'] = df['shape'].fillna(df['shape'].value_counts().index[0])
```

5. Now, we will start preparing our dataset by transforming some of the values into the correct data types.

We are going to:

- Convert the reportedTimestamp and eventDate to datetime data types
- Convert the shape and weather to a category data type

- Map the physicalEvidence and contact from ‘Y,’ ‘N’ to 0, 1
- Convert the researchOutcome to category data.type (target attribute)

Once we successfully run these, we should be able to see the data types in the new format.

```
In [ ]: df['reportedTimestamp'] = pd.to_datetime(df['reportedTimestamp'])
        df['eventDate'] = pd.to_datetime(df['eventDate'])

        df['shape'] = df['shape'].astype('category')
        df['weather'] = df['weather'].astype('category')

        df['physicalEvidence'] = df['physicalEvidence'].replace({'Y': 1, 'N': 0})
        df['contact'] = df['contact'].replace({'Y': 1, 'N': 0})

        df['researchOutcome'] = df['researchOutcome'].astype('category')

In [ ]: df.dtypes
```

6. If we type **df.dtypes**, we should be able to see the different data types that our attributes are in now.

```
In [7]: df.dtypes

Out[7]: reportedTimestamp    datetime64[ns]
        eventDate            datetime64[ns]
        eventTime                    object
        shape                      category
        duration                      int64
        witnesses                     int64
        weather                    category
        firstName                    object
        lastName                     object
        latitude                    float64
        longitude                   float64
        sighting                     object
        physicalEvidence              int64
        contact                       int64
        researchOutcome            category
        dtype: object
```

7. We can visualize some of our data to find any important information about that data. We might be

able to find things like outliers, or we might be able to find the distribution of our data. We may be able to find any imbalances in our data as well.

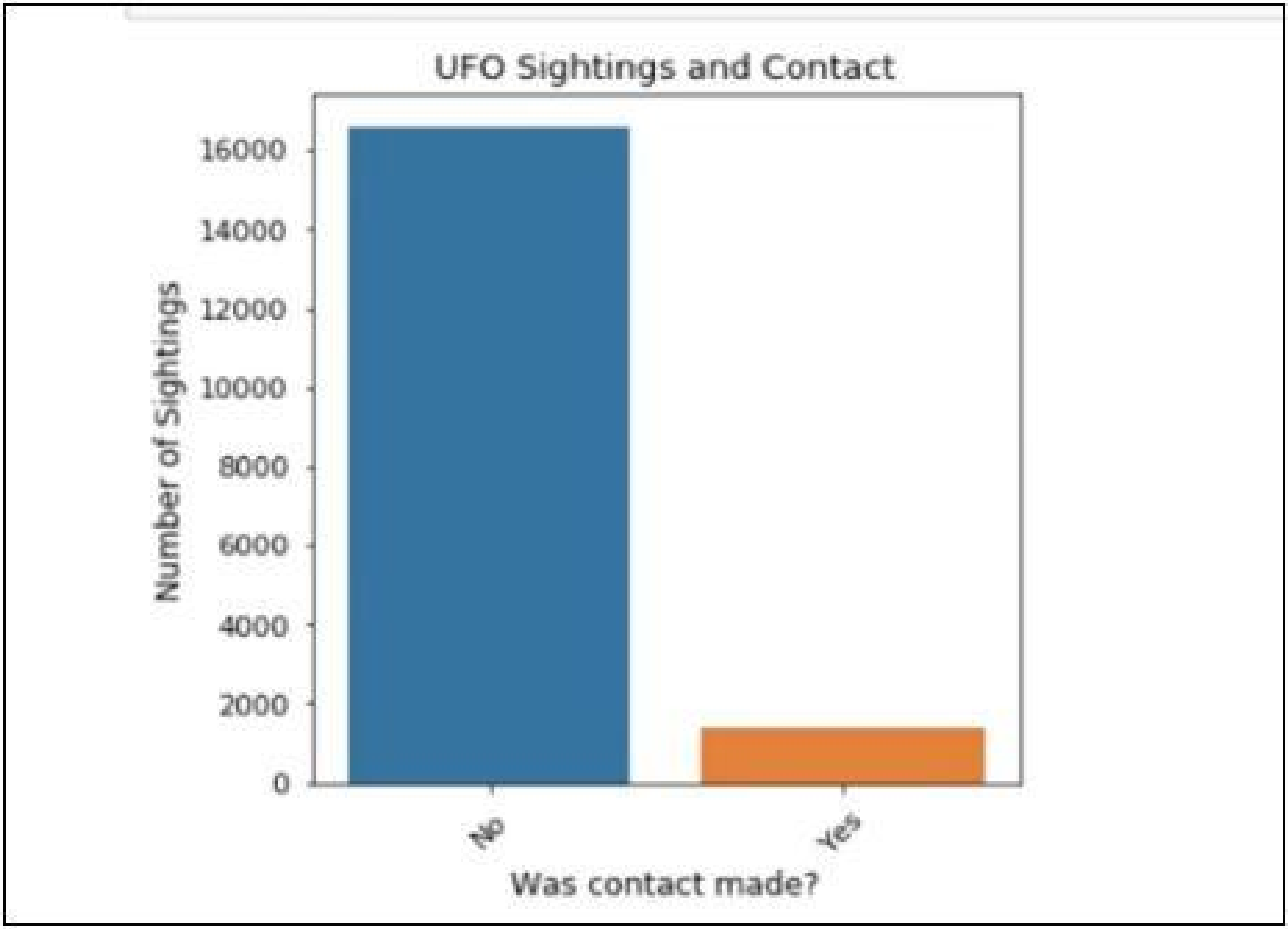
8. We will configure Matplotlib and the Seaborn library, so we can start drawing out some visualizations.

The first visualization is the UFO Sightings and Contact. Was contact made?

```
In [8]: %matplotlib inline
        sns.set_context("paper", font_scale=1.4)
```

```
In [ ]: m_cts = (df['contact'].value_counts())
        m_ctsx = m_cts.index
        m_ctsy = m_cts.get_values()
        f, ax = plt.subplots(figsize=(5,5))

        sns.barplot(x=m_ctsx, y=m_ctsy)
        ax.set_title('UFO Sightings and Contact')
        ax.set_xlabel('Was contact made?')
        ax.set_ylabel('Number of Sightings')
        ax.set_xticklabels(['No', 'Yes'])
        plt.xticks(rotation=45)
        plt.show()
```

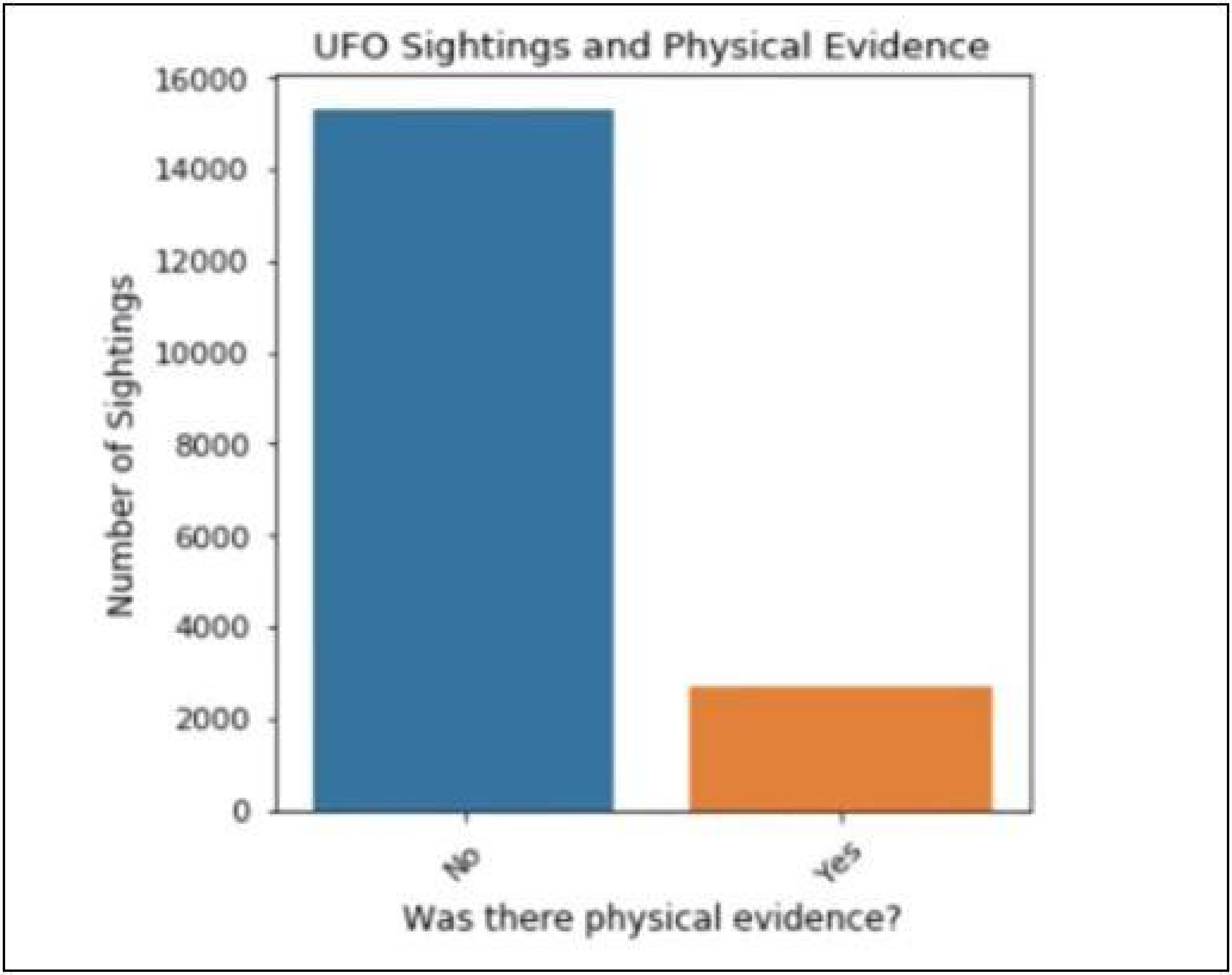
9. We can see that with most of our sightings, no contact was made.



10. Now, to see how many sightings had some physical evidence associated with them.

```
In [ ]: m_cts = (df['physicalEvidence'].value_counts())
        m_ctsx = m_cts.index
        m_ctsy = m_cts.get_values()
        f, ax = plt.subplots(figsize=(5,5))

        sns.barplot(x=m_ctsx, y=m_ctsy)
        ax.set_title('UFO Sightings and Physical Evidence')
        ax.set_xlabel('Was there physical evidence?')
        ax.set_ylabel('Number of Sightings')
        ax.set_xticklabels(['No', 'Yes'])
        plt.xticks(rotation=45)
        plt.show()
```

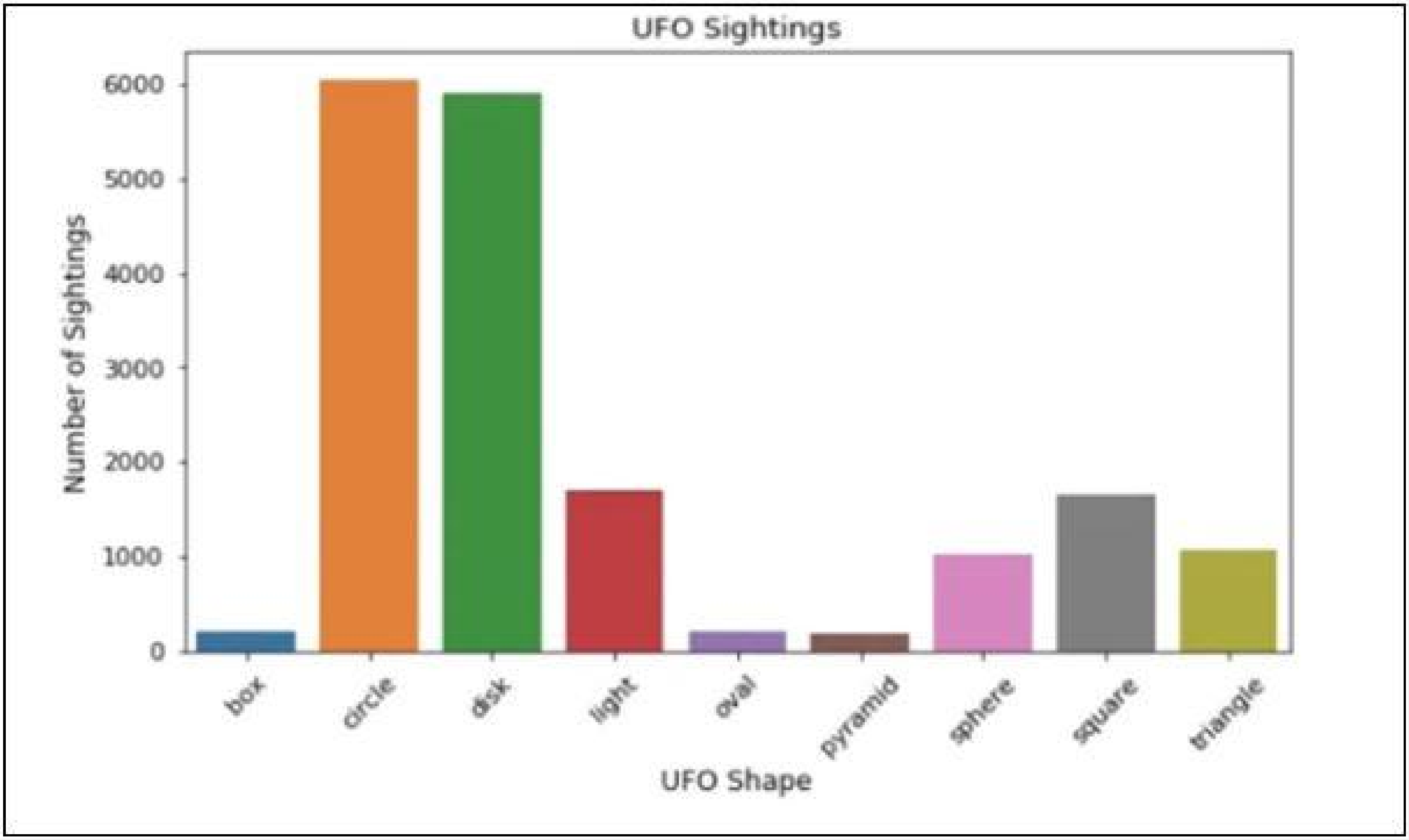
11. Once we create this visual, you can see that the majority, again, there was not much physical evidence left behind.



12. Next, look at the distribution of the different shapes for the UFO sightings.

```
In [ ]: m_cts = (df['shape'].value_counts())
        m_ctsx = m_cts.index
        m_ctsy = m_cts.get_values()
        f, ax = plt.subplots(figsize=(9,5))

        sns.barplot(x=m_ctsx, y=m_ctsy)
        ax.set_title('UFO Sightings')
        ax.set_xlabel('UFO Shape')
        ax.set_ylabel('Number of Sightings')
        plt.xticks(rotation=45)
        plt.show()
```

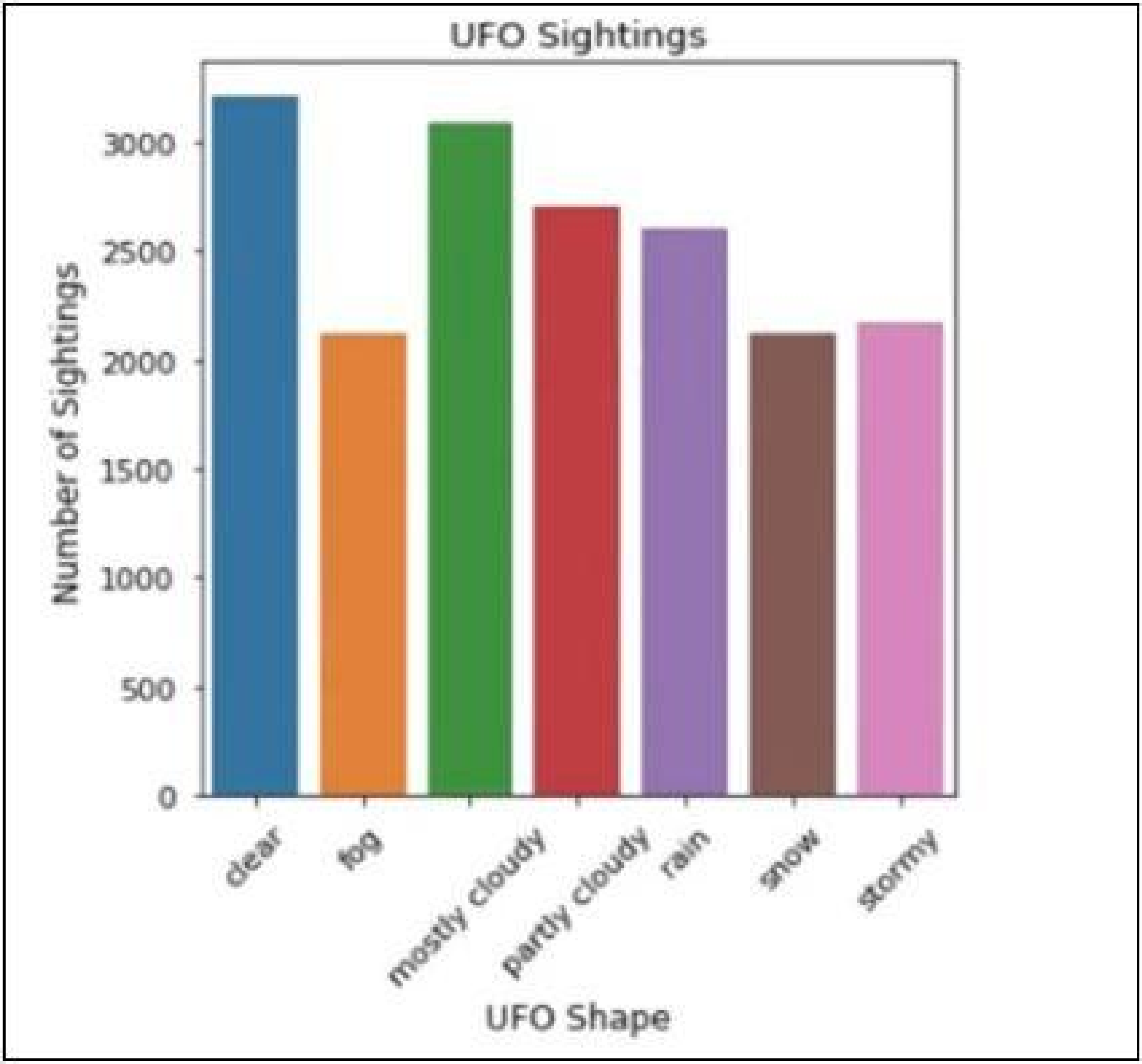
13. We can see that most of these sightings were either circles or disks, with a few more that were light and square. Then, they slowly decrease in value with the other shapes.



14. Now, to look at the distribution of weather, we can run the code below.

```
In [ ]: m_cts = (df['weather'].value_counts())
        m_ctsx = m_cts.index
        m_ctsy = m_cts.get_values()
        f, ax = plt.subplots(figsize=(5,5))

        sns.barplot(x=m_ctsx, y=m_ctsy)
        ax.set_title('UFO Sightings')
        ax.set_xlabel('UFO Shape')
        ax.set_ylabel('Number of Sightings')
        plt.xticks(rotation=45)
        plt.show()
```

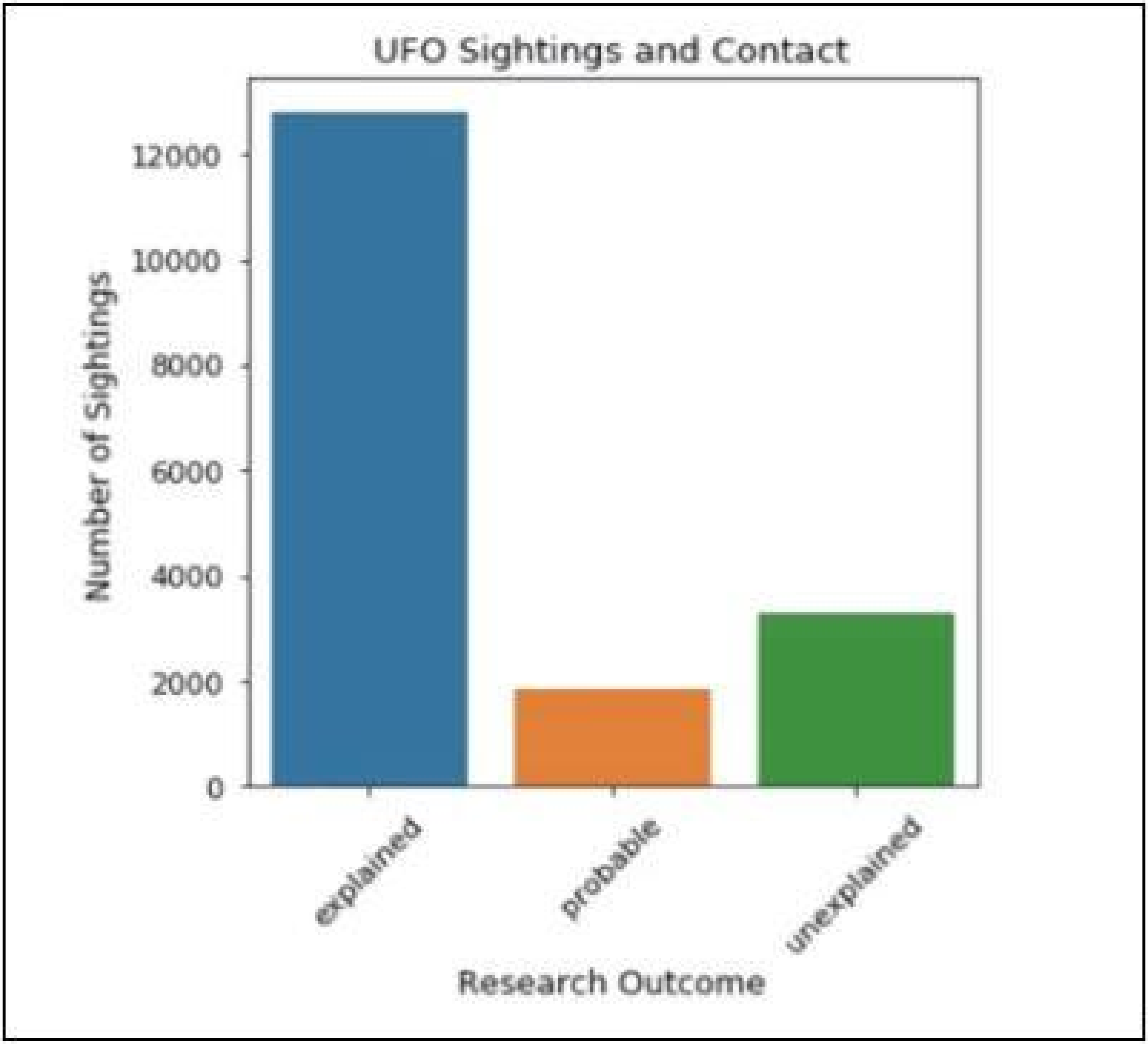
15. We can see here that most of the sightings were either clear or cloudy, and the other weather attributes follow. This distribution looks fairly even, so this may not correlate much with the outcome.



16. Now, we can run the code below to see what the researchOutcome distribution looks like.

```
In [ ]: m_cts = (df['researchOutcome'].value_counts())
        m_ctsx = m_cts.index
        m_ctsy = m_cts.get_values()
        f, ax = plt.subplots(figsize=(5,5))

        sns.barplot(x=m_ctsx, y=m_ctsy)
        ax.set_title('UFO Sightings and Contact')
        ax.set_xlabel('Research Outcome')
        ax.set_ylabel('Number of Sightings')
        # ax.set_xticklabels(['Unexplained', 'Explained', 'Probable'])
        plt.xticks(rotation=45)
        plt.show()
```

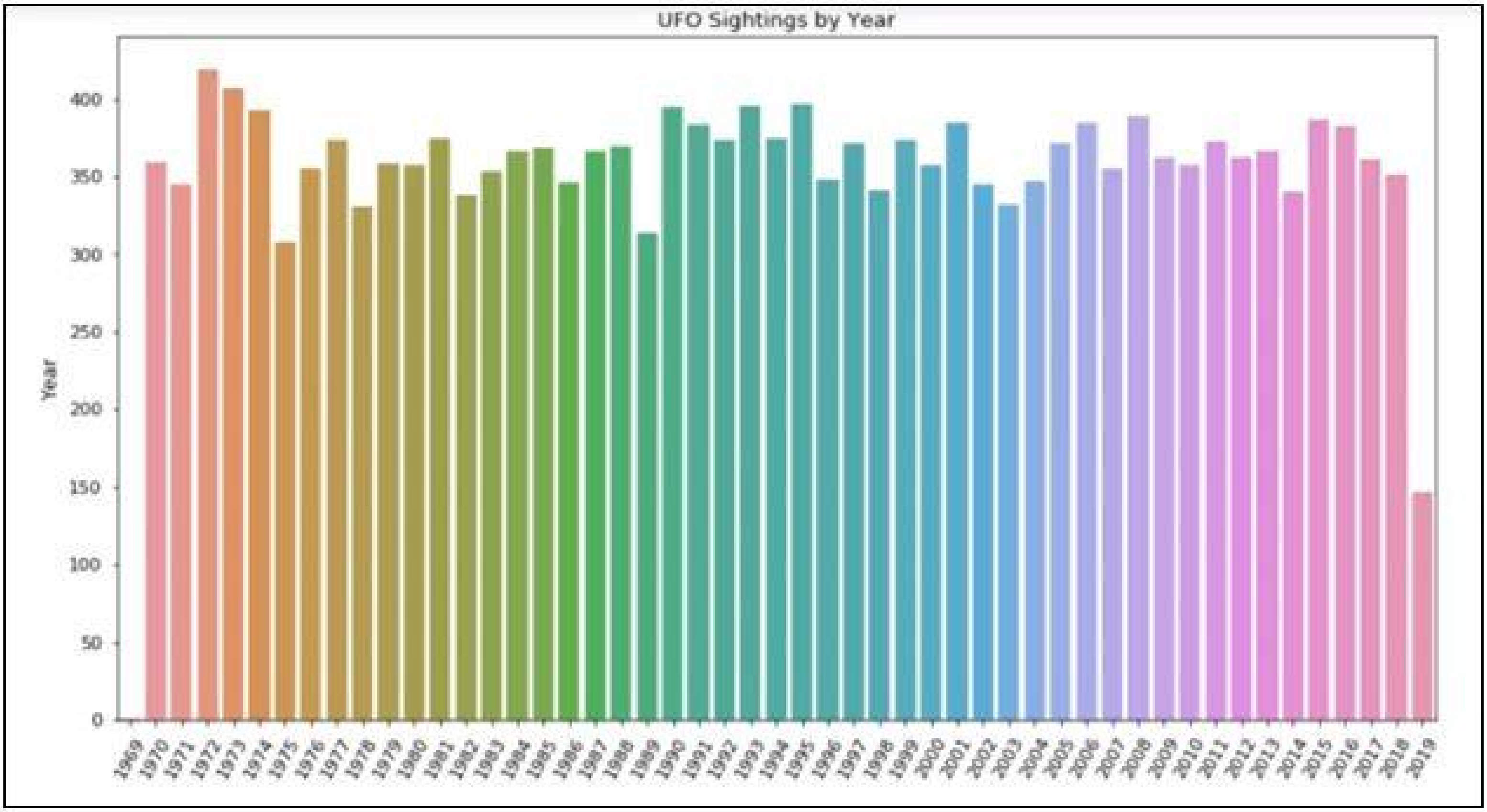
17. We can see the distribution in the researchOutcomes looks like the majority of our sightings were explained.



18. Looking at the distribution of sightings per year, we can run the code below.

```
In [ ]: ufo_yr = df['eventDate'].dt.year  # series with the year exclusively

        ## Set axes ##
        years_data = ufo_yr.value_counts()
        years_index = years_data.index  # x ticks
        years_values = years_data.get_values()

        ## Create Bar Plot ##
        plt.figure(figsize=(15,8))
        plt.xticks(rotation = 60)
        plt.title('UFO Sightings by Year')
        plt.ylabel('Number of Sightings')
        plt.ylabel('Year')

        years_plot = sns.barplot(x=years_index[:60],y=years_values[:60])
```

19. You can see that the distribution has the sightings per year is fairly even. This may not be much of a deciding factor in what the researchOutcome is.



20. We are going to look at the correlation of these values.

```
In [15]: df.corr()
```

Out[15]:

	duration	witnesses	latitude	longitude	physicalEvidence	contact
duration	1.000000	0.020679	0.000243	-0.010529	0.016430	0.015188
witnesses	0.020679	1.000000	0.010229	0.003449	0.009186	-0.000651
latitude	0.000243	0.010229	1.000000	-0.394536	0.006465	0.004284
longitude	-0.010529	0.003449	-0.394536	1.000000	-0.004519	-0.004828
physicalEvidence	0.016430	0.009186	0.006465	-0.004519	1.000000	0.693276
contact	0.015188	-0.000651	0.004284	-0.004828	0.693276	1.000000

21. You can see some strong correlations between physical evidence and contact. So usually, there is physical evidence at least 69% of the time whenever there is contact. We need to include both of these within our training data. We do not see shape and weather because these are categorical variables, so we cannot find any correlation between them. We will include those within our training data as well.

22. We are going to drop the columns that are not important. If you look at the dataset, there are several columns that are not important to include within our training dataset. For instance, we can drop the sighting because it is always Yes. We can also drop the first and last names because they are unimportant when determining the researchOutcome. We can also drop the reportedTimeStamp. And,

we could create some buckets for the eventDate and the eventTime, for instance, the season or what quarter it was in. But since the distribution of the dates is even, we can drop those.

```
In [ ]: df.drop(columns=['firstName', 'lastName', 'sighting', 'reportedTimestamp', 'eventDate', 'eventTime'], inplace=True)

In [ ]: df.head()
```

To drop these values, we can simply call df.drop and pass in the name of the columns we want to get rid of.

23. If we take a look at our dataset by calling df.head, we can see that these are the only attributes left within our dataset.

```
In [16]: df.drop(columns=['firstName', 'lastName', 'sighting', 'reportedTimestamp', 'eventDate'

In [17]: df.head()
Out[17]:
```

	shape	duration	witnesses	weather	latitude	longitude	physicalEvidence	contact	researchOutcome
0	circle	4	1	rain	47.329444	-122.578889	0	0	explained
1	disk	4	1	partly cloudy	52.664913	-1.034894	1	0	explained
2	circle	49	1	clear	38.951667	-92.333889	0	0	explained
3	disk	13	1	partly cloudy	41.496944	-71.367778	0	0	explained
4	circle	17	1	mostly cloudy	47.606389	-122.330833	0	0	explained

24. The XGBoost and the Linear Learner are both expecting numeric inputs. Hence, shape, weather, and researchOutcome need to be converted into numeric values.

Therefore, we will hot encode the weather and the shape attribute and map the researchOutcome, or the target attribute, into numeric values.

25. We will assign unexplained to zero, explained to one, and probable to two. Our machine learning algorithm is going to take all of the numeric input attributes from our dataset and try to find some correlation or some mapping that can be done to determine the researchOutcome as zero, one, or two.

```
In [18]: # Let's one-hot the weather and shape attribute
         df = pd.get_dummies(df, columns=['weather', 'shape'])

         # Let's replace the researchOutcome values with 0, 1, 2 for Unexplained, Explained, and Probable
         df['researchOutcome'] = df['researchOutcome'].replace({'unexplained': 0, 'explained': 1, 'probable': 2})

In [ ]: display(df.head())
        display(df.shape)
```

26. We can call df.head and df.shape to see what our dataset looks like now and how many attributes and columns we have.

```
In [19]: display(df.head())
         display(df.shape)
```

	duration	witnesses	latitude	longitude	physicalEvidence	contact	researchOutcome	weather_clear	weather_fog	weather_mostly cloudy	...	weather_stormy
0	4	1	47.329444	-122.578889	0	0	1	0	0	0	...	0
1	4	1	52.664913	-1.034894	1	0	1	0	0	0	...	0
2	49	1	38.951667	-92.333889	0	0	1	1	0	0	...	0
3	13	1	41.496944	-71.367778	0	0	1	0	0	0	...	0
4	17	1	47.606389	-122.330833	0	0	1	0	0	1	...	0

5 rows × 23 columns

(18000, 23)

We can see here that we have one hot encoded the weather and the shape attribute, and every other attribute is in numeric format.

27. Before we load our data into our machine learning algorithm, we will randomize and split the data into a couple of different datasets. We will randomize the data and split it into training, validation, and a testing dataset. We will split the data into 80% training, 10% validation, and 10% testing.

```
In [ ]: # Let's go ahead and randomize our data.
        df = df.sample(frac=1).reset_index(drop=True)

        # Next, Let's split the data into a training, validation, and testing.
        rand_split = np.random.rand(len(df))
        train_list = rand_split < 0.8                        # 80% for training
        val_list = (rand_split >= 0.8) & (rand_split < 0.9)  # 10% for validation
        test_list = rand_split >= 0.9                        # 10% for testing

         # This dataset will be used to train the model.
        data_train = df[train_list]

        # This dataset will be used to validate the model.
        data_val = df[val_list]

        # This dataset will be used to test the model.
        data_test = df[test_list]
```

You can see here that the first thing we do is randomize the data and then split it into three different datasets; data_train, data_val, and data_test.

28. Now, we need to rearrange our attributes, so the very first attribute is our target attribute or researchOutcome. The reason for doing this is that this is what XGBoost requires for the input data. It requires the first attribute in the dataset by the target attribute.

29. After rearranging those columns, we then want to store those as CSV files in our notebook instance. And then, we can load those files to S3 into their specific folders.

We will create a new folder called algorithms_lab and a sub_folder called xgboost_train and xgboost_validation. We will create the appropriate files for the training.csv and the validation.csv.

```
In [ ]: # Simply moves the researchOutcome attribute to the first position before creating CSV files
        pd.concat([data_train['researchOutcome'], data_train.drop(['researchOutcome'], axis=1)], axis=1).to_csv('train.csv', in
        pd.concat([data_val['researchOutcome'], data_val.drop(['researchOutcome'], axis=1)], axis=1).to_csv('validation.csv', i

        # Next we can take the files we just stored onto our Notebook instance and upload them to S3.
        boto3.Session().resource('s3').Bucket(bucket).Object('algorithms_lab/xgboost_train/train.csv').upload_file('train.csv')
        boto3.Session().resource('s3').Bucket(bucket).Object('algorithms_lab/xgboost_validation/validation.csv').upload_file('v
```

We are using CSV and not LibSVM because a data frame can quite easily be transformed into a CSV file by calling CSV.

Step 5: Creating and Training our model (XGBoost)

1. The first thing is to get the container hosted in the Elastic Container Repository within AWS. We will specify the region name and the algorithm for which we would like to get the container.

```
In [ ]: from sagemaker.amazon.amazon_estimator import get_image_uri
        container = get_image_uri(boto3.Session().region_name, 'xgboost')
```

Next, because we're training with the CSV file format, we'll create inputs that our training function can use as a pointer to the files in S3, which also specify that the content type is CSV.

```
In [ ]: s3_input_train = sagemaker.s3_input(s3_data='s3://{}/algorithms_lab/xgboost_train'.format(bucket), content_type='csv')
        s3_input_validation = sagemaker.s3_input(s3_data='s3://{}/algorithms_lab/xgboost_validation'.format(bucket), content_ty
```

We will create a pointer that points to our S3 files for our training and our validation data.

2. First, we will create a specific job name with the job we are about to run.

```
In [ ]: # Create a training job name
        job_name = 'ufo-xgboost-job-{}'.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 = 's3://{}/algorithms_lab/xgboost_output'.format(bucket)
```

We can run the cell and see the job name.

```
In [24]: # Create a training job name
         job_name = 'ufo-xgboost-job-{}'.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 = 's3://{}/algorithms_lab/xgboost_output'.format(bucket)

         Here is the job name ufo-xgboost-job-20190523170413
```

3. Next, we can specify the training parameters we want to use with the XGBoost algorithm. We will pass in the algorithm container, IAM role, training instance type, and the count. In our case, we will

use an ml.m4.xlarge as our training instance type and only use one of them. We will also pass in the S3 location for our output model artifact. We will also pass in the XGBoost hyperparameters. These are the only hyperparameters we will include within our XGBoost algorithm.

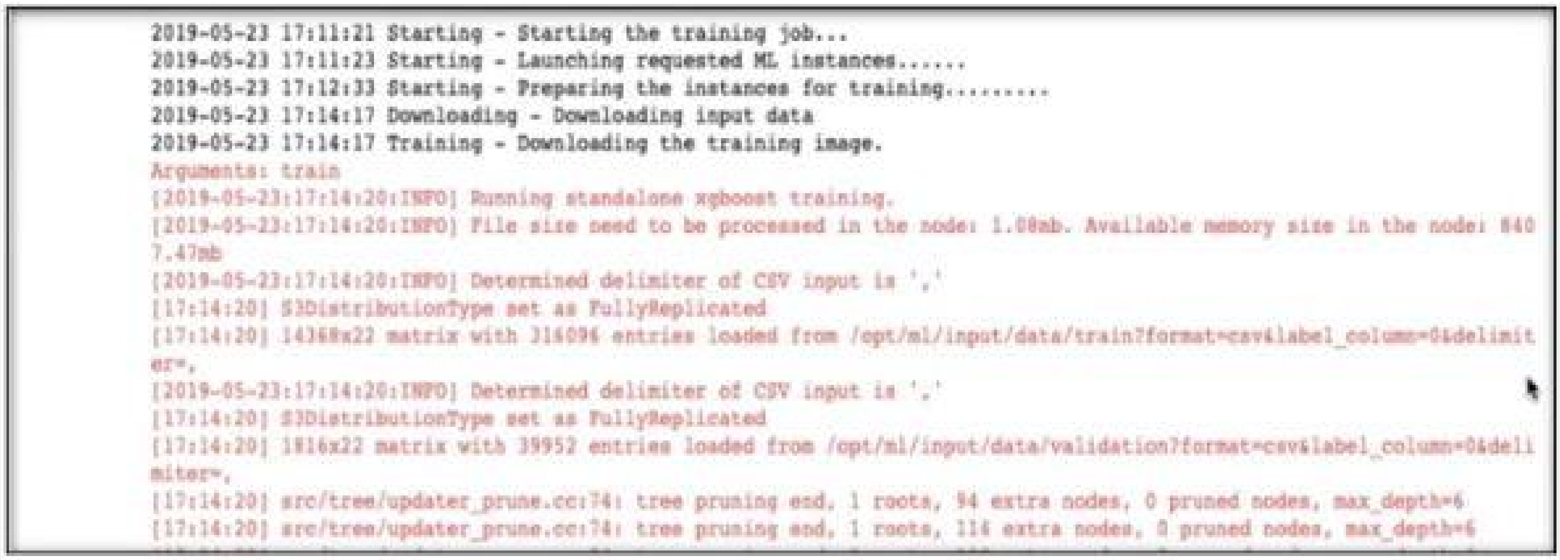
We can see that our objective is going to be multisoftmax. We could use softprob, but since we have a multi-classification problem, we need to set this to multisoftmax or multisoftprob. Both of these will give us our desired output.

We will also include the number of classes, which is three, for explained, unexplained, and probable. We will also pass in the num_rounds, the number of rounds we want our algorithm to pass over our data.

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

        xgb = sagemaker.estimator.Estimator(container,
                                            role,
                                            train_instance_count=1,
                                            train_instance_type='ml.m4.xlarge',
                                            output_path=output_location,
                                            sagemaker_session=sess)

        xgb.set_hyperparameters(objective='multi:softmax',
                                num_class=3,
                                num_round=100)

        data_channels = {
            'train': s3_input_train,
            'validation': s3_input_validation
        }
        xgb.fit(data_channels, job_name=job_name)
```

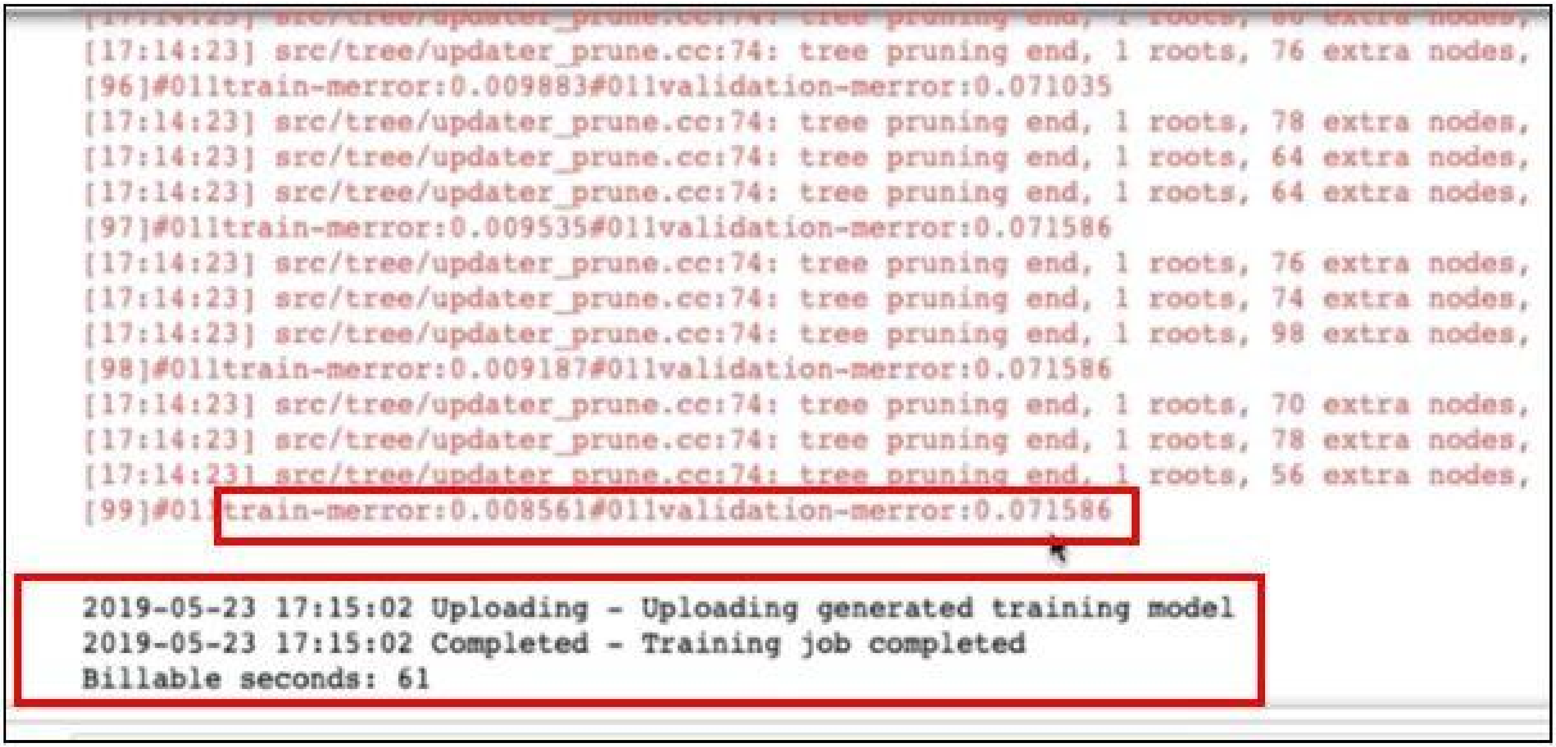
4. To start the job and our training process, we can use the .fit method and pass in the training and validation data.

5. You can see that the training job has started.



6. As we scroll down, we can see the model's output as it was training. Here, we have 0.03, which is about a 3% error; for validation, we can see that it has about a 6% error. That is around 94 to 97% accuracy.

7. We will continue to scroll down and see what the final output was. You can see that the final output was about .8% training error, which is about 99% accuracy. And about a 7% error, which is around 93% accuracy.



```
[17:14:23] src/tree/updater_prune.cc:74: tree pruning end, 1 roots, 76 extra nodes,
[96]#011train-merror:0.009883#011validation-merror:0.071035
[17:14:23] src/tree/updater_prune.cc:74: tree pruning end, 1 roots, 78 extra nodes,
[17:14:23] src/tree/updater_prune.cc:74: tree pruning end, 1 roots, 64 extra nodes,
[17:14:23] src/tree/updater_prune.cc:74: tree pruning end, 1 roots, 64 extra nodes,
[97]#011train-merror:0.009535#011validation-merror:0.071586
[17:14:23] src/tree/updater_prune.cc:74: tree pruning end, 1 roots, 76 extra nodes,
[17:14:23] src/tree/updater_prune.cc:74: tree pruning end, 1 roots, 74 extra nodes,
[17:14:23] src/tree/updater_prune.cc:74: tree pruning end, 1 roots, 98 extra nodes,
[98]#011train-merror:0.009187#011validation-merror:0.071586
[17:14:23] src/tree/updater_prune.cc:74: tree pruning end, 1 roots, 70 extra nodes,
[17:14:23] src/tree/updater_prune.cc:74: tree pruning end, 1 roots, 78 extra nodes,
[17:14:23] src/tree/updater_prune.cc:74: tree pruning end, 1 roots, 56 extra nodes,
[99]#011train-merror:0.008561#011validation-merror:0.071586

2019-05-23 17:15:02 Uploading - Uploading generated training model
2019-05-23 17:15:02 Completed - Training job completed
Billable seconds: 61
```

This is exactly what we need. We have some data to show to our organization.

8. Now, we can print out the location of our trained model. If we look at this location S3, we can see our model artifact, and we can use this model artifact for inference to deploy it into a production environment later.

```
In [26]: print('Here is the location of the trained XGBoost model: {}/{}/output/model.tar.gz'.format(output_location, job_name))

Here is the location of the trained XGBoost model: s3://ml-labs-acg/algorithms_lab/xgboost_output/ufo-xgboost-job-201
90523170413/output/model.tar.gz
```

Step 6: Create and train our model (Linear Learner)

Now, we can start creating and training a Linear Learner model, where we can start to compare the accuracy between the two.

1. Since our data is already prepared and ready to go, we will randomize and split the data again for the Linear Learner algorithm.

2. We will split the data into three different datasets, a training, validation, and testing dataset. It will be 80% training, 10% validation, and 10% testing.

```
In [ ]: np.random.seed(0)
        rand_split = np.random.rand(len(df))
        train_list = rand_split < 0.8
        val_list = (rand_split >= 0.8) & (rand_split < 0.9)
        test_list = rand_split >= 0.9

         # This dataset will be used to train the model.
        data_train = df[train_list]

        # This dataset will be used to validate the model.
        data_val = df[val_list]

        # This dataset will be used to test the model.
        data_test = df[test_list]
```

3. It will create a data frame for each dataset type, and it will rearrange the researchOutcome to where it is the very first column within the datasets for each of them.

We will also break the datasets into NumPy ndarrays. This is getting our data ready for the recordIO-protobuf. These are the necessary steps to get it ready for that format.

```
# This rearranges the columns
cols = list(data_train)
cols.insert(0, cols.pop(cols.index('researchOutcome')))
data_train = data_train[cols]

cols = list(data_val)
cols.insert(0, cols.pop(cols.index('researchOutcome')))
data_val = data_val[cols]

cols = list(data_test)
cols.insert(0, cols.pop(cols.index('researchOutcome')))
data_test = data_test[cols]

# Breaks the datasets into attribute numpy.ndarray and the same for target attribute.
train_X = data_train.drop(columns='researchOutcome').as_matrix()
train_y = data_train['researchOutcome'].as_matrix()

val_X = data_val.drop(columns='researchOutcome').as_matrix()
val_y = data_val['researchOutcome'].as_matrix()

test_X = data_test.drop(columns='researchOutcome').as_matrix()
test_y = data_test['researchOutcome'].as_matrix()
```

4. Now, we will create a recordIO file for training data and upload it to S3.

```
In [ ]: train_file = 'ufo_sightings_train_recordIO_protobuf.data'

        f = io.BytesIO()
        smac.write_numpy_to_dense_tensor(f, train_X.astype('float32'), train_y.astype('float32'))
        f.seek(0)

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

It is building a recordIO-protobuf for that dataset, which in our case is the training dataset, by taking the x-values, which are everything but the researchOutcome, and the y-values, which are only the researchOutcome.

5. When we run the code, we can see this is the location of where our recordIO-protobuf is stored.

```
In [28]: train_file = 'ufo_sightings_train_recordIO_protobuf.data'

         f = io.BytesIO()
         smac.write_numpy_to_dense_tensor(f, train_X.astype('float32'), train_y.astype('float32'))
         f.seek(0)

         boto3.Session().resource('s3').Bucket(bucket).Object('algorithms_lab/linearlearner_train/{}'.format(train_file)).uploac
         training_recordIO_protobuf_location = 's3://{}/algorithms_lab/linearlearner_train/{}'.format(bucket, train_file)

         The Pipe mode recordIO protobuf training data: s3://ml-labs-acg/algorithms_lab/linearlearner_train/ufo_sightings_trai
         n_recordIO_protobuf.data
```

6. We are going to perform the same steps for validation data.

When we run the code, we can see the S3 location of our validation data in recordIO-protobuf format.

```
In [29]: validation_file = 'ufo_sightings_validatioin_recordIO_protobuf.data'

         f = io.BytesIO()
         smac.write_numpy_to_dense_tensor(f, val_X.astype('float32'), val_y.astype('float32'))
         f.seek(0)

         boto3.Session().resource('s3').Bucket(bucket).Object('algorithms_lab/linearlearner_validation/{}'.format(validation_fil
         validate_recordIO_protobuf_location = 's3://{}/algorithms_lab/linearlearner_validation/{}'.format(bucket, validation_fi
         print('The Pipe mode recordIO protobuf validation data: {}'.format(validate_recordIO_protobuf_location))

         The Pipe mode recordIO protobuf validation data: s3://ml-labs-acg/algorithms_lab/linearlearner_validation/ufo_sightin
         gs_validatioin_recordIO_protobuf.data
```

7. We will go through the same setup we did for the XGBoost algorithm, but we will use the Linear Learner algorithm.

We are going to get everything we need from the ECR repository. This will create a container containing the Linear Learner docker image that we can use to train our model with.

```
In [ ]: from sagemaker.amazon.amazon_estimator import get_image_uri
        import sagemaker

        container = get_image_uri(boto3.Session().region_name, 'linear-learner', "1")
```

8. We will create a custom job name and specify the output location for our model artifact.

```
In [ ]: # Create a training job name
        job_name = 'ufo-linear-learner-job-{}'.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 = 's3://{}/algorithms_lab/linearlearner_output'.format(bucket)
```

9. This will be the name of the training job we are about to create.

```
In [31]: # Create a training job name
         job_name = 'ufo-linear-learner-job-{}'.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 = 's3://{}/algorithms_lab/linearlearner_output'.format(bucket)

         Here is the job name ufo-linear-learner-job-20190523180102
```

10. We will call the SageMaker library to create an estimator, which is a way to create a machine learning model by inputting the container and some other parameters about the algorithm we want to use. We can include the container, the IAM role, the training instance count, and the training instance type, and AWS recommends using a CPU instance. Hence, we will use an ml.c4.xlarge to train our model. We can specify the output location where the model artifact will live. We will pass in the SageMaker session and include the input_mode type of Pipe.

We can take advantage of using recordIO-protobuf as the input type and stream the data directly from S3. This will optimize the streaming process of getting the data from S3 onto our notebook instance.

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

In [ ]: sess = sagemaker.Session()

        # Setup the LinearLeaner algorithm from the ECR container
        linear = sagemaker.estimator.Estimator(container,
                                               role,
                                               train_instance_count=1,
                                               train_instance_type='ml.c4.xlarge',
                                               output_path=output_location,
                                               sagemaker_session=sess,
                                               input_mode='Pipe')
        # Setup the hyperparameters
```

11. We can specify all the hyperparameters associated with the Linear Learner algorithm. We can scroll down and take a look at which ones are required. The ones that are required are the featured dimensions. In our case, it will be the number of attributes minus the researchOutcome, so we have 22 total features or 22 total attributes.

12. We will specify the predictor type, which is a multiclass_classifier because this is a multi-classification problem. We will also specify the number of classes, similar to the XGBoost, we have explained unexplained and probable. These are the only hyperparameters that are required for the Linear Learner algorithm.

```
# Setup the hyperparameters
linear.set_hyperparameters(feature_dim=22, # number of attributes (minus the researchOutcome attribute)
                           predictor_type='multiclass_classifier', # type of classification problem
                           num_classes=3)  # number of classes in out researchOutcome (explained, unexplained, probable


# Launch a training job. This method calls the CreateTrainingJob API call
data_channels = {
    'train': training_recordIO_protobuf_location,
    'validation': validate_recordIO_protobuf_location
}
linear.fit(data_channels, job_name=job_name)
```

13. We will leave the others as default and see what our results look like. We will call the .fit method, pass in our training data, which in this case is recordIO-protobuf, and pass in the job name that we created earlier.

```
# Launch a training job. This method calls the CreateTrainingJob API call
data_channels = {
    'train': training_recordIO_protobuf_location,
    'validation': validate_recordIO_protobuf_location
}
linear.fit(data_channels, job_name=job_name)

2019-05-23 18:05:07 Starting - Starting the training job.
```

14. The first thing you can do is the default configurations for the hyperparameters. So if we look at this JSON file or this JSON data, we should be able to see each hyperparameter that the Linear Learner algorithm uses to create the model.

```
2019-05-30 13:17:16 Downloading - Downloading input data...
2019-05-30 13:17:27 Training - Downloading the training image...
2019-05-30 13:18:17 Training - Training image download completed. Training in progress..
Docker entrypoint called with argument(s): train
[05/30/2019 13:18:19 INFO 139734425163584] Reading default configuration from /opt/amazon/lib/python2.7/site-package
s/algorithm/default-input.json: {u'loss_insensitivity': u'0.01', u'epochs': u'15', u'init_bias': u'0.0', u'lr_schedul
er_factor': u'auto', u'num_calibration_samples': u'10000000', u'accuracy_top_k': u'3', u'_num_kv_servers': u'auto',
 u'use_bias': u'true', u'num_point_for_scaler': u'10000', u'_log_level': u'info', u'quantile': u'0.5', u'bias_lr_mul
t': u'auto', u'lr_scheduler_step': u'auto', u'init_method': u'uniform', u'init_sigma': u'0.01', u'lr_scheduler_minimu
m_lr': u'auto', u'target_recall': u'0.8', u'num_models': u'auto', u'early_stopping_patience': u'3', u'momentum': u'au
to', u'unbias_label': u'auto', u'wd': u'auto', u'optimizer': u'auto', u'_tuning_objective_metric': u'', u'early_stopp
ing_tolerance': u'0.001', u'learning_rate': u'auto', u'_kvstore': u'auto', u'normalize_data': u'true', u'binary_class
ifier_model_selection_criteria': u'accuracy', u'use_lr_scheduler': u'true', u'target_precision': u'0.8', u'unbias_dat
a': u'auto', u'init_scale': u'0.07', u'bias_wd_mult': u'auto', u'f_beta': u'1.0', u'mini_batch_size': u'1000', u'hube
r_delta': u'1.0', u'num_classes': u'1', u'beta_1': u'auto', u'loss': u'auto', u'beta_2': u'auto', u'_enable_profile
r': u'false', u'normalize_label': u'auto', u'_num_gpus': u'auto', u'balance_multiclass_weights': u'false', u'positive
_example_weight_mult': u'1.0', u'l1': u'auto', u'margin': u'1.0'}
[05/30/2019 13:18:19 INFO 139734425163584] Reading provided configuration from /opt/ml/input/config/hyperparameters.j
son: {u'feature_dim': u'22', u'predictor_type': u'multiclass_classifier', u'num_classes': u'3'}
```

15. In our case, we can see the hyperparameters that we input. We can see num_classes.

16. We can also see the featured dimensions.

17. If we scroll down, we can see the output from our Linear Learner algorithm. The metrics that mean a lot to us are the validation_score metric and the quality_metric.

18. We can start to see the multiclass_accuracy is around 0.94%, which is around 94% accuracy. And

we can also see recall, precision, and the F1 score.

```
[2019-05-30 13:18:25.939] [tensorio] [info] data_pipeline_stats={"name": "/opt/ml/input/data/validation", "epoch": 1
1, "duration": 53, "num_examples": 2}
[05/30/2019 13:18:25 INFO 139734425163584] #validation_score (algo-1) : ('multiclass_cross_entropy_objective', 0.1959
2909105605347)
[05/30/2019 13:18:25 INFO 139734425163584] #validation_score (algo-1) : ('multiclass_accuracy', 0.9447264523406655)
[05/30/2019 13:18:25 INFO 139734425163584] #validation_score (algo-1) : ('multiclass_top_k_accuracy_3', 1.0)
[05/30/2019 13:18:25 INFO 139734425163584] #validation_score (algo-1) : ('dcg', 0.9776801697224866)
[05/30/2019 13:18:25 INFO 139734425163584] #validation_score (algo-1) : ('macro_recall', 0.9270454)
[05/30/2019 13:18:25 INFO 139734425163584] #validation_score (algo-1) : ('macro_precision', 0.89689428)
[05/30/2019 13:18:25 INFO 139734425163584] #validation_score (algo-1) : ('macro_f_1.000', 0.91093159)
[05/30/2019 13:18:25 INFO 139734425163584] #quality_metric: host=algo-1, validation multiclass_cross_entropy_objectiv
e <loss>=0.195929091056
[05/30/2019 13:18:25 INFO 139734425163584] #quality_metric: host=algo-1, validation multiclass_accuracy <score>=0.944
726452341
[05/30/2019 13:18:25 INFO 139734425163584] #quality_metric: host=algo-1, validation multiclass_top_k_accuracy_3 <scor
e>=1.0
[05/30/2019 13:18:25 INFO 139734425163584] #quality_metric: host=algo-1, validation dcg <score>=0.977680169722
[05/30/2019 13:18:25 INFO 139734425163584] #quality_metric: host=algo-1, validation macro_recall <score>=0.9270454049
11
```

19. If we scroll down all the way to the bottom, we can see that the training job is completed.

```
[2019-05-30 13:18:25.993] [tensorio] [info] data_pipeline_stats={"name": "/opt/ml/input/data/validation", "epoch": 1
2, "duration": 54, "num_examples": 2}
[2019-05-30 13:18:25.993] [tensorio] [info] data_pipeline_stats={"name": "/opt/ml/input/data/validation", "duration":
6663, "num_epochs": 13, "num_examples": 25}
#metrics {"Metrics": {"totaltime": {"count": 1, "max": 6755.892992019653, "sum": 6755.892992019653, "min": 6755.89299
2019653}, "finalize.time": {"count": 1, "max": 113.7399673461914, "sum": 113.7399673461914, "min": 113.739967346191
4}, "initialize.time": {"count": 1, "max": 439.47577476501465, "sum": 439.47577476501465, "min": 439.47577476501465},
"check_early_stopping.time": {"count": 11, "max": 1.132965087890625, "sum": 6.217479705810547, "min": 0.1821517944335
9375}, "setuptime": {"count": 1, "max": 15.5029296875, "sum": 15.5029296875, "min": 15.5029296875}, "update.time":
 {"count": 10, "max": 700.509786605835, "sum": 6071.507215499878, "min": 522.4170684814453}, "epochs": {"count": 1,
 "max": 15, "sum": 15.0, "min": 15}}, "EndTime": 1559222305.99424, "Dimensions": {"Host": "algo-1", "Operation": "tra
ining", "Algorithm": "Linear Learner"}, "StartTime": 1559222299.314418}


2019-05-30 13:18:34 Uploading - Uploading generated training model
2019-05-30 13:18:34 Completed - Training job completed
Billable seconds: 78
```

20. If we run the cell, we can see where the Linear Learner model artifact is stored.

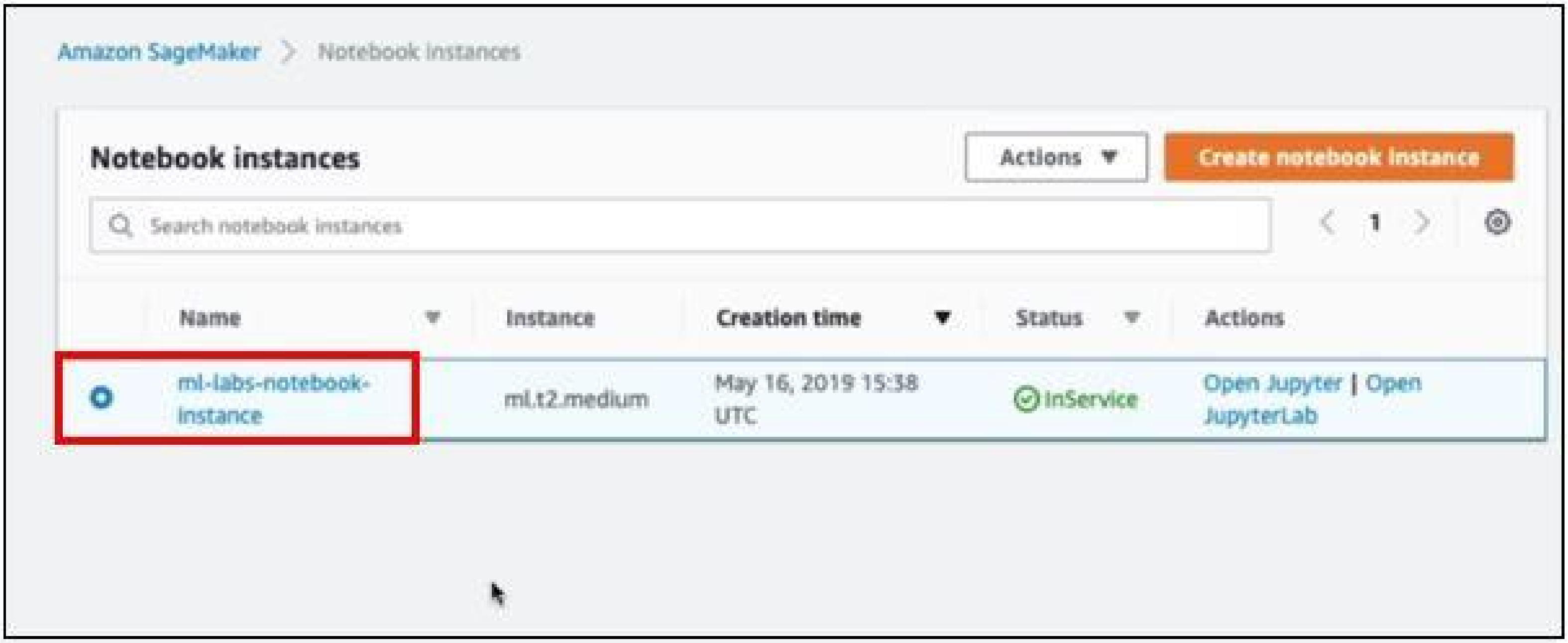
```
In [35]: print('Here is the location of the trained Linear Learner model: {}/{}/output/model.tar.gz'.format(output_location, job

         Here is the location of the trained Linear Learner model: s3://ml-labs-acg/algorithms_lab/linearlearner_output/ufo-li
         near-learner-job-20190530131517/output/model.tar.gz
```

21. It is important to know how long the training job took, and you can see the billable seconds were 78 seconds. So, it charged us 78 seconds when it spun up the compute-optimized instance and trained our model.

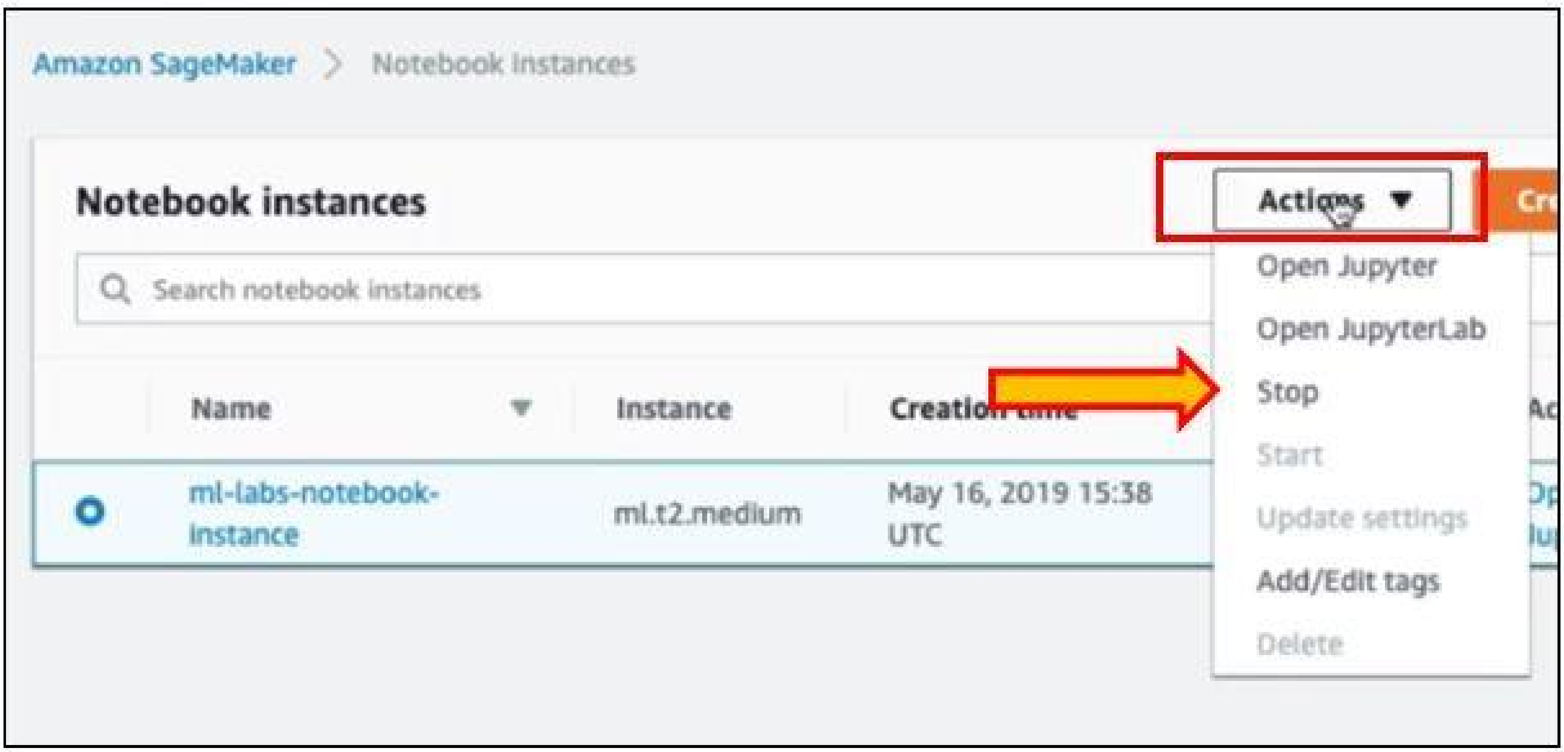
```
2019-05-30 13:18:34 Uploading - Uploading generated training model
2019-05-30 13:18:34 Completed - Training job completed
Billable seconds: 78
```

Hence, we have created an XGBoost model and a Linear Learner model. Our XGBoost model has around 93% accuracy, and our Linear Learner has around 94% accuracy. Therefore, we have two models that are above 90% accurate. We can use these metrics to show the organization that we can use them in production or make inference calls.

22. We can now clean up our environment. Navigate back to SageMaker and select your notebook instance.



23. Use the Actions dropdown and select Stop.



Lab 10: Understanding CloudFormation Template Anatomy

Certifications	AWS Machine Learning - Specialty
Cloud Provider	AWS
Time	

Required	1 Hr.
Tags	CF, AWS

Lab Prerequisites

- Familiarity with basic AWS Cloud Computing concepts and terminology
- An AWS account with an active subscription

Service Introduction

AWS CloudFormation is a tool that makes it simple for developers and organizations to construct a collection of linked AWS and third-party resources and then provision and manage them logically and reasonably.

Developers may use a simple, declarative approach to deploy and change compute, database, and many other resources, abstracting away the complexities of individual resource APIs. AWS CloudFormation is meant to make resource lifecycle management repeatable, predictable, and safe, with features such as automatic rollbacks, automated state management, and resource management across accounts and regions. Multiple ways to generate resources have recently been added, including leveraging AWS CDK for writing in higher-level languages, importing existing resources, and detecting configuration drift. A new registry simplifies the creation of custom types that inherit several of CloudFormation's fundamental features.

Case Study Security Products Manufacturers – Foxis Security Solutions

Background

Foxis Security Systems International manufactures security equipment, such as surveillance systems, alarms, bulletproof glass and vehicles, safes, and associated technology. Foxis is a 2,500-employee business with global offices. Its IT team of 15 is in charge of the day-to-day operation of its datacenters and the improvement of the business through innovative use of technology.

Foxis Security Solutions has two datacenters, one in New York and the other in Los Angeles, housed in a co-location facility. The environment is largely comprised of Web servers, with around 60% running on a VMware cluster, 30% on a newer Hyper-V cluster, and the rest on older, physical servers. Some Linux servers and two major, in-house-developed Line-Of-Business (LOB) applications. The IT department is currently facing a difficult scenario, with outdated technology in both datacenters and a declining budget.

Mike, the CIO, joins a meeting with the whole IT employees to discuss how to improve Foxis' IT environment over the next 12 to 24 months. Selecting a public/hybrid cloud provider is a long-term plan, and CIO believes Amazon Web Services provides the best solution for a gradual transition from Foxis' present systems to a hybrid/mostly public cloud-based infrastructure.

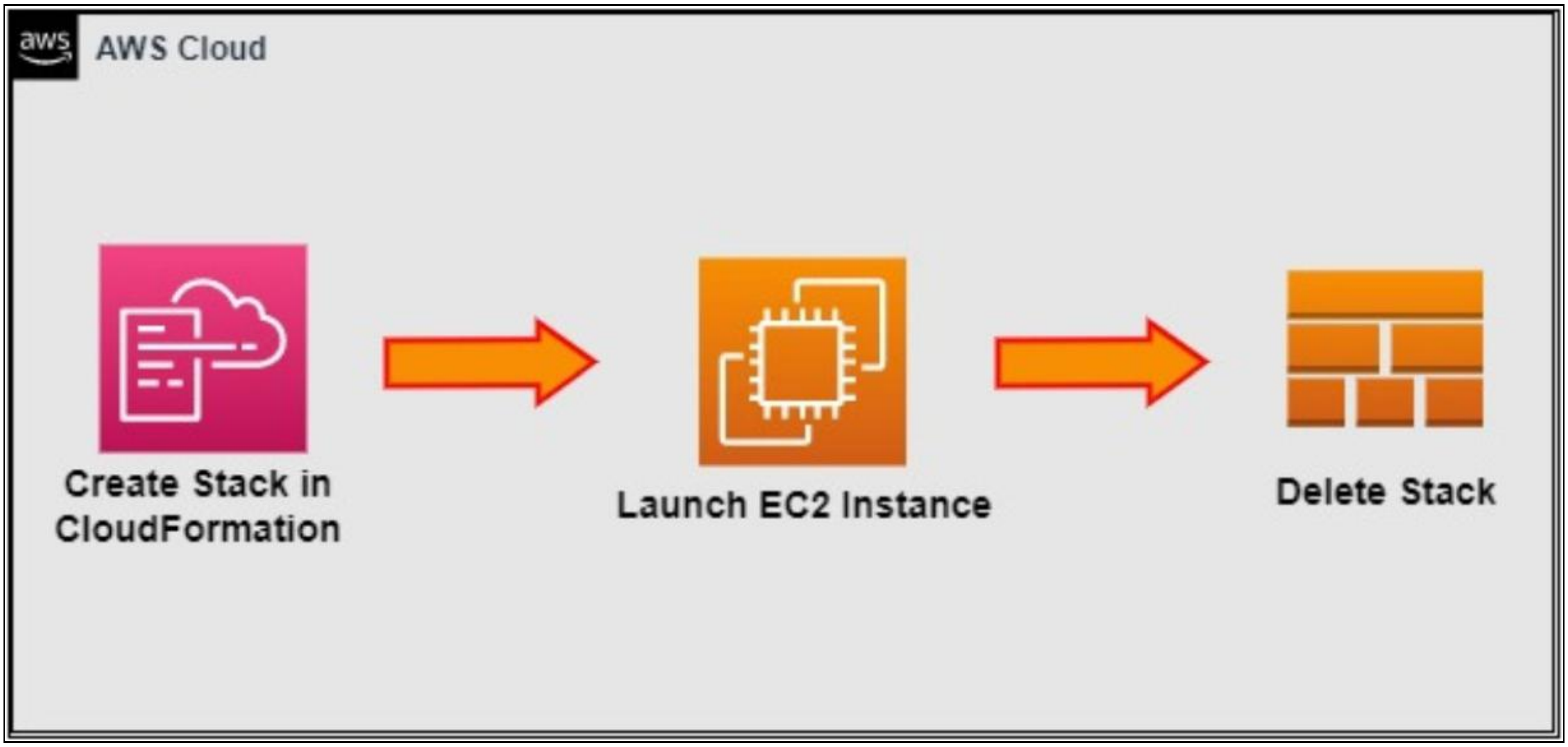
Business Challenge

You are a DevOps engineer in Foxis Security Solutions. Your task is to continue innovating with technology and platform improvements to create an excellent customer experience. For this, you need a tool that will automate all of your manual work, such as launching and configuring EC2 instances or creating S3 storage buckets. How can you automate this work?

Proposed Solution

As the DevOps engineer, you have been asked to use the AWS CloudFormation service to automate all the work. In CloudFormation, you run a template of JSON or YAML to deploy the entire frontend and backend stack into the AWS environment.

Lab Diagram



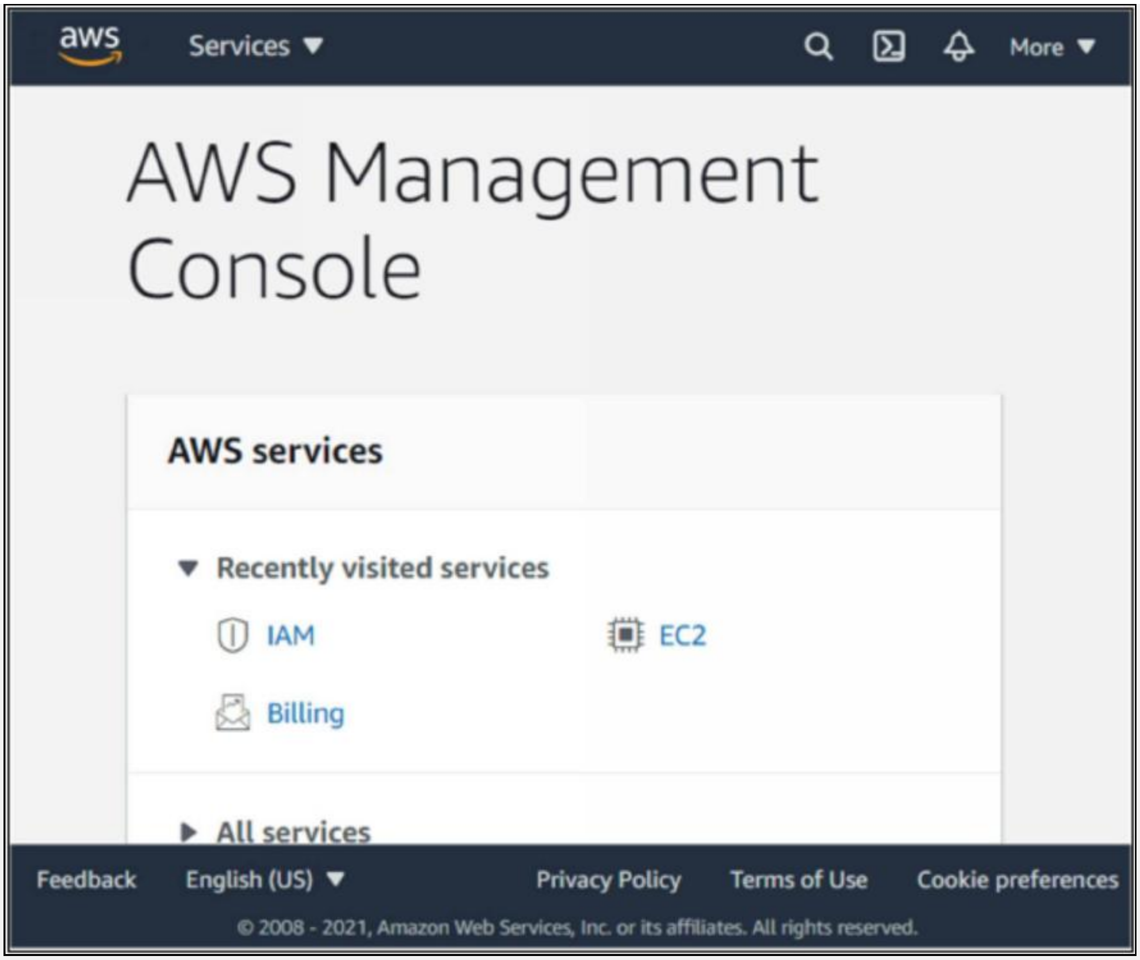
Implementation Steps

1. Create Stack in CloudFormation.
2. Observe the Stack Events.
3. Delete a CloudFormation stack.

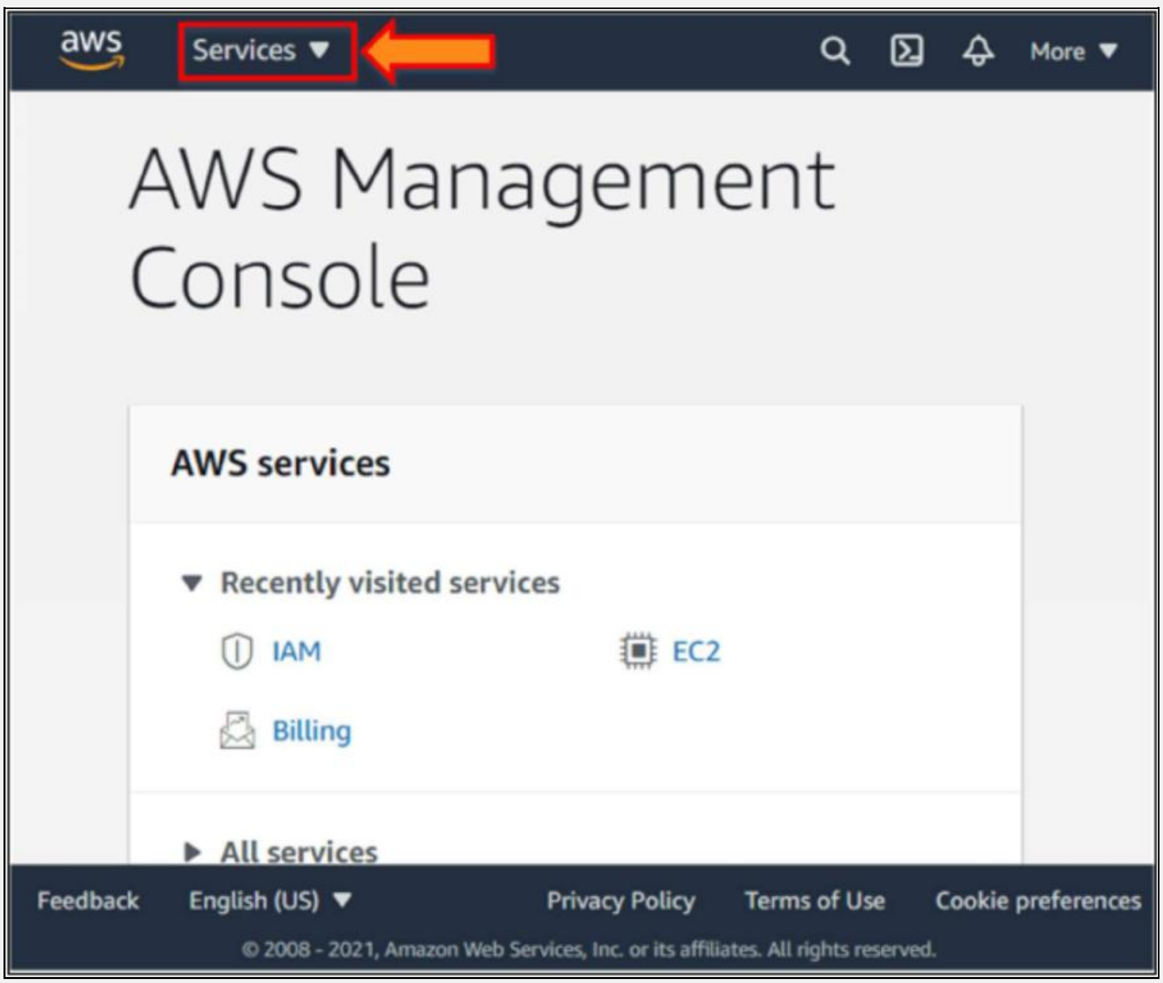
Solution

Step 1: Create Stack in CloudFormation

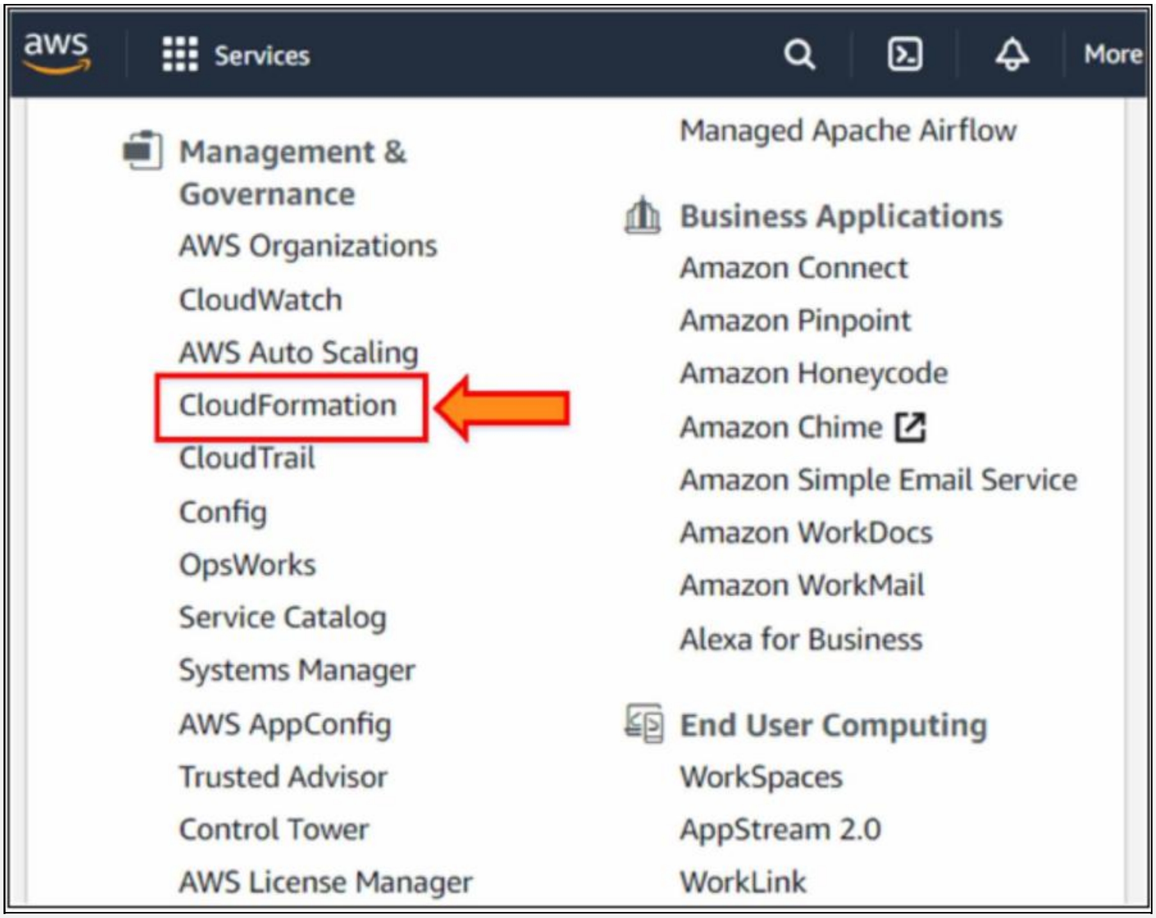
1. Log in to the **AWS Management Console.**



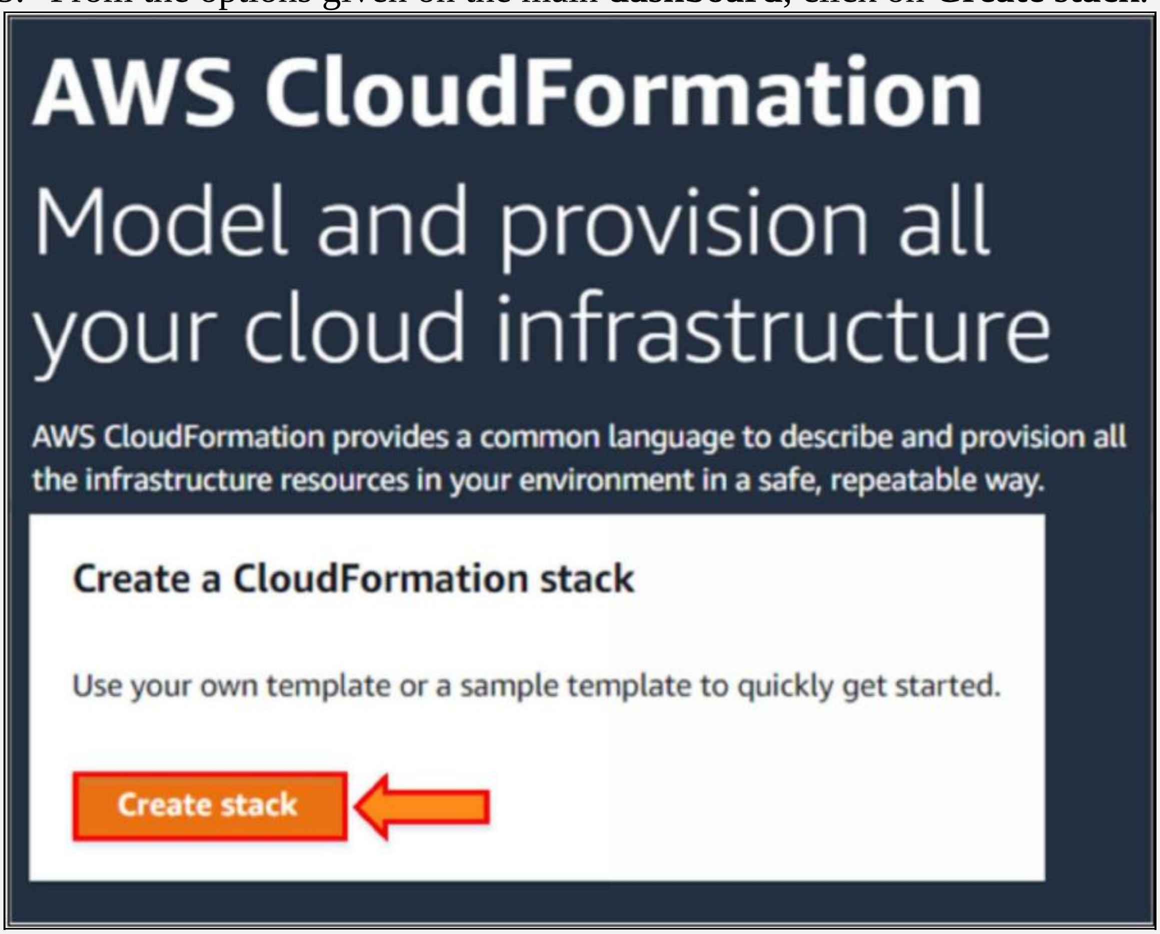
2. Click on **Services**.



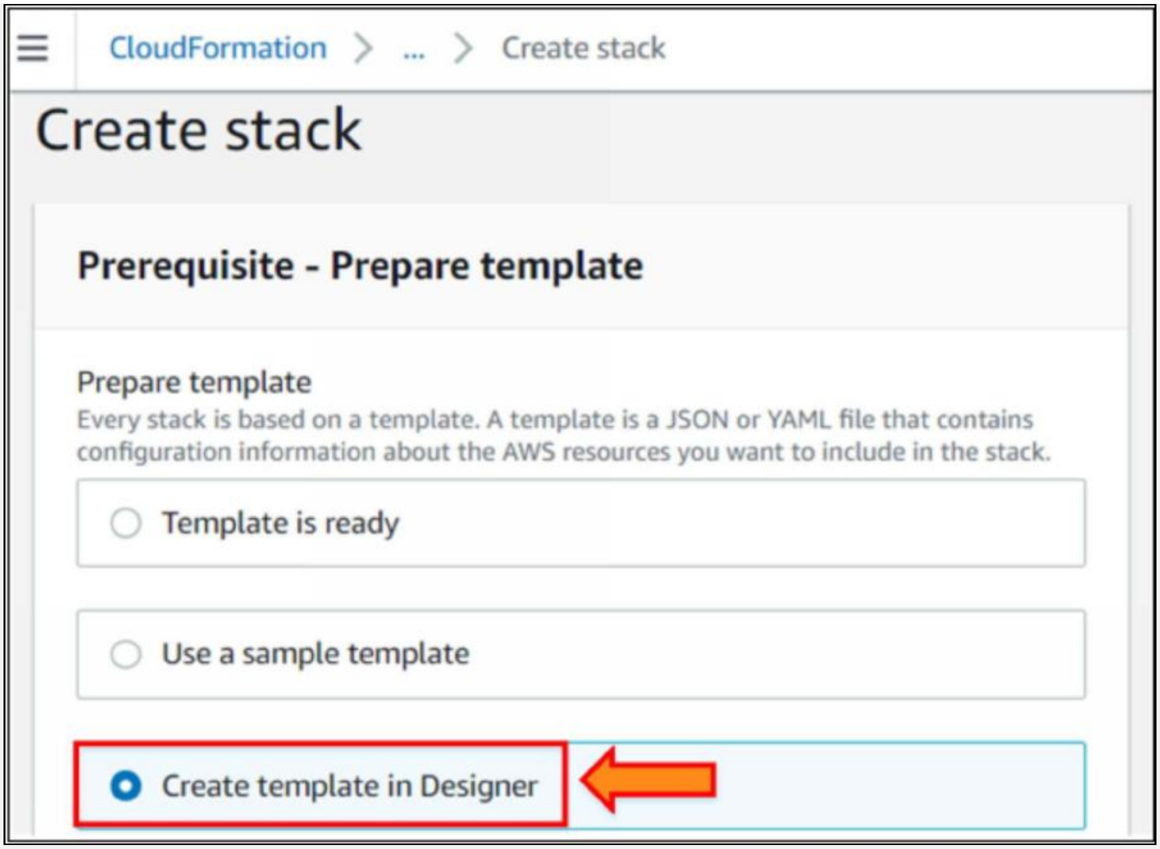
3. Scroll down to **Management & Governance**. Select **Cloud Formation**.



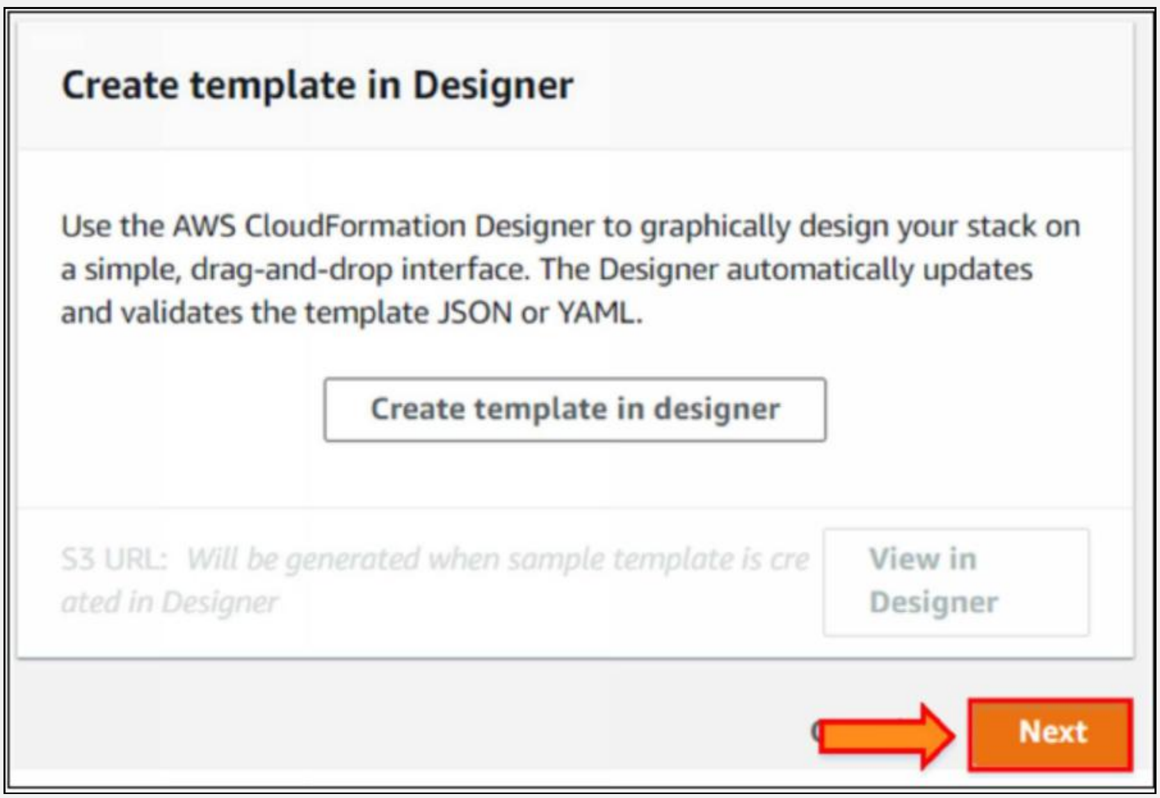
4. If you have never used **CloudFormation** before, you will be getting a screen like the following.
5. From the options given on the main **dashboard**, click on **Create stack**.



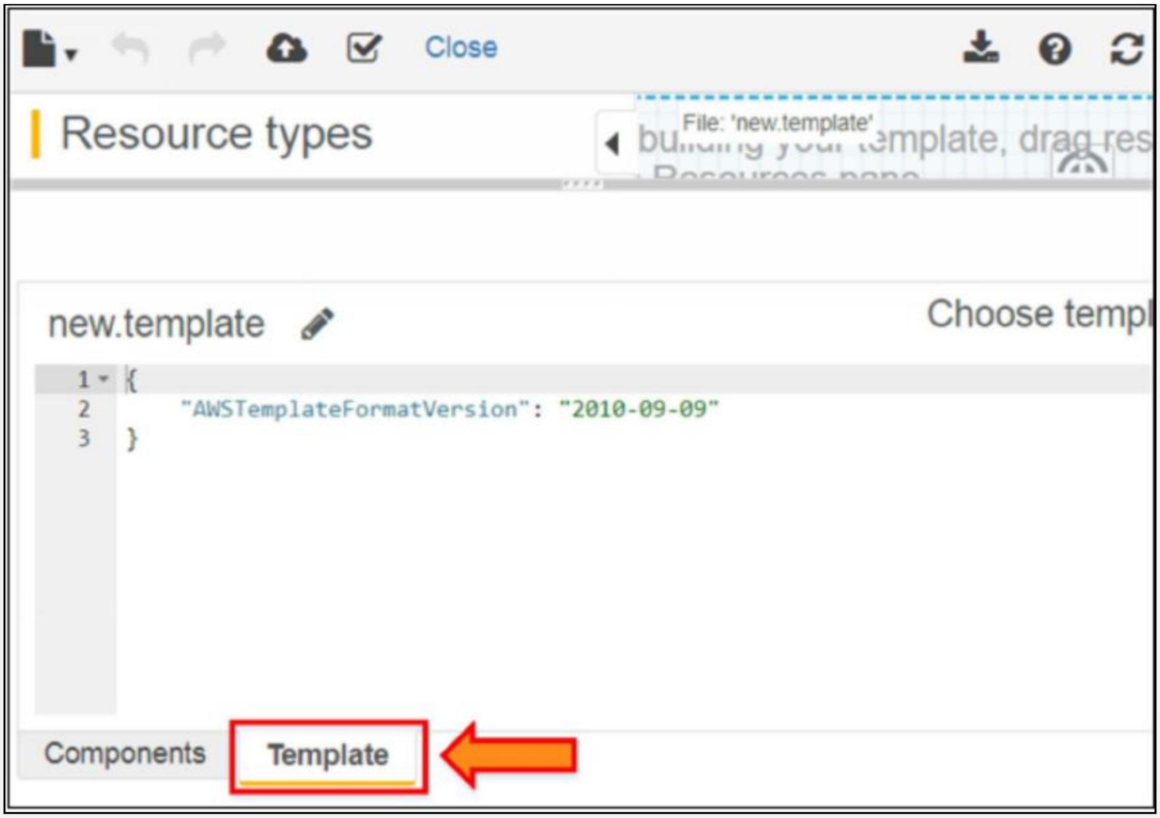
6. On the **Create stack** page, select **Create template in Designer**.



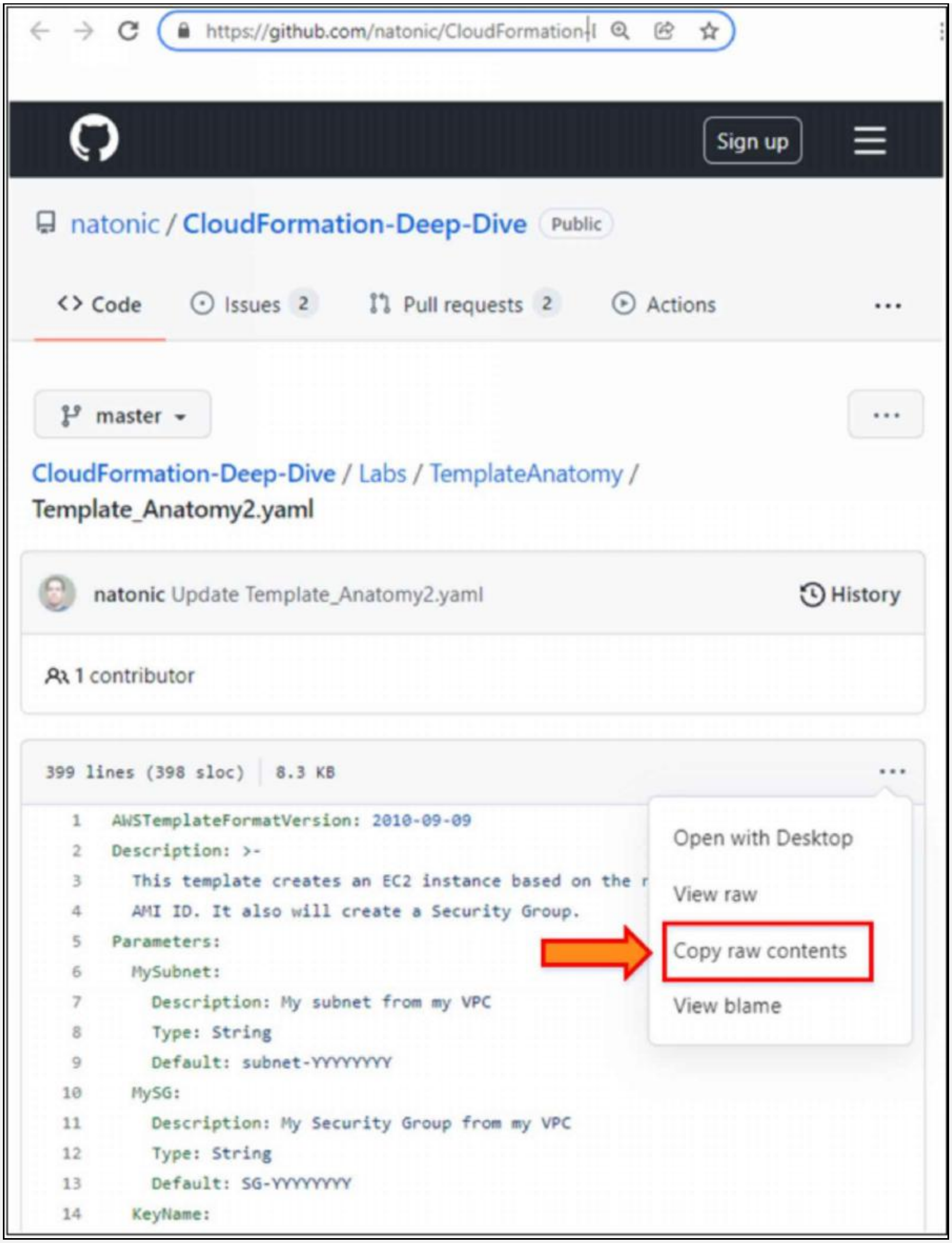
7. Then, click **Next**.



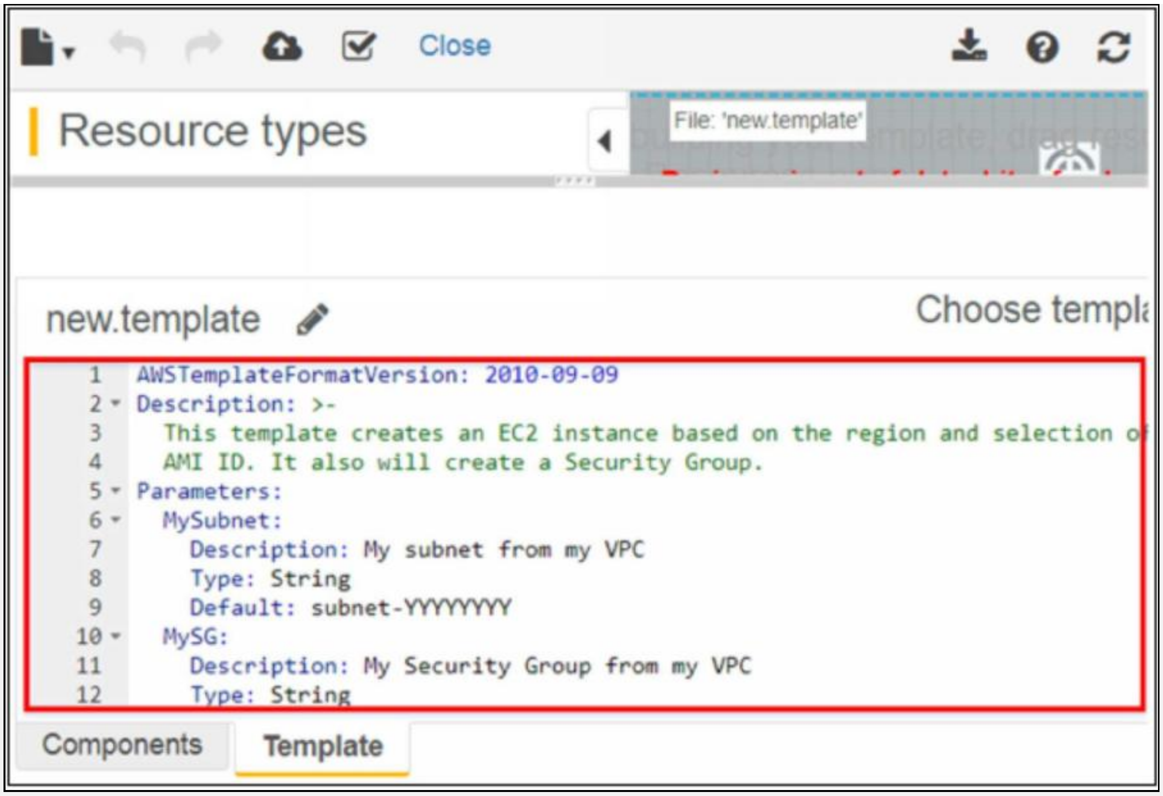
8. Click the **Template** tab at the bottom.



9. Copy everything in the **Template_Anatomy2.yaml** file (found on GitHub).

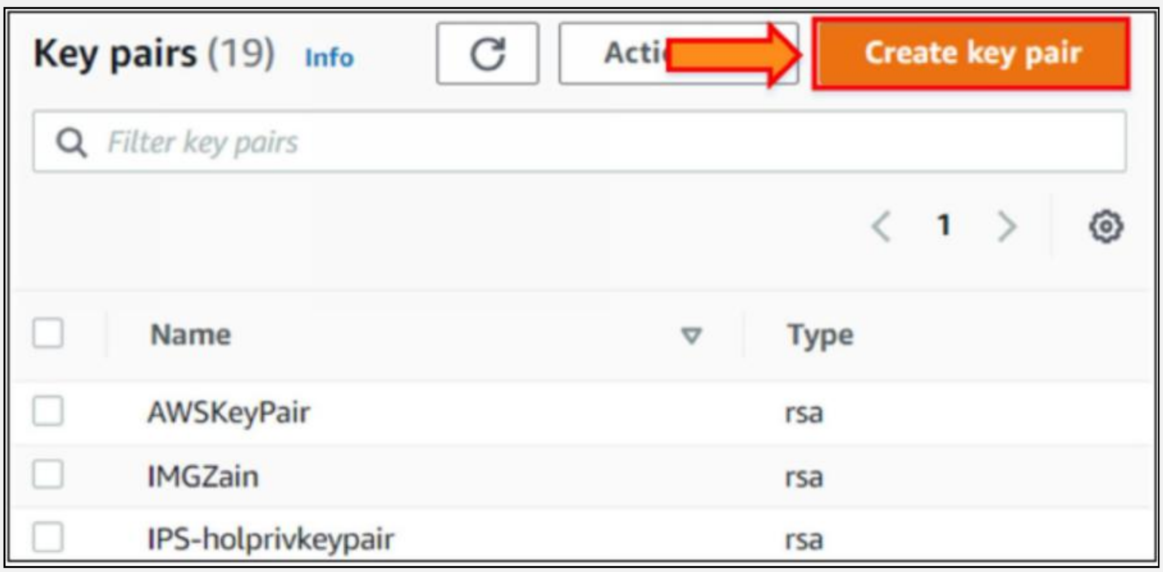


10. Then, paste it into the **Template** window.

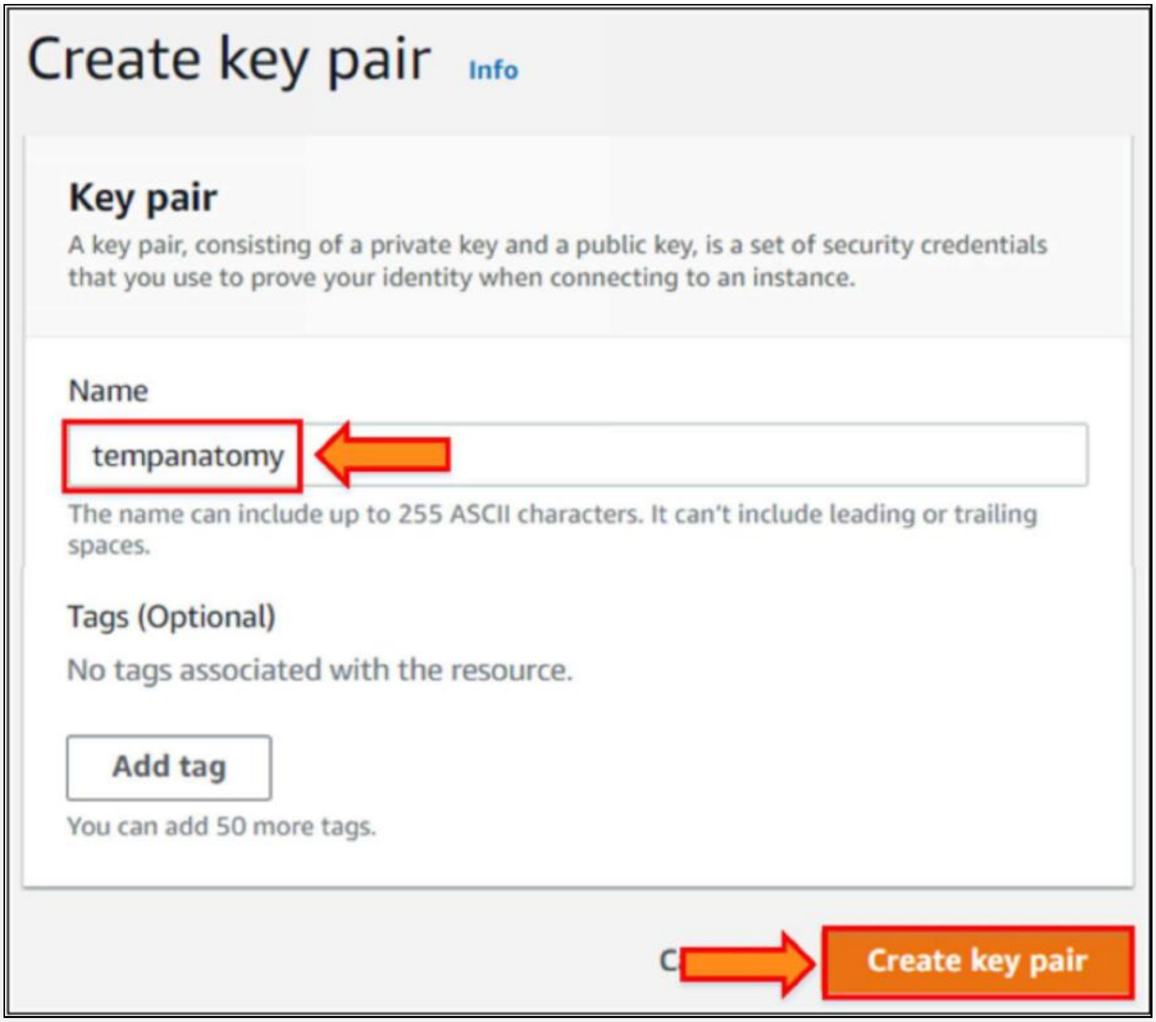


11. In a new browser tab, navigate to **EC2** > **Key Pairs**.

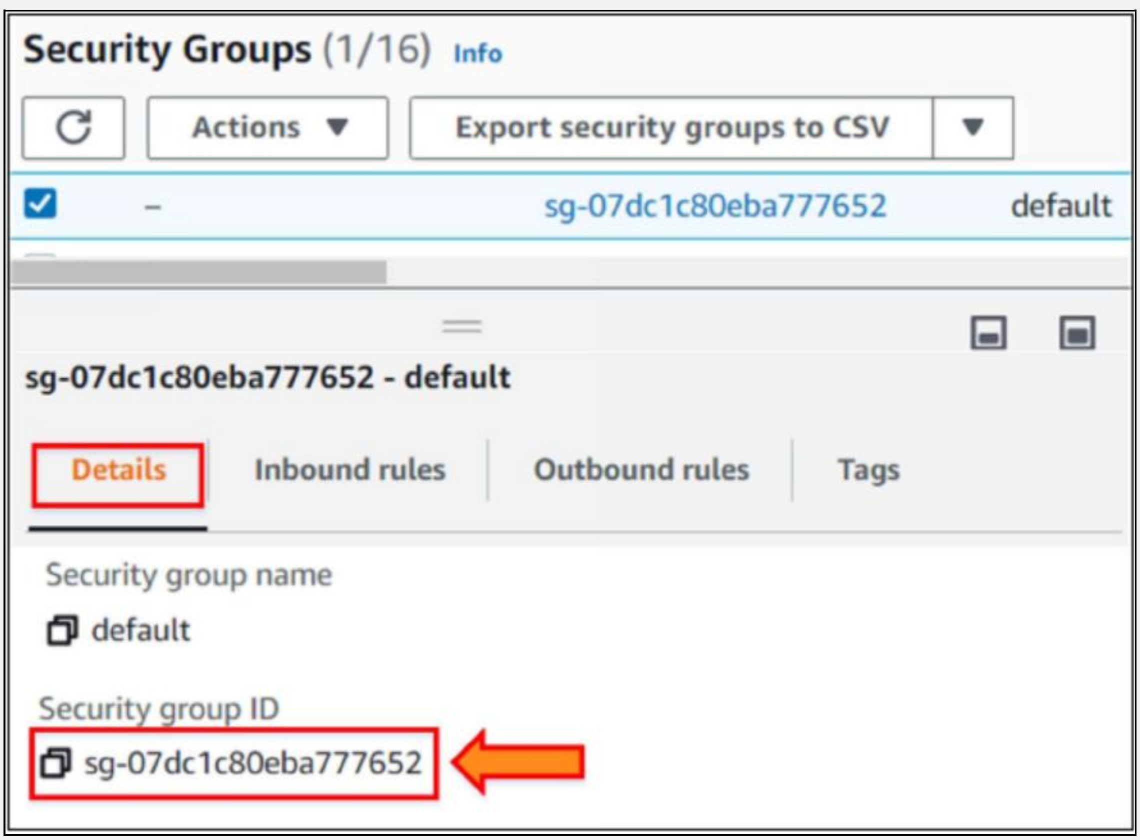
12. Click **Create Key Pair**.



13. Give it a **key pair name** of **tempanatomy**, and click **Create**.

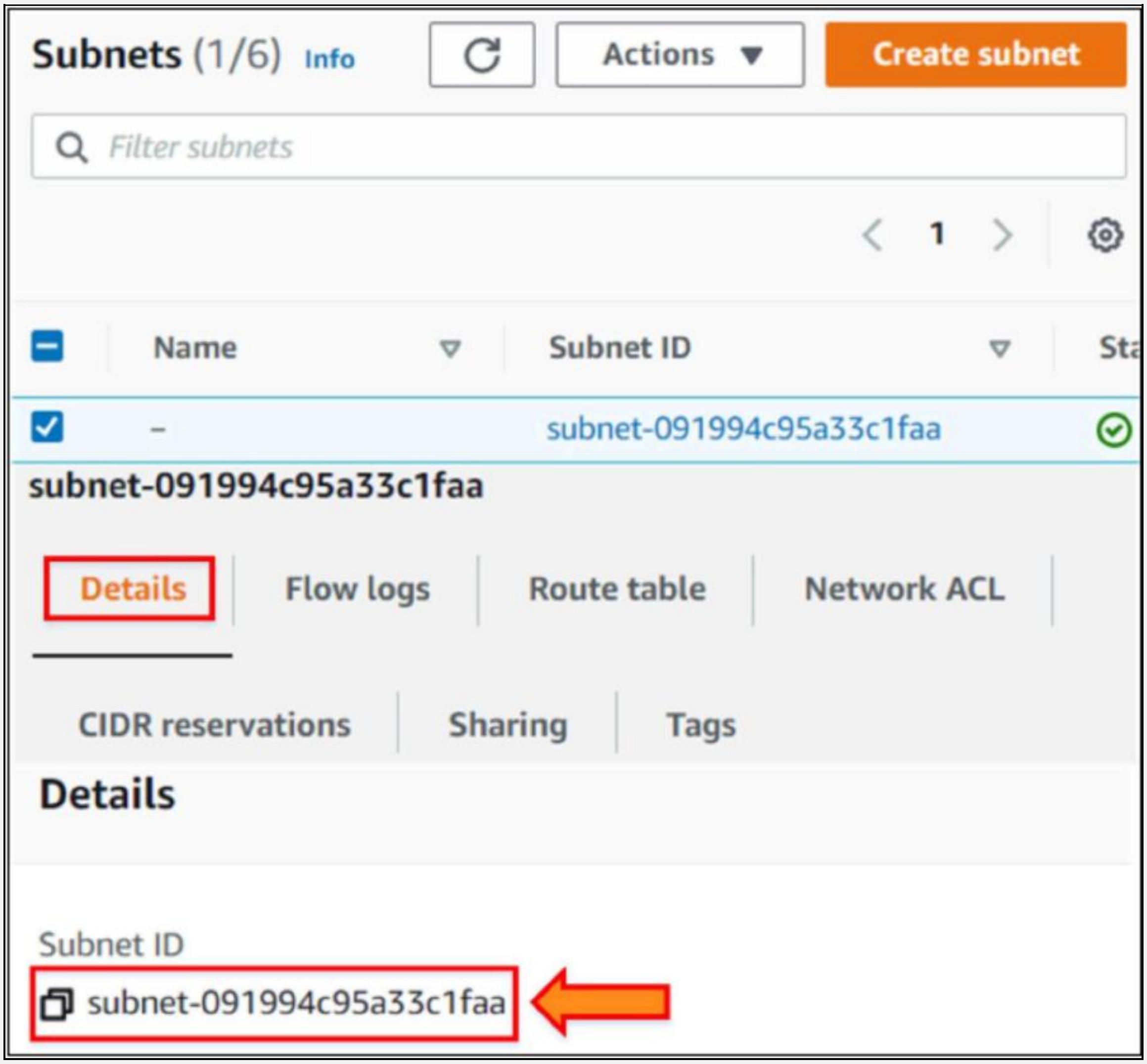


14. Click **Security Groups** in the left-hand side given menu.
15. Copy the **Security Group ID** and paste it into a text file; we will need it later.

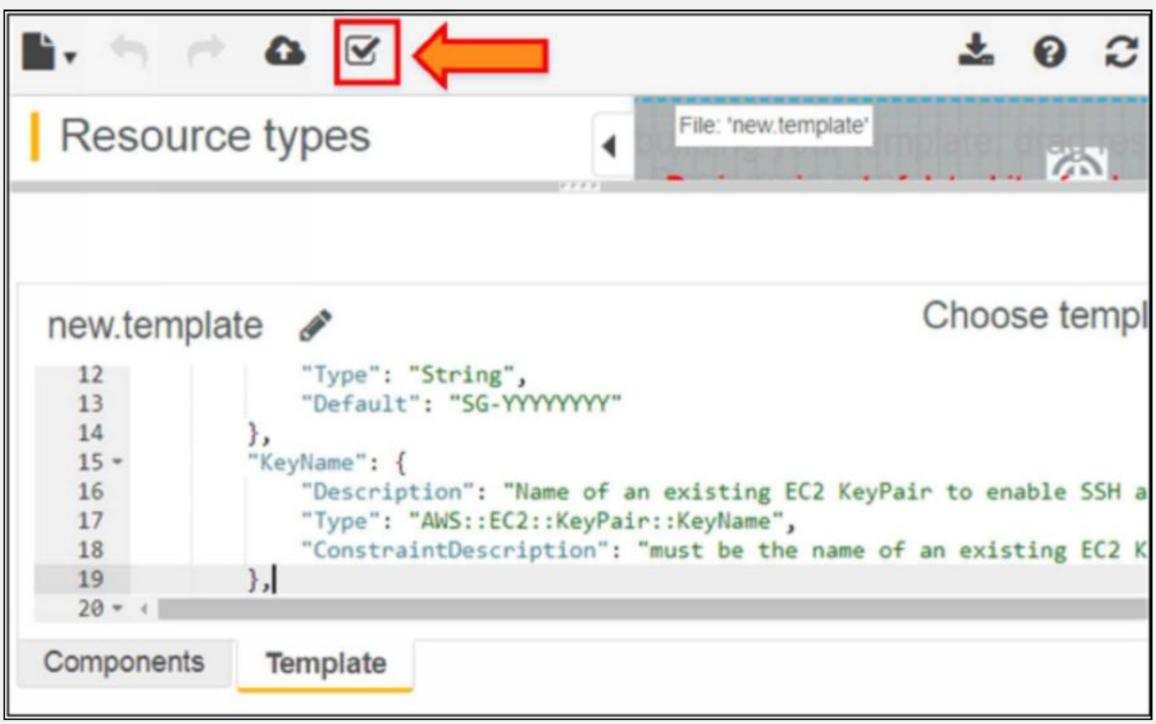


16. Navigate to **VPC** > **Subnets**.

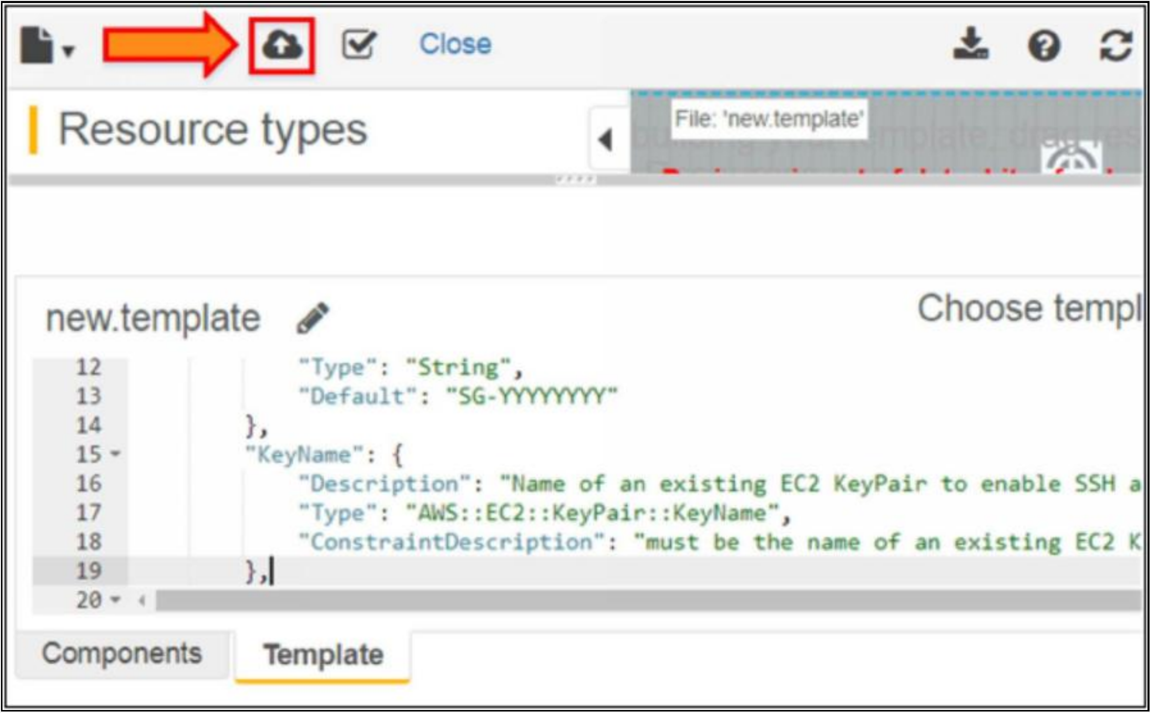
17. Select one of the listed **Subnets**, and copy its **Subnet ID**. Paste it into a text file since we will also need it later.



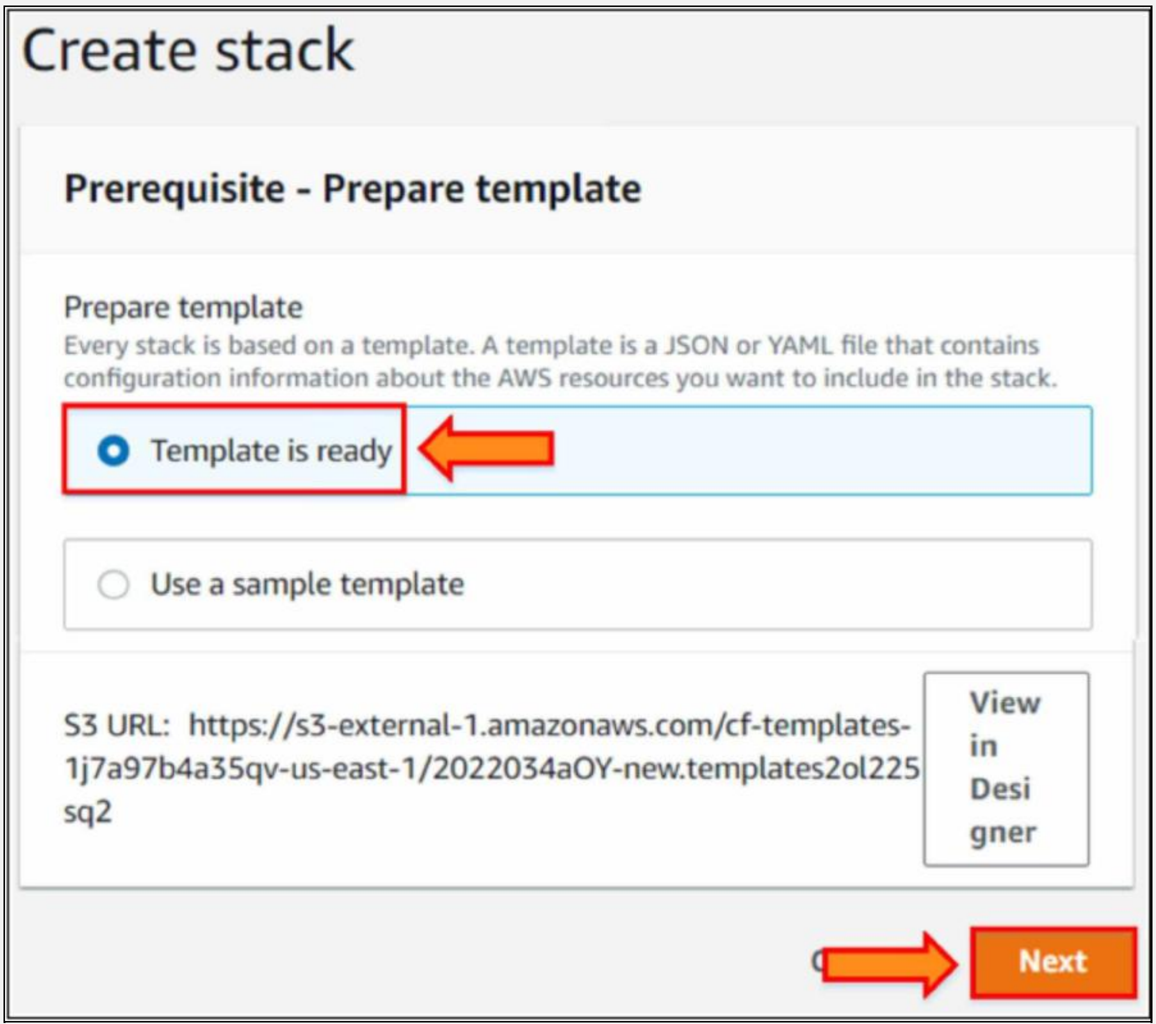
18. Back in the **CloudFormation template window**, click the **checkbox** at the top to **validate the template.**



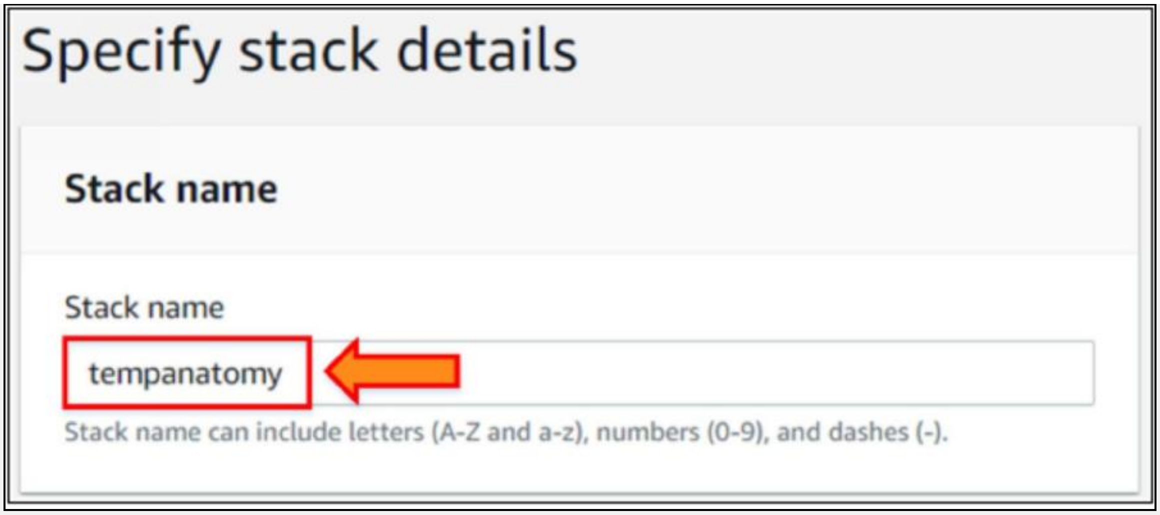
19. Then, click the **cloud icon** with the up arrow to **create the stack**.



20. It will be redirected to the **Create stack** page where the **Template is ready** already selected then click **Next**.

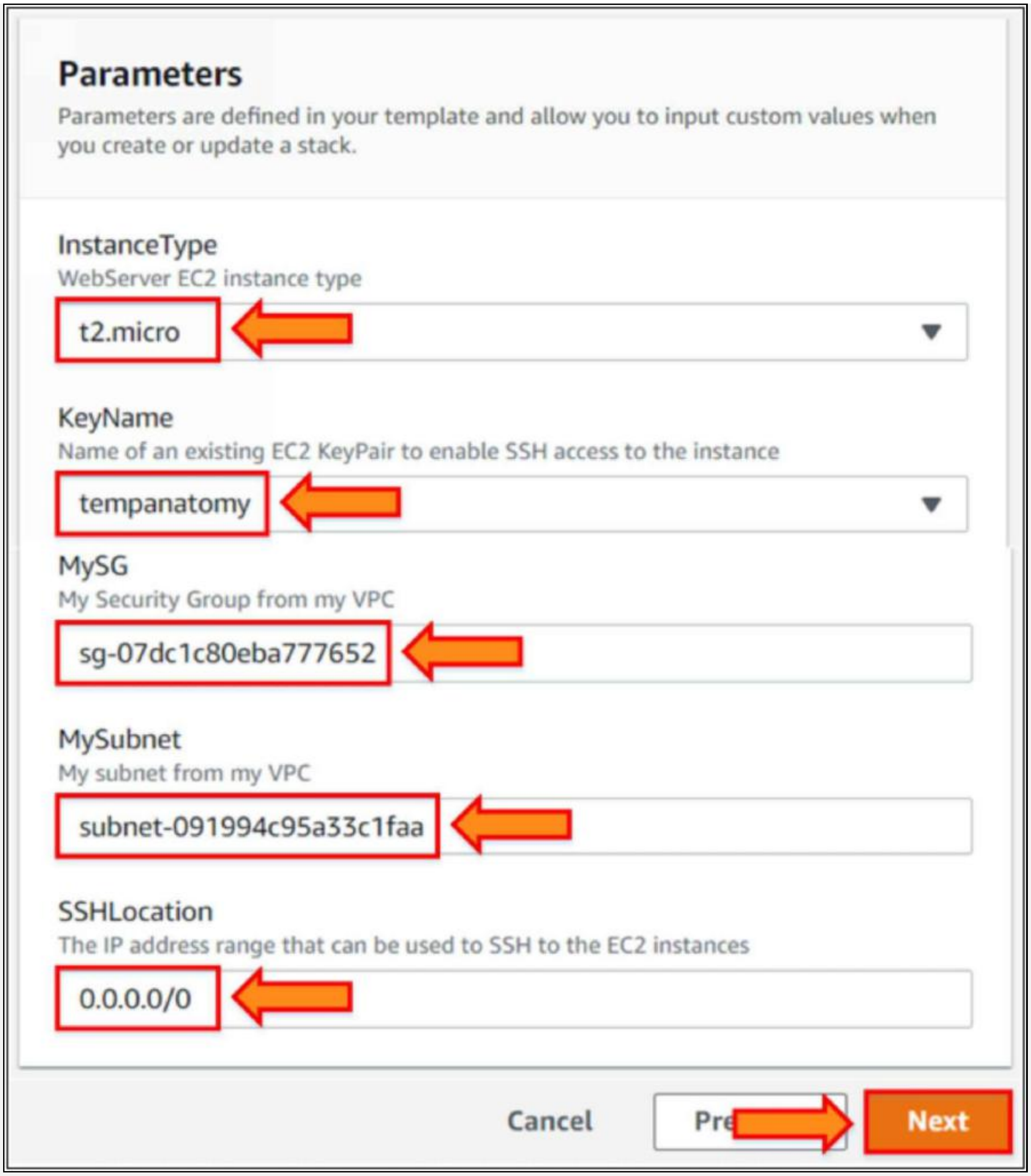


21. On the **Specify stack details** page, set a *Stack name* as **tempanatomy**.

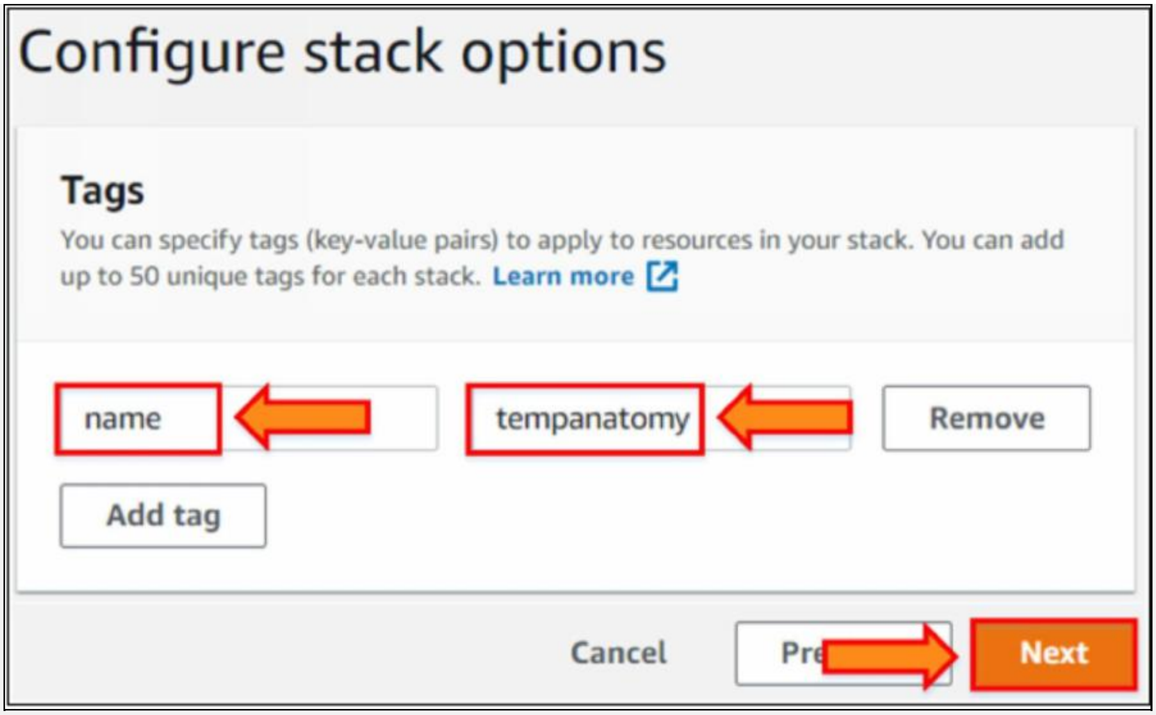


22. In the **Parameters** option, set the following values:
InstanceType: **t2.micro**
KeyName: **tempanatomy**
MySG: Paste in the security group ID you copied earlier
MySubnet: Paste in the subnet ID you copied earlier
SSHLocation: **0.0.0.0/0**

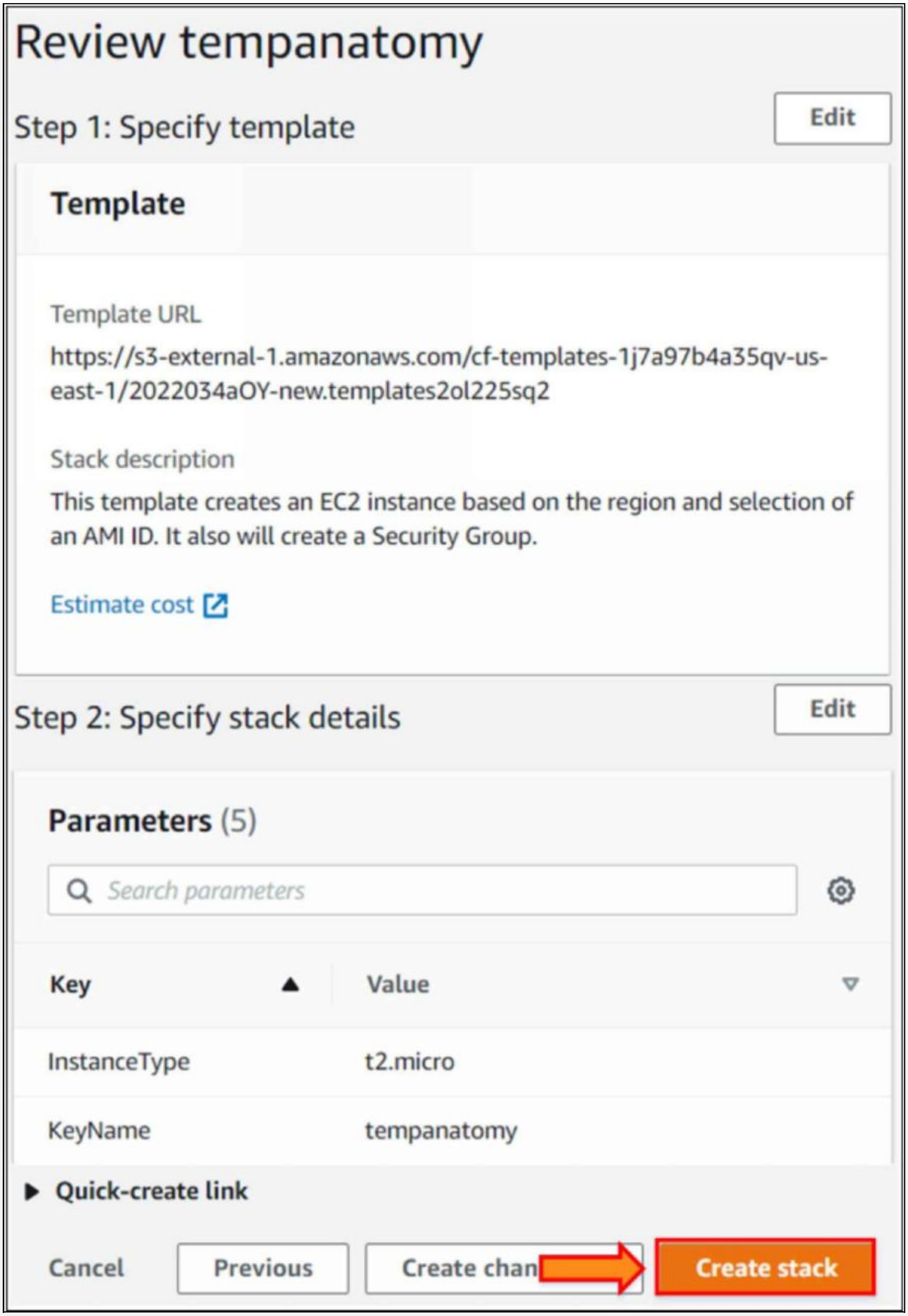
23. Click **Next**.



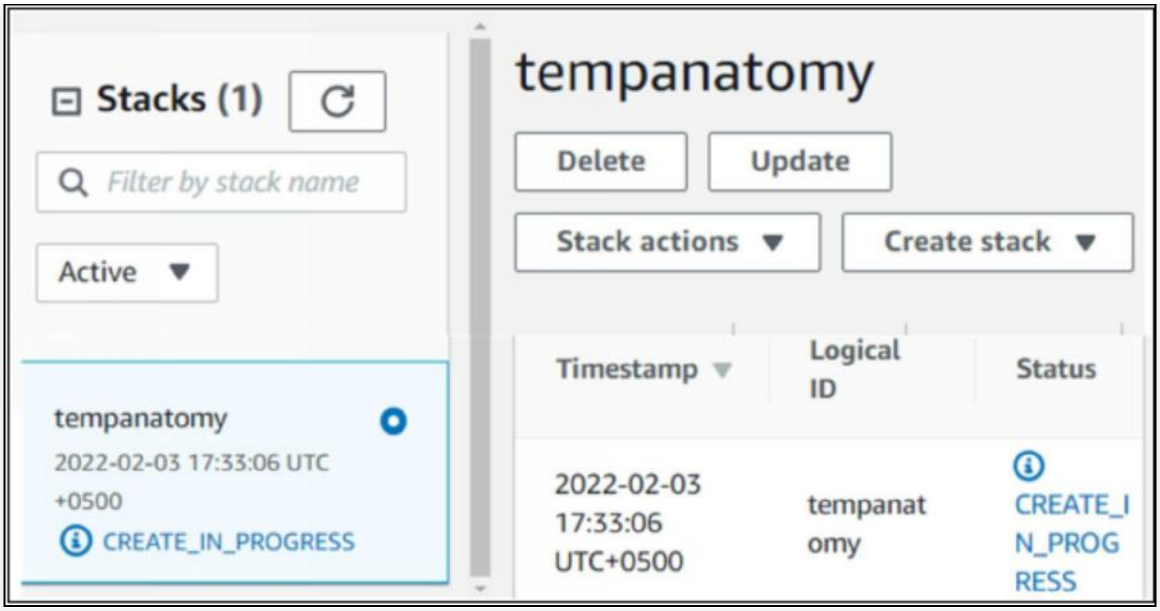
24. On the stack options page, set the *Key* as **name** and *Value* as **tempanatomy**.
25. Click **Next**.



26. Review the provided details and click **Create stack**.

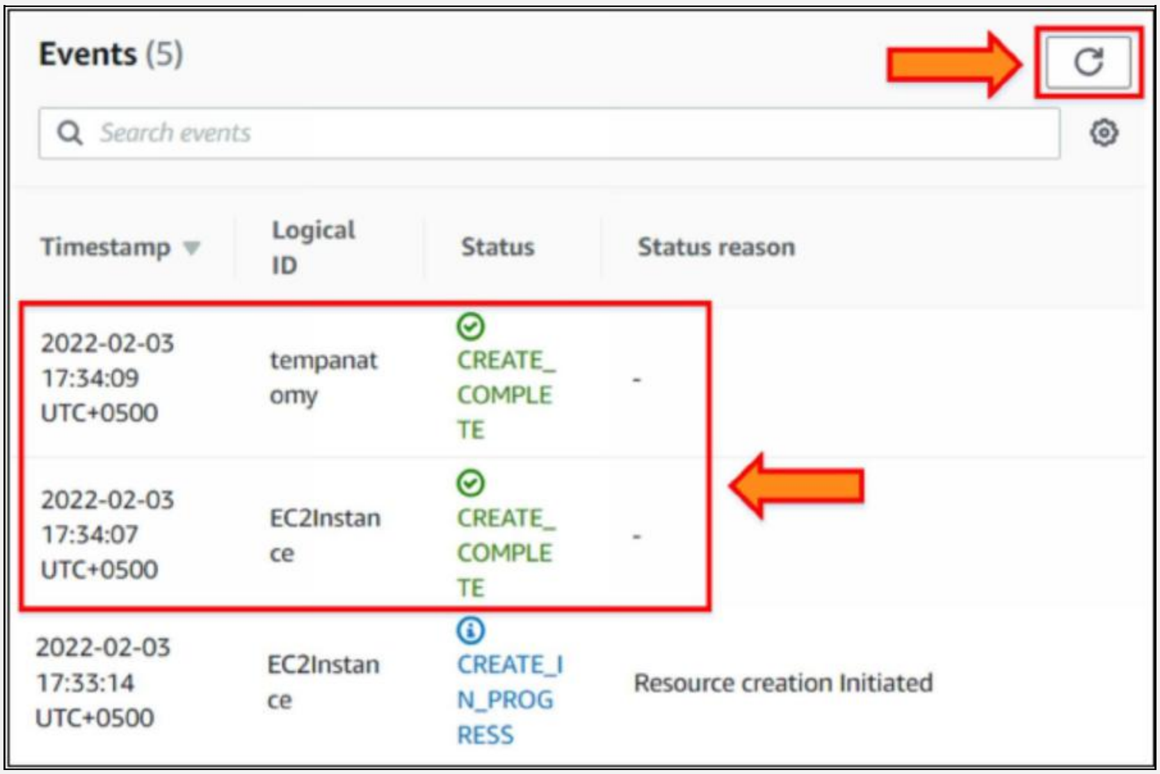


27. It will take a few minutes for it to be created fully.

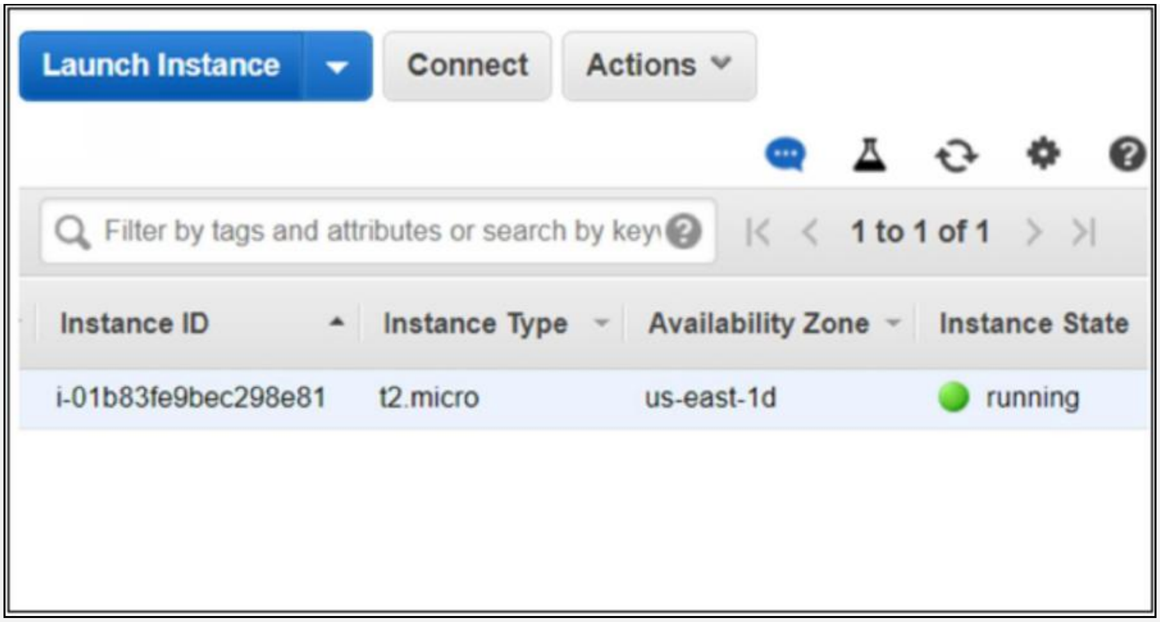


Step 2: Observe the Stack Events

1. When the **tempanatomy** stack is created, click **refresh,** and you will see an **EC2** is also **created**.



2. Go to the EC2; you will see EC2 instance is running there.

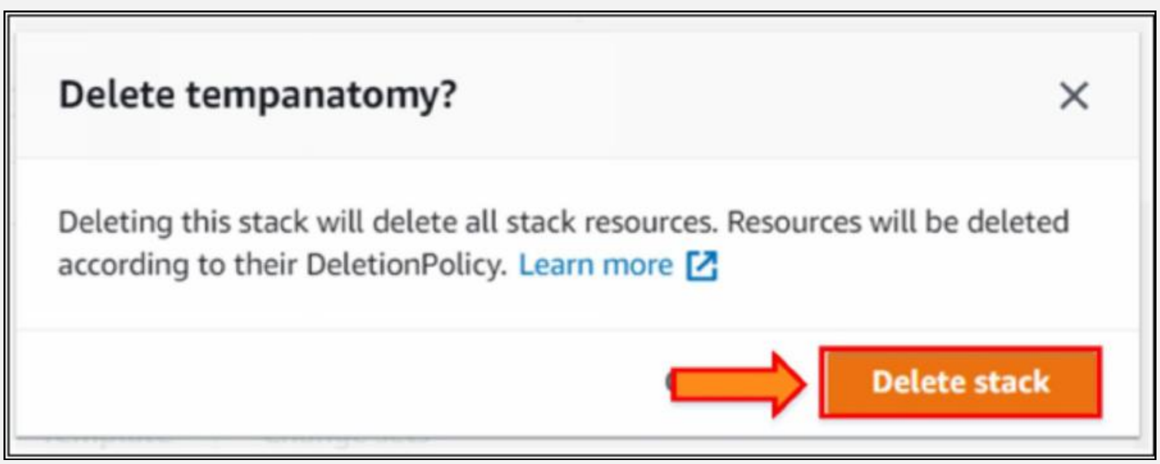


3. EC2 is running because of the **template** we are using in the stack **template**. If we delete the **Stack**, the **EC2 instance** is also **deleted**.

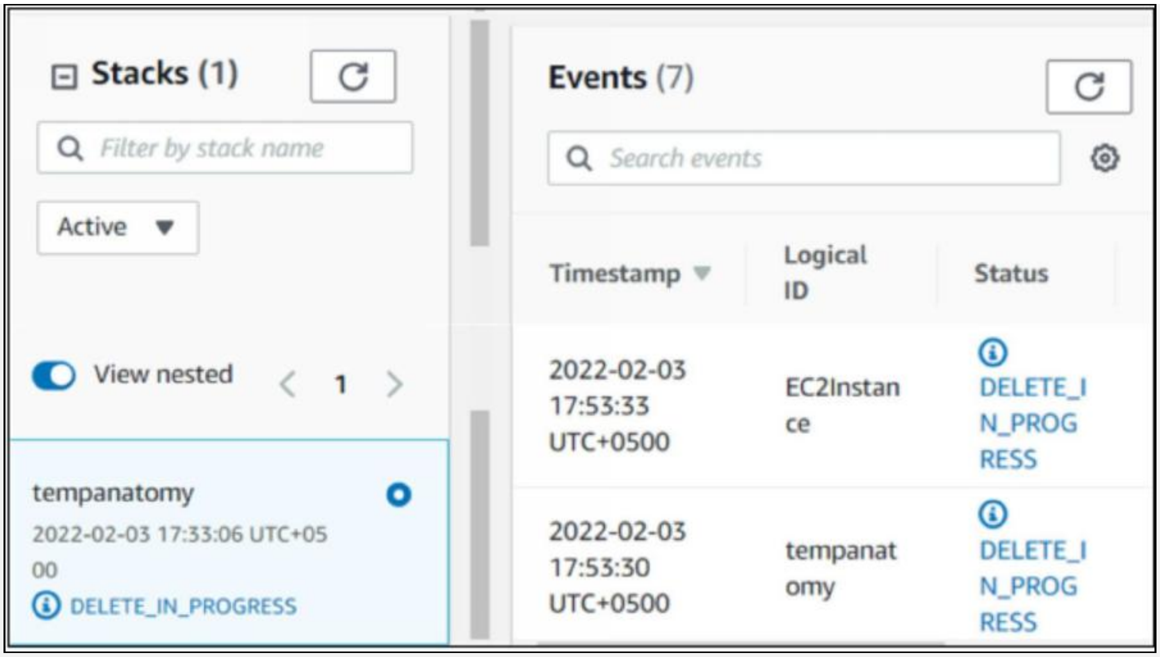
Step 3: Delete a CloudFormation Stack

1. Once it is created, click **Delete** at the top.

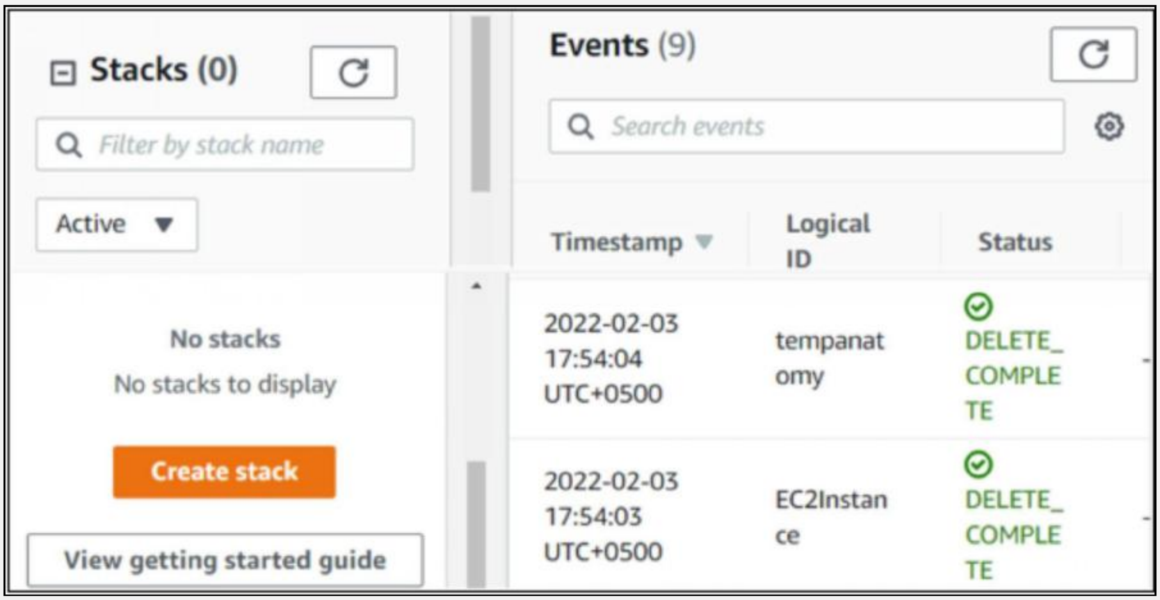
2. In the confirmation dialog, click **Delete stack**.

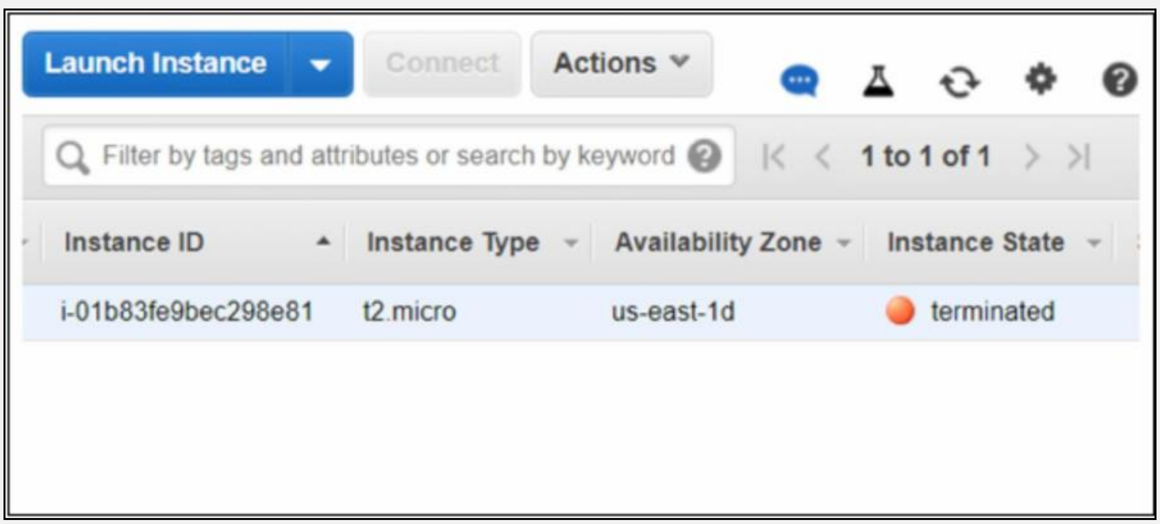


3. Monitor the deletion process by watching the **Events** tab.



4. When **Stack** is deleted, you will see an **EC2 instance** also **deleted**.





Lab 11: Creating a TensorFlow Image Classifier in AWS SageMaker

Certifications	Specialty
Cloud Provider	AWS
Time Required	1 Hr.
Tags	SageMaker, AWS

Lab Prerequisites

- Familiarity with basic AWS Cloud Computing concepts and terminology
- An AWS account with an active subscription

Service Introduction

TensorFlow is a popular open-source Machine Learning (ML) framework for creating neural networks and machine learning models. To speed up the training, such models are generally trained on many GPU instances, resulting in longer training times and model sizes of several gigabytes. These models are then used in production to make conclusions once trained. Workloads might be synchronous, asynchronous, or batch-based. Those endpoints must be highly scalable and robust to process zero to millions of requests.

Case Study Enterprise Financial Service– Kasasa

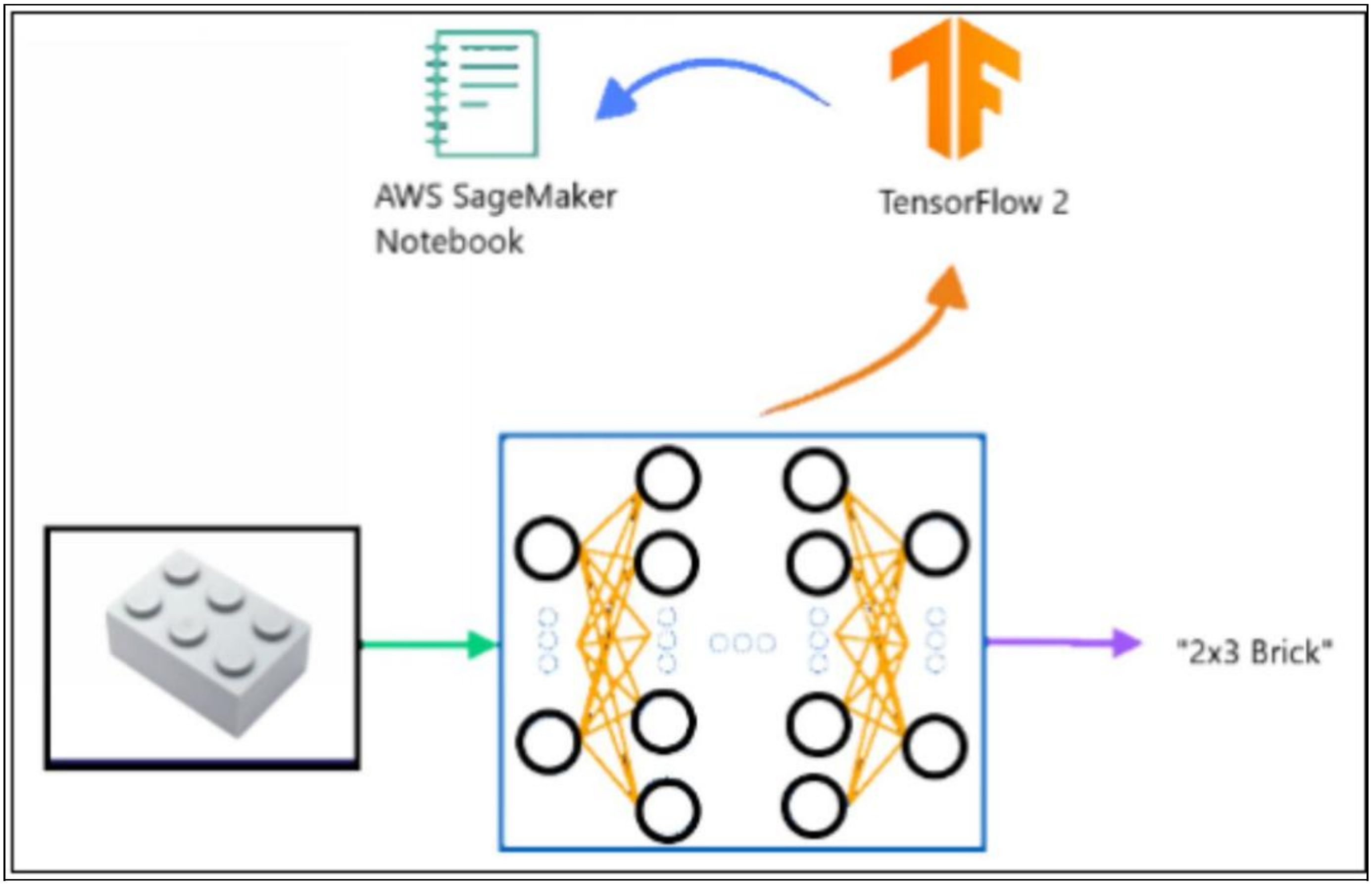
Business Challenge

You work as an AWS architect in Kasasa. They gave the task of building a machine that could reliably sort lego bricks. You have got a lot of LEGO bricks to sort through. It will take even longer to do it by hand, and you will get a lot of eye strain from gazing at the small parts.

Proposed Solution

The solution is you use AWS services to automate your task, and you will need to teach a machine to recognize the bricks and inform you (or, better yet, a robot) which bin each piece belongs in. You already have a lot of LEGO block images, which you will utilize to teach the model you are using.

Lab Diagram



Implementation Steps

1. Navigate to the Jupyter Notebook.
2. Load and Prepare the Data.
3. Train the TensorFlow Model.
4. Test and Analyze the Model.
5. Make a Batch Prediction Using the Testing Data.

Solution

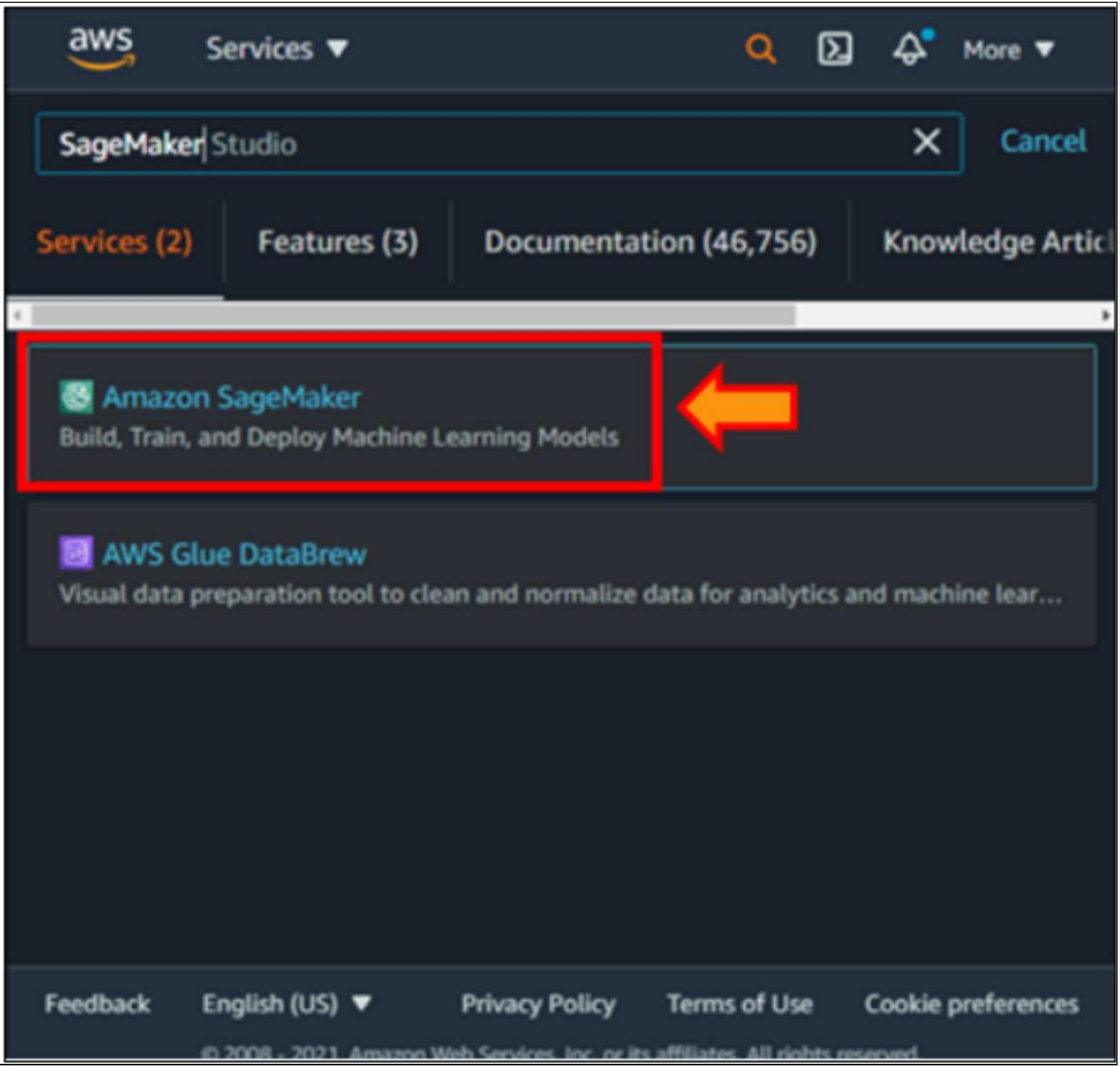
Step 1: Navigate to the Jupyter Notebook

Note: Before starting the lab, you must have a Jupyter notebook. You can create it by following the steps given in the link:

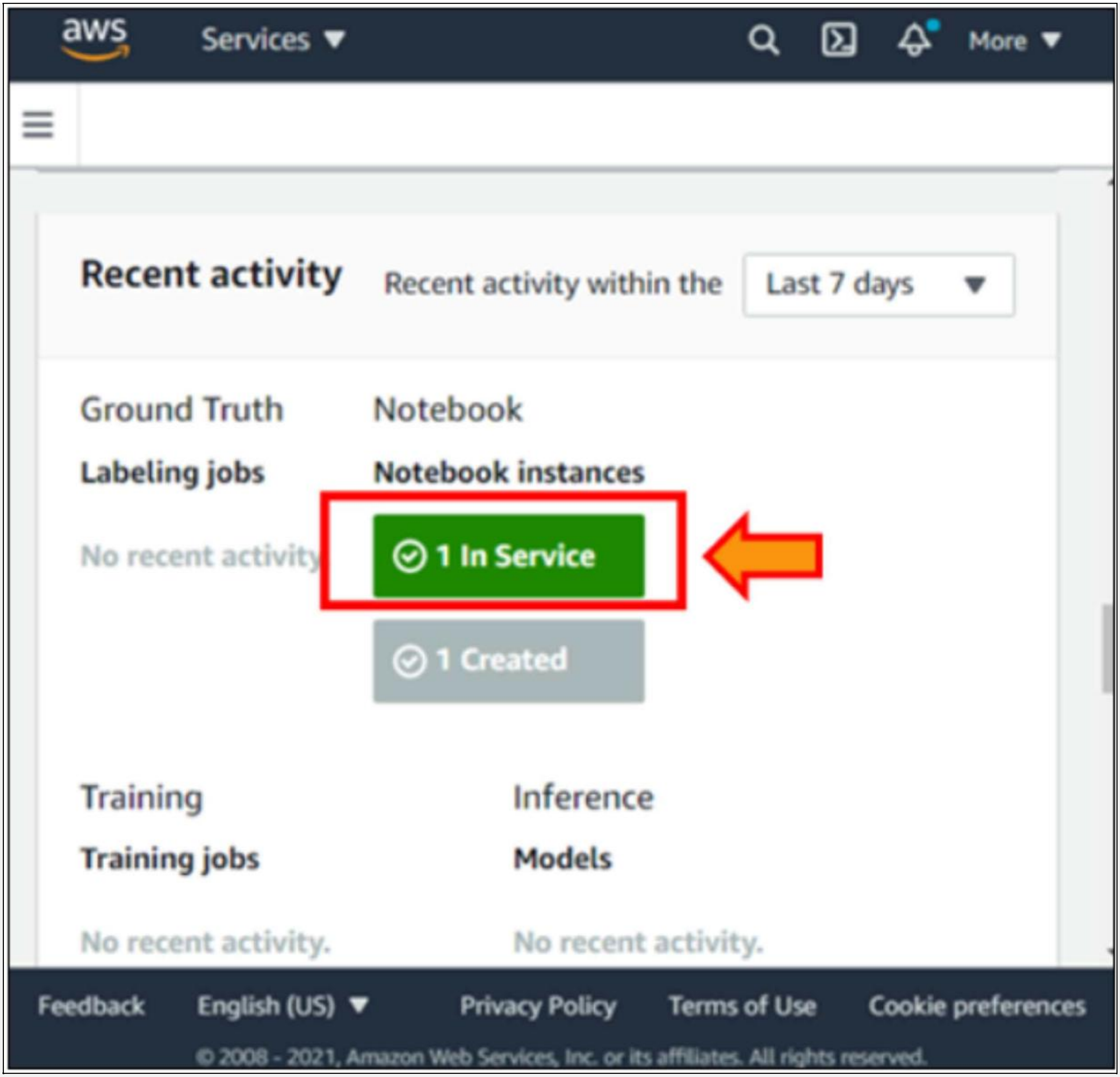
https://docs.aws.amazon.com/dlami/latest/devguide/setup-jupyter.html

1. Log in to the **AWS Console**.

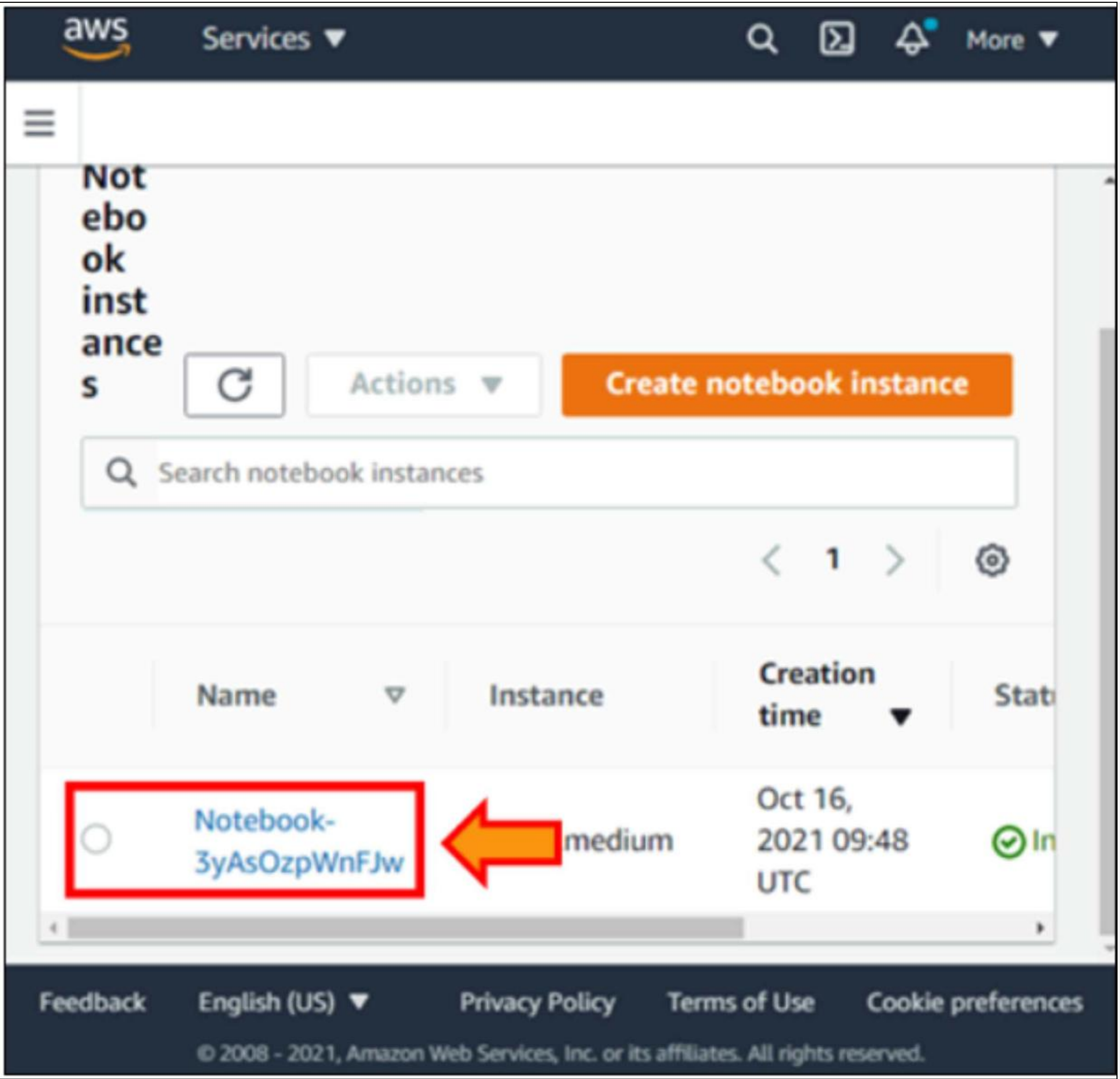
2. Go to **Services** and click on **SageMaker.**



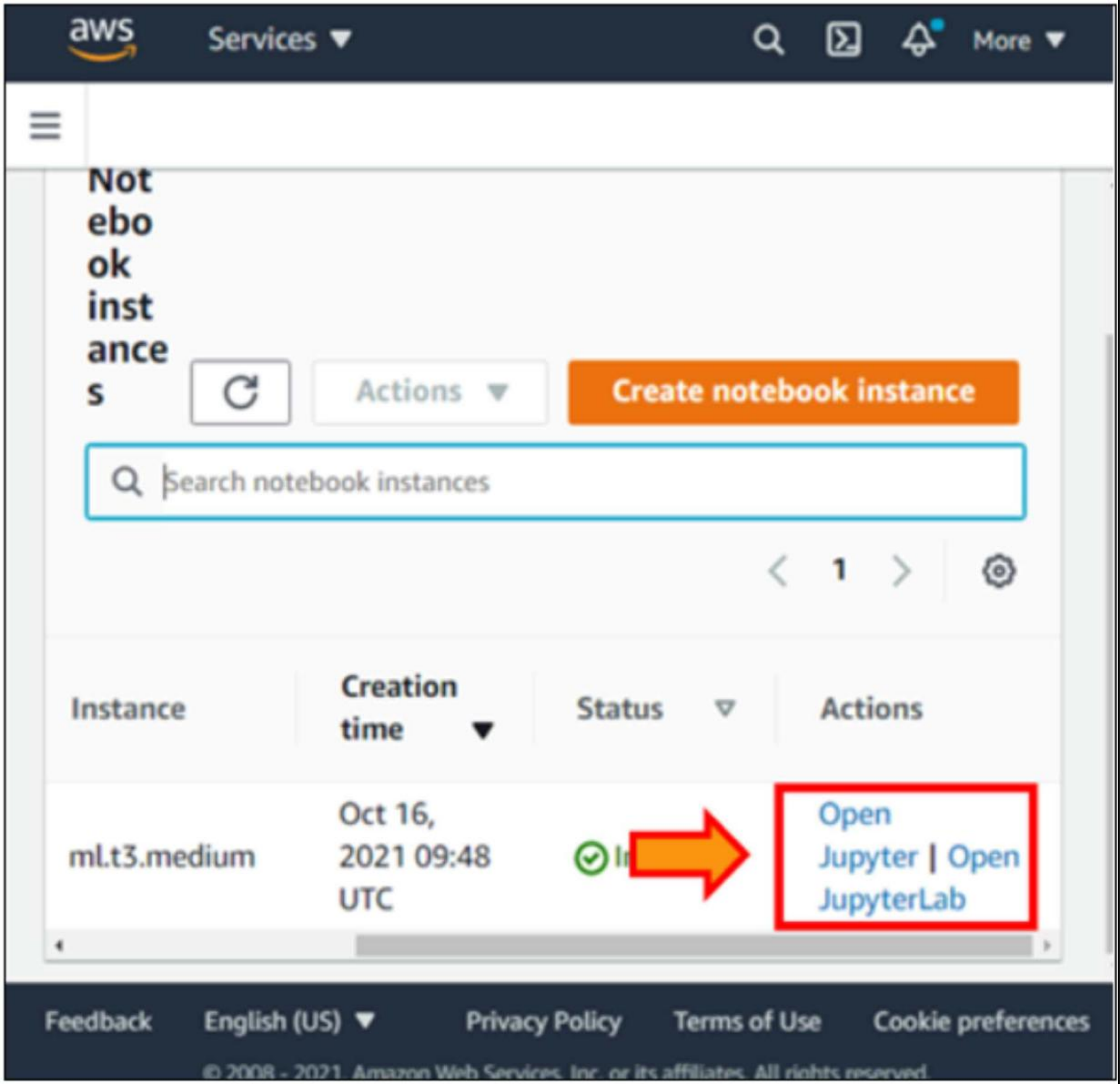
3. Click on **In-Service.**



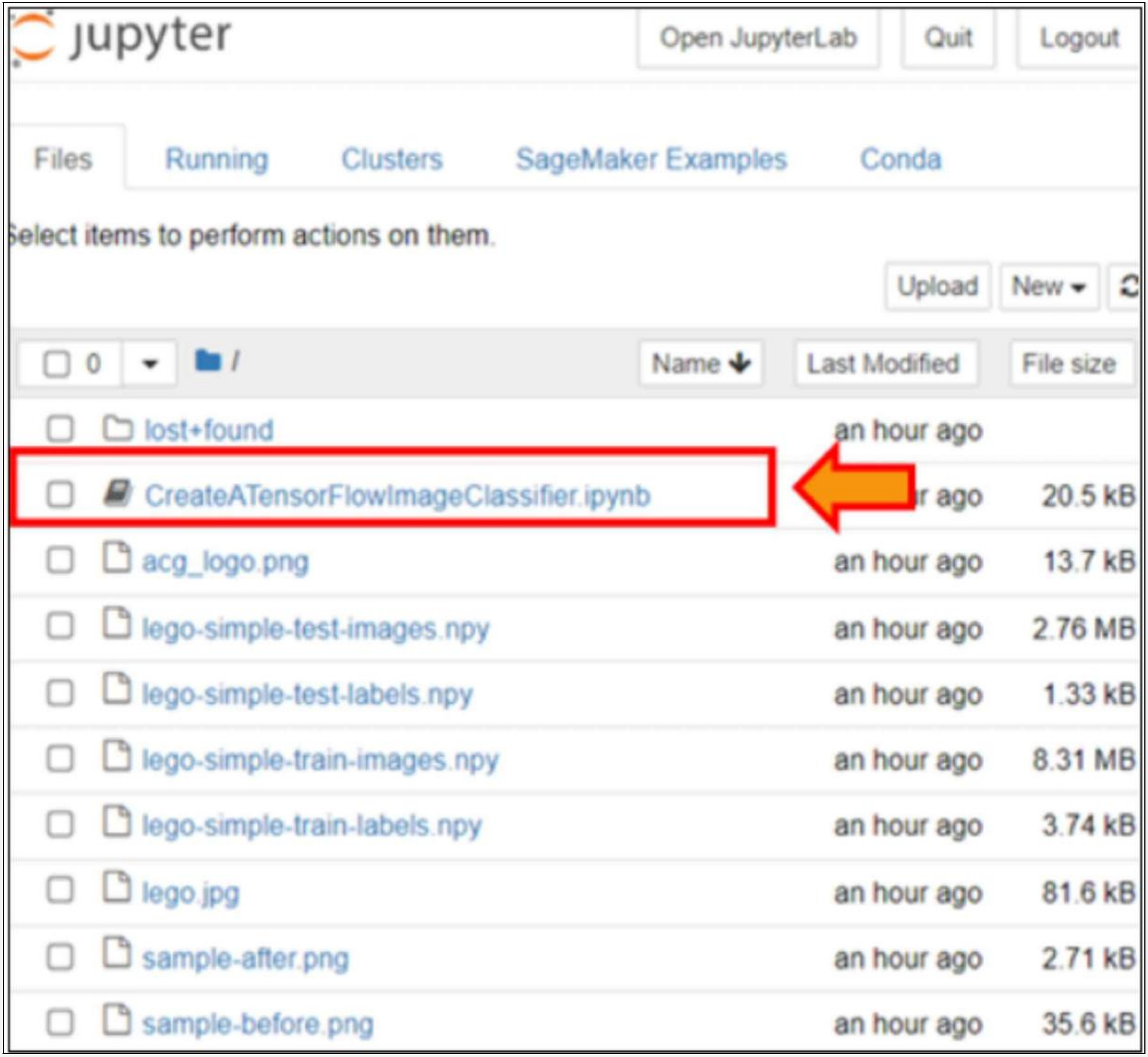
4. Click on the **Notebook instance.**



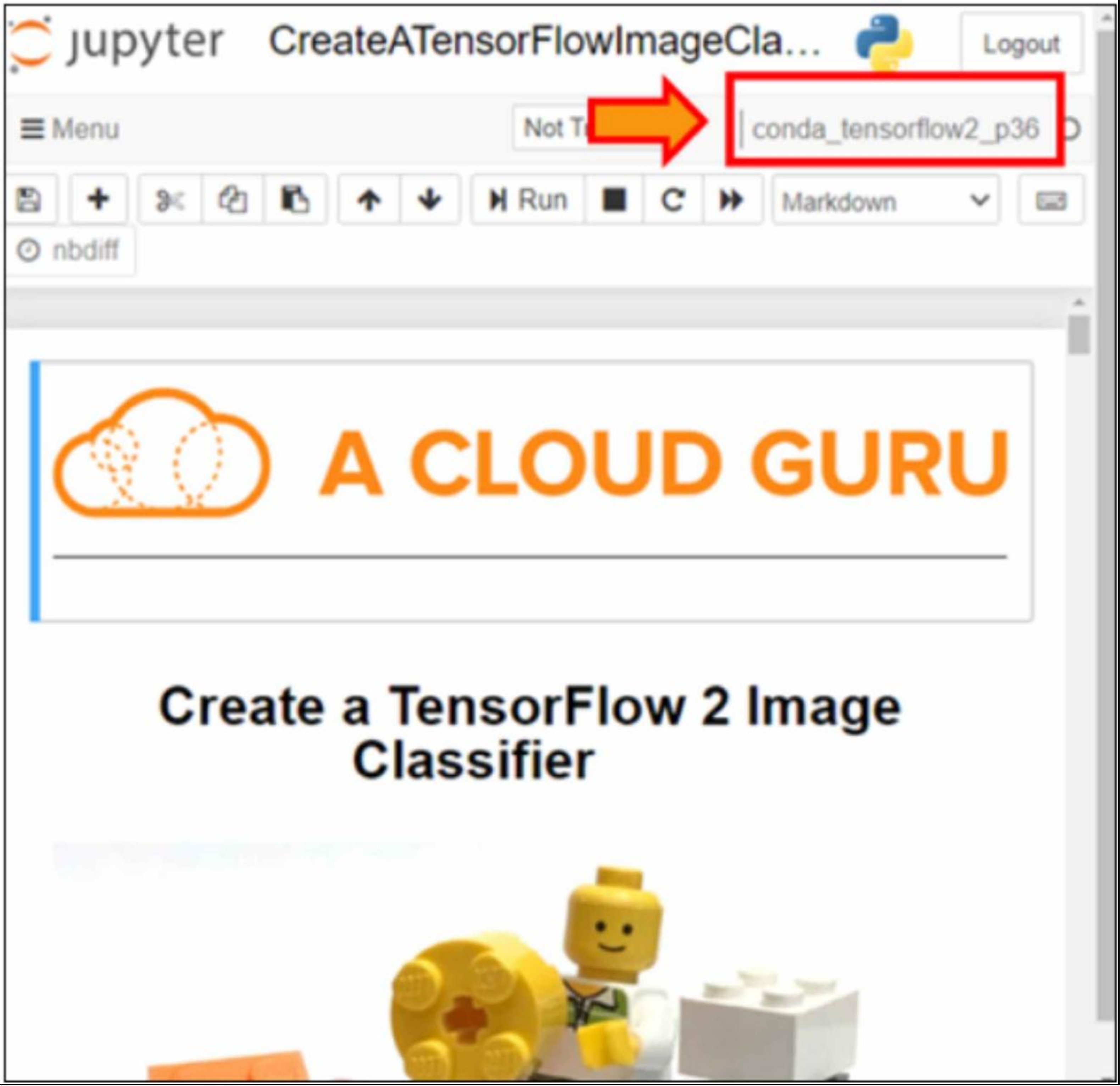
5. Click on **Open Jupyter.**



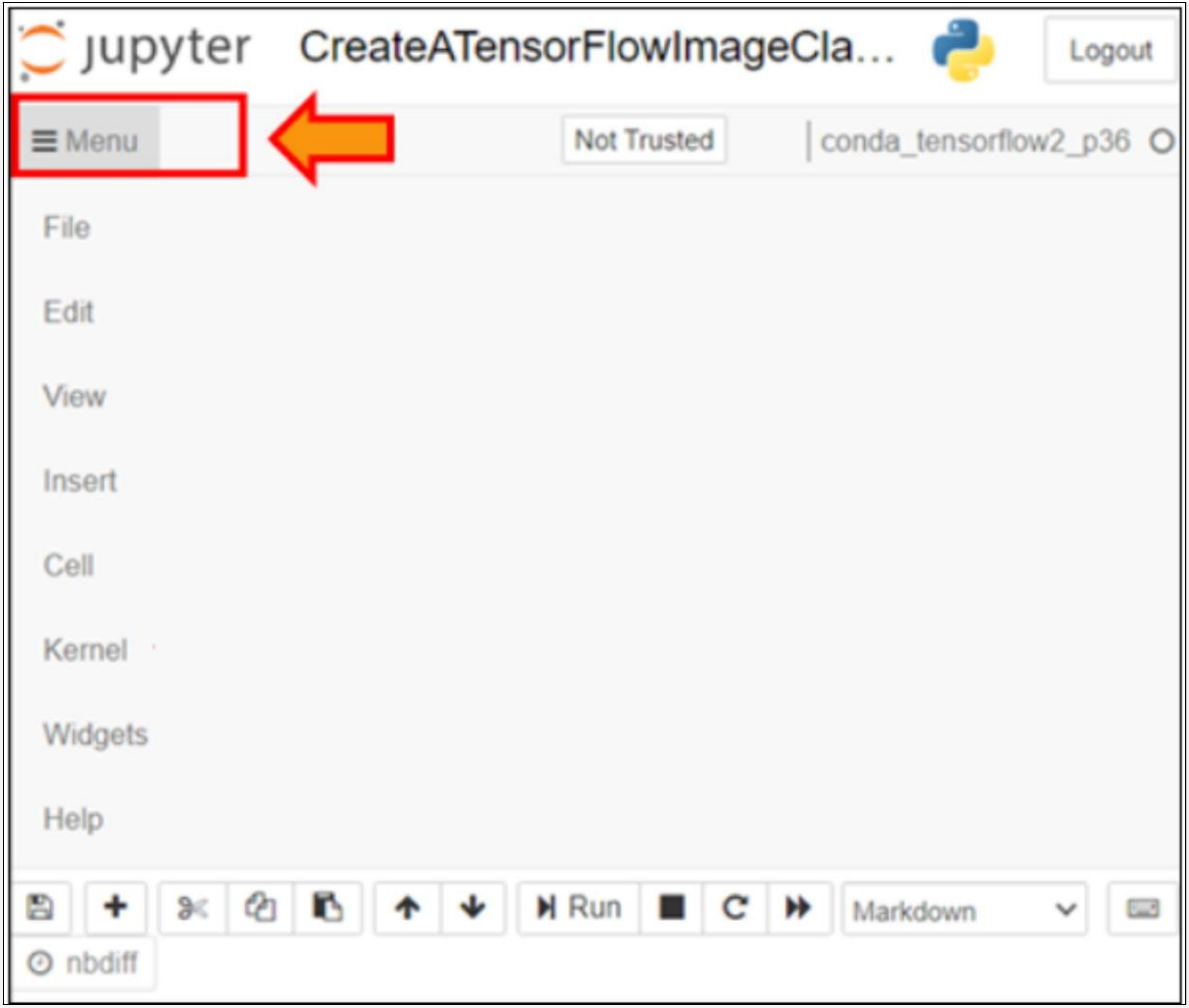
6. Click on **CreateATensorFlowImageClassifier.ipynb.**



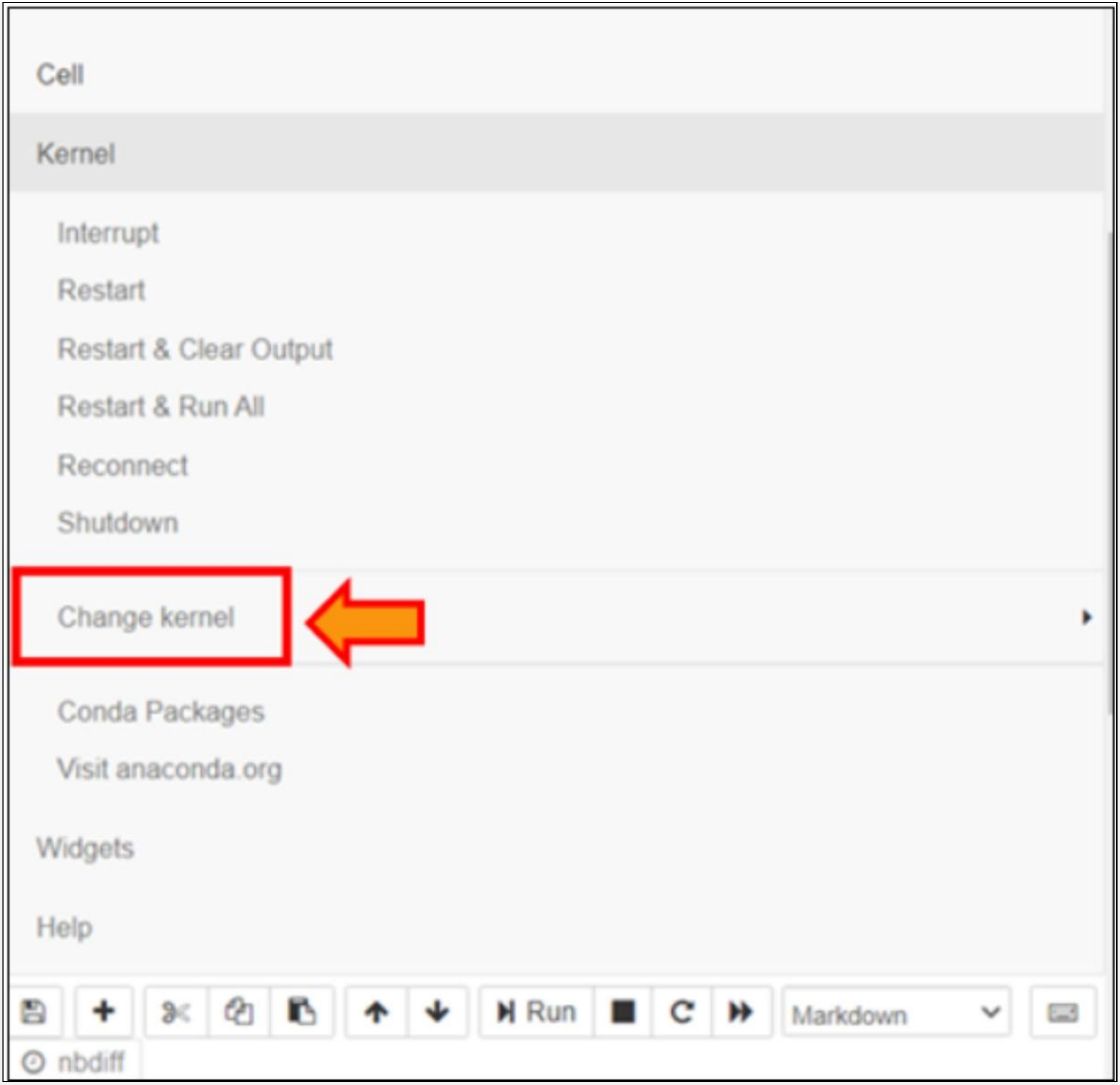
7. Make sure our kernel will support **TensorFlow 2.**



8. If the kernel is not set to TensorFlow 2, go to **Menu** and click on **Kernel.**

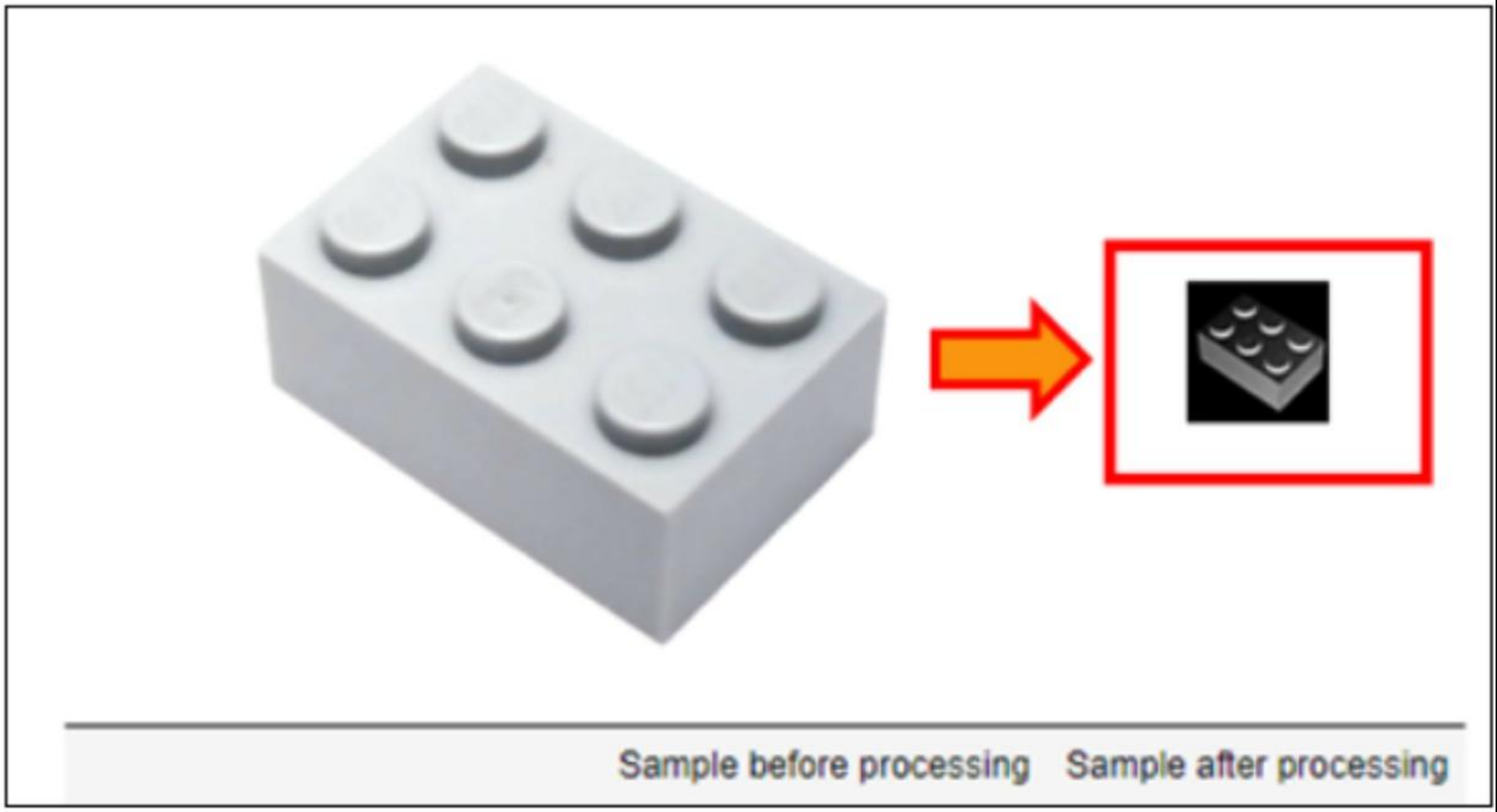


9. Click on **Change Kernel.**



10. We have a collection of photographs of different LEGO bricks taken from various perspectives. There are 600 pictures in all, all tagged with the brick type.

Each image has been enhanced. Increasing the contrast, sharpening, eliminating the color, inverting the color, and shrinking the image were all part of the process. We put them all into data arrays to make importing the photos into the notebook easier. The folders lego-simple-train-images.npy and lego-simple-test-images.npy include these pictures.



11. Our first step is to import all the necessary libraries. We need to ensure we are using the right version of tensor flow; hence, we will print the version as well.

```
In [*]: import tensorflow as tf
        from tensorflow import keras

        import numpy as np
        import matplotlib.pyplot as plt

        print(tf.__version__)
```

```
In [1]: import tensorflow as tf
        from tensorflow import keras

        import numpy as np
        import matplotlib.pyplot as plt

        print(tf.__version__)

        2.1.3
```

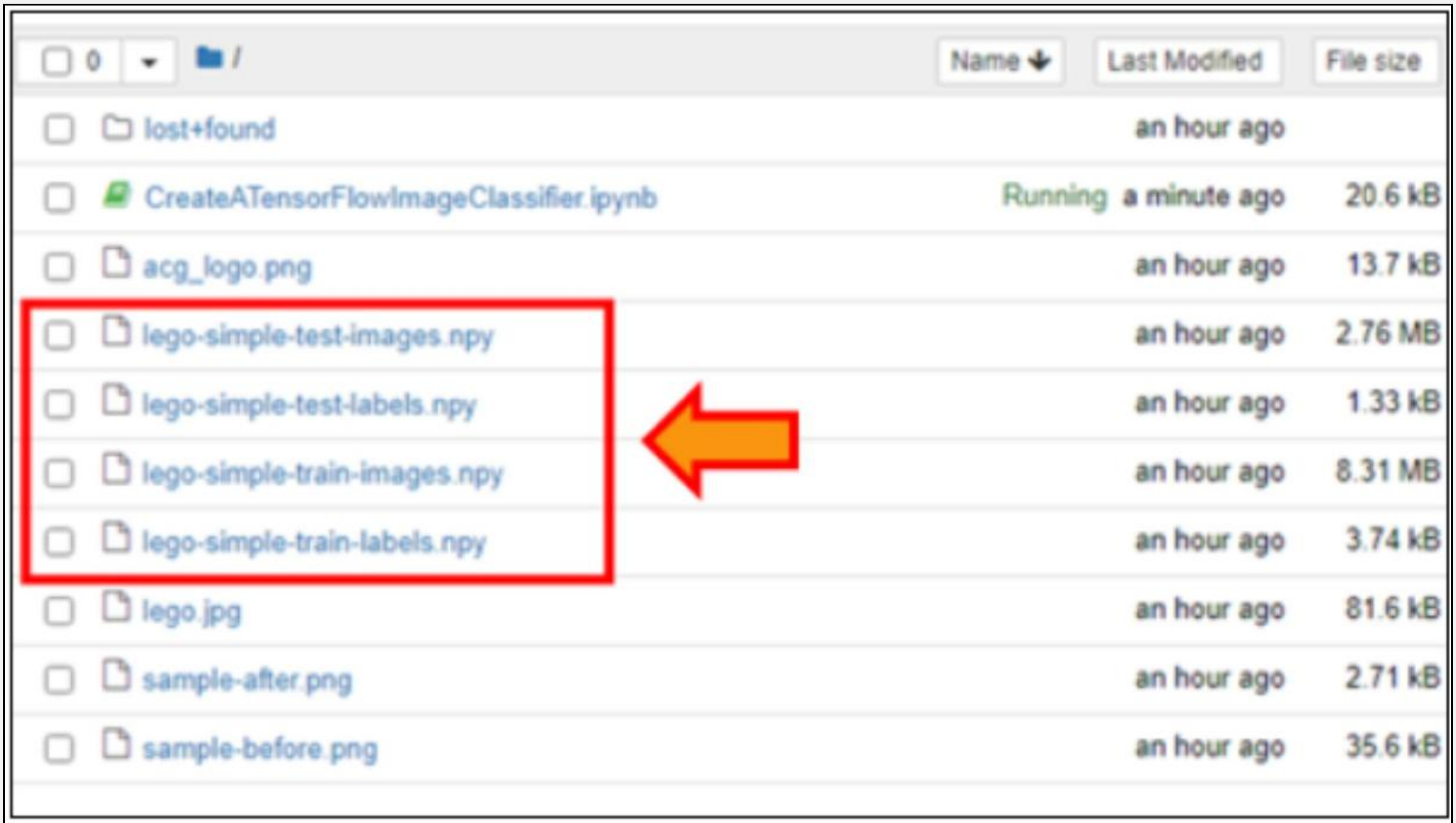
Step 2: Load and Prepare the Data

1. We have a dataset built from a collection of LEGO block pictures. A total of four data arrays have been stored in files as NumPy arrays. The data is first loaded into arrays in the notebook.

- `lego-simple-train-images.npy` - Training images, around 80% of the data collected.
- `lego-simple-train-labels.npy` - A list of integer labels identifying the classes of the training images.
- `lego-simple-test-images.npy` - Testing images, around 20% of the data collected.
- `lego-simple-test-labels.npy` - A list of integer labels identifying the classes of the testing images.

```
In [ ]: train_images = np.load('lego-simple-train-images.npy')
        train_labels = np.load('lego-simple-train-labels.npy')
        test_images = np.load('lego-simple-test-images.npy')
        test_labels = np.load('lego-simple-test-labels.npy')
```

2. If we look back in our jupyter notebook server, we can see they are stored already.



Integer values 1 through 10 are represented by the labels we loaded. Integer values are useful for the model to understand and forecast, but they are difficult to comprehend while working with data.

3. Next, we will give the labels we are trying to anticipate human-readable names.

```
In [3]: # For humans:
        class_names = ['2x3 Brick', '2x2 Brick', '1x3 Brick', '2x1 Brick',
                       '2x2 Macaroni', '2x2 Curved End', 'Cog 16 Tooth', '

        # Or the real LEGO codes:
        # class_names = ['3002', '3003', '3622', '3004', '3005', '3063', '
```

4. Take a look at one of the photos loaded with data. This tells us that our image is 48 by 48 pixels. We have normalized the values in this array from 0-1 by dividing by 255; since it is a greyscale image normally, it would be between zero and 255 as brightness for an eight-bit image, but with a floating-point value, it is easier for the machine learning model to understand.

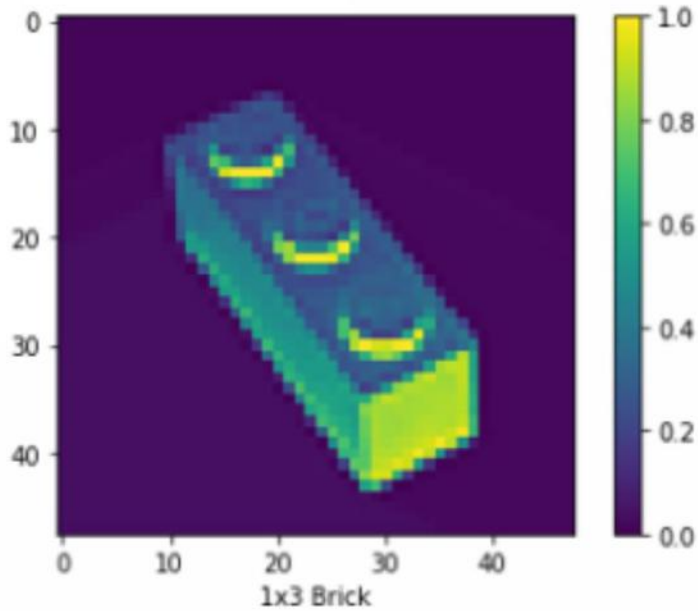
```
In [4]: print(train_images[0].shape)
        train_images[0]

        (48, 48)

Out[4]: array([[0.01568627, 0.01568627, 0.01568627, ..., 0.01568627, 0.01
        568627,
                0.01568627],
               [0.01568627, 0.01568627, 0.01568627, ..., 0.01568627, 0.01
        568627,
                0.01568627],
               [0.01568627, 0.01568627, 0.01568627, ..., 0.01568627, 0.01
        568627,
                0.01568627],
               ...,
               [0.03921569, 0.03921569, 0.03921569, ..., 0.01960784, 0.01
        960784,
                0.01960784],
               [0.03921569, 0.03921569, 0.03921569, ..., 0.01960784, 0.01
        960784,
                0.01960784],
               [0.03529412, 0.03529412, 0.03529412, ..., 0.01960784, 0.01
        960784,
                0.01960784]])
```

5. We will also look at the testing data to see whether it is similar.

```
In [5]: print(test_images[0].shape)
        test_images[0]

        (48, 48)

Out[5]: array([[0.00784314, 0.00784314, 0.00784314, ..., 0.00784314, 0.00
        784314,
                0.00784314],
               [0.00784314, 0.00784314, 0.00784314, ..., 0.00784314, 0.00
        784314,
                0.00784314],
               [0.00784314, 0.00784314, 0.00784314, ..., 0.00784314, 0.00
        784314,
                0.00784314],
               ...,
               [0.01568627, 0.01568627, 0.01568627, ..., 0.01568627, 0.01
        568627,
                0.01568627],
               [0.01960784, 0.01960784, 0.01960784, ..., 0.01568627, 0.01
        568627,
                0.01568627],
               [0.01960784, 0.01960784, 0.01960784, ..., 0.01568627, 0.01
        568627,
                0.01568627]])
```

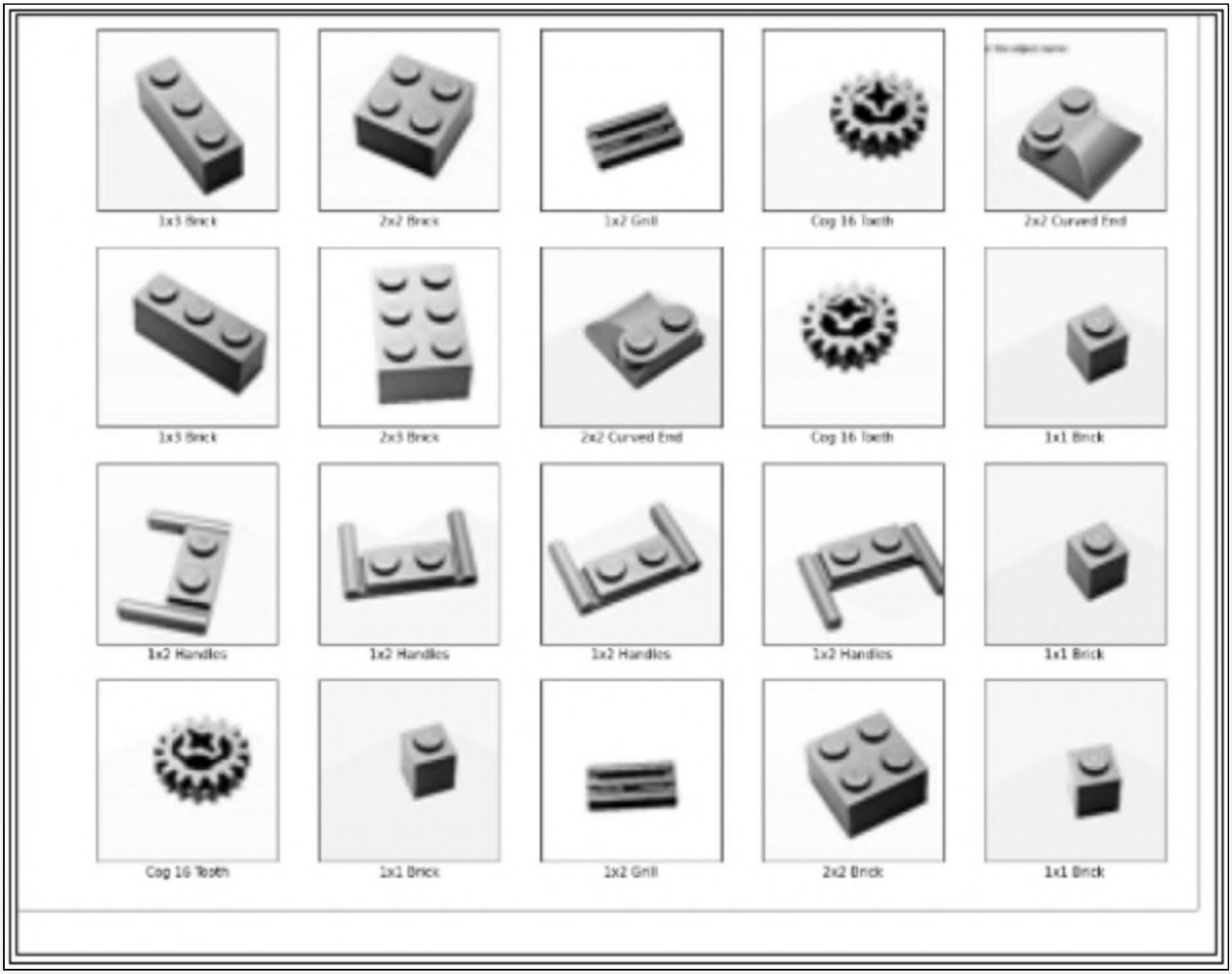
6. We will then plot these values to obtain a clearer view.

```
In [6]: plt.figure()
        plt.imshow(train_images[0])
        plt.colorbar()
        plt.xlabel(class_names[train_labels[0]])
        plt.show()
```



7. The top and bottom few rows are dark, as can be seen. Those floating-point values are close to zero, as we observed previously. We will take a look at some additional data in a more appealing manner. This will change the colors of the output. As a result, the background appears light, but the bricks are dark, making them easier to notice.

```
In [ ]: plt.figure(figsize=(15,15))
        for i in range(20):
            plt.subplot(5,5,i+1)
            plt.xticks([])
            plt.yticks([])
            plt.imshow(train_images[i], cmap=plt.cm.binary)
            plt.xlabel(class_names[train_labels[i]])
        plt.show()
```



Step 3: Train the TensorFlow Model

1. Now that the data has been imported, we must construct a model. We are constructing an artificial neural network. For this, we will utilize Keras and create a sequential model, which means each layer will pass to the next.

We are going to utilize the first layer as a flattened layer. Our input shape is truly 48 by 48, as seen above. Still, the model works better if it is a one-dimensional array rather than a two-dimensional array, which will be 2,304 values, which is actually 48 by 48. We will link the flattened layer's output to our next layer, which will be our concealed layer. Because this is a dense layer, every one of the input values will be linked to each of the 128 neurons triggered by the ReLU function. The network's final layer will be a dense layer of ten nodes, one for each of the classes we are trying to predict, which ten distinct sorts of Lego bricks are. We utilize the softmax activation function for this layer, which generates a set of probabilities for each of the ten nodes. The neuron will determine the class that the model predicts with the highest probability.

```
In [8]: model = keras.Sequential([
            keras.layers.Flatten(input_shape=(48, 48)),
            keras.layers.Dense(128, activation='relu'),
            keras.layers.Dense(10, activation='softmax')
        ])
```

2. Now, we will compile the model.

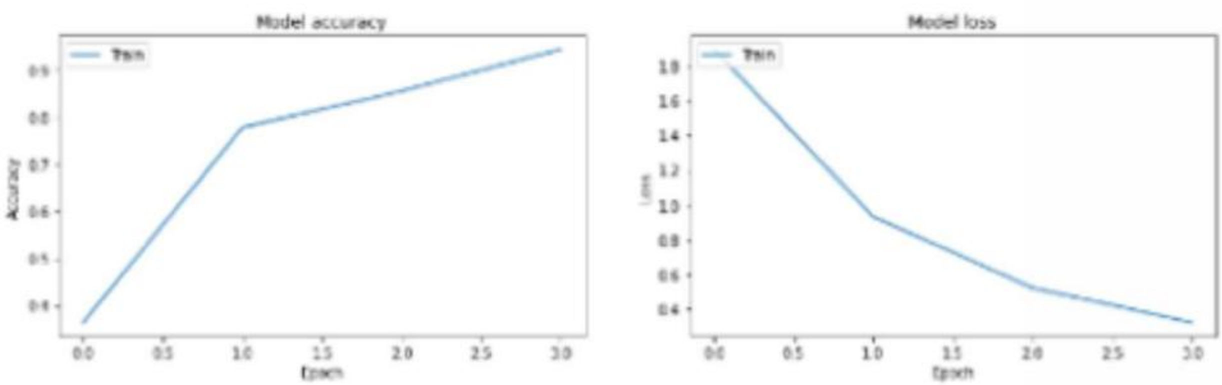
```
In [9]: model.compile(optimizer='adam',
                      loss='sparse_categorical_crossentropy',
                      metrics=['accuracy'])
```

3. Finally, it is time to put the model through its tests. Keep track of the epochs, and the accuracy as the process unfolds.

```
In [11]: history = model.fit(train_images, train_labels, epochs=4)

[2021-10-16 11:09:13.815 ip-172-16-35-238:27397 INFO utils.py:27]
RULE_JOB_STOP_SIGNAL_FILENAME: None
[2021-10-16 11:09:13.922 ip-172-16-35-238:27397 INFO profiler_con
fig_parser.py:111] Unable to find config at /opt/ml/input/config/
profilerconfig.json. Profiler is disabled.
Train on 451 samples
Epoch 1/4
451/451 [==============================] - 2s 5ms/sample - loss:
1.8926 - accuracy: 0.3636
Epoch 2/4
451/451 [==============================] - 0s 97us/sample - loss:
0.9308 - accuracy: 0.7783
Epoch 3/4
451/451 [==============================] - 0s 100us/sample - los
s: 0.5223 - accuracy: 0.8559
Epoch 4/4
451/451 [==============================] - 0s 93us/sample - loss:
0.3239 - accuracy: 0.9446
```

4. As you can see from the output above, the model's accuracy score improves with each epoch. When we trained the model, we saved the training history in a variable from plotting the training progress. Over the epochs, we may plot accuracy and loss.

```
In [12]: plt.figure(figsize=(15, 4))

         plt.subplot(1, 2, 1)
         # Plot training accuracy values
         plt.plot(history.history['accuracy'])
         plt.title('Model accuracy')
         plt.ylabel('Accuracy')
         plt.xlabel('Epoch')
         plt.legend(['Train', 'Test'], loc='upper left')

         plt.subplot(1, 2, 2)
         # Plot training loss values
         plt.plot(history.history['loss'])
         plt.title('Model loss')
         plt.ylabel('Loss')
         plt.xlabel('Epoch')
         plt.legend(['Train', 'Test'], loc='upper left')
         plt.show()
```



Step 4: Test and Analyze the Model

1. Now, we will compare the testing results to the accuracy. The model is probably overfitted if the accuracy attained during training is significantly higher than what we compute for the testing data. This implies it is good for classifying training data but not for classifying testing data.

```
In [13]: test_loss, test_acc = model.evaluate(test_images, test_labels)
         print('Test accuracy:', test_acc)

         150/150 [==============================] - 0s 498us/sample - los
         s: 0.4387 - accuracy: 0.8667
         Test accuracy: 0.8666667
```

2. We will now apply our model to our testing data to generate some predictions.

3. We will use one of the images from the test dataset.

```
In [ ]: test_image_number = 8

        img = test_images[test_image_number]
        plt.figure()
        plt.xticks([])
        plt.yticks([])
        plt.imshow(img, cmap=plt.cm.binary)
        plt.show()
```

4. Before we can examine what the model thinks, we will quickly modify this image. We build a collection of one image since the forecast function wants a collection of photos.

```
In [15]: img = (np.expand_dims(img,0))
         img.shape

Out[15]: (1, 48, 48)
```

5. The shape is now a collection of one 48x48 data item. The picture is then passed to the predict method. The end result is a list of probability that the image belongs to a certain class.

```
In [16]: predictions_single = model.predict(img)
         predictions_single

Out[16]: array([[0.00144515, 0.01388346, 0.00972838, 0.5691016 , 0.0557792
         3,
                 0.23145372, 0.09599604, 0.01162628, 0.00249234, 0.0084938
         5]],
               dtype=float32)
```

6. The argmax function in NumPy can be used to discover the index of the biggest value in a list. We may utilize a second NumPy function to get the real maximum value.

```
In [17]: prediction_result = np.argmax(predictions_single[0])
         probability = np.max(predictions_single[0])
         print(f"Predicted class {prediction_result}, which is {class_names
         print(f"The model thinks there is a {probability*100:.2f}% probabi

         Predicted class 3, which is 2x1 Brick.
         The model thinks there is a 56.91% probability of this brick bein
         g correct.
```

7. Next, we will search for the test image's class name (i.e., the real class name, not the predicted one). We are hoping for the same!

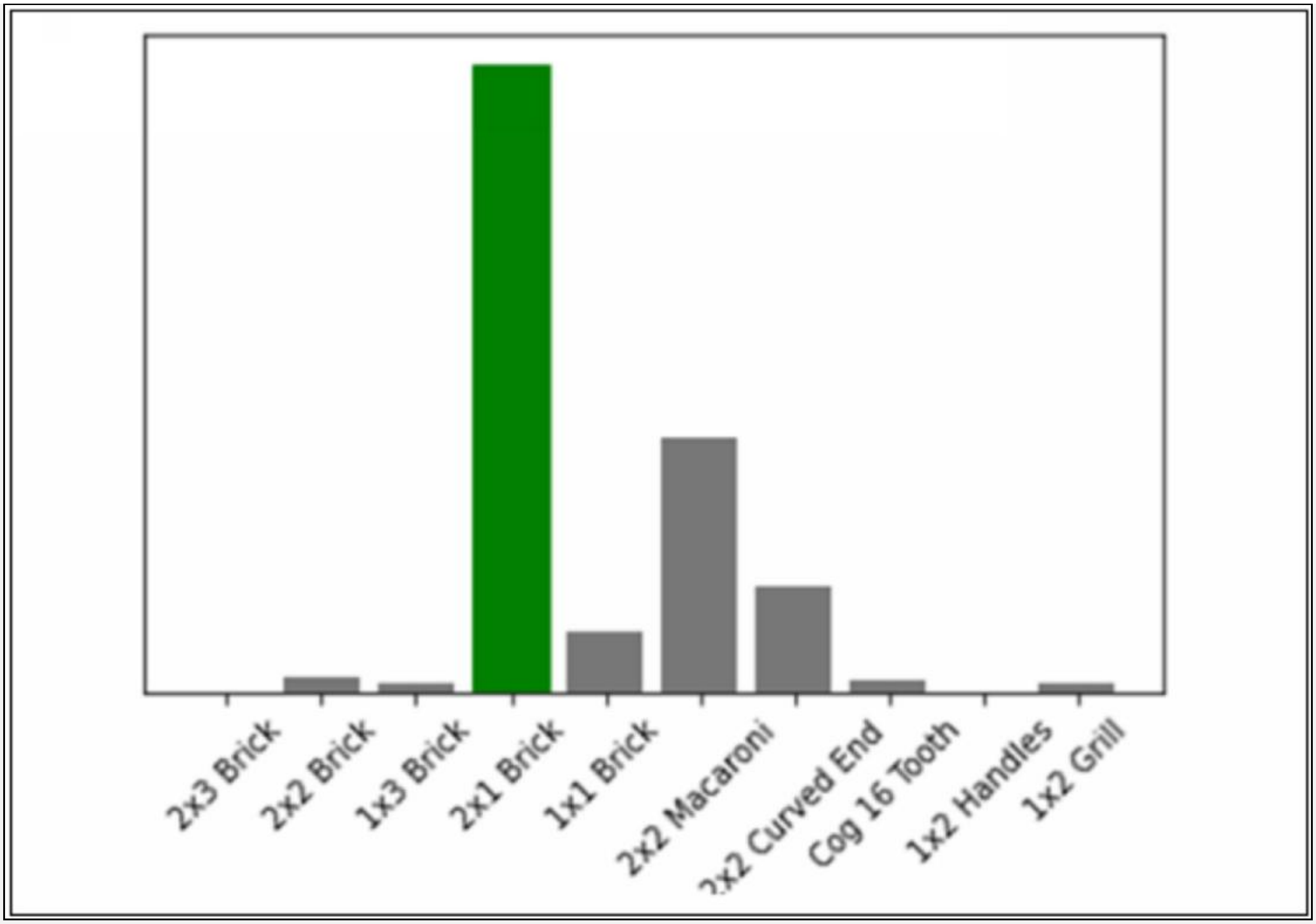
```
In [18]: class_names[test_labels[test_image_number]]

Out[18]: '2x1 Brick'
```

8. These functions aid in the presentation of prediction data.

```
In [19]: # Function to display the image:
def plot_image(predictions_array, true_label, img):
    plt.xticks([])
    plt.yticks([])
    plt.imshow(img, cmap=plt.cm.binary)
    predicted_label = np.argmax(predictions_array)
    predicted_probability = 100 * np.max(predictions_array)
    color = 'green' if predicted_label == true_label else 'red'
    # Print a label with 'predicted class', 'probability %', 'actu
    plt.xlabel(f"{class_names[predicted_label]} {predicted_probabi
                                  color=color)

# Function to display the prediction results in a graph:
def plot_value_array(predictions_array, true_label):
    plt.xticks(range(10))
    plt.yticks([])
    plot = plt.bar(range(10), predictions_array, color="#777777")
    predicted_label = np.argmax(predictions_array)
    plot[predicted_label].set_color('red')
    plot[true_label].set_color('green')
```

9. We will then use the helper function to create a bar chart. This tells us how accurate the model was in classifying this image. If the projected label differs from the actual label, it will be highlighted in red. Green will be the real label.

```
In [ ]: plot_value_array(predictions_single[0], test_labels[test_image_num
        plt.xticks(range(10), class_names, rotation=45)
        plt.show()
```



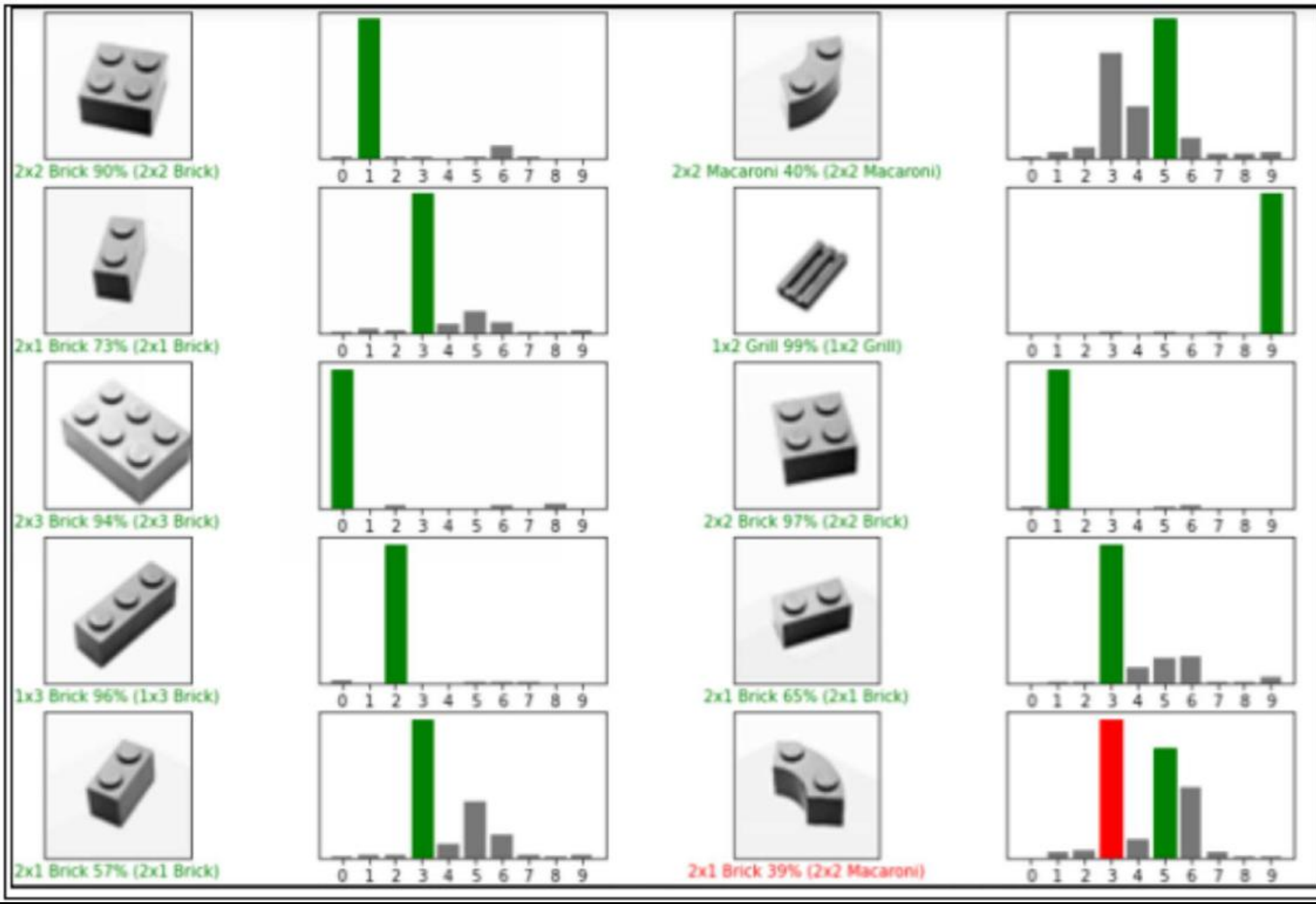
Step 5: Make a Batch Prediction Using the Testing Data

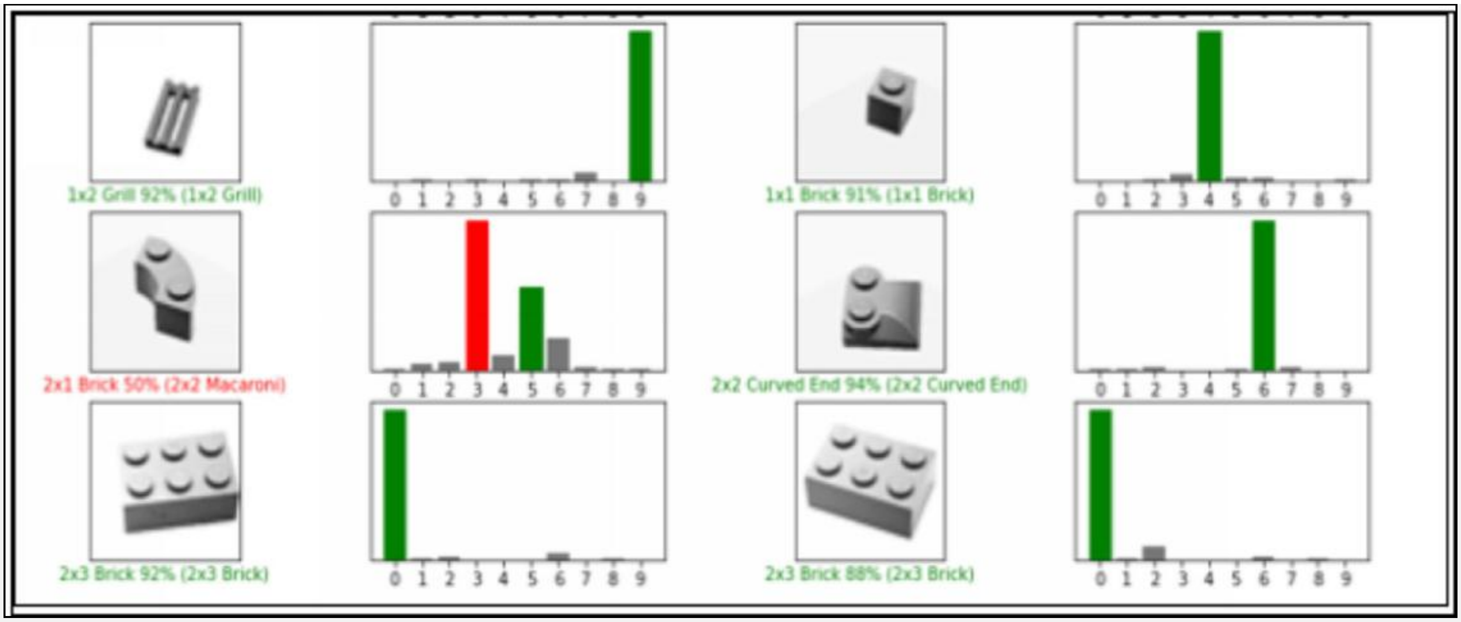
1. We will now obtain prediction values for our test photos.

```
In [21]: predictions = model.predict(test_images)

In [22]: predictions[0]

Out[22]: array([5.2462332e-03, 9.0270740e-01, 1.1476018e-03, 2.3488151e-0
         3,
                7.4006454e-07, 4.8946785e-03, 8.1795849e-02, 1.6155281e-0
         3,
                4.2367505e-05, 2.0071988e-04], dtype=float32)
```

2. Our helper functions will now be used to summarize the first 16 photos in our test data.

```
In [ ]: num_rows = 8
        num_cols = 2
        num_images = num_rows*num_cols
        plt.figure(figsize=(15, 16))
        for i in range(num_images):
            plt.subplot(num_rows, 2*num_cols, 2*i+1)
            plot_image(predictions[i], test_labels[i], test_images[i])
            plt.subplot(num_rows, 2*num_cols, 2*i+2)
            plot_value_array(predictions[i], test_labels[i])
        plt.show()
```





Given that our model is about 90% accurate, one or two of them may have been classified incorrectly. That is still not bad for only training for four epochs on such little data!

Lab 12: Creating an MXNet Image Classifier in AWS SageMaker

Certifications	AWS Machine Learning - Specialty
Cloud Provider	AWS
Time Required	1 Hr.
Tags	SageMaker, AWS

Lab Prerequisites

- Familiarity with basic AWS Cloud Computing concepts and terminology
- An AWS account with an active subscription

Service Introduction

Apache MXNet is an open-source machine learning framework that focuses on deep neural network learning. Apache MXNet is a fast and scalable machine learning training and inference framework with a simple and clear API.

The Gluon interface included in MXNet allows developers of various skill levels to get started with deep learning on the cloud, in edge devices, and mobile apps. You can construct linear regression, convolutional networks, and recurrent LSTMs for object identification, speech recognition, recommendation, and customization with just a few lines of Gluon code.

Case Study Enterprise Financial Service– Kasasa

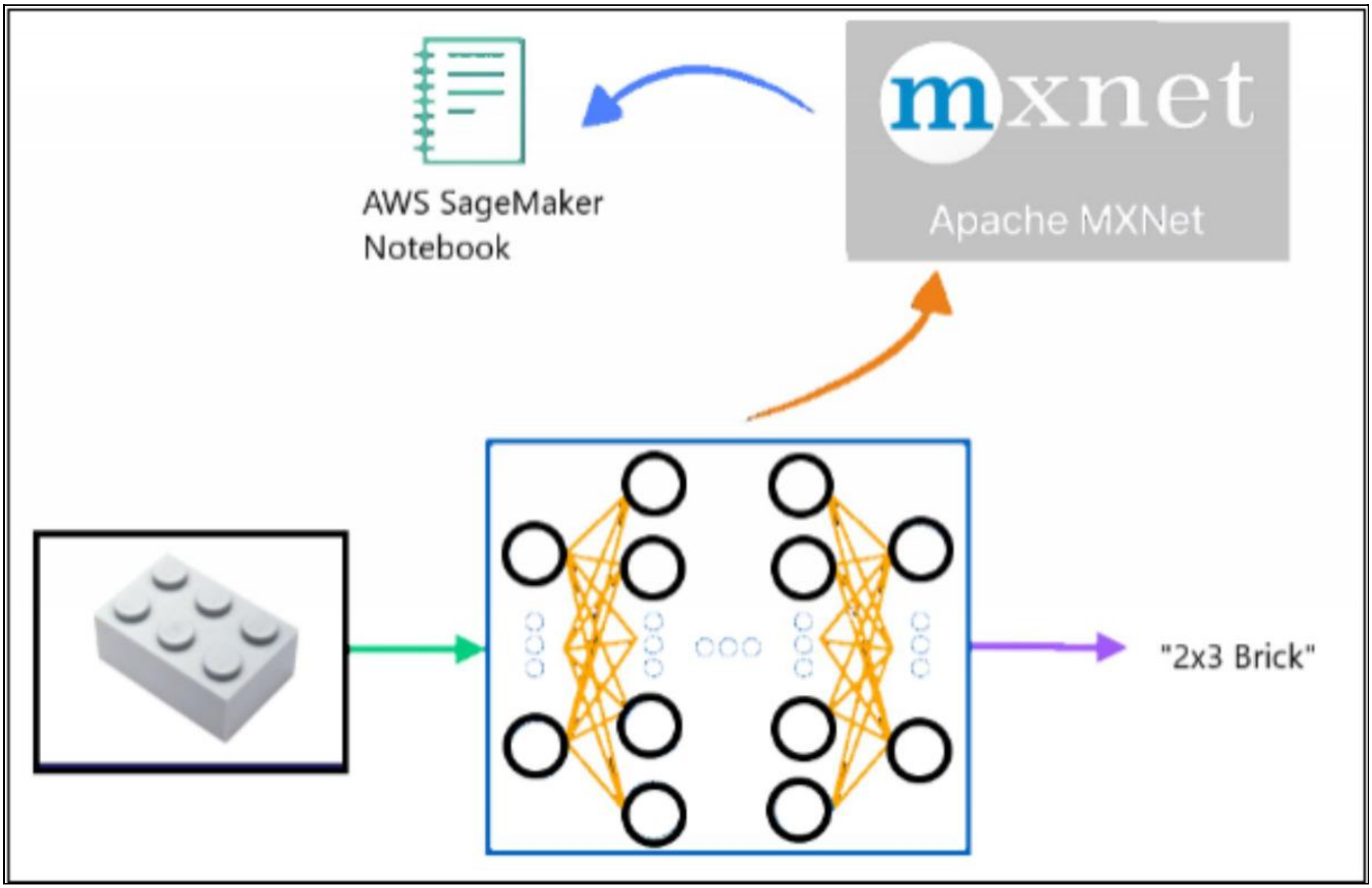
Business Challenge

You work as an AWS architect in Kasasa. They gave you the task of building a machine that can reliably sort lego bricks. You have a lot of LEGO bricks to sort. Doing it by hand will take forever and cause you so much eye strain from looking at the tiny pieces. Hence, you need to train a machine to recognize the bricks and tell you (or ideally, a robot) which bin the piece should be placed in. You already have a lot of pictures of LEGO bricks, which you will use to train the model.

Proposed Solution

The solution is you use AWS services to automate your task, and you will need to teach a machine to recognize the bricks and inform you (or, better yet, a robot) which bin each piece belongs in. You already have a lot of LEGO block images, which you will utilize to teach the model you are using.

Lab Diagram



Implementation Steps

1. Navigate to Jupyter Notebook.
2. Load and Prepare the Data.
3. Train the MXNet Model.
4. Evaluate the Model.
5. Make a Batch Prediction.

Solution

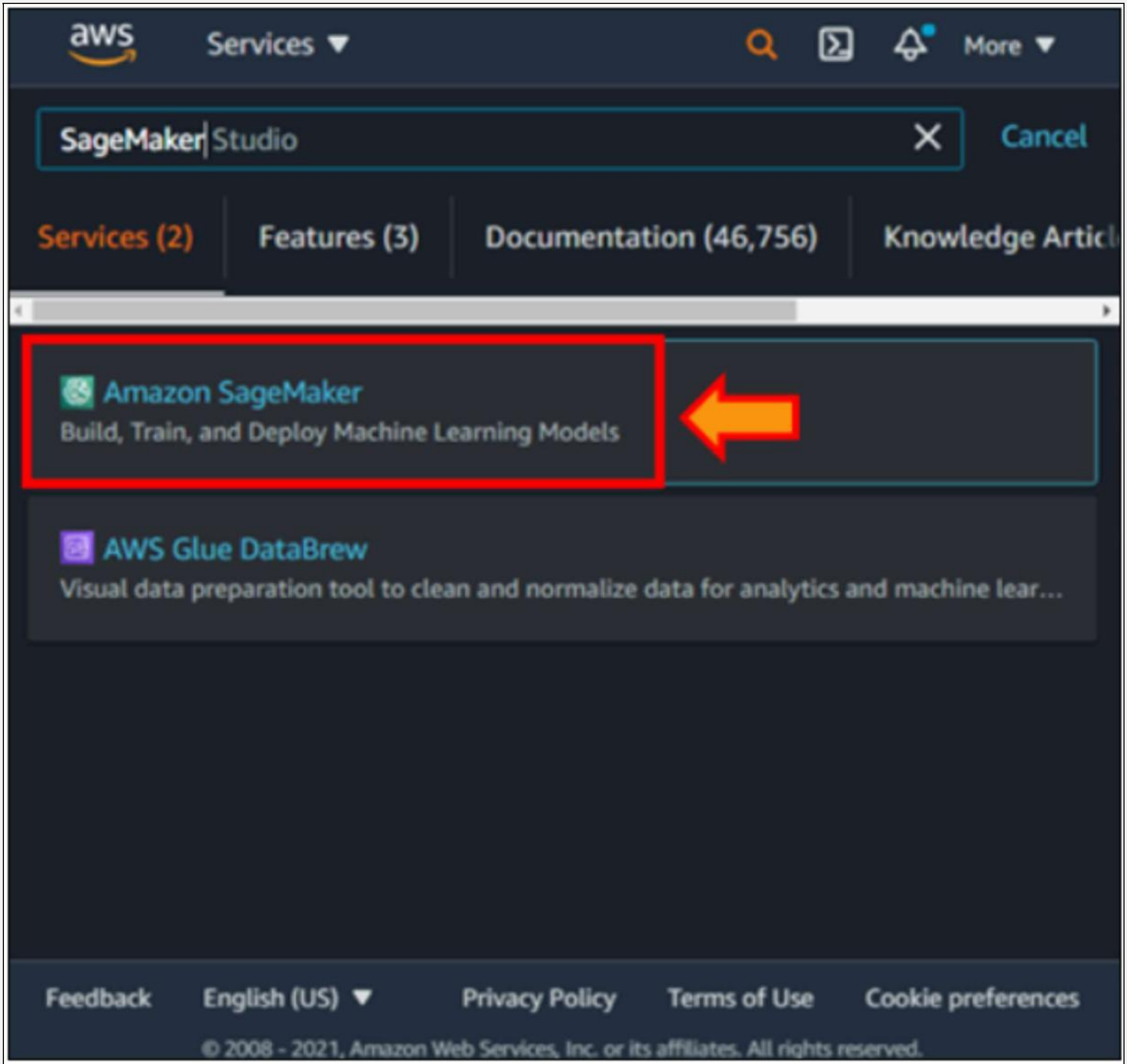
Step 1: Navigate to Jupyter Notebook

Note: Before starting the lab, you must have a Jupyter notebook. You can create it by following the steps given in the link:

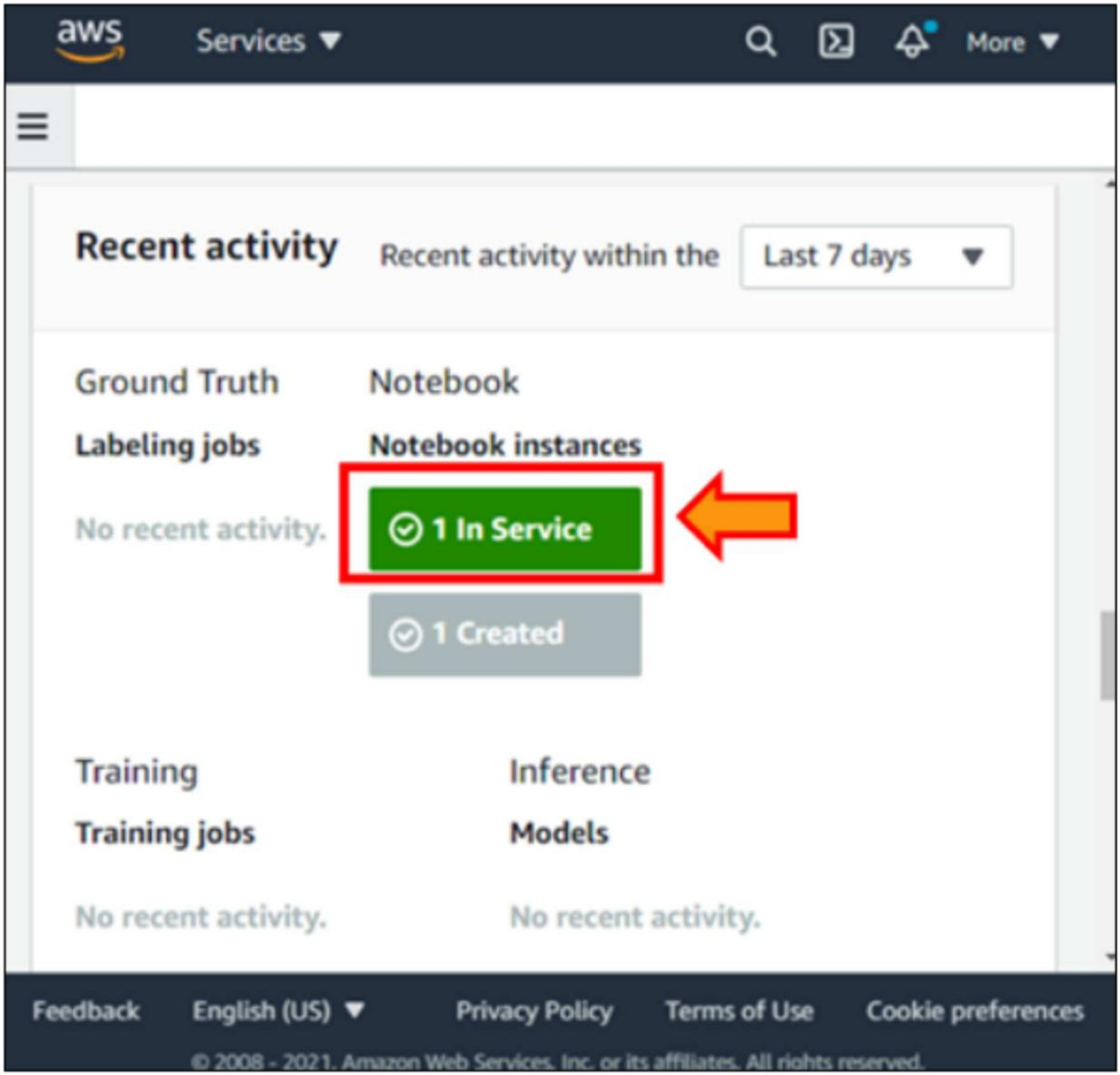
https://docs.aws.amazon.com/dlami/latest/devguide/setup-jupyter.html

1. Log in to the **AWS Console**.

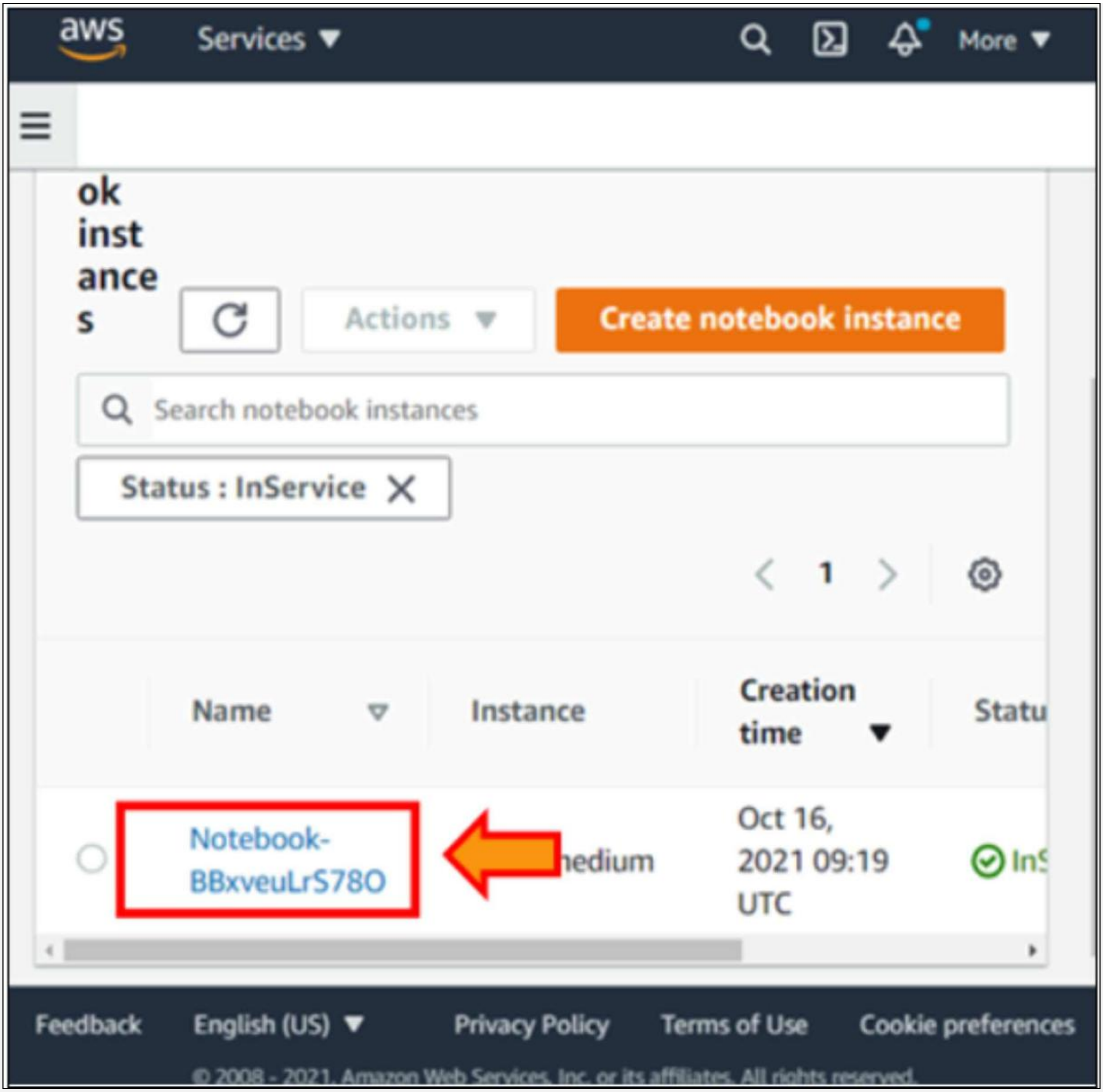
2. Go to **Services** and click on **SageMaker.**



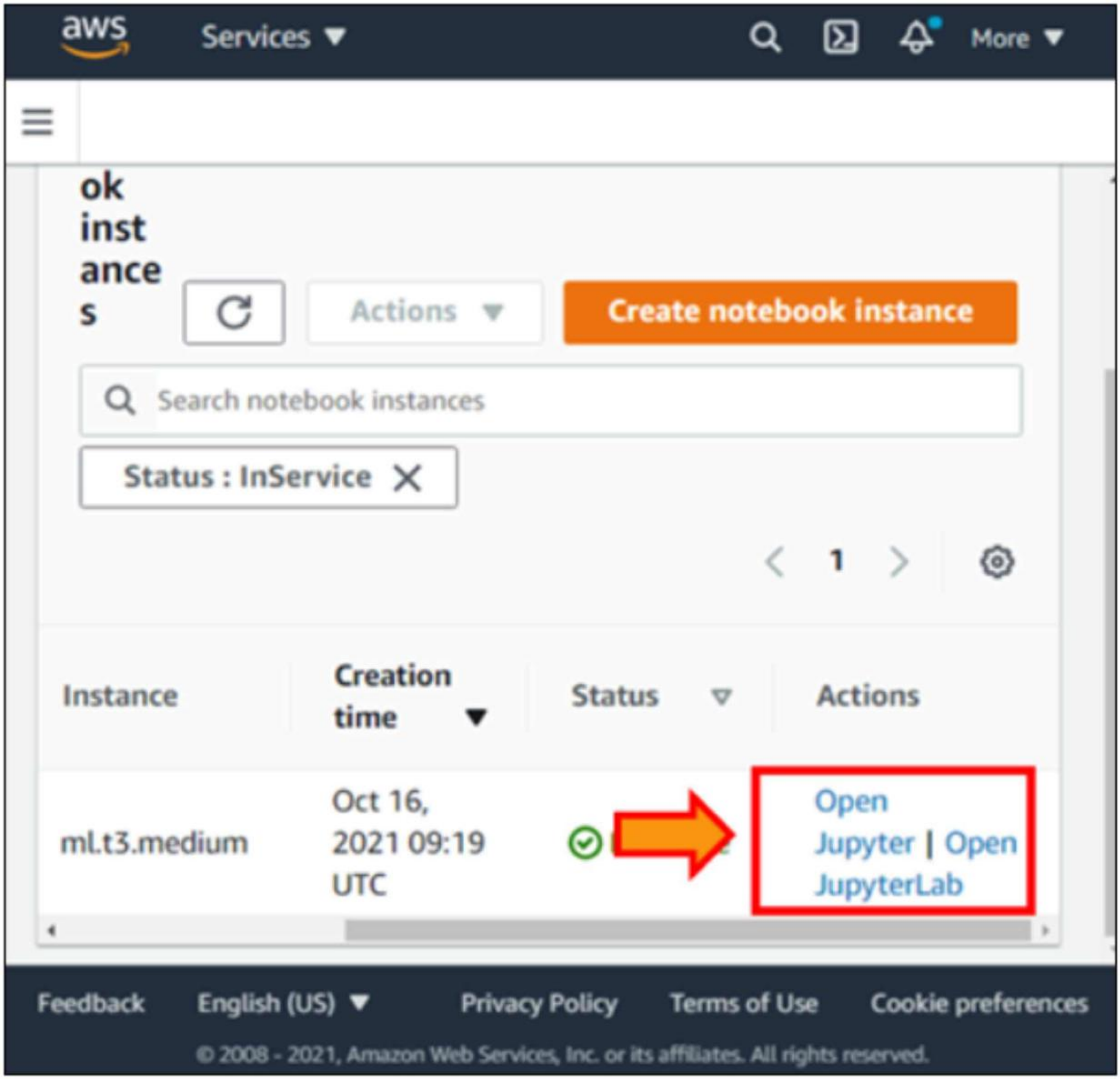
3. Click on **In-Service.**



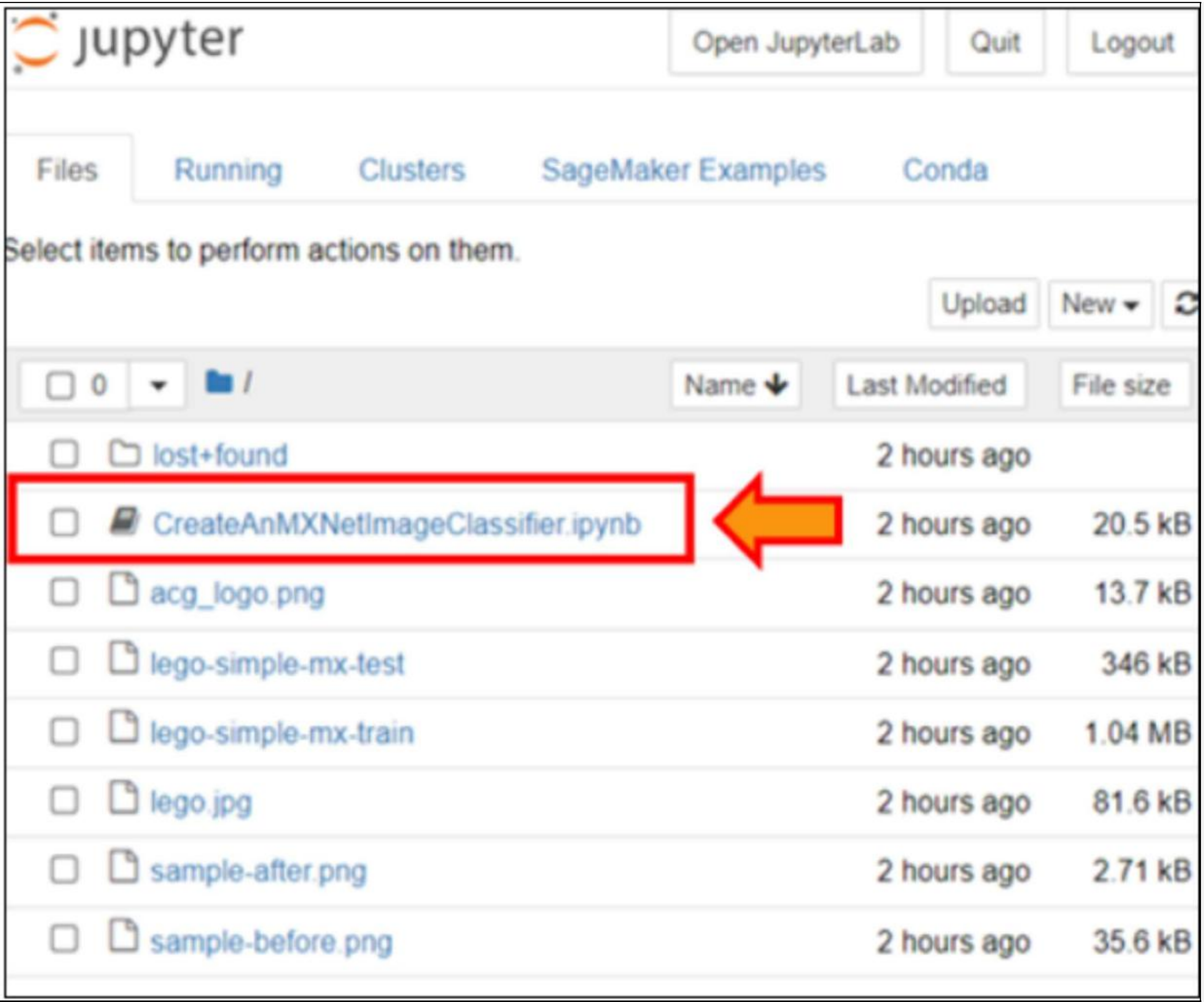
4. Click on the **Notebook instance.**



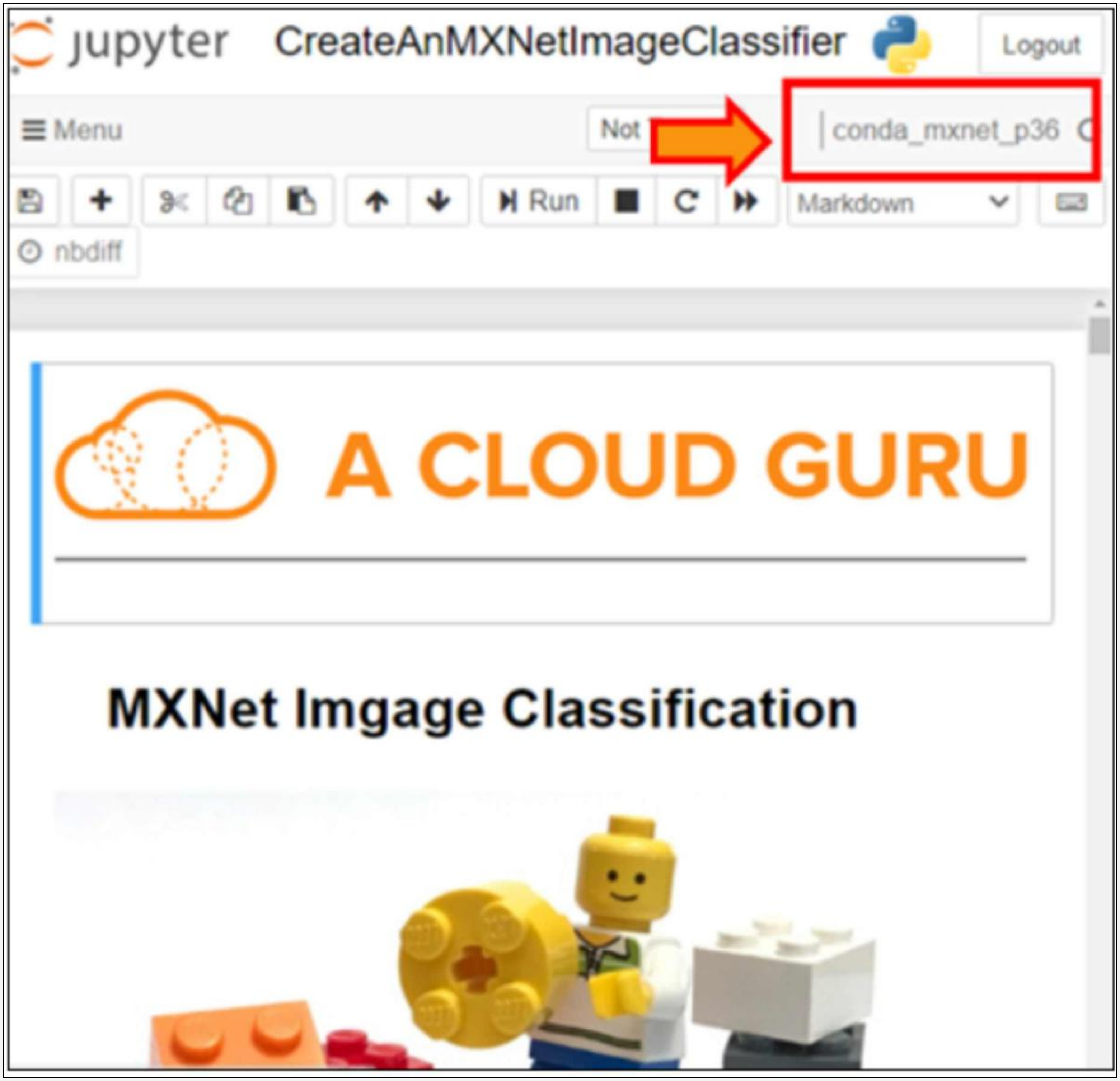
5. Click on **Open Jupyter.**



6. Click on **CreateAnMXNetImageClassifier.ipynb.**

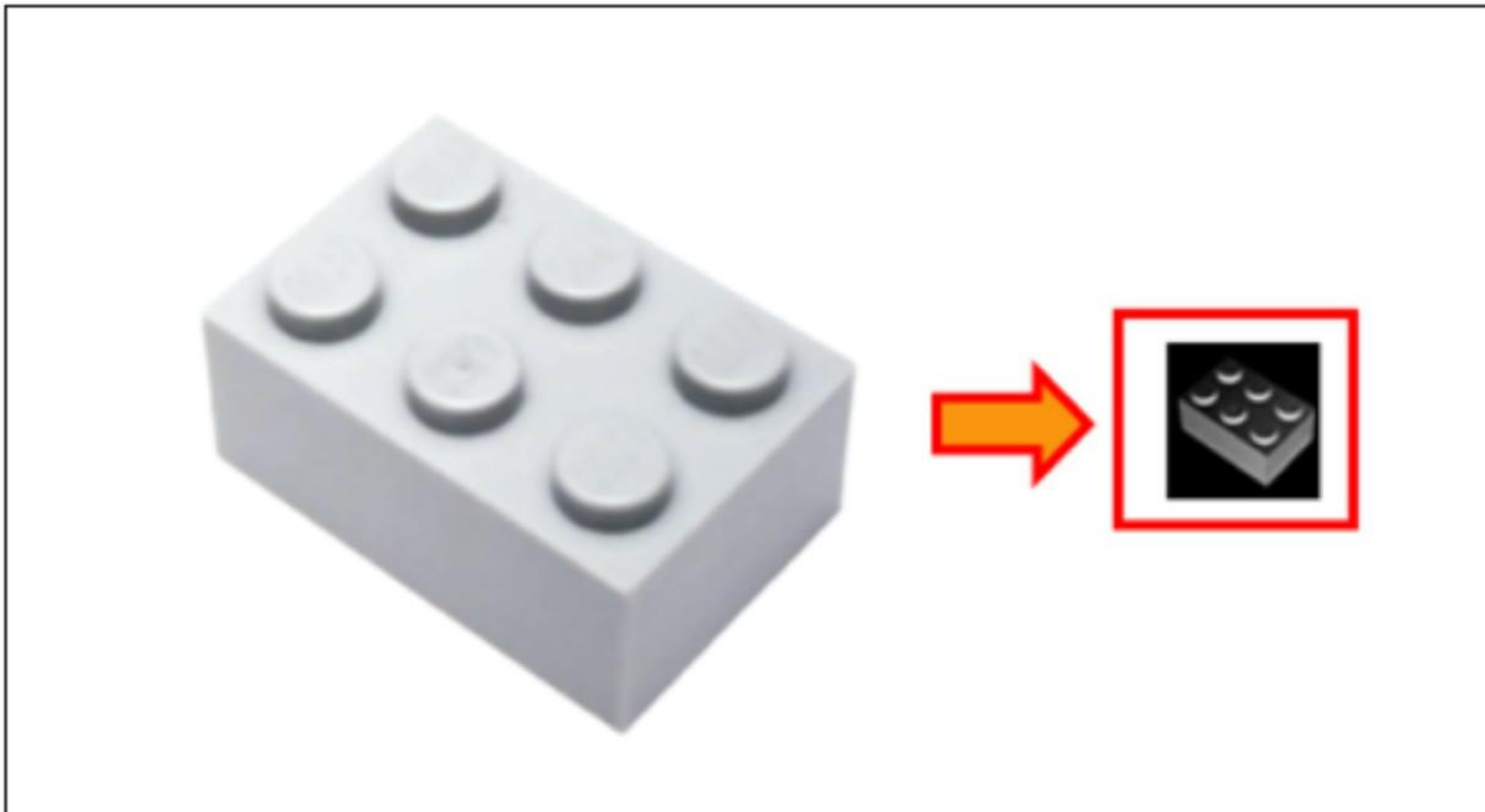


7. Make sure the kernel supports **MXNet.**



8. We have a collection of photographs of different LEGO bricks taken from various perspectives. There are 600 pictures in all, all tagged with the brick type.

Each image has been enhanced. Increasing the contrast, sharpening, eliminating the color, inverting the color, and shrinking the image were all part of the process. We put them all into data arrays to make importing the photos into the notebook easier. The folders lego-simple-train-images.npy and lego-simple-test-images.npy include these pictures.



Sample before processing

Sample after processing

10. Our first step is to import all the necessary libraries. We will use a fixed seed that is generated at random to ensure that our models are reproducible.

```
In [1]: import mxnet as mx
        from mxnet.gluon.data.vision import transforms

        from mxnet import nd, gluon, autograd
        from mxnet.gluon import nn

        import pickle

        import numpy as np
        import matplotlib.pyplot as plt

        # Select a fixed random seed for reproducibility
        mx.random.seed(42)
```

11. MXNet's processor type will be determined. We let MXNet figure out that this Juypter Notebook server does not have access to GPUs.

```
In [2]: ctx = mx.gpu(0) if mx.context.num_gpus() > 0 else mx.cpu(0
```

Step 2: Load and Prepare the Data

1. We have a dataset built from a collection of LEGO block pictures. We have two sets of data saved to files as NDArrays in total.

- lego-simple-mx-train - Combines training pictures and labels, accounting for around 80% of the data gathered
- lego-simple-mx-test - Images and labels for testing or validation, accounting for around 20% of the data gathered

The data is first loaded into runtime arrays. Because these object files were created with Pickle, we will also use Pickle to load them.

```
In [3]: # Open the files:
        train_fh = open('lego-simple-mx-train', 'rb')
        test_fh = open('lego-simple-mx-test', 'rb')

        # Use pickle to load files into runtime objects:
        train_data = pickle.load(train_fh)
        test_data = pickle.load(test_fh)
```

2. We loaded integer values for the label data (1,2,3). We want to give the data classes we are working with human names.

```
In [5]: # For humans:
        class_names = ['2x3 Brick', '2x2 Brick', '1x3 Brick', '2x1
                       '2x2 Macaroni', '2x2 Curved End', 'Cog 16 T

        # Or the real LEGO codes:
        # class_names = ['3002', '3003', '3622', '3004', '3005', '
```

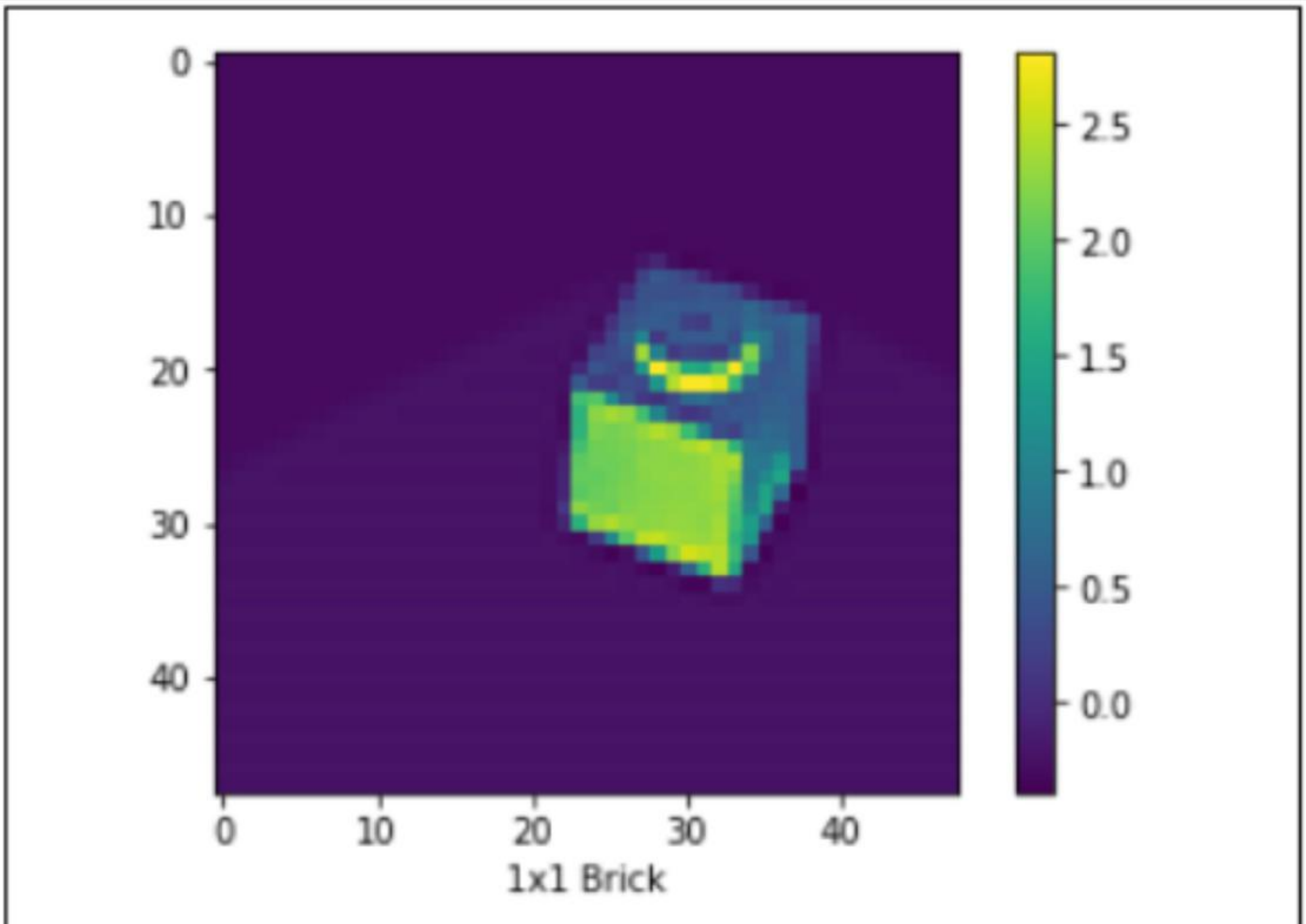
3. The data has been put into NDArrays. We now convert the data to MXNet tensors.

Tensors are similar to arrays, but they have more capabilities.

```
In [6]: transformer = transforms.Compose([
            transforms.ToTensor(),
            transforms.Normalize(0.13, 0.31)])

        train_data = train_data.transform_first(transformer)
        test_data = test_data.transform_first(transformer)
```

4. We will look at one of the pictures with the data loaded on it.

```
In [ ]: train_image_no = 0

        images_data, label_data = train_data[train_image_no]
        plt.figure()
        plt.imshow(images_data.reshape((48,48)).asnumpy())
        plt.colorbar()
        plt.xlabel(class_names[label_data])
        plt.show()
```



5. We will look closely at some more data and improve the formatting.

```
In [ ]: plt.figure(figsize=(15,15))
        for i in range(20):
            images_data, label_data = train_data[i]
            plt.subplot(5,5,i+1)
            plt.xticks([])
            plt.yticks([])
            plt.imshow(images_data.reshape((48,48)).asnumpy(), cma
            plt.xlabel(class_names[label_data])
        plt.show()
```

Step 3: Train the MXNet Model

1. Now that the data has been imported, we can begin training. We must first construct a model. The model object net is what we call MXNet. We are constructing an artificial neural network. It is made up of four layers:

- The input layer for our picture data, with enough nodes
- A 128-node hidden layer with ReLU activation

- A 64-node hidden layer with ReLU activation
- There are ten nodes in the output layer, one for each of the classes we wish to identify

Each layer is densely connected, which means that every neuron in one layer is linked to every neuron in the next.

```
In [9]: net = nn.HybridSequential(prefix='MLP_')
        with net.name_scope():
            net.add(
                nn.Flatten(),
                nn.Dense(128, activation='relu'),
                nn.Dense(64, activation='relu'),
                nn.Dense(10, activation=None)
            )
```

2. We may utilize data loaders provided by MXNet Gluon to make data loading easier when training our model. We will get them set up.

```
In [10]: batch_size = 34
         train_loader = mx.gluon.data.DataLoader(train_data, shuffl
```

3. We will now initialize the model.

```
In [11]: net.initialize(mx.init.Xavier(), ctx=ctx)
```

4. A trainer object is provided by Gluon to keep track of the training state. It is something we make here and use in the training process.

```
In [12]: trainer = gluon.Trainer(
             params=net.collect_params(),
             optimizer='sgd',
             optimizer_params={'learning_rate': 0.04},
         )
```

5. First, we choose a measure and a loss function to utilize when training. We just utilize the softmax loss function provided by Gluon.

```
In [13]: metric = mx.metric.Accuracy()
         loss_function = gluon.loss.SoftmaxCrossEntropyLoss()
```

6. We are now in the process of training. We could put the following code in a fit function, but this inline code is sufficient.

```
In [ ]: num_epochs = 10
        history = []

        for epoch in range(num_epochs):
            for inputs, labels in train_loader:
                # Possibly copy inputs and labels to the GPU
                inputs = inputs.as_in_context(ctx)
                labels = labels.as_in_context(ctx)

                # Forward pass
                with autograd.record():
                    outputs = net(inputs)
                    loss = loss_function(outputs, labels)

                # Backpropagation
                loss.backward()
                metric.update(labels, outputs)

                # Update
                trainer.step(batch_size=inputs.shape[0])

            # Print the evaluation metric and reset it for the next epoch
            name, acc = metric.get()
            history.insert(epoch,acc)
            print('.', end='')
            metric.reset()

        print('[Done]')
```

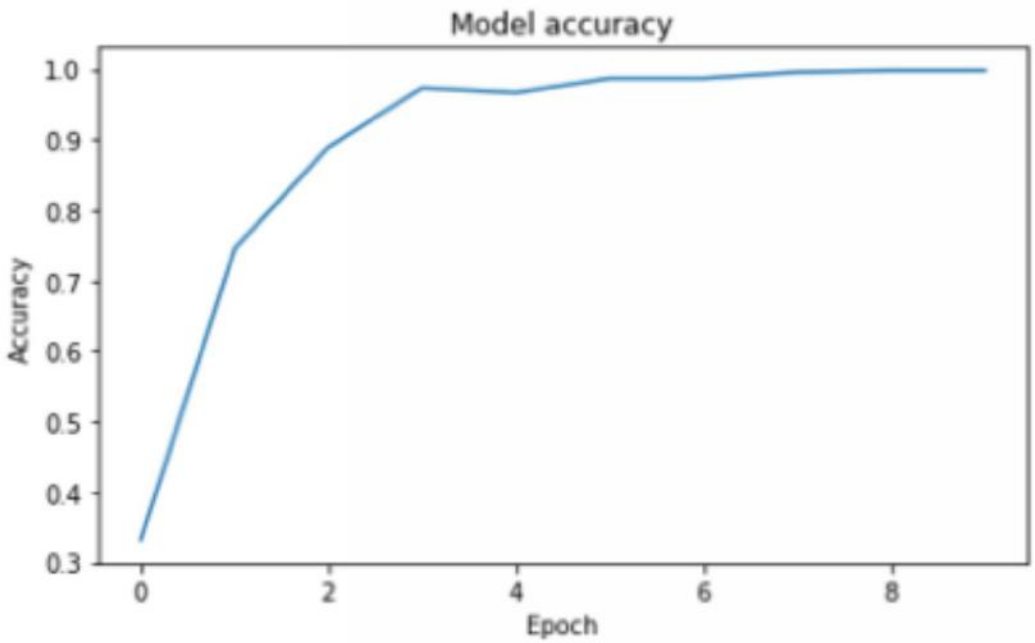

```
    # Print the evaluation metric and reset it for the next epoch
    name, acc = metric.get()
    history.insert(epoch,acc)
    print('.', end='')
    metric.reset()

print('[Done]')
```


```
..........[Done]
```

Step 4: Evaluate the Model

1. We recorded accuracy statistics for each epoch within the training loop. We will plot this data to obtain a feel of how the training went.

```
In [15]: plt.figure(figsize=(7, 4))
         plt.plot(history)
         plt.title('Model accuracy')
         plt.ylabel('Accuracy')
         plt.xlabel('Epoch')

Out[15]: Text(0.5, 0, 'Epoch')
```



2. We utilize the Gluon data loader once more with the test data.

```
In [16]: test_loader = mx.gluon.data.DataLoader(test_data, shuffle=False, b
```

3. We will evaluate the accuracy.

```
In [17]: metric = mx.metric.Accuracy()
         for inputs, labels in test_loader:
             # Possibly copy inputs and labels to the GPU
             inputs = inputs.as_in_context(ctx)
             labels = labels.as_in_context(ctx)
             metric.update(labels, net(inputs))
         metric_name, metric_value = metric.get()
         print(f'Validation: {metric_name} = {metric_value}')

         Validation: accuracy = 0.9733333333333334
```

4. We write a few functions to show the results to make our tests seem good.

```
In [18]: # Function to display the image:
         def plot_image(predictions_array, true_label, img):
             plt.xticks([])
             plt.yticks([])
             #Image data is currently a flat array. Convert to matrix for c
             plt.imshow(img.reshape((48,48)).asnumpy(), cmap=plt.cm.binary)
             predicted_label = np.argmax(predictions_array)
             probability = 100 * np.max(predictions_array)
             color = 'green' if predicted_label == true_label else 'red'
             # Print a label with 'predicted class' 'actual class'
             plt.xlabel(f"{class_names[predicted_label]} ({true_label} - {c
                                color=color)

         # Function to display the prediction results in a graph:
         def plot_value_array(predictions_array, true_label):
             plt.xticks(range(10))
             plt.yticks([])
             plot = plt.bar(range(10), predictions_array, color="#777777")
             predicted_label = np.argmax(predictions_array)
             plot[predicted_label].set_color('red')
             plot[true_label].set_color('green')
```

5. We will test our model using one of the photos from our test set.

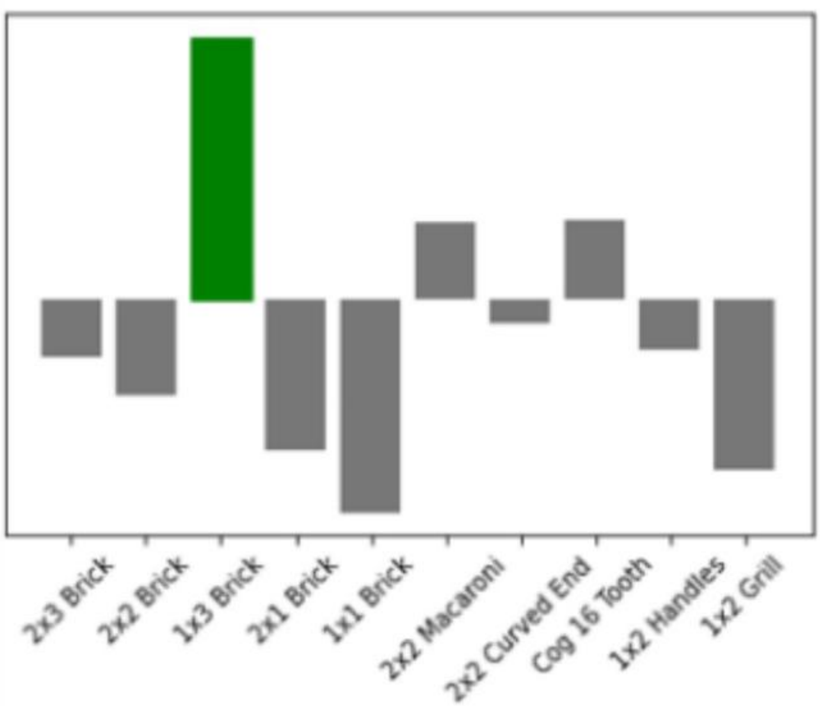
```
In [19]: prediction_image_number = 25
```

6. We are going to make a prediction now.

```
In [20]: prediction_image, prediction_label = test_data[prediction_image_nu
         predictions_single = net(prediction_image)
         predictions_single

Out[20]:
         [[-1.5633558 -2.5973158  6.8950944 -4.0451927 -5.649682   1.98286
         66
           -0.6797639  2.0568316 -1.3924415 -4.5649447]]
         <NDArray 1x10 @cpu(0)>
```

7. Using the helper function, we will create a bar chart. This tells us how accurate the model was in classifying this image. If the projected label differs from the actual label, it will be highlighted in red. Green will be the real label.

```
In [21]: plot_value_array(predictions_single[0].asnumpy(), prediction_label
         plt.xticks(range(10), class_names, rotation=45)
         plt.show()
```



8. To see which block we should have found, we run the following code.

```
In [22]: plot_image(predictions_single[0].asnumpy(), prediction_label, prec
```

1x3 Brick (2 - 1x3 Brick)

Step 5: Make a Batch Prediction

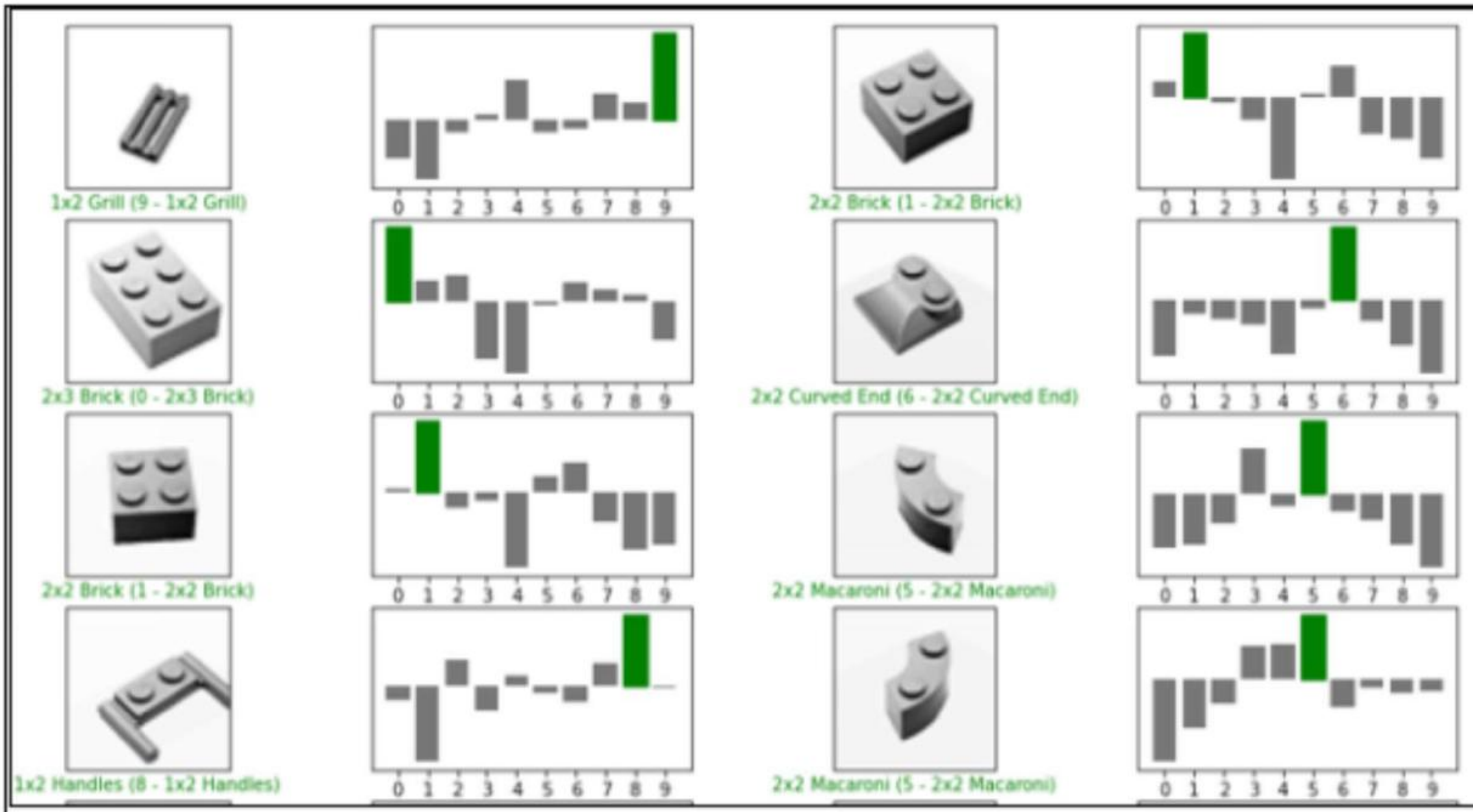
1. We will now obtain prediction values for our test photos. We will not utilize a data loader; instead, we will loop through the test picture data in its raw form.

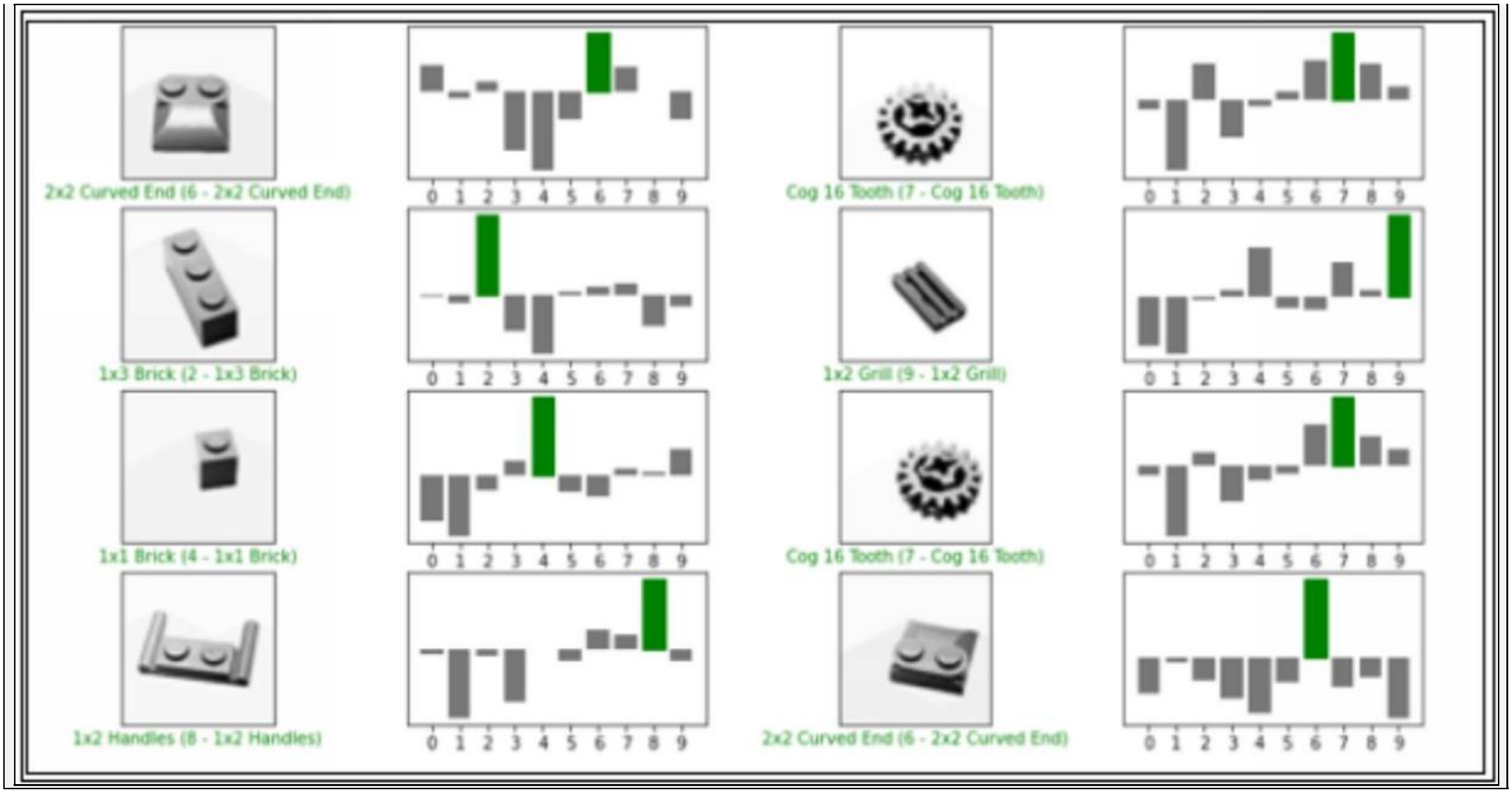
```
In [23]: predictions = []
         test_labels = []

         for i in test_data:
             pred_image, pred_label = i
             p = net(pred_image)
             predictions.append(p)
             test_labels.append(pred_label)
```

2. To summarize the first 16 photos in our test set, we will use our helper methods.

In [ ]:

```
num_rows = 8
num_cols = 2
num_images = num_rows*num_cols
plt.figure(figsize=(15, 16))
for i in range(num_images):
    plt.subplot(num_rows, 2*num_cols, 2*i+1)
    plot_image(predictions[i].asnumpy(), test_data[i][1], test_dat
    plt.subplot(num_rows, 2*num_cols, 2*i+2)
    plot_value_array(predictions[i][0].asnumpy(), test_data[i][1])
plt.show()
```





Lab 13: Creating a scikit-learn Random Forest Classifier in AWS SageMaker

Certifications	AWS Machine Learning - Specialty
Cloud Provider	AWS
Time Required	1 Hr.
Tags	SageMaker, AWS

Lab Prerequisites

- Familiarity with basic AWS Cloud Computing concepts and terminology
- An AWS account with an active subscription

Service Introduction

Scikit-learn (Sklearn) is the most usable and robust machine learning package in Python. It uses a Python consistent interface to give a set of fast tools for machine learning and statistical modelings, such as classification, regression, clustering, and dimensionality reduction.

A random forest is a meta estimator that utilizes averaging to enhance prediction accuracy and control over-fitting by fitting several decision tree classifiers on various sub-samples of the dataset. If bootstrap=True (default), the sub-sample size is regulated by the max samples argument; otherwise, the whole dataset is utilized to create each tree.

Case Study Enterprise Financial Service– Kasasa

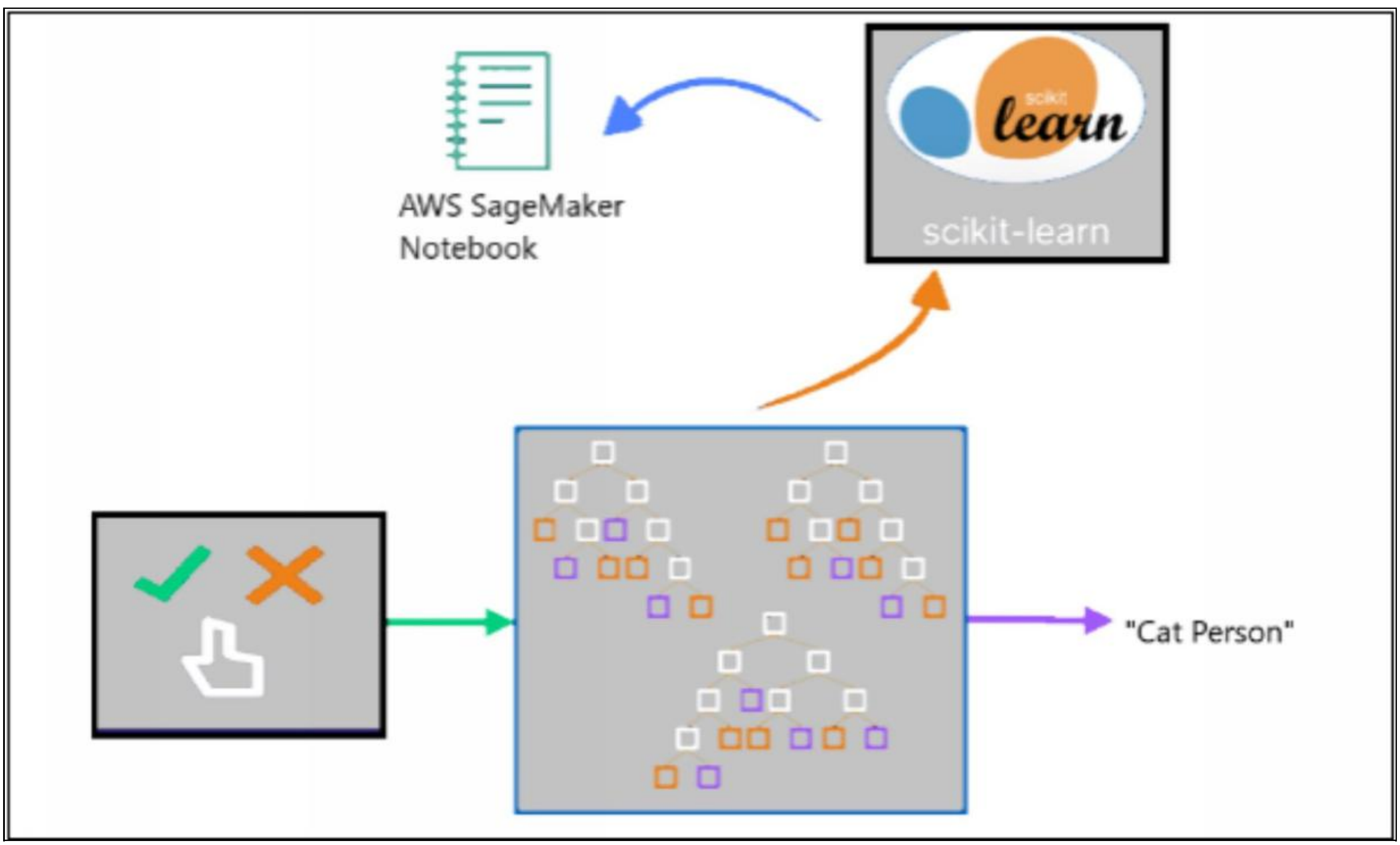
Business Challenge

Kasasa owns and operates a pet store in its neighborhood. Kasasa hired an agency to survey 199 of its current customers to assist you in figuring out what type of items to promote to new consumers. Kasasa clients were asked what their favorite color is, if they enjoy walking or running, and how many kilometers they walk each day. Last but not least, they were asked if they preferred cats or dogs. Subsequently, you will construct a Random Forest Classifier with scikit-learn to determine whether your customers like cats or dogs.

Proposed Solution

The solution is you use AWS services to automate your task and analyze the stored data in real-time within minutes. For storing the data in real-time, you use Amazon Simple Storage S3. You will also construct a Random Forest Classifier with scikit-learn to determine whether your customers like cats or dogs.

Lab Diagram



Implementation Steps

1. Navigate to Jupyter Notebook.
2. Load and Prepare the Data.
3. Create the scikit-learn Model.

4. Evaluate the Model.
5. Predict for Yourself.

Solution

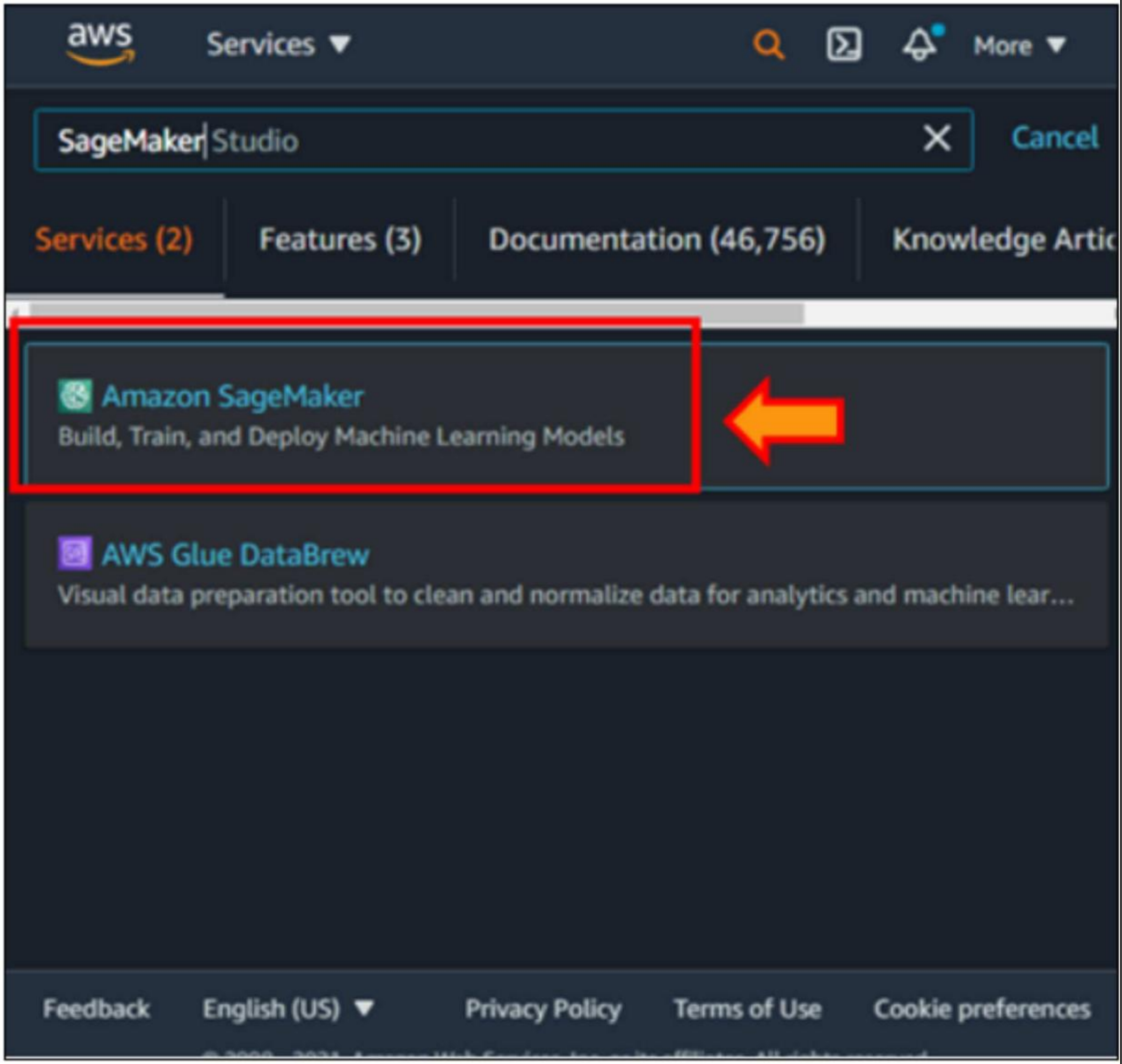
Step 1: Navigate to the Jupyter Notebook

Note: Before starting the lab, you must have a Jupyter notebook. You can create it by following the steps given in the link:

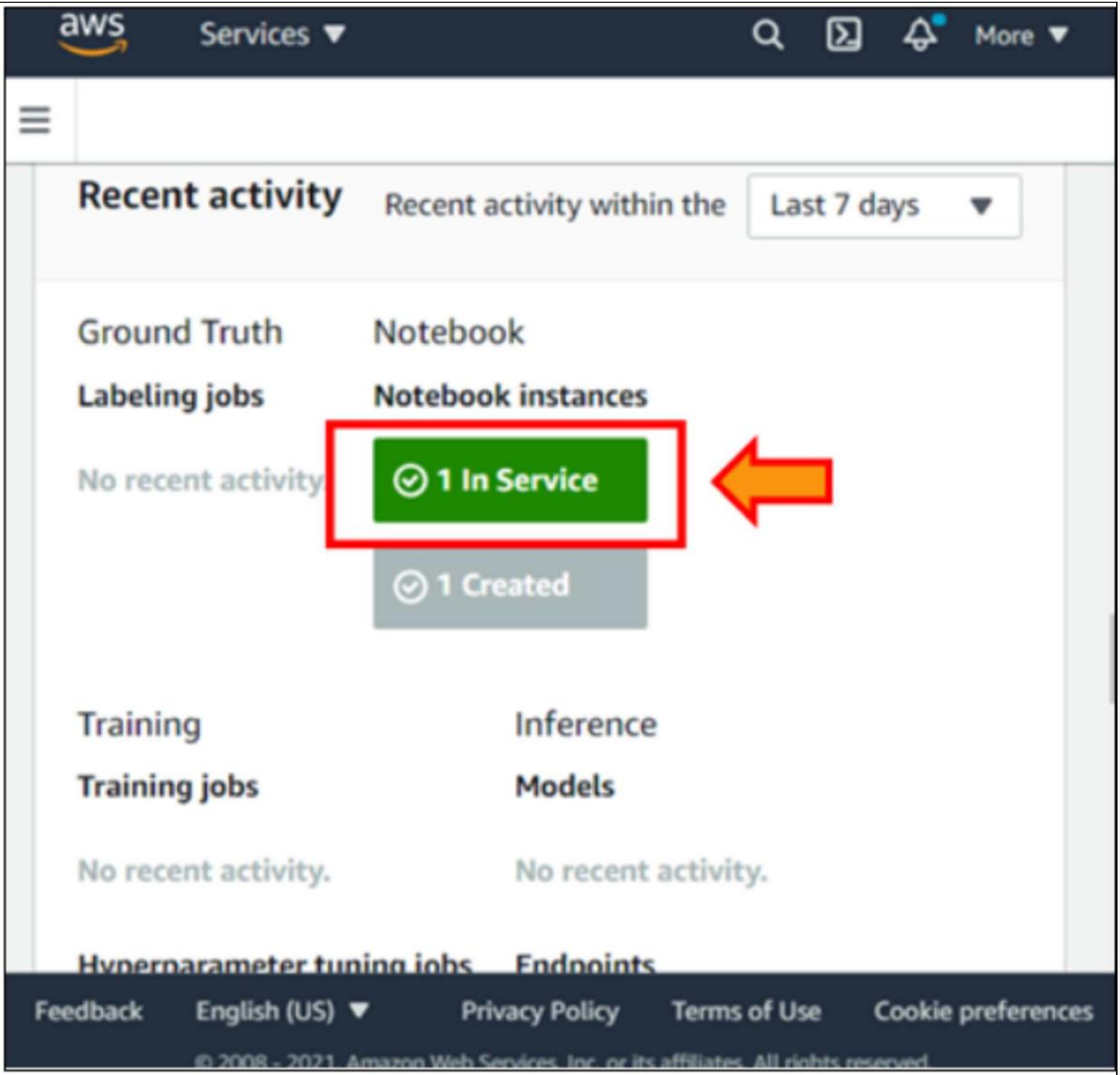
https://docs.aws.amazon.com/dlami/latest/devguide/setup-jupyter.html

1. Log in to the **AWS Console**.

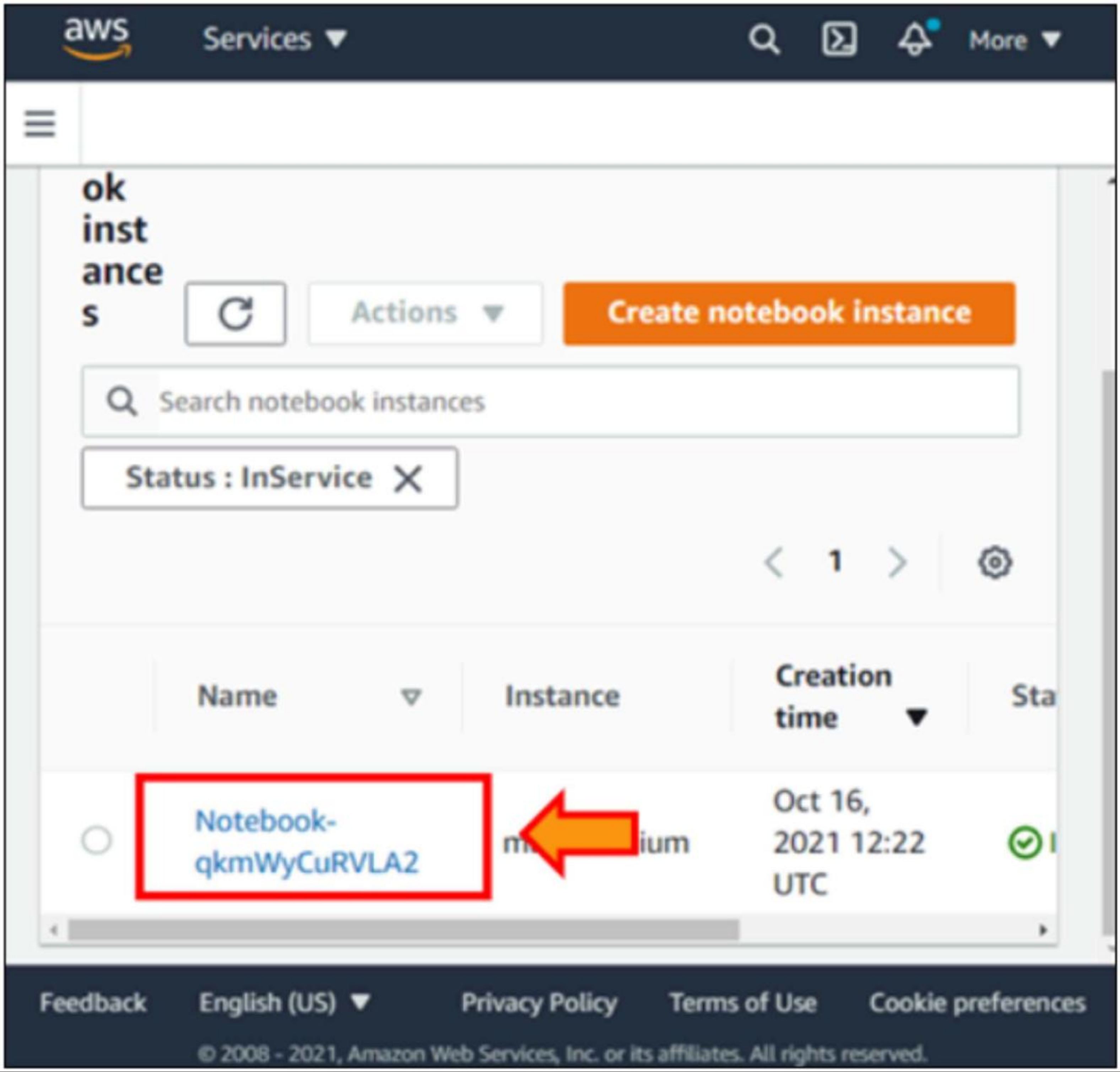
2. Go to **Services** and click on **SageMaker.**



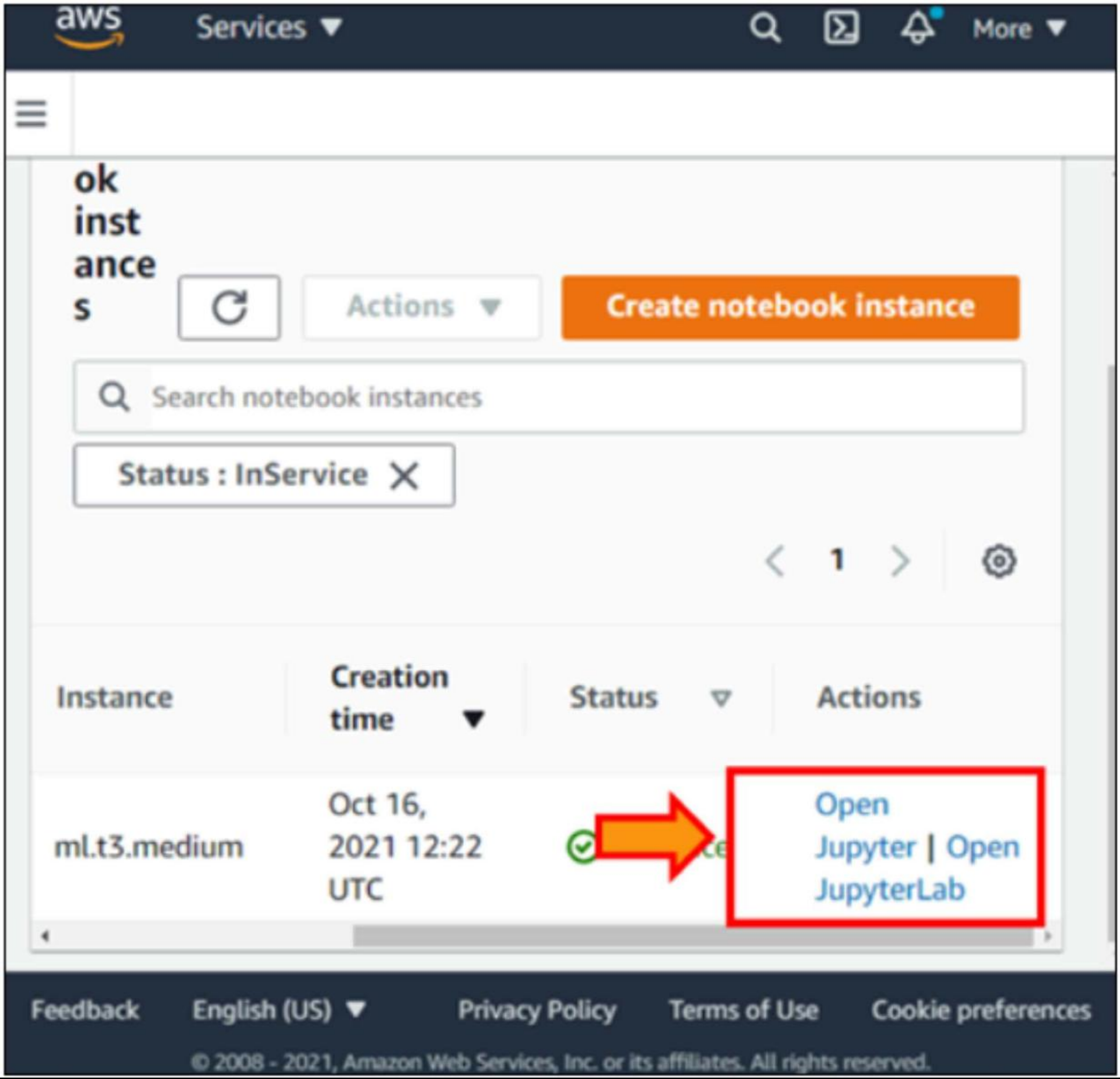
3. Click on **In-Service.**



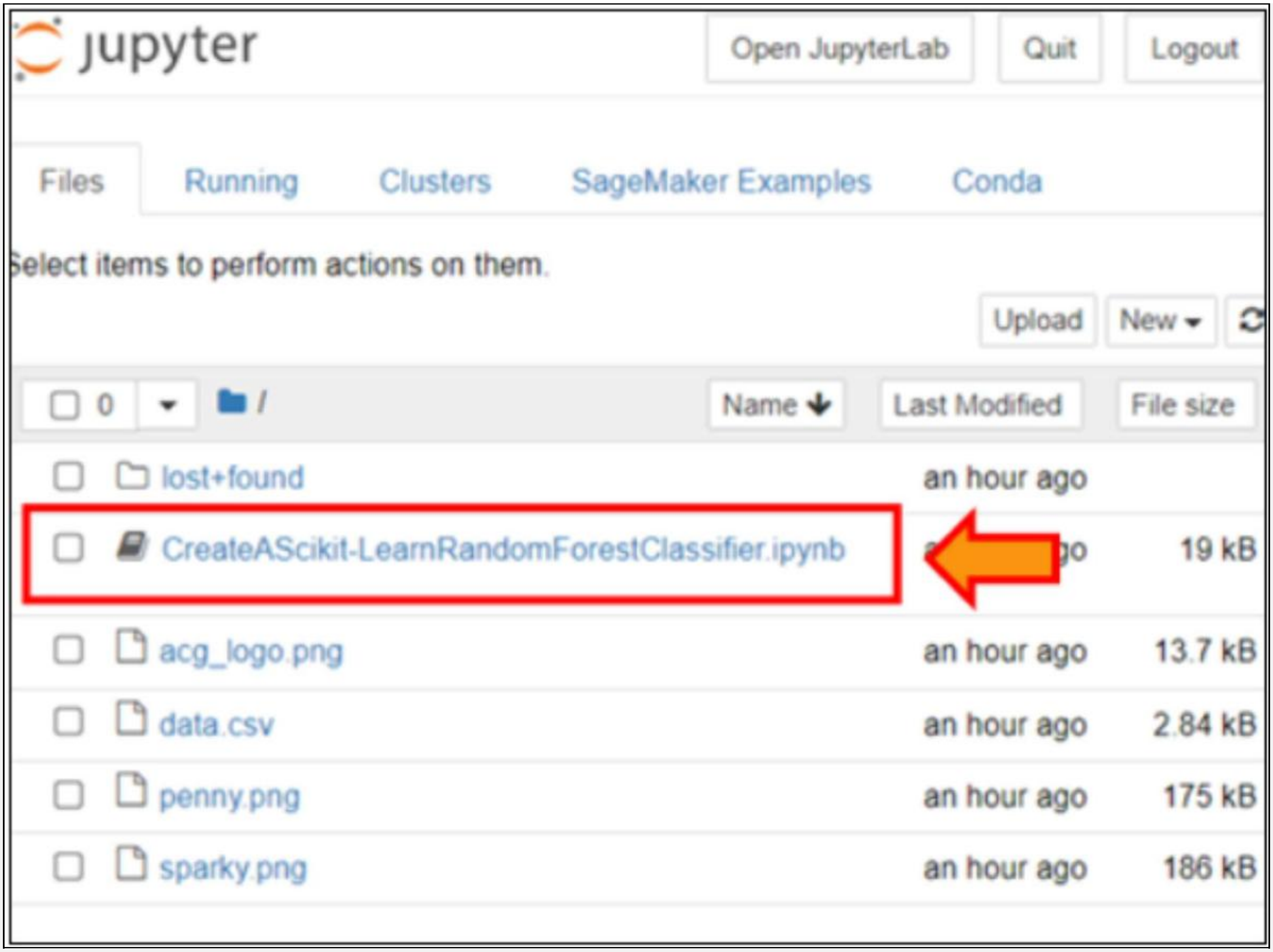
4. Click on the **Notebook instance.**



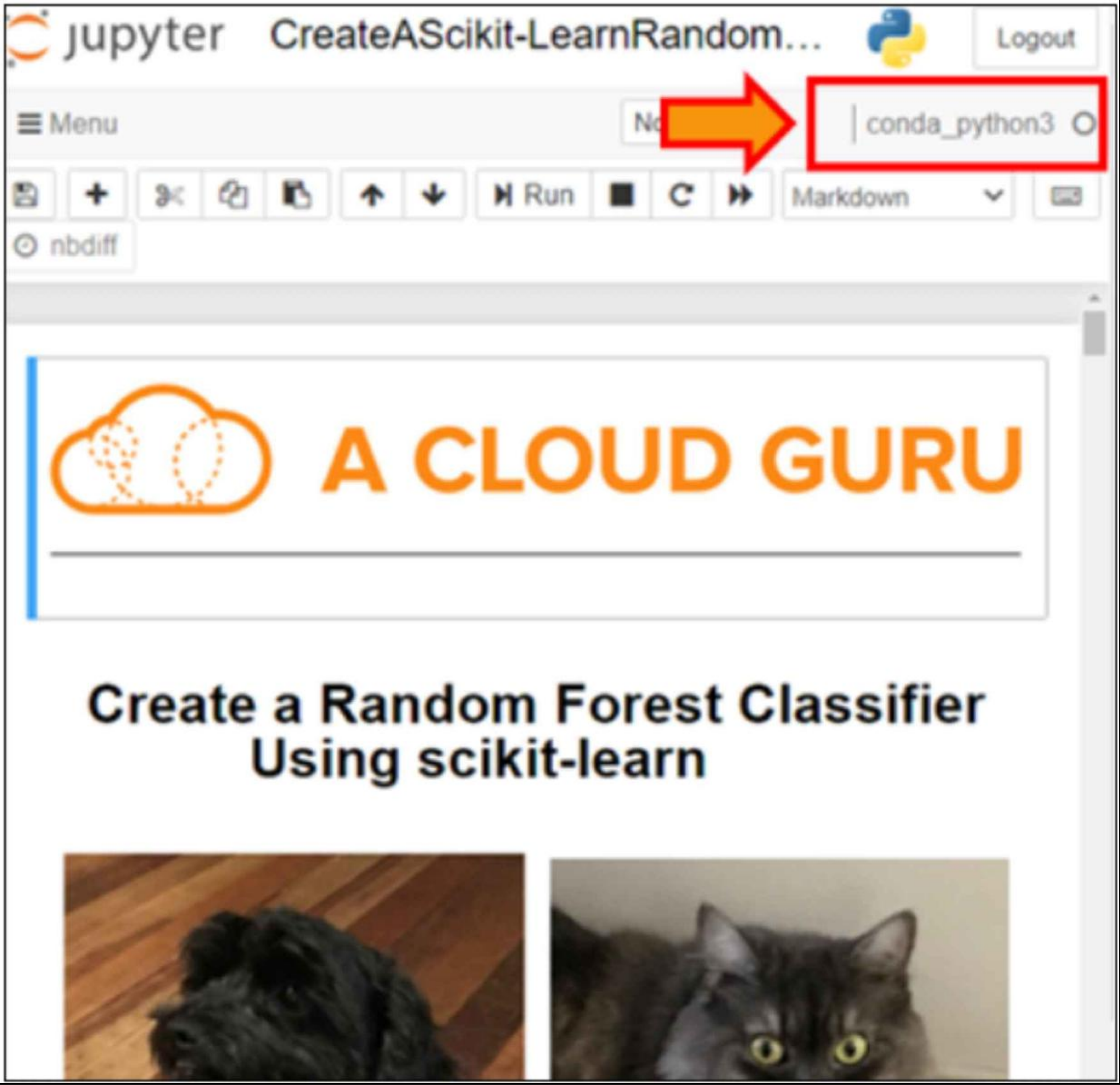
5. Click on **Open Jupyter.**



6. Click on **CreateAScikit-LearnRandomForestClassifier.ipynb**



7. Make sure you are using the **conda_python3** kernel.



8. We will import all the necessary libraries. We use a magic command to have matplotlib dump the information directly into the notebook. We will use a random seed that ensures that our models are reproducible.

```
In [1]: import numpy as np
        import pandas as pd
        from pandas.api.types import CategoricalDtype
        import matplotlib.pyplot as plt
        import matplotlib.image as mpimg
        import seaborn as sn
        #Use a magic command to display the graphs better
        %matplotlib inline

        from sklearn.ensemble import RandomForestClassifier
        from sklearn.tree import DecisionTreeClassifier
        from sklearn.tree import export_graphviz
        from sklearn.metrics import confusion_matrix
        from sklearn.metrics import roc_curve, auc
        from sklearn.model_selection import train_test_split

        # Set the random seed to have reproducible results
        np.random.seed(42)

        # These two includes help us render a graph
        from subprocess import call
        from IPython.display import Image
```

Step 2: Load and Prepare the Data

1. We have a dataset of people who like dogs or cats. We begin by loading the data. Pandas have a very simple way of importing CSV data, which is how our survey results are saved.

```
In [2]: df = pd.read_csv("data.csv")
```

2. We will take a look at the top 10 results.

```
In [3]: df.head(10)
```

Out[3]:

	walking	running	color	distance_walked	dogs_cats
0	1	1	red	1	1
1	1	1	blue	2	1
2	1	1	green	3	1
3	1	1	blue	4	1
4	1	1	blue	5	1
5	1	1	red	6	1
6	1	1	green	7	1
7	1	1	green	8	1
8	1	1	blue	9	1
9	1	1	red	1	1

3. We will check to see what data types we have.

```
In [4]: df.dtypes

Out[4]: walking              int64
        running              int64
        color               object
        distance_walked      int64
        dogs_cats            int64
        dtype: object
```

4. We will start by changing the labels of some of the columns and identifying dogs_cats as an out label for our first data preparation.

```
In [5]: df.columns = ['walk', 'run', 'color', 'distance', 'label']
```

5. The data is not adequately shuffled. We come across a lot of information that is similar. We do not wish to examine the remainder of our data separately to better understand. Rather, we want to describe the data as its whole. We will use Pandas to provide extra statistical information about the data.

```
In [6]: df.describe()
```

Out[6]:

	walk	run	distance	label
count	199.000000	199.000000	199.000000	199.000000
mean	0.497487	0.497487	4.979899	0.321608
std	0.501255	0.501255	2.597512	0.468272
min	0.000000	0.000000	1.000000	0.000000
25%	0.000000	0.000000	3.000000	0.000000
50%	0.000000	0.000000	5.000000	0.000000
75%	1.000000	1.000000	7.000000	1.000000
max	1.000000	1.000000	9.000000	1.000000

6. We will structure the data in columns for our next data preparation phase so the model can comprehend it better.

Even though the **walk, run, and label** columns are integers, they indicate a binary value of 1 or 0. This relates to True or False for liking that activity for walk and run. This indicates whether you favor cats or dogs as a label. We will use the boolean data type instead of numbers for these fields.

Color_type is not just any text. It belongs to one of three groups (red, green, or blue). We will convert it to a definite data type instead of considering it as text.

```
In [7]: df['walk'] = df['walk'].astype('bool')
        df['run'] = df['run'].astype('bool')
        color_type = CategoricalDtype(categories=['red', 'green',
        df['color'] = df['color'].astype(color_type)
        df['label'] = df['label'].astype('bool')
```

7. We will check to see what data types we have.

```
In [8]: df.dtypes

Out[8]: walk            bool
        run             bool
        color       category
        distance       int64
        label           bool
        dtype: object
```

8. The algorithm will not understand color as a list of strings. When we try to match the data with the current data, we may expect to obtain the following error:

`ValueError: could not convert string to float: 'green.'`

As a result, we utilize the 'get dummies' command to conduct one-hot encoding straight into the dataframe.

```
In [9]: df = pd.get_dummies(df, prefix=['color'])
```

9. We will take a quick look at our data now that the data formatting is complete.

```
In [10]: df.head(10)

Out[10]:
```

	walk	run	distance	label	color_red	color_green	color_blue
0	True	True	1	True	1	0	0
1	True	True	2	True	0	0	1
2	True	True	3	True	0	1	0
3	True	True	4	True	0	0	1
4	True	True	5	True	0	0	1
5	True	True	6	True	1	0	0
6	True	True	7	True	0	1	0
7	True	True	8	True	0	1	0
8	True	True	9	True	0	0	1
9	True	True	1	True	1	0	0

10. We use train_test_split from scikit-learn to split out 20% of the data for testing, leaving us with 80% for training. We also set random_state to ensure that the data split is random yet reproducible.

```
In [11]: X_train, X_test, y_train, y_test = train_test_split(df.dro
```

Step 3: Create the scikit-learn Model

1. We are going to create a RandomForestClassifier model.

```
In [12]: model = RandomForestClassifier(max_depth=5)
```

2. Now, we will fit the model on the training data.

```
In [13]: model.fit(X_train, y_train)

Out[13]: RandomForestClassifier(max_depth=5)
```

Step 4: Evaluate the Model

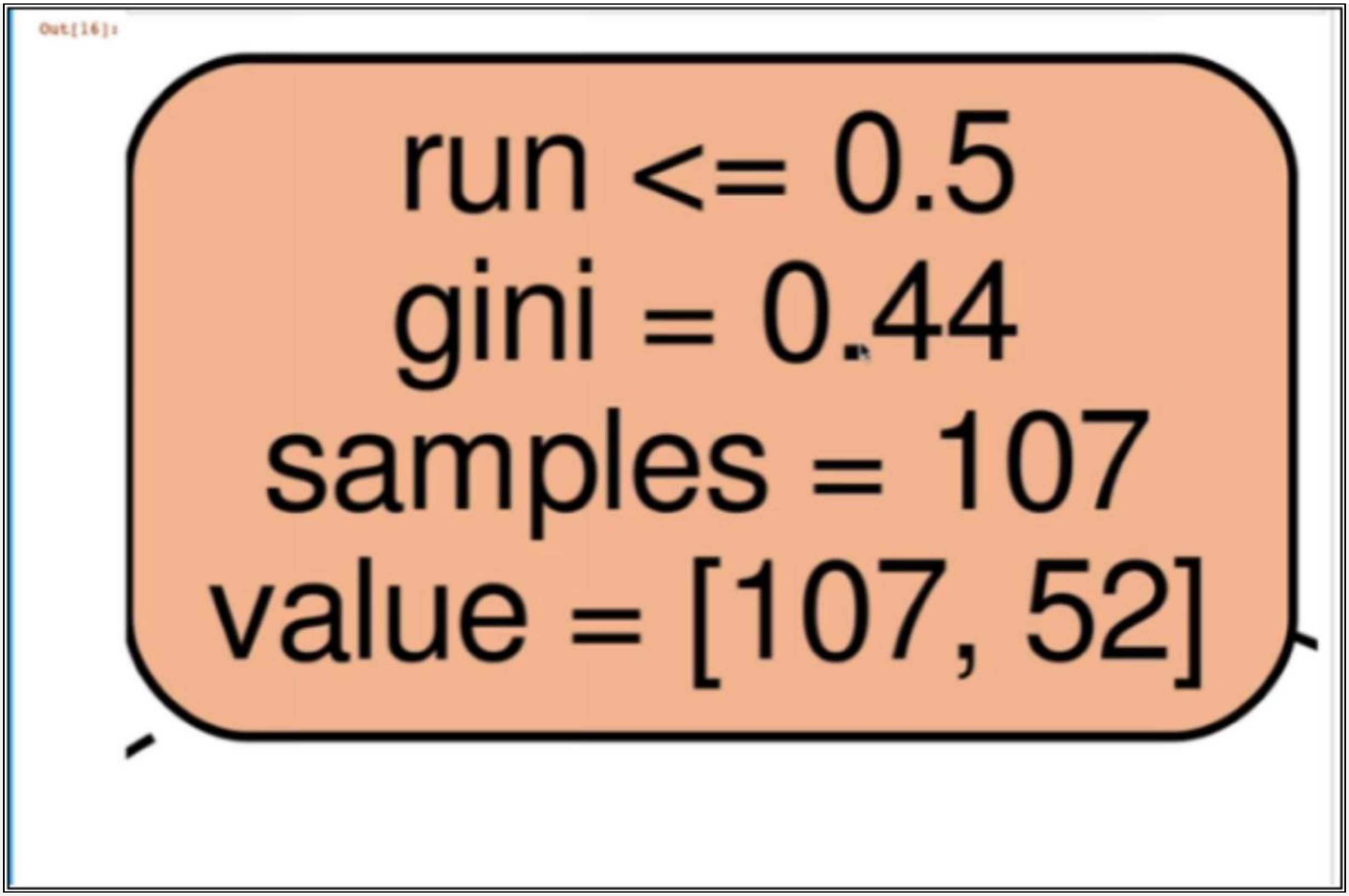
1. First, we will grab the estimator from the trained model.

```
In [14]: estimator = model.estimators_[0]
         feature_names = [i for i in X_train.columns]
```

2. Then, we will use export_graphviz to get a graphical representation of one of the trees in our forest.

```
In [15]: export_graphviz(estimator, out_file='tree.dot',
                         feature_names = feature_names,
                         rounded = True,
                         filled = True)

         # Use CLI Graphviz to create a png file from our graph data:
         call(['dot', '-Tpng', 'tree.dot', '-o', 'tree.png', '-Gdpi=600'])

         Image(filename = 'tree.png')

Out[15]:
```

You may double-click to zoom the image.



3. Now, we will run our testing data through the model.

```
In [16]: y_predict = model.predict(X_test)
```

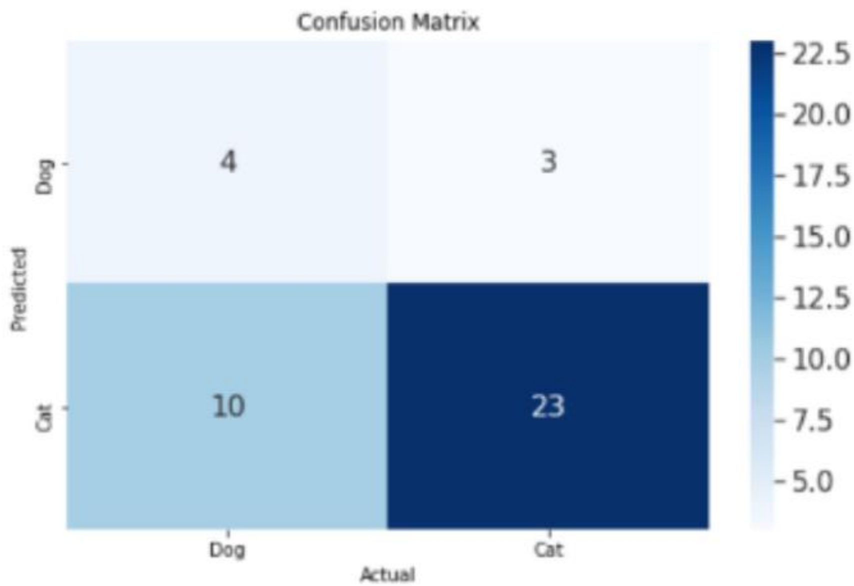
4. To see how it performed, we will use confusion_matrix to create a confusion matrix for the test data predictions.

```
In [17]: conf_matrix = confusion_matrix(y_test, y_predict)
         conf_matrix

Out[17]: array([[23,  3],
                [10,  4]])
```

5. Since it is not easy to read, we will add some style to it using the seaborn library.

```
In [18]: tn, fp, fn, tp = conf_matrix.ravel()
         cleaned_conf_matrix = np.array([[tp, fp], [fn, tn]])
         y_true = ["Dog", "Cat"]
         df_cm = pd.DataFrame(cleaned_conf_matrix, columns=y_true, index=y_true)
         df_cm.index.name = 'Predicted'
         df_cm.columns.name = 'Actual'

         df_cm.dtypes

         plt.figure(figsize = (8,5))
         plt.title('Confusion Matrix')
         sn.set(font_scale=1.4)#for label size
         sn.heatmap(df_cm, cmap="Blues", annot=True, annot_kws={"size": 16})# font size
```

```
Out[18]: <AxesSubplot:title={'center':'Confusion Matrix'}, xlabel='Actual', ylabel='Predicted'>
```

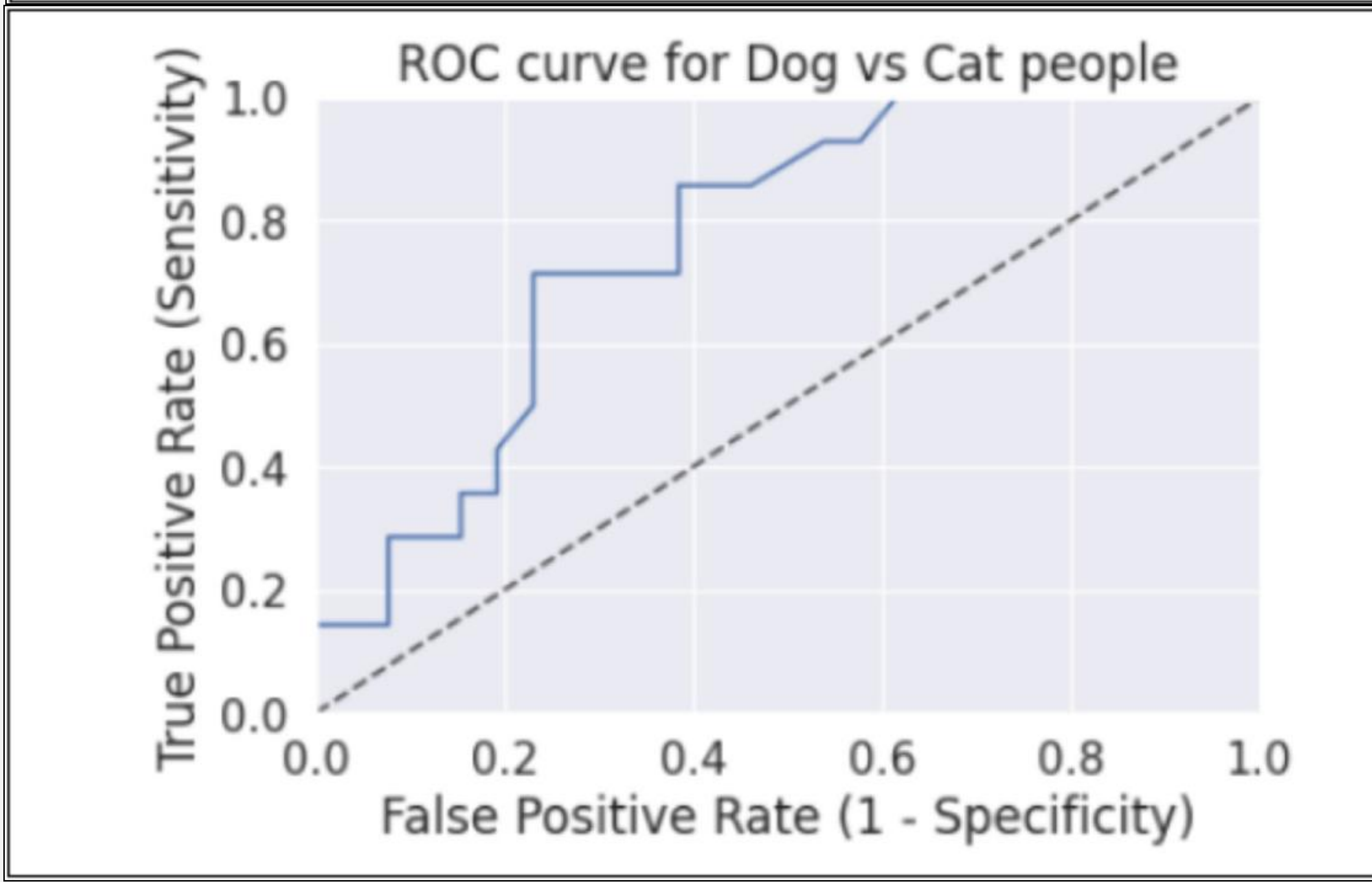


6. From the confusion matrix, we will calculate sensitivity and specificity.

```
In [19]: sensitivity = tp / (tp + fn)
         print('Sensitivity : ', sensitivity )

         specificity = tn / (tn + fp)
         print('Specificity : ', specificity)

         Sensitivity :  0.2857142857142857
         Specificity :  0.8846153846153846
```

7. Now, we will look at the ROC graph.

```
In [20]: y_pred_proba = model.predict_proba(X_test)[:, 1]
         fpr, tpr, thresholds = roc_curve(y_test, y_pred_proba)

         fig, ax = plt.subplots()
         ax.plot(fpr, tpr)
         ax.plot([0, 1], [0, 1], transform=ax.transAxes, ls="--", c=".3")
         plt.xlim([0.0, 1.0])
         plt.ylim([0.0, 1.0])
         plt.rcParams['font.size'] = 12
         plt.title('ROC curve for Dog vs Cat people')
         plt.xlabel('False Positive Rate (1 - Specificity)')
         plt.ylabel('True Positive Rate (Sensitivity)')
         plt.grid(True)
```



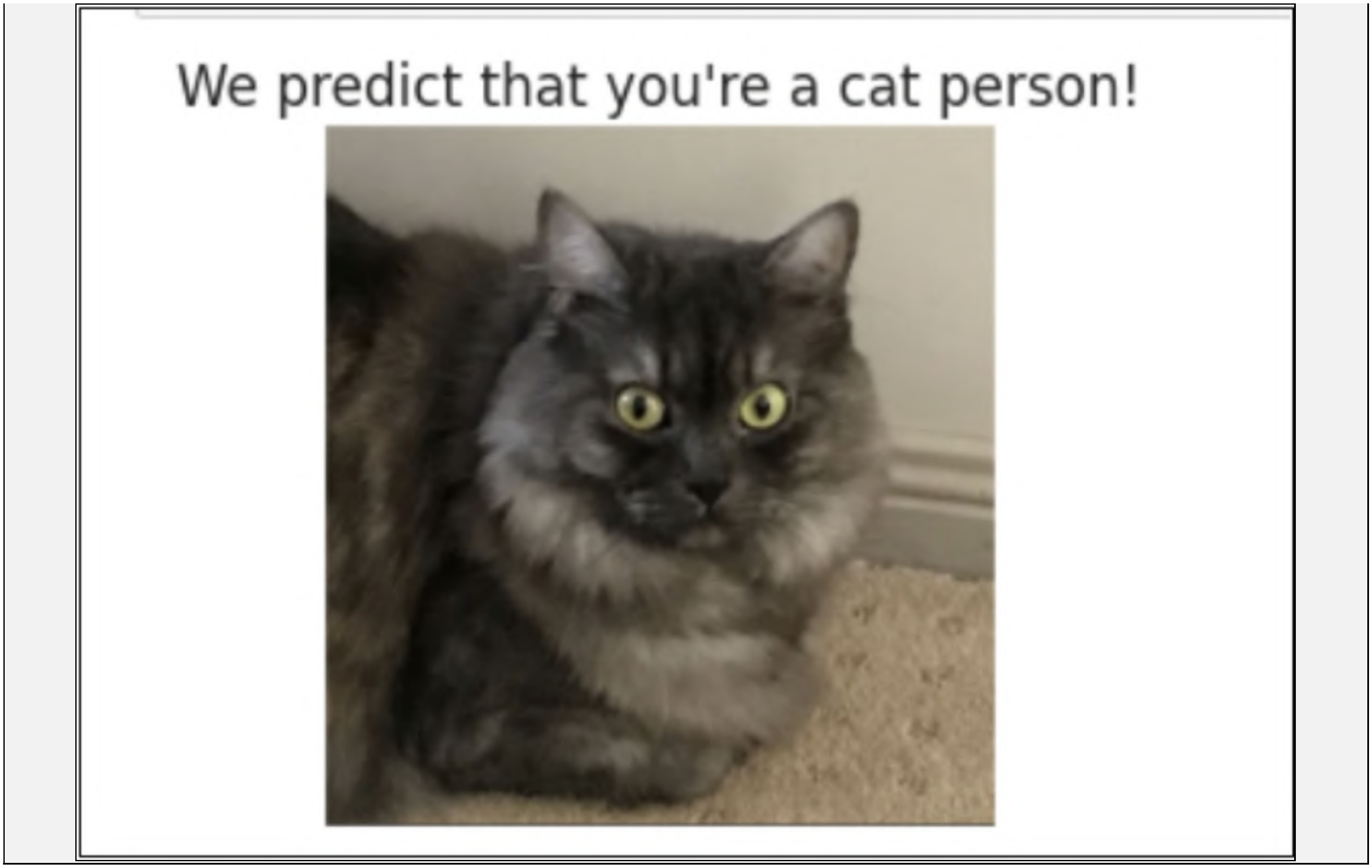
8. Now, we will calculate the AUC (area under the curve).

```
In [21]: auc(fpr, tpr)

Out[21]: 0.7664835164835165
```

Step 5: Predict for Yourself

1. You can change the values for the survey questions below and see what the model predicts for you!

```
In [ ]: # Do you like walking?  1=yes, 0=no
        like_walking = 1

        # Do you like running?  1=yes, 0=no
        like_running = 0

        # How far do you walk in a day?
        distance = 3

        # What is your favorite color? Answer ONLY ONE with a 1
        red = 0
        green = 0
        blue = 1

        # Get the prediction:
        prediction = model.predict([[like_walking, like_running, distance, red, green, blue]])

        # The following is all display code:
        if prediction[0]:
            animal = 'dog'
            pic = 'sparky.png'
        else:
            animal = 'cat'
            pic = 'penny.png'

        img=mpimg.imread(pic)
        imgplot = plt.imshow(img)
        plt.grid(False)
        plt.xticks([])
        plt.yticks([])
        plt.title("We predict that you're a %s person!" % animal)
        plt.show()
```



Lab 14: Evaluation and Optimization

Certifications	AWS Machine Learning - Specialty
Cloud Provider	AWS
Time Required	1 Hr.
Tags	CloudWatch, S3, SageMaker, AWS

Lab Prerequisites

- Familiarity with basic AWS Cloud Computing concepts and terminology
- An AWS account with an active subscription

Service Introduction

AWS SageMaker

Amazon SageMaker is a fully managed service that allows any developer or data scientist to quickly create, train, and deploy Machine Learning (ML) models. SageMaker makes it easy to create high-quality models by removing the heavy lifting from each phase of the machine learning process.

AWS Simple Storage Service (S3)

Amazon S3 is a type of object storage that allows you to store and recover any quantity of data from any location. It is a low-cost storage solution with business resilience, reliability, efficiency, privacy, and infinite expansion.

Amazon S3 is a web service that allows you to store and retrieve an infinite quantity of data from any place and at any time. You may quickly create projects that integrate cloud-native storage using this service. Because Amazon S3 is easily customizable and you only pay for what you use, you can start small and scale up as needed without sacrificing performance or dependability.

Amazon S3 is also built to be highly adaptable. Instead of finding out how to store their data, Amazon S3 allows developers to focus on innovation. You can build a simple FTP system or a complex web application like the Amazon.com retail website, read the same piece of data a million times or only for emergency disaster recovery, and store whatever type and amount of data you desire.

AWS CloudWatch

Amazon CloudWatch is a tracking service for Amazon Web Services (AWS) cloud services and software. Amazon CloudWatch may be used to collect and monitor data, monitor log files, and trigger alarms. Amazon CloudWatch can monitor AWS resources, such as Amazon EC2 instances, Amazon DynamoDB tables, Amazon RDS DB instances, and custom metrics and log files created by your applications and services. You can watch your system's resource use, application performance, and operational health using Amazon CloudWatch. These insights might help you react and keep your app working smoothly.

Case Study Enterprise Retail – LexisNexis Legal & Professional

Background

A leading global source of legal, regulatory, and business information and analytics, LexisNexis Legal & Professional supports customers' efforts to strengthen the rule of law globally, boost productivity, and improve decision-making and outcomes.

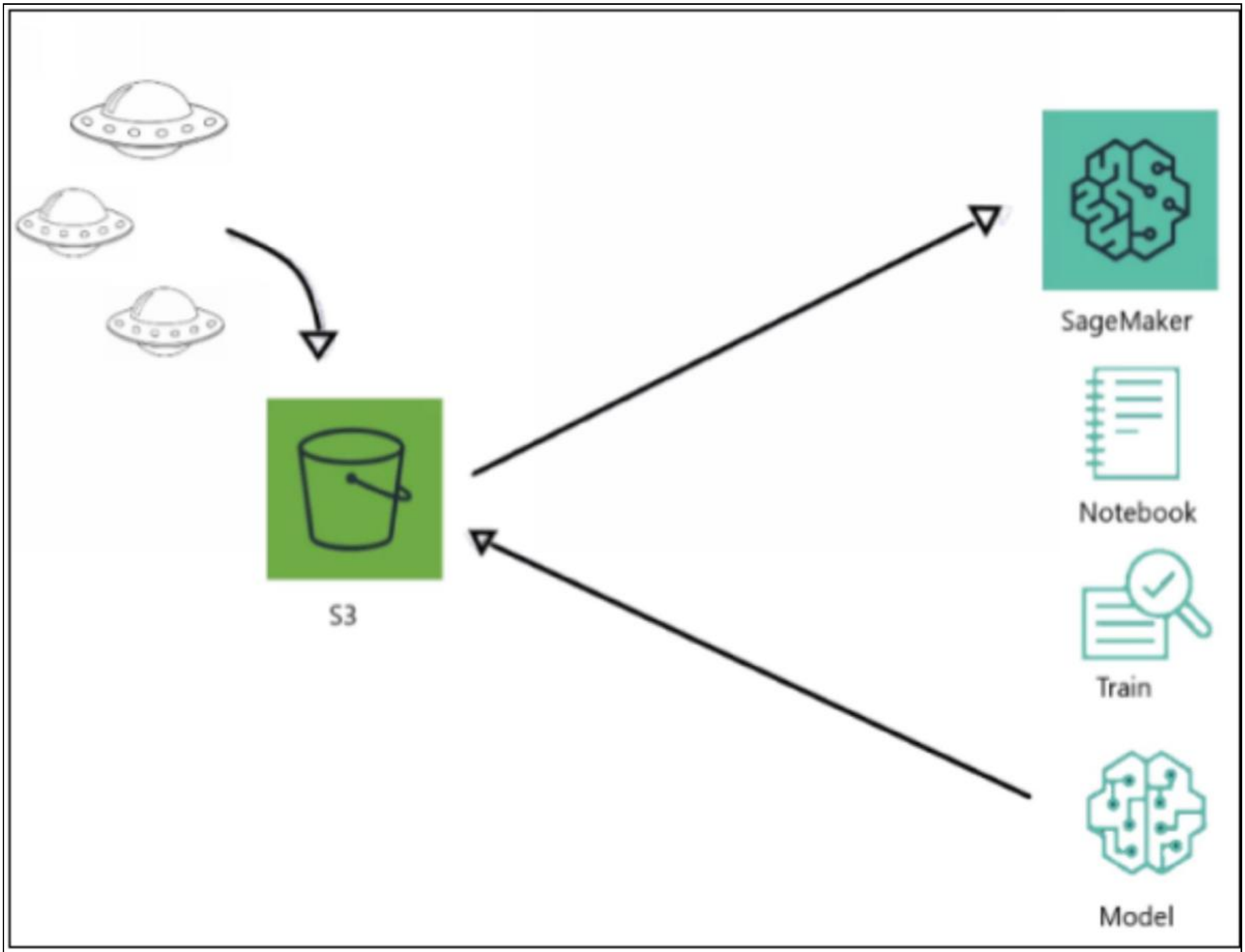
Business Challenge

You work as a Machine Learning developer in LexisNexis Legal & Professional. Your organization gives you a task to train the customer classification model. They also give you the task tune model to find the most optimized model for the problem. Determine if the model is less accurate or about the same. What objective metric would you want to monitor to ensure this? How do you plan on measuring success? Which hyperparameters need to be tuned? What combination do hyperparameters need to be used? How much faster was training time improved?

Proposed Solution

The solution is for you to use AWS services to automate all the tasks. You create an AWS SageMaker hyperparameter tuning job with different ranges of values for the hyperparameter to find the best configuration, which minimizes the validation: objective_loss metric. The reason that you use this metric is that it is used in multi-classification problems. This metric measures the performance of the classification model, and what it does is repeatably calculate the difference between the values that the model predicts and the actual values of a label. Hence, every time it passes over data and makes predictions, it recalculates the objective loss and tries to minimize this value overall. Hence, that is the task here. AWS recommends that you minimize this value when using it as our objective metric.

Lab Diagram



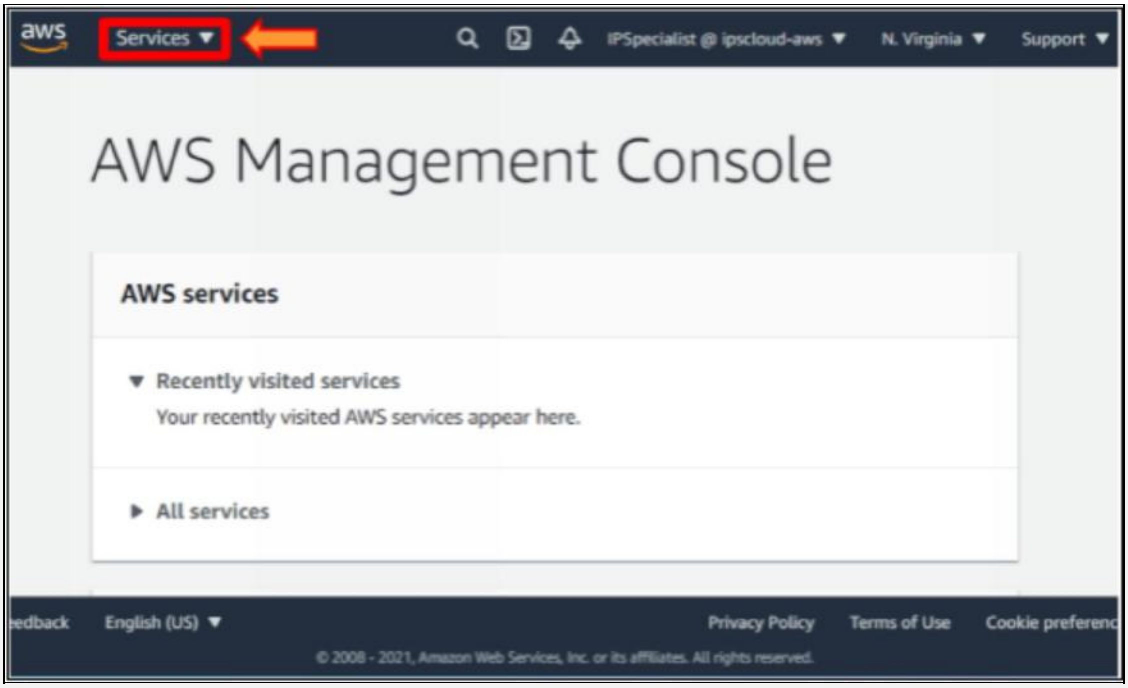
Implementation Steps

1. Create an S3 Bucket.
2. Create SageMaker Hyperparameter Training Job.
3. Analyze CloudWatch Training Logs.
4. Create SageMaker Notebook.

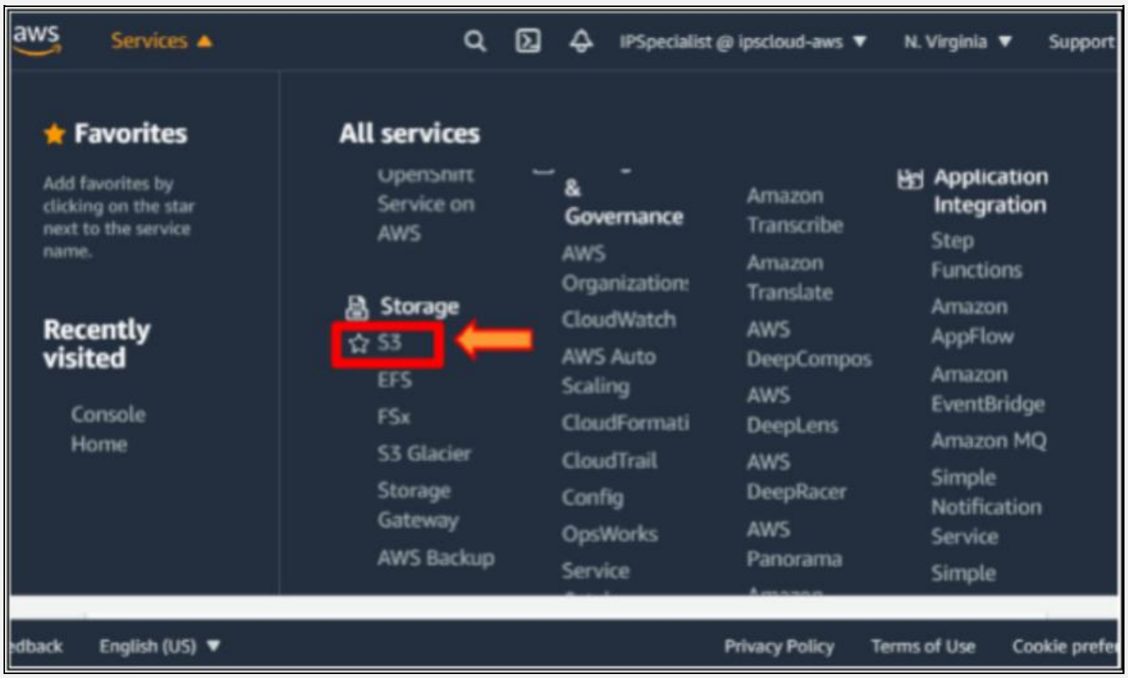
Solution

Step 1: Create S3 Bucket

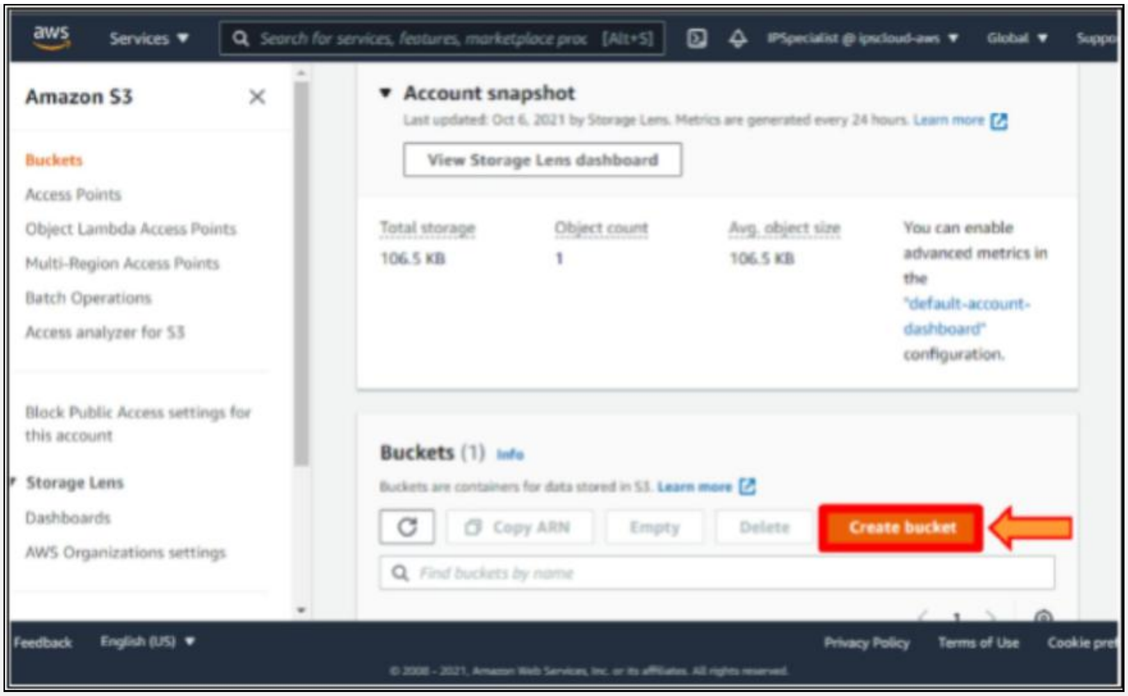
1. Log in to the **AWS Console.**
2. Click on the **Services.**



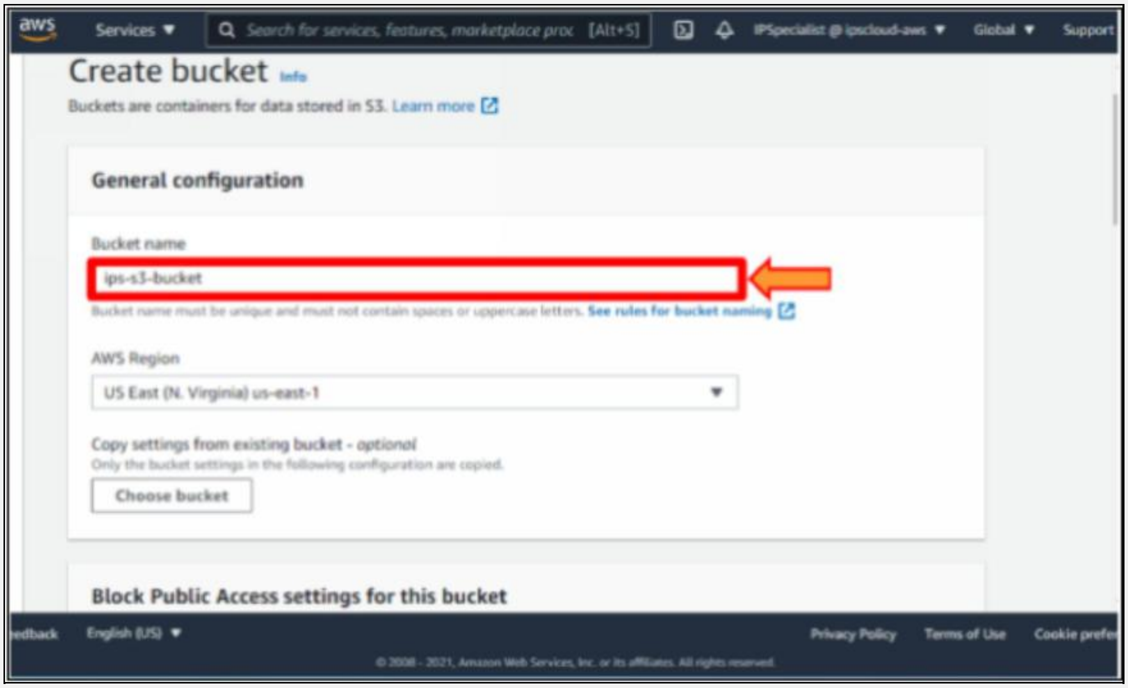
3. Select **S3** from the **Storage.**



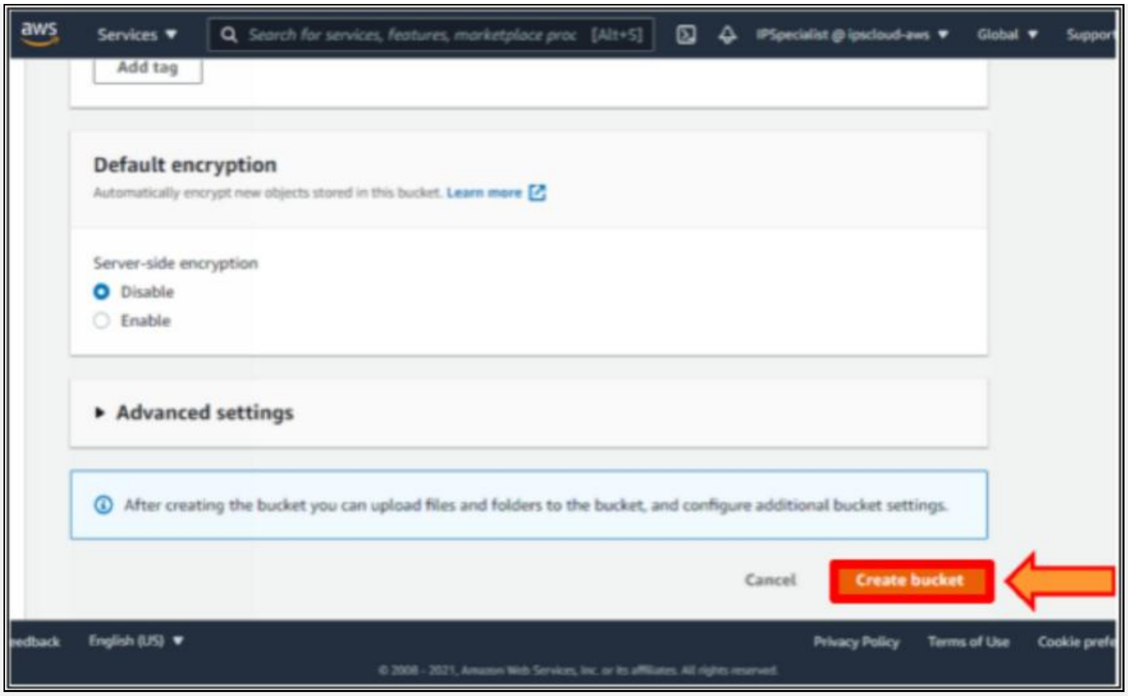
4. Click on the **Create bucket** button.



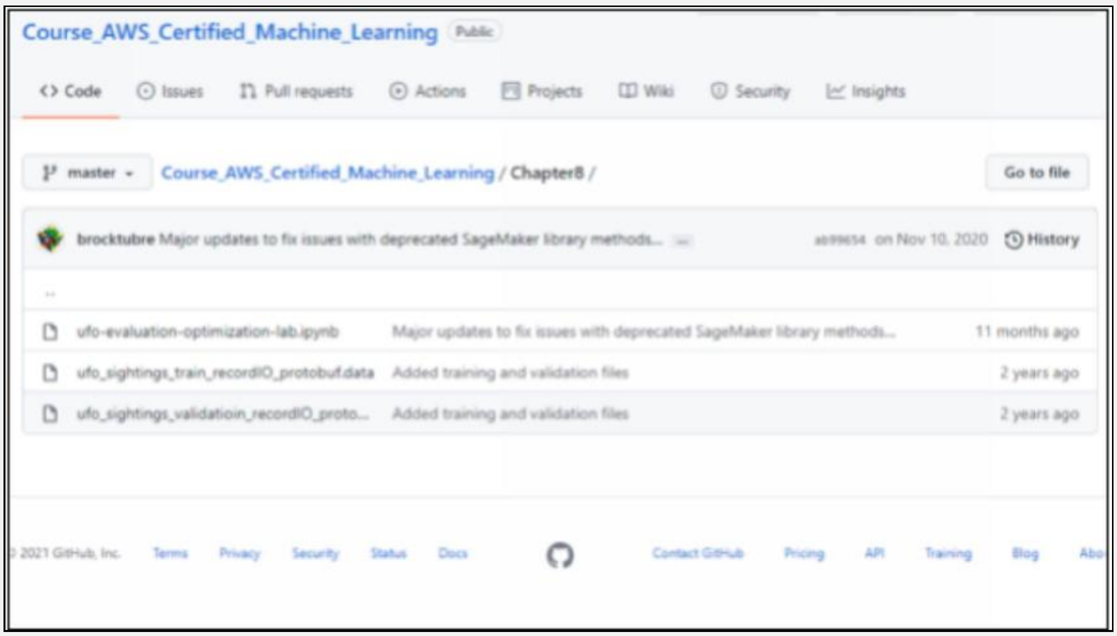
5. Give a bucket name **ips-s3-bucket.**



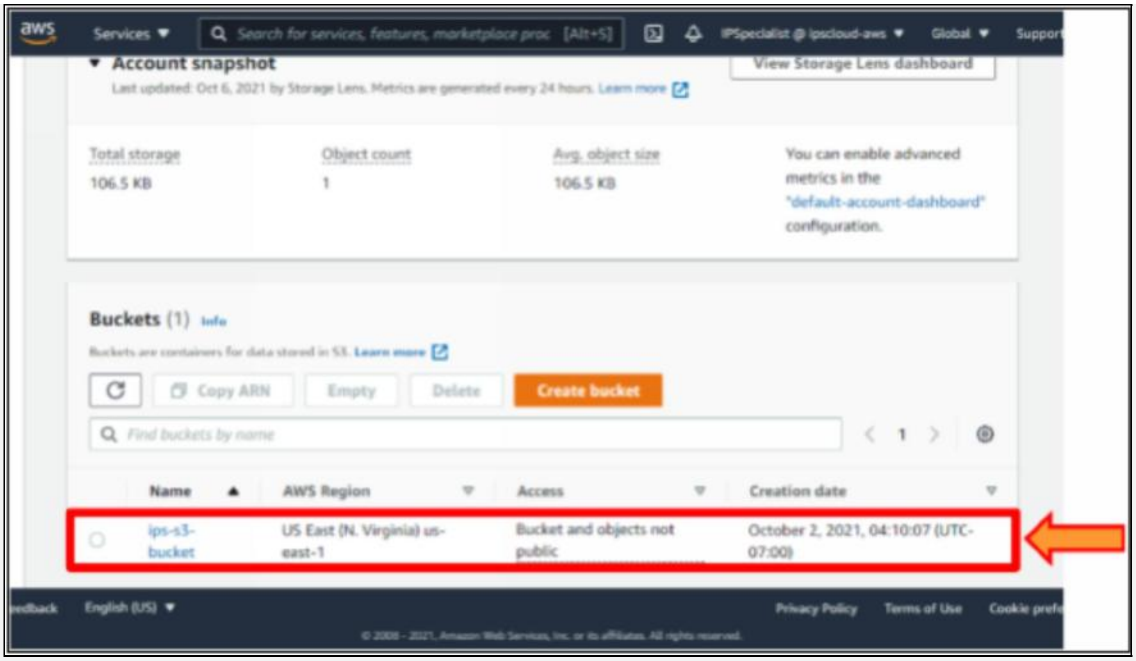
6. Scroll down. Click on the **Create bucket** button.



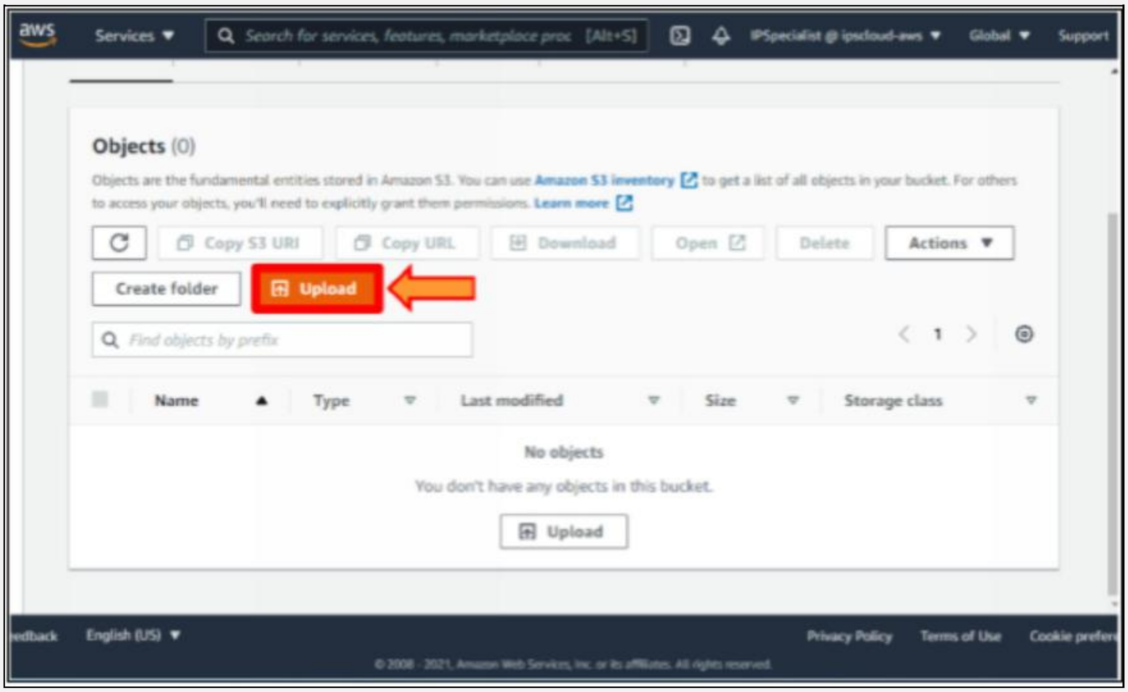
7. Download the **ufo_sightings_train_recordIO_protobuf.data** and **ufo_sightings_validatioin_recordIO_protobuf.data** files from the following Github link:
https://github.com/ACloudGuru-Resources/Course_AWS_Certified_Machine_Learning/tree/master/Chapter



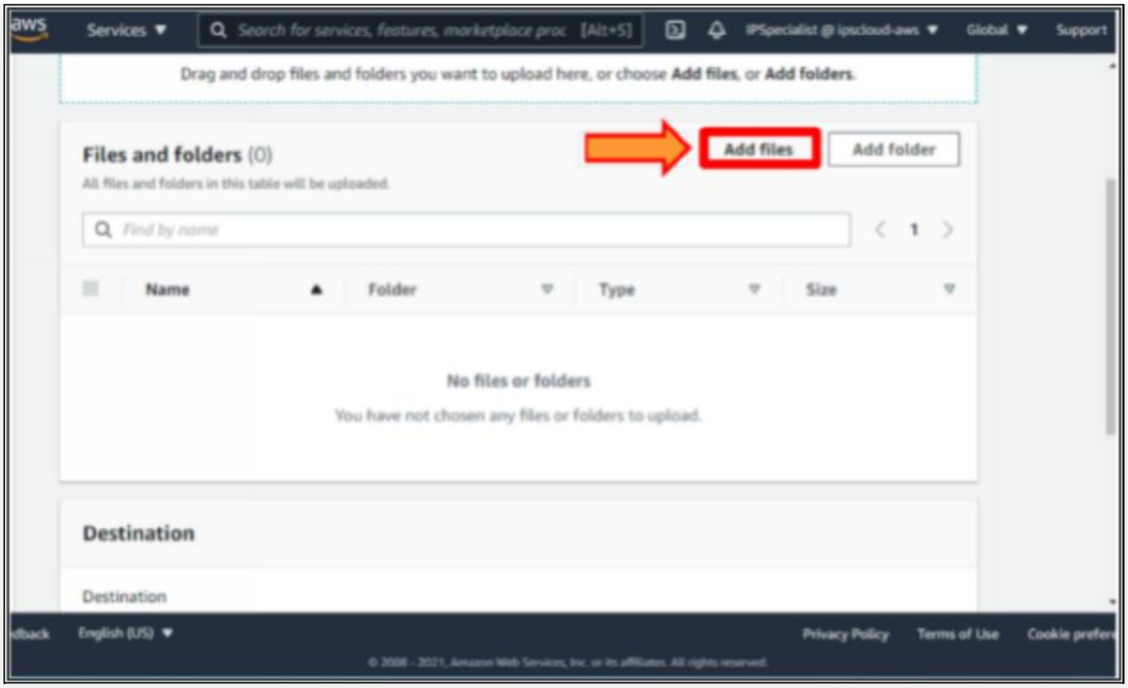
8. Click on **ips-s3-bucket.**



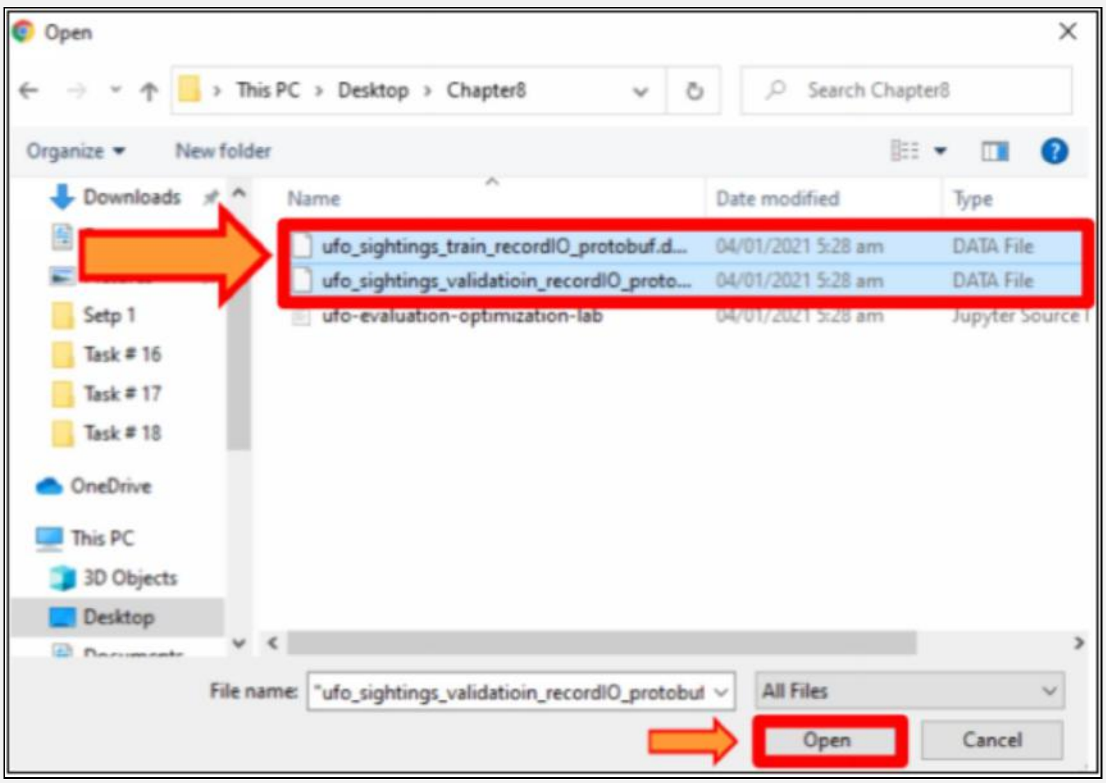
9. Click on the **Upload** button.



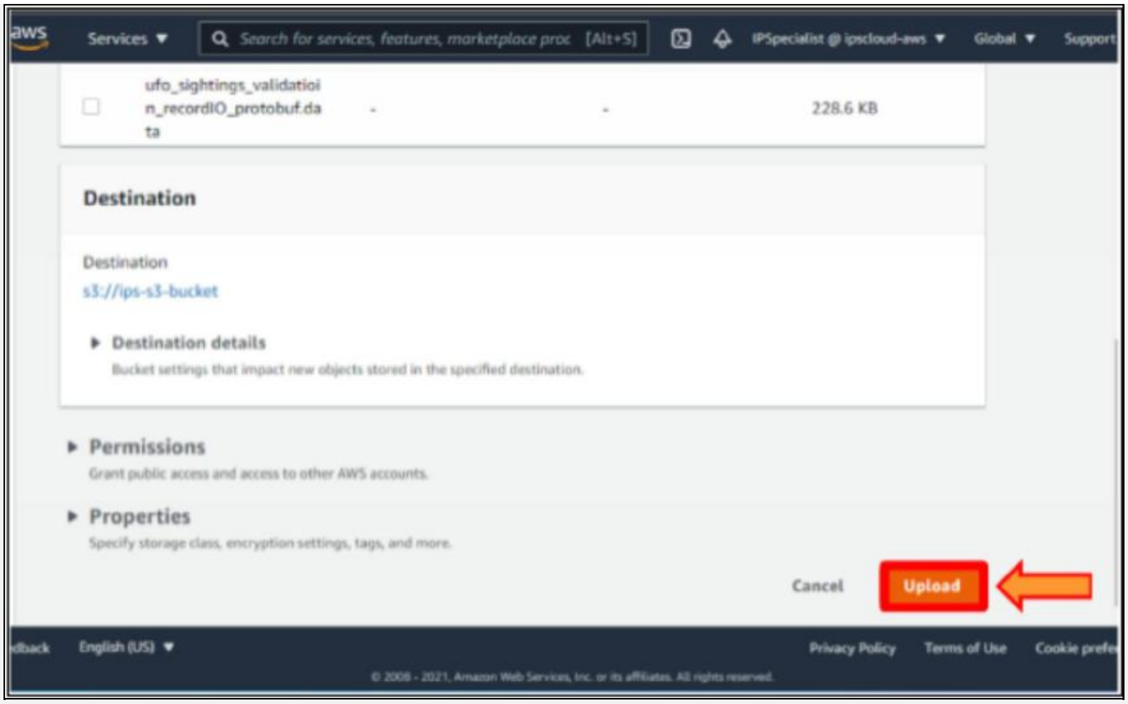
10. Click on the **Add files** button.



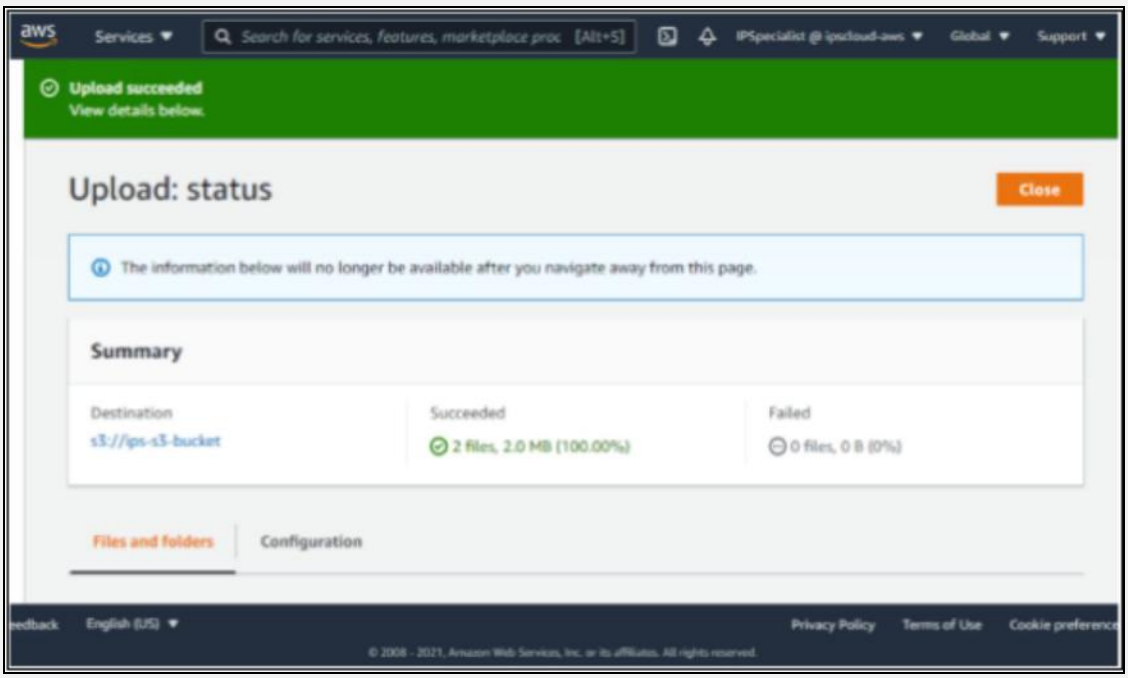
11. Select the **ufo_sightings_train_recordIO_protobuf.data** and **ufo_sightings_validatioin_recordIO_protobuf.data** files.
12. Click on the **Open** button.



13. Scroll down. Click on the **Upload** button.

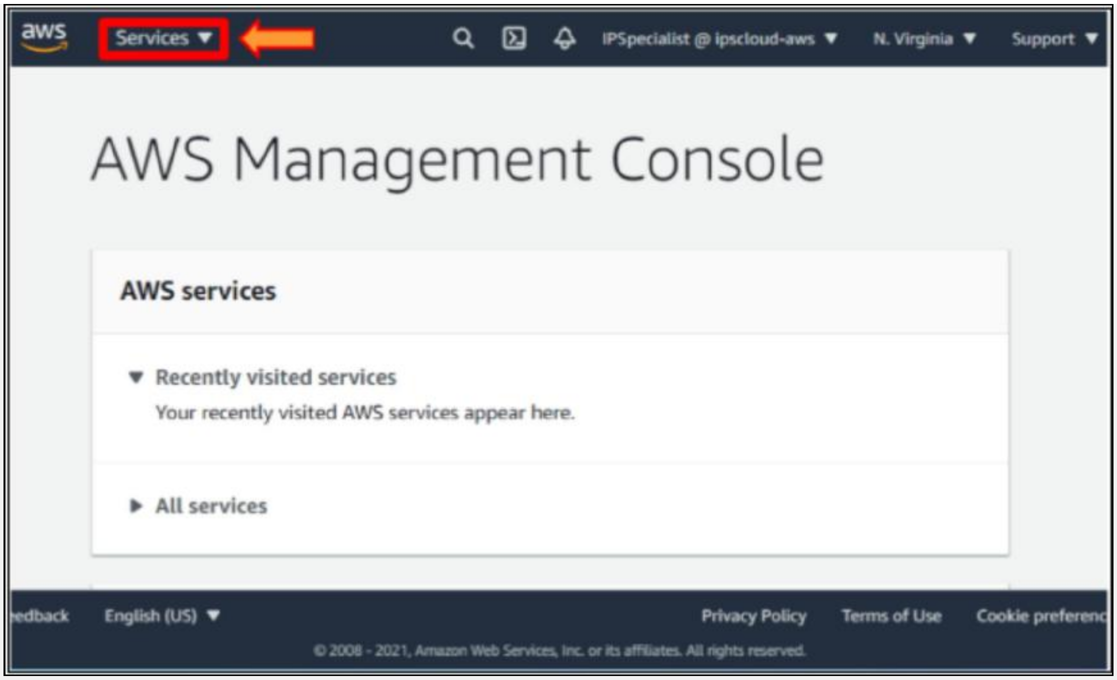


Hence, the files have been uploaded.

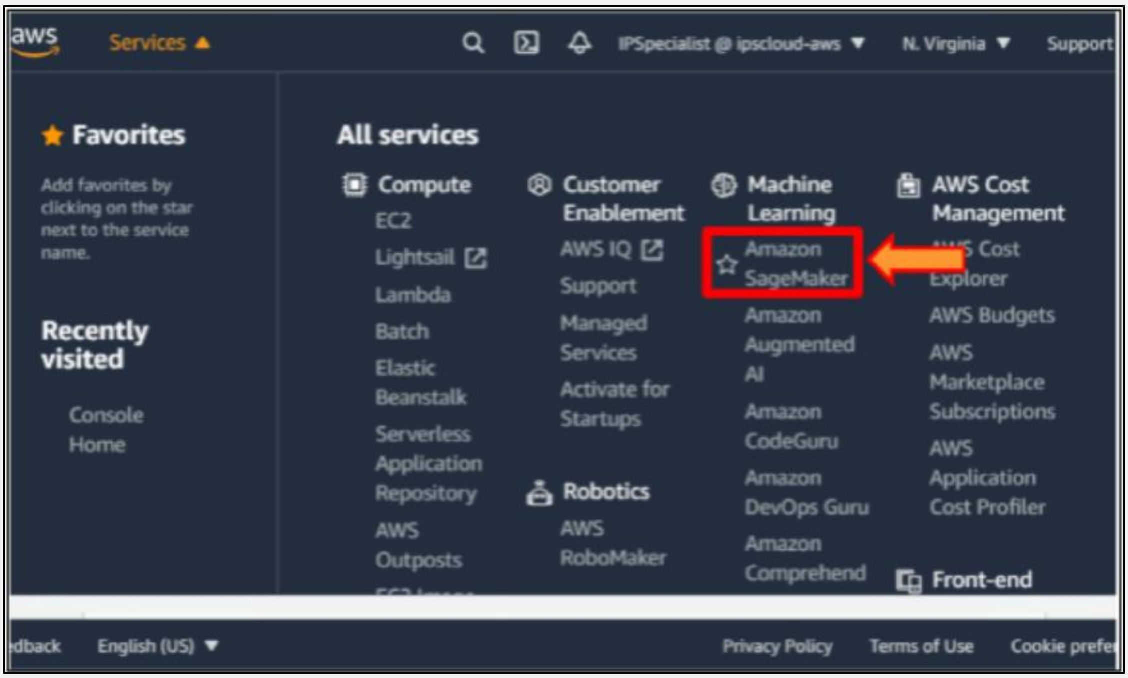


Step 2: Create SageMaker Hyperparameter Training Job

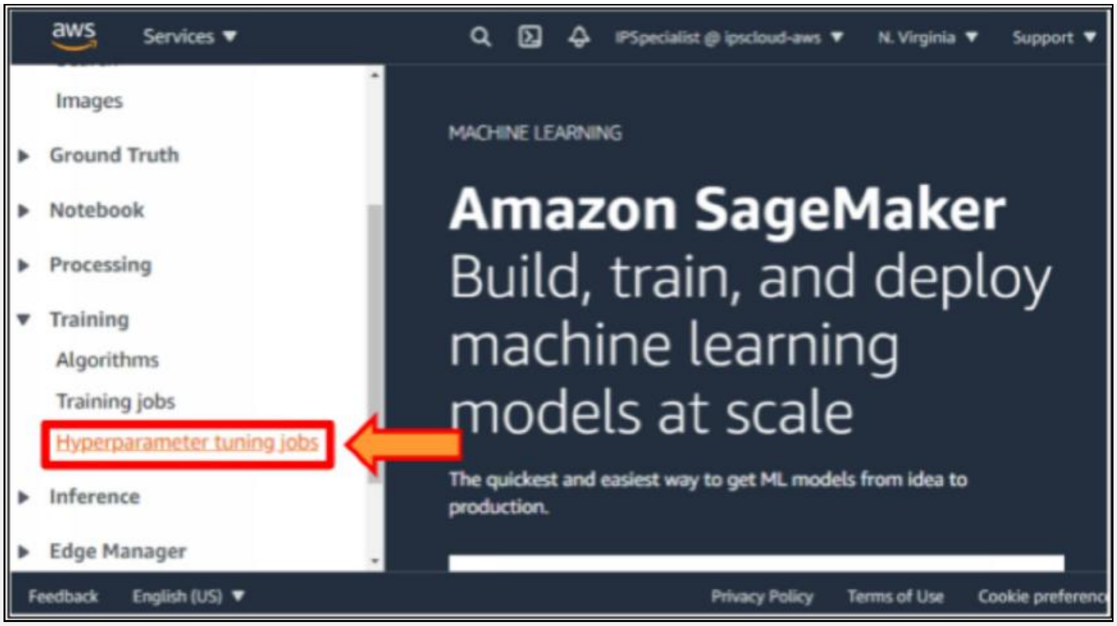
1. Click on **Services.**



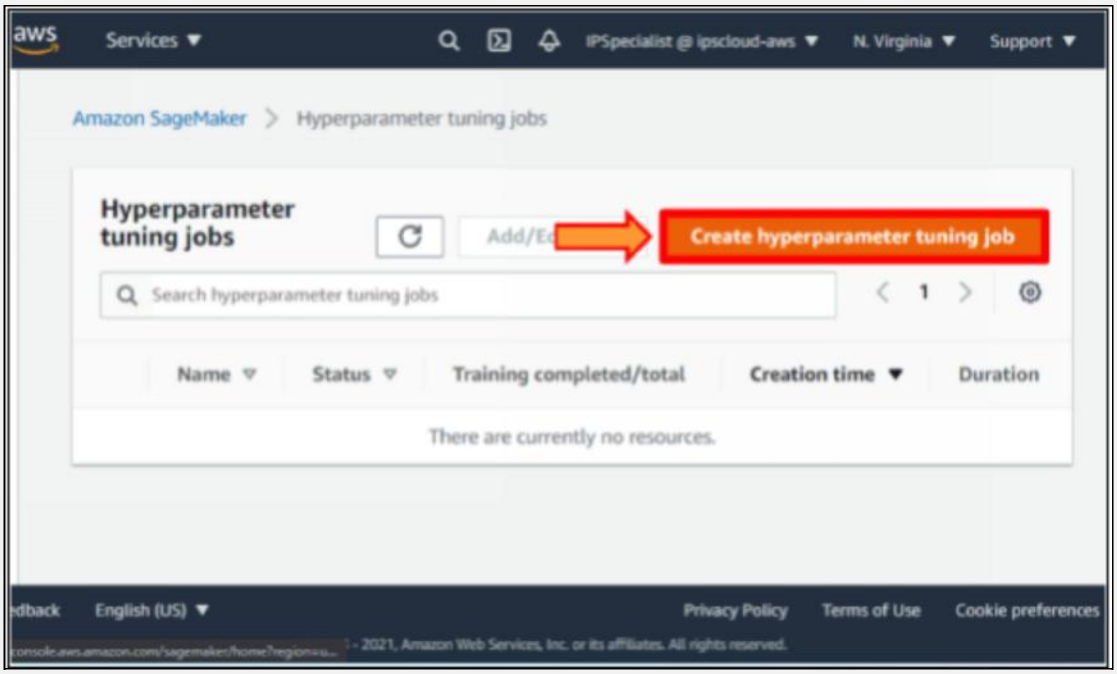
2. Select **SageMaker** from **Machine Learning**.



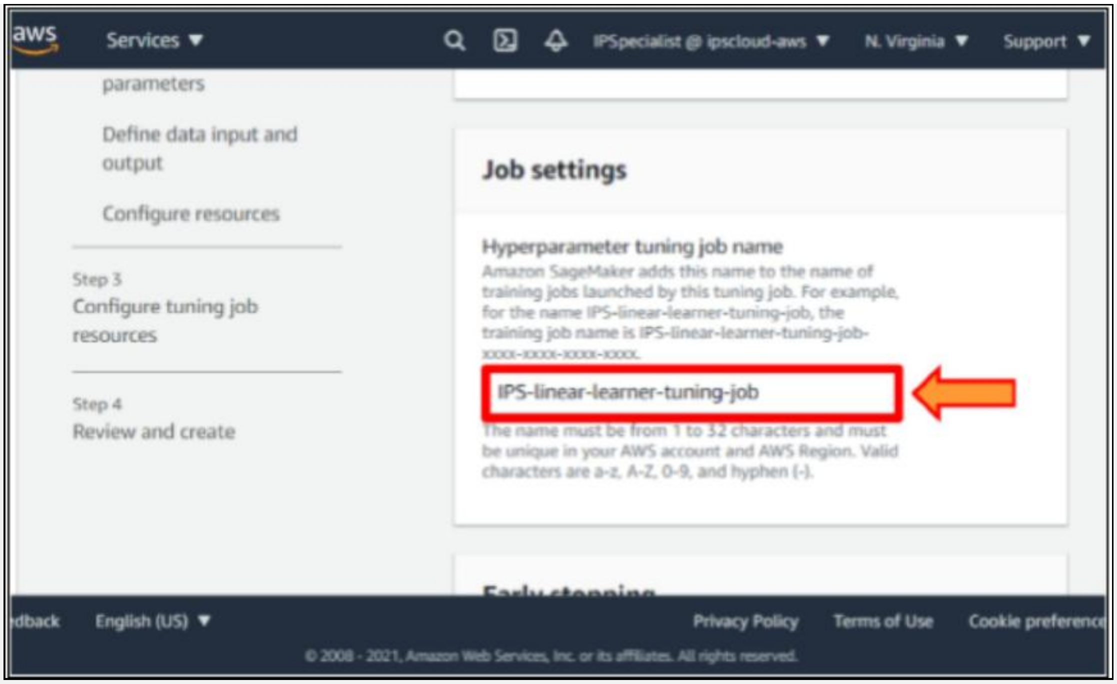
3. Click on the **Hyperparameter training jobs** from the left-hand side.



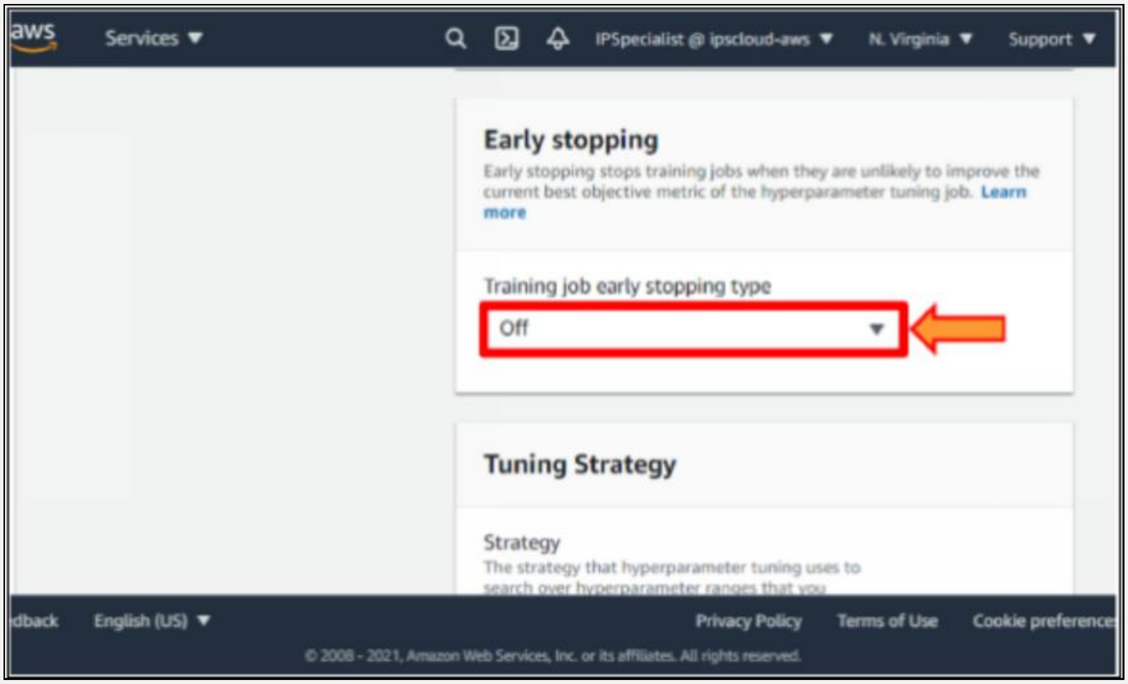
4. Click on the **Create hyperparameter tuning job** button.



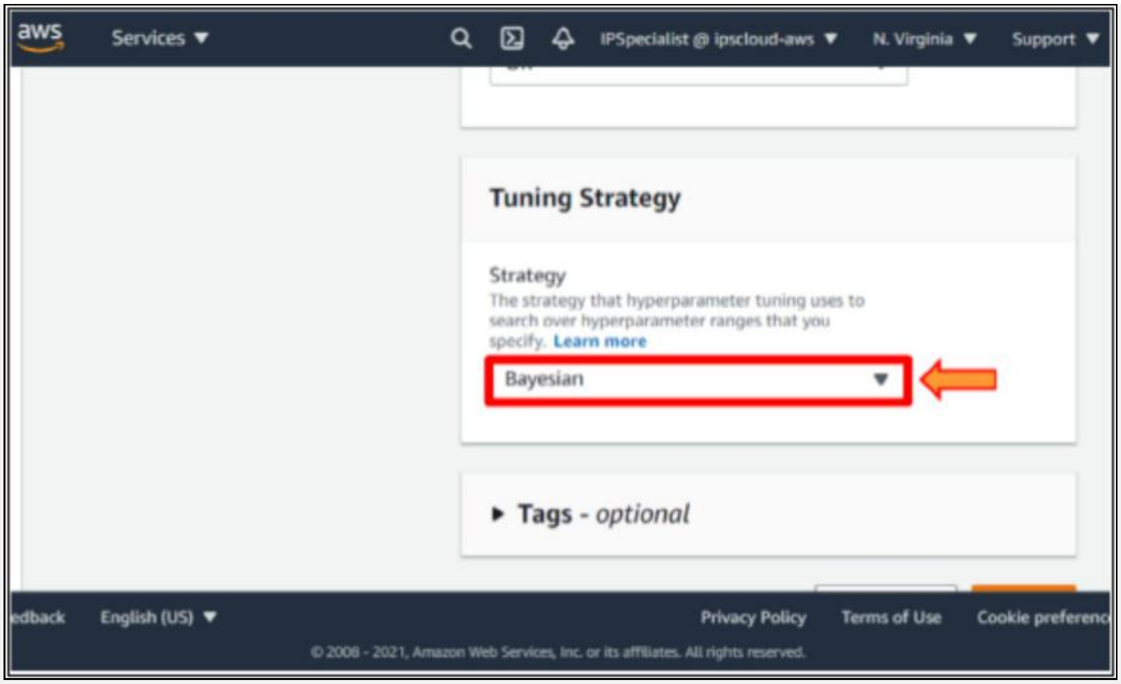
5. Give the job name **IPS-linear-learner-tuning-job.**



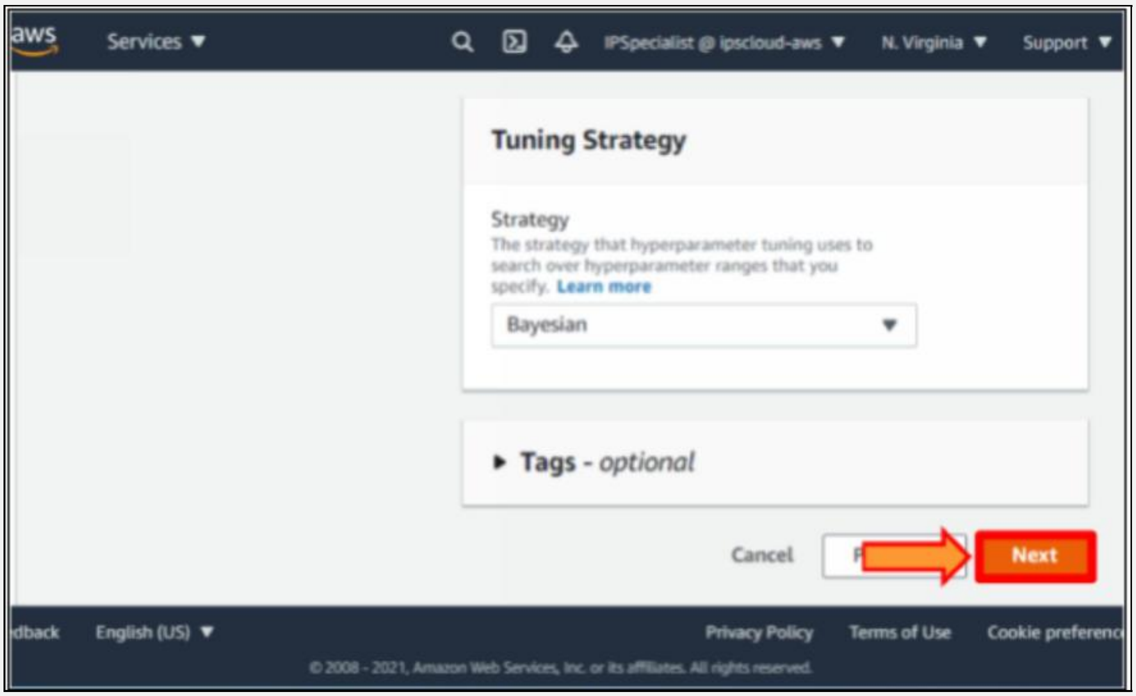
6. Select **off** for early stopping.



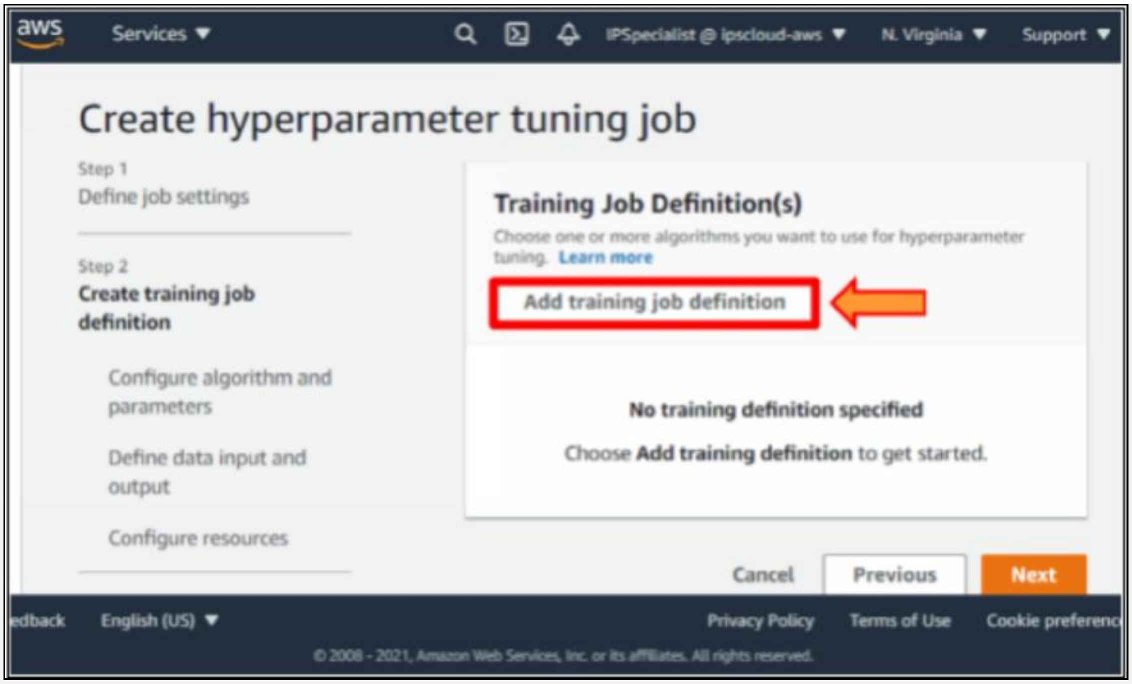
7. Select the **Bayesian** strategy.



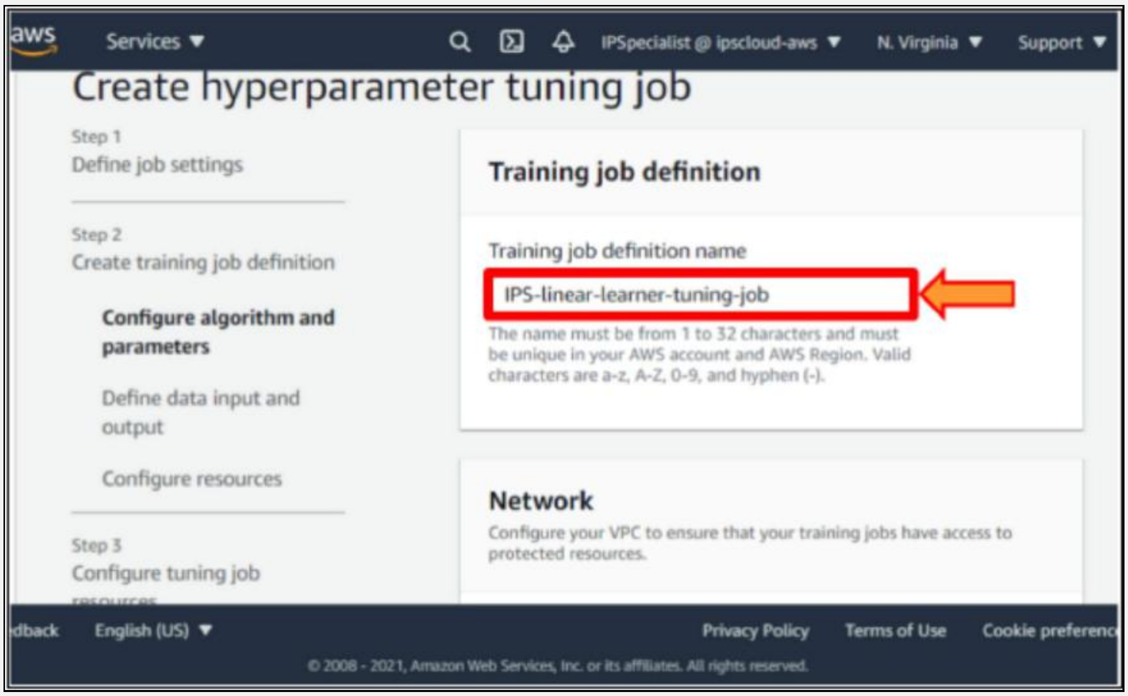
8. Click on the **Next** button.



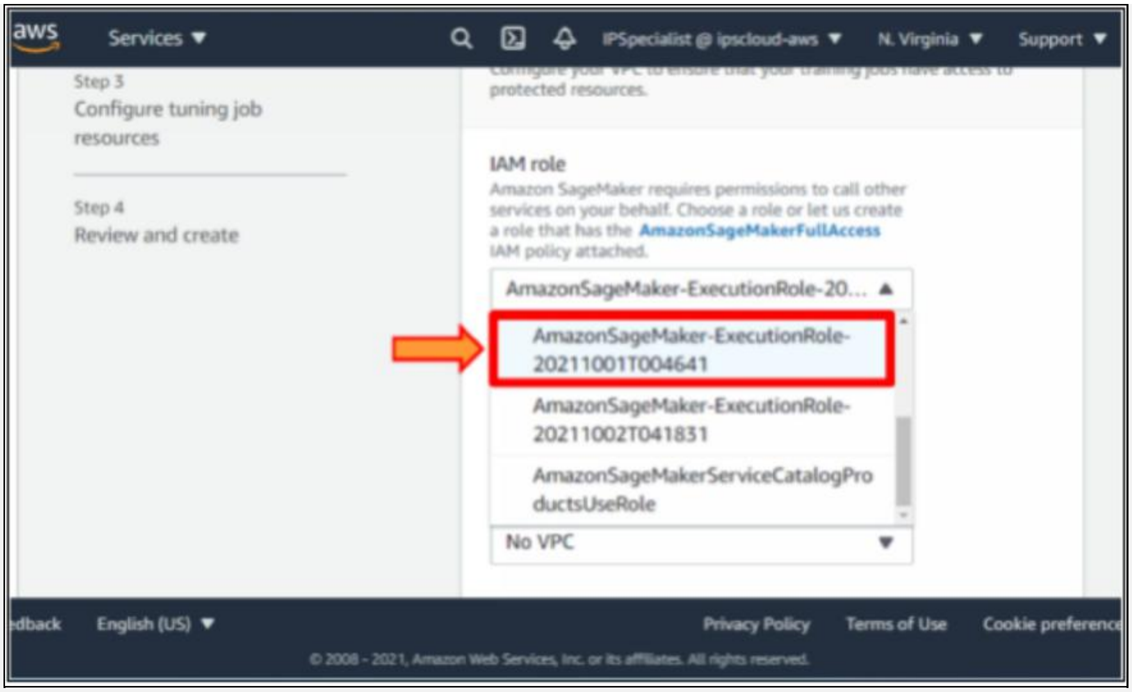
9. Click on the **Add training job definition** button.



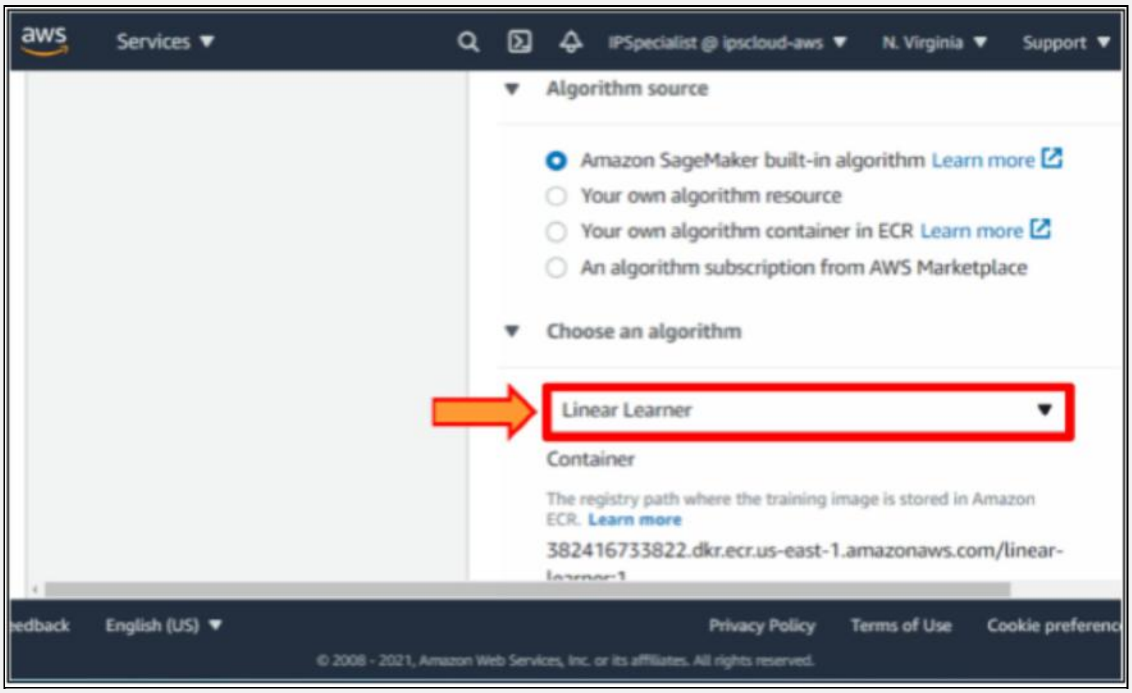
10. Give the training job definition name **IPS-linear-learner-tuning-job**.



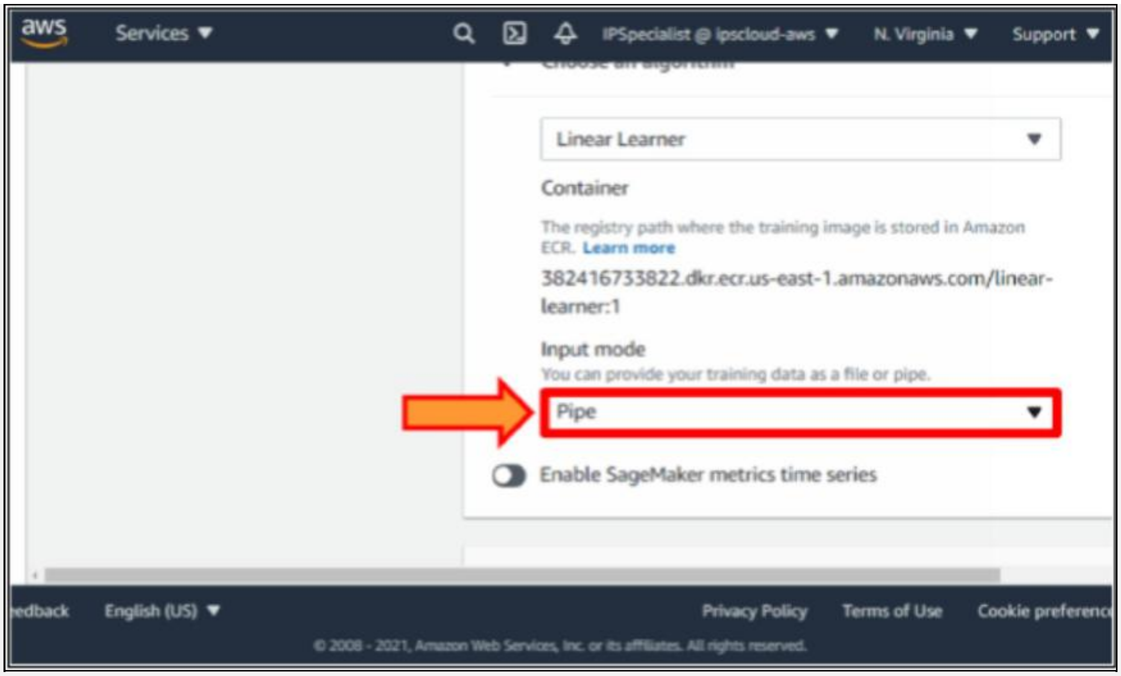
11. Select the **AmazonSageMaker-ExecutionRole** IAM role.



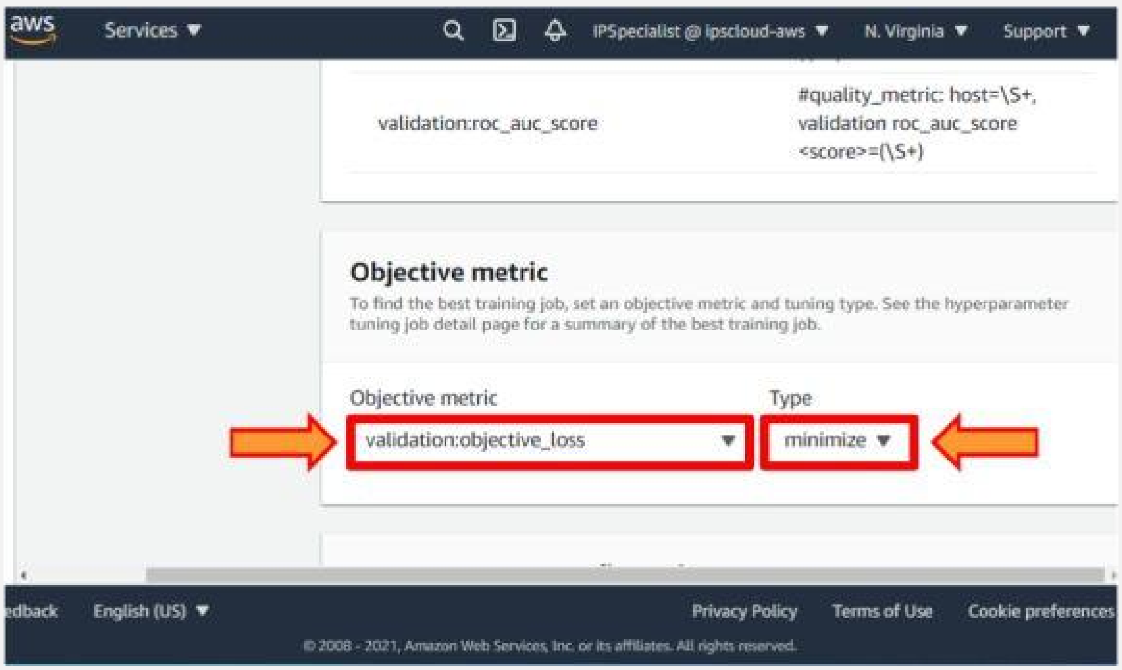
12. Scroll down. Select the **Linear Learner** algorithm.



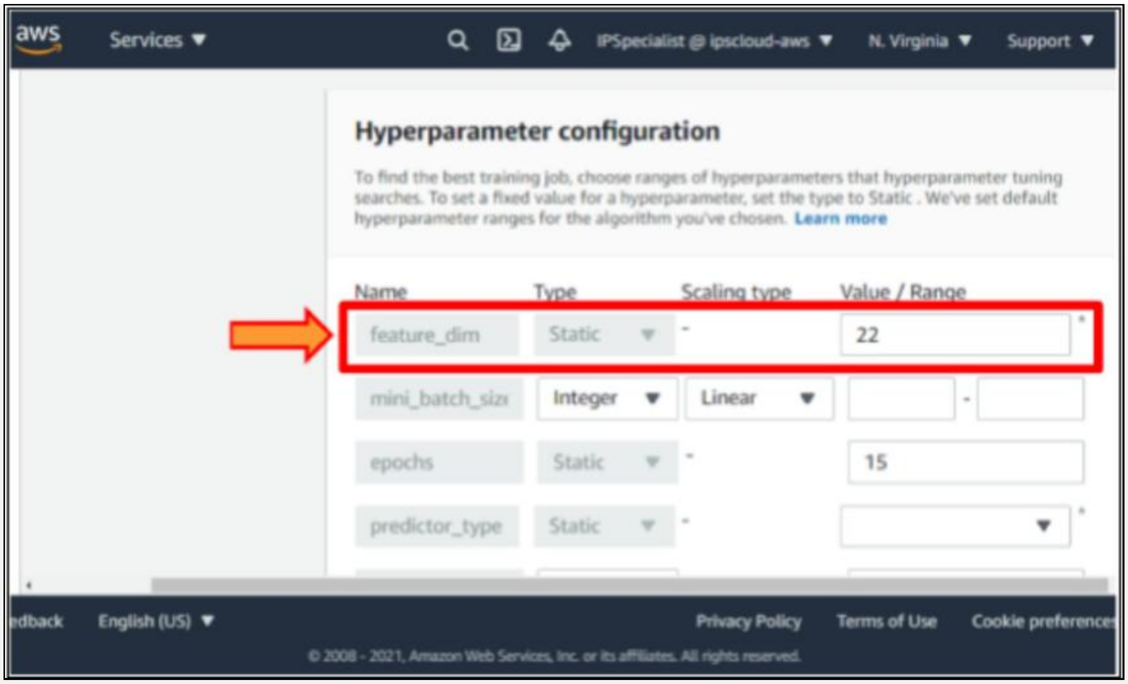
13. Select the **Pipe** input mode.



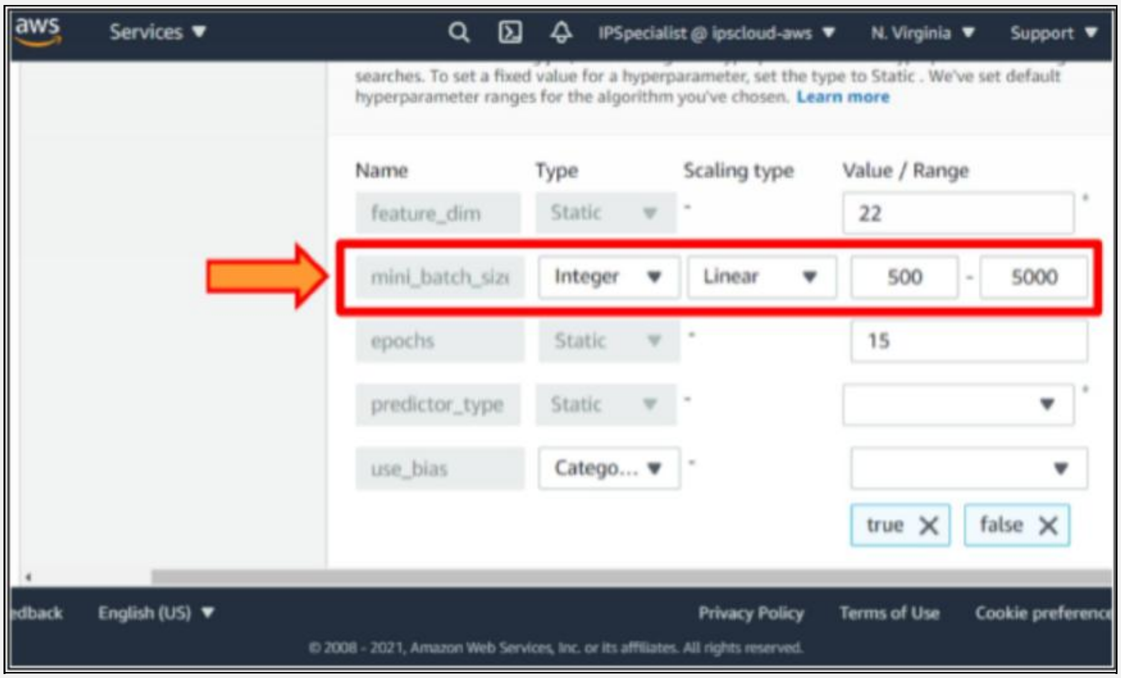
14. Select the **validation:objective_loss** objective. Then Select the **minimize** type.



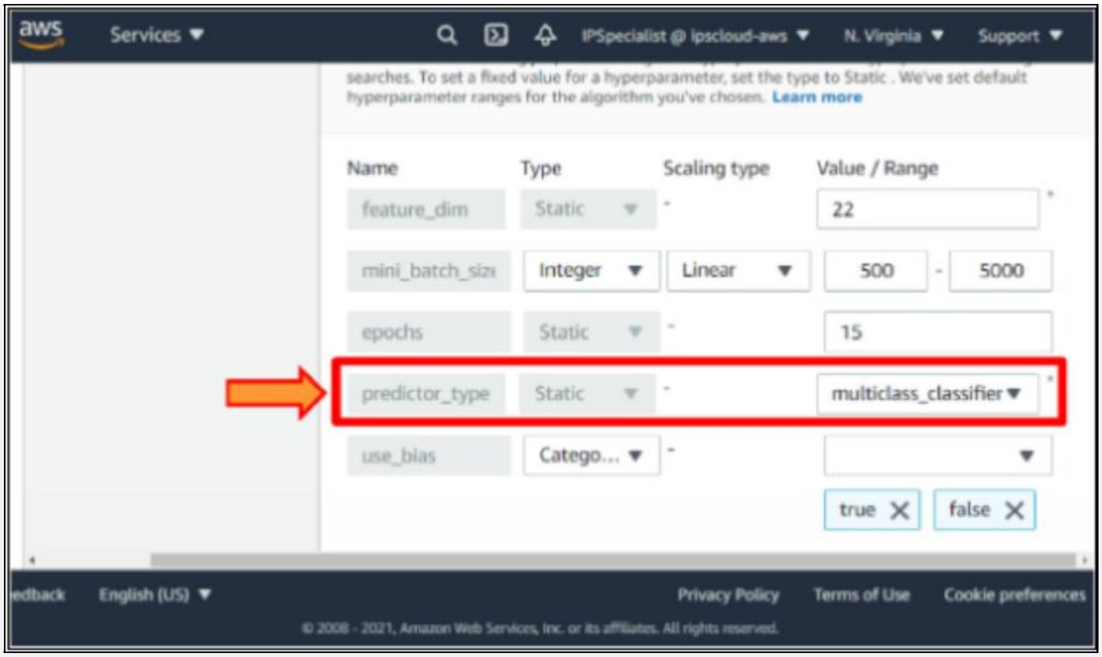
15. Type **22** in the **feature_dim.**



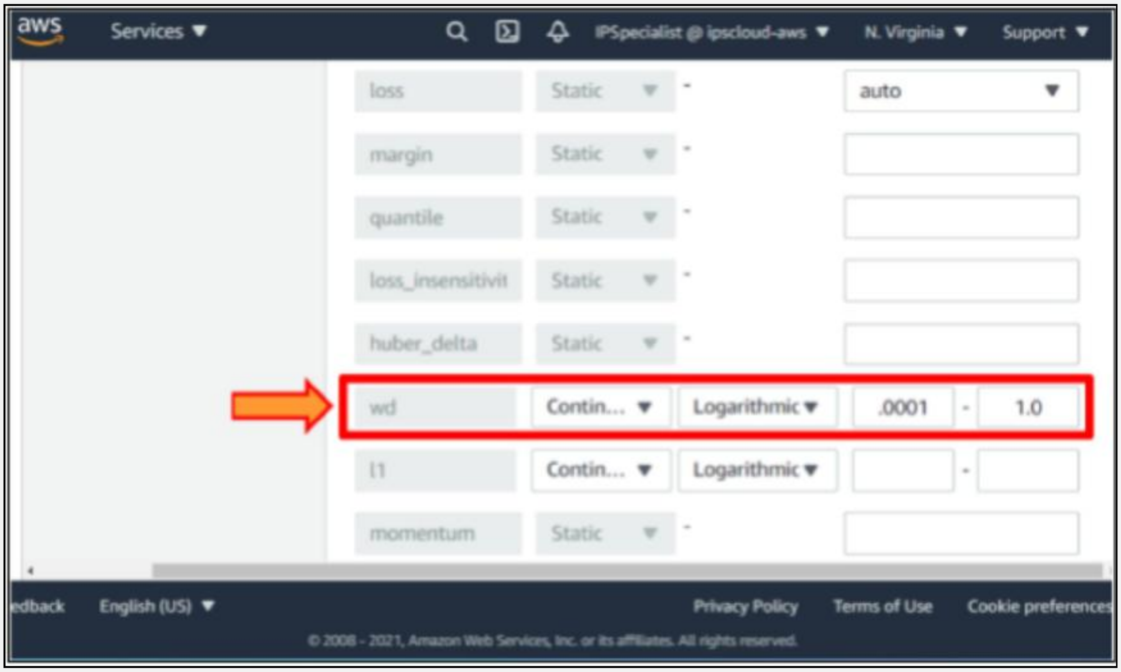
16. Type **500 – 5000** in the **mini_batch_size**



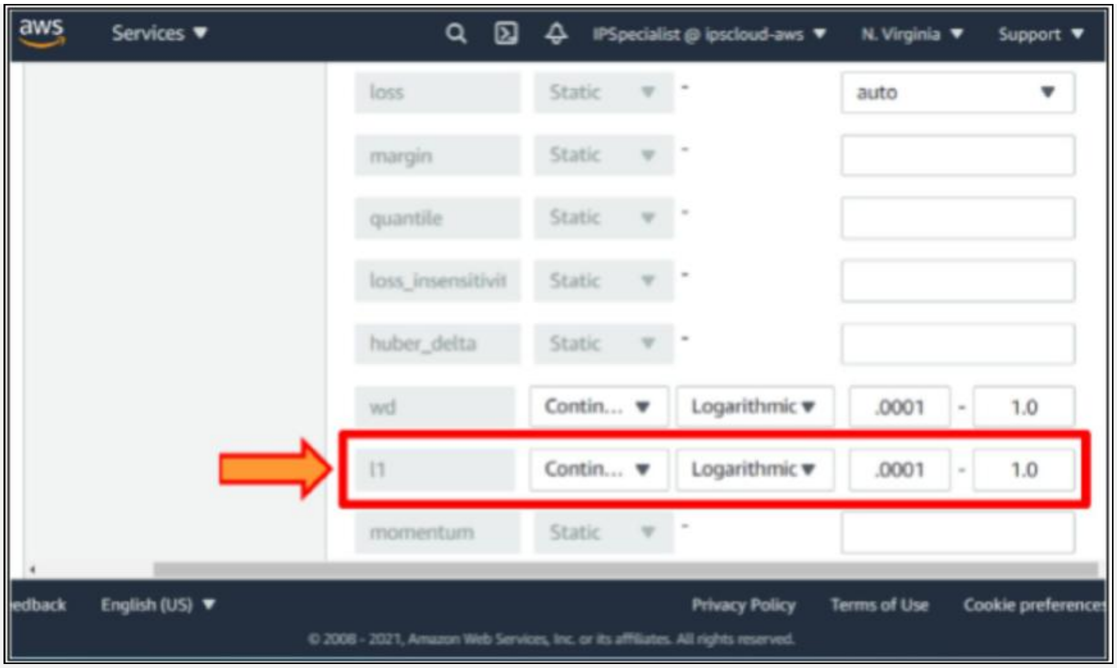
17. Select the **multiclass_classifier** in the **predictor_type.**



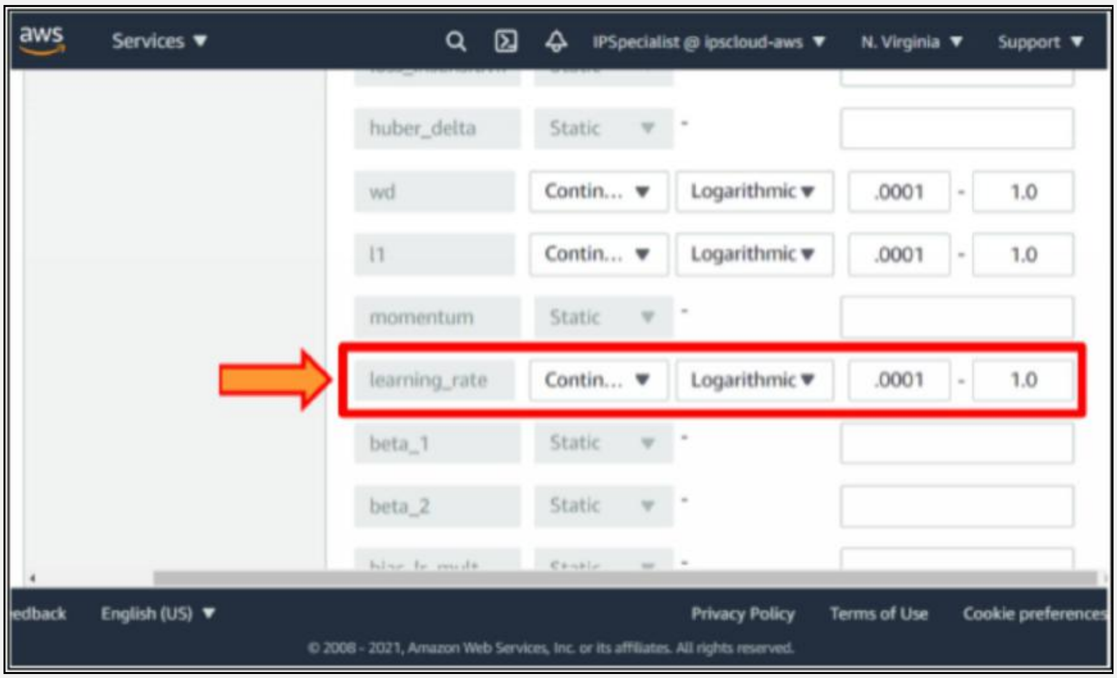
18. Scroll down. Type **.0001 – 1.0** in the **wd**.



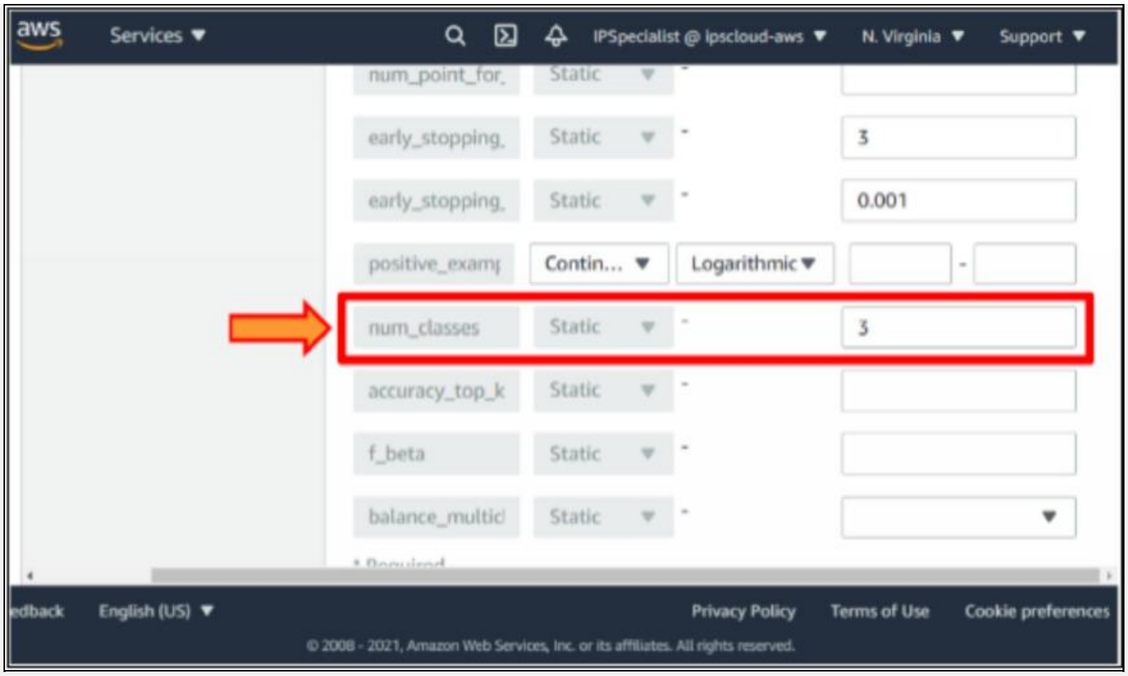
19. Type the **.0001 – 1.0** in the **L1.**



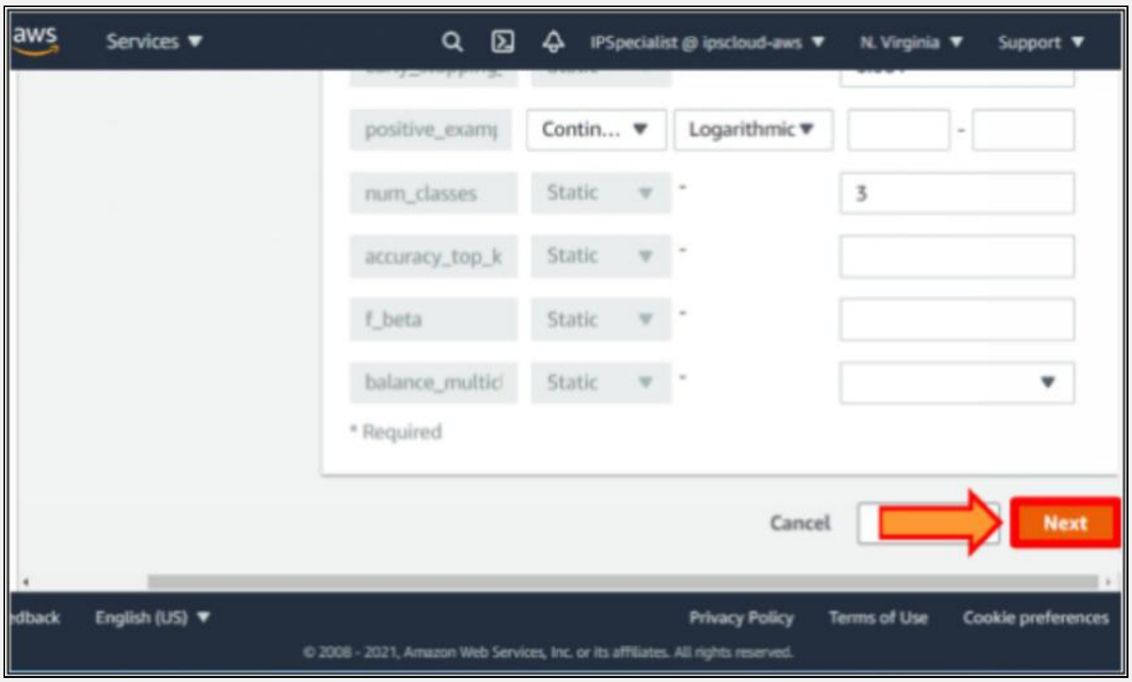
20. Type the **.0001 – 1.0** in the **learning_rate.**



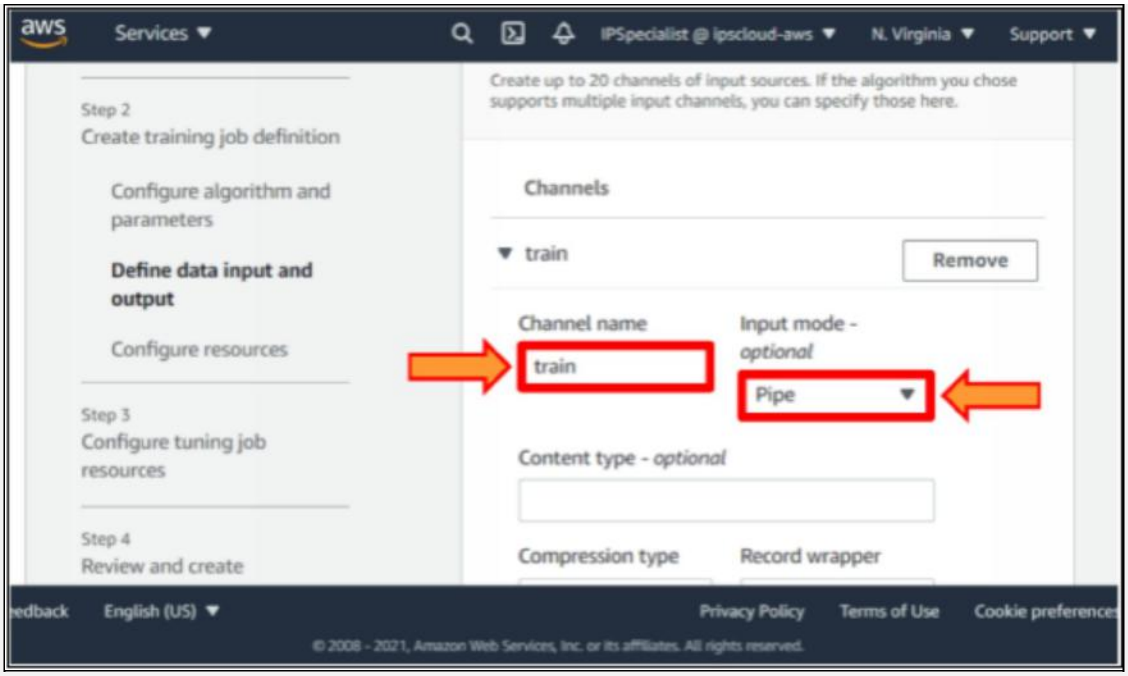
21. Scroll down. Type the **3** in the **num_classes.**



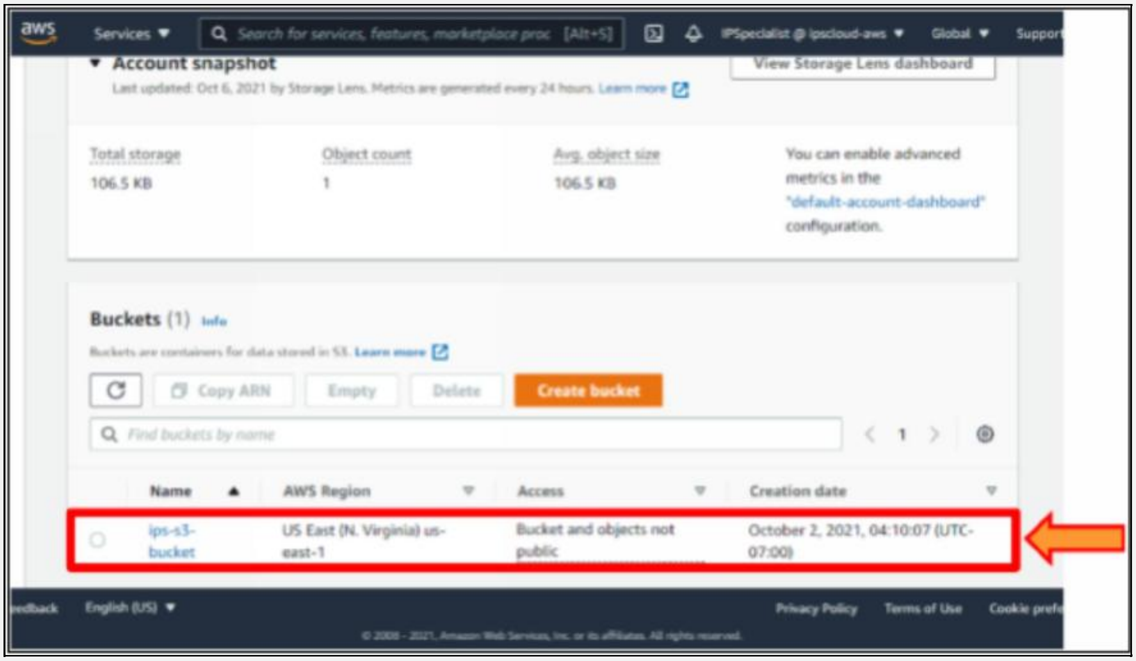
22. Click on the **Next** button.



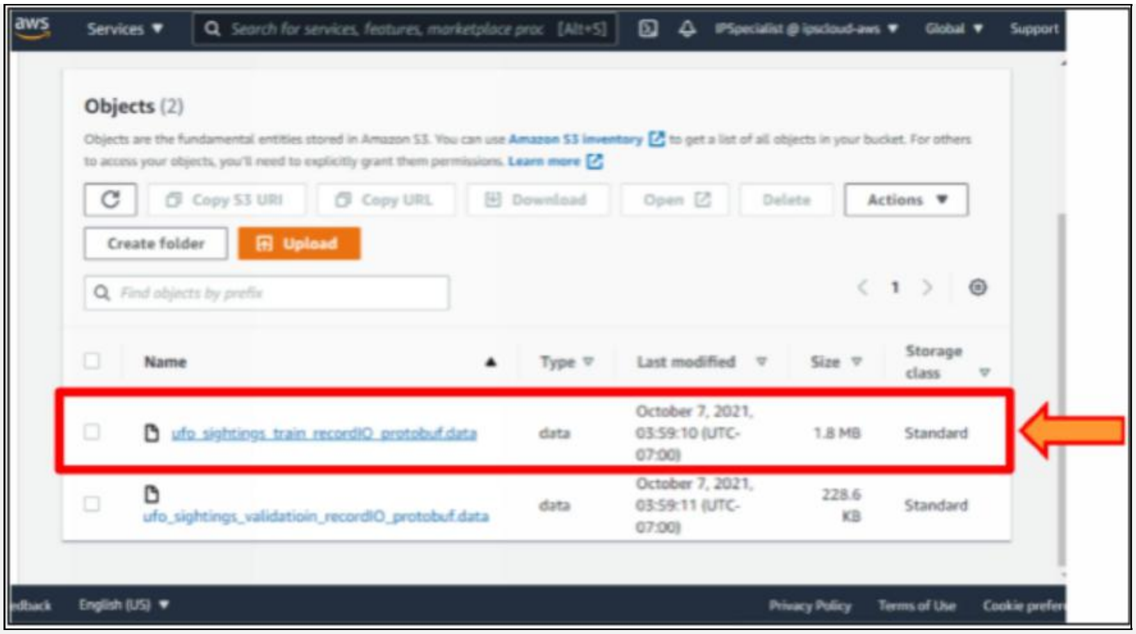
23. Select the **Pipe** input mode for the train channel.



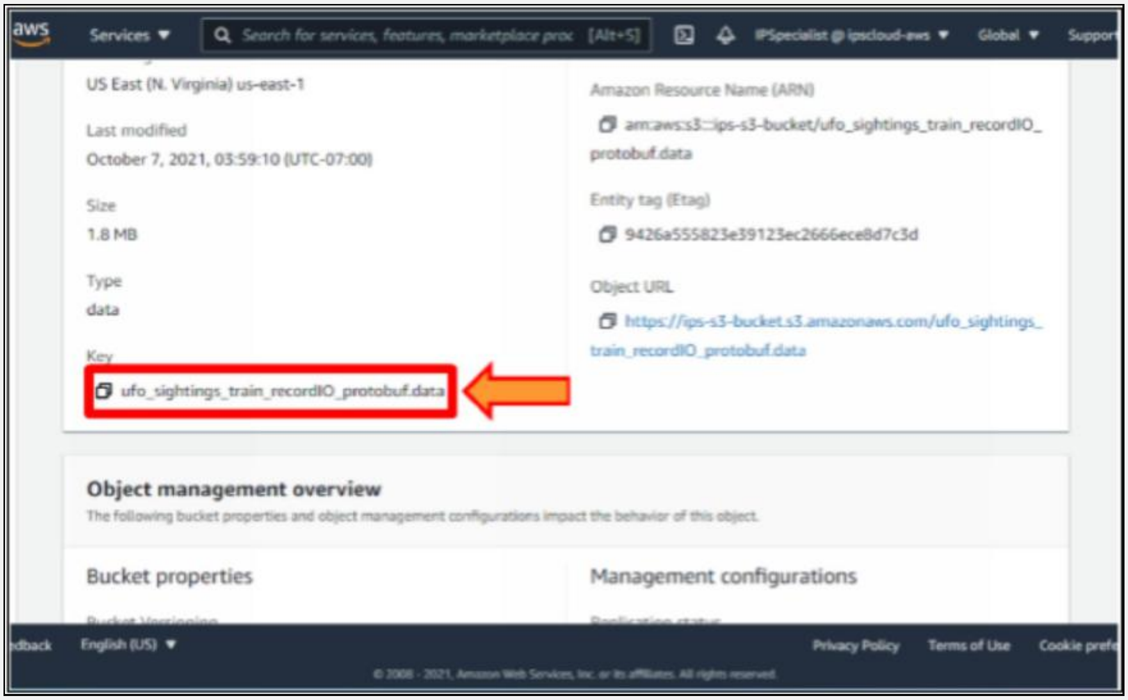
24. Go to the **S3 dashboard**.
25. Click on **ips-s3-bucket.**



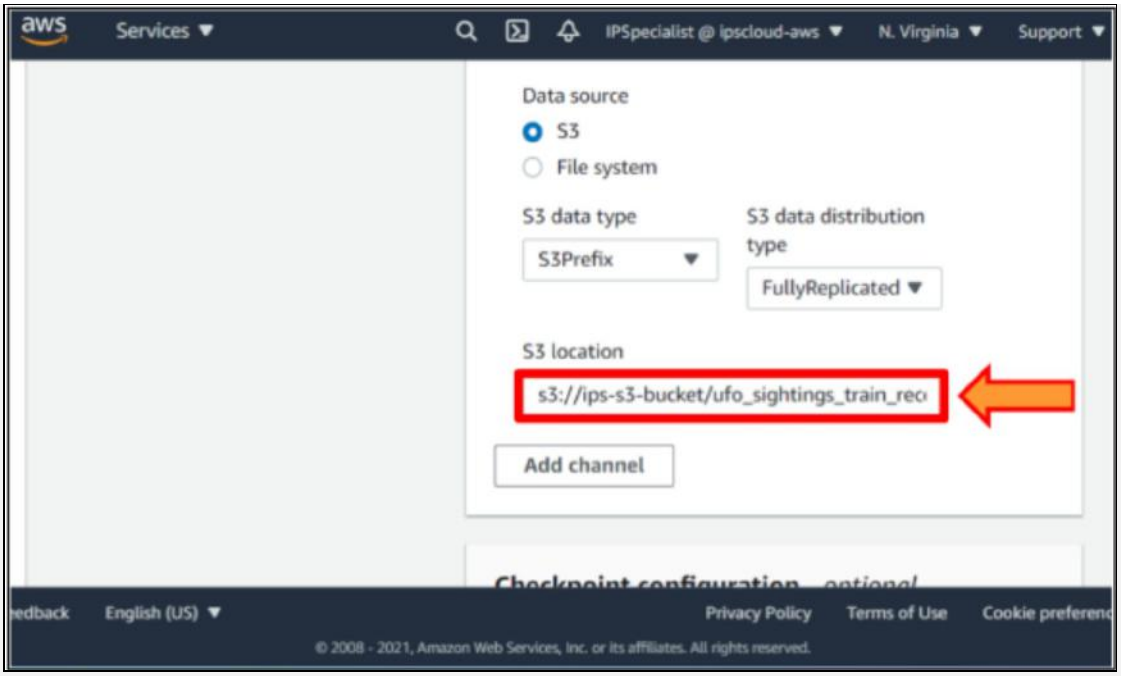
26. Click on the **ufo_sightings_train_recordIO_protobuf.data** file.



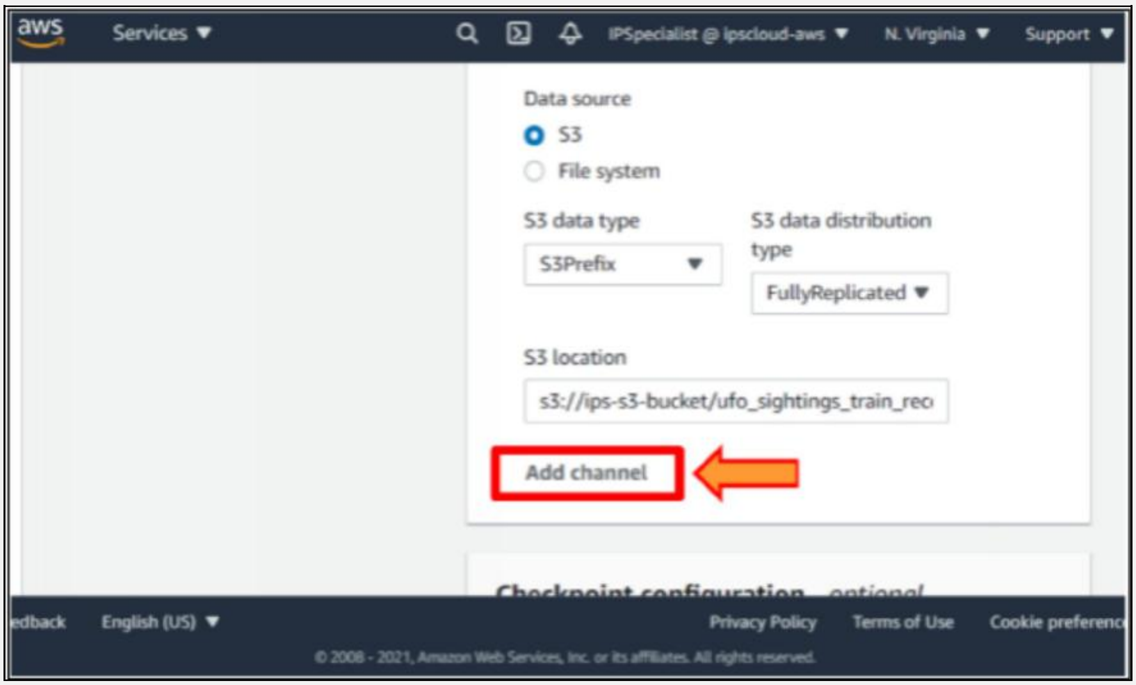
27. Copy the **key name.**



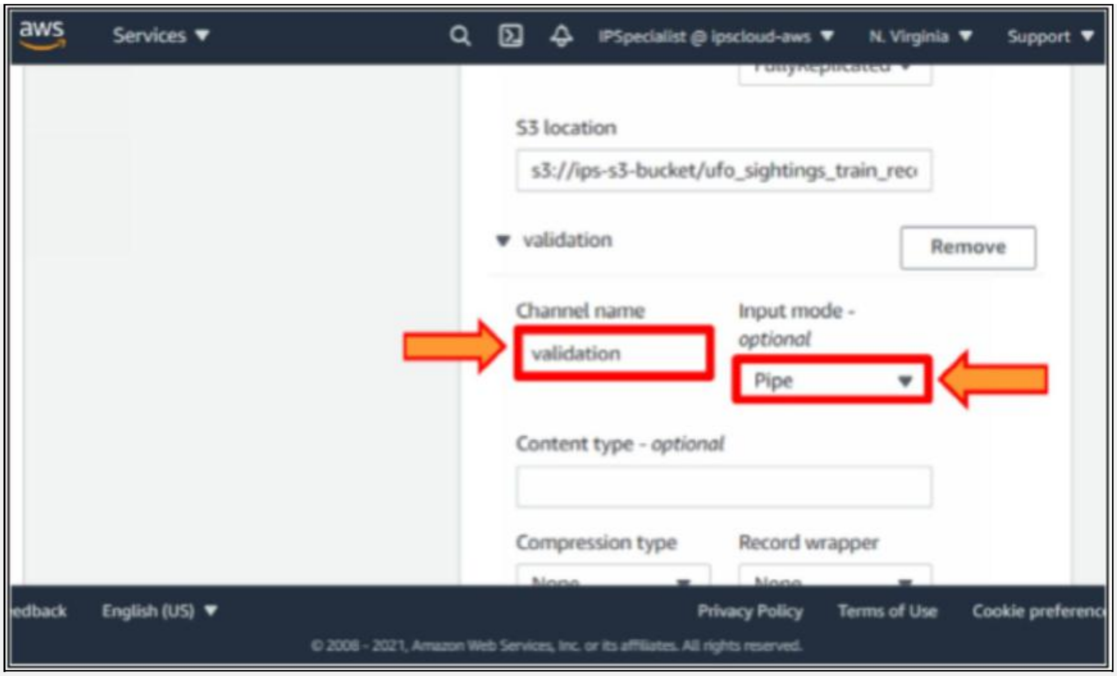
28. Go back to the SageMaker dashboard.
29. Give the S3 location path of the training file in the following manner **s3://ips-s3—bucket/<Train_Key_Name>.**



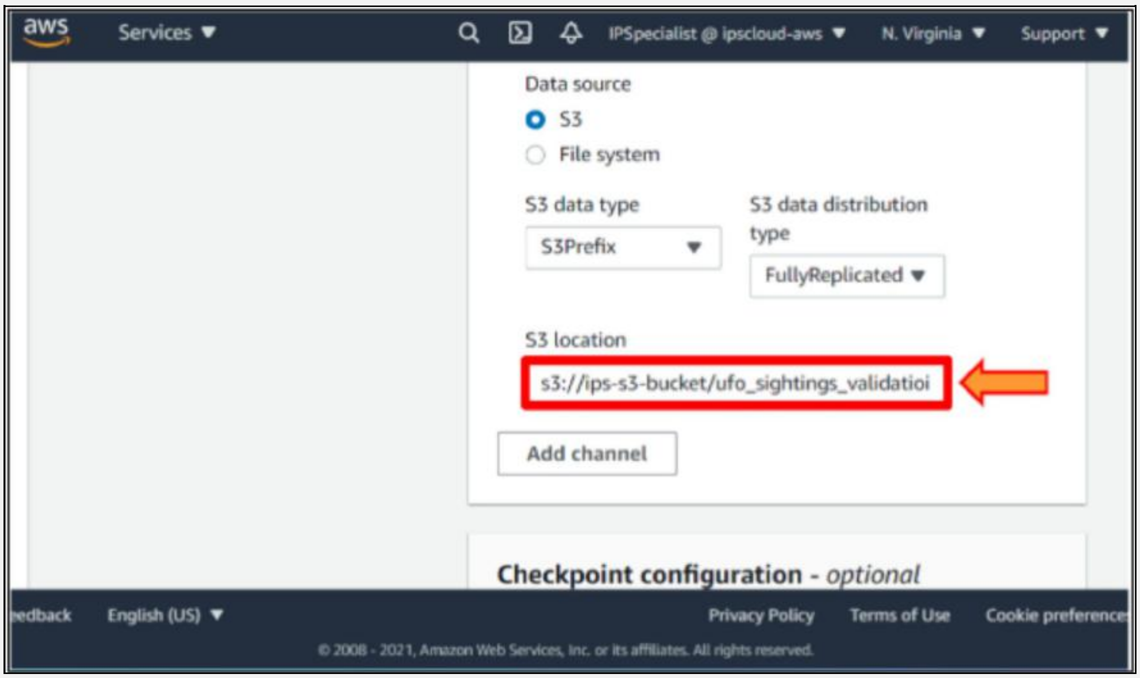
30. Click on the **Add channel** button.



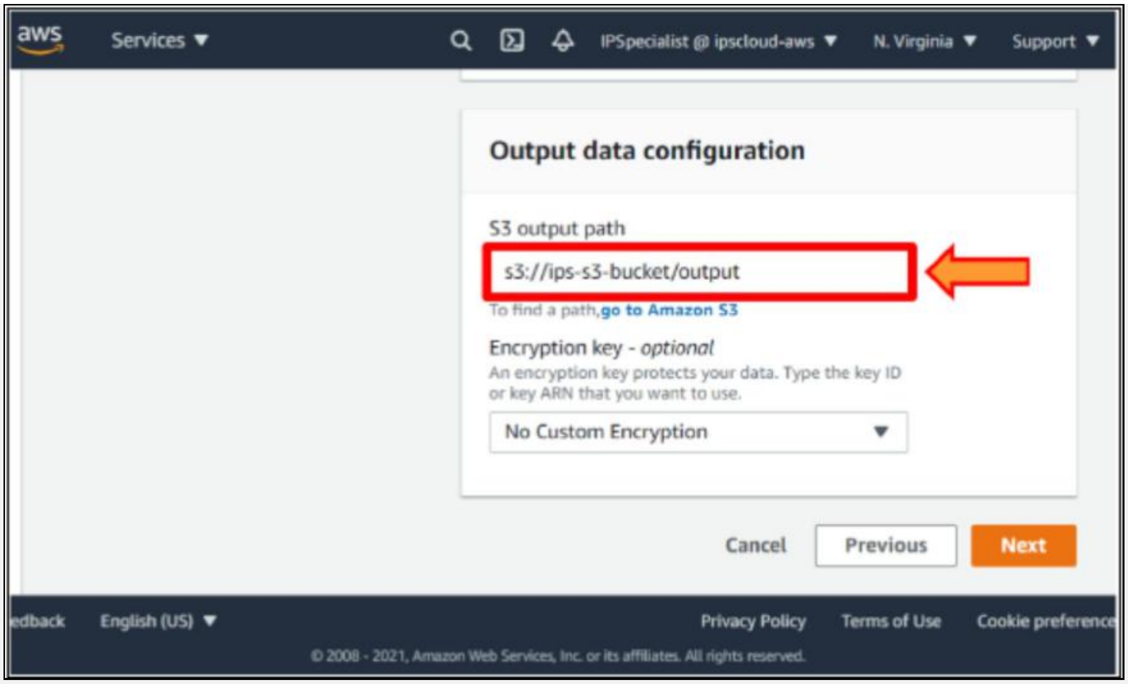
31. Give a channel name: **Validation**.
32. Select the **Pipe** input mode for the validation channel.



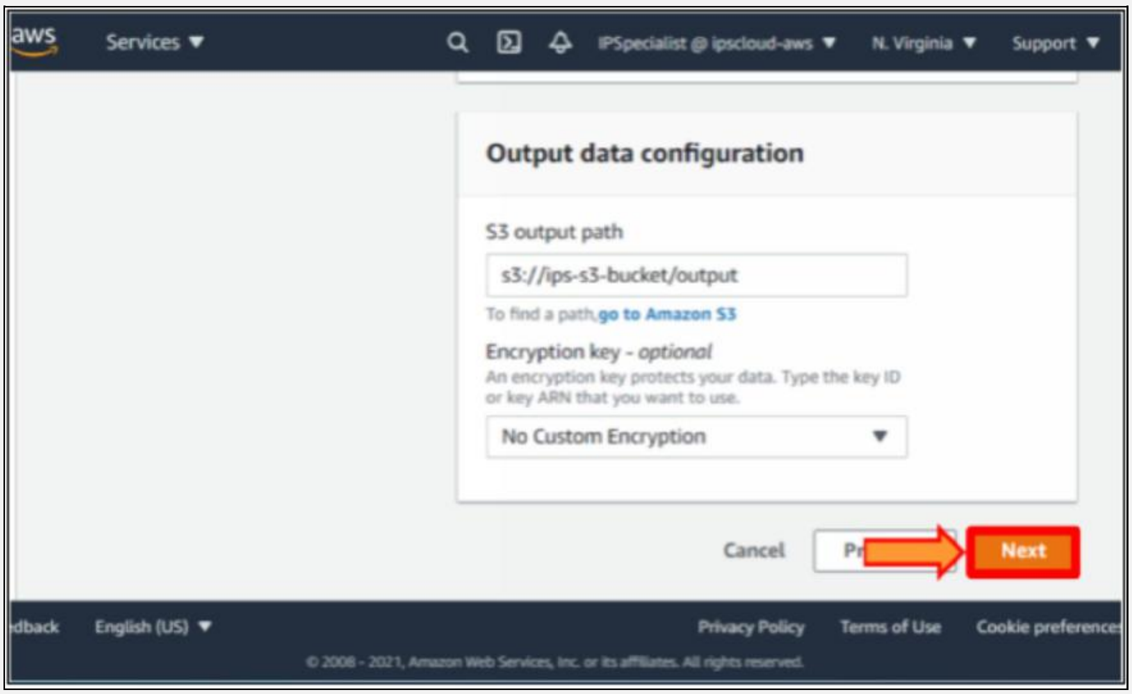
33. Give the S3 location path of the validation file in the following manner **s3://ips-s3-bucket/<Validation_Key_Name>.**



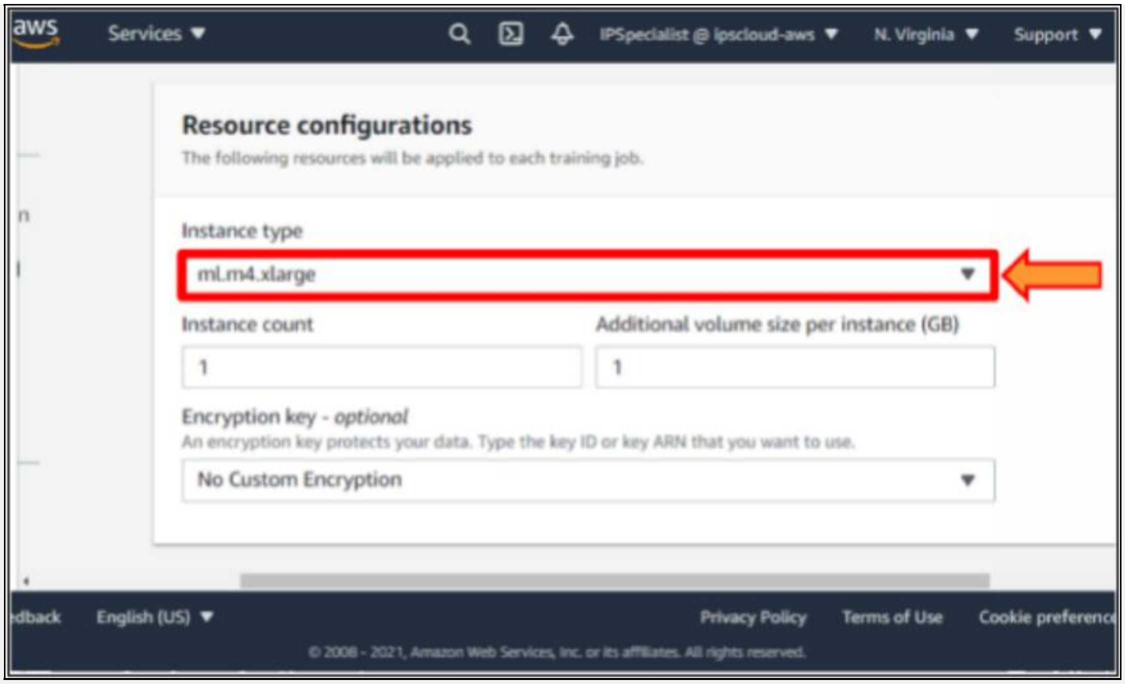
34. Give the **S3 location path** of output data. **s3://ips-s3-bucket-output.**



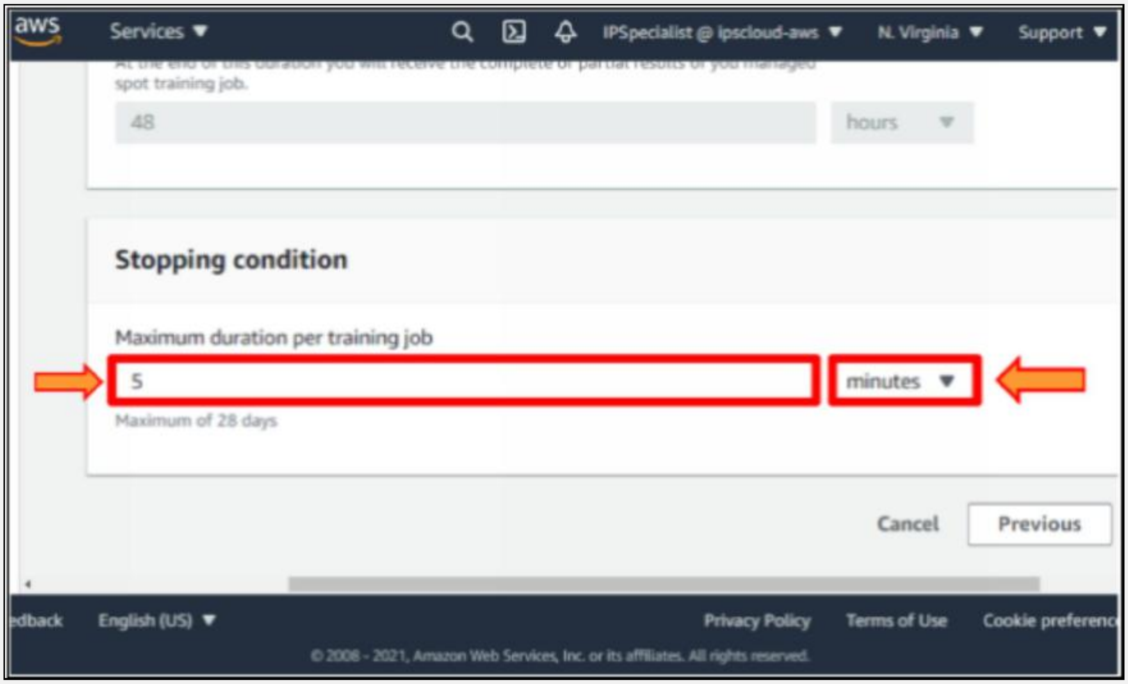
35. Click on the **Next** button.



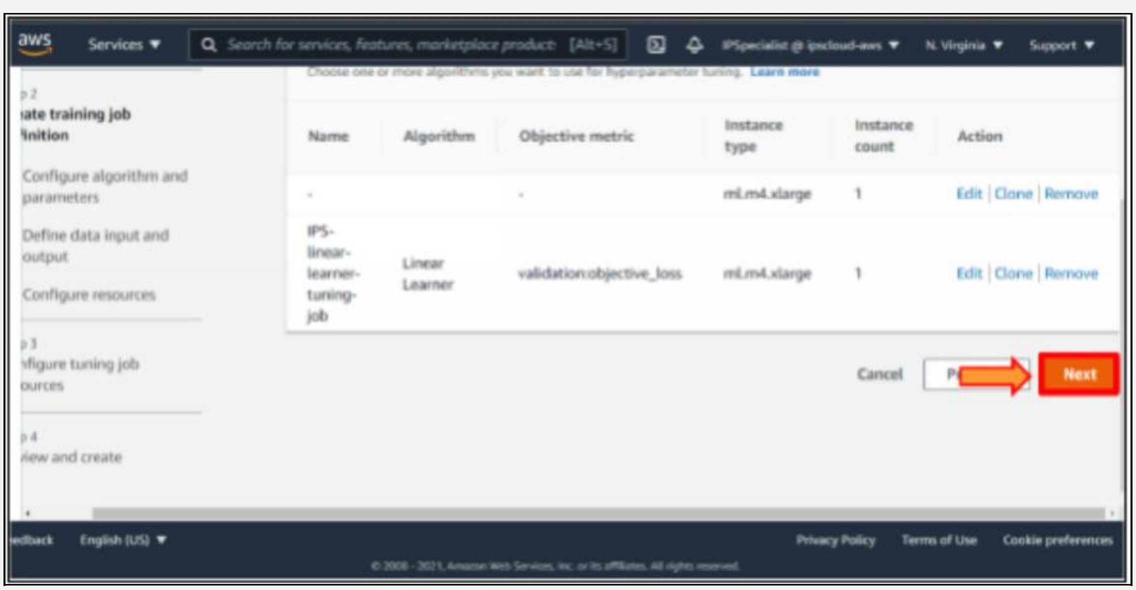
36. Select the **ml.m4.xlarge** instance type.



37. Set **5 minutes** in **Maximum duration per training job.**

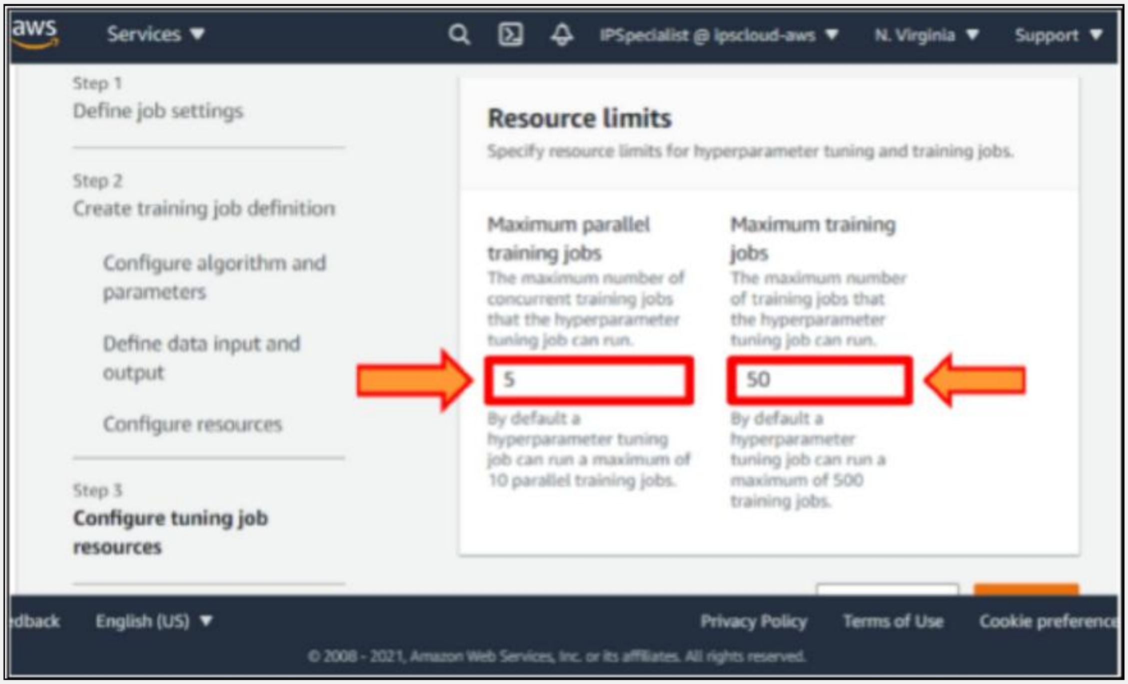


38. Click on the **Next** button.

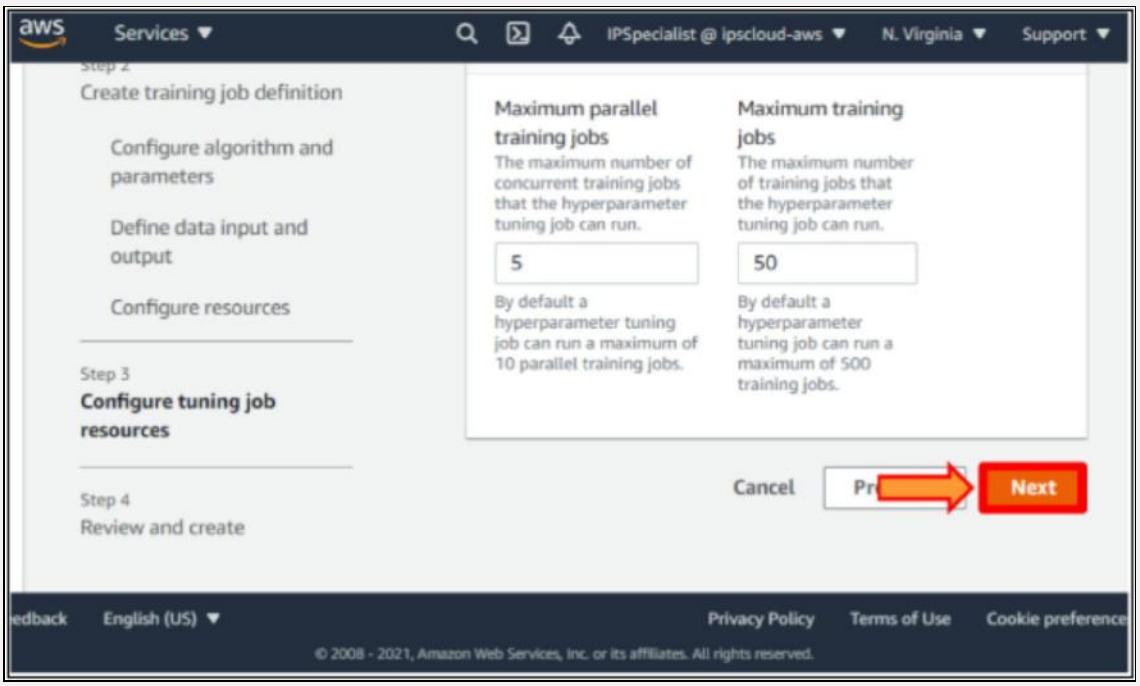


39. Type **5** in the **Maximum Parallel Training Jobs** box.

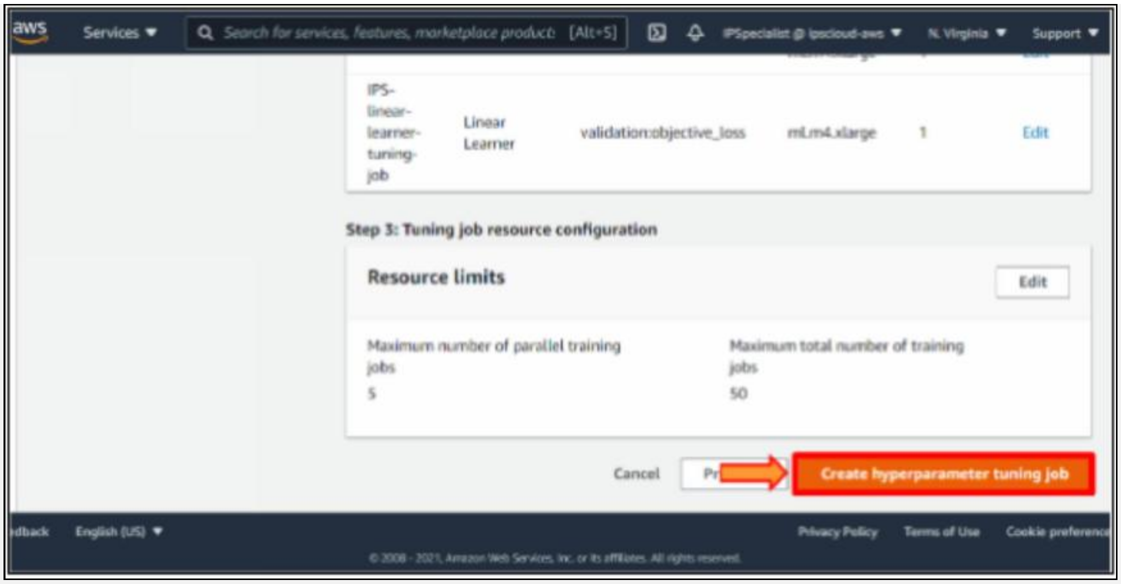
40. Type **50** in the **Maximum Training Jobs** box.



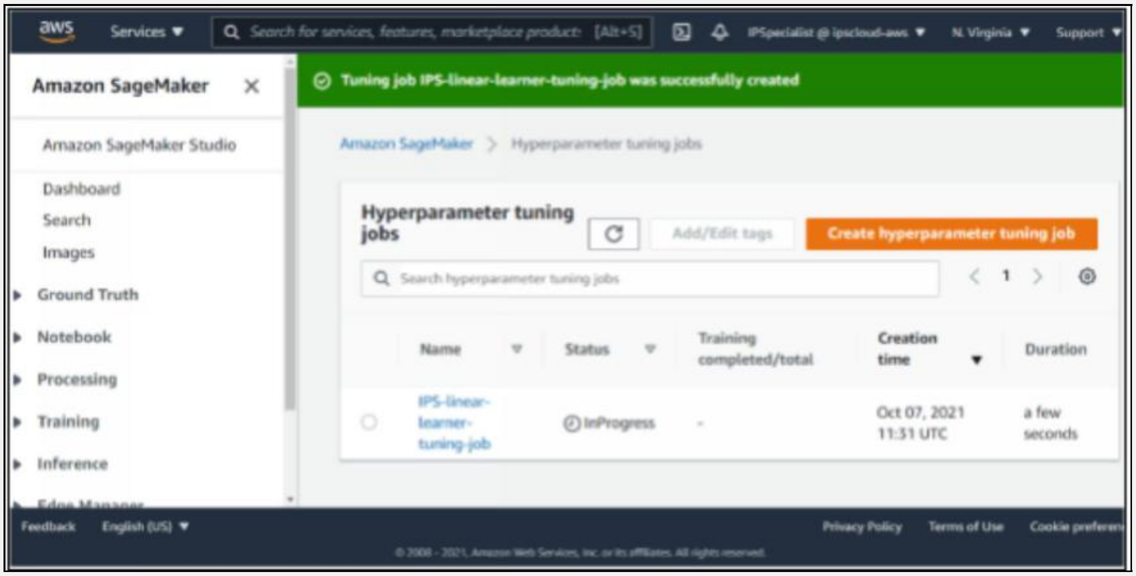
41. Click on the **Next** button.



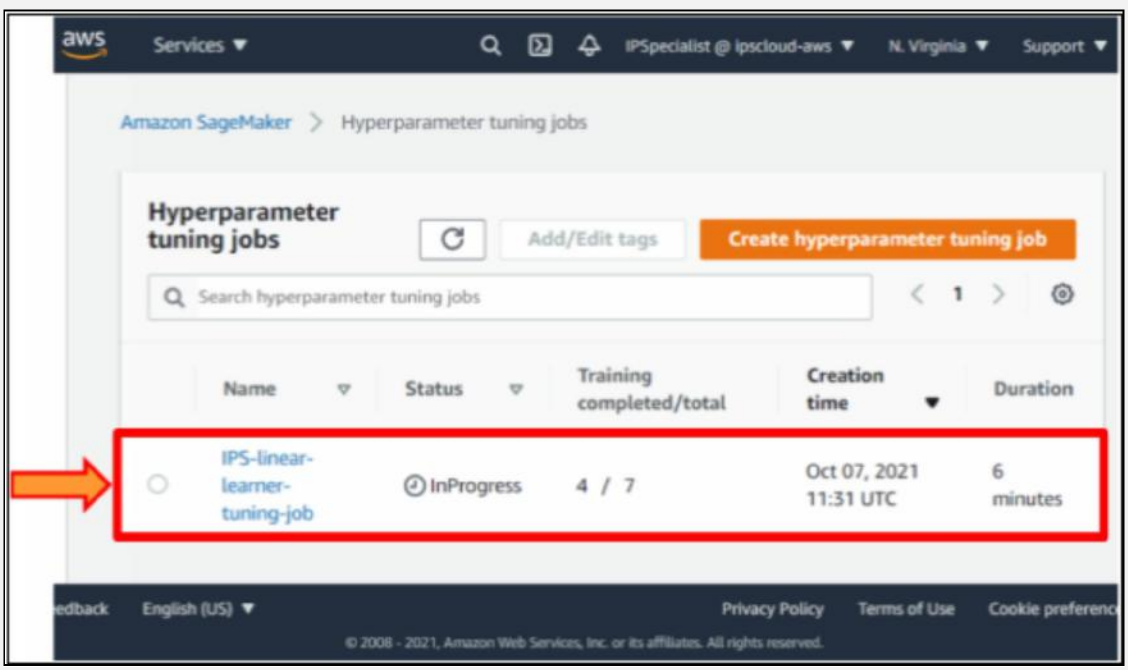
42. Click on the **Create hyperparameter tuning job** button.



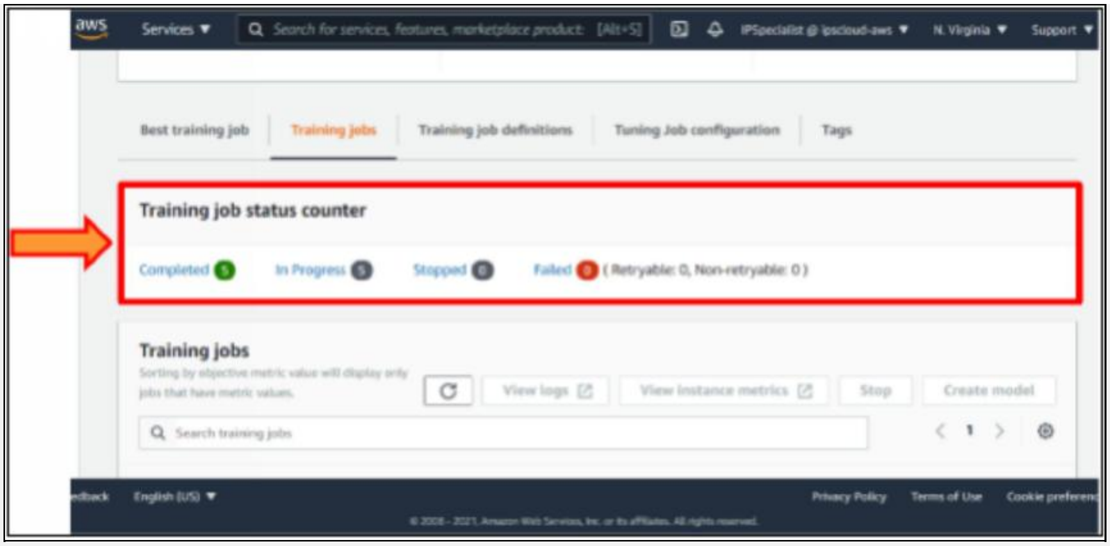
A successfully created job will take time up to 30 minutes to complete.



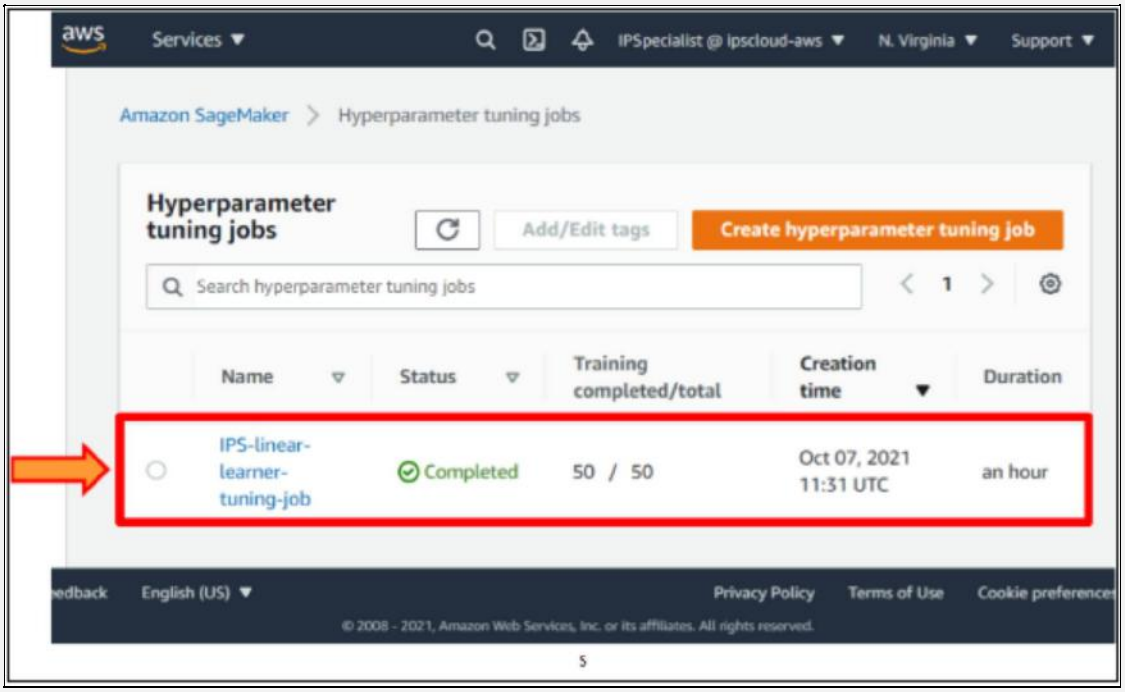
43. Click on the **IPS-linear-learner-tuning-job**



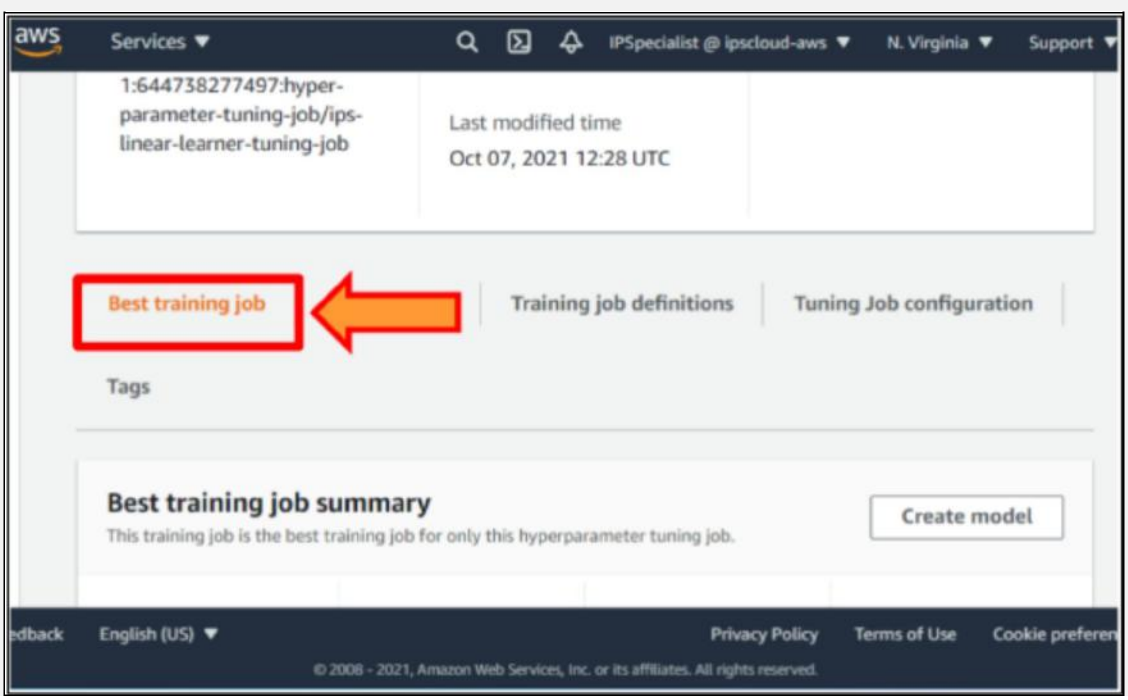
44. Check the **Training job status counter.**



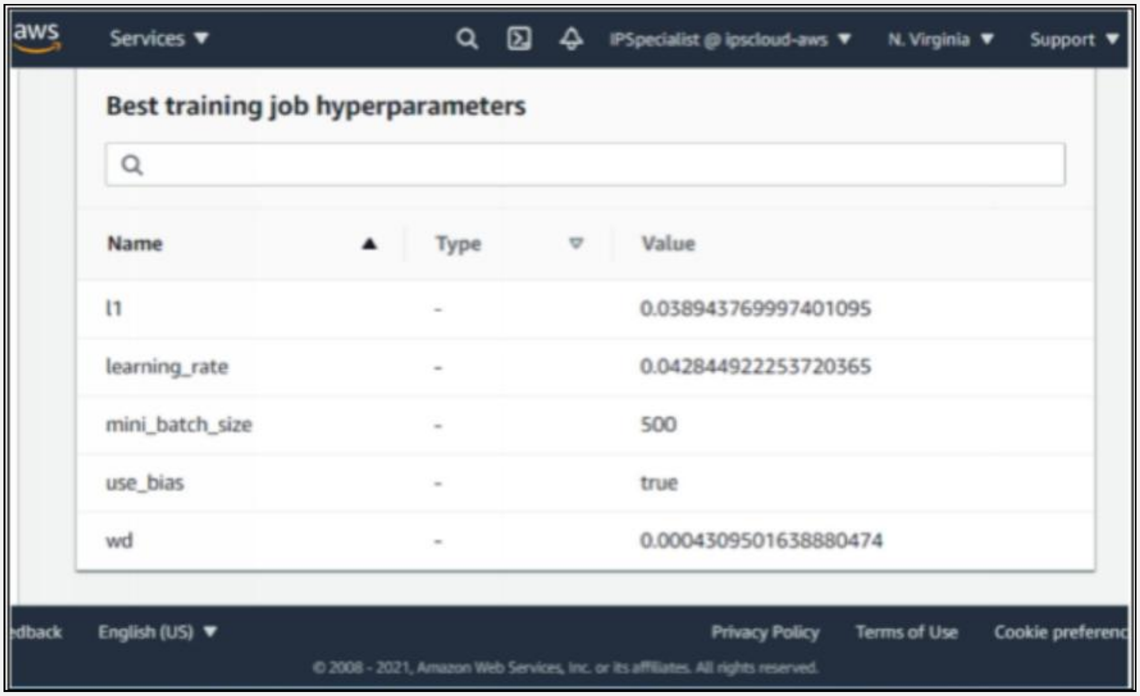
45. Hence, the training is completed.
46. Click on **IPS-linear-learner-tuning-job.**



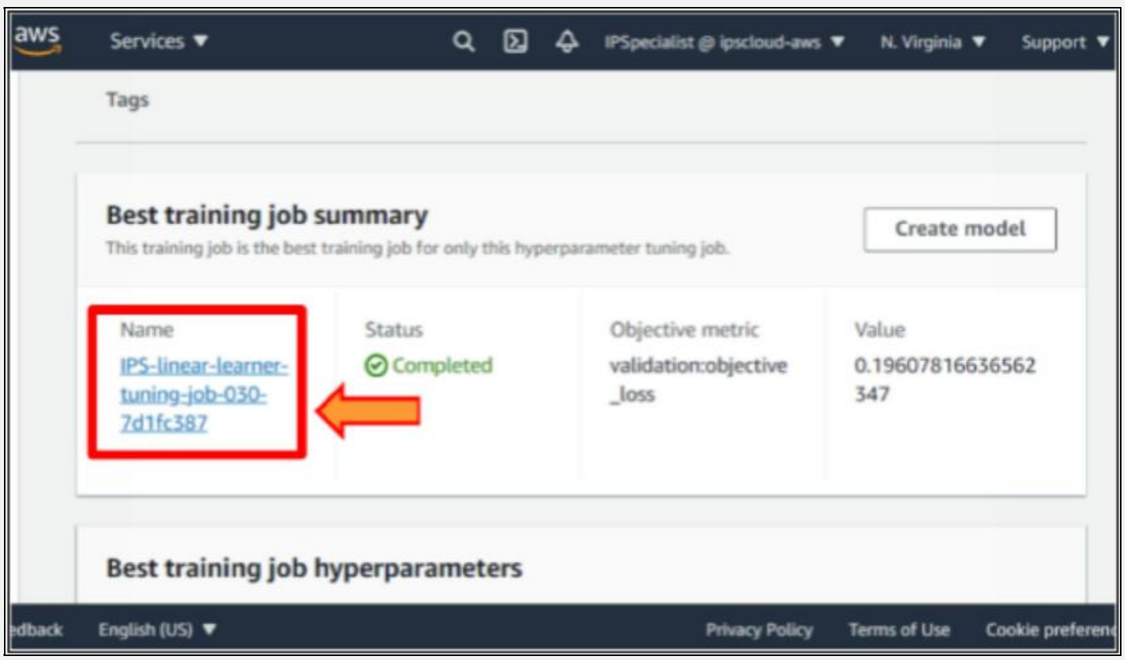
47. Click on the **Best training job** tab.



48. Scroll down you will see the best training job hyperparameters.

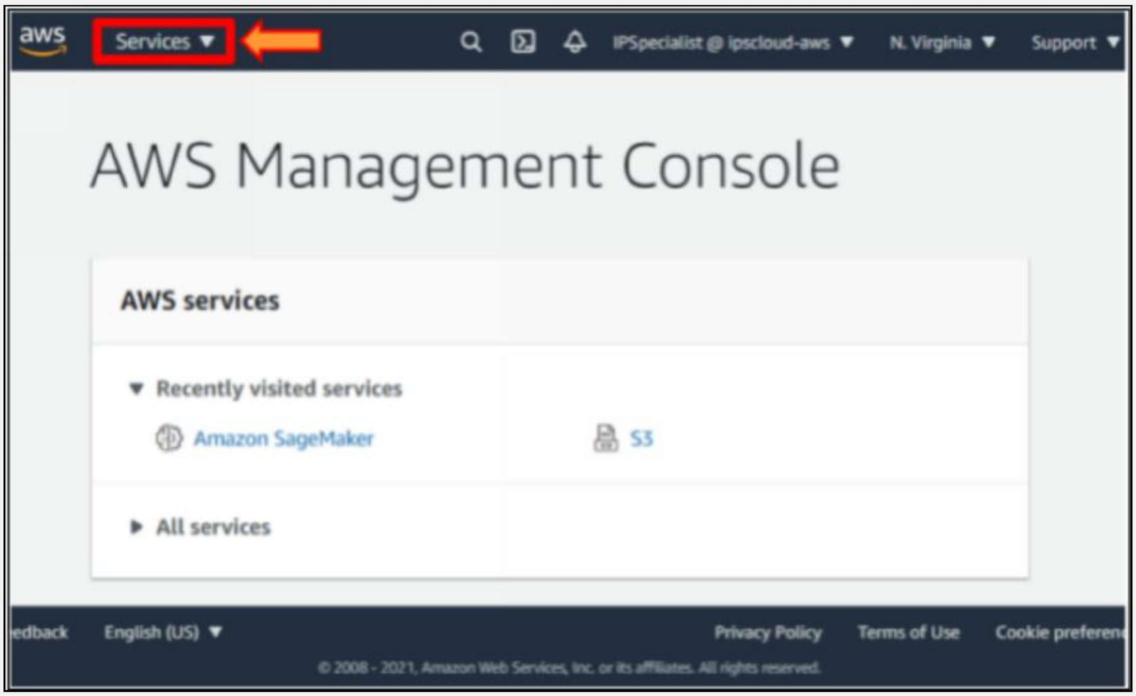


49. Copy the name **IPS-linear-learner-tuning-job.**

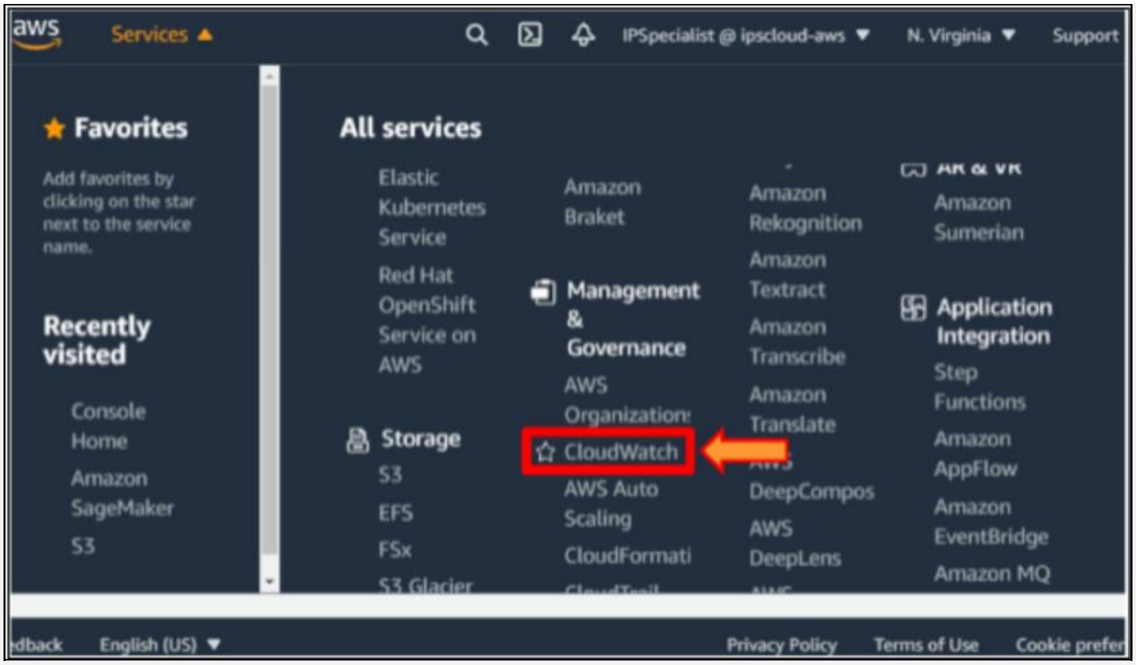


Step 3: Analyze CloudWatch Training Logs

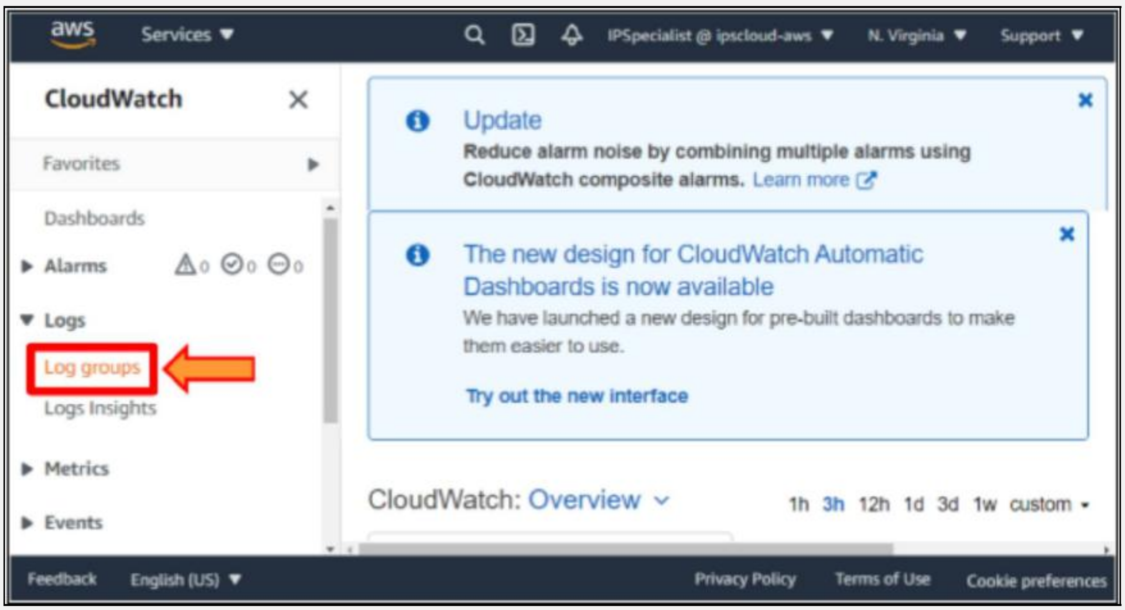
1. Click on the **Services.**



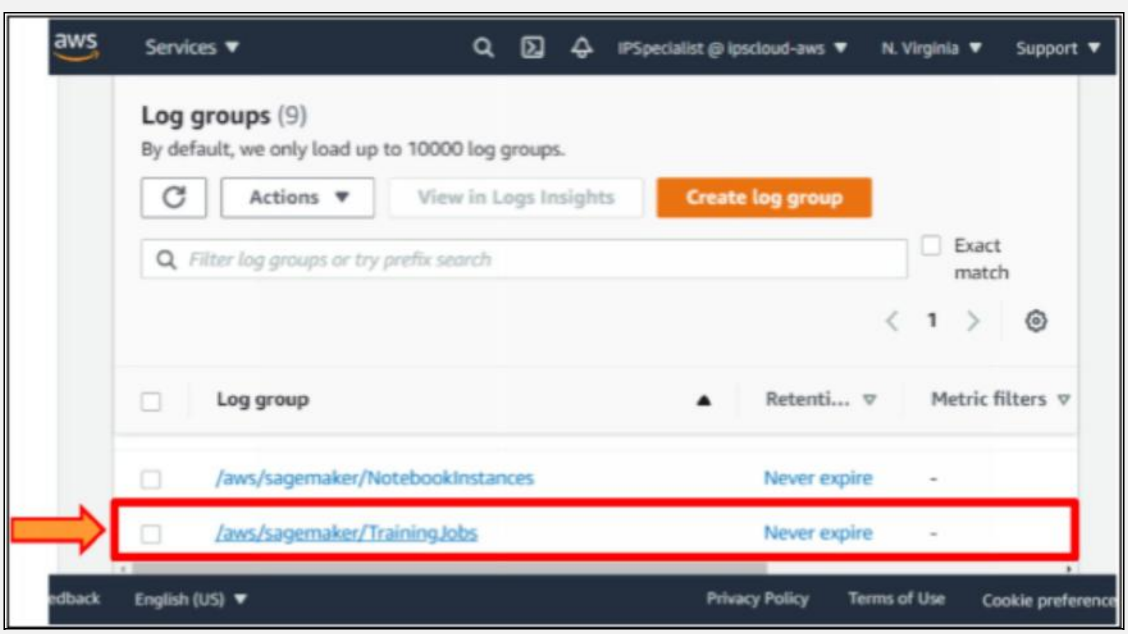
2. Select the **CloudWatch** from the **Management & Governance.**



3. Click on the **Logs groups** from the left-hand side.

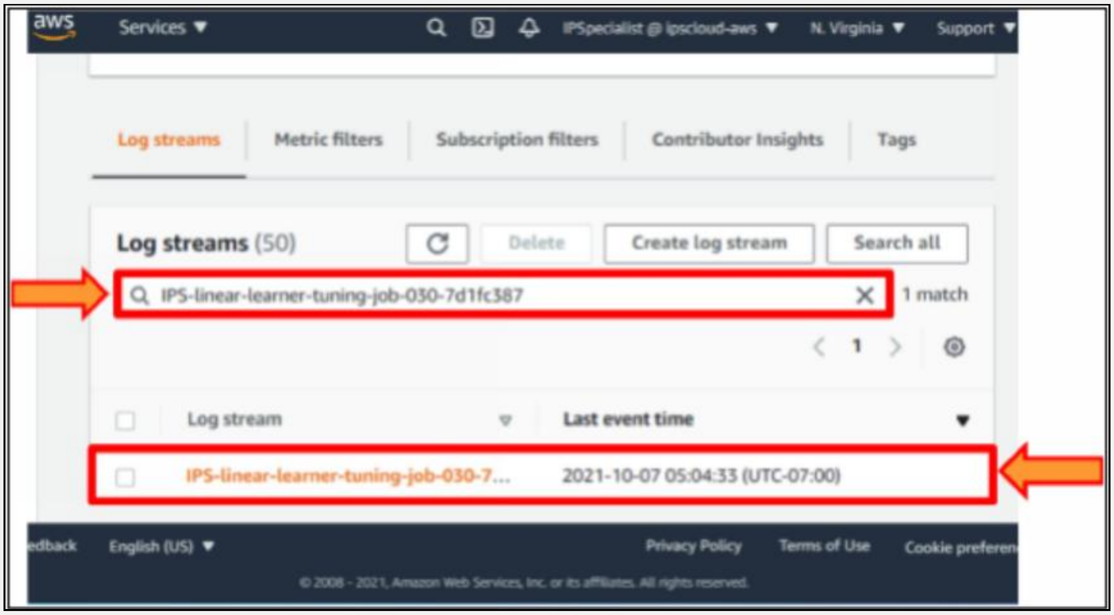


4. Click on the **/aws/sagemaker/TrainingJobs.**

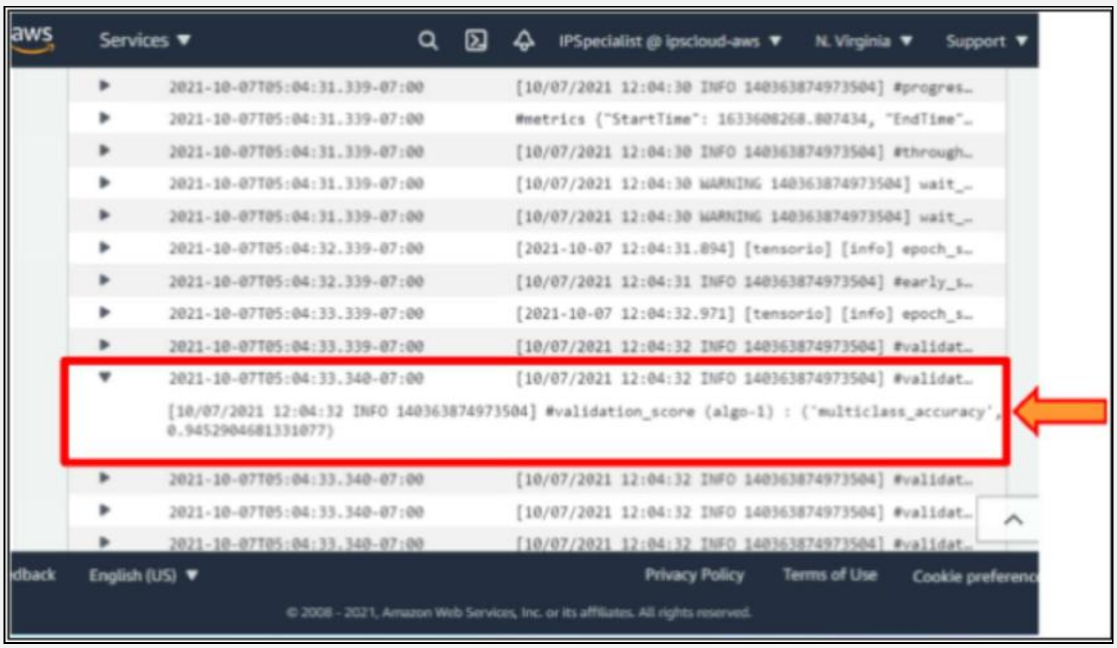


5. Paste the name **ips-linear-learner-tuning-job.**

6. Click on **ips-linear-learner-tuning-job.**

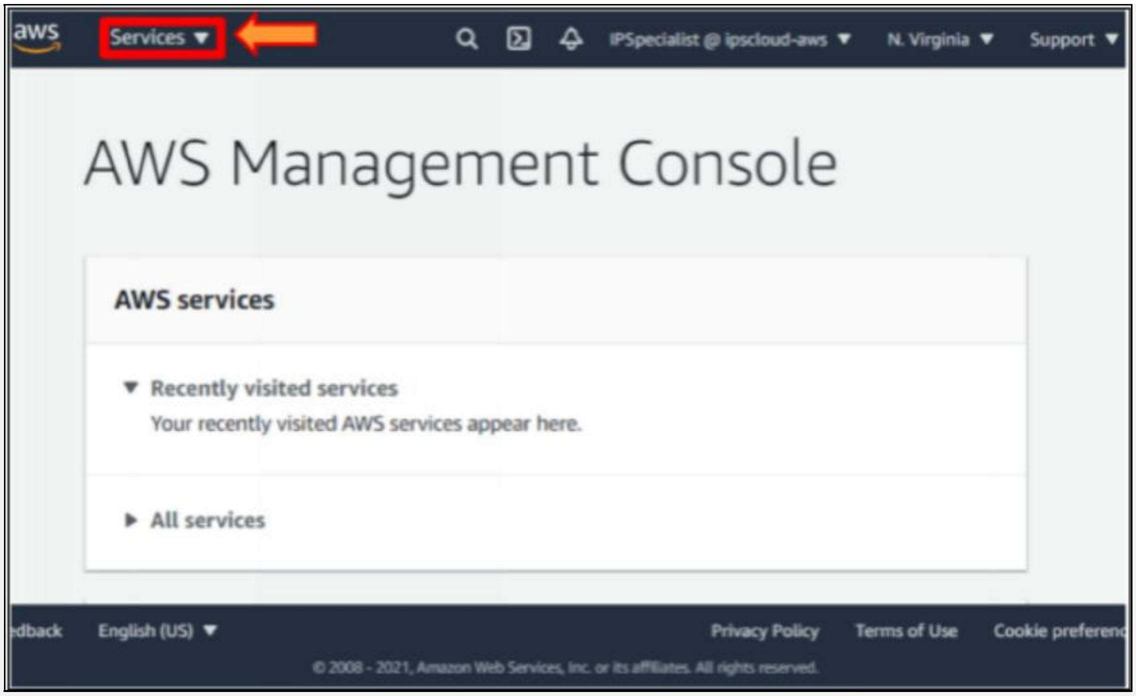


7. You will see the accuracy and other metrics.

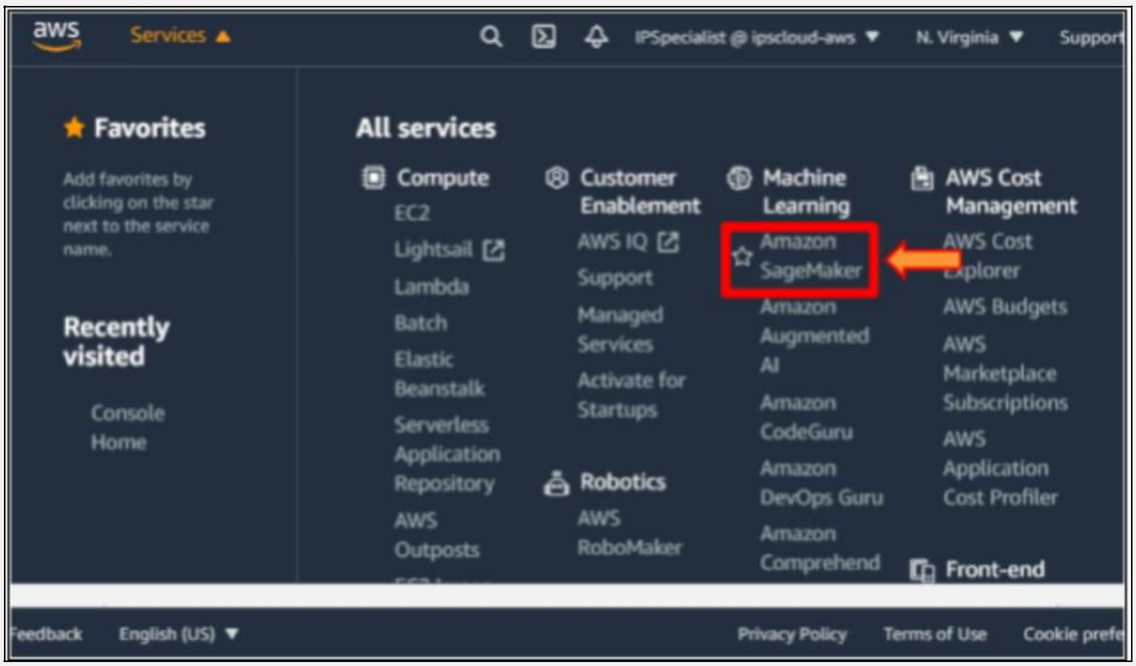


Step 4: Create SageMaker Notebook

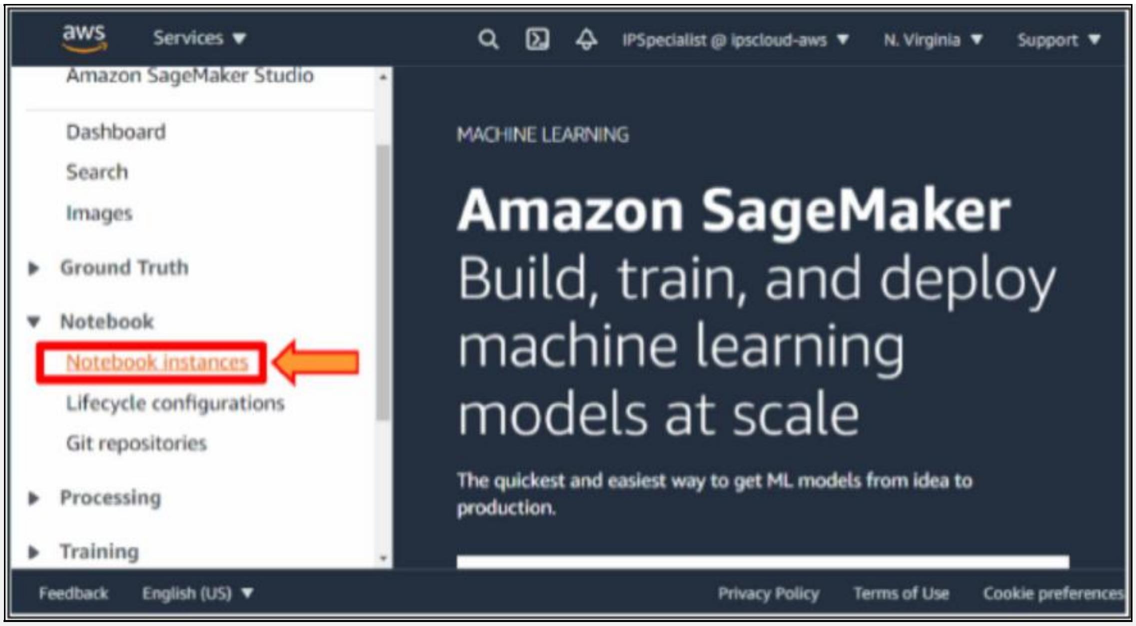
1. Click on **Services.**



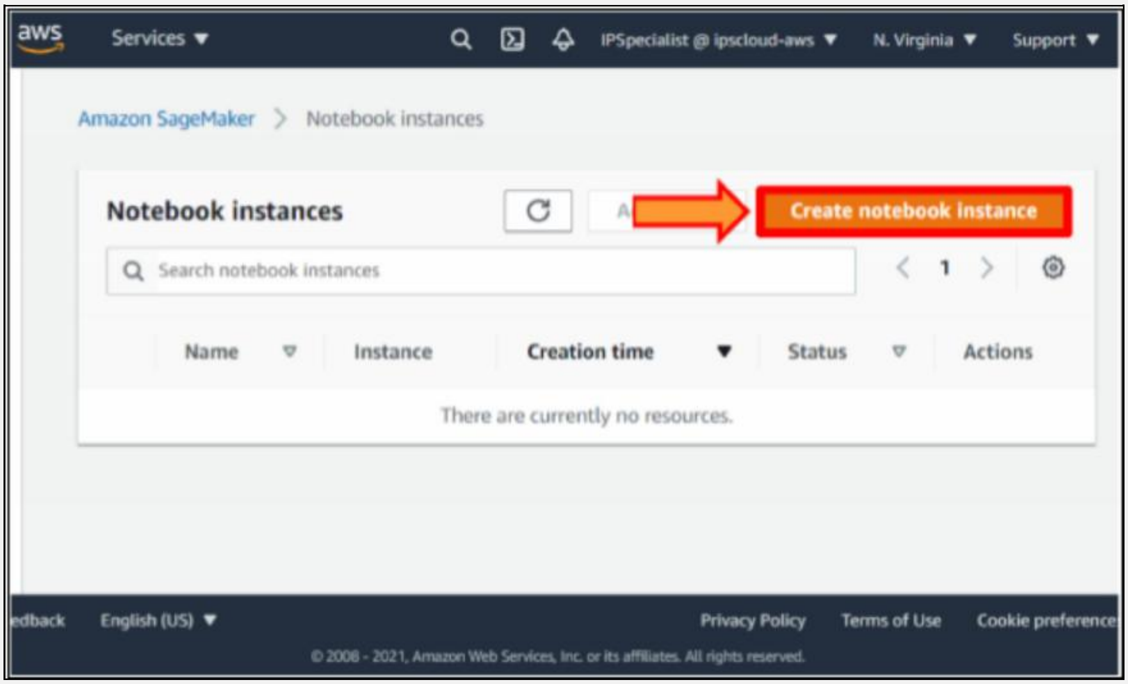
2. Select **SageMaker** from **Machine Learning.**



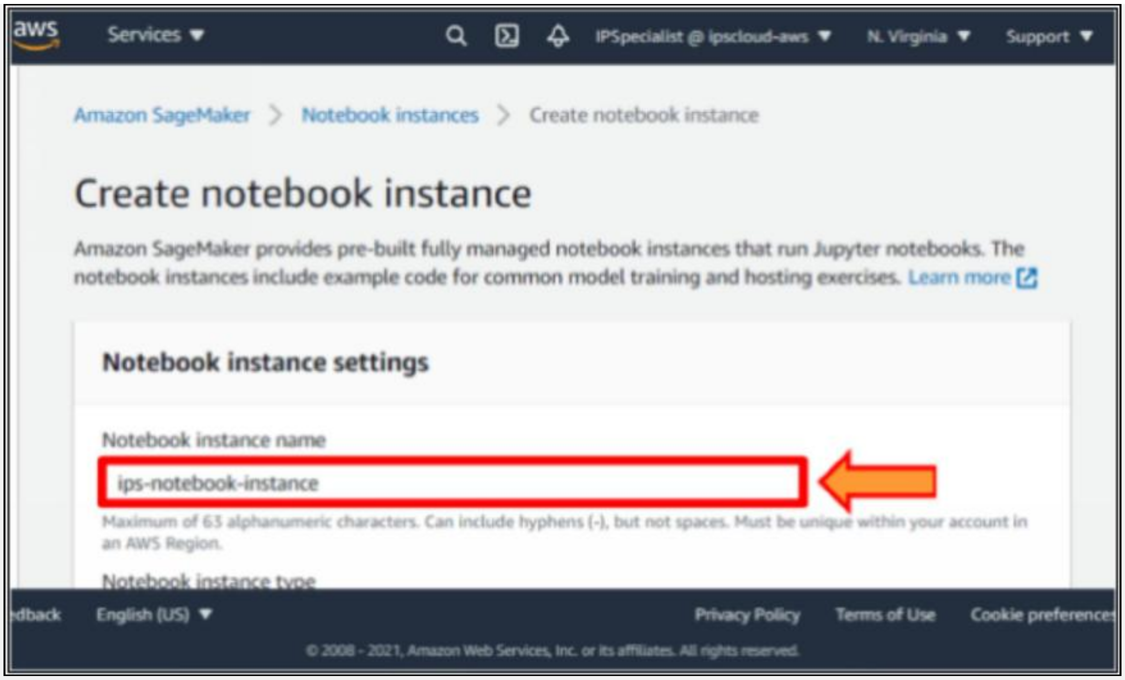
3. Click on **Notebook Instance** on the left-hand side menu.



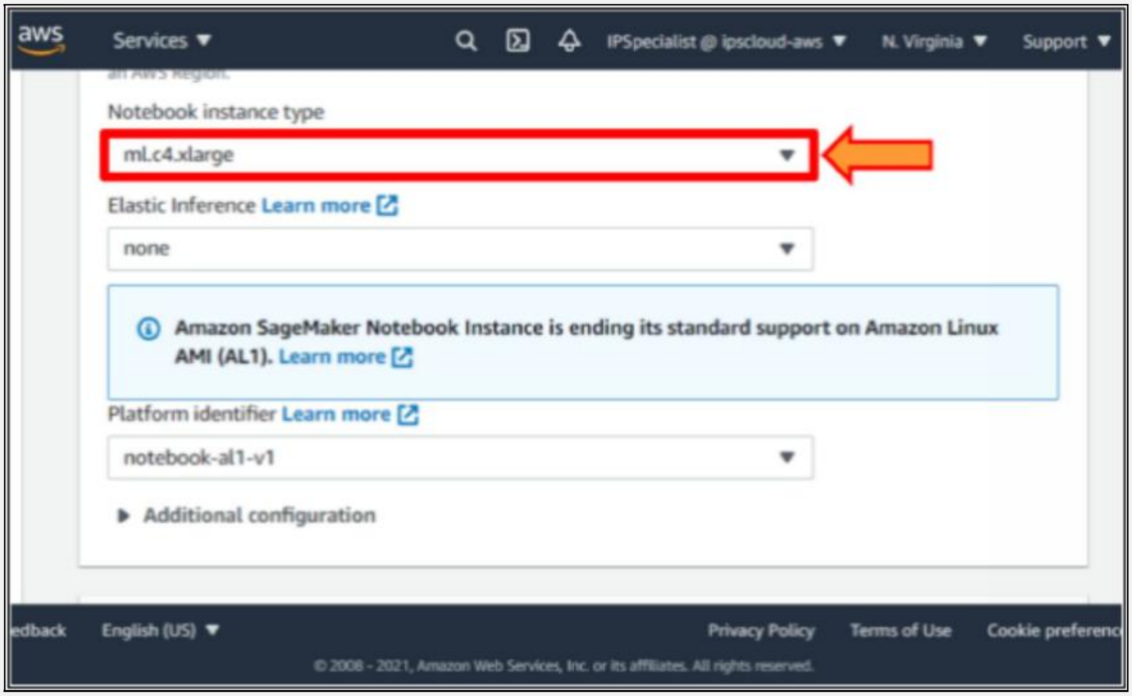
4. Click on the **Create notebook instance** button.



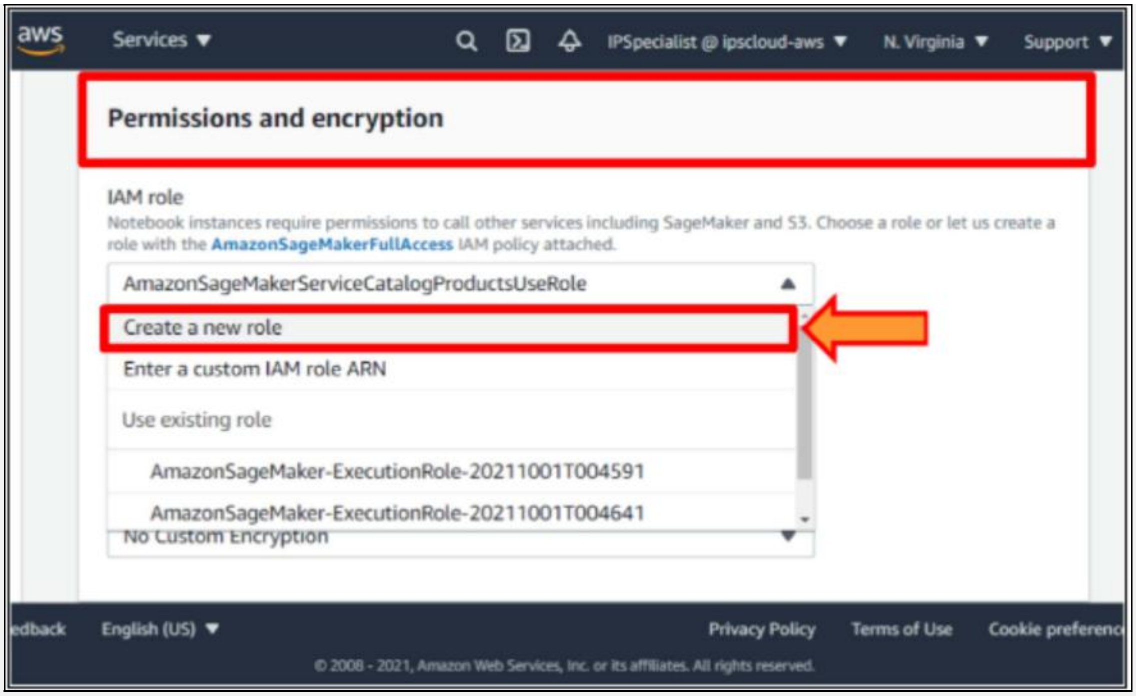
5. Give the name **ips-notebook-instance.**



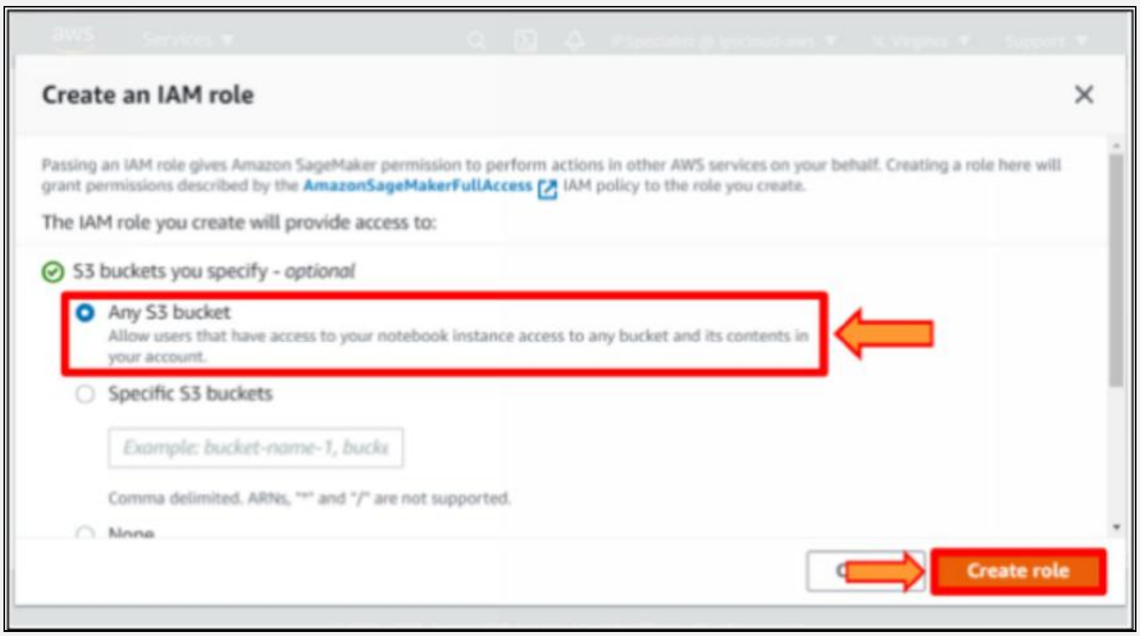
6. Select the **ml.c4.medium** Notebook instance type.



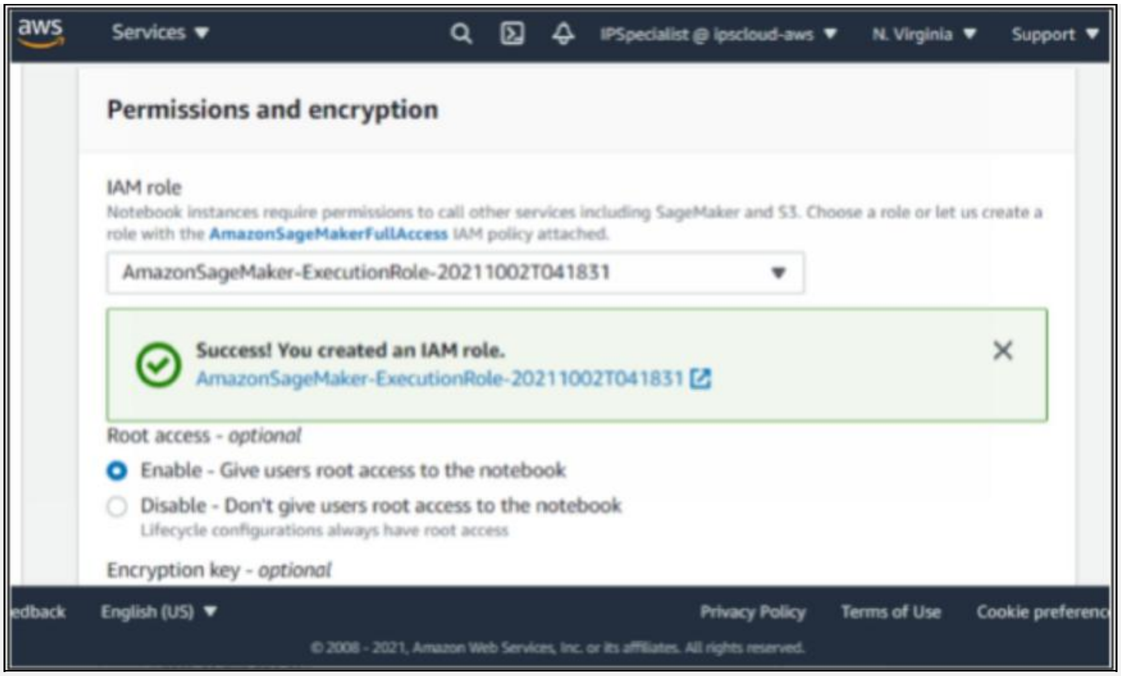
7. Scroll down under permission and encryption.
8. Click on **Create a new role.**



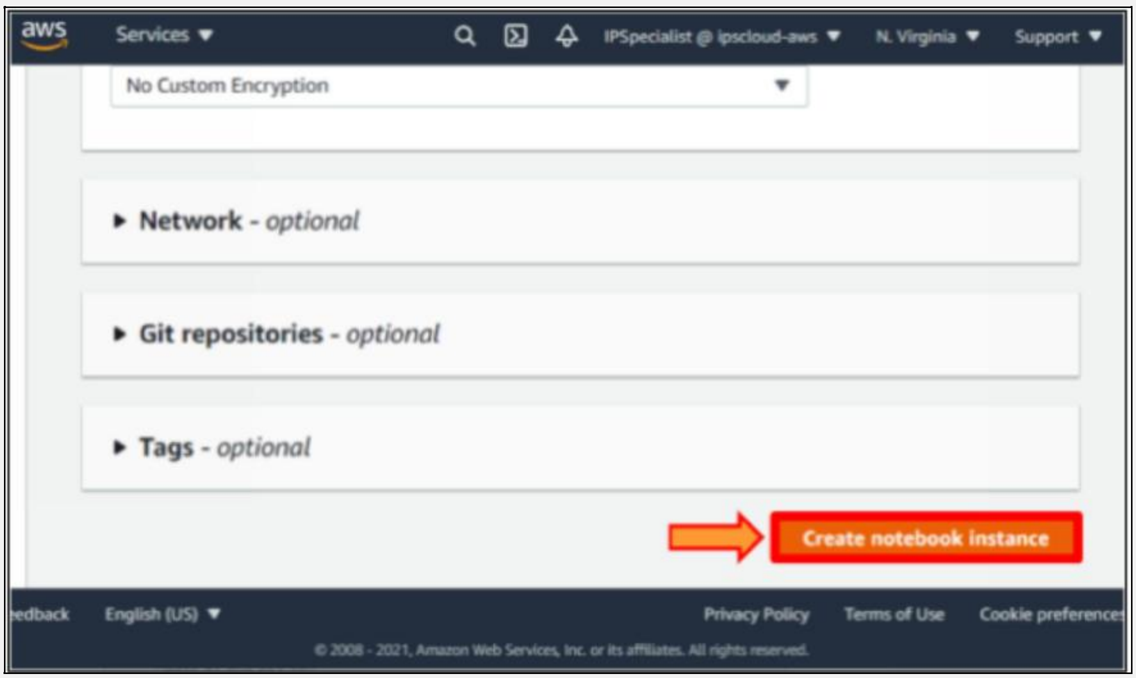
9. Select **Any S3 Bucket.**
10. Click on the **Create role** button.



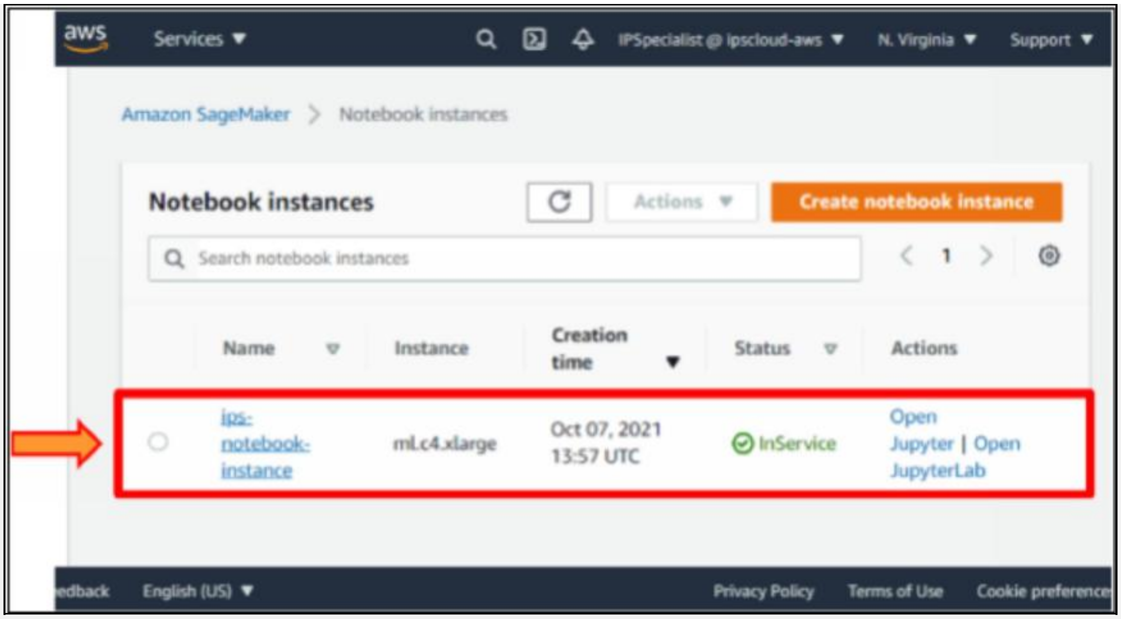
Hence, you have successfully created an IAM role.



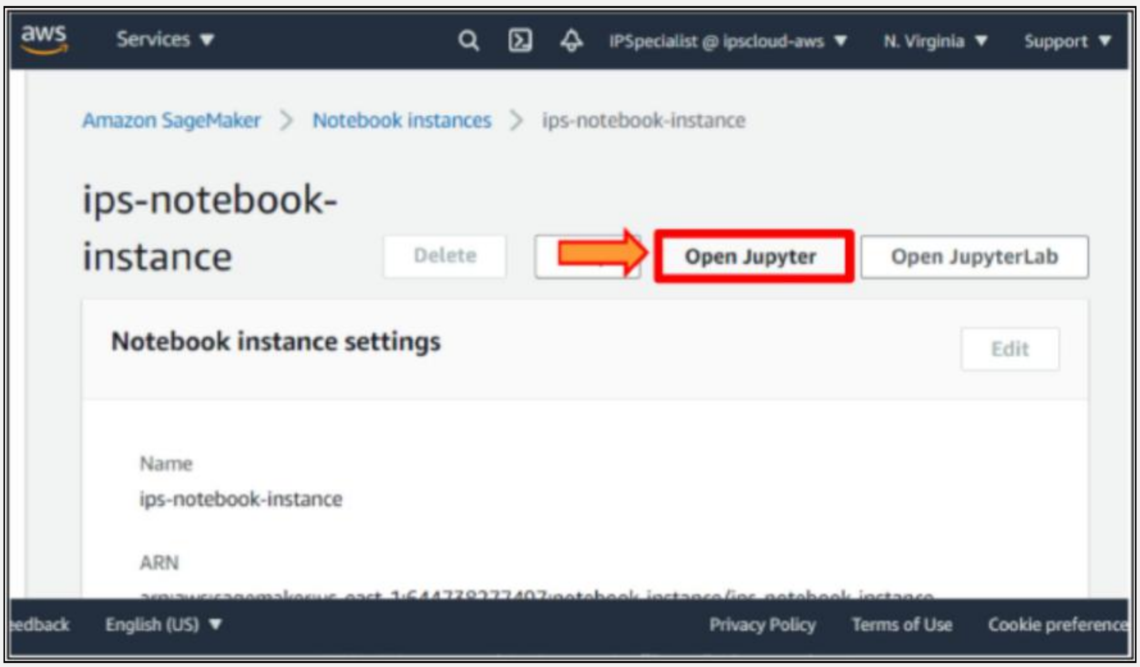
11. Scroll down. Click on the **Create notebook instance** button. It will take a few minutes.



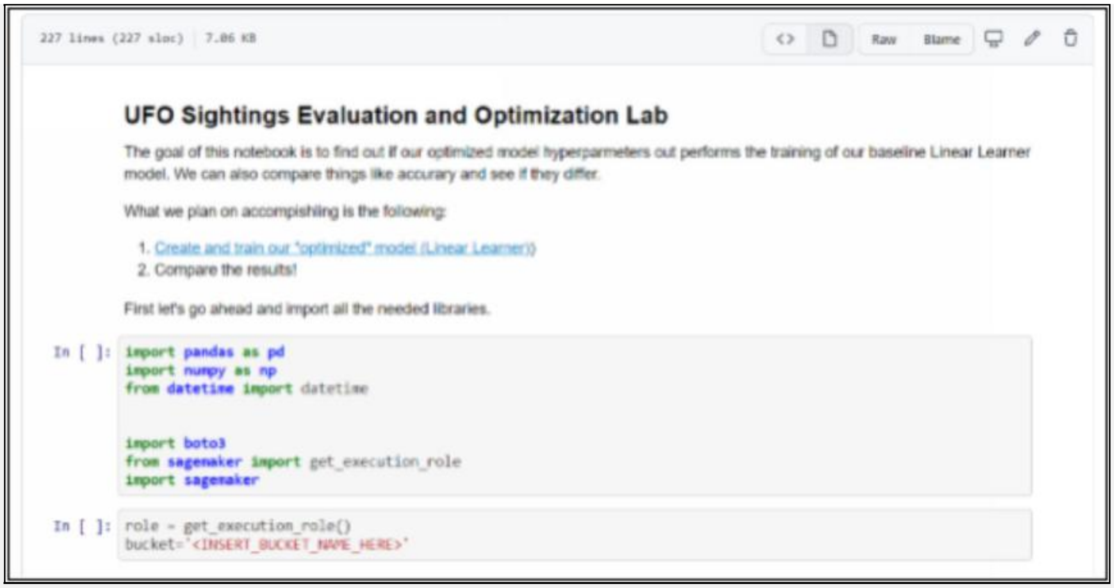
12. Click on **ips-notebook-instance.**



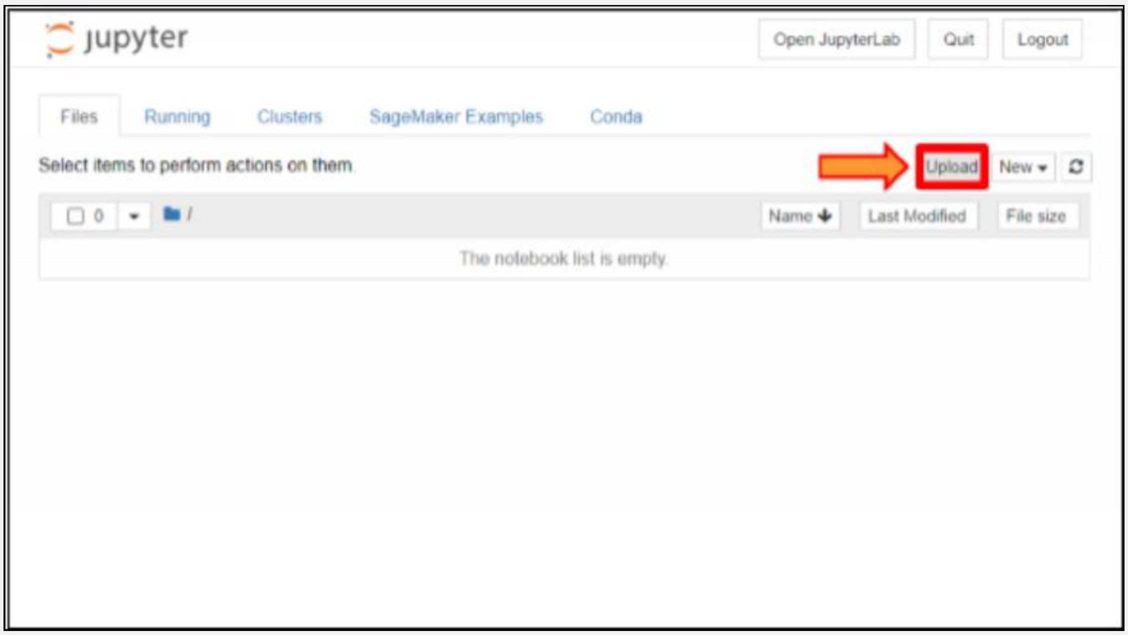
13. Click on the **Open Jupyter** button.



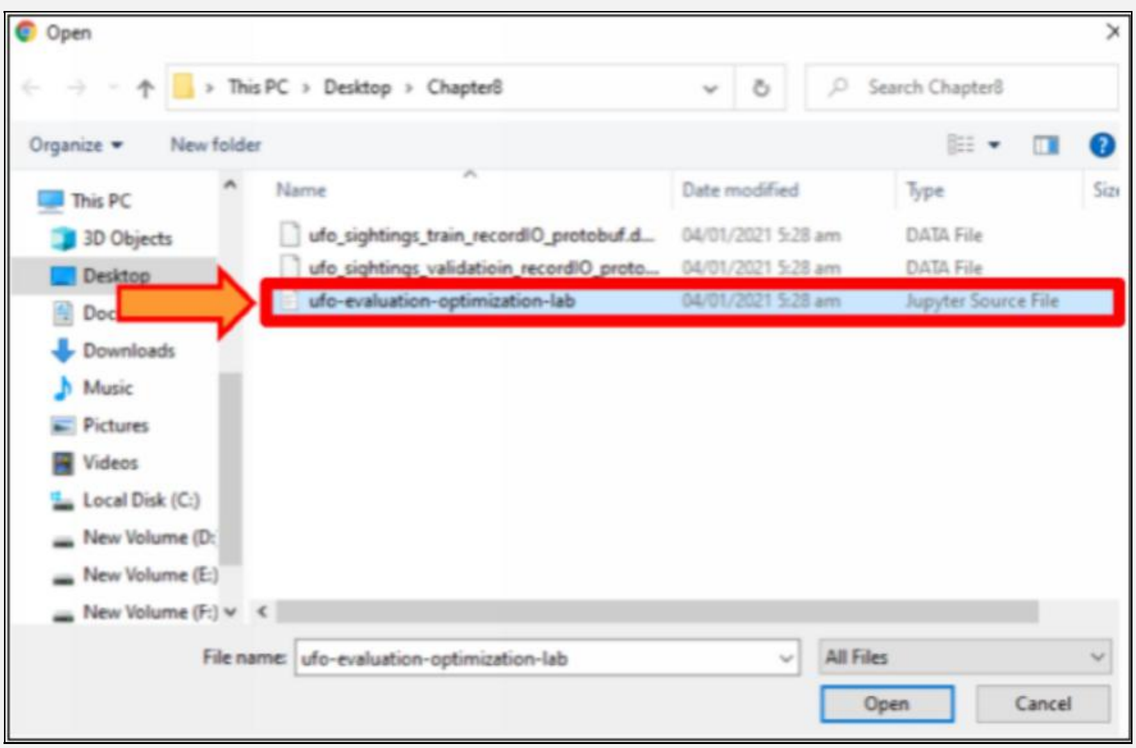
14. Download the jupyter notebook from the following Github link: https://github.com/ACloudGuru-Resources/Course_AWS_Certified_Machine_Learning/blob/master/Chapter8/ufo-evaluation-optimization-lab.ipynb



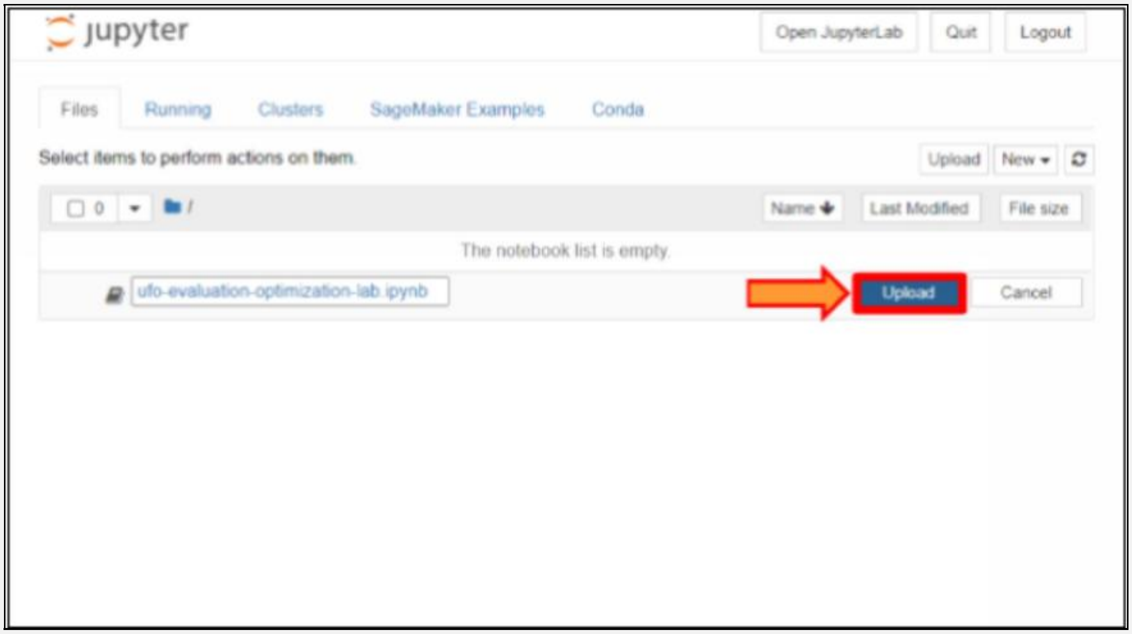
15. Click on the U**pload** button.



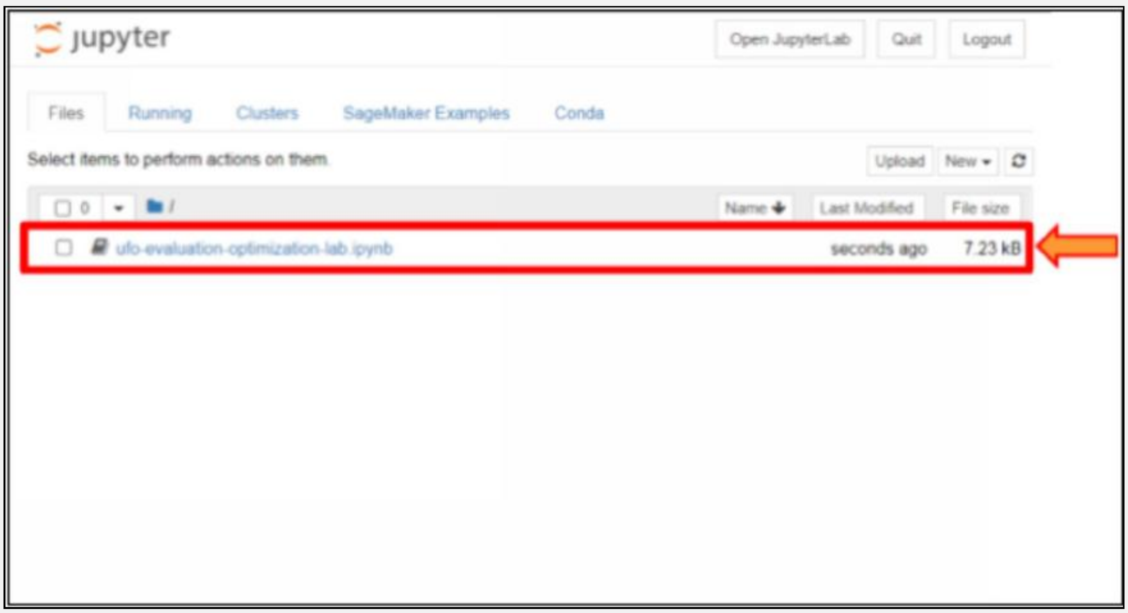
16. Select the **ufo-evaluation-optimization-lab.ipynb** jupyter notebook file.



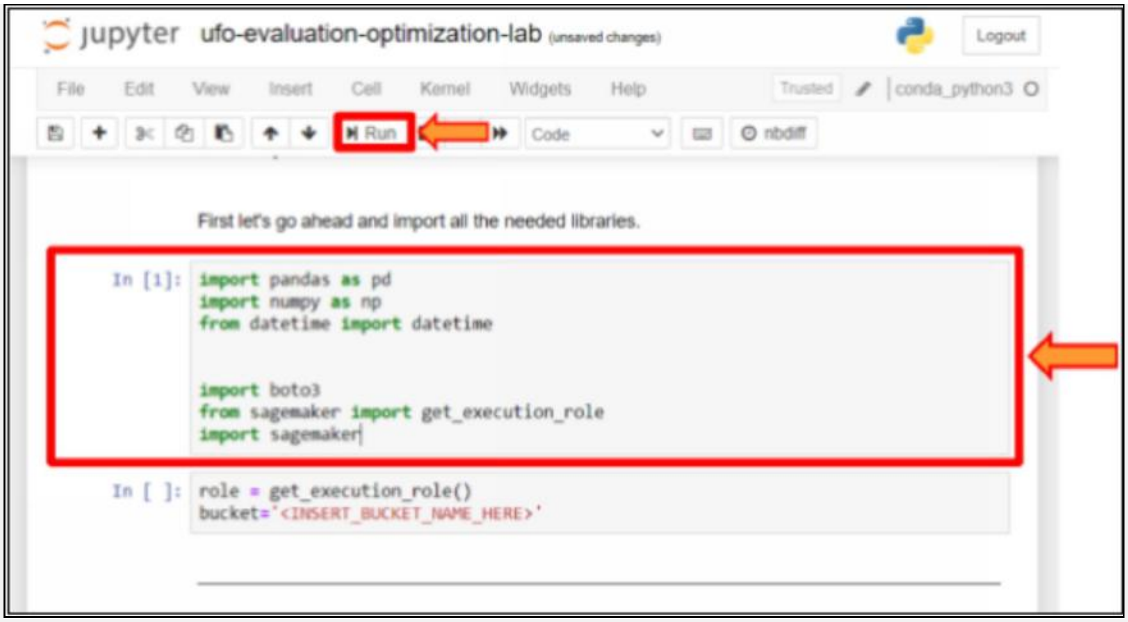
17. Click on the U**pload** button.



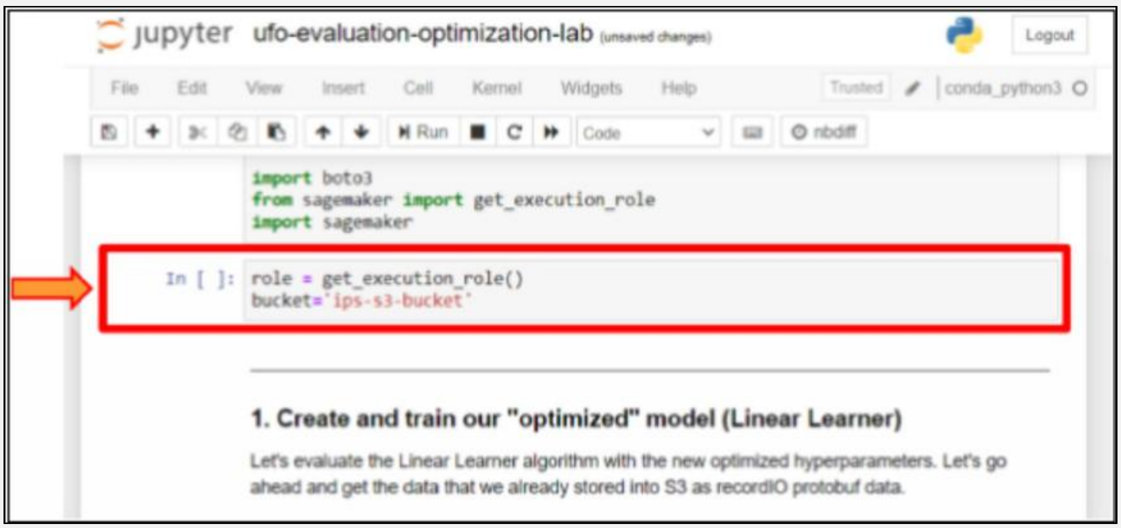
18. Click on the **ufo-evaluation-optimization-lab.ipynb** jupyter notebook.



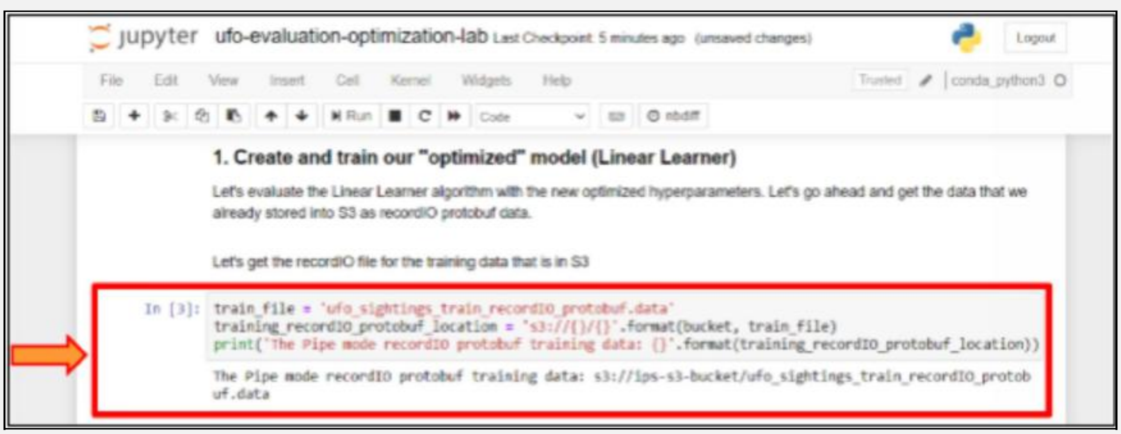
19. Run the cell to import python libraries. Click on the **Run** button or press **Shift + Enter** to execute the cell.



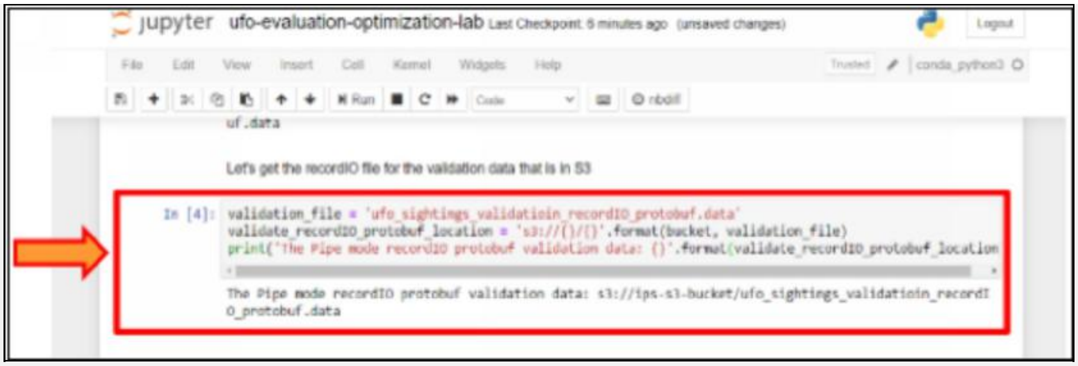
20. Insert the bucket name **ips-s3-bucket**. Then press **Shift + Enter** to execute the cell.



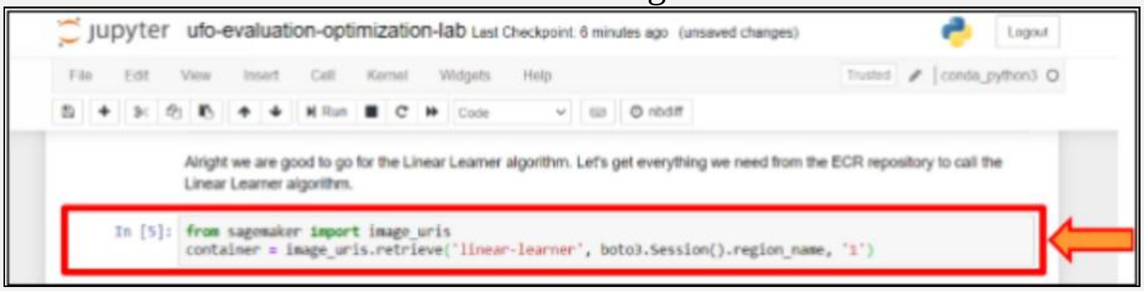
21. Run the cell to get the train data stored in S3. Press **Shift + Enter** to execute the cell.



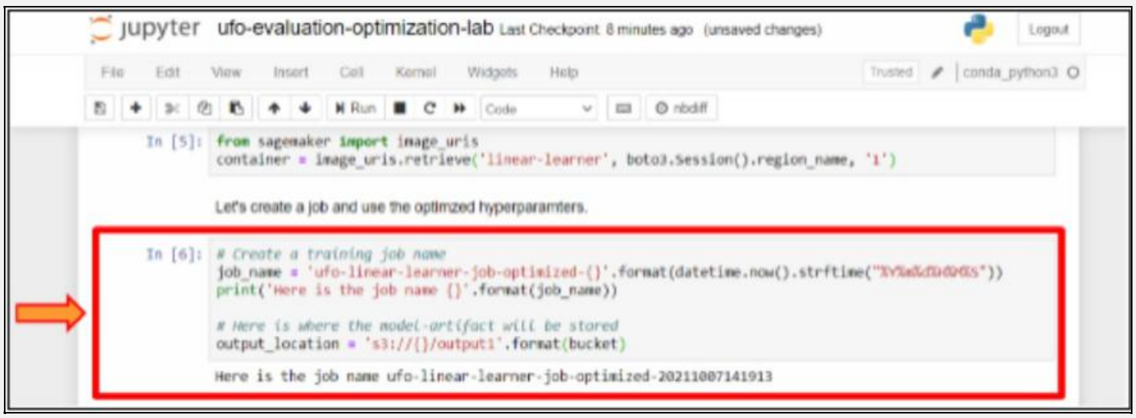
22. Run the cell to get the validation data stored in S3.



23. Run the cell to call the Linear Learner algorithm.



24. Run the cell to create a job and use the optimized hyperparameters.

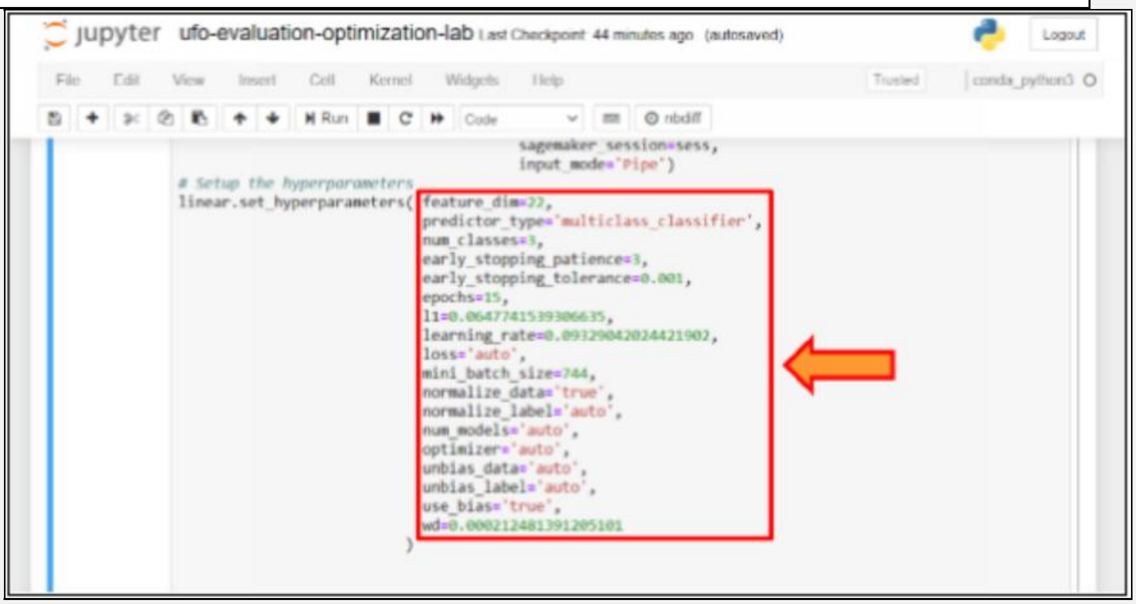


25. Insert the hyperparameters in the red box below.

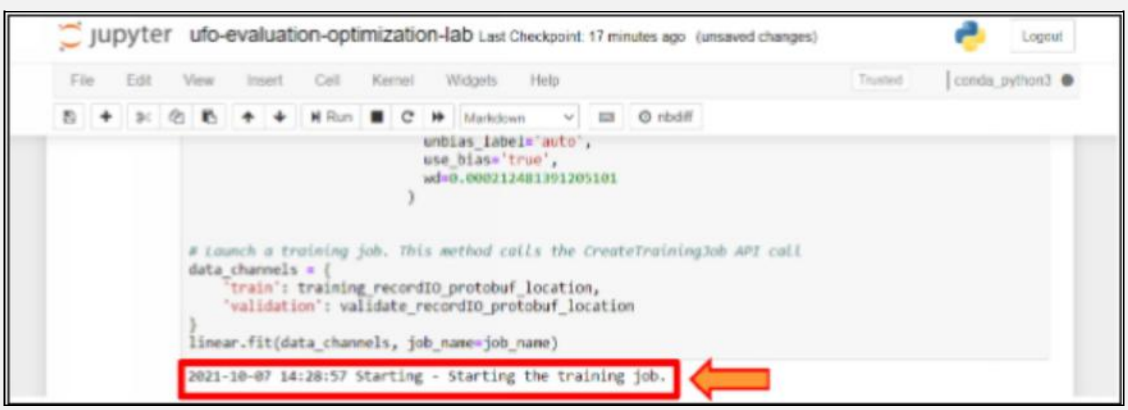
26. Insert the below hyperparameters.

```
early_stopping_patience=3
early_stopping_tolerance=0.001
epochs=15
```

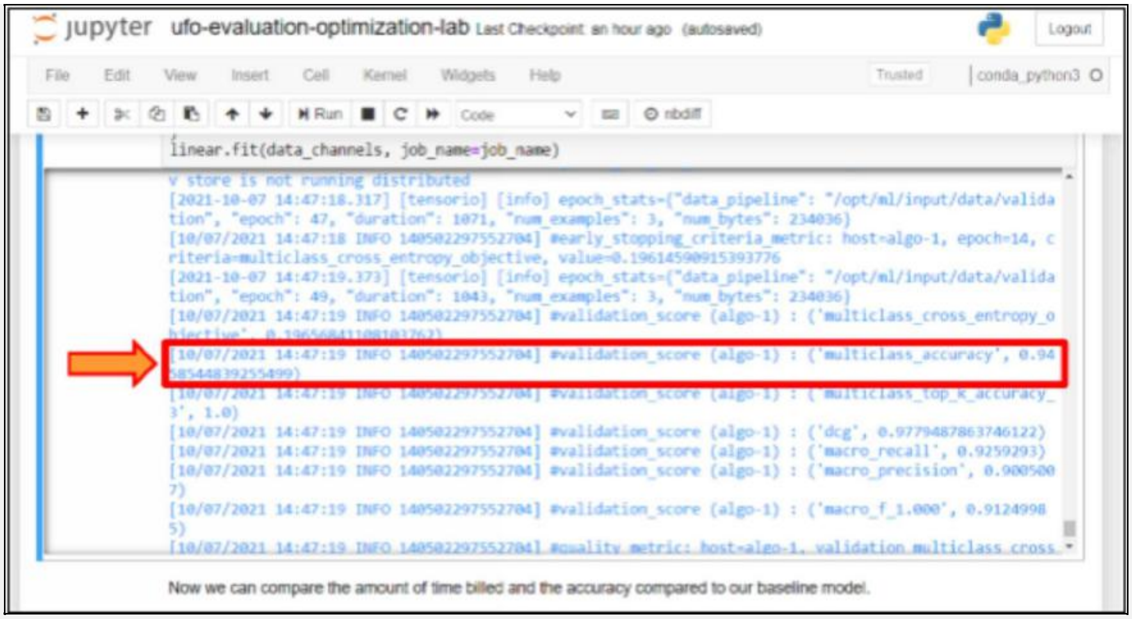
l1=0.0647741539306635
learning_rate=0.09329042024421902
loss='auto'
mini_batch_size=744
normalize_data='true'
normalize_label='auto'
num_models='auto'
optimizer='auto'
unbias_data='auto'
unbias_label='auto'
use_bias='true'
wd=0.000212481391205101



27. Run the cell to start the training job.
28. The training has started. It will take a few minutes to complete.



29. Now, the training is completed, and you will see the training accuracy logs.



30. After completing the lab, delete all the AWS services used in this lab so you would not get charged.

Lab 15: Categorizing Uploaded Data Using AWS Step Functions

Certifications	AWS Machine Learning - Specialty
Cloud Provider	AWS
Time Required	1 Hr.
Tags	Lambda, S3, AWS

Lab Prerequisites

- Familiarity with basic AWS Cloud Computing concepts and terminology
- An AWS account with an active subscription

Service Introduction

<u>Amazon Identity Access Management (IAM)</u>

Individuals and groups can be granted secure access to your AWS resources by using IAM. It allows you to create and manage IAM users and provide them access to your resources. Additionally, you have the option of granting access to users outside of AWS (federated users).

- **Managed Policy:** This contains the permission required to stop an EC2 instance
- **Inline Policy:** This allows this role to be passed to another service
- **Trust Policy:** This allows the System Manager and EC2 to assume the role. It enables EC2 to register with the Systems Manager and Systems Manager to stop the EC2 instance

Amazon Step Function

AWS Step Functions is a fully managed solution that makes it simple to coordinate the components of distributed applications and microservices using visual workflows. You may easily grow and alter applications by creating them from separate components that execute a specific function. Step Operations is a dependable method for coordinating components and stepping through your application's functions. Step Functions provides a graphical interface for organizing and visualizing your application's components as a series of stages. It makes creating and running multi-step apps a breeze. Step Functions automates the triggering and tracking of each step and retries when problems occur, ensuring that your application runs smoothly. Step Functions records the condition of each step so that you can immediately detect and troubleshoot problems if something goes wrong. You can alter and add stages without writing code, allowing you to quickly grow and innovate your application.

Amazon Simple Storage Service (S3)

Amazon S3 is a type of object storage that allows you to store and recover any quantity of data from any location. It is a low-cost storage solution with business resilience, reliability, efficiency, privacy, and infinite expansion.

Amazon S3 provides a straightforward web service interface for storing and retrieving any amount of data at any time and from any location. You may quickly create projects that integrate cloud-native storage using this service. Because Amazon S3 is easily customizable and you only pay for what you use, you can start small and scale up as needed without sacrificing performance or dependability.

Amazon S3 is also built to be highly adaptable. Instead of finding out how to store their data, Amazon S3 allows developers to focus on innovation. Build a simple FTP system or a complex web application like the amazon.com retail website; read the same piece of data a million times or only for emergency disaster recovery; store whatever type and amount of data you desire.

Amazon Lambda Function

AWS Lambda allows you to run code without creating or managing servers. There is no charge when your code is not executing; you only pay for the compute time you use. You can run code for nearly any application or backend service with Lambda, and you do not have to worry about administration. Upload your code, and Lambda will handle everything necessary to run and grow it with high availability. You may configure your code to be automatically triggered by other AWS services, or you can access it directly from any computer or smartphone app.

Amazon Transcribe

Conduct a (text-based) content analysis on audio/video content using Automatic Speech Recognition (ASR) technology. Amazon Transcribe is an AWS service that makes converting speech to text simple for customers. Customers may utilize Amazon Transcribe for various commercial applications, such as voice-based customer care call transcription and the production of subtitles for audio/video material.

Case Study Enterprise Internet Television Network – Netflix

Background

Netflix is the world’s leading internet television network, with over 200 million members in more than 190 countries enjoying 125 million hours of TV shows and movies daily. Netflix uses AWS for nearly all its computing and storage needs, including databases, analytics, recommendation engines, video transcoding, and more—hundreds of functions that in total use more than 100,000 server instances on AWS.

Business Challenge

You are a Machine Learning engineer at Netflix. Your firm has just begun recording meetings to increase openness across the board. Your organization creates a lot of audio daily because of the number of meetings they have. It is wonderful to have these recordings, but being able to search through them as text would be far more beneficial since it would save a lot of time listening back over the material. This appears to be a machine learning job! One problem is that discussions regarding secret projects are taped because every meeting is taped. For better access control, those should be relocated to the side.

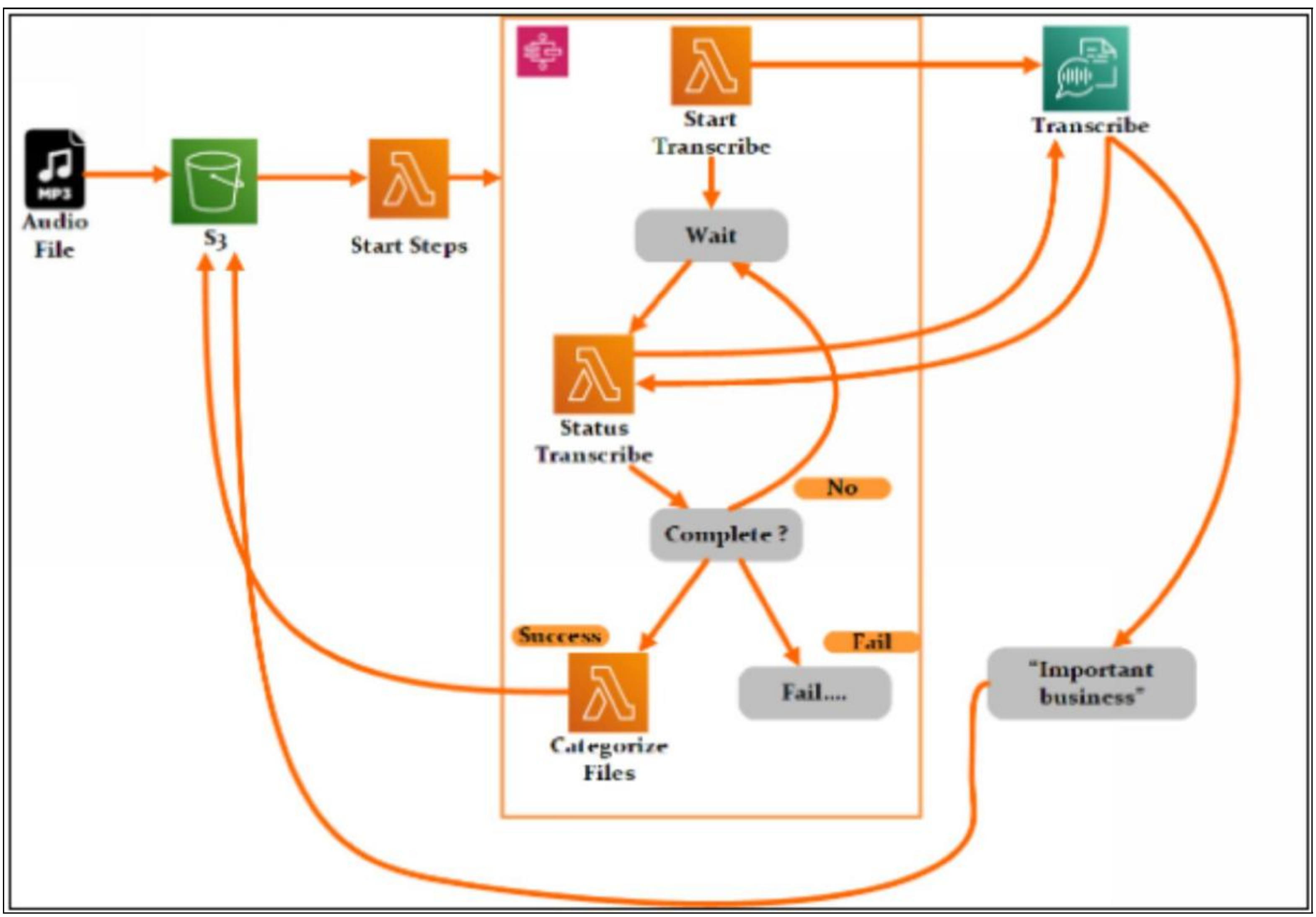
Your company asks you to extract the key features from the customer calls. Your goal is to create a pipeline to process all this audio and convert it to text. The transcript must be checked to determine whether any secret project names appear. Audio containing secret projects should be placed in the "important" folder rather than the normal "processed" folder.

Proposed Solution

The solution is you use AWS services to automate this task. You use AWS transcribe, which converts real-time speech into text. Then you use Lambda functions as a helping service. After that, you use the step function to create a model. To store sensitive data, you use S3.

Before starting the lab, create an S3 bucket and lambda functions mentioned in previous labs.

Lab Diagram



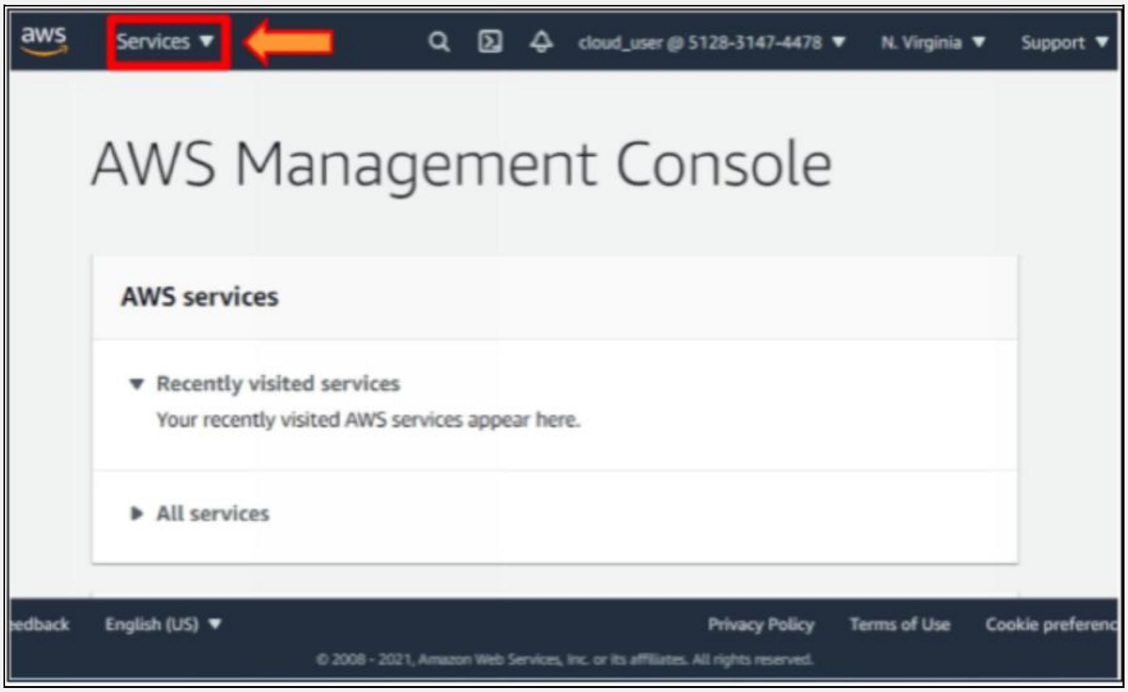
Implementation Steps

1. Create an IAM Role.
2. Create Step Function.
3. Update Lambda to Work with Step Functions.
4. Update S3 to Work with Step Functions.
5. Create Step Function Flow.
6. Create the Lambda Business Logic.
7. Categorize Audio Data.

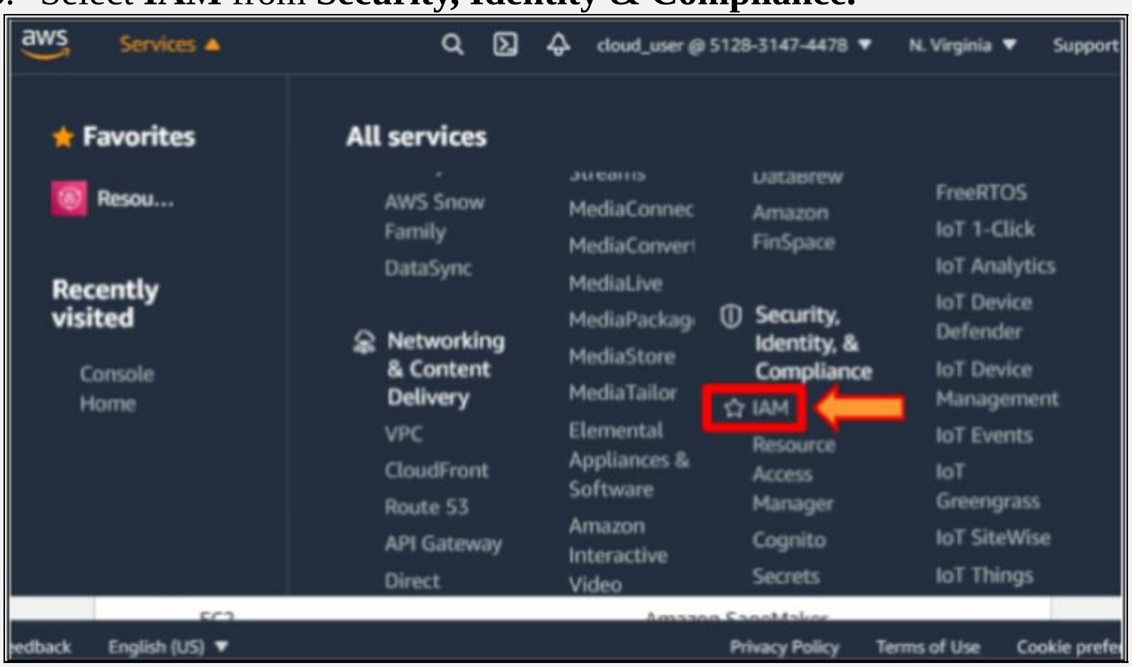
Solution

Step 1: Create an IAM Role

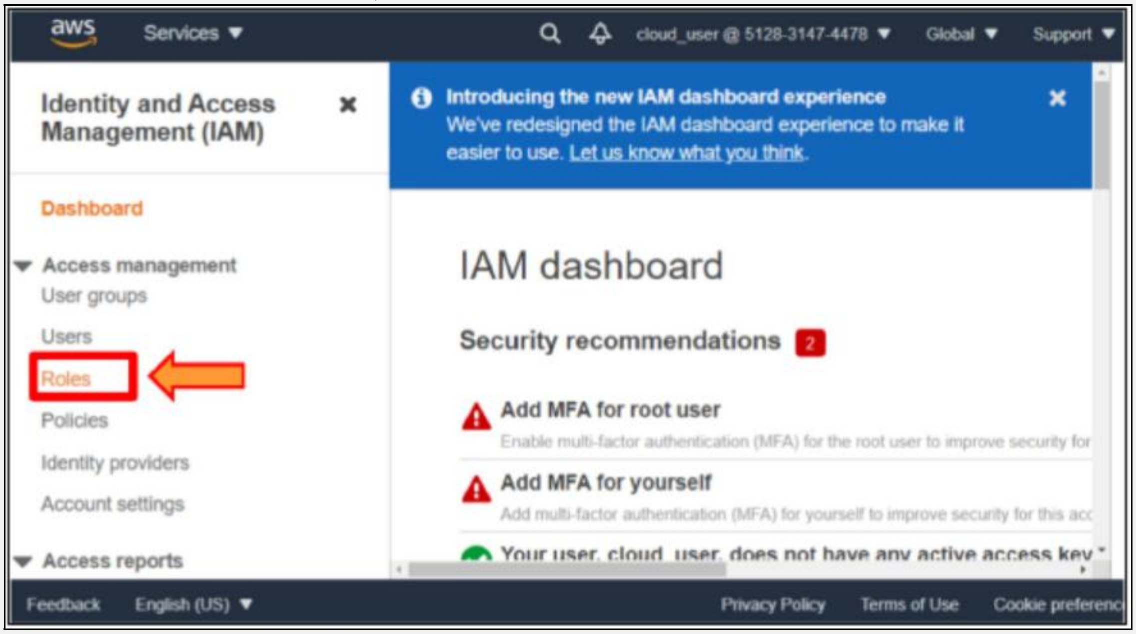
1. Log in to the **AWS Console.**
2. Click on **Services.**



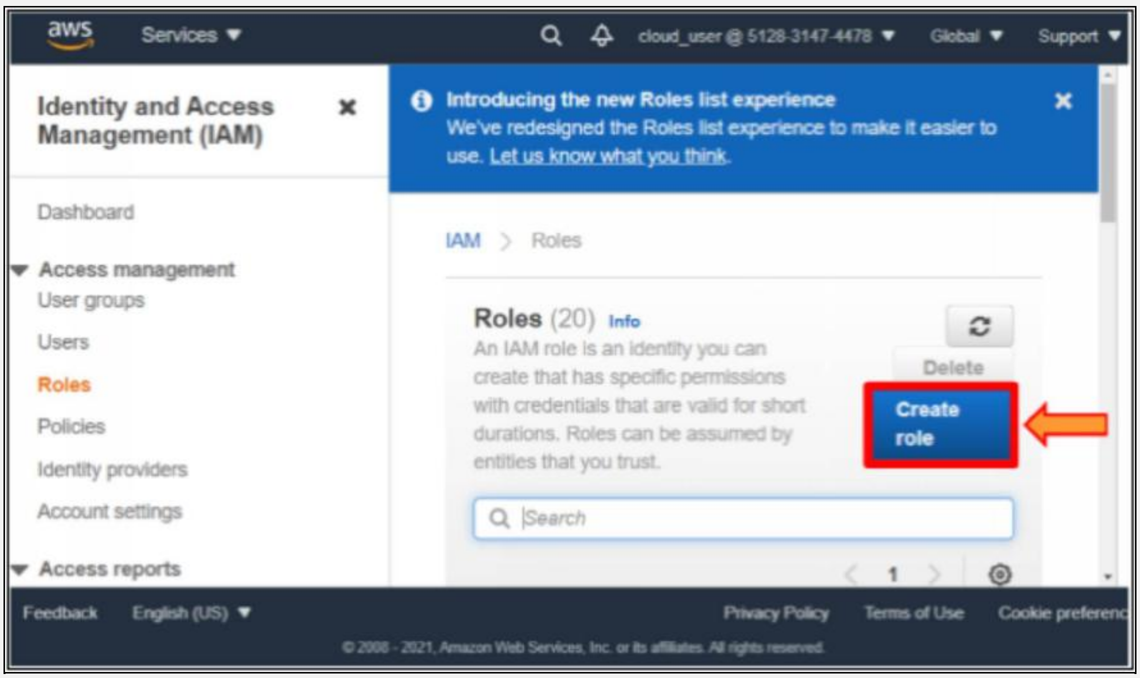
3. Select **IAM** from **Security, Identity & Compliance.**



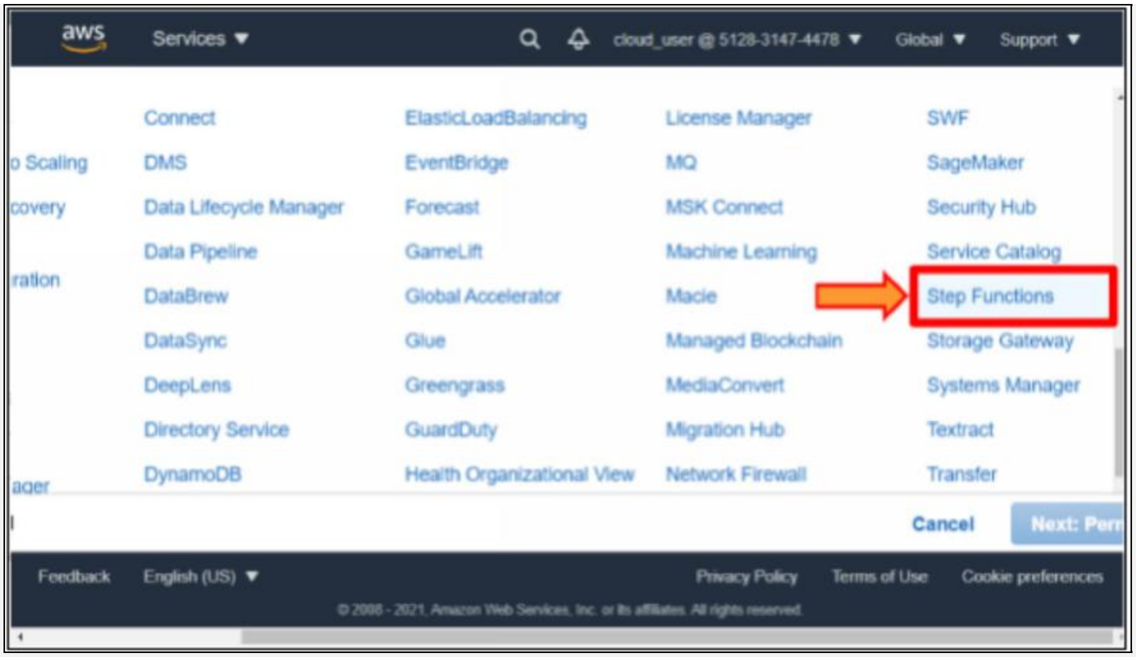
4. In the left-hand menu, click on **Roles**.



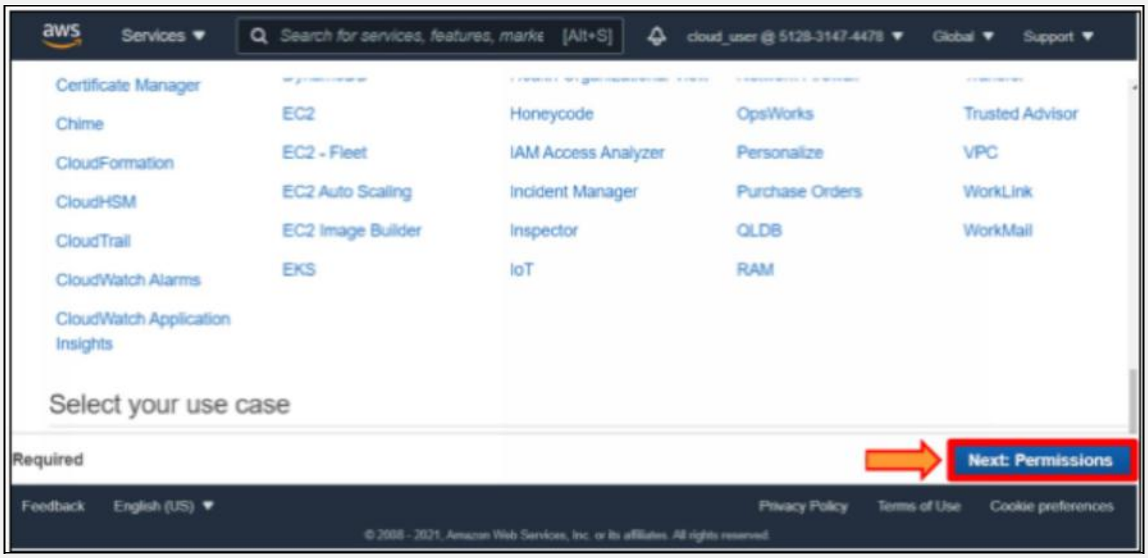
5. Click on the **Create role** button.



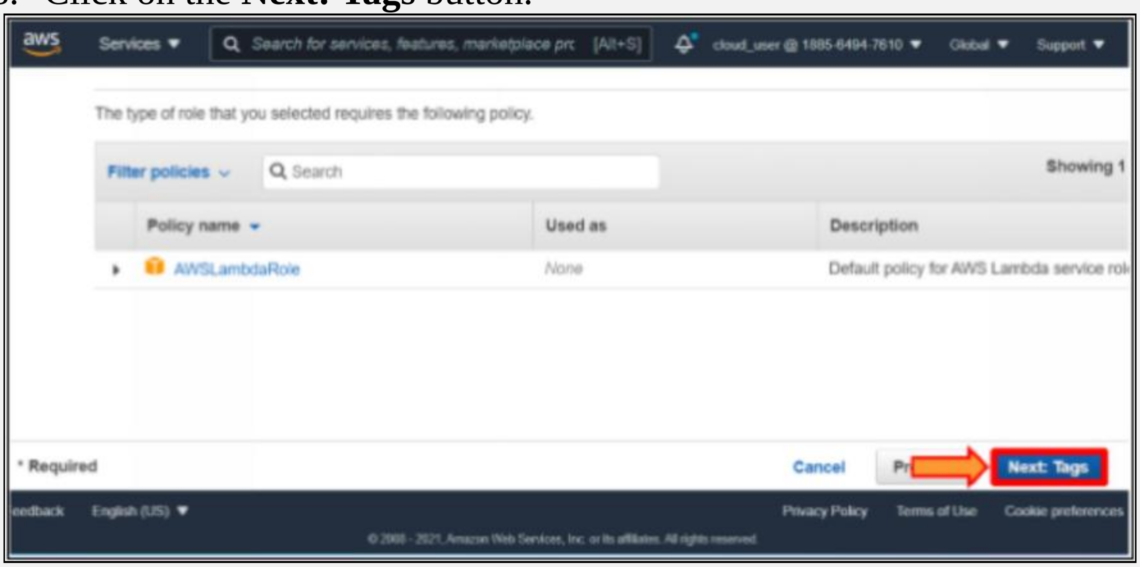
6. Click on **Step Functions**.



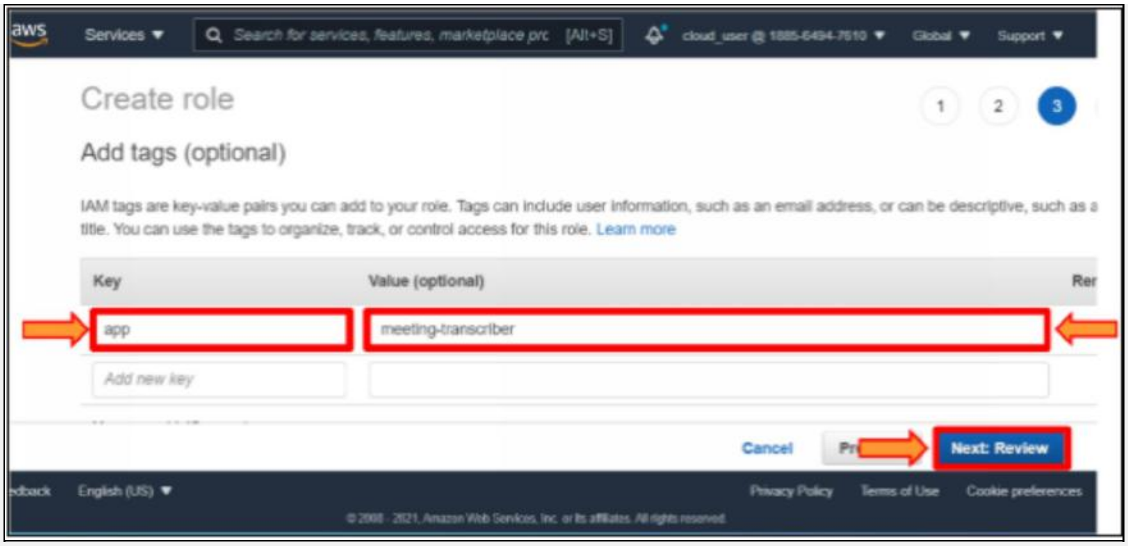
7. Click on the **Next: Permissions** button.



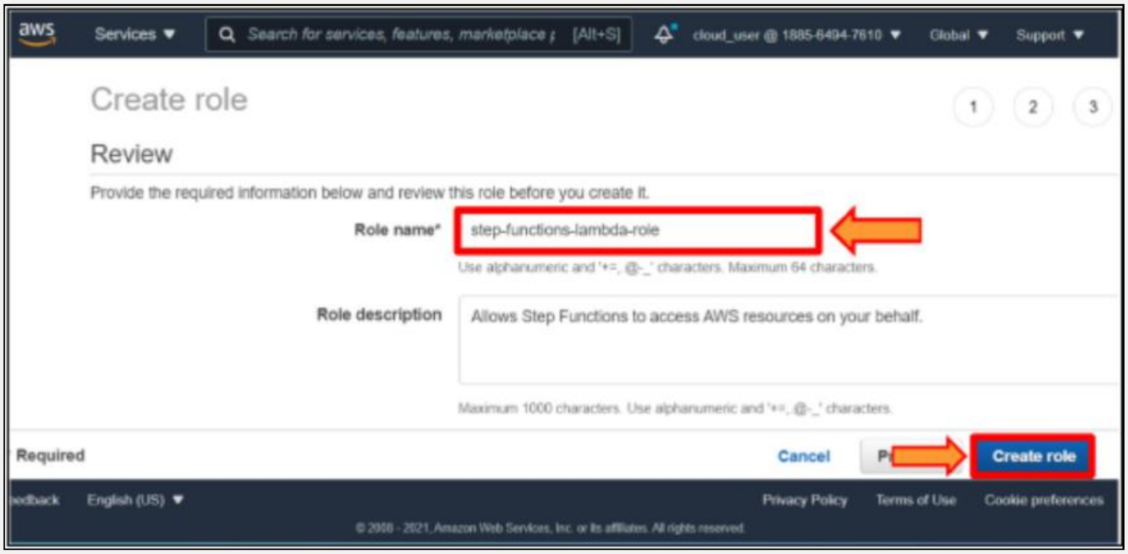
8. Click on the **Next: Tags** button.



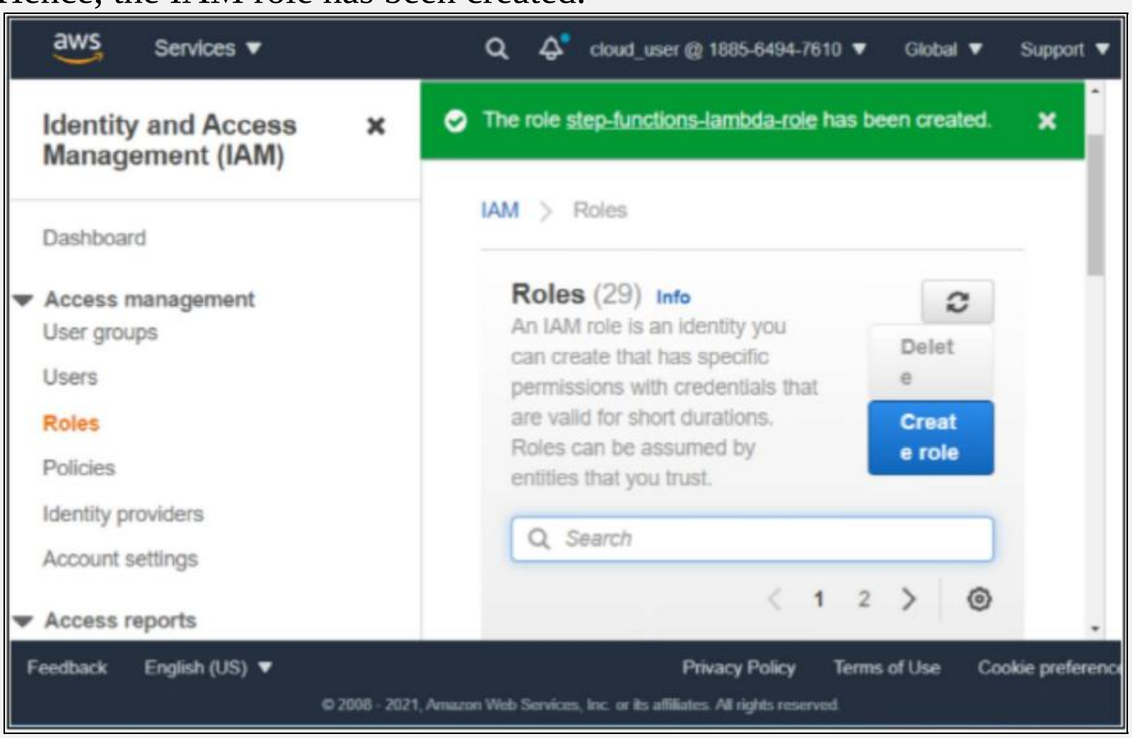
9. Create a tag by entering the **app** under **Key** and **meeting-transcriber** under **Value**.
10. Click on the **Next: Review** button.



11. In Role name, enter the name **step-functions-lambda-role**.
12. Click on the **Create role** button.

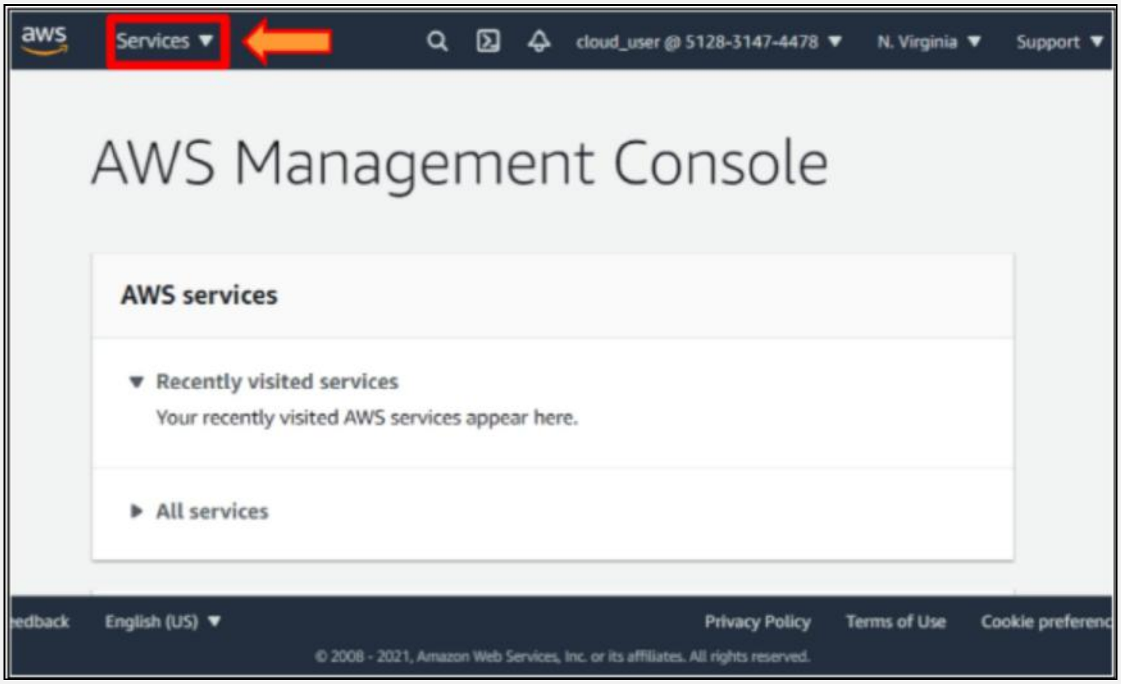


Hence, the IAM role has been created.

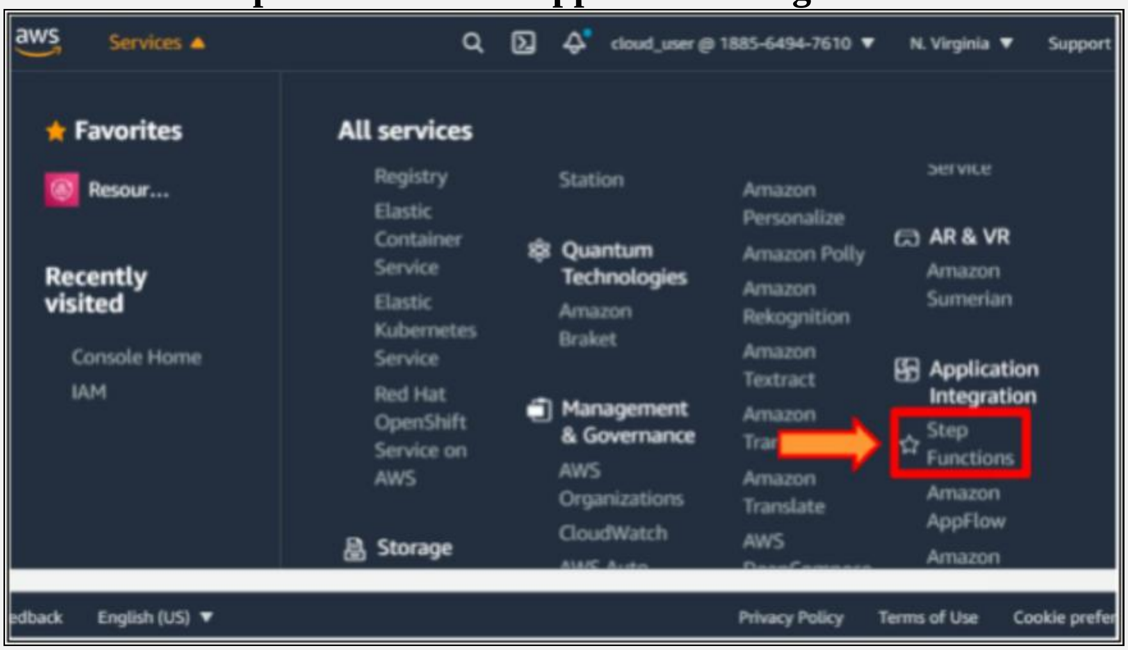


Step 2: Create Step Function

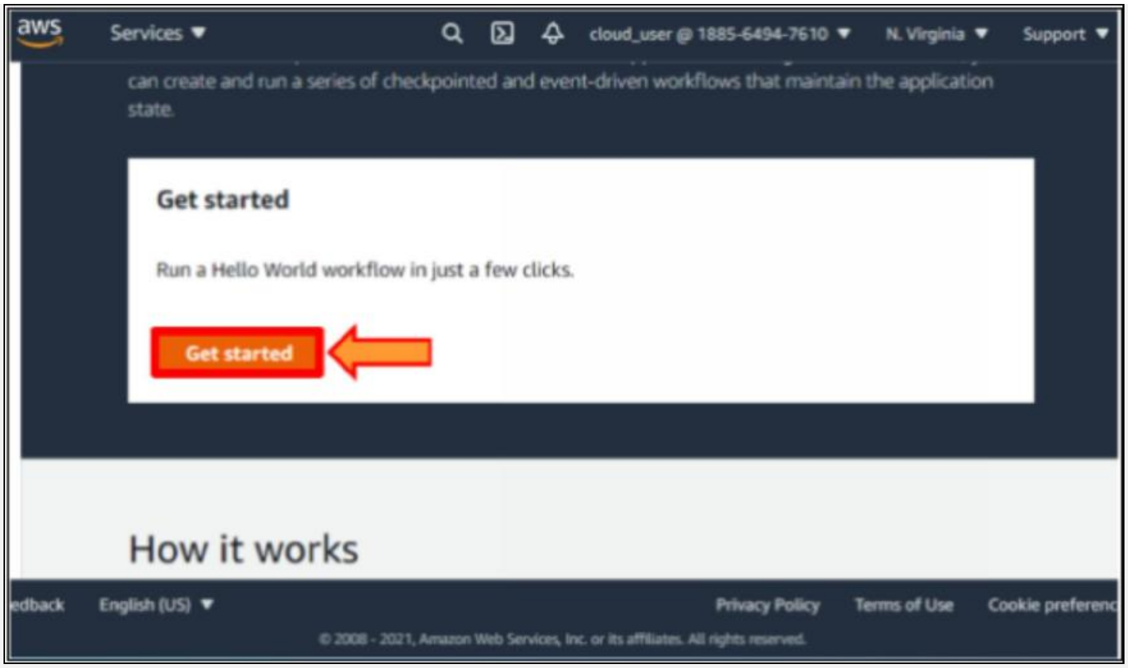
1. Click on **Services.**



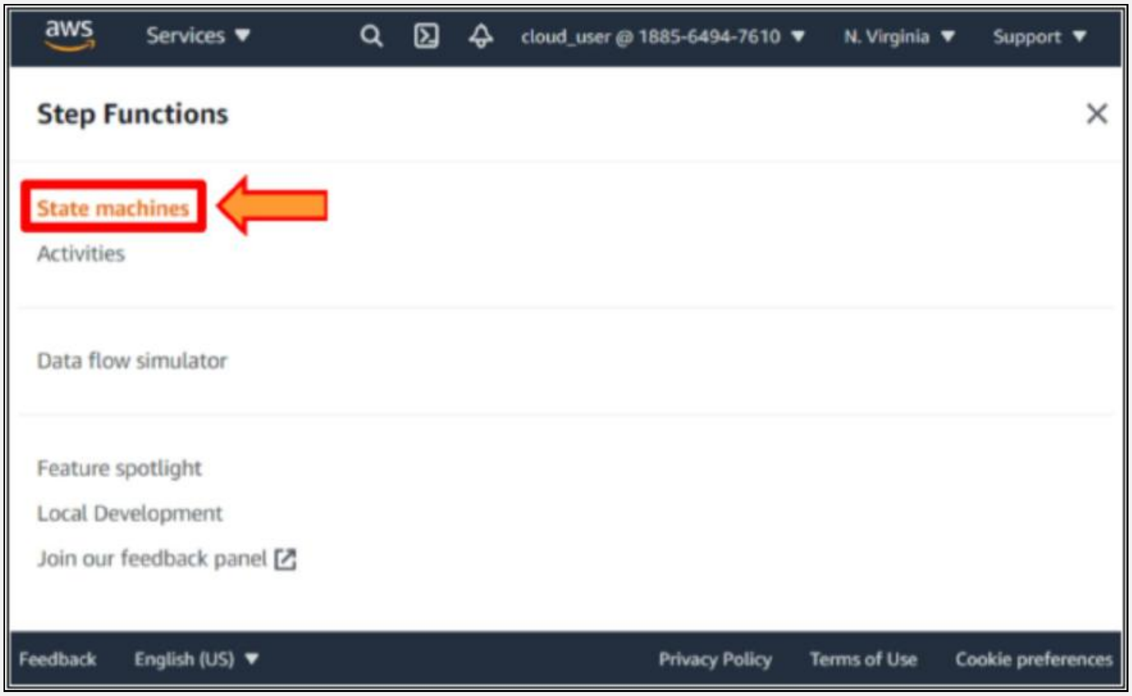
2. Select the **Step Function** from **Application Integration.**



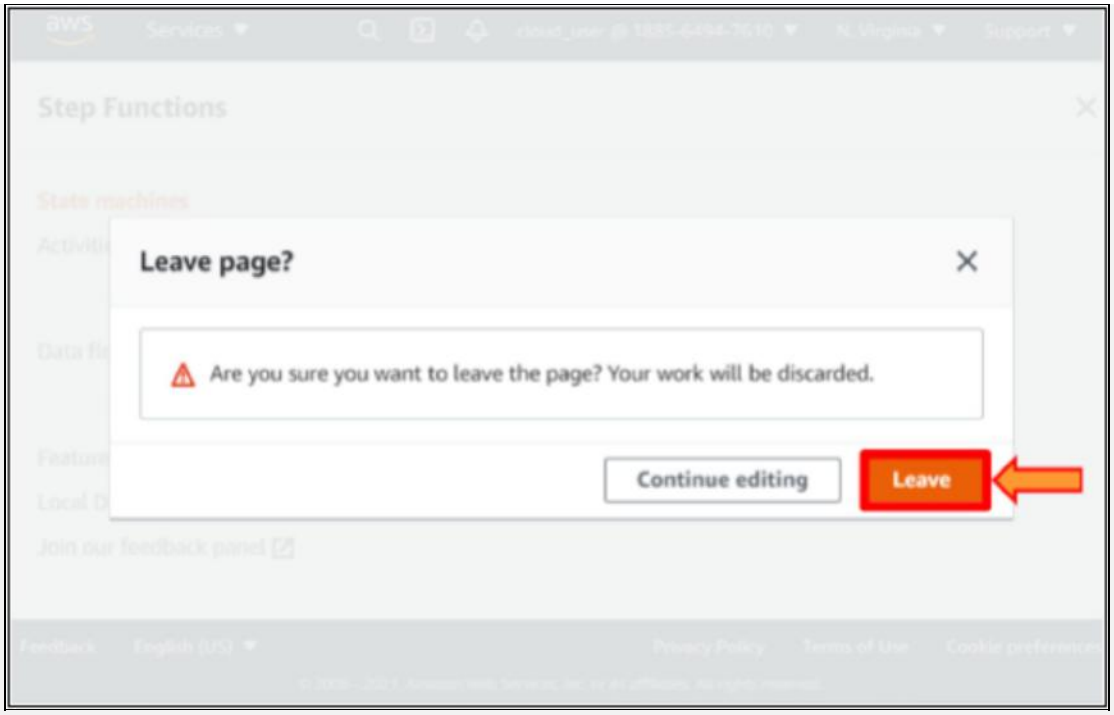
3. Click on the **Get started** button.



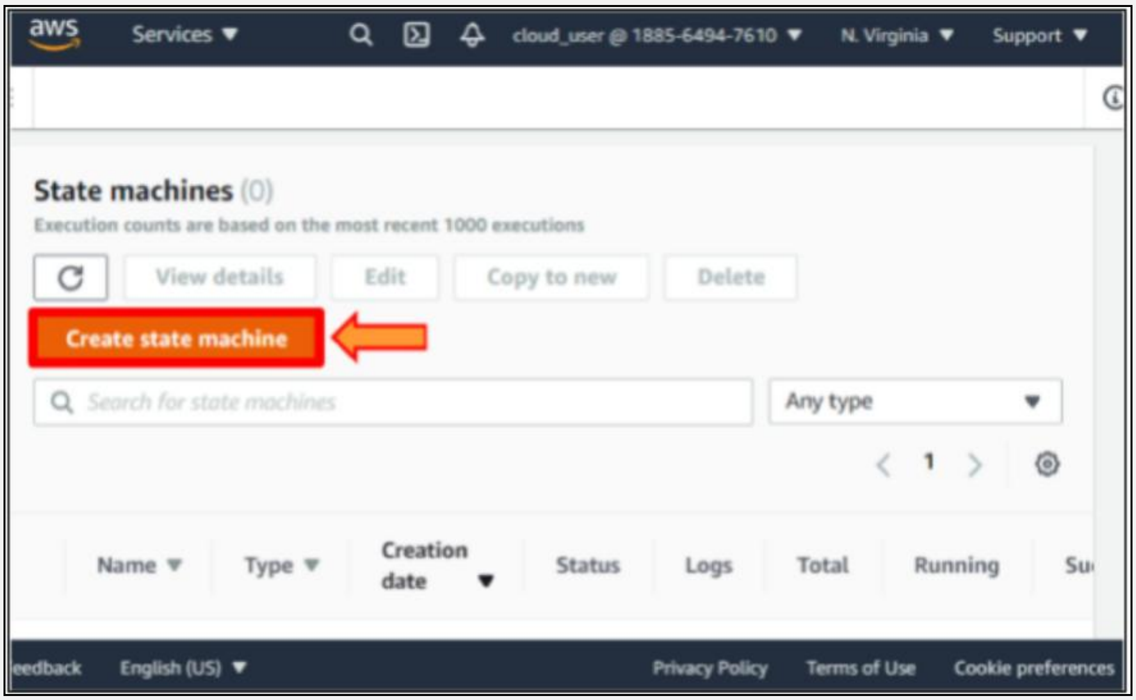
4. Click on **State machines** from the left-hand side menu.



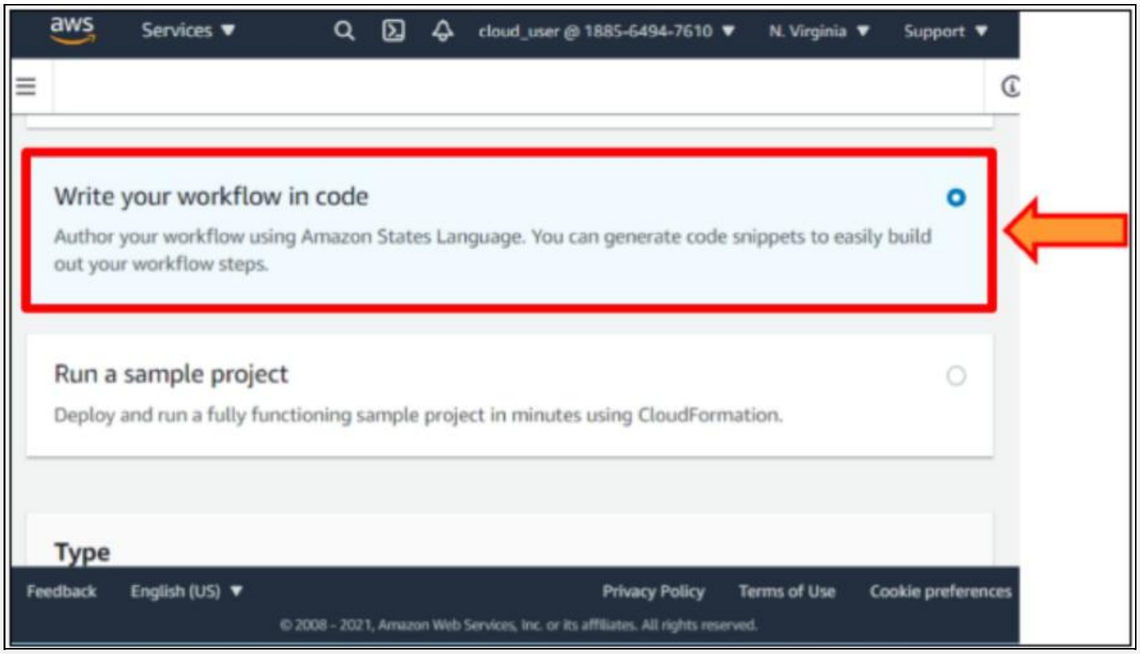
5. Click on the **Leave** button.



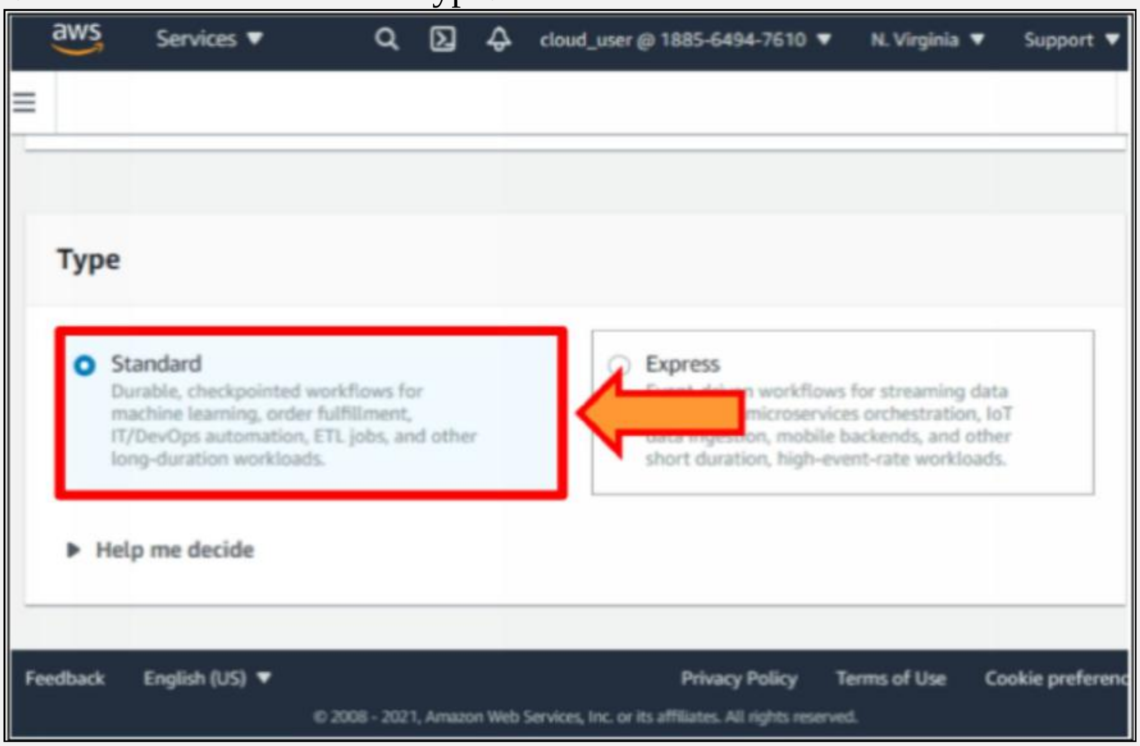
6. Click on the **Create state machine** button.



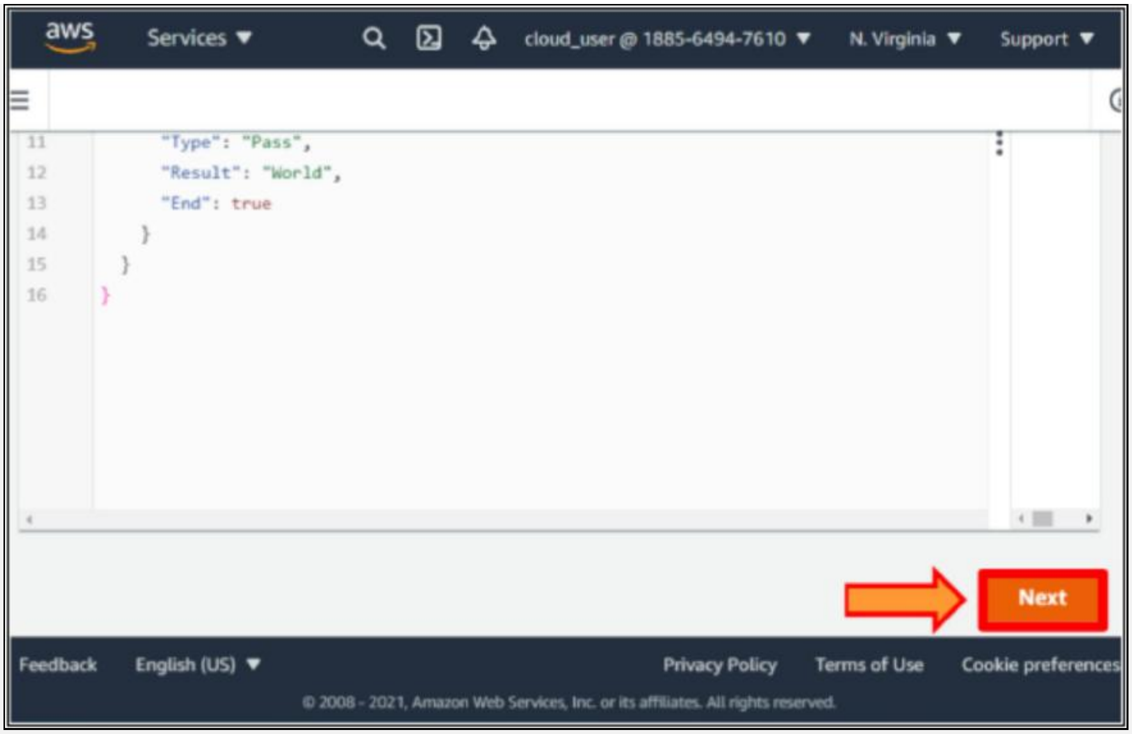
7. Select **Write your workflow in code**.



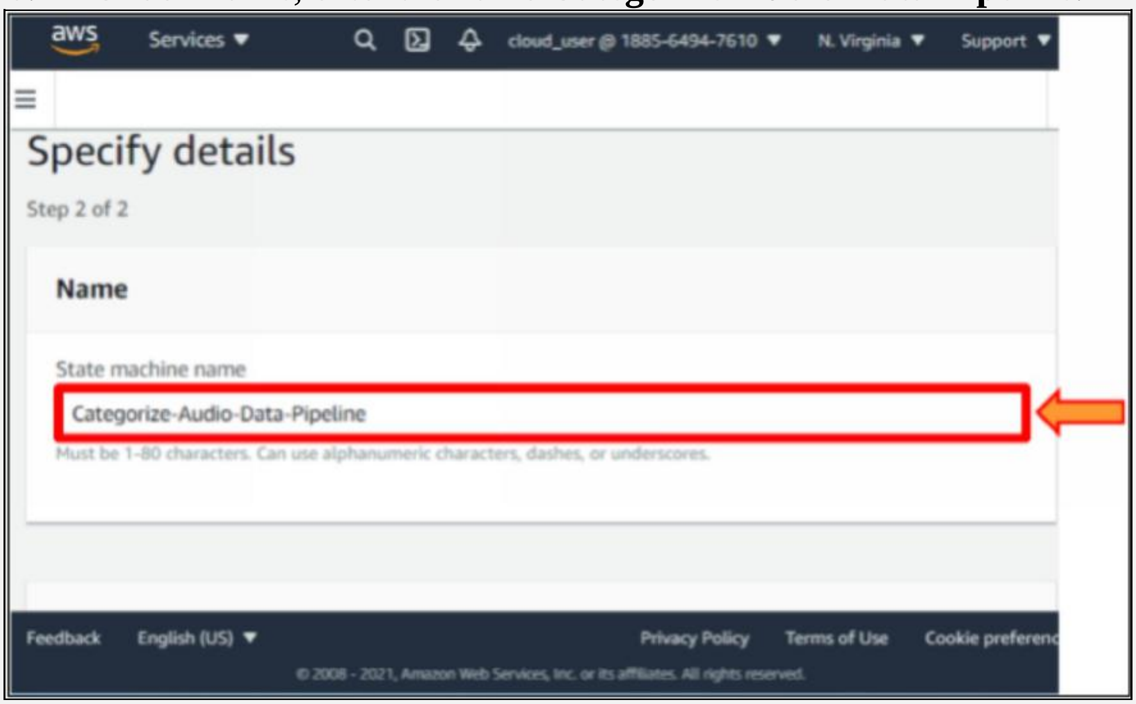
8. Select **Standard** under Type.



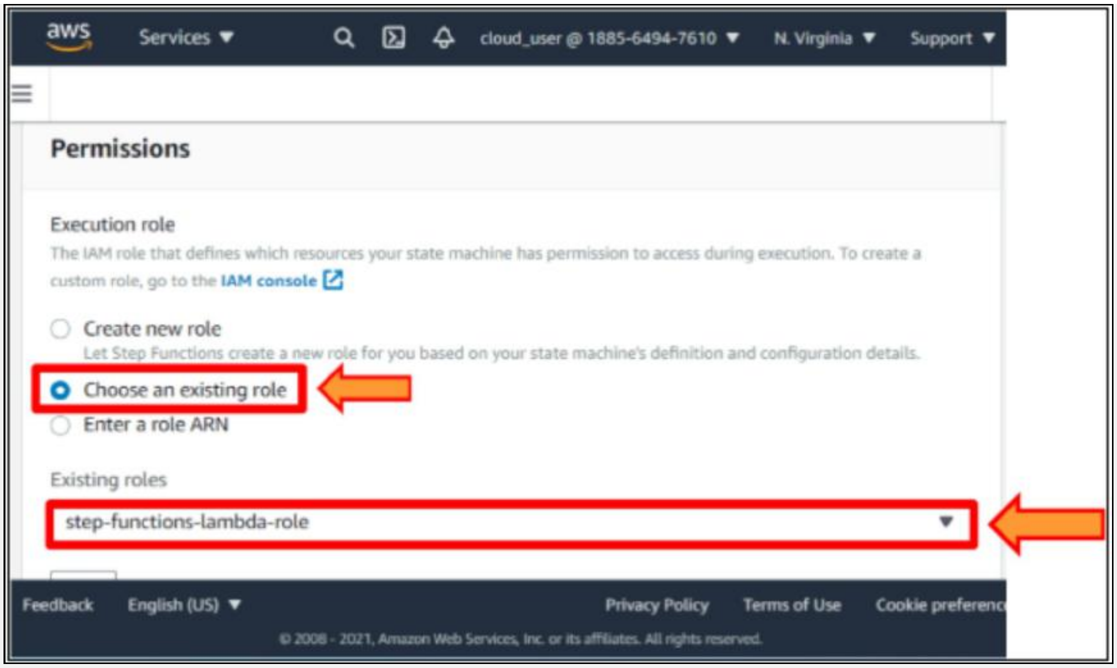
9. Click on the **Next** button.



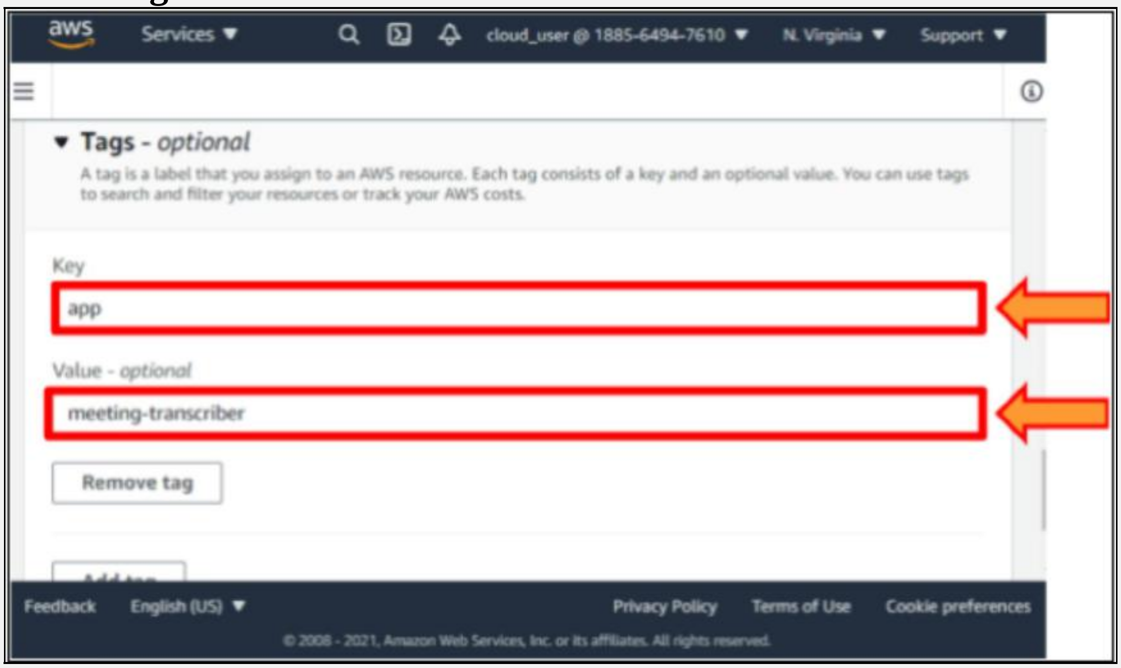
10. Under **Name**, enter the name **Categorize-Audio-Data-Pipeline**.



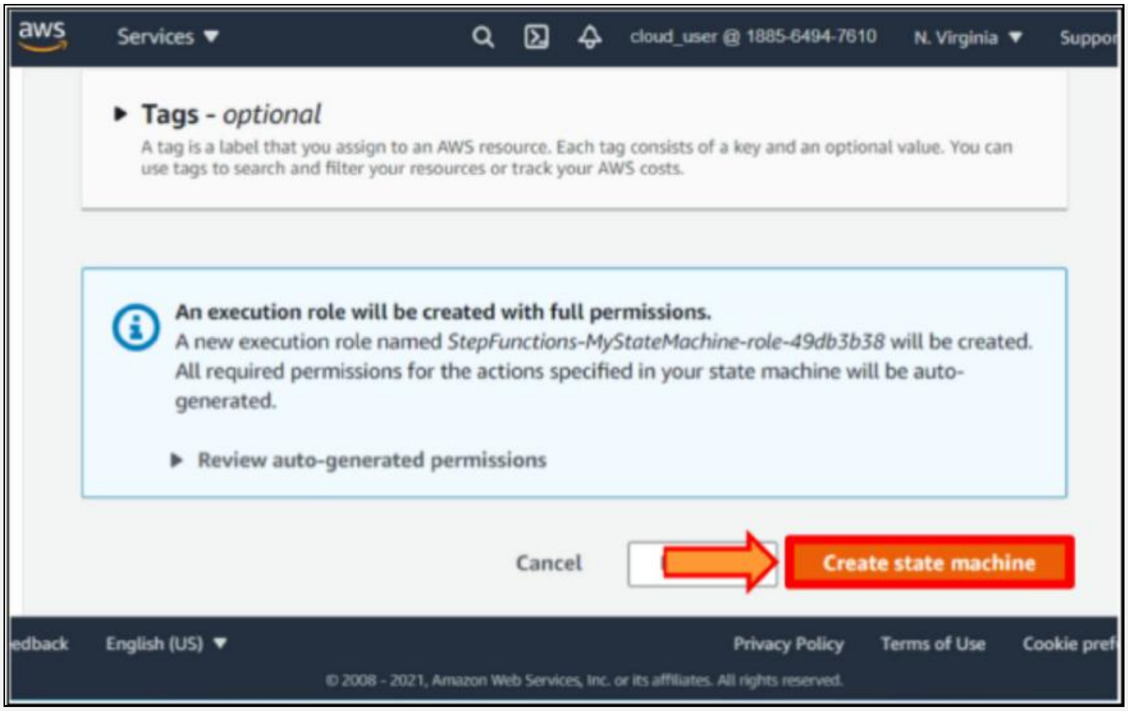
11. Click on **Choose an existing role**.



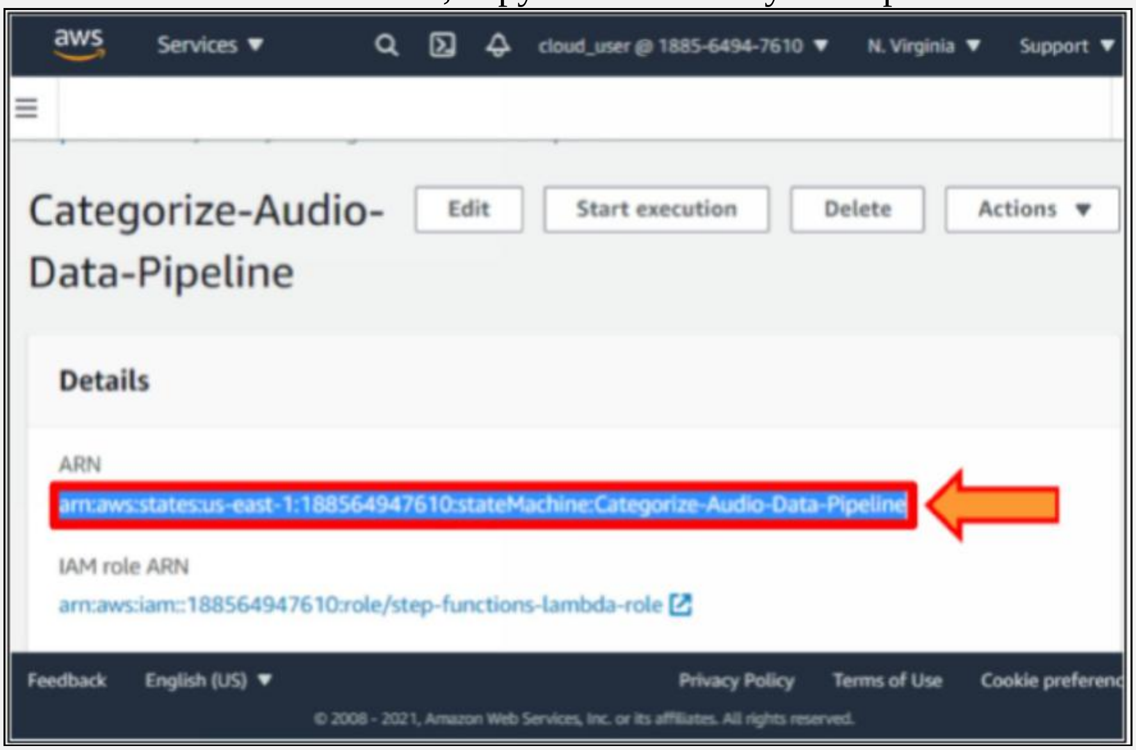
12. Under Tags - optional, create a tag by entering **app** under Key and **meeting-transcriber** under **Value**.



13. Click on the **Create state machine** button.

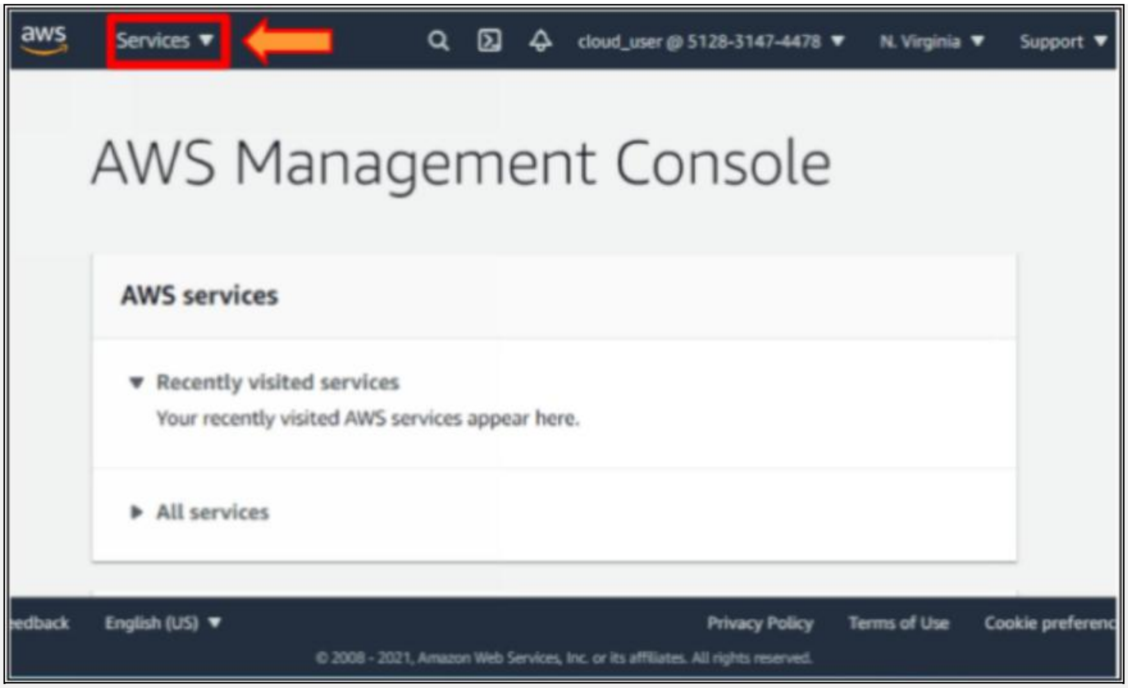


14. Under **ARN** in **Details**, copy the **ARN** into your clipboard.

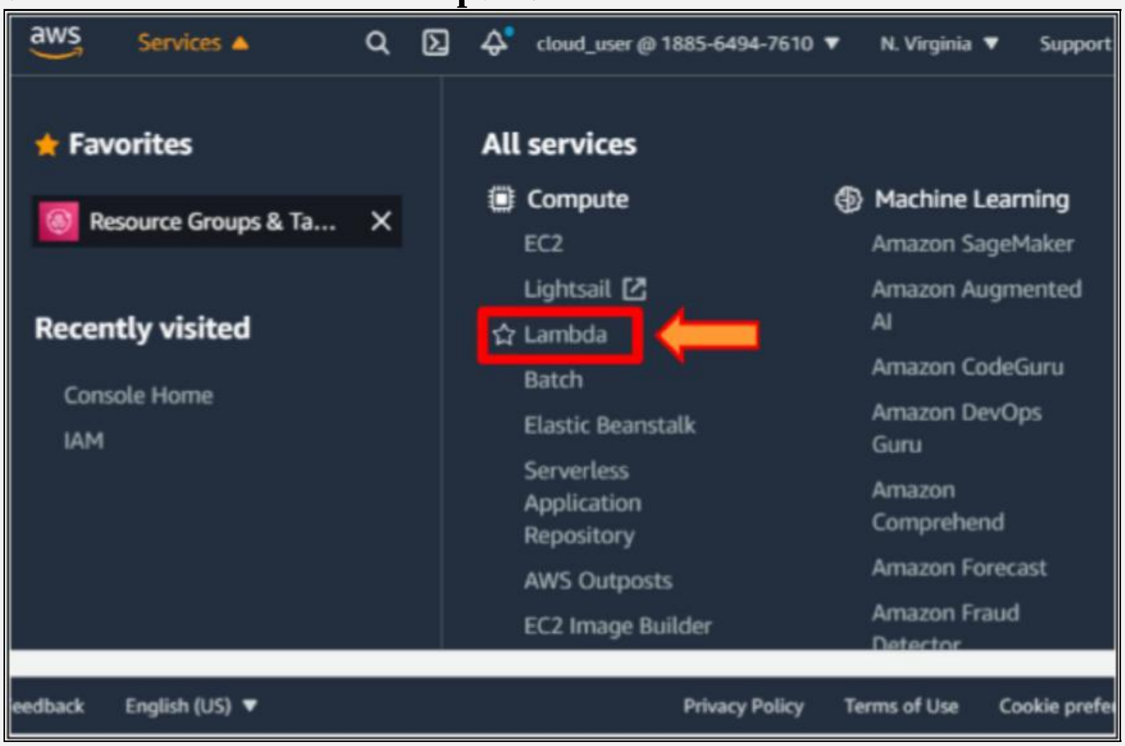


Step 3: Update Lambda to Work with Step Functions

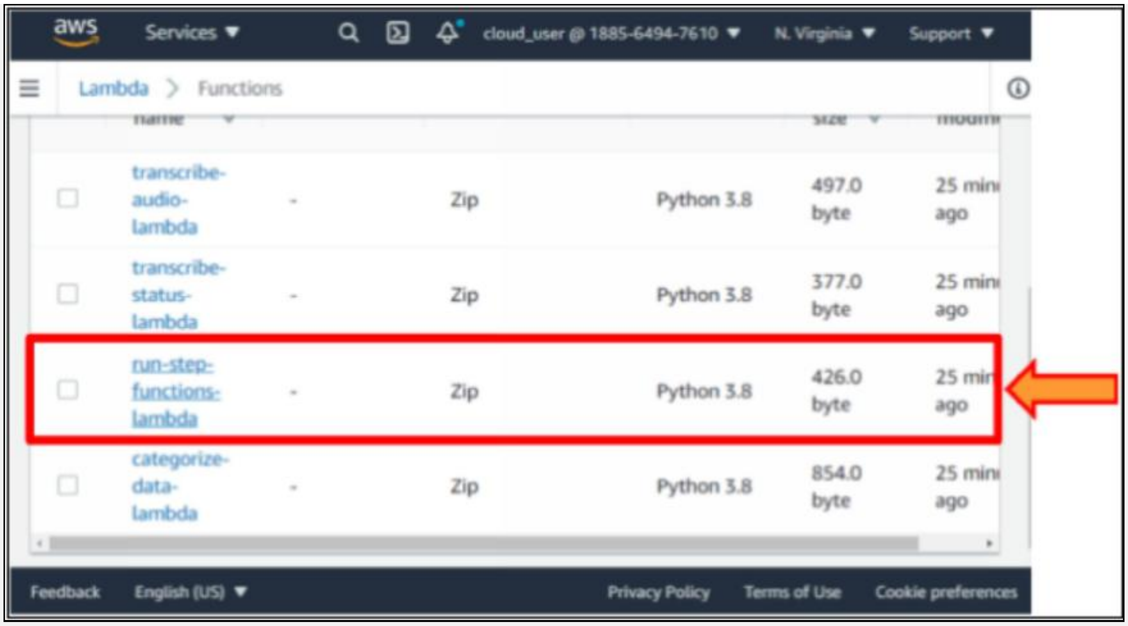
1. Click on **Services.**



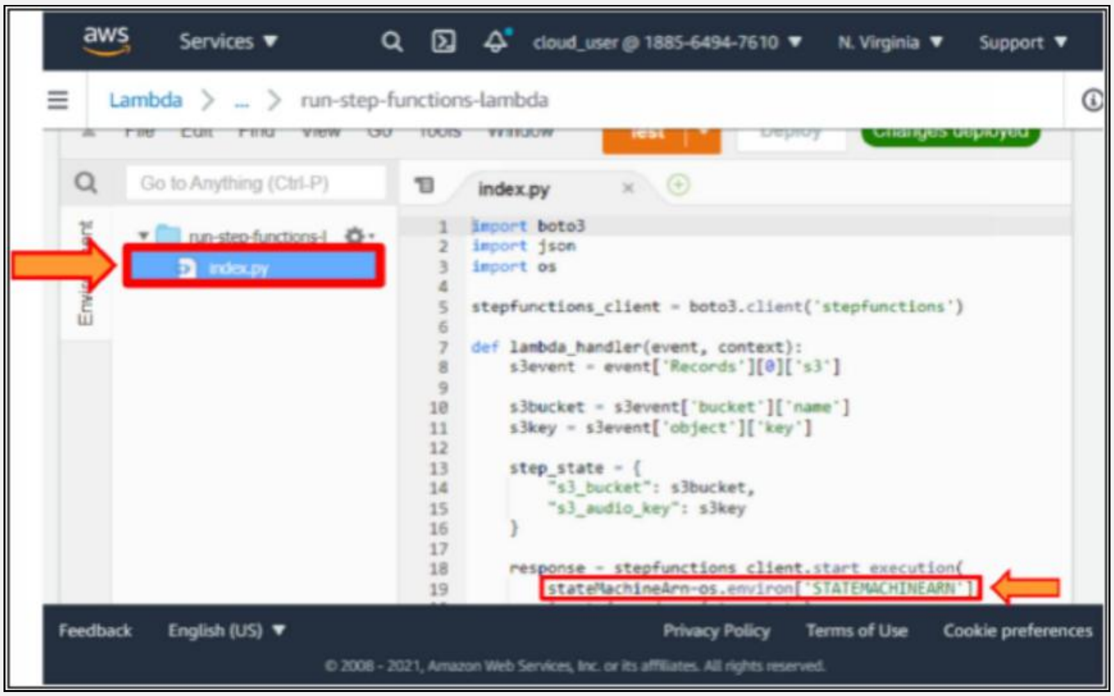
2. Select **Lambda** from **Compute.**



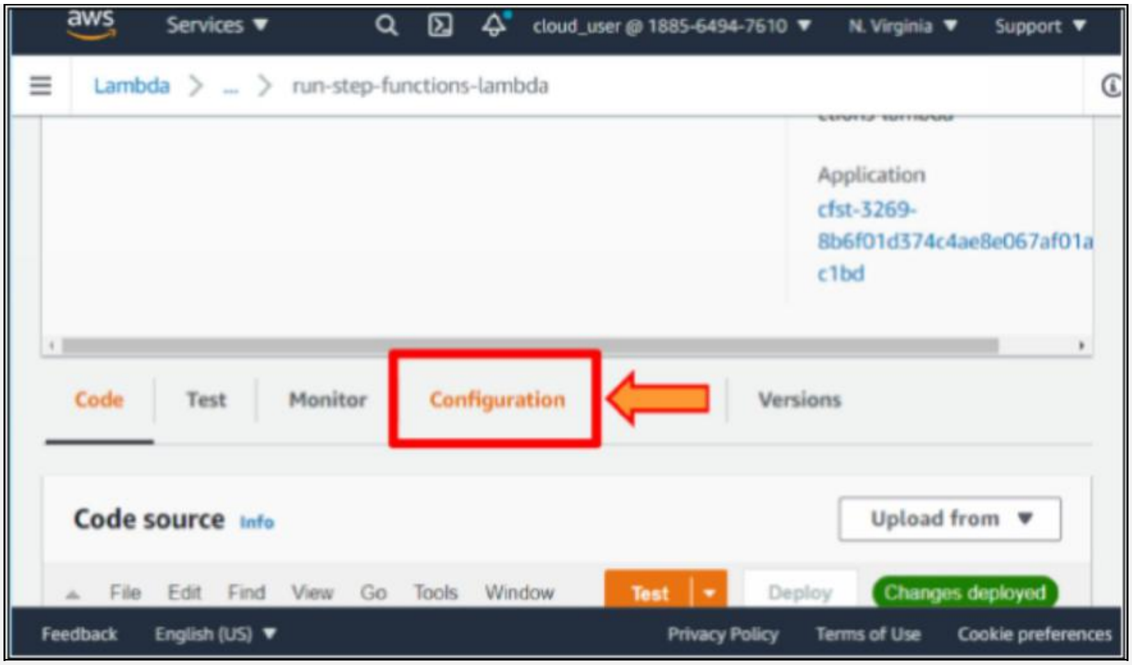
3. Click on **run-step-functions-lambda**.



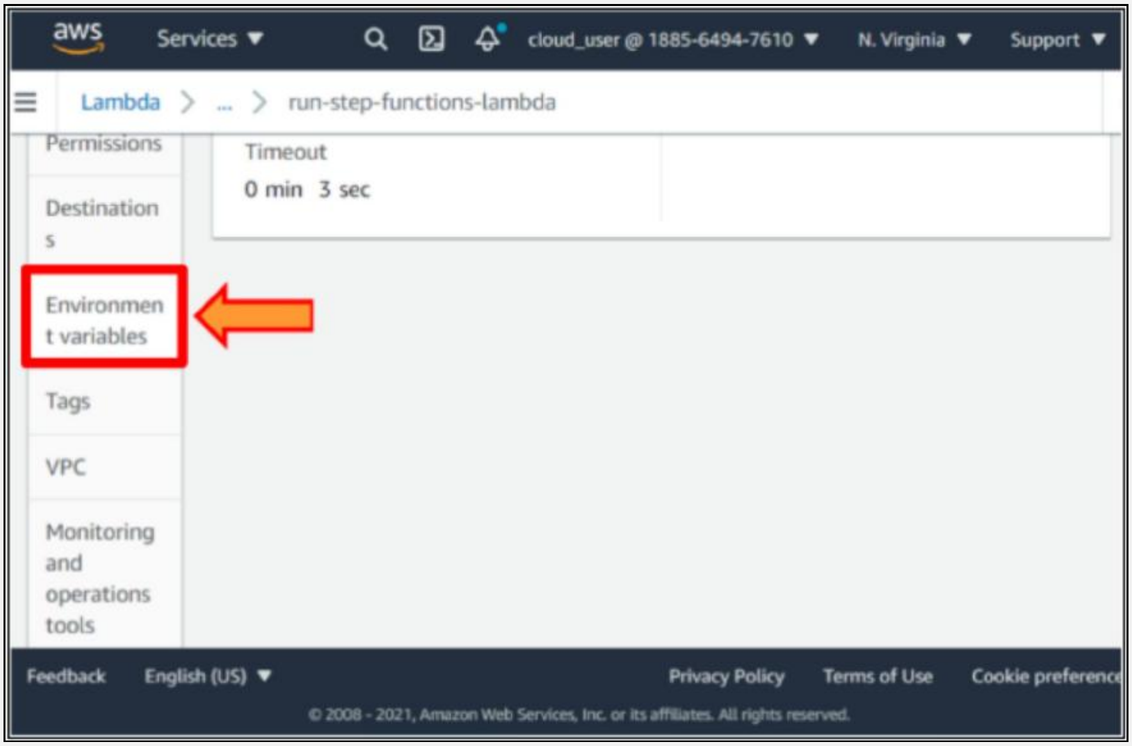
4. Under Function code, click on the **index.py**.
5. In the Python file, note that it contains a placeholder at the end called **STATEMACHINEARN**.



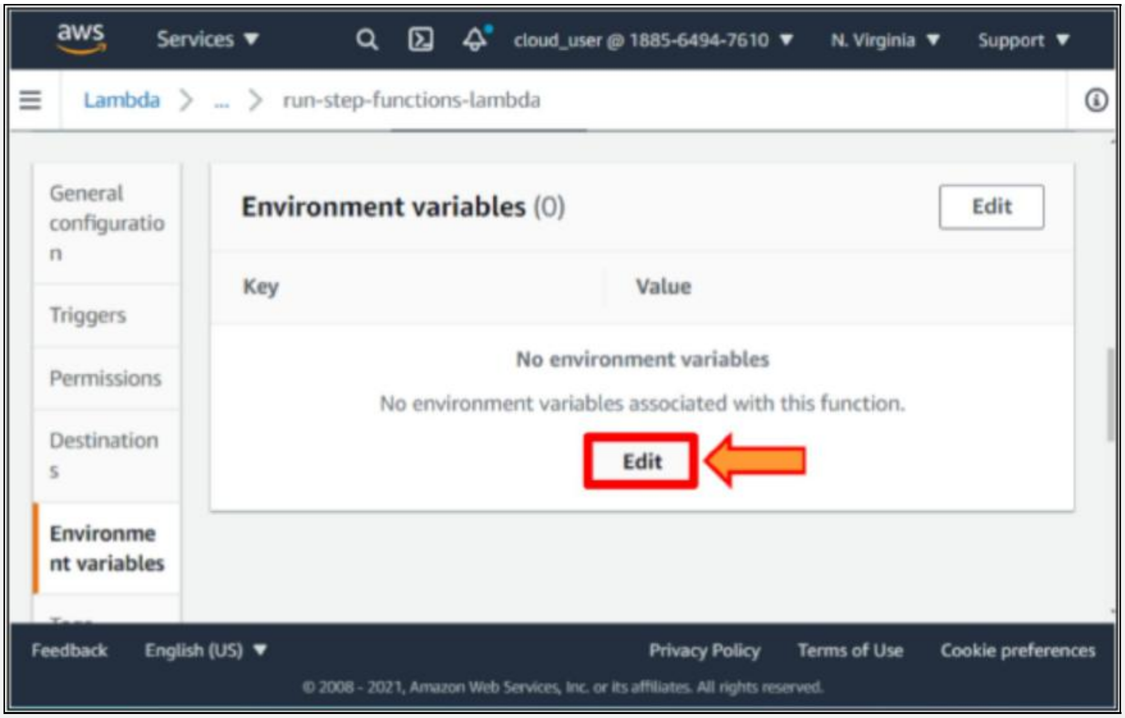
6. Click on the **Configuration** tab.



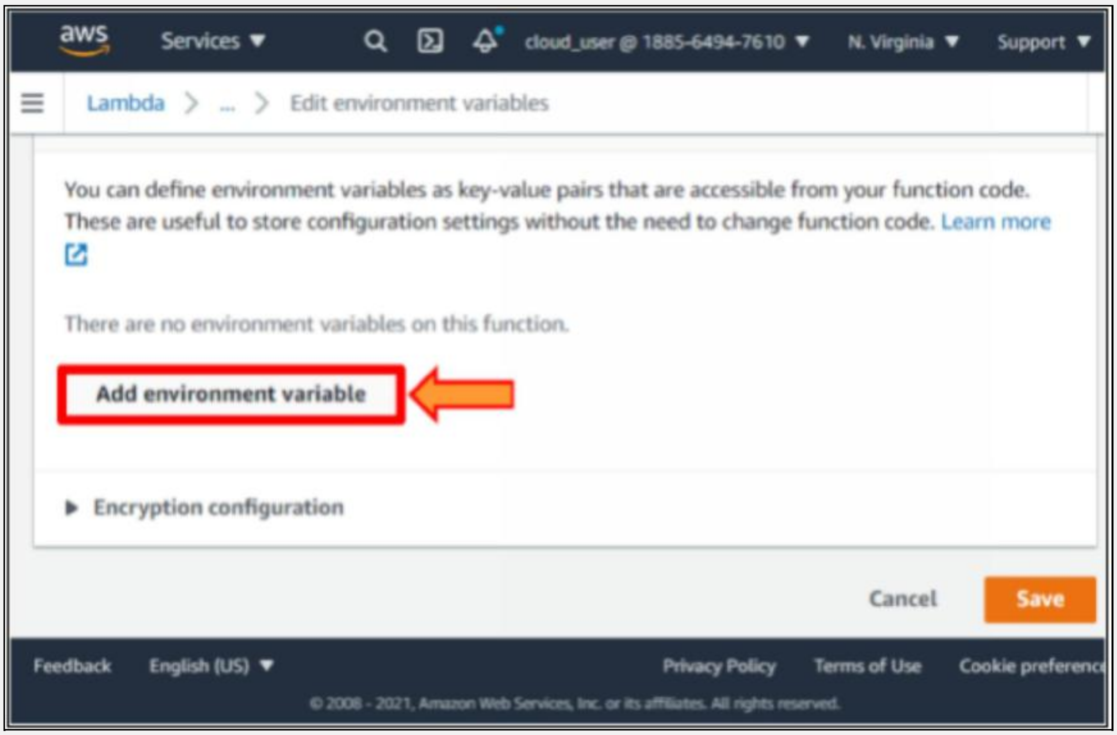
7. Click on **Environment variables.**



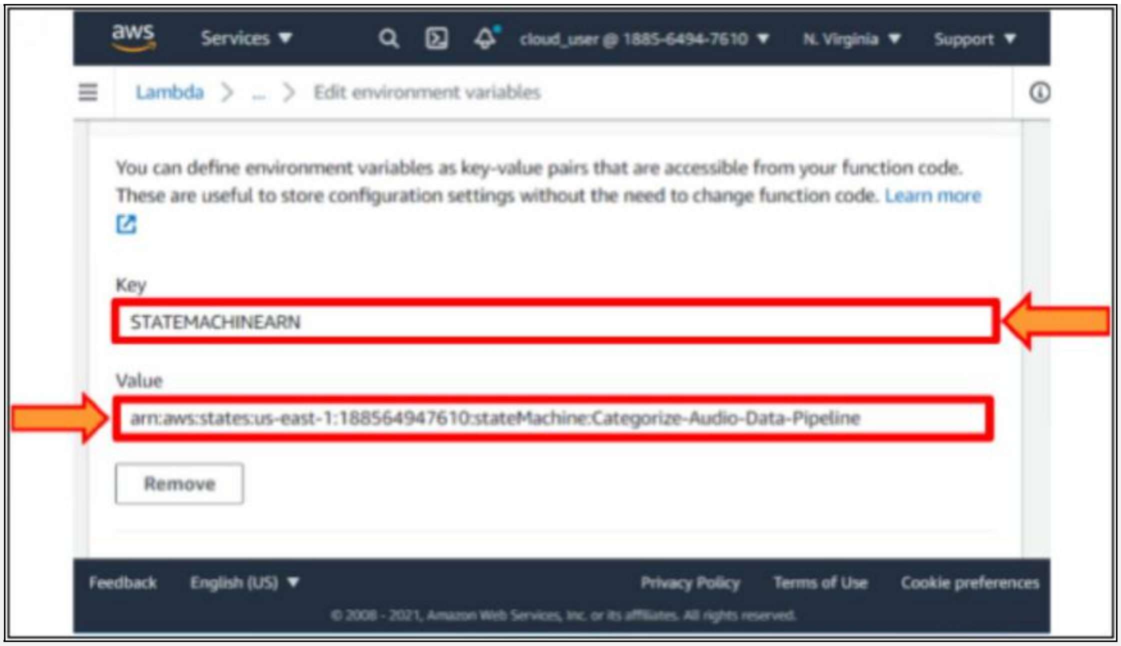
8. Click on the **Edit** button.



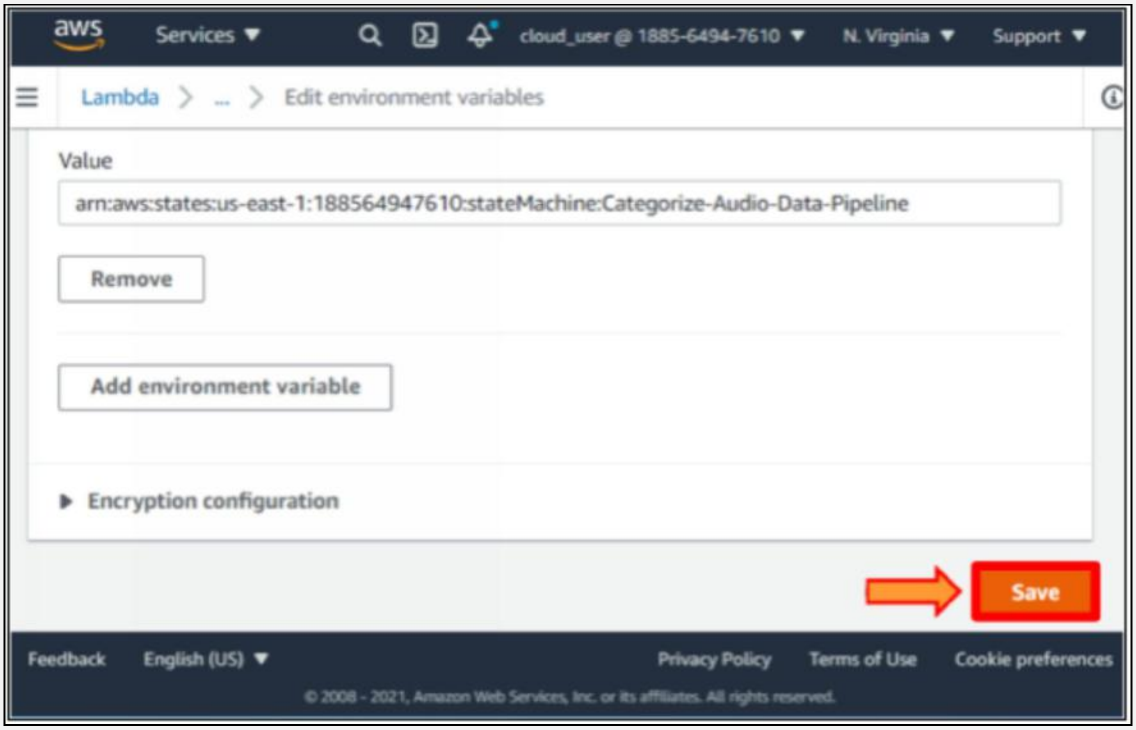
9. Click on the **Add environment variable** button.



10. Add an environment variable by entering **STATEMACHINEARN** under Key and pasting your state machine's ARN under **Value**.

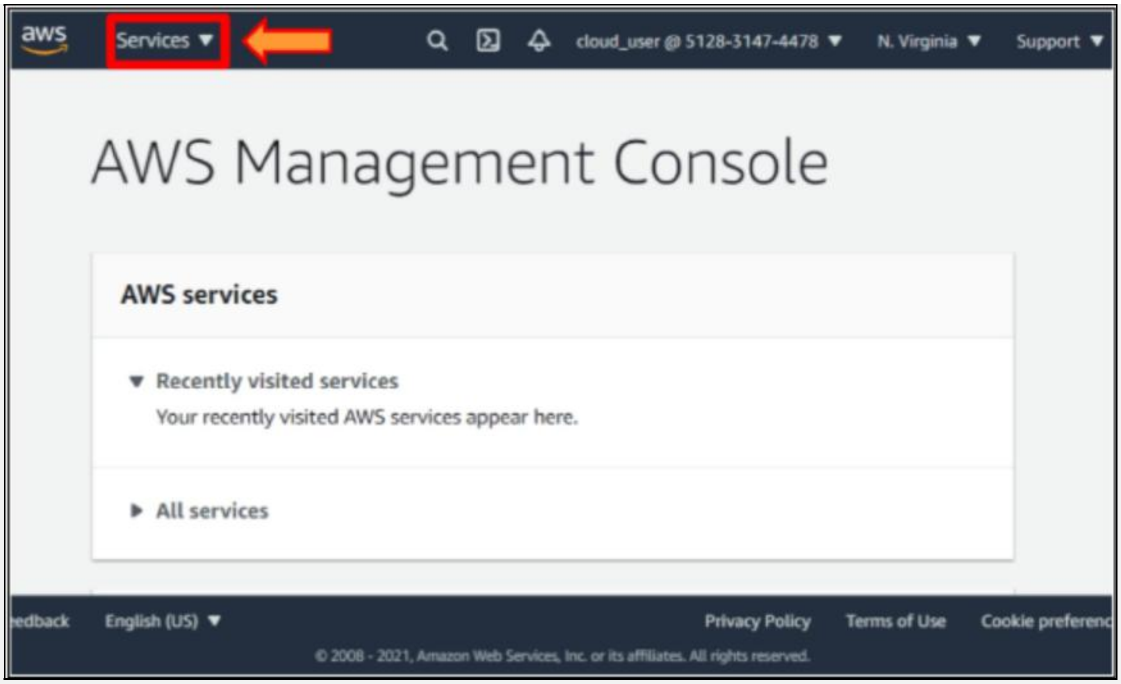


11. Click on the **Save** button.

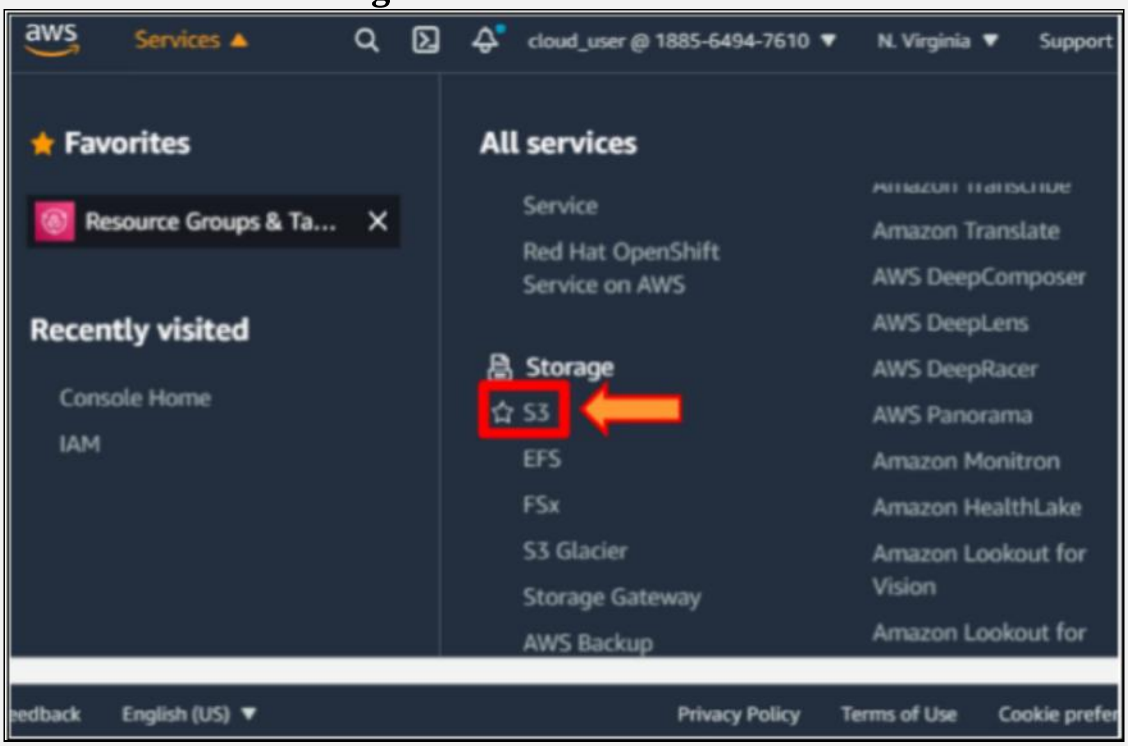


Step 4: Update S3 to Work with Step Functions

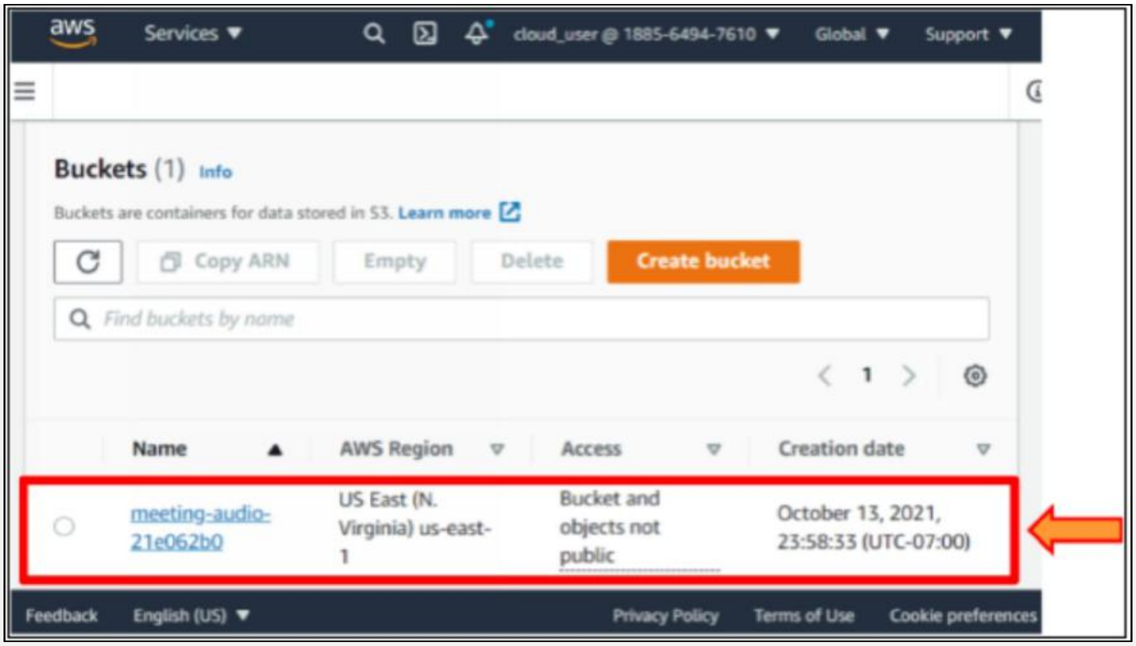
1. Click on **Services**.



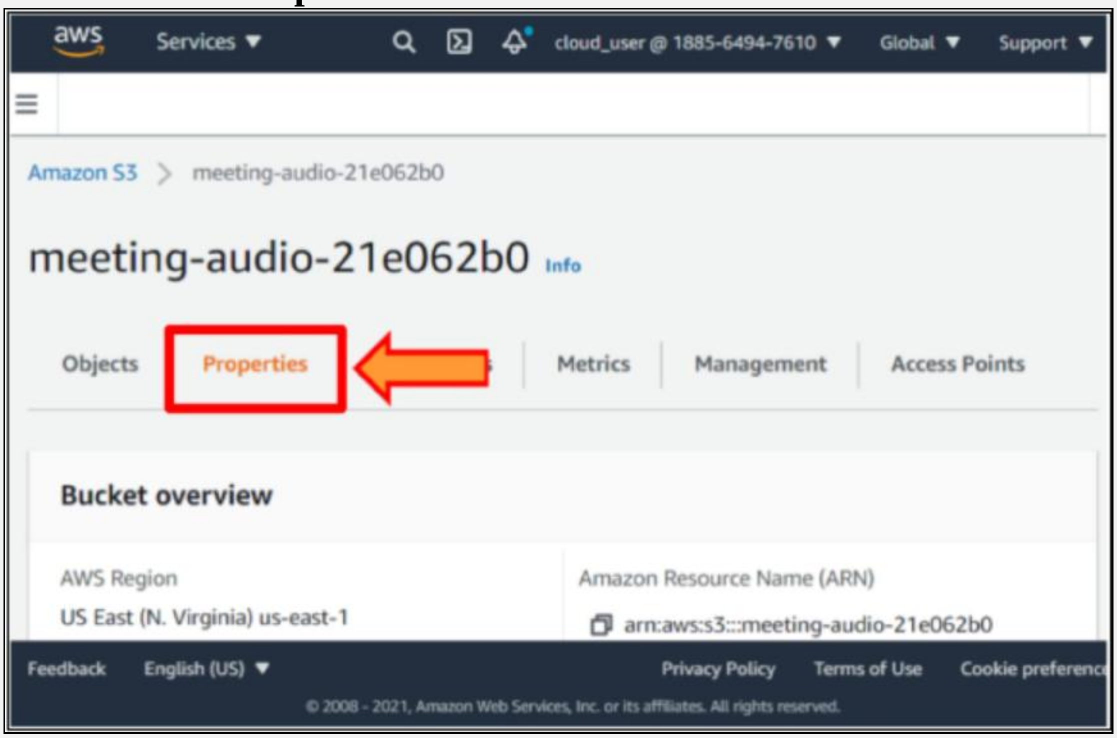
2. Select **S3** from **Storage**.



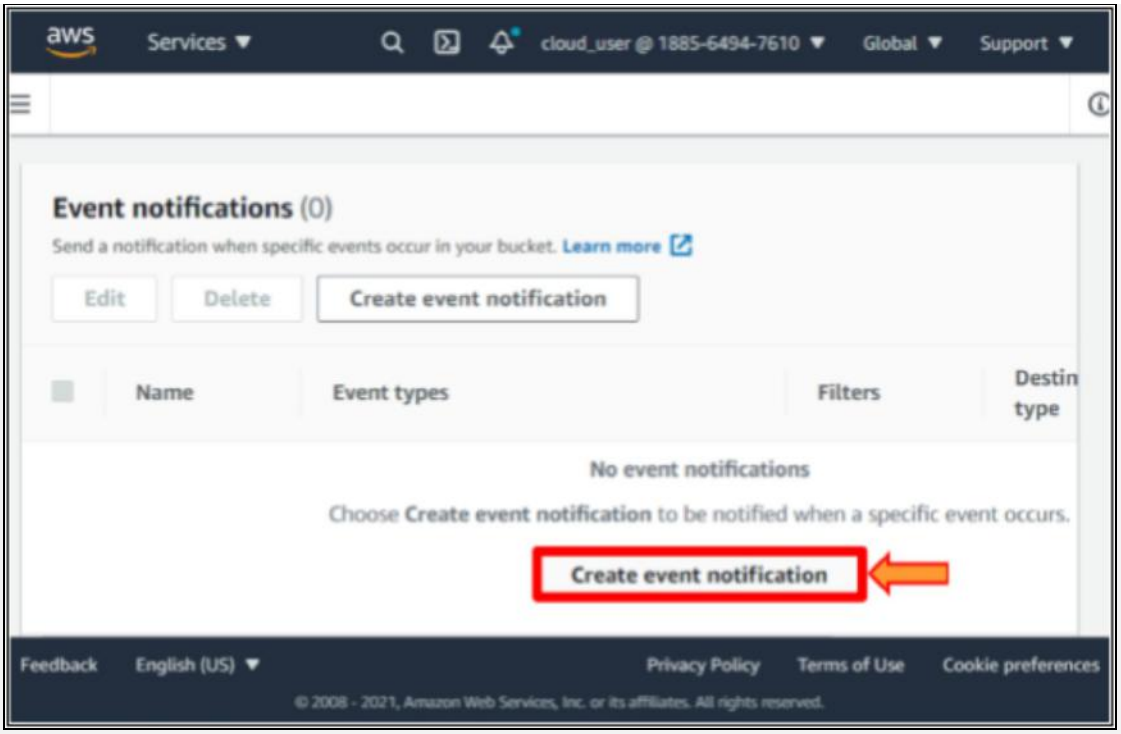
3. Click on the created bucket.



4. Click on the **Properties** tab.

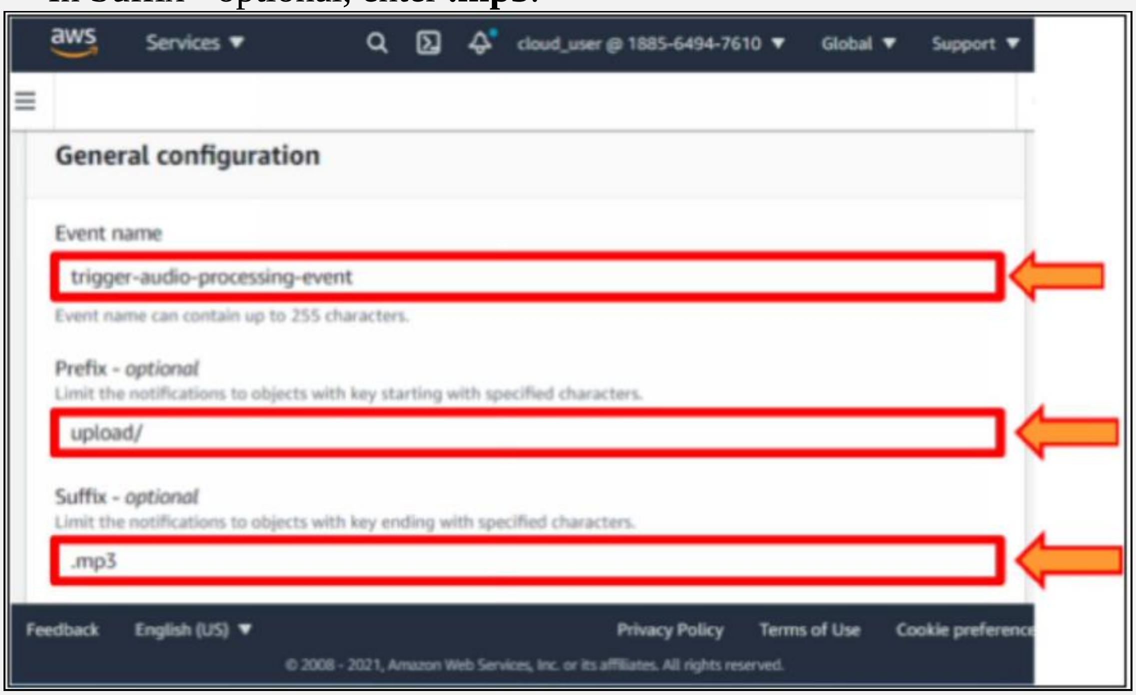


5. Click on the **Create event notification** button.

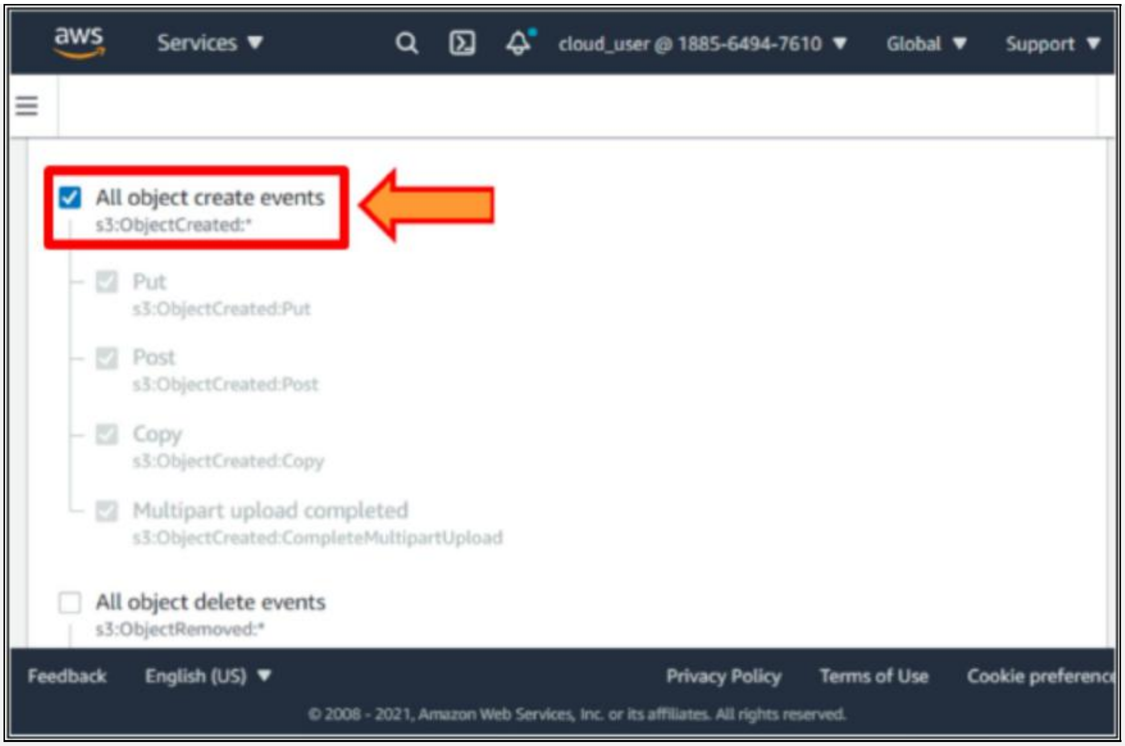


6. Under General configuration, set the following values:

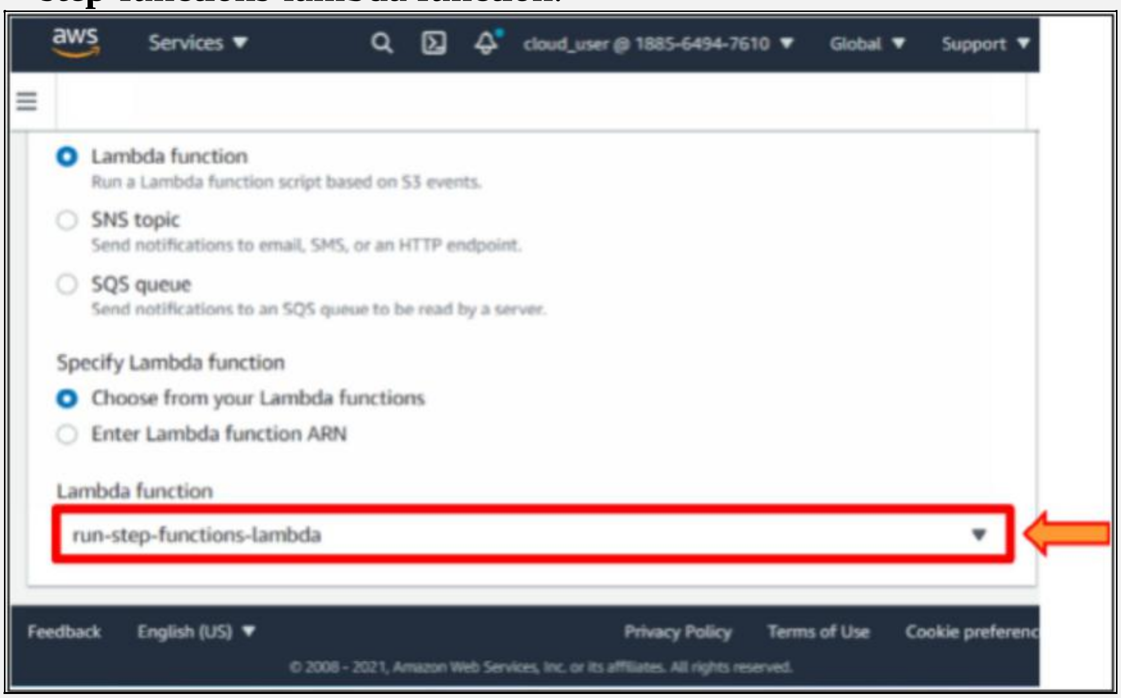
- In the **Event** name, enter the name **trigger-audio-processing-event**.
- In Prefix - optional, enter **upload/**.
- In Suffix - optional, enter **.mp3**.



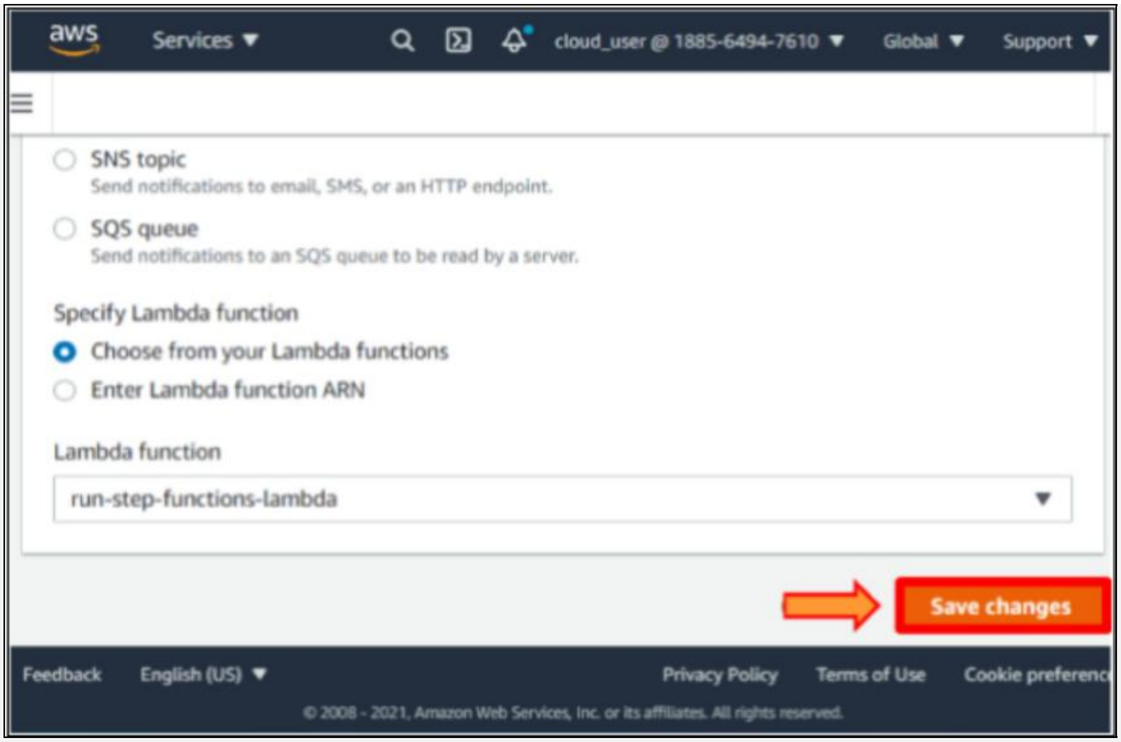
7. Select the checkbox next to **All object create events**.



8. In the dropdown menu under the Lambda function, select the **run-step-functions-lambda function**.

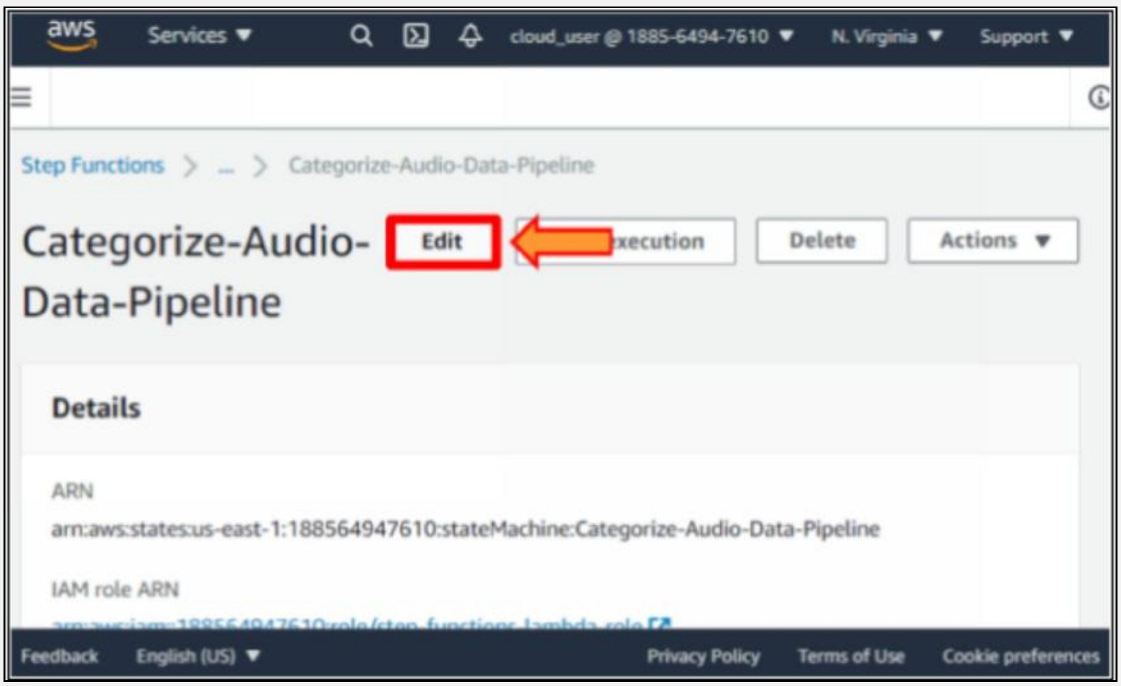


9. Click on the **Save changes** button.

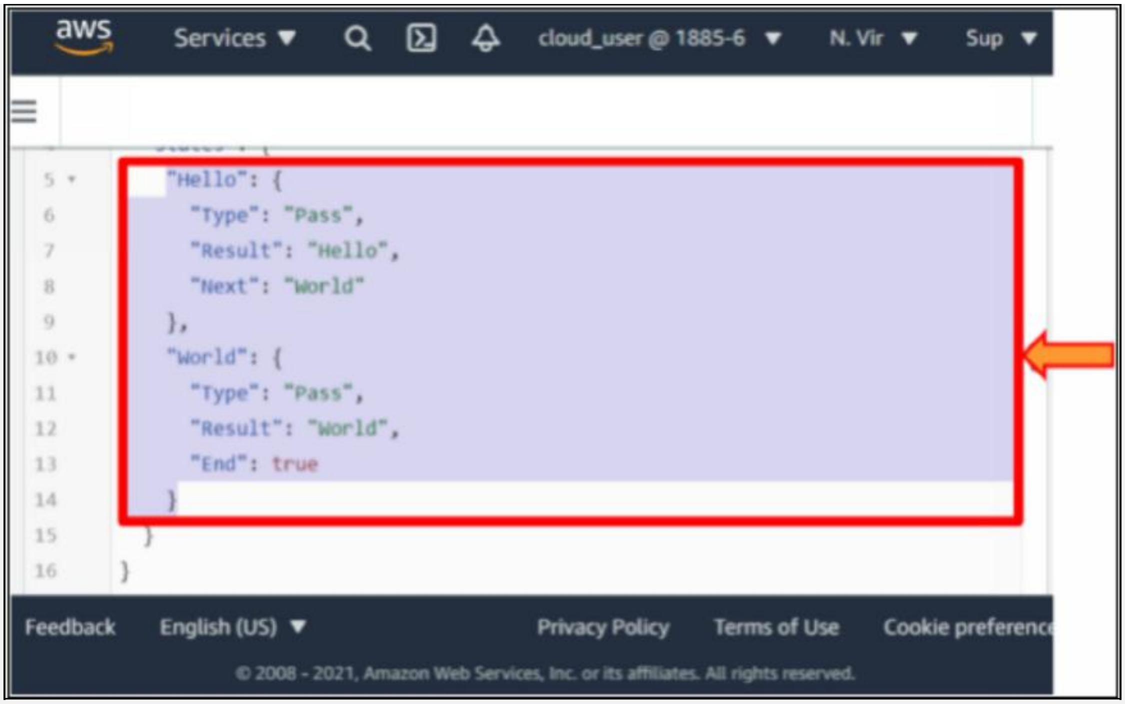


Step 5: Create Step Function Flow

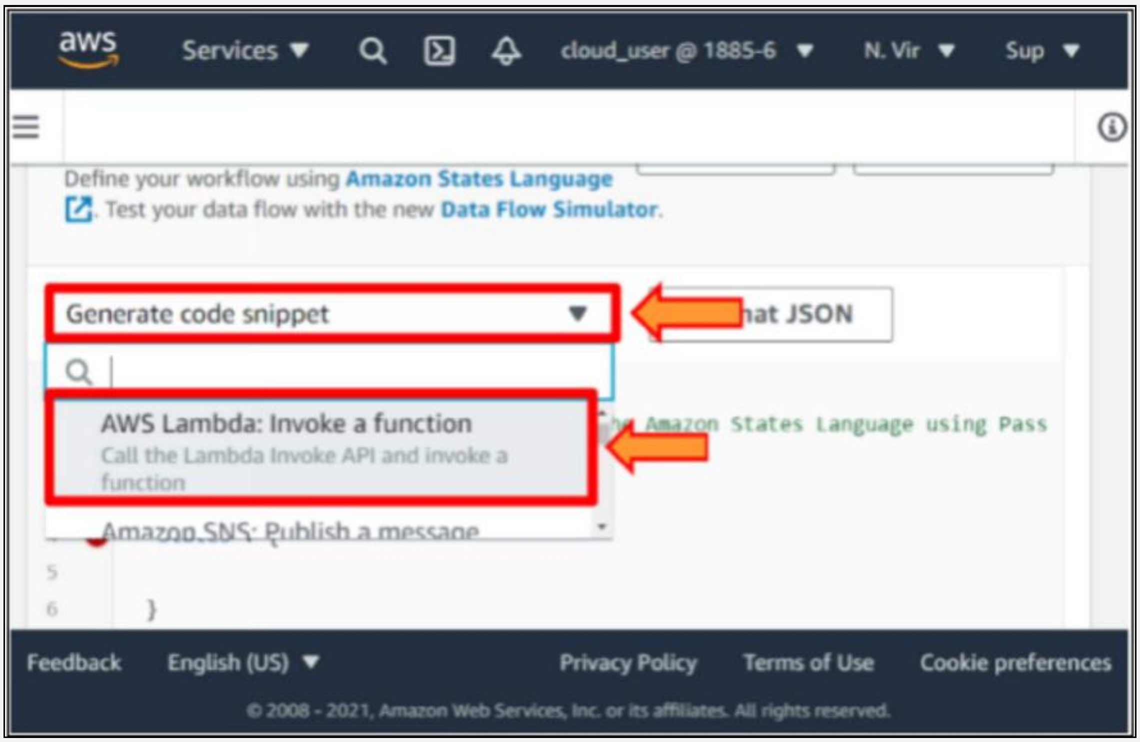
1. Go back to the **Step Function** dashboard.
2. Click on the **Edit** button.



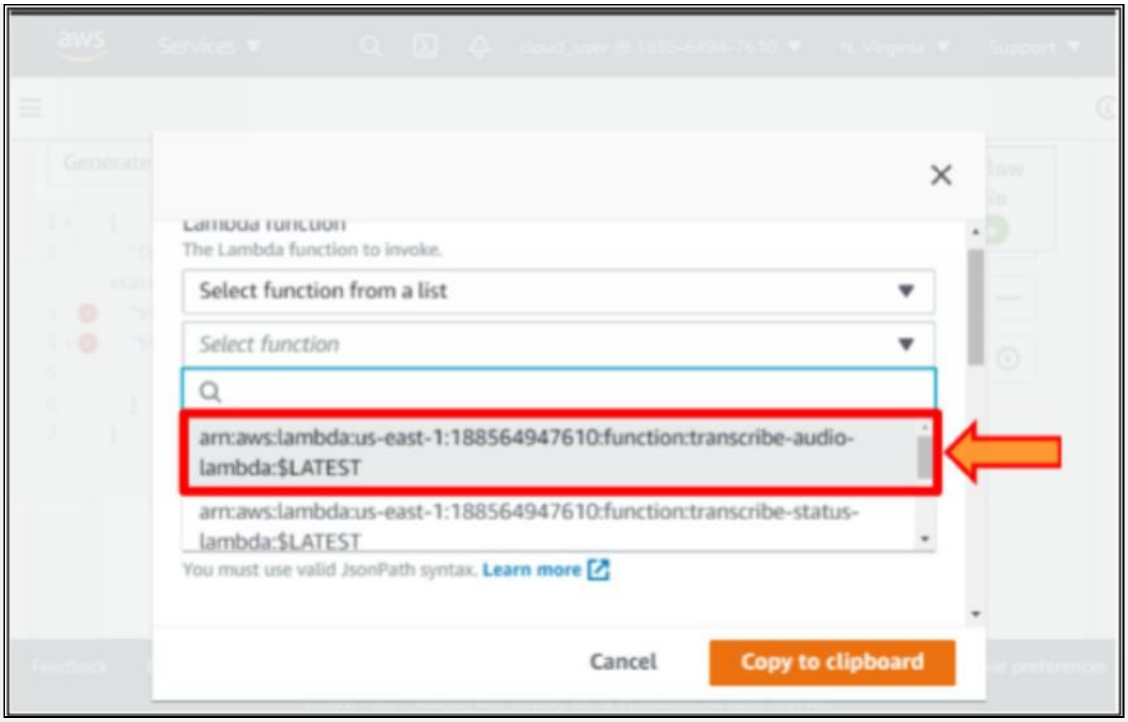
3. Create a Transcribe Job.
4. From the code snippet, delete the ‘**Hello**’ ‘**World**’ states after "**States**": {.



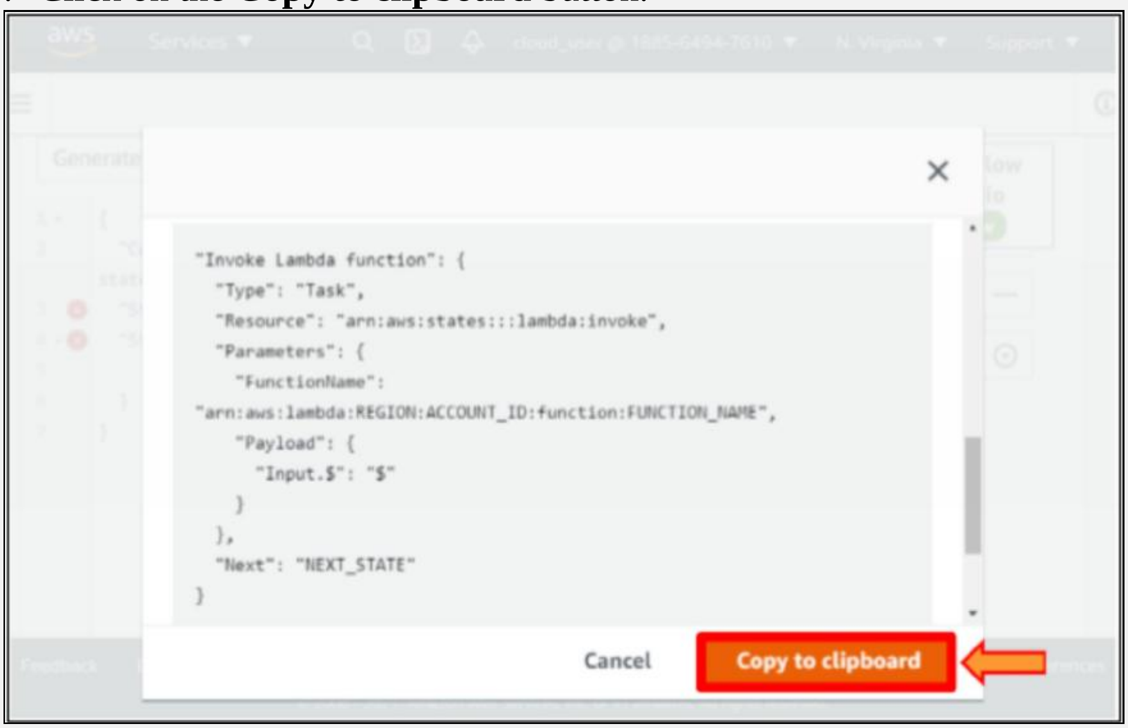
5. In the **Generate code snippet** dropdown, select the **AWS Lambda: Invoke a function.**



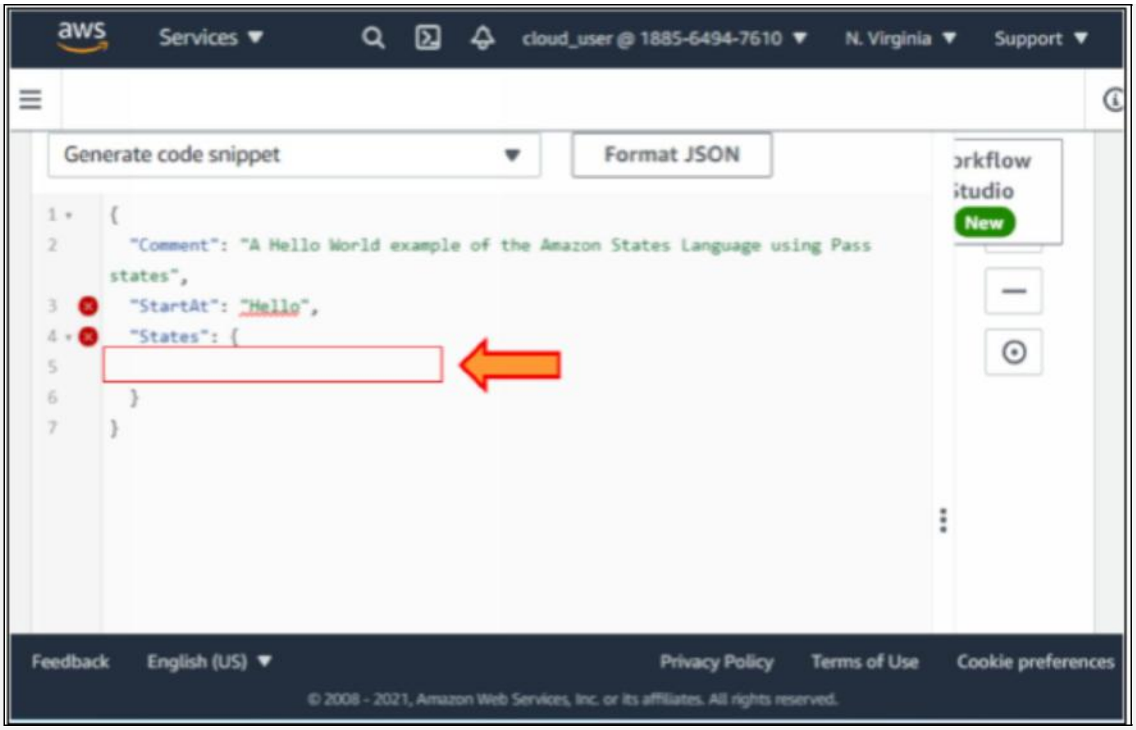
6. Select the **transcribe-audio-lambda** function.



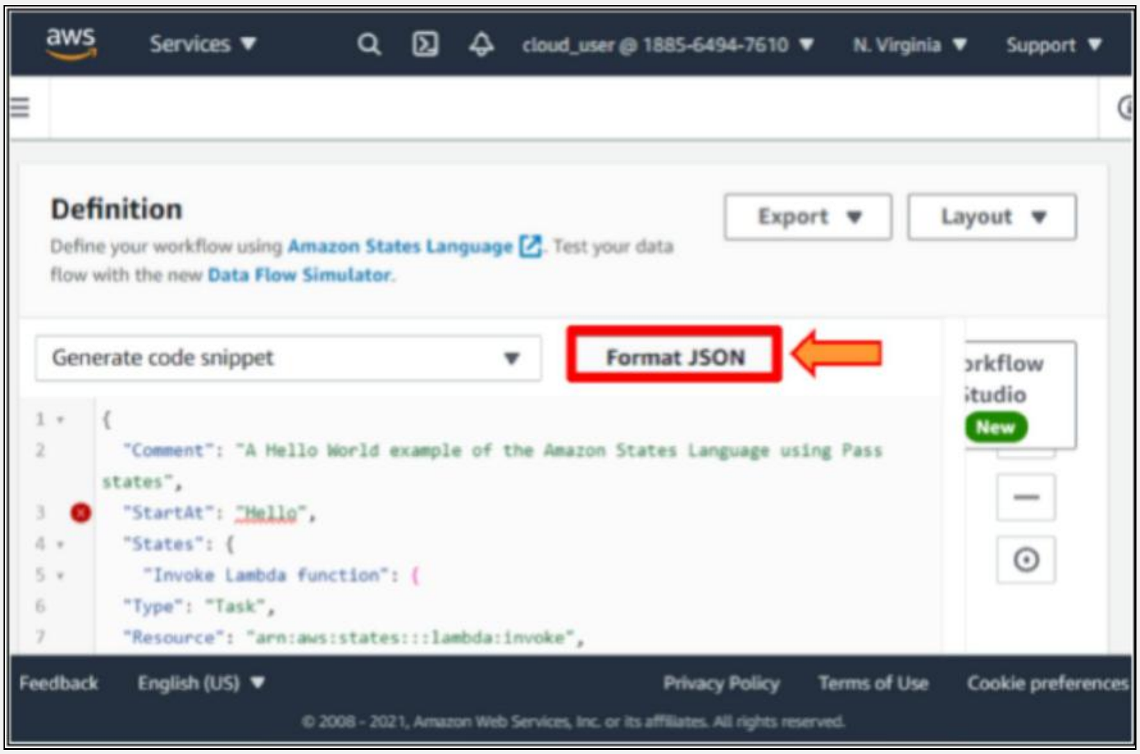
7. Click on the **Copy to clipboard** button.



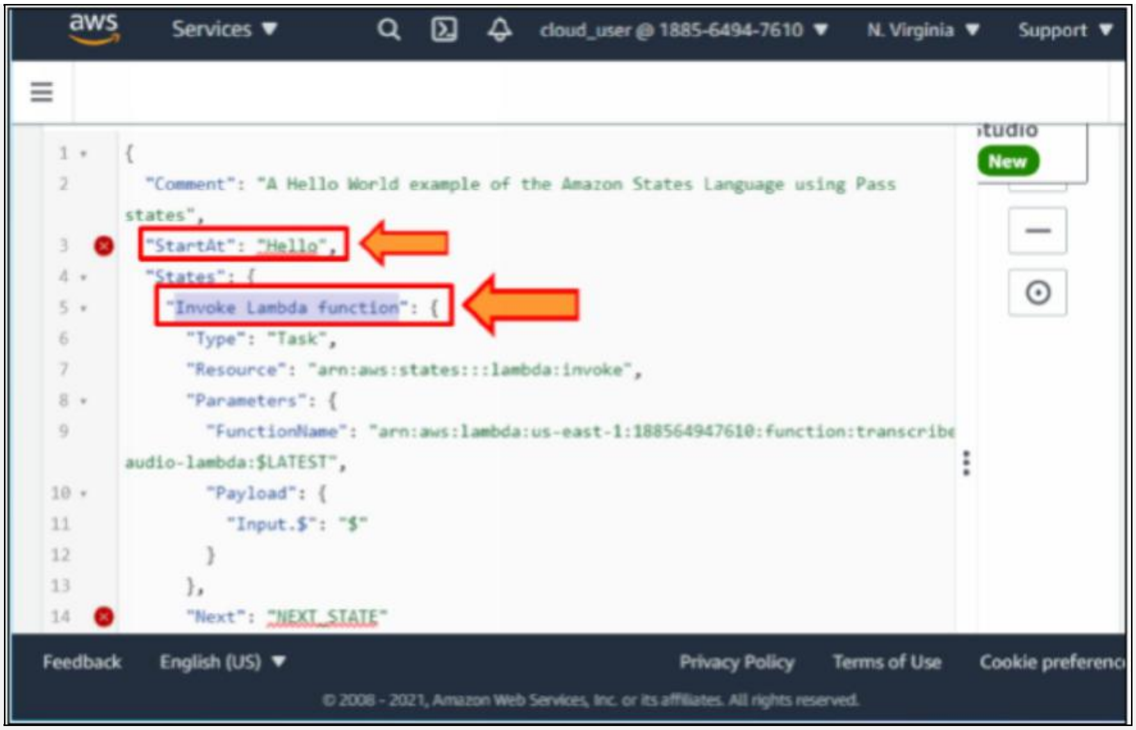
8. Paste the code into the code snippet after "**States**": {.



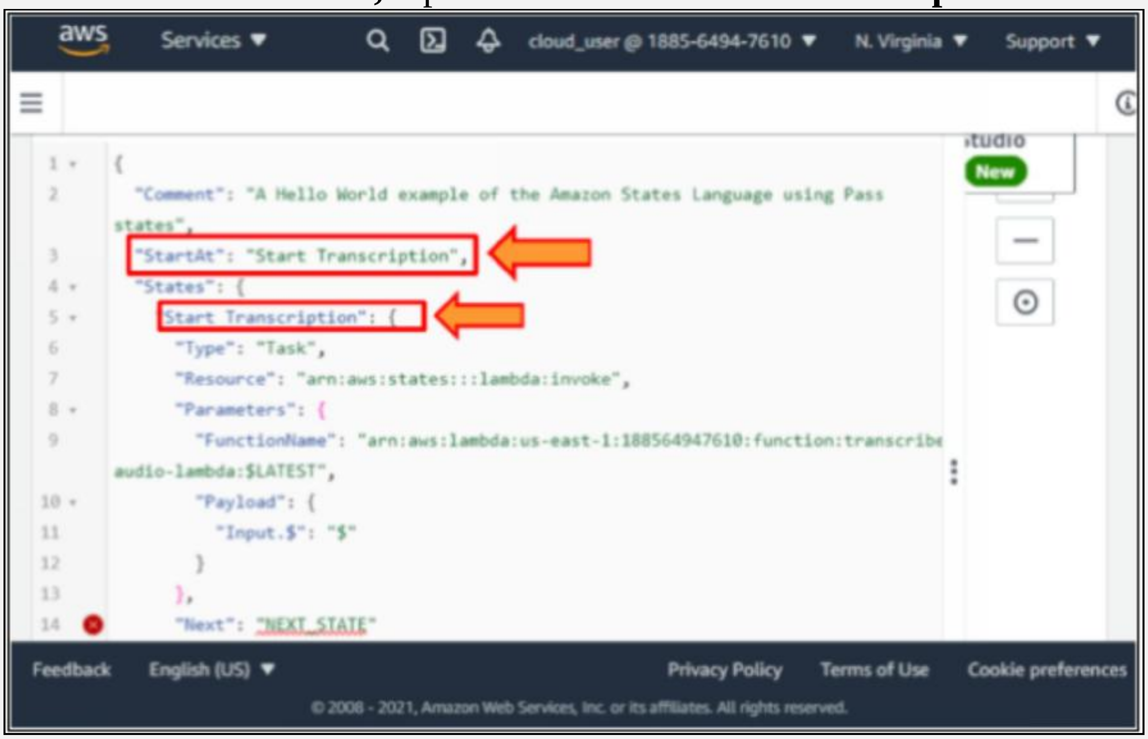
9. Click on the **Format JSON** button.



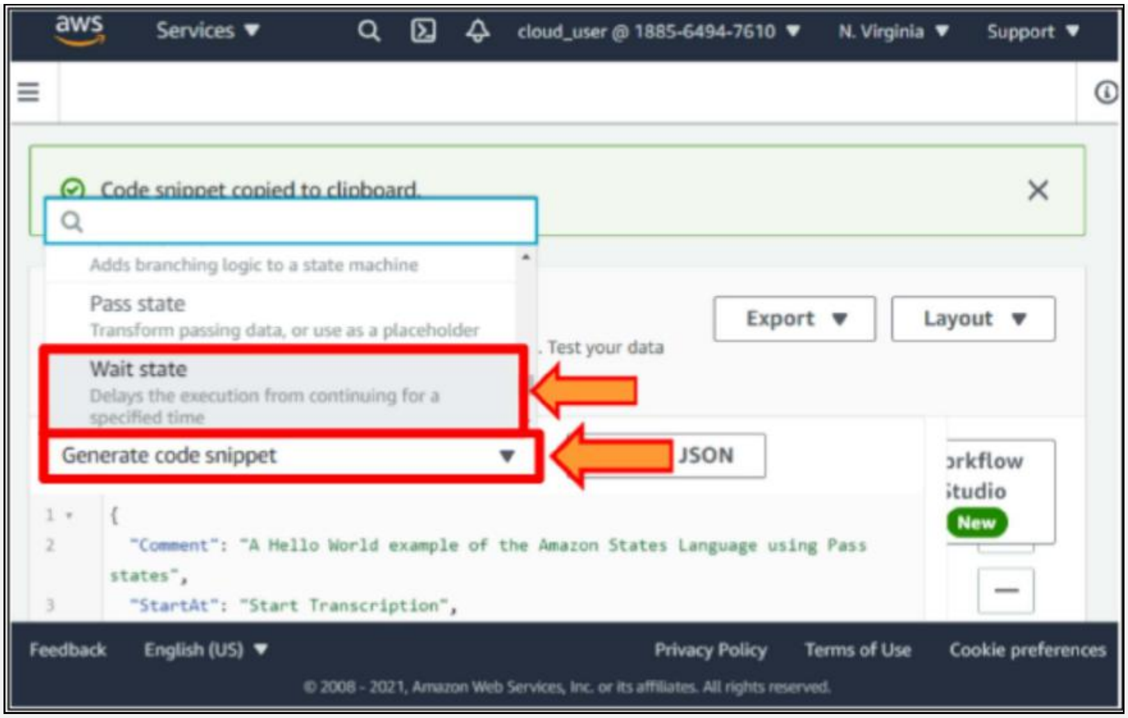
10. Rename **Invoke lambda function** to **Start Transcription.**



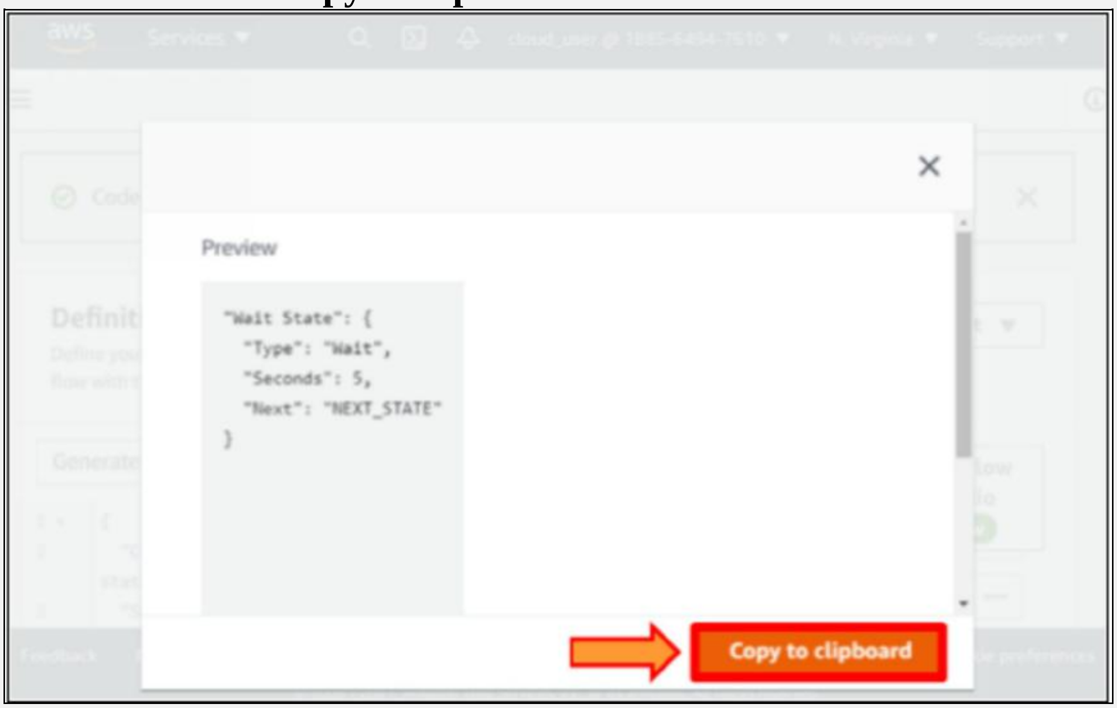
11. At **StartAt: Hello,** replace Hello with**: Start Transcription.**



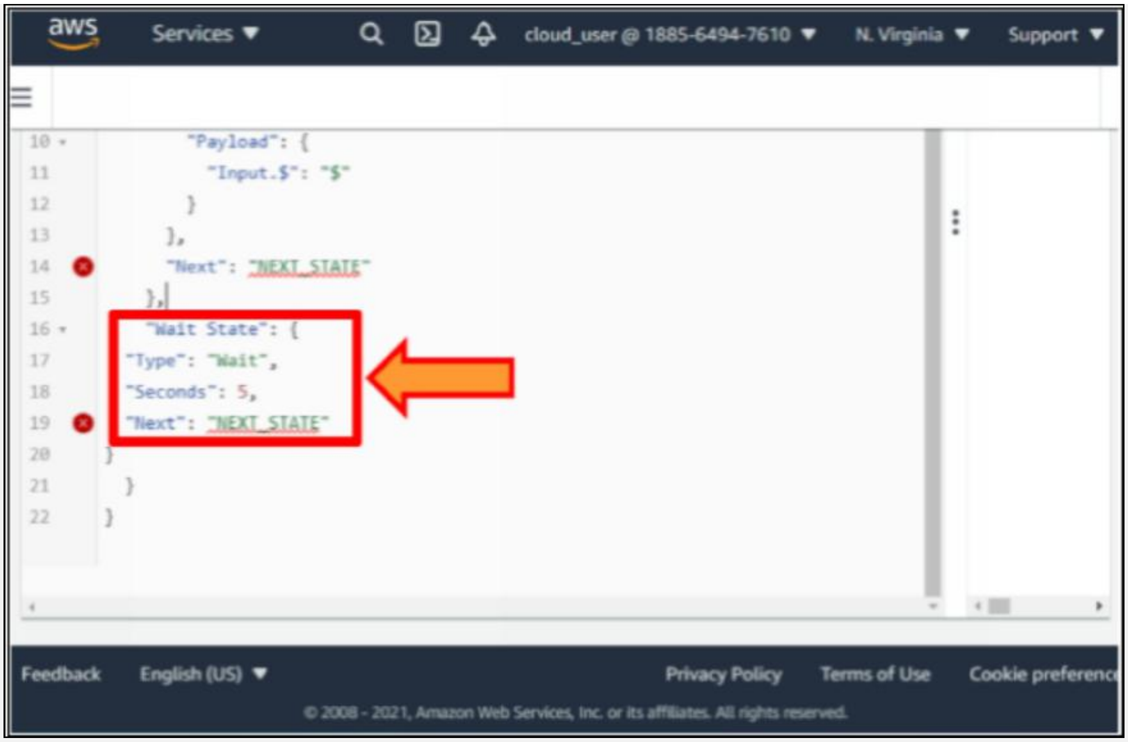
12. Create a Wait State.
13. In the **Generate code snippet** dropdown, select the **Wait state**.



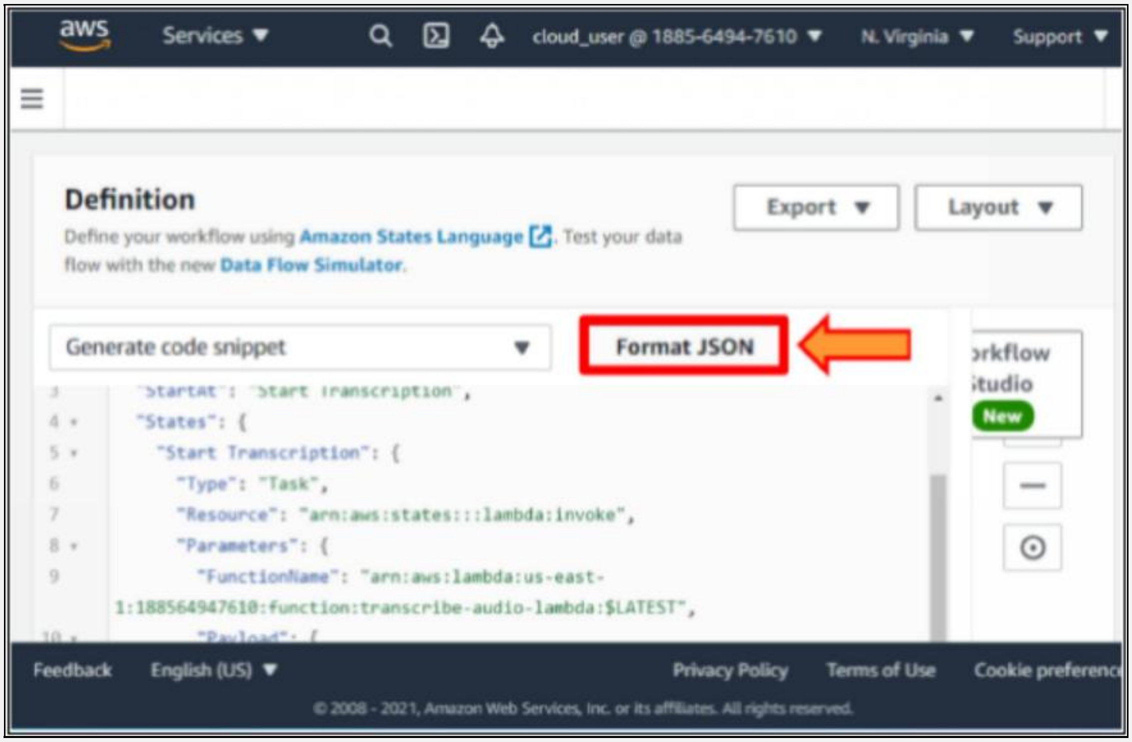
14. Click on the **Copy to clipboard** button.



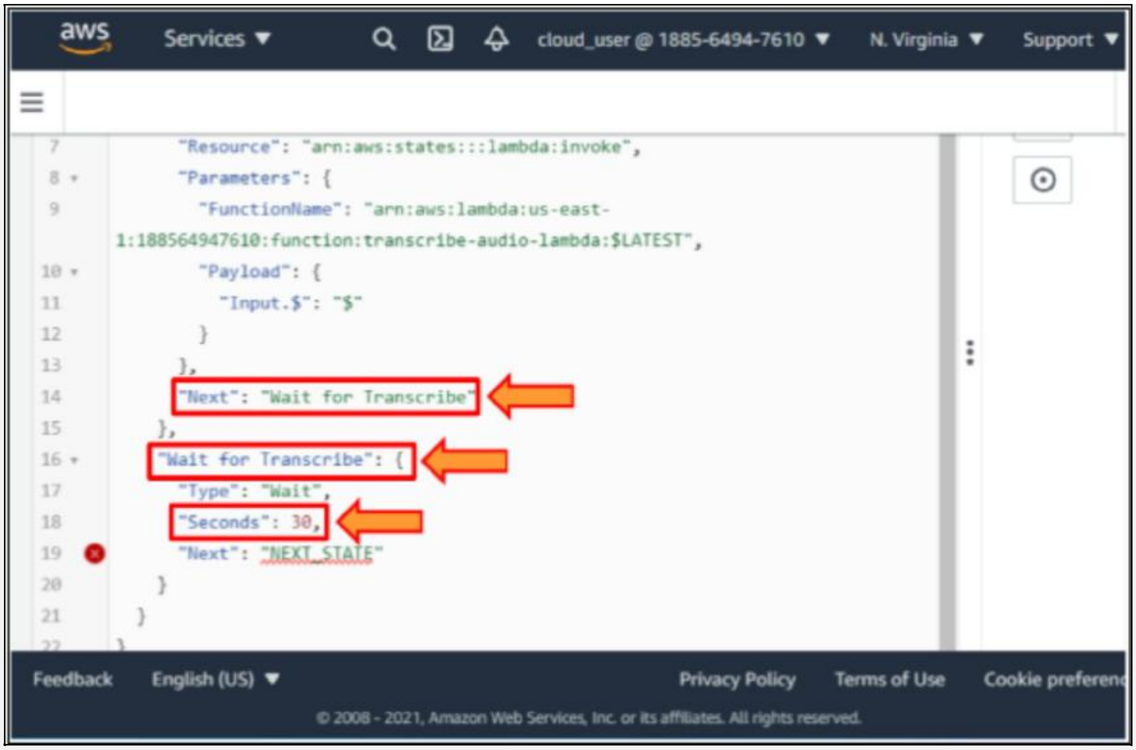
15. Paste it under the **}**, that is under **Next**: **NEXT_STATE**.



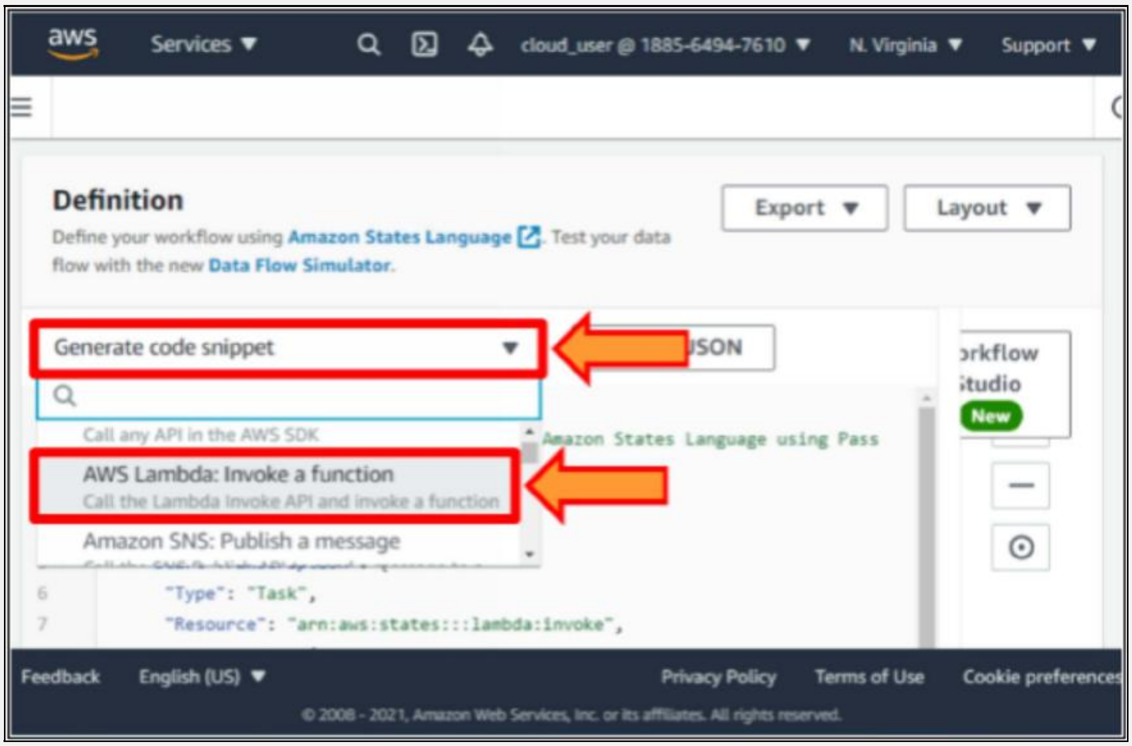
16. Click on the **Format JSON** button.



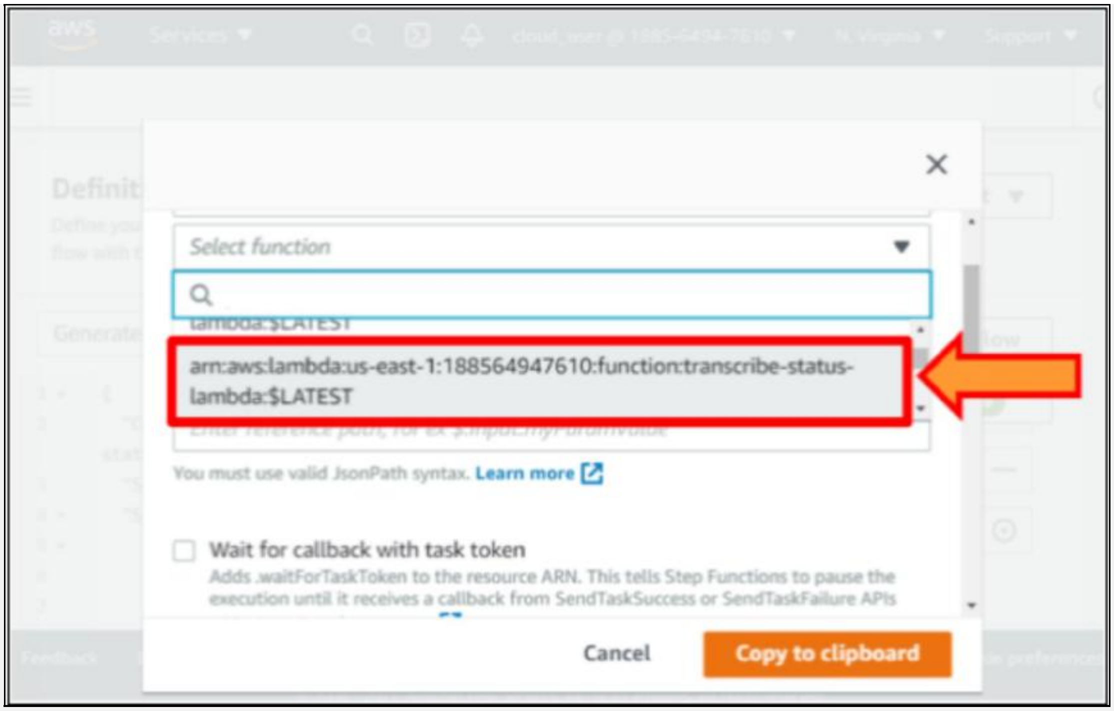
17. Replace **Wait State with**: **Wait for Transcribe**.
18. Change **Seconds: 5,** to **Seconds": 30**.
19. In the Start Transcription state, change **NEXT_STATE** to: **Wait for Transcribe**.



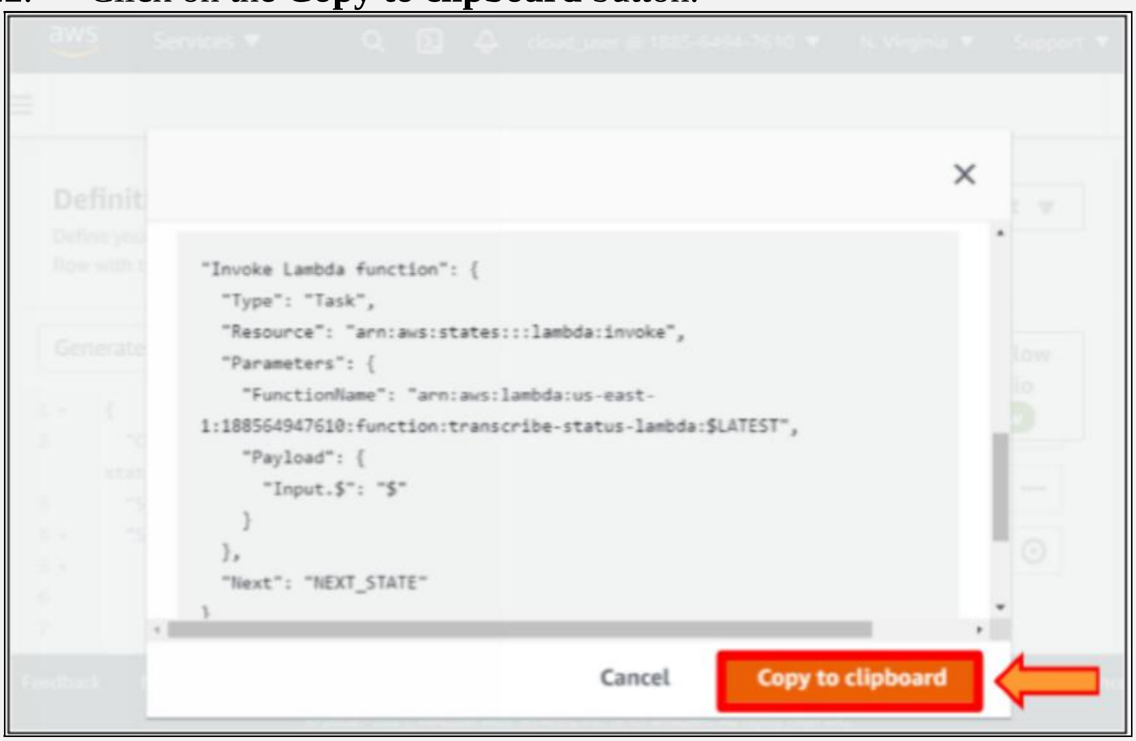
20. In the **Generate code snippet** dropdown, select **AWS Lambda: Invoke a function.**



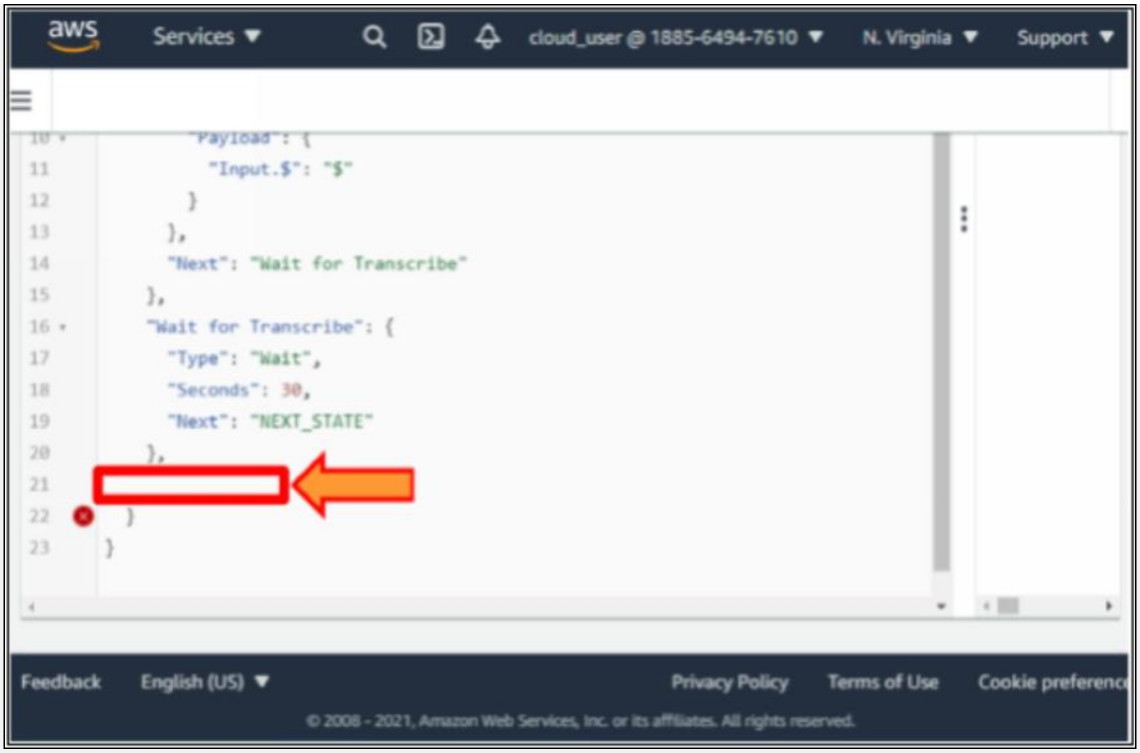
21. Select the **transcribe-status-lambda** function.



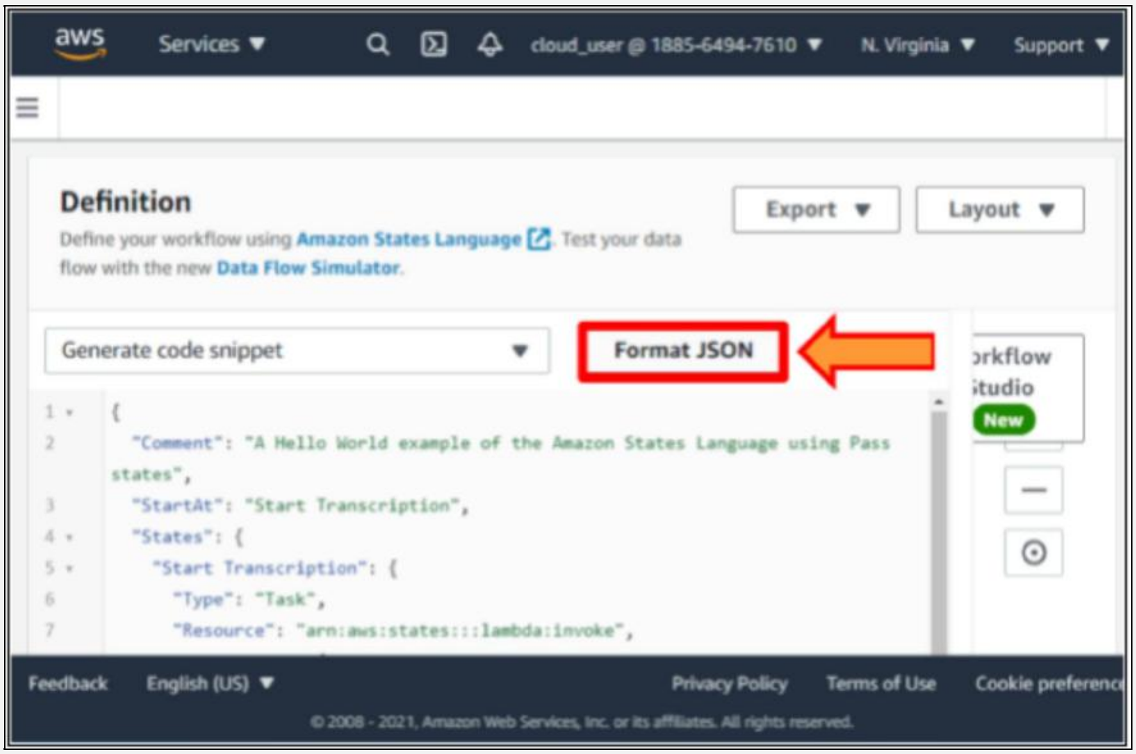
22. Click on the **Copy to clipboard** button.



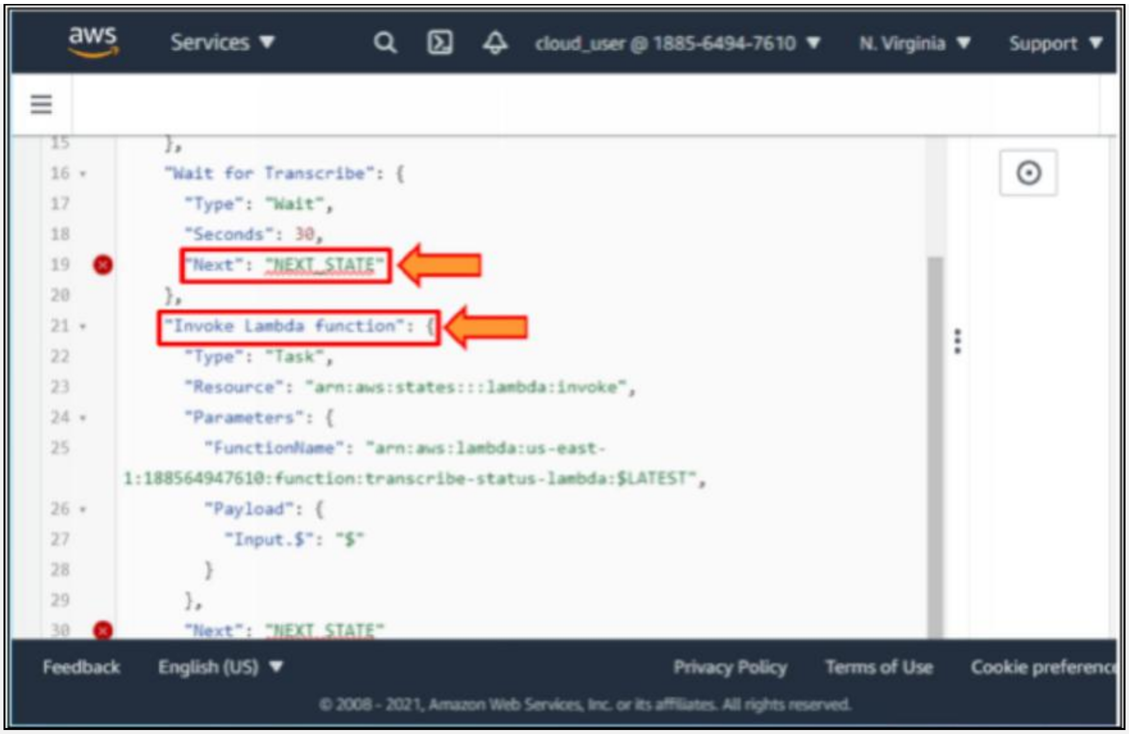
23. Paste it under the **}**, that is under **Next**: **NEXT_STATE**.



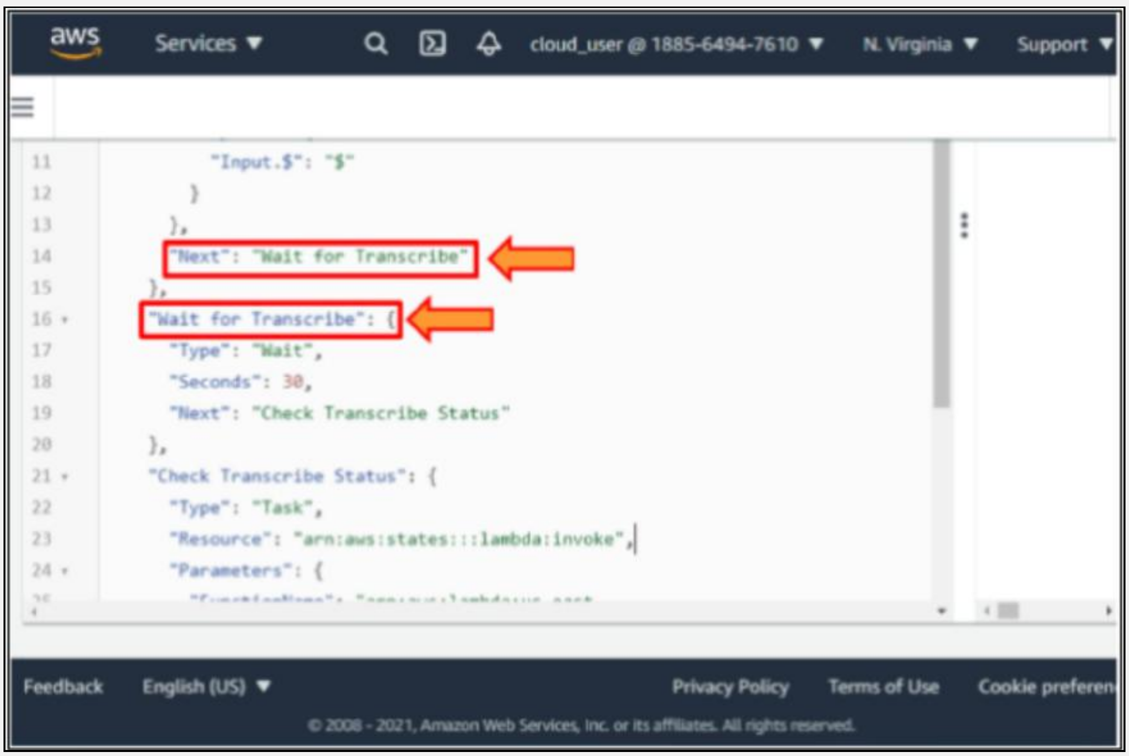
24. Click on the **Format JSON** button.



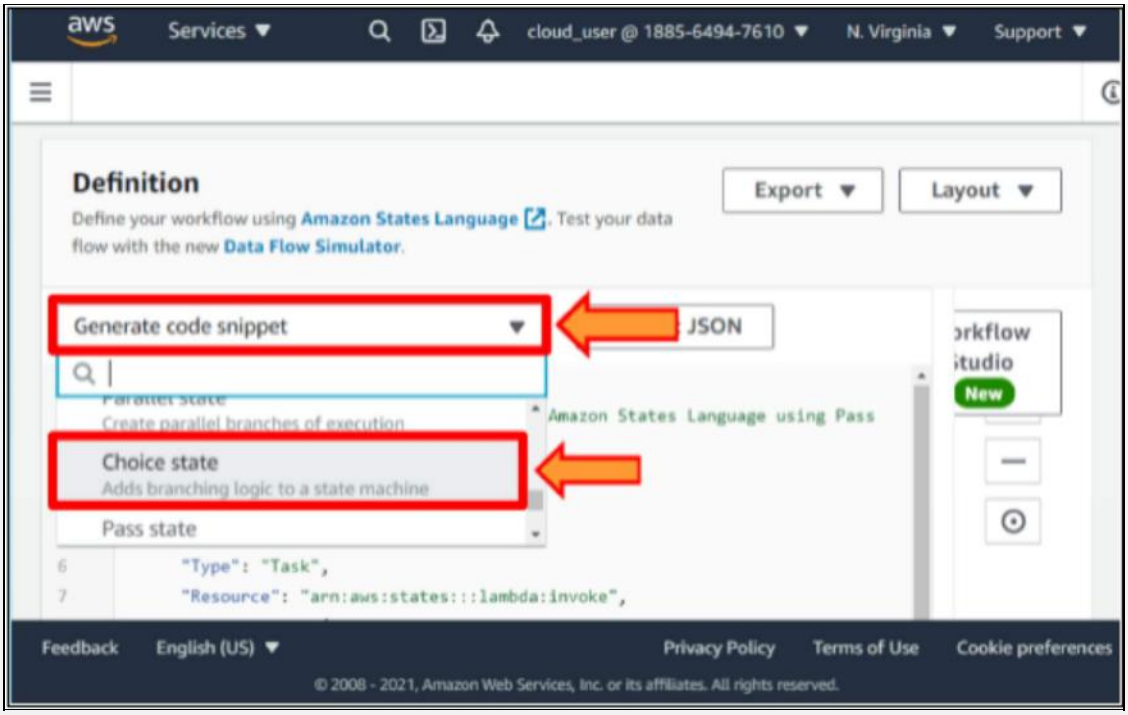
25. Rename **Invoke lambda function** to **Check Transcribe Status.**



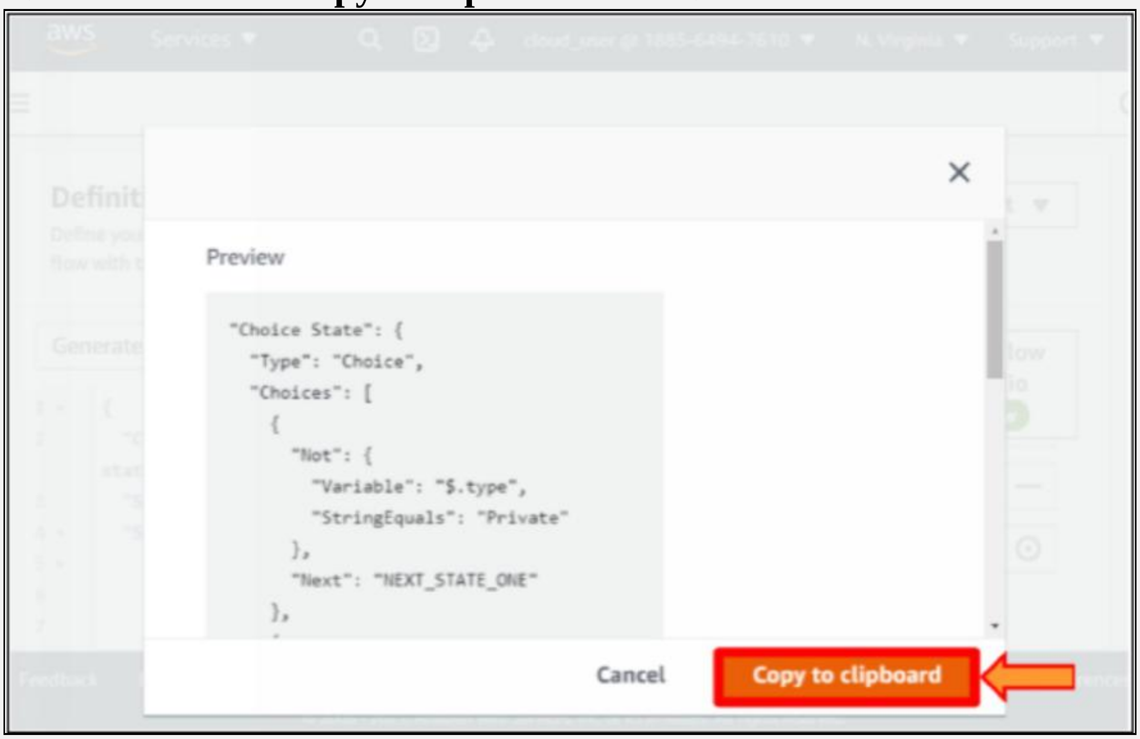
26. In the Wait State, change **NEXT_STATE** to **Check Transcribe Status.**



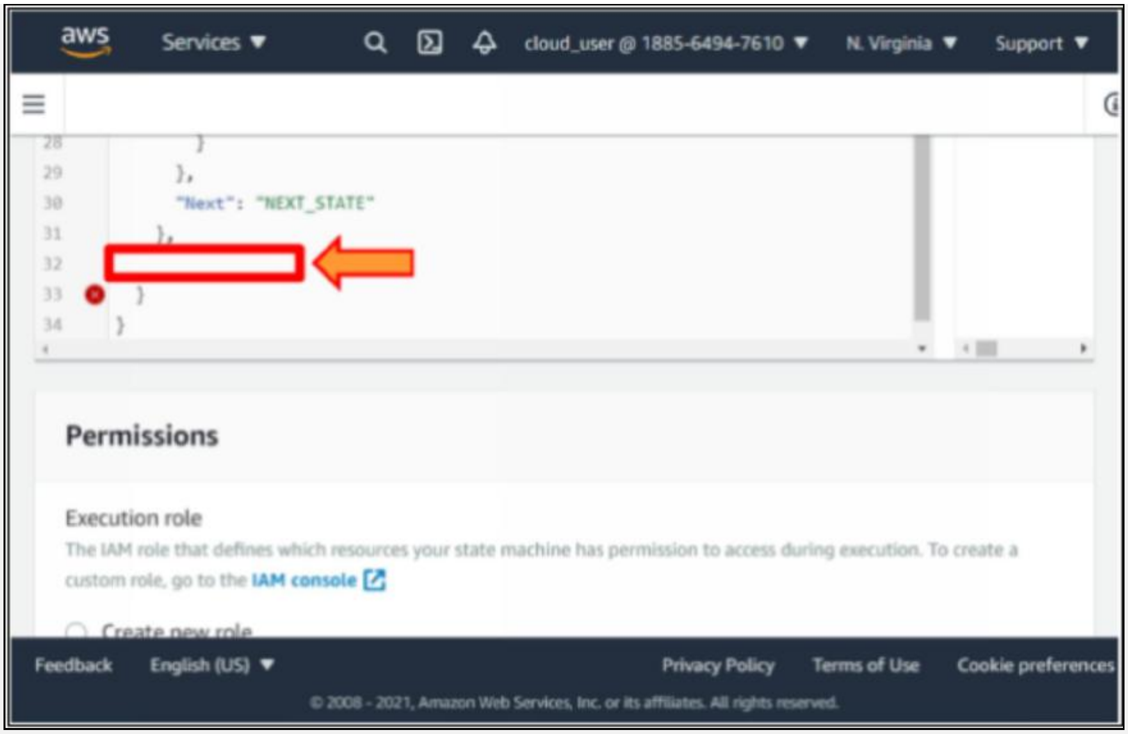
27. Create a Completed and Failed State
28. In the **Generate code snippet** dropdown, select **Choice state**.



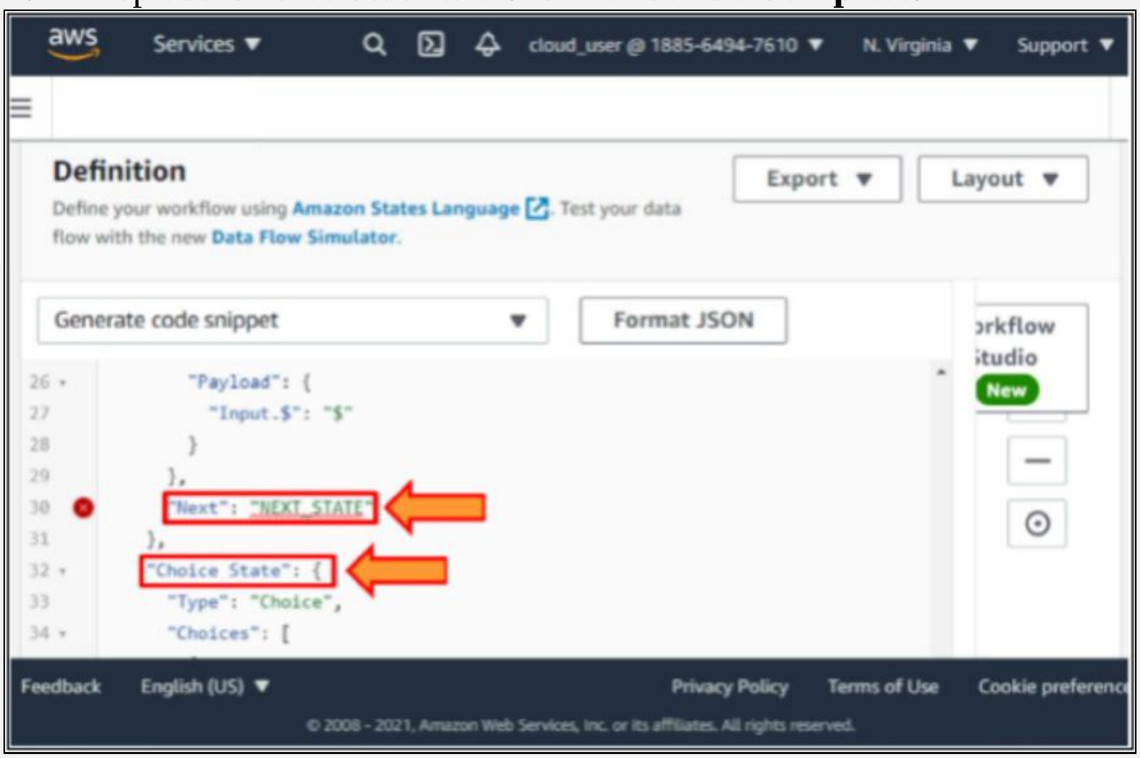
29. Click on the **Copy to clipboard** button.



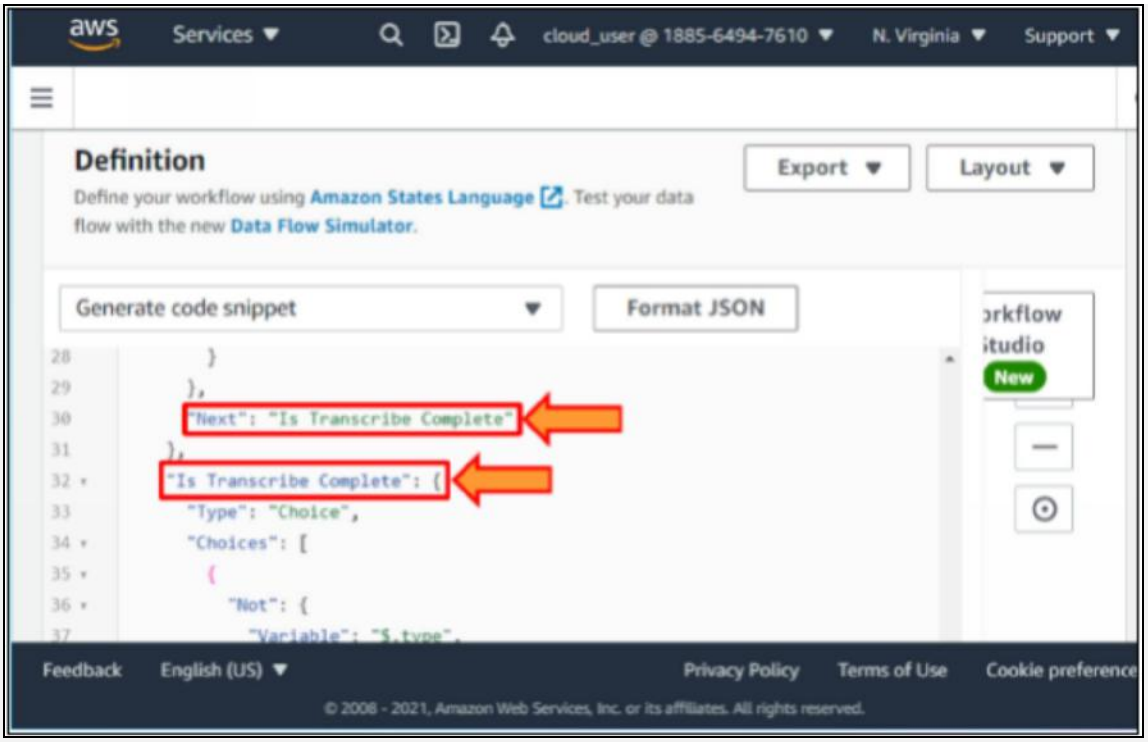
30. Paste it under } found under **Next: NEXT_STATE**.



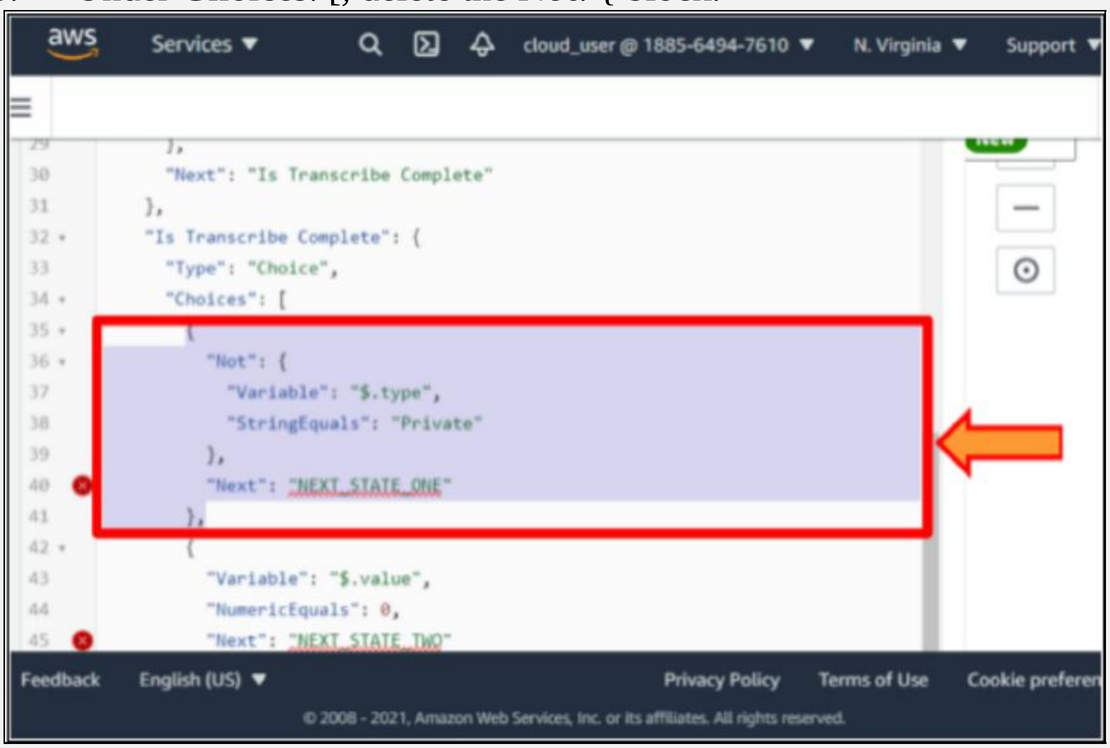
31. Replace **Choice State** with: **Is Transcribe Complete.**



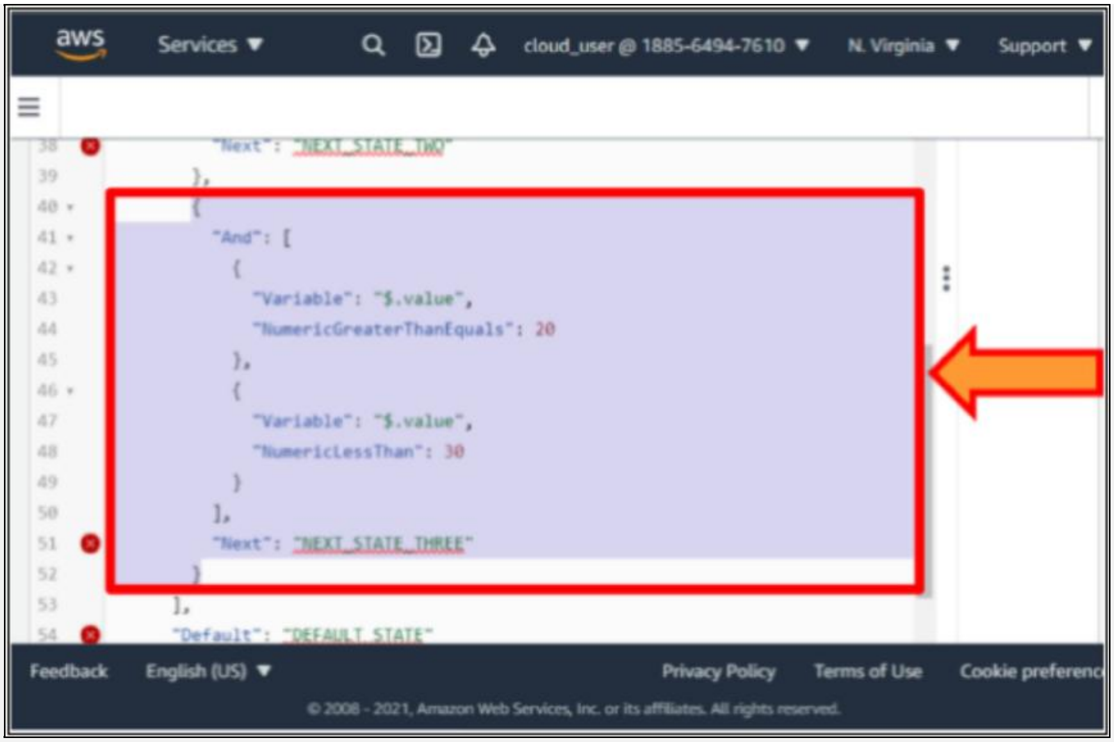
32. In the **Check Transcribe Status** state, change **NEXT_STATE** to: **Is Transcribe Complete.**



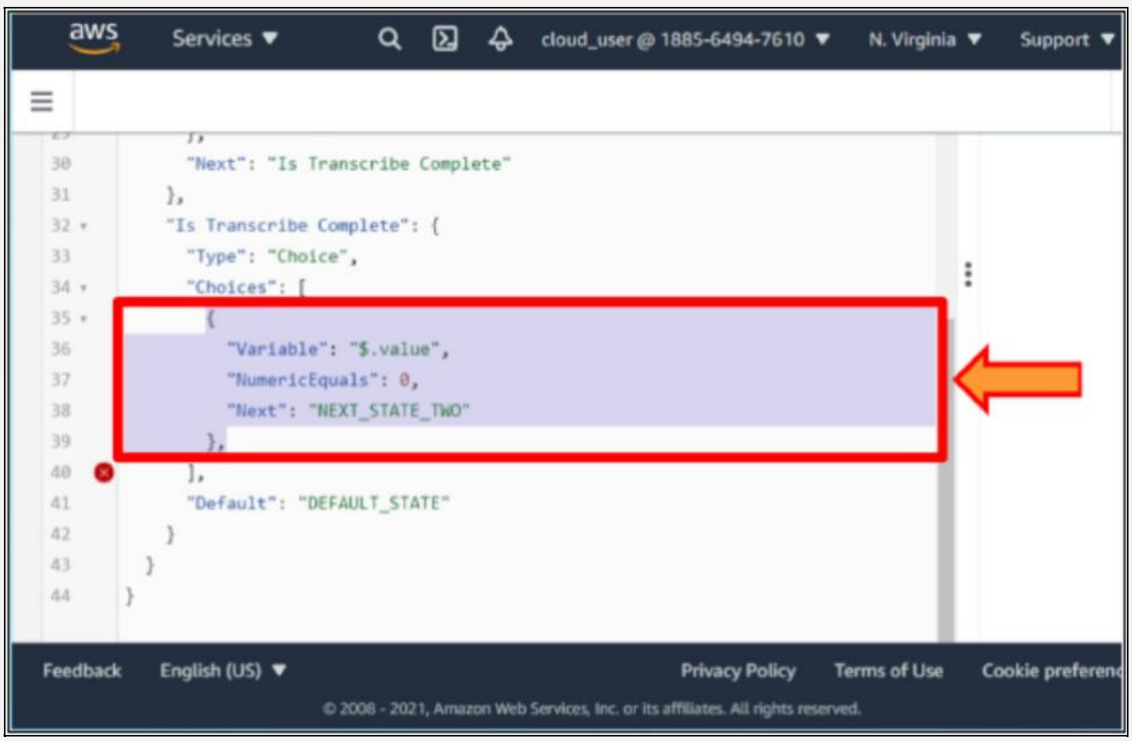
33. Under **Choices**: [, delete the **Not**: { block.



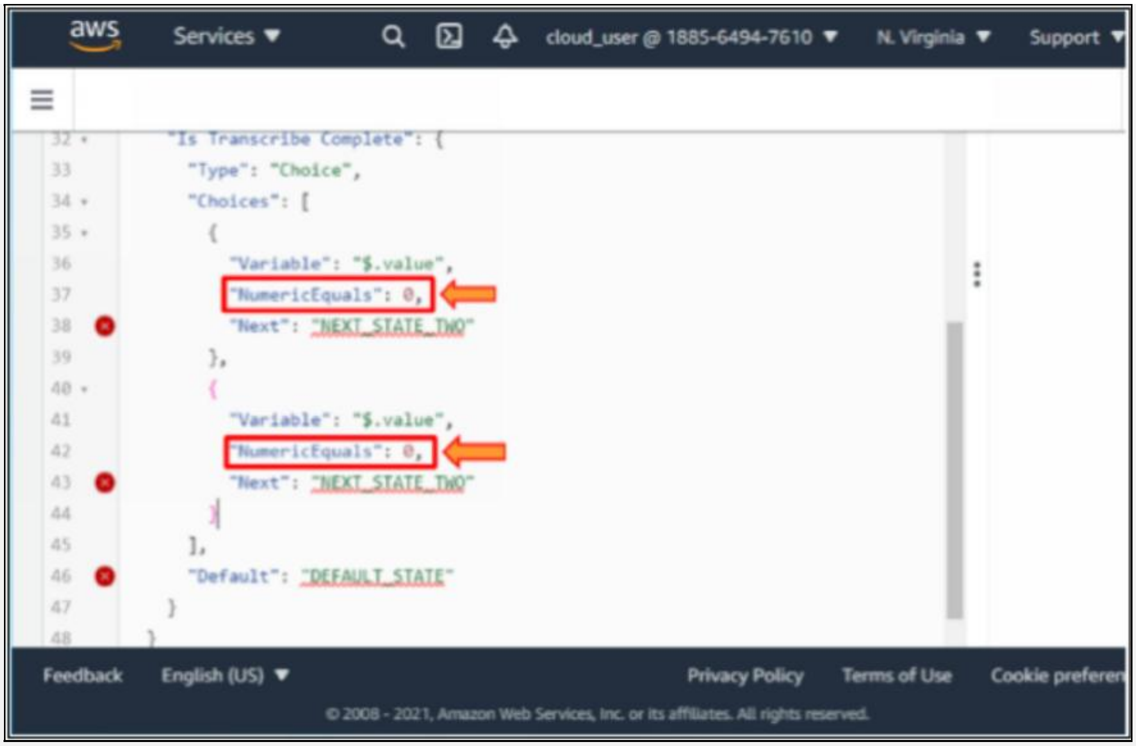
34. Delete the **And**: { block.



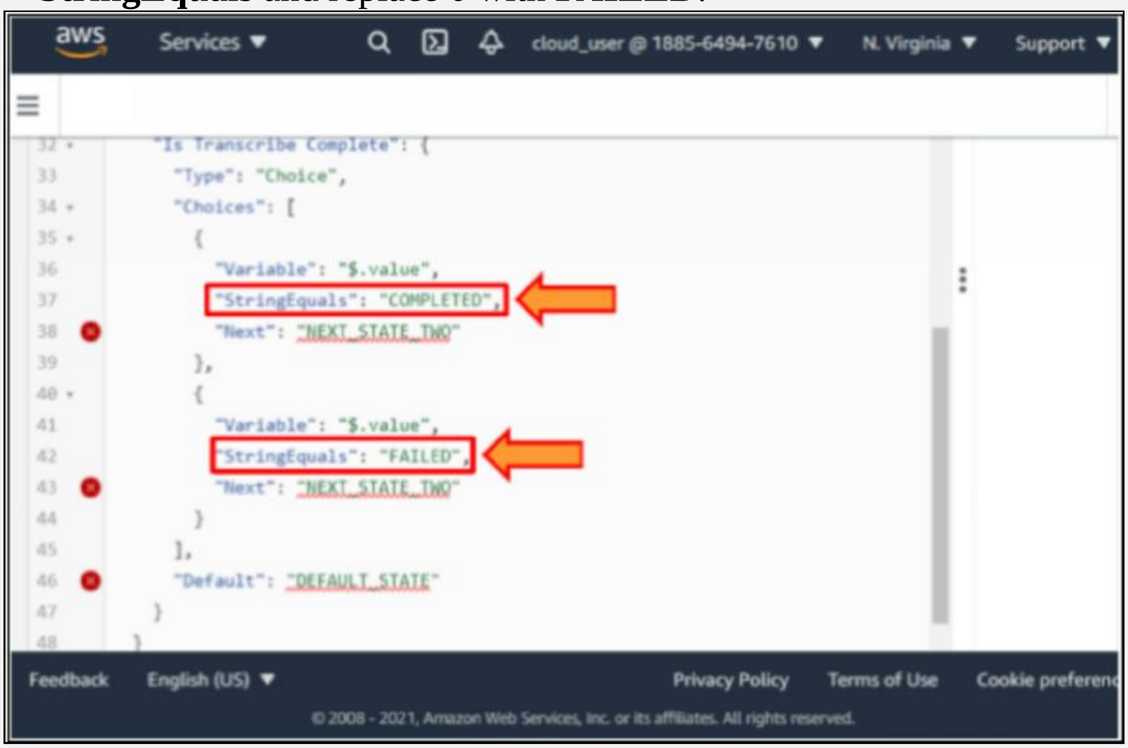
35. Copy the block under **Choices**: [and paste it under the existing block.



36. In the first Choices block, replace **NumericEquals** with **StringEquals** and replace **0** with **COMPLETED**.

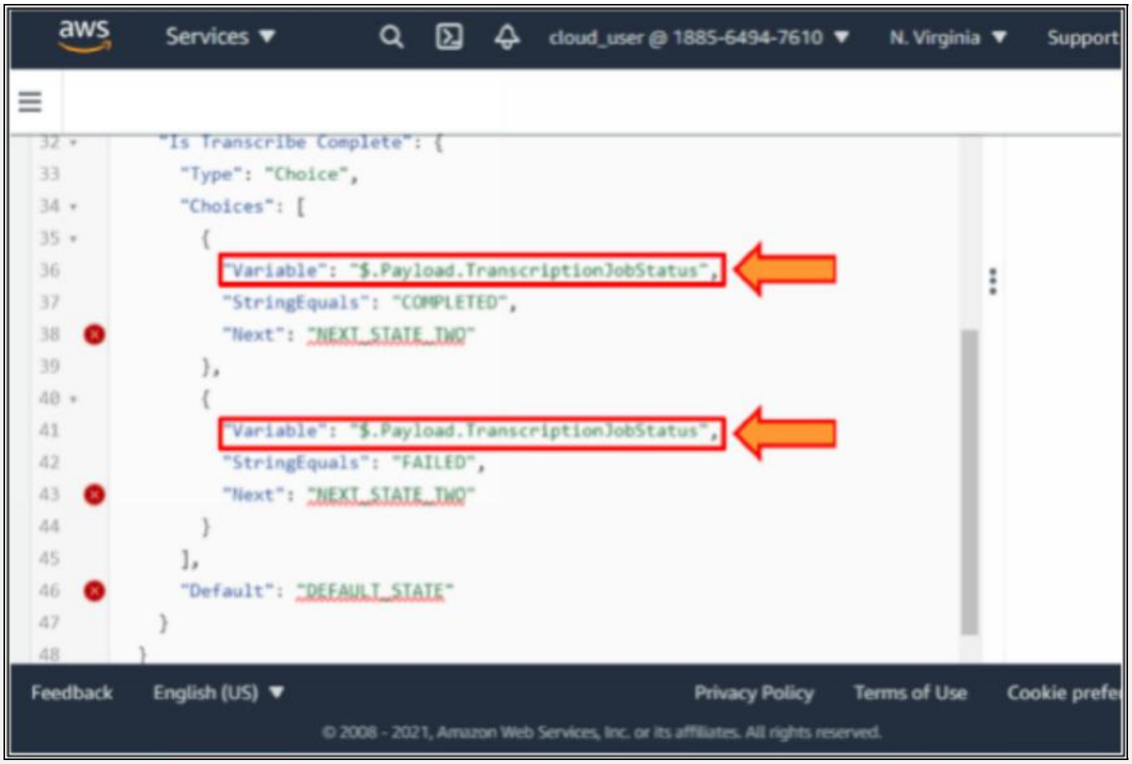


37. In the second Choices block, replace **NumericEquals** with **StringEquals** and replace **0** with **FAILED**.

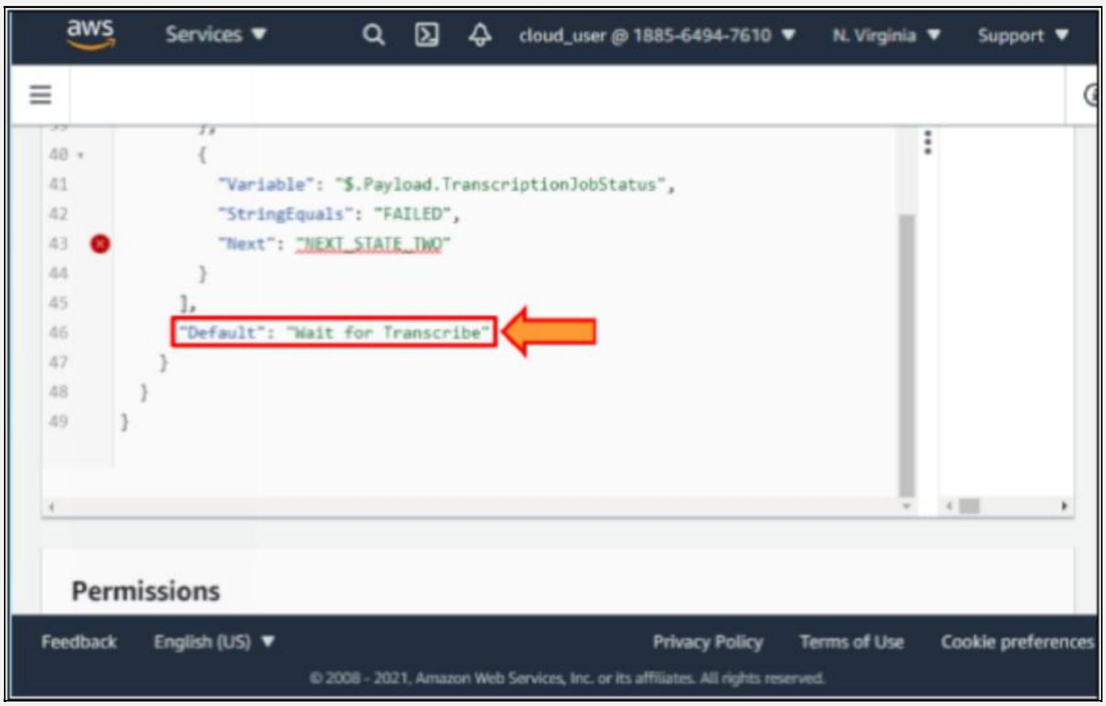


38. In the first Choices block, replace **$.value** with: **$.Payload.TranscriptionJobStatus.**

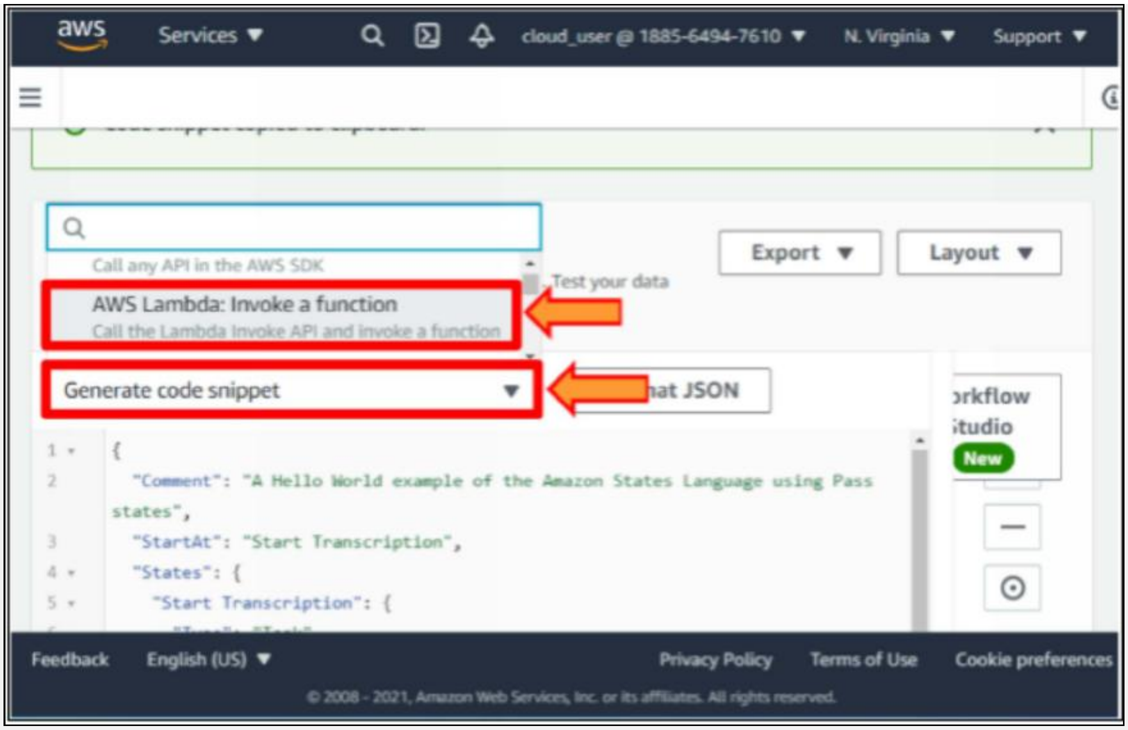
39. In the second Choices block, replace **$.value** with: **$.Payload.TranscriptionJobStatus.**



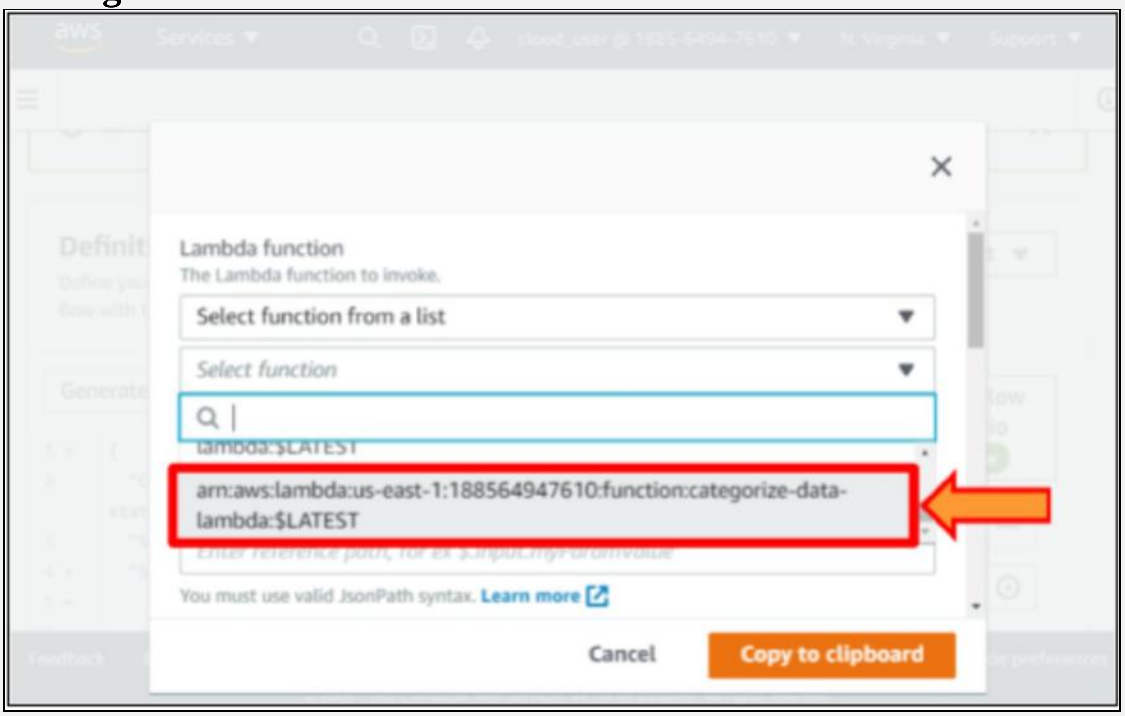
40. Replace **DEFAULT_STATE** with: **Wait for Transcribe**



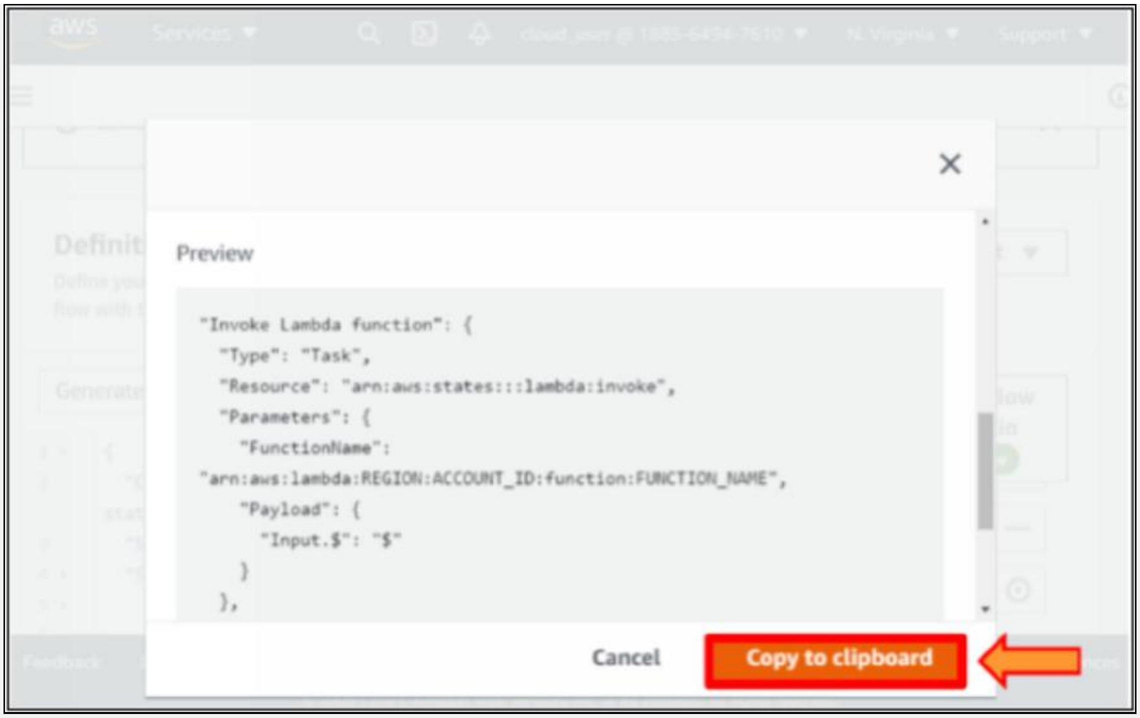
41. In the **Generate code snippet** dropdown, select **AWS Lambda: Invoke a function.**



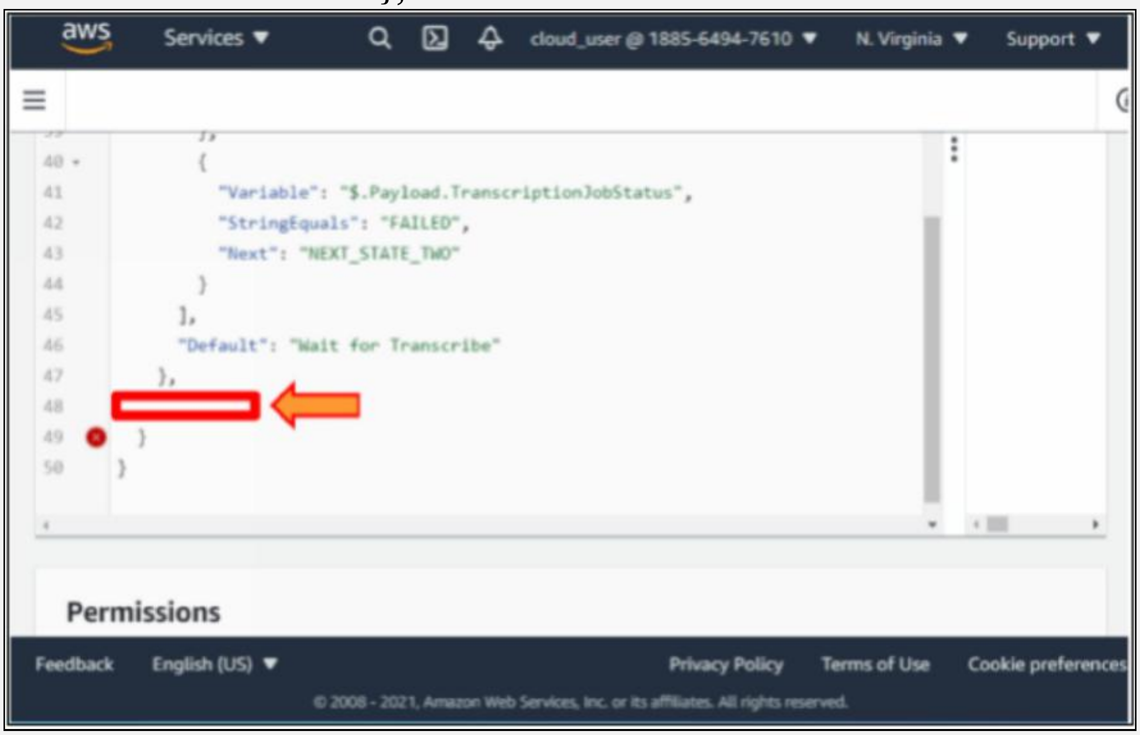
42. In the Select function from a list dropdown menu. Select the **categorize-data-lambda** function.



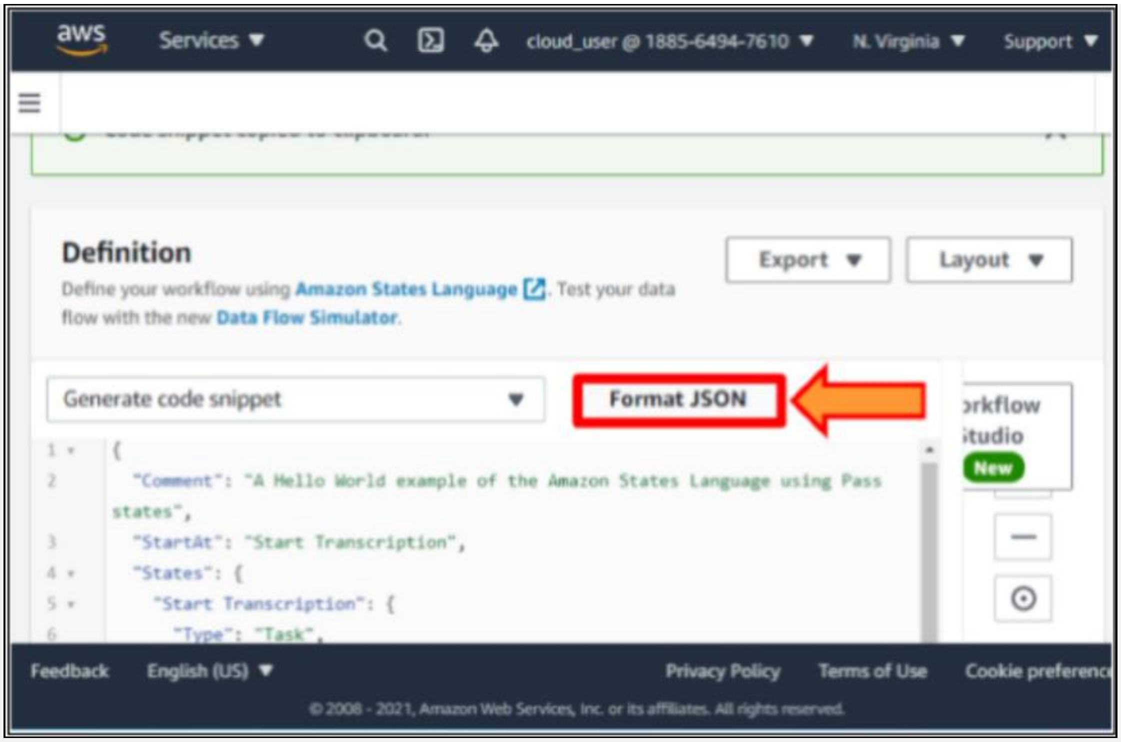
43. Click on the **Copy to clipboard** button.



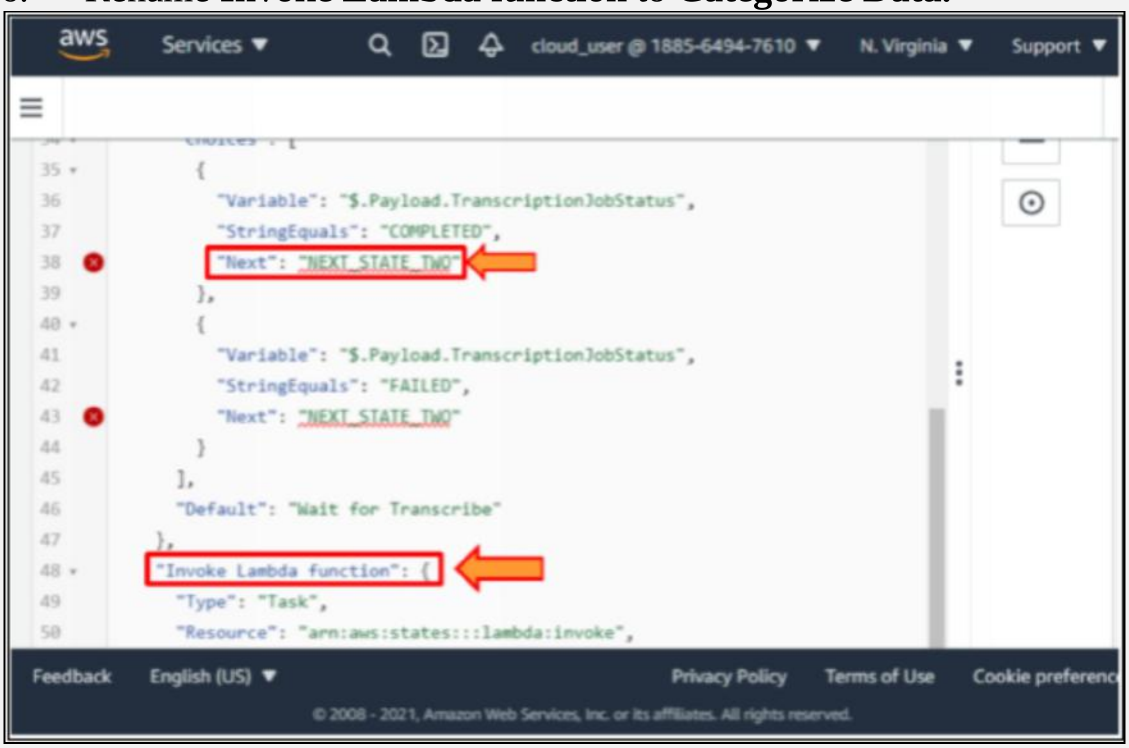
44. Paste it under the '},' that is under **Default**: **Wait For Transcribe.**



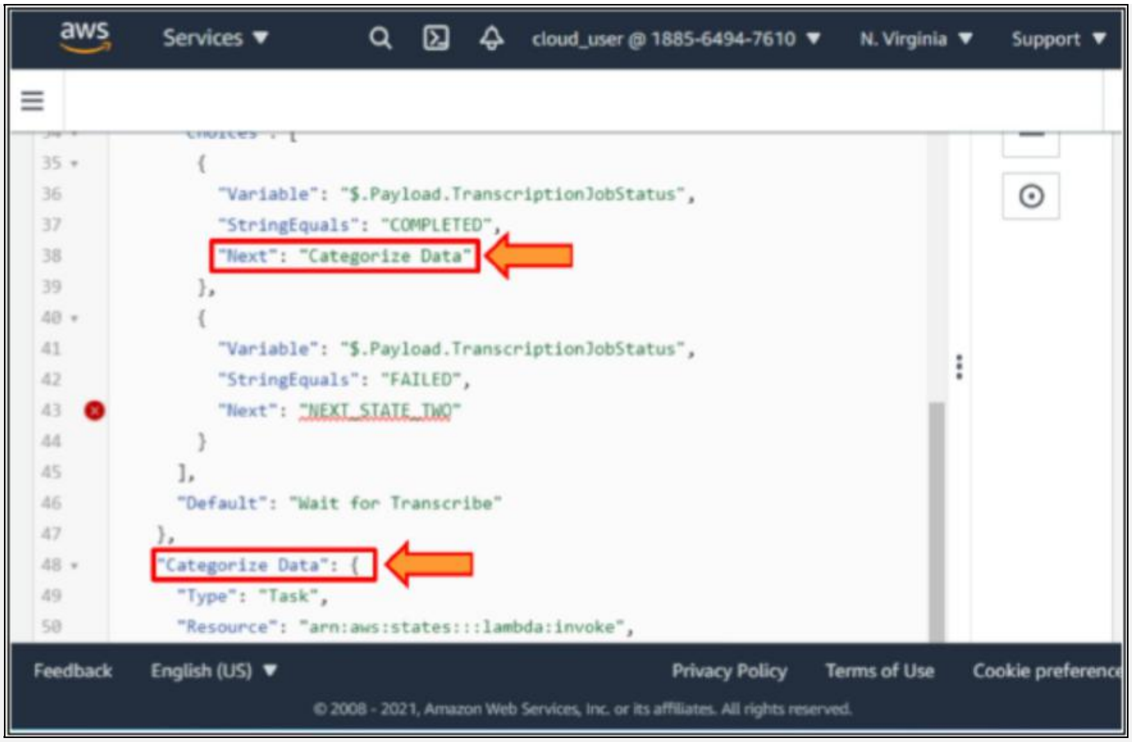
45. Click on the Format JSON button.



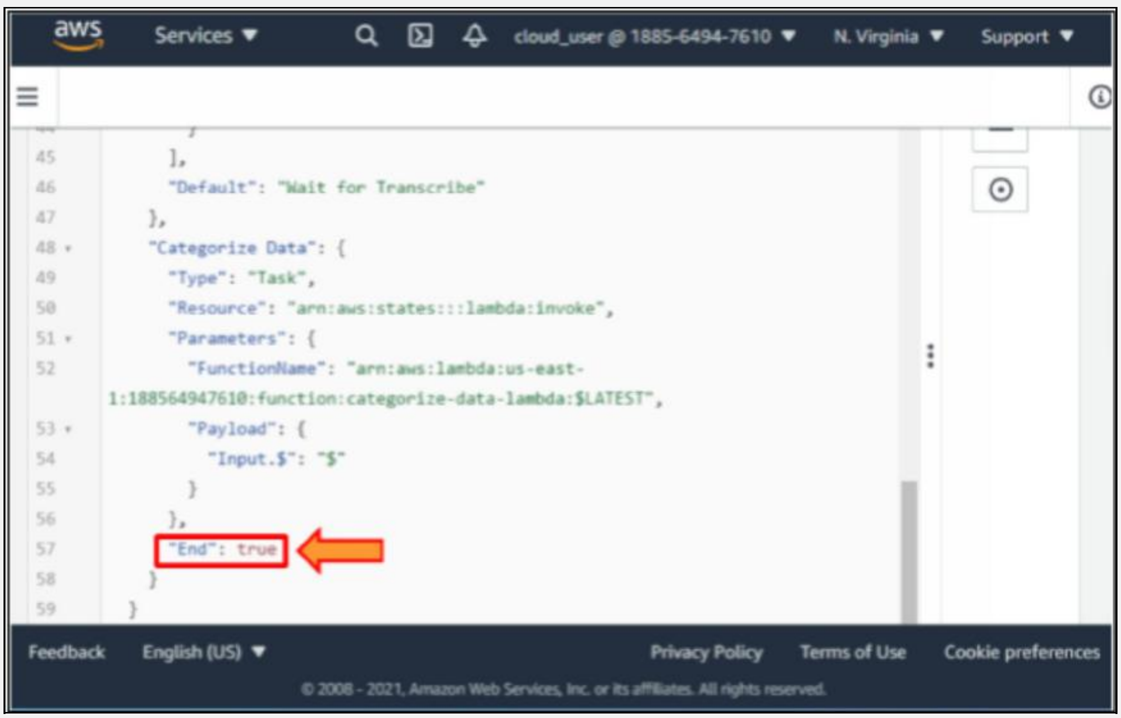
46. Rename **Invoke Lambda function** to **Categorize Data.**



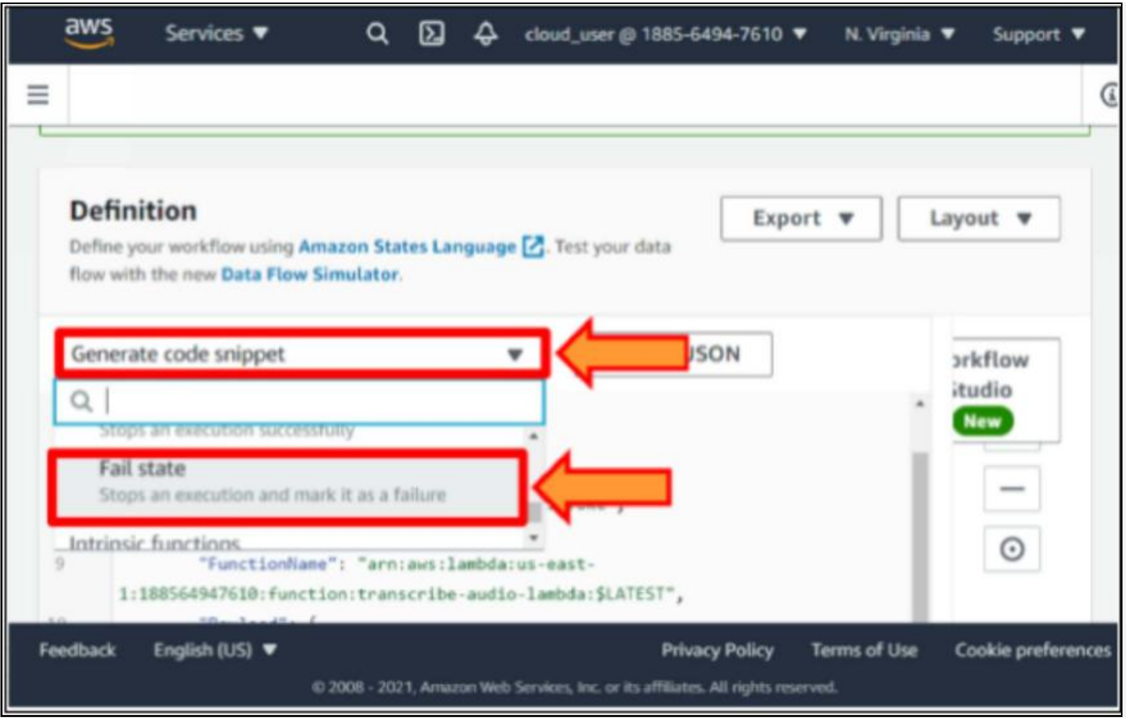
47. In the Completed block for Choices, replace **NEXT_STATE_TWO** with **Categorize Data.**



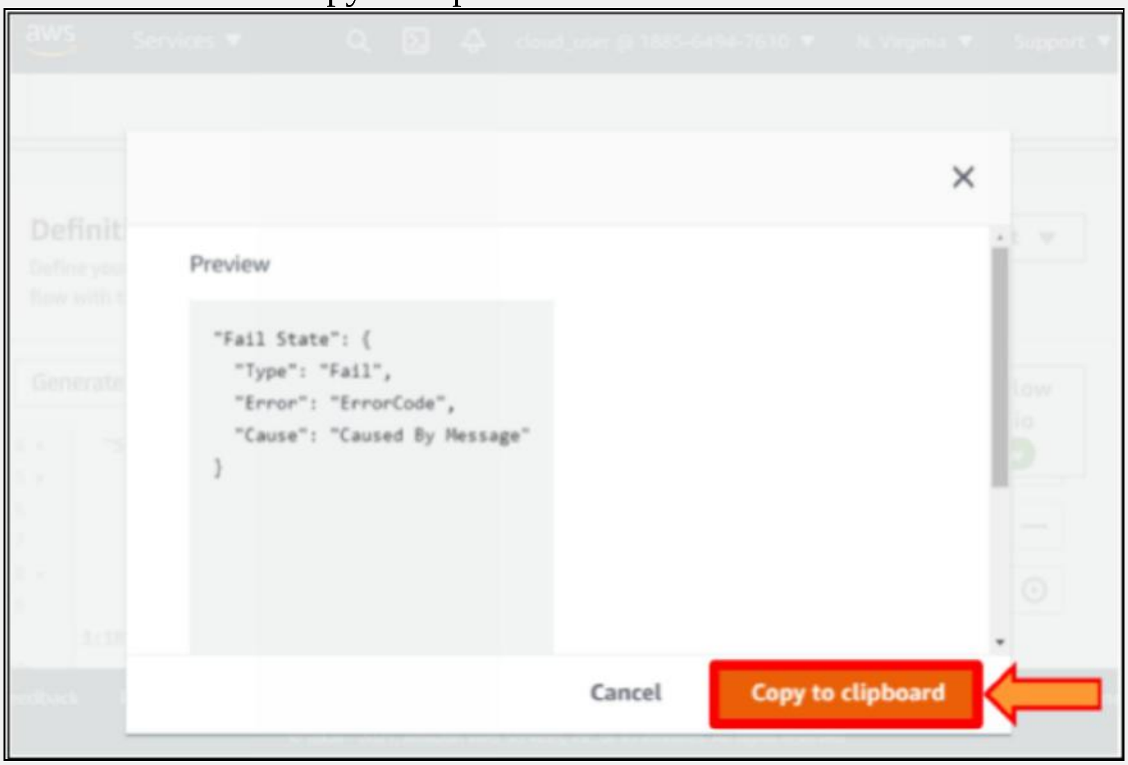
48. In the Categorize Data block, replace **Next: NEXT_STATE** with: **End:** true



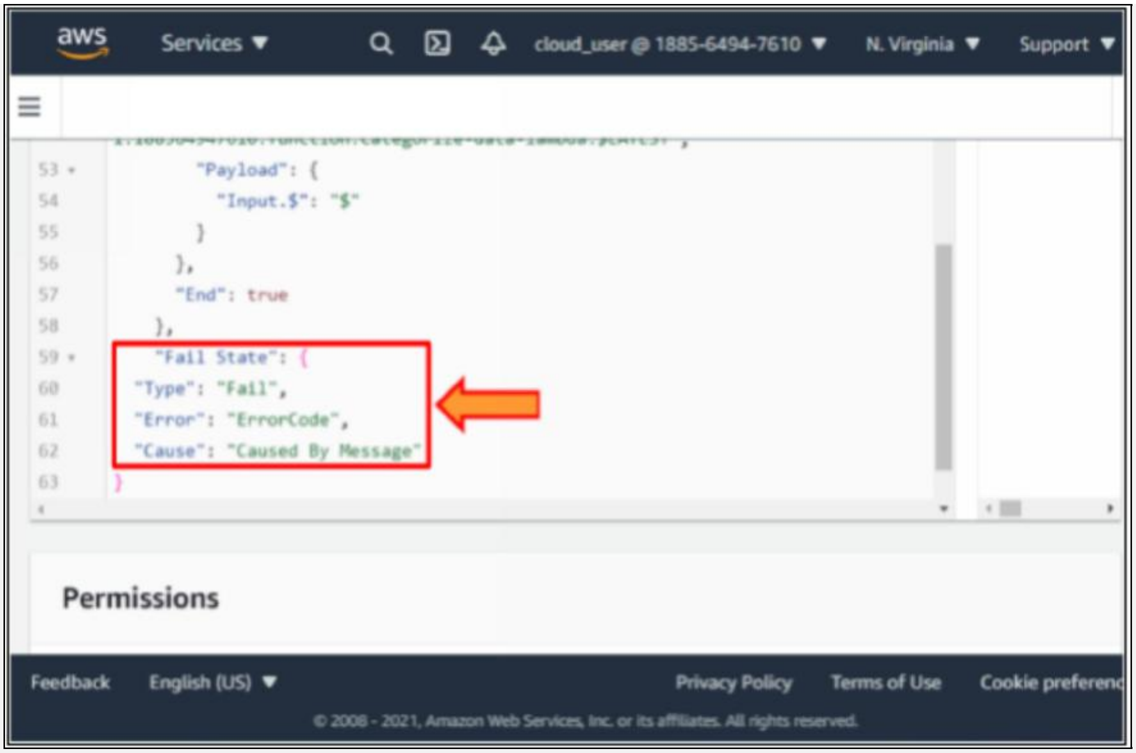
49. In the **Generate code snippet** dropdown, select the **Fail state**.



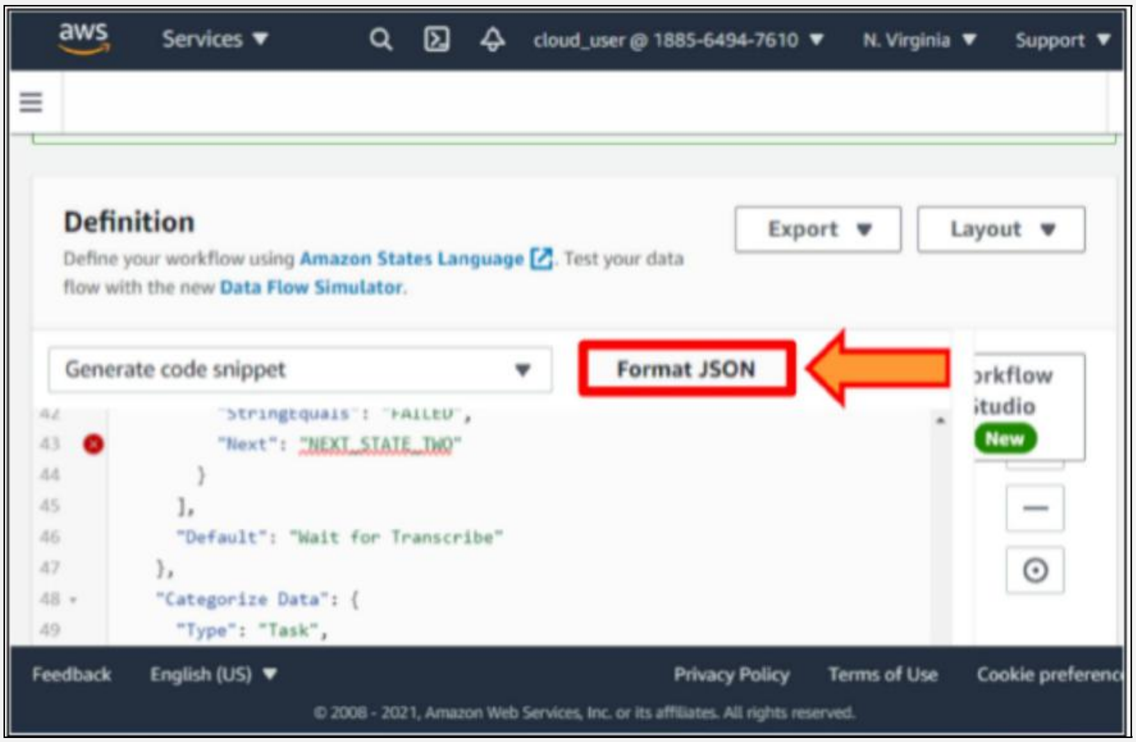
50. Click on the Copy to clipboard button.



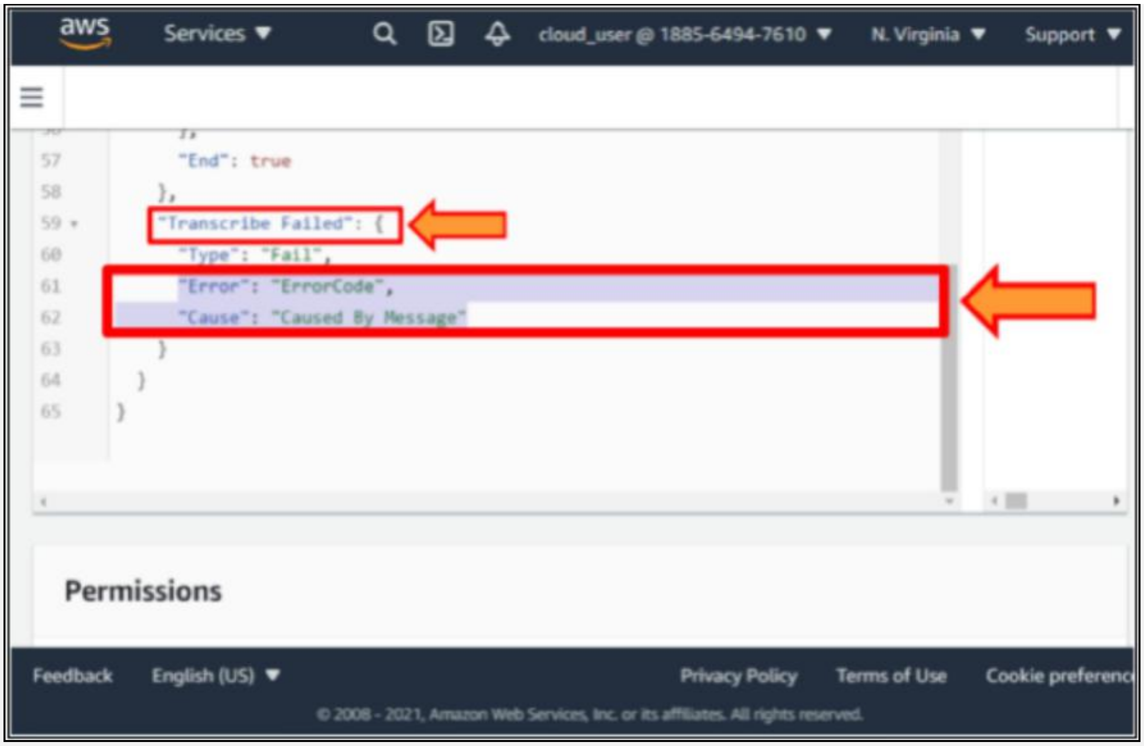
51. Paste it under the **}**, found under **End:** true.



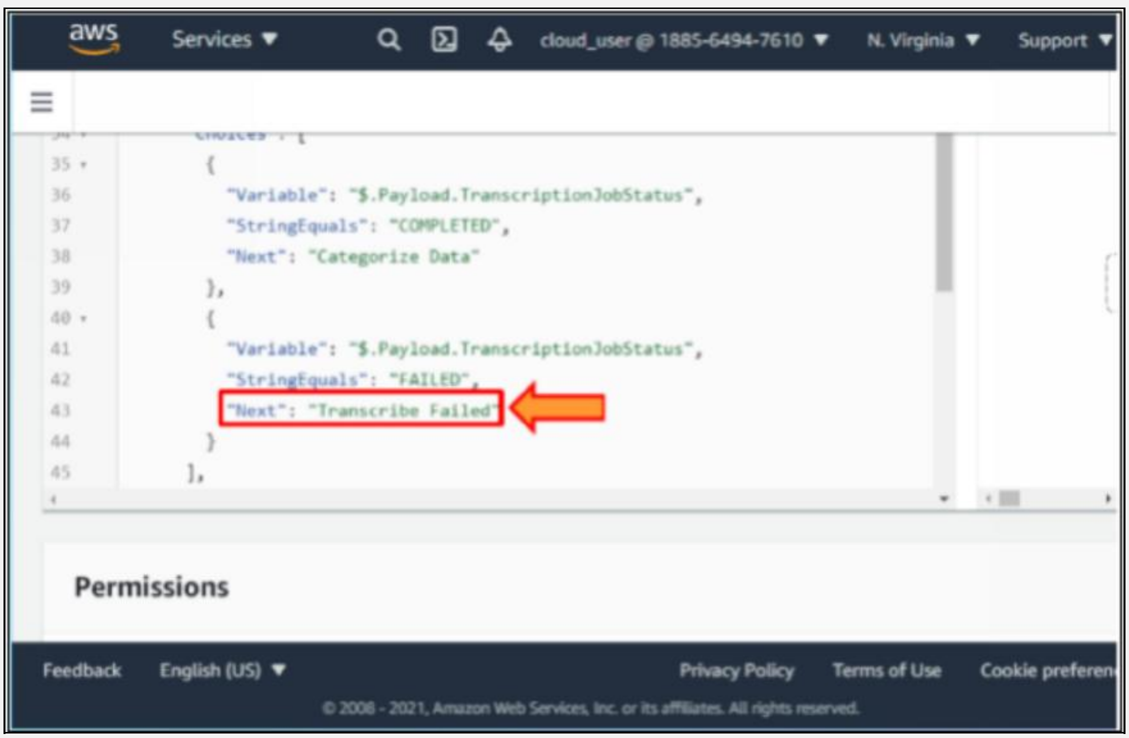
52. Click on the **Format JSON** button.



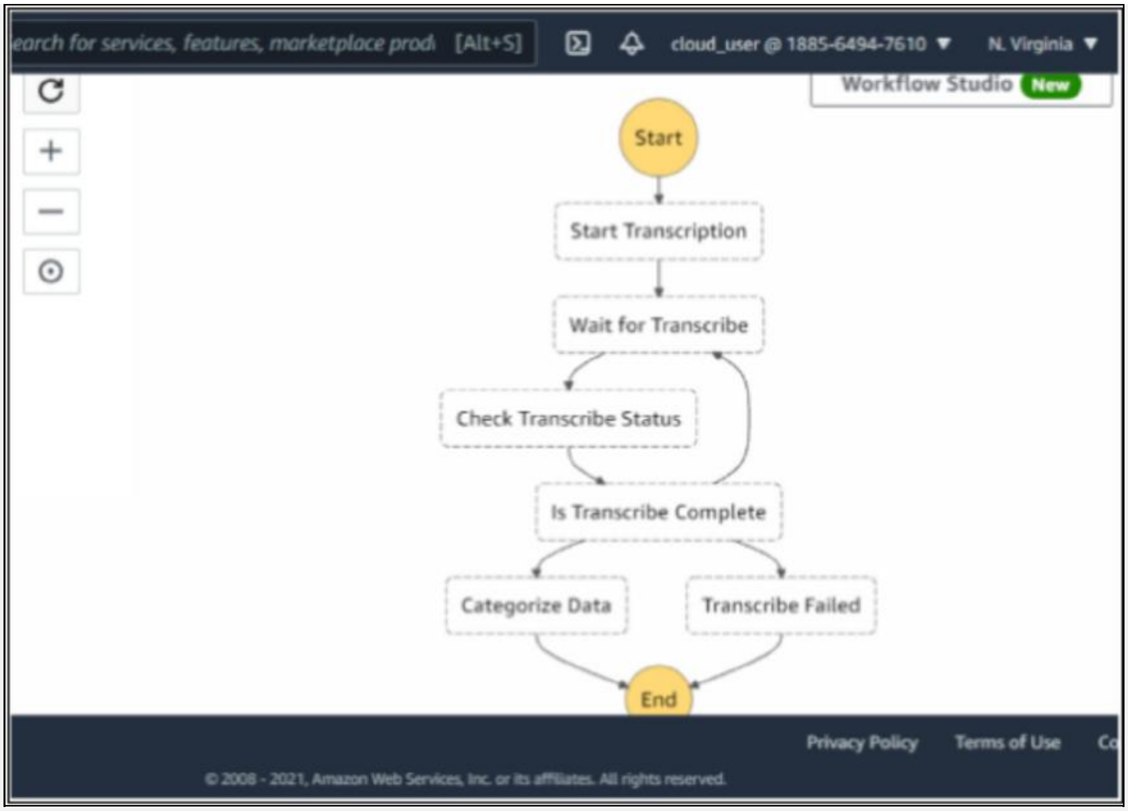
53. Replace **Fail State** with **Transcribe Failed.**
54. Delete the **Error** and **Cause** lines.



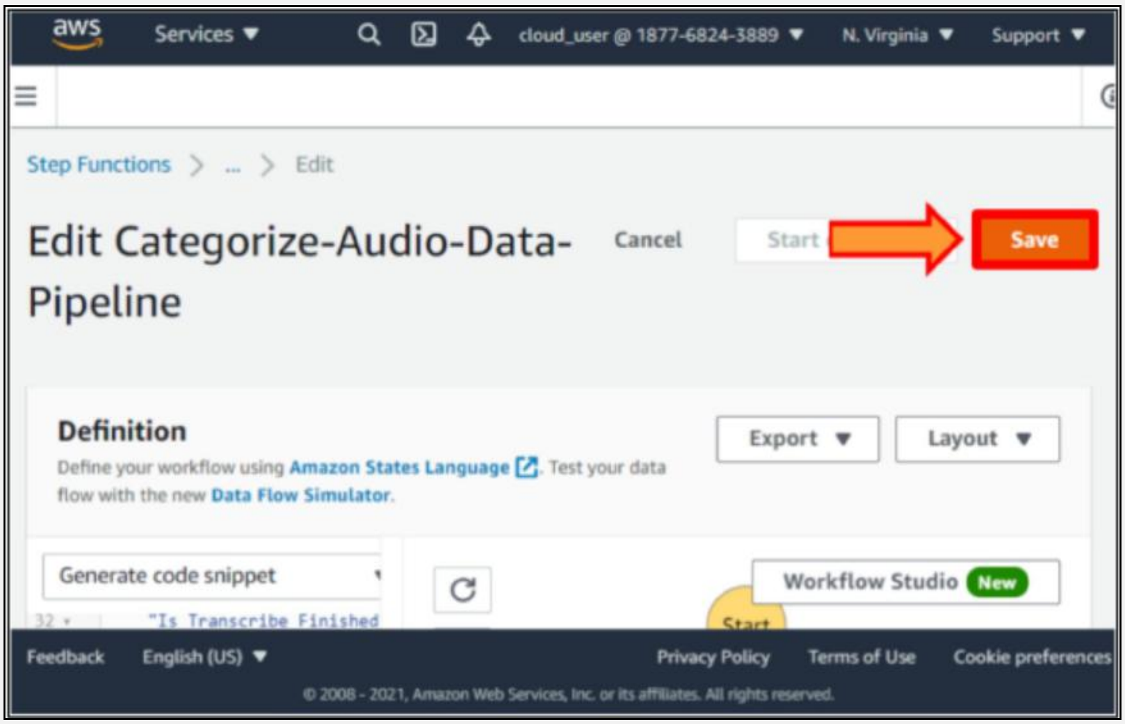
55. In the Failed block for Choices, replace **NEXT_STATE_TWO** with: **Transcribe Failed.**



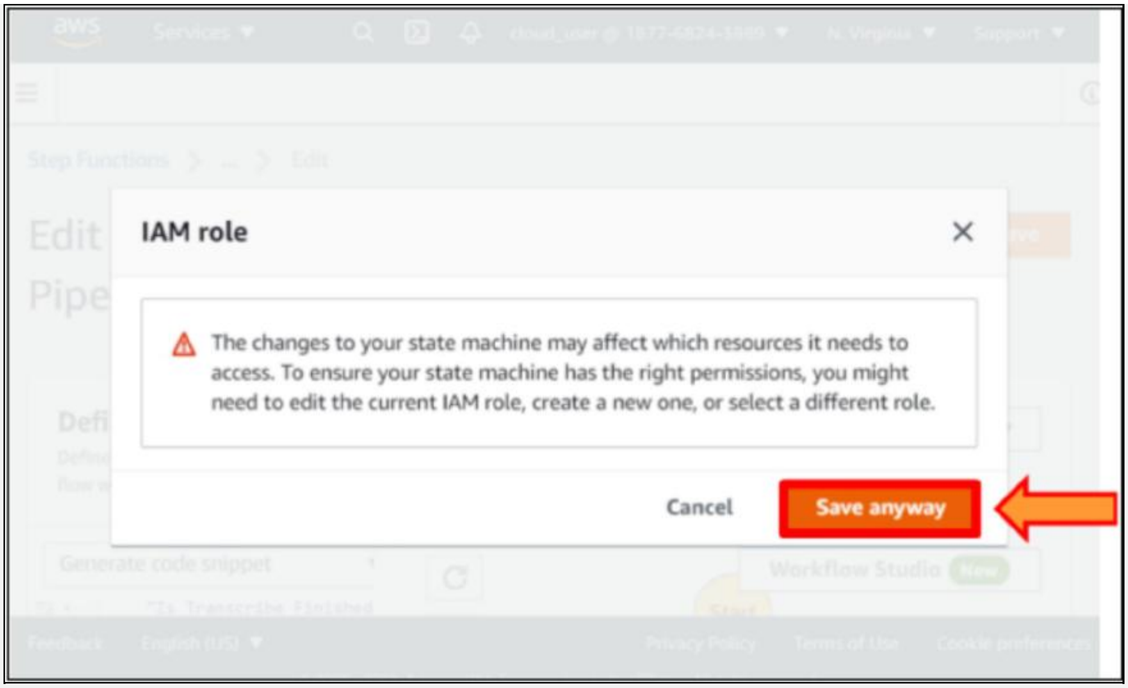
56. After that, you will see the block graph of the model.



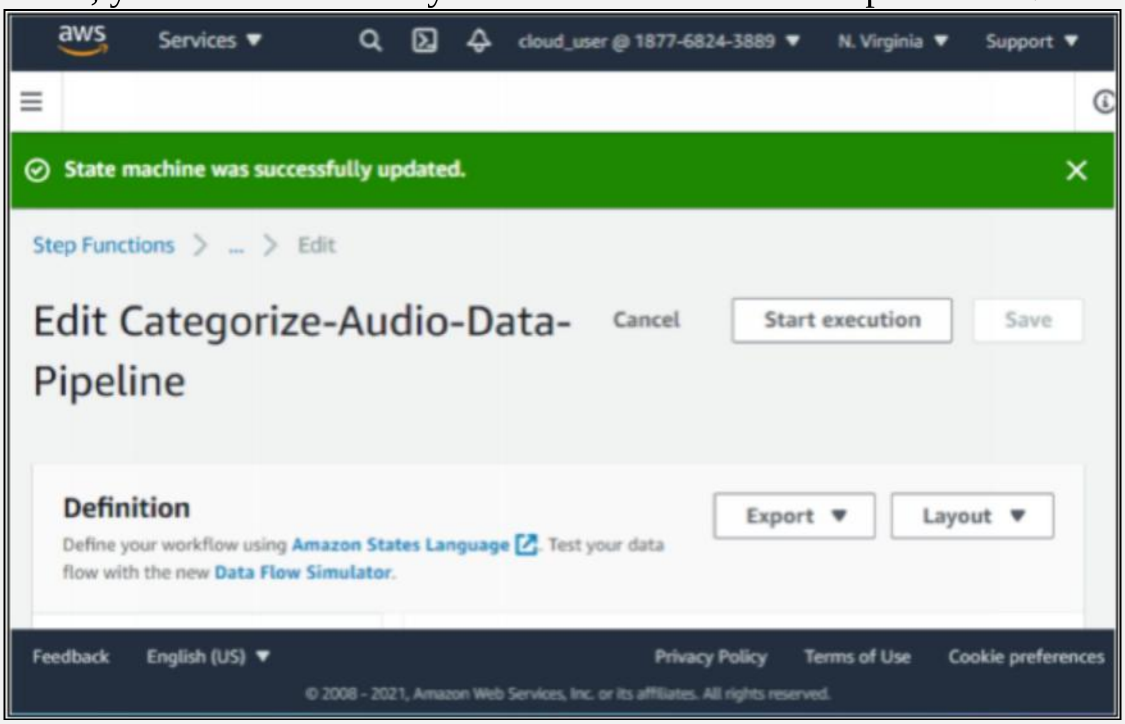
57. Click on the **Save** button.



58. Click on the **Save anyway** button.

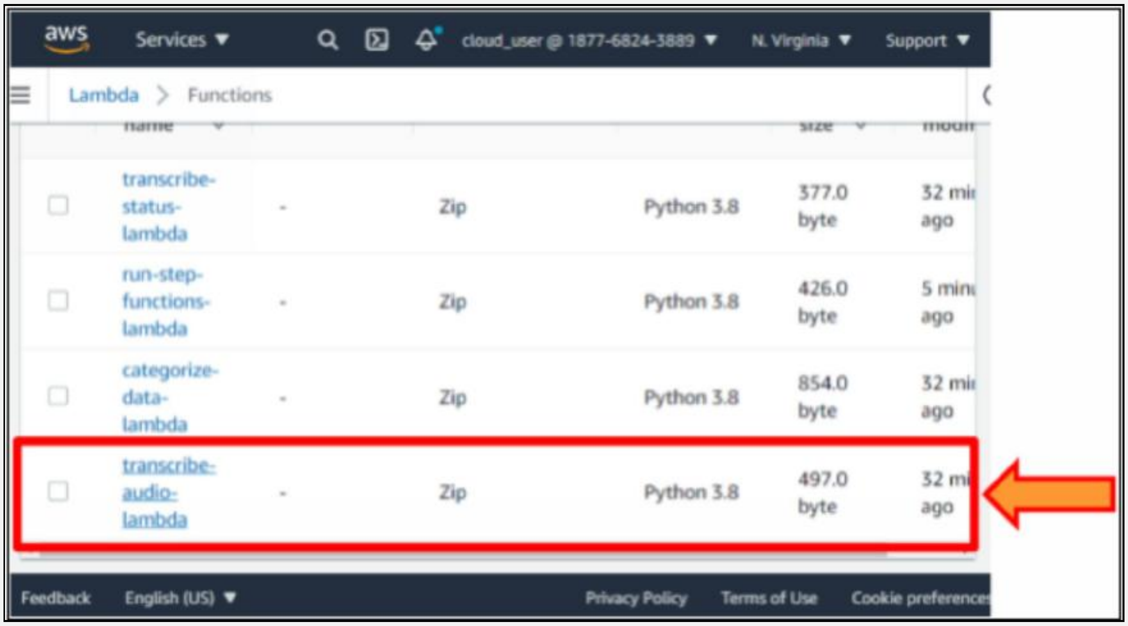


Hence, you have successfully created the model with Step Function.

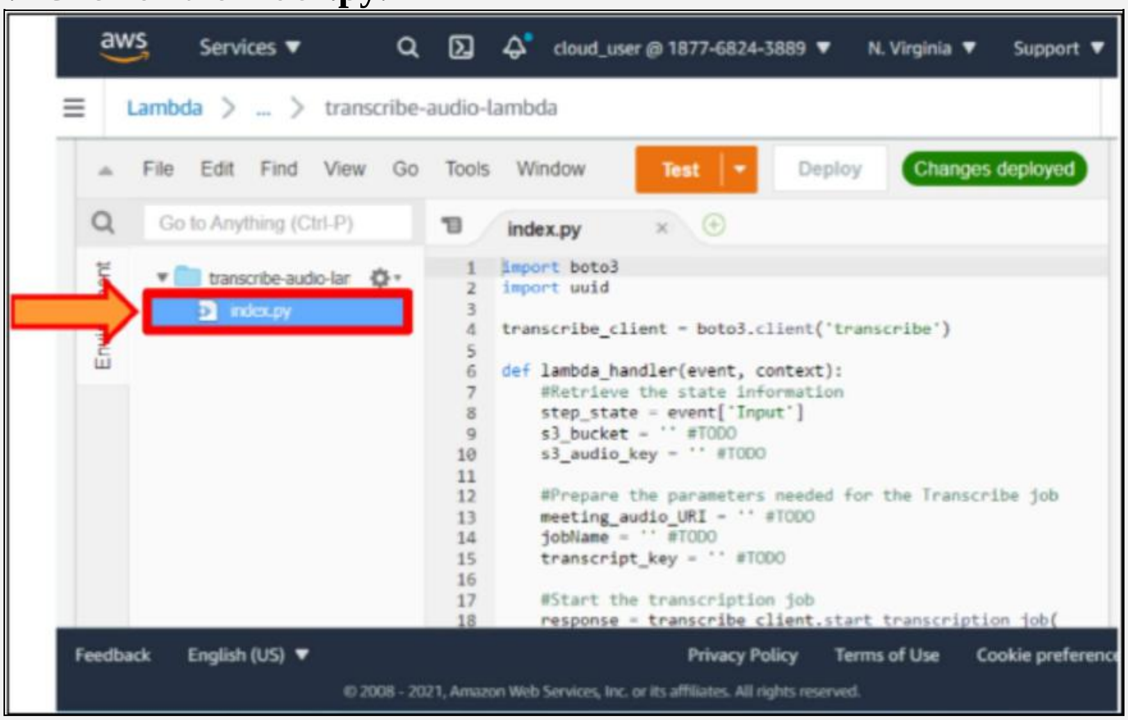


Step 6: Create the Lambda Business Logic

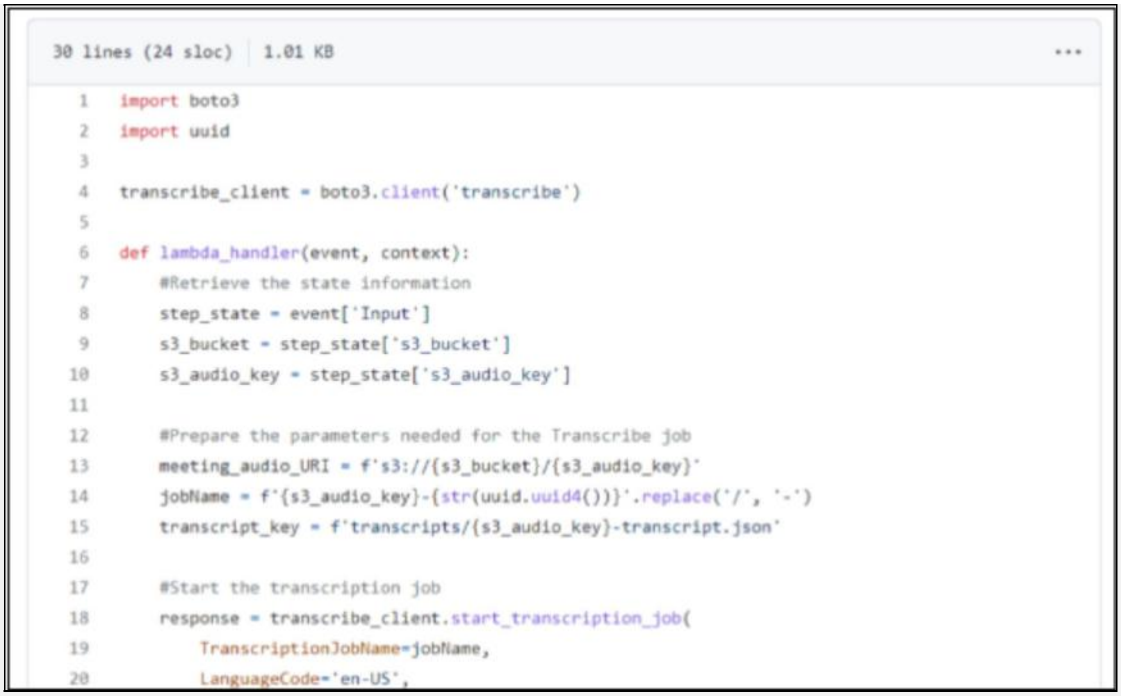
1. Go back to the **Lambda Function** dashboard.
2. Write the Transcribe Audio Function.
3. Click on the **transcribe-audio-lambda**.



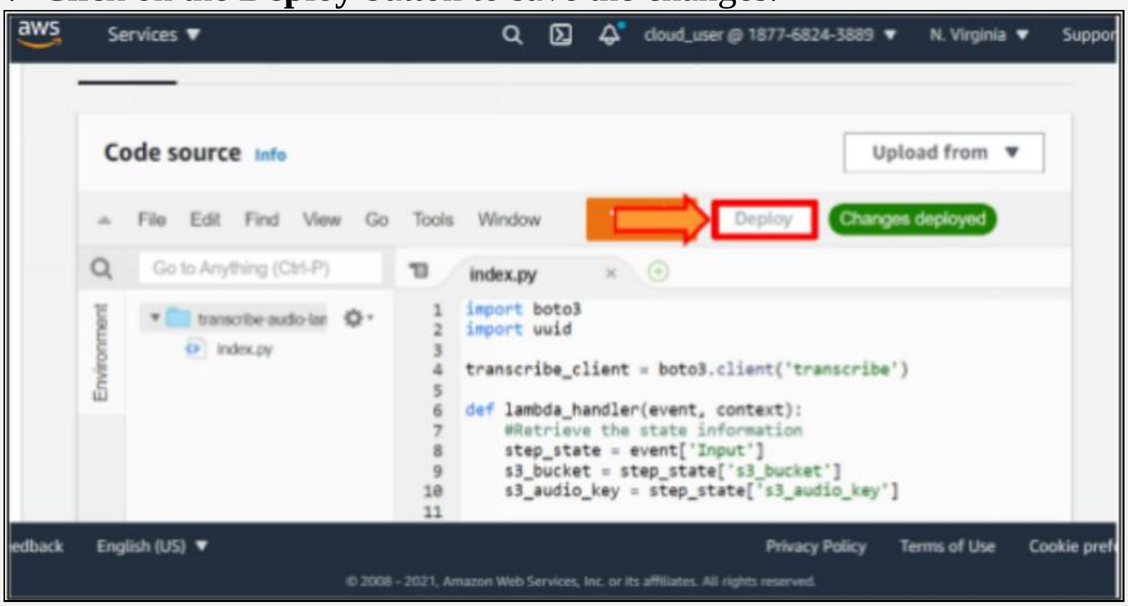
4. Click on the **index.py**.



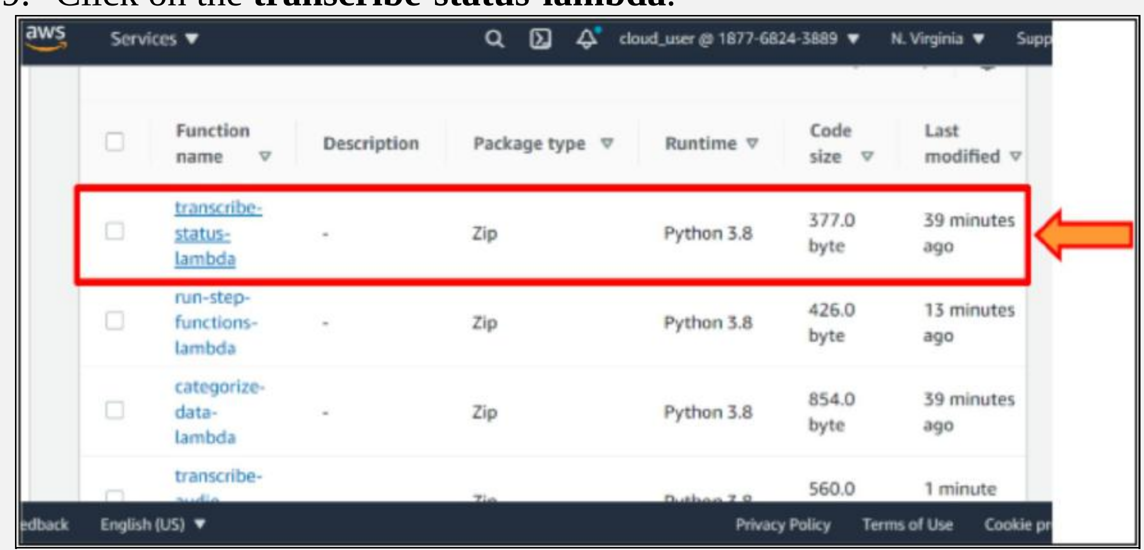
5. Copy the code from the following Github link: https://github.com/linuxacademy/content-aws-mls-c01/blob/master/CategorizeDataUploadsUsingStepFunctions/transcribe-audio-lambda.py



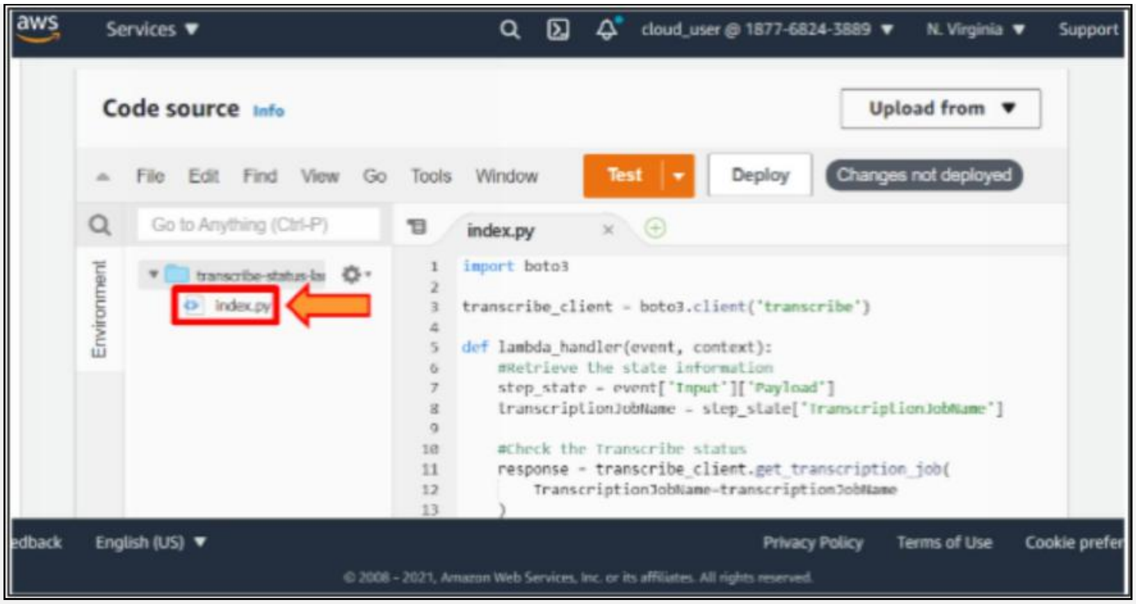
6. Paste the code into the Lambda Function code editor.
7. Click on the **Deploy** button to save the changes.



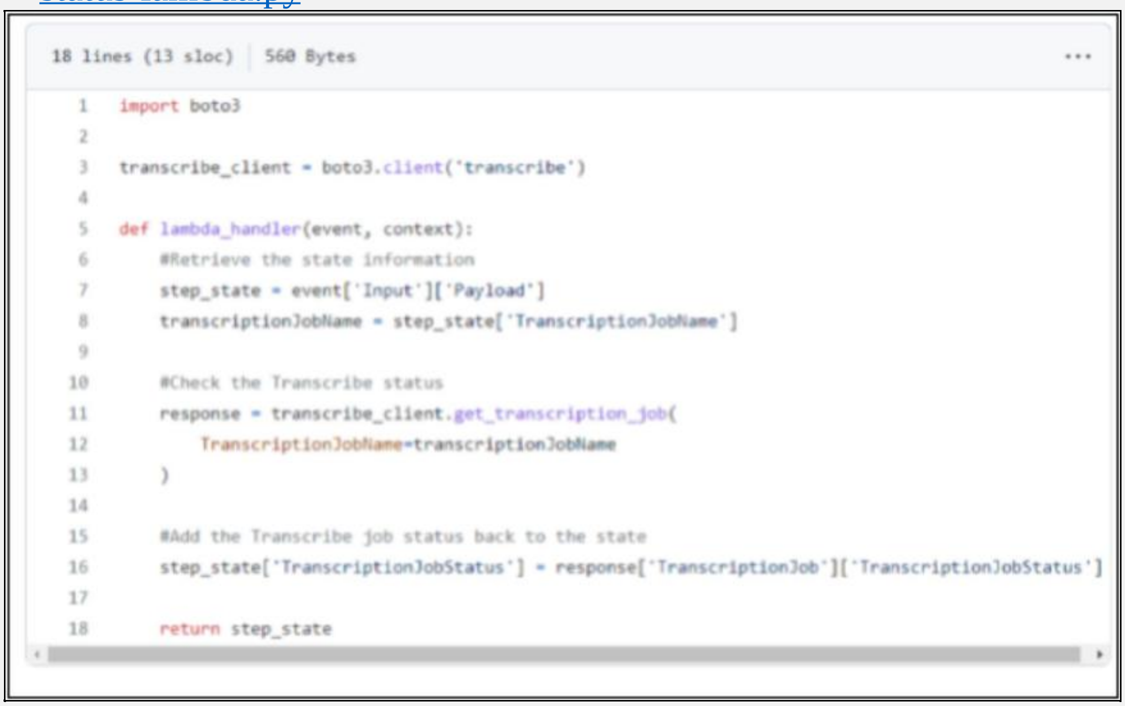
8. Write the Transcribe Status Function.
9. Click on the **transcribe-status-lambda**.



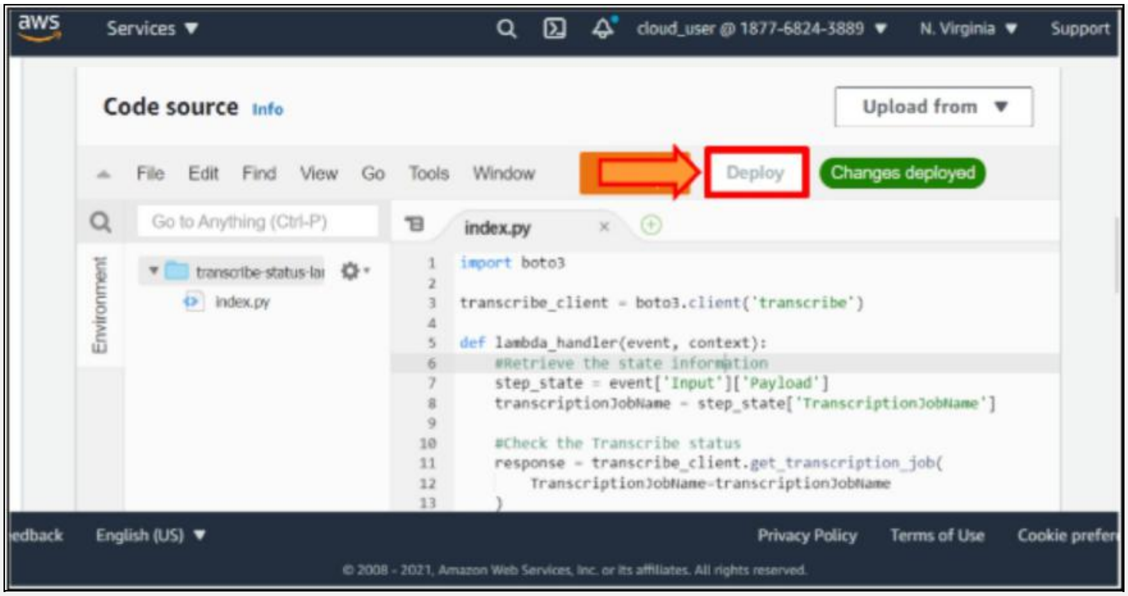
10. Click on the **index.py**.



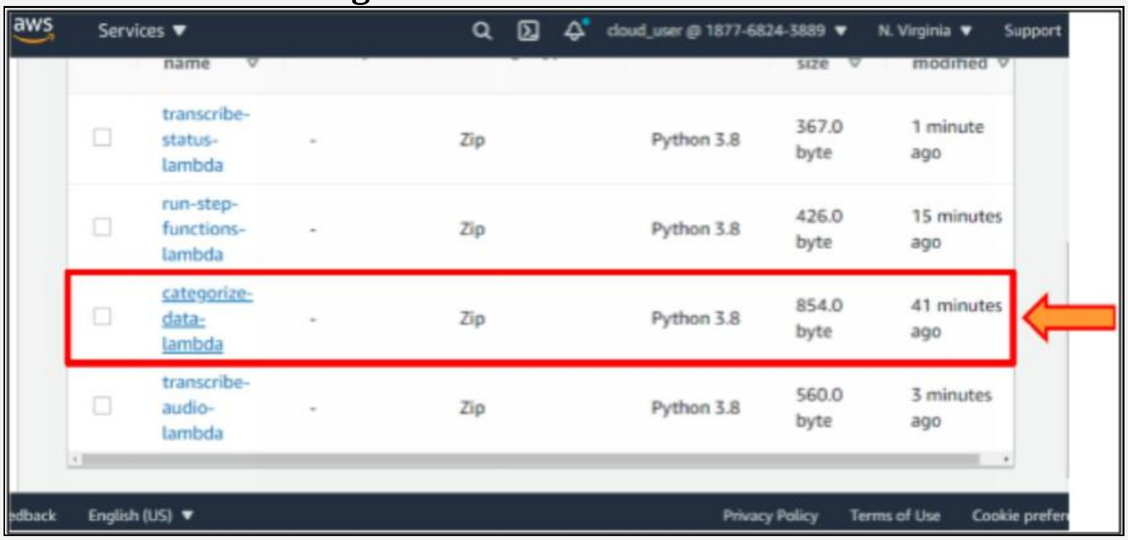
11. Copy the code from the following Github link: https://github.com/linuxacademy/content-aws-mls-c01/blob/master/CategorizeDataUploadsUsingStepFunctions/transcribe-status-lambda.py



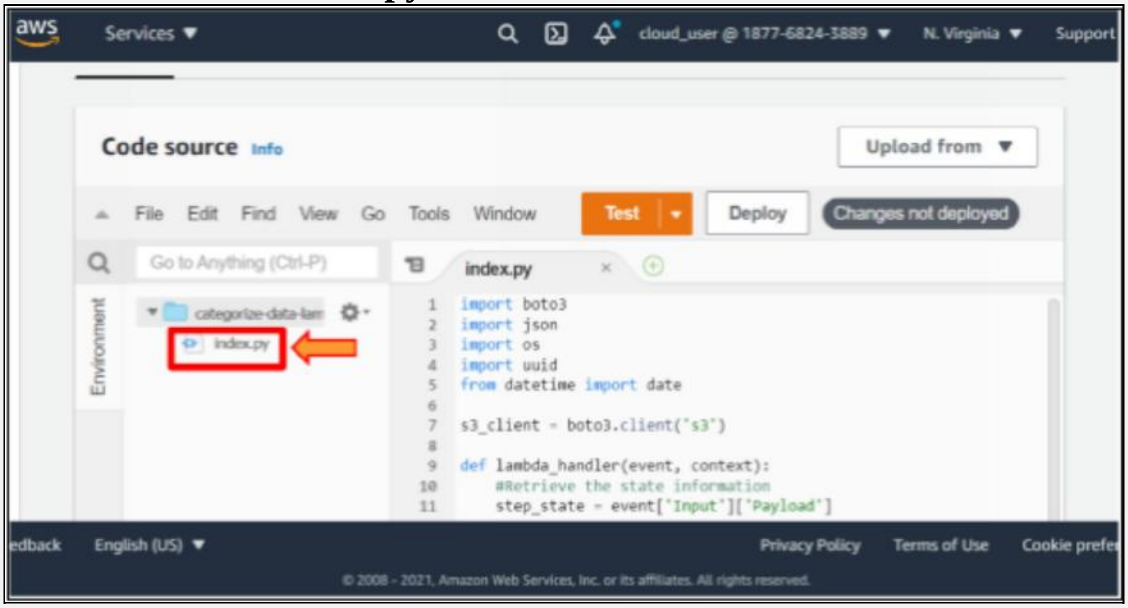
12. Paste the code into the Lambda Function code editor.
13. Click on the **Deploy** button to save the changes.



14. Write the Categorize Data Function.
15. Click on the **categorize-data-lambda.**



16. Click on the **index.py.**

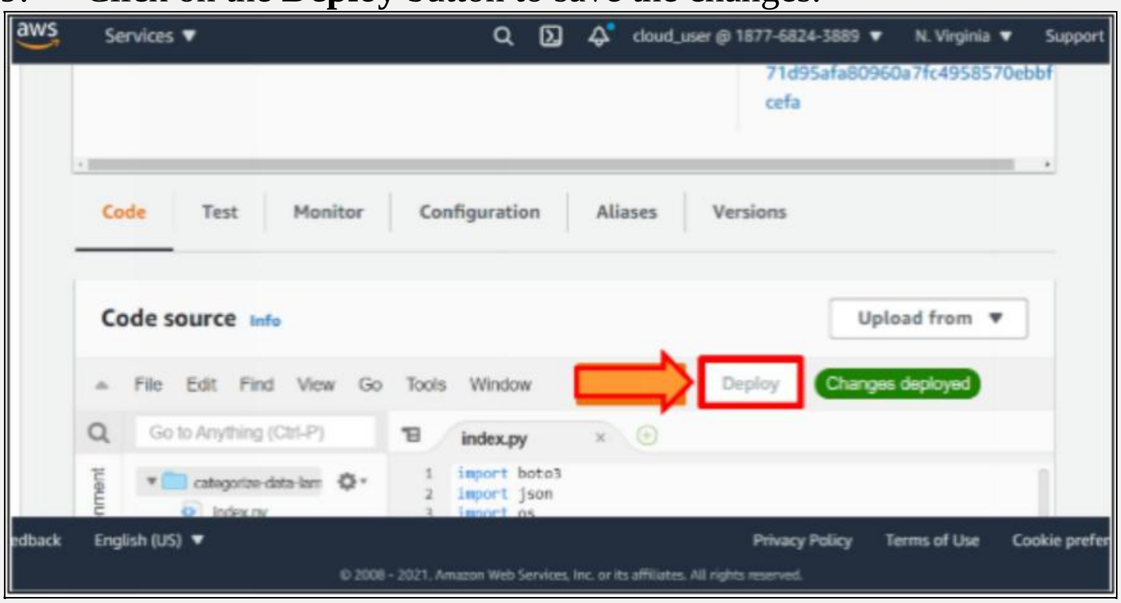


17. Copy the code from the following Github link: https://github.com/linuxacademy/content-aws-mls-c01/blob/master/CategorizeDataUploadsUsingStepFunctions/categorize-data-lambda.py

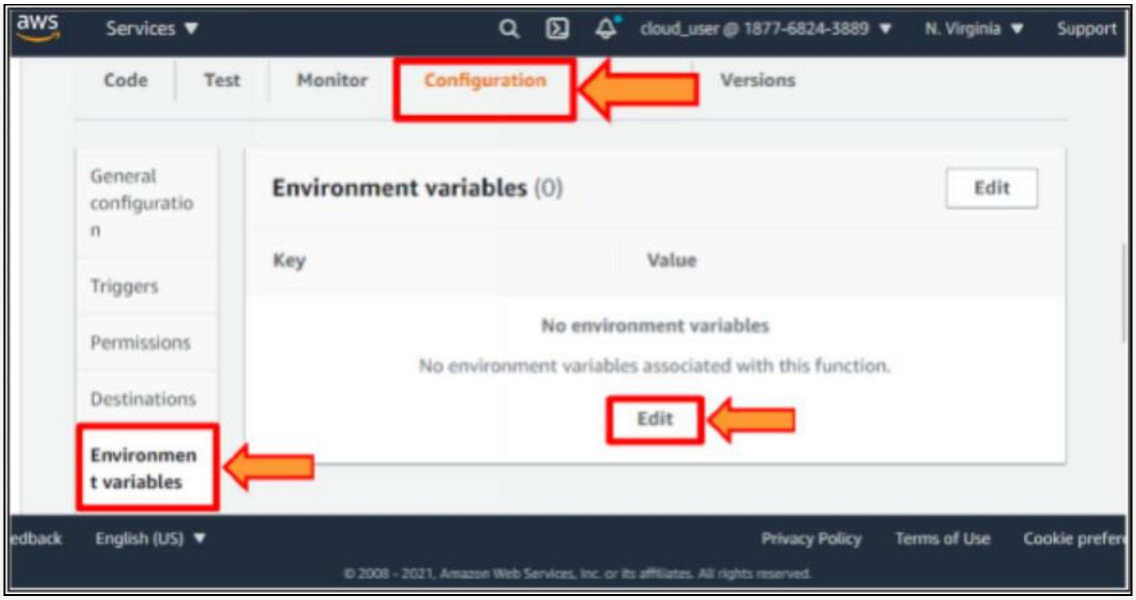
66 lines (54 sloc) | 2.36 KB

```
import boto3
import json
import os
import uuid
from datetime import date

s3_client = boto3.client('s3')

def lambda_handler(event, context):
    #Retrieve the state information
    step_state = event['Input']['Payload']

    #Get the S3 information for the audio file and transcript
    s3_bucket = step_state['s3_bucket']
    s3_audio_key = step_state['s3_audio_key']
    s3_transcript_key = step_state['transcript_key']

    base_audio_key = os.path.basename(s3_audio_key)
    base_transcript_key = os.path.basename(s3_transcript_key)

```

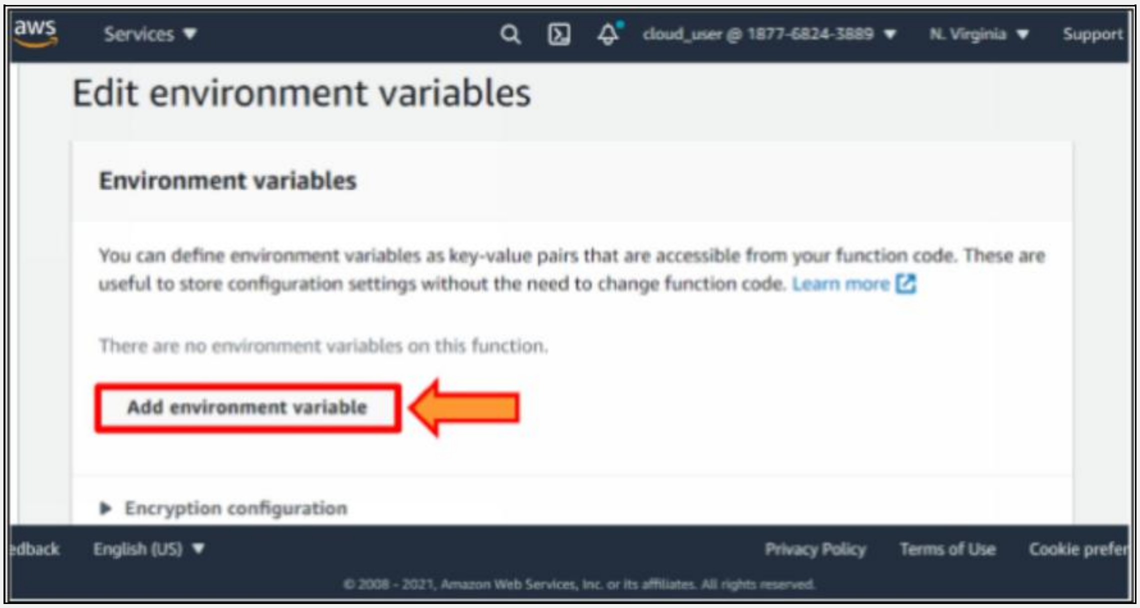
18. Paste the code into the Lambda Function code editor.
19. Click on the **Deploy** button to save the changes.



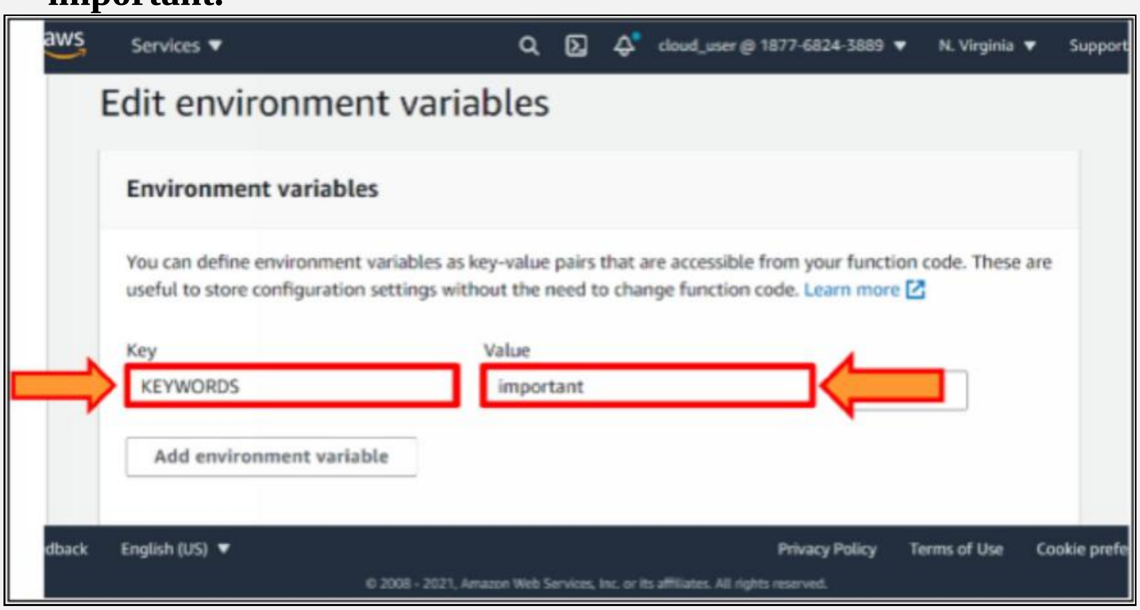
20. Click on the **Configuration tab.**
21. Click on the **Environment variables.**
22. Click on the **Edit** button.



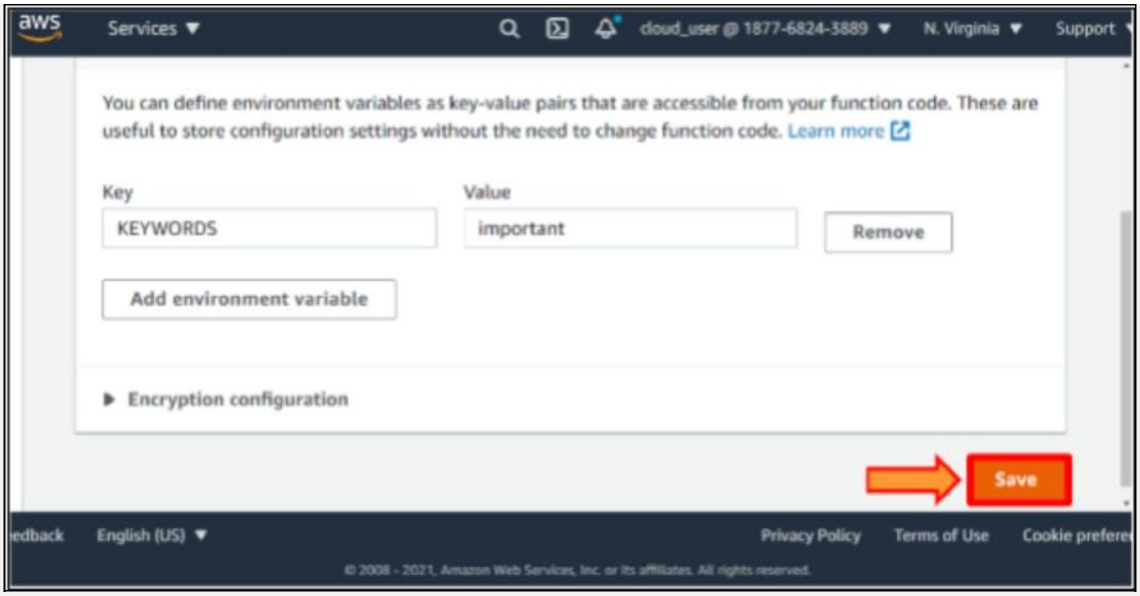
23. Click on the **Add environment variable** button.



24. Under Key, enter **KEYWORDS**, and under Value, enter **important.**

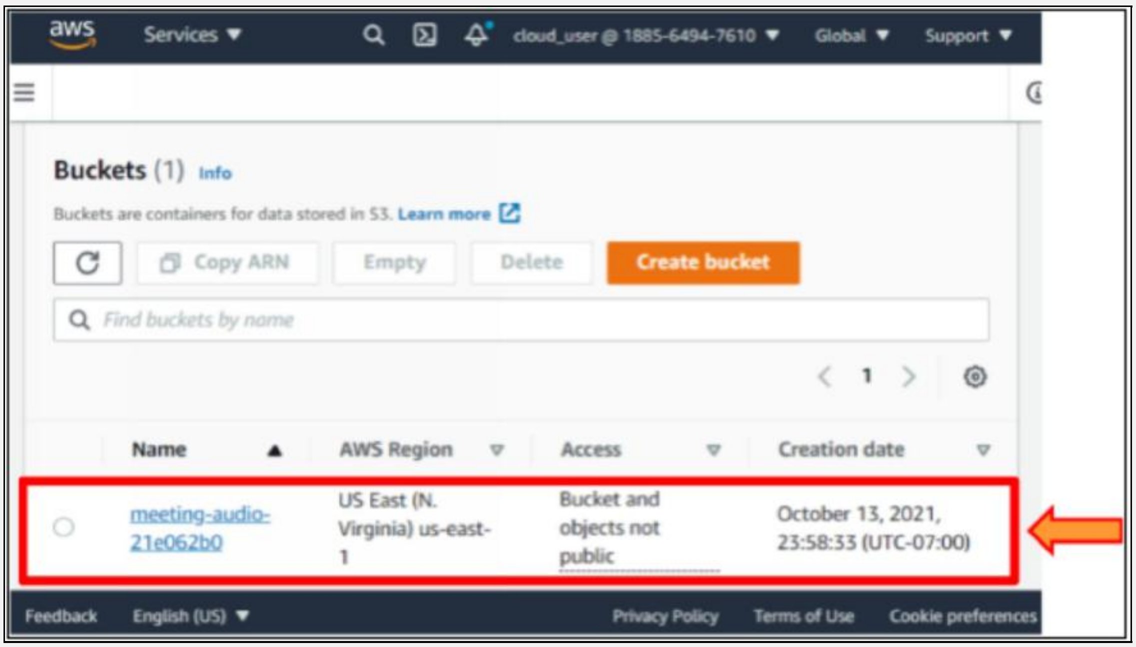


25. Click on the **Save** button.

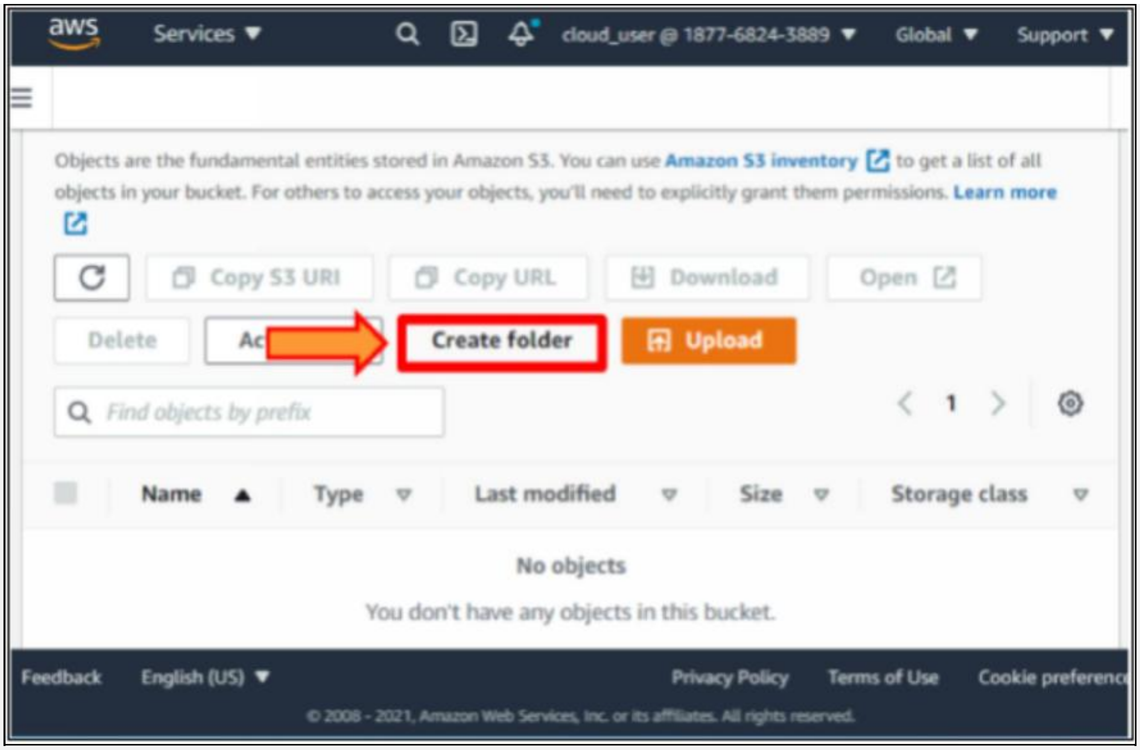


Step 7: Categorize Audio Data

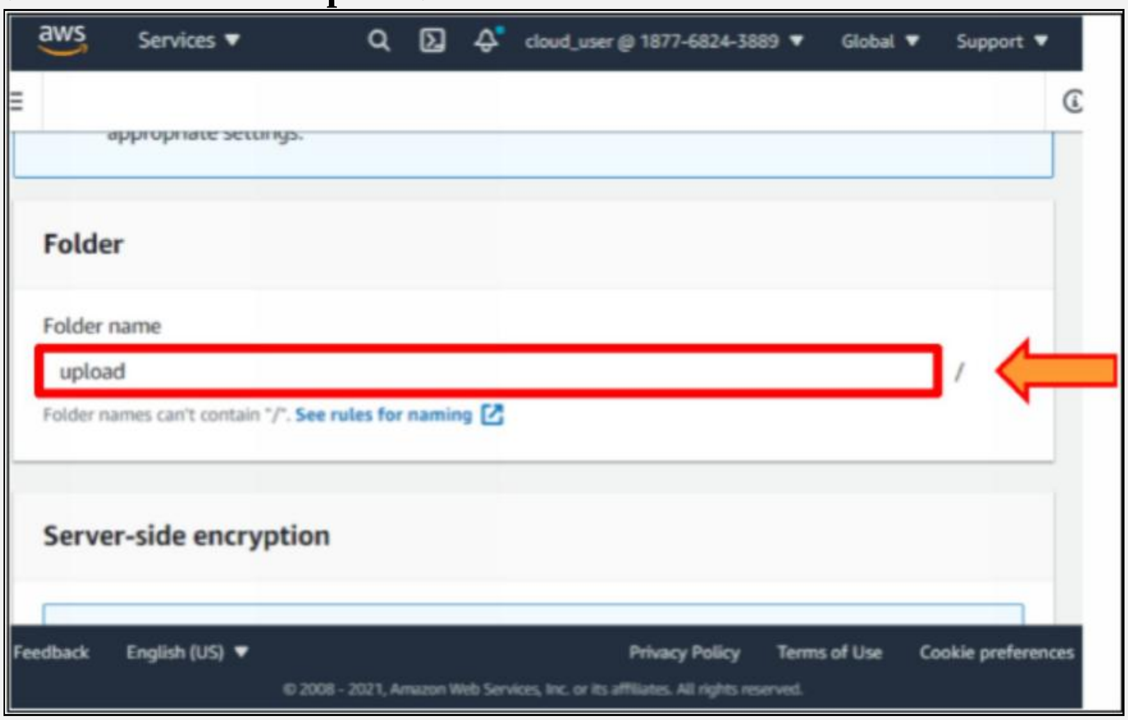
1. Go back to the **S3 Bucket** console.
2. Click on the bucket.



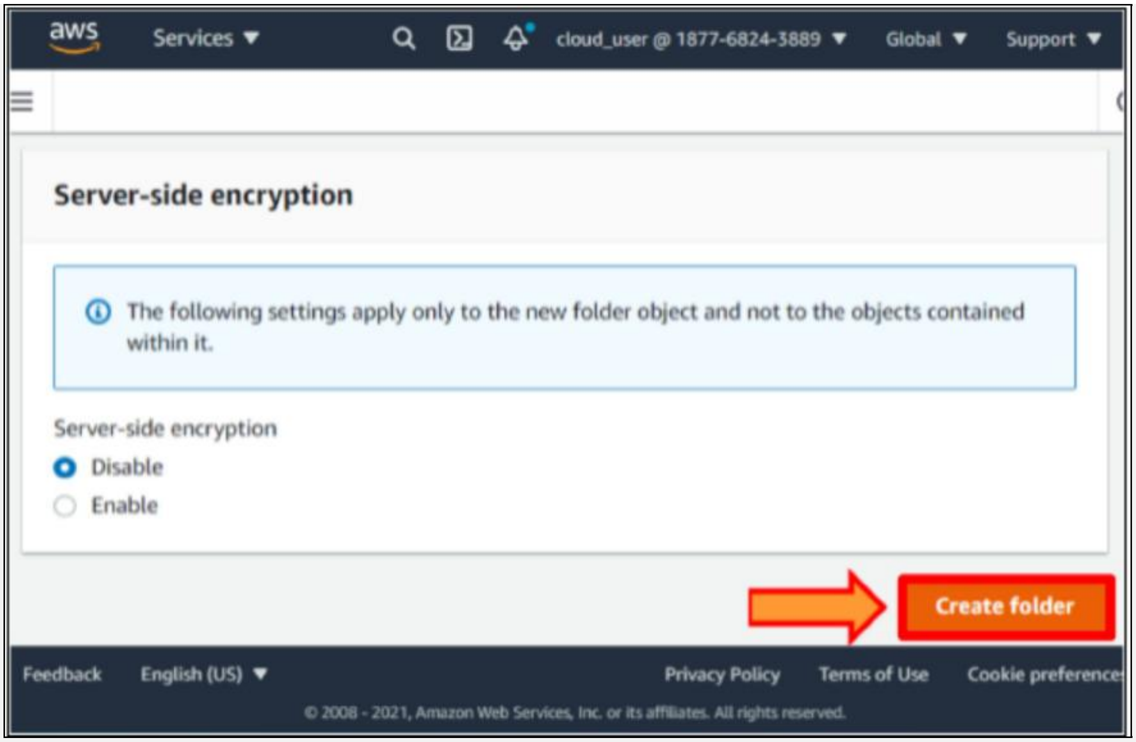
3. Click on the **Create folder** button.



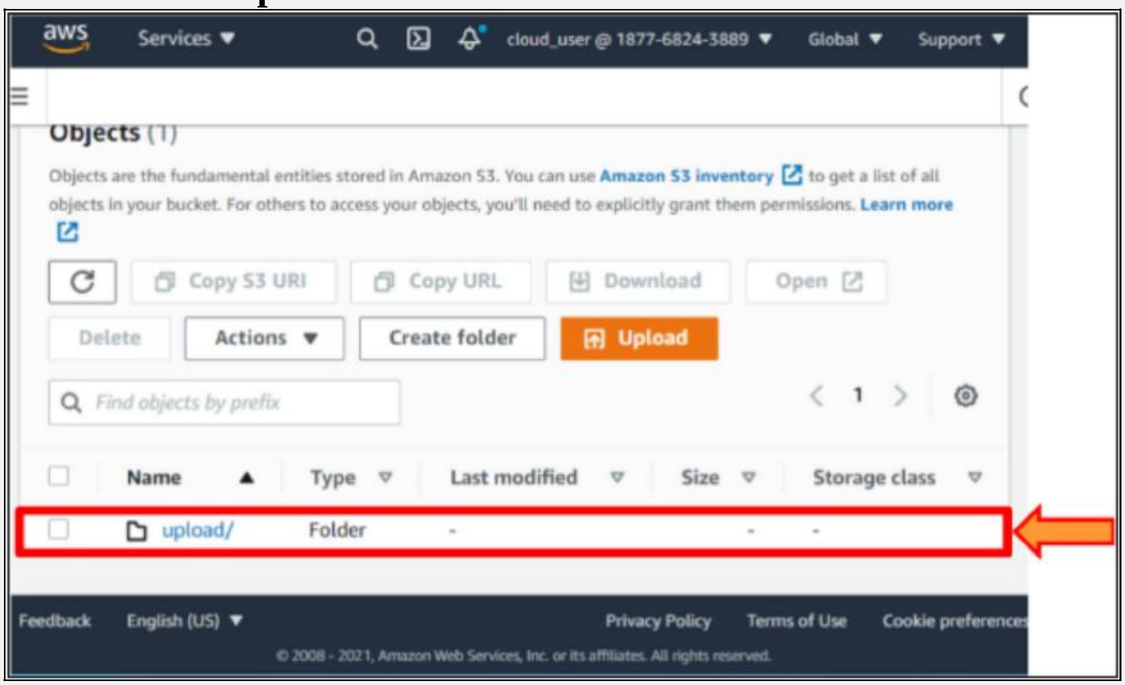
4. Name the folder **Upload**.



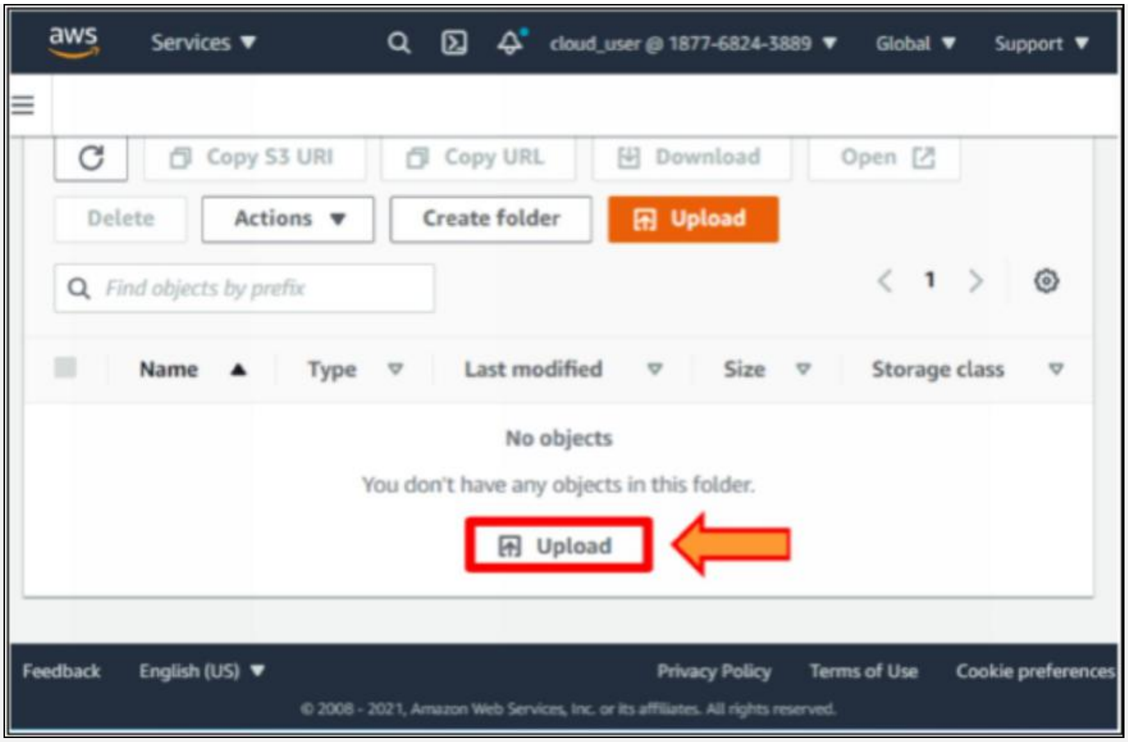
5. Click on the **Create folder** button.



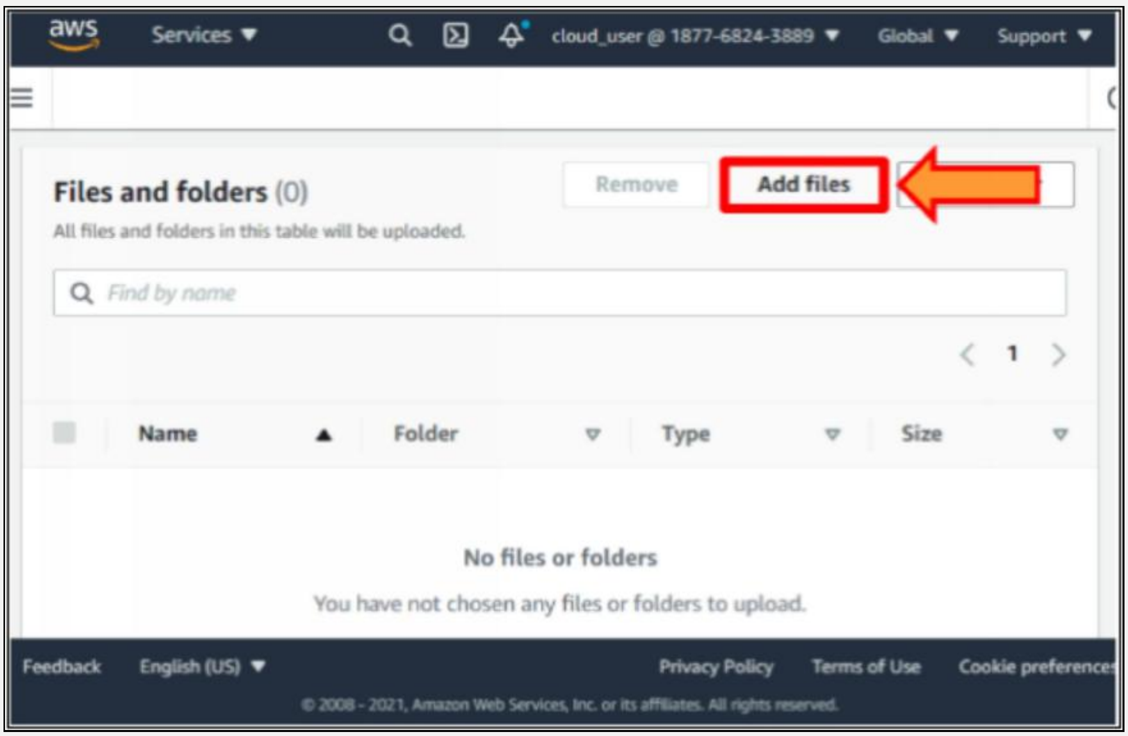
6. Click on the **upload/** folder.



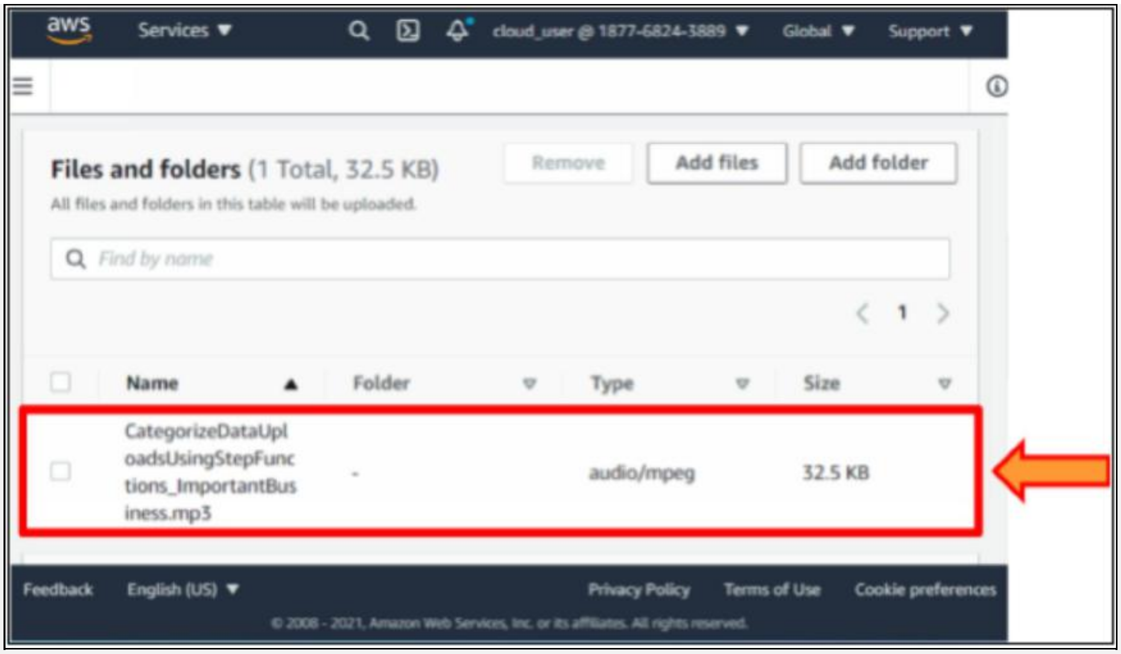
7. Click on the **Upload** button.



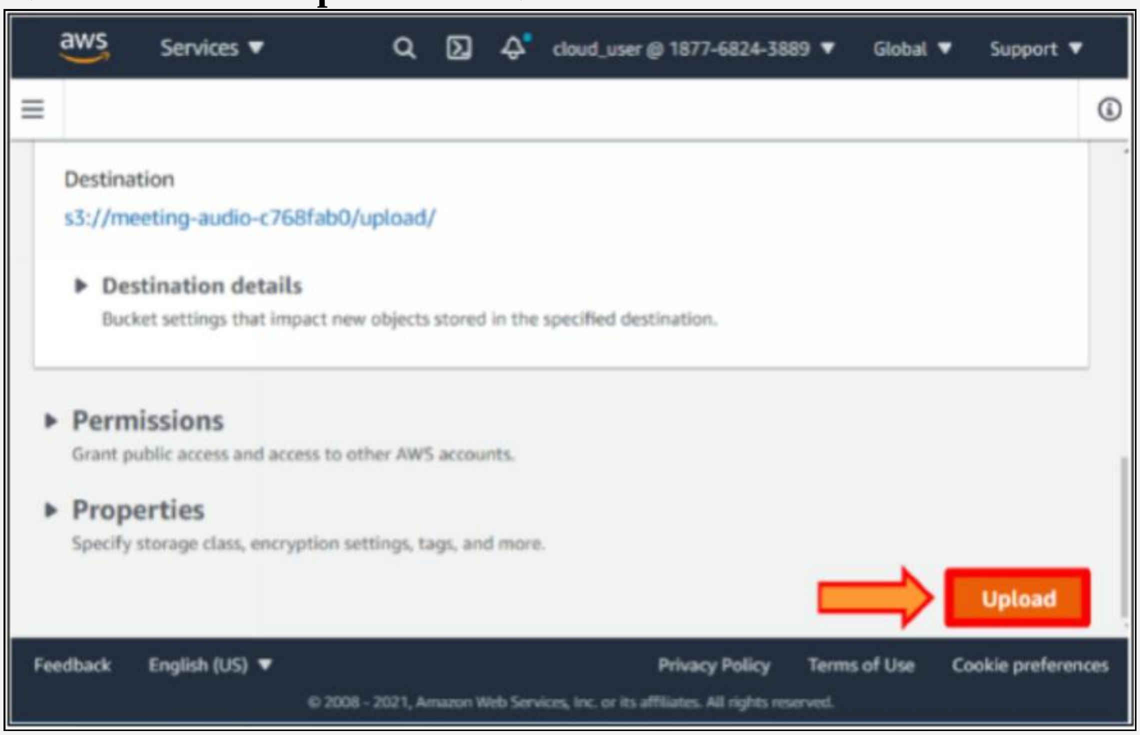
8. Click on the **Add files** button.



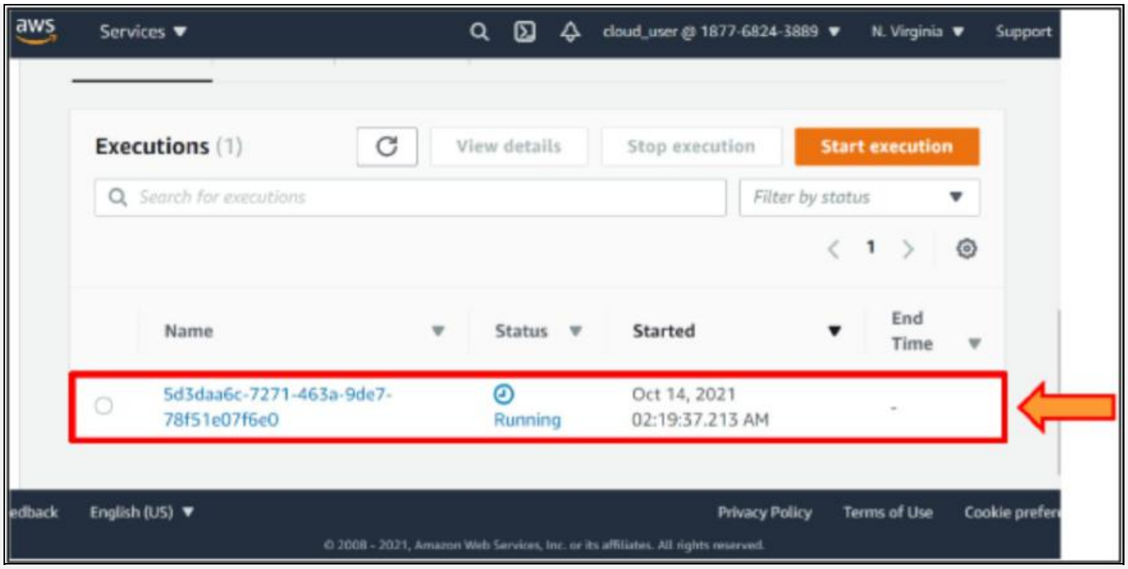
9. Select an mp3 file to upload. Download the ImportantBusiness.mp3 file from the GitHub link: https://github.com/linuxacademy/content-aws-mls-c01/tree/master/CategorizeDataUploadsUsingStepFunctions



10. Click on the **Upload** button.

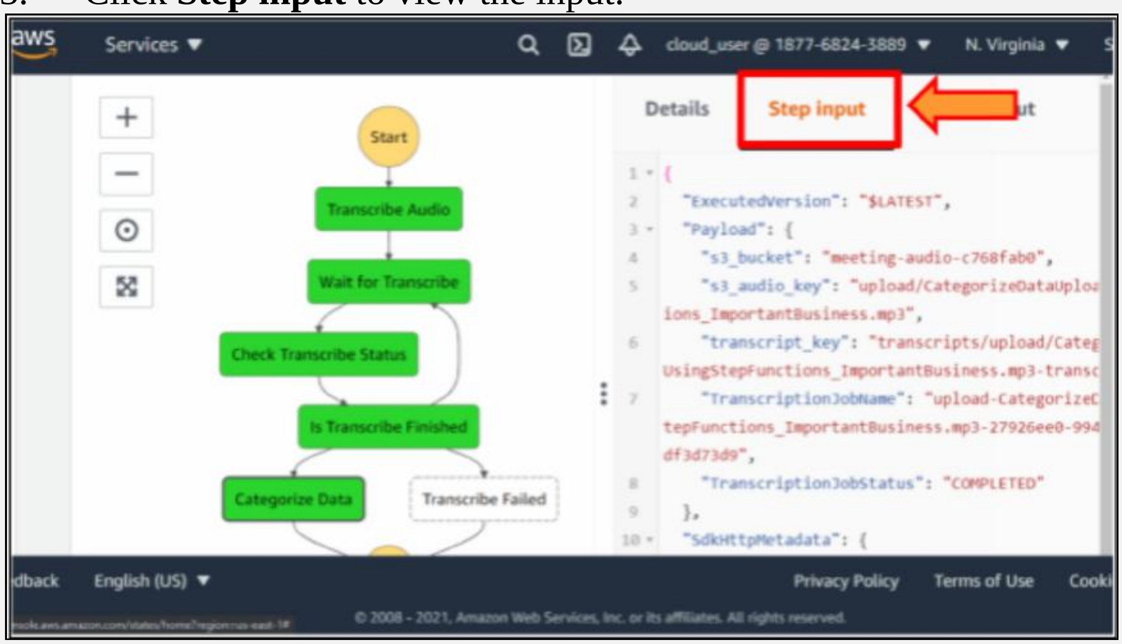


11. Go back to the **Step Function** dashboard. The step function should now be running.



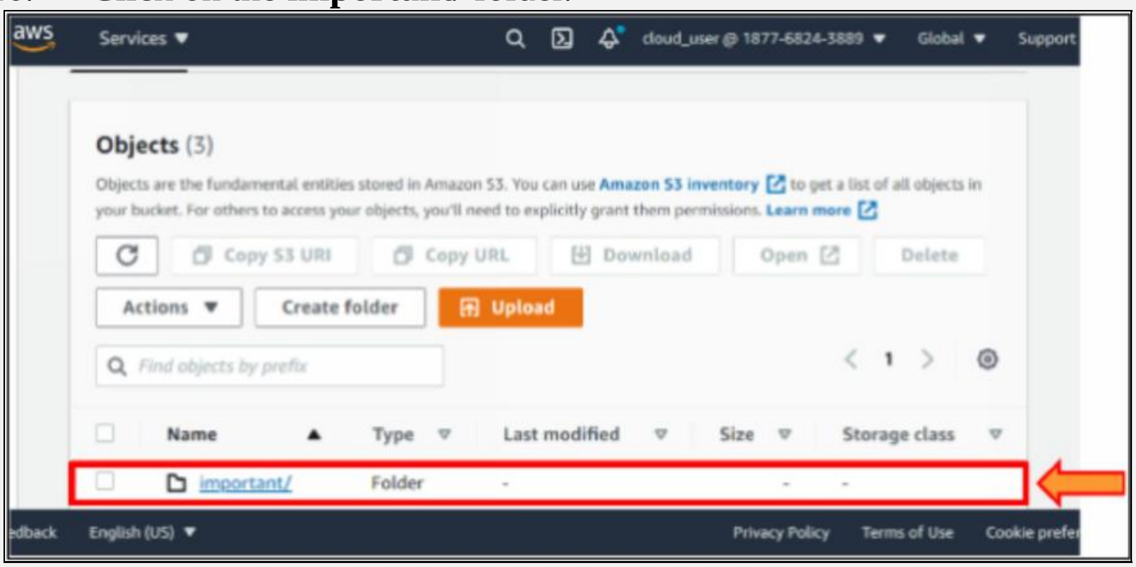
12. Click on the **step function** name. You will see the model graph.

13. Click **Step input** to view the input.

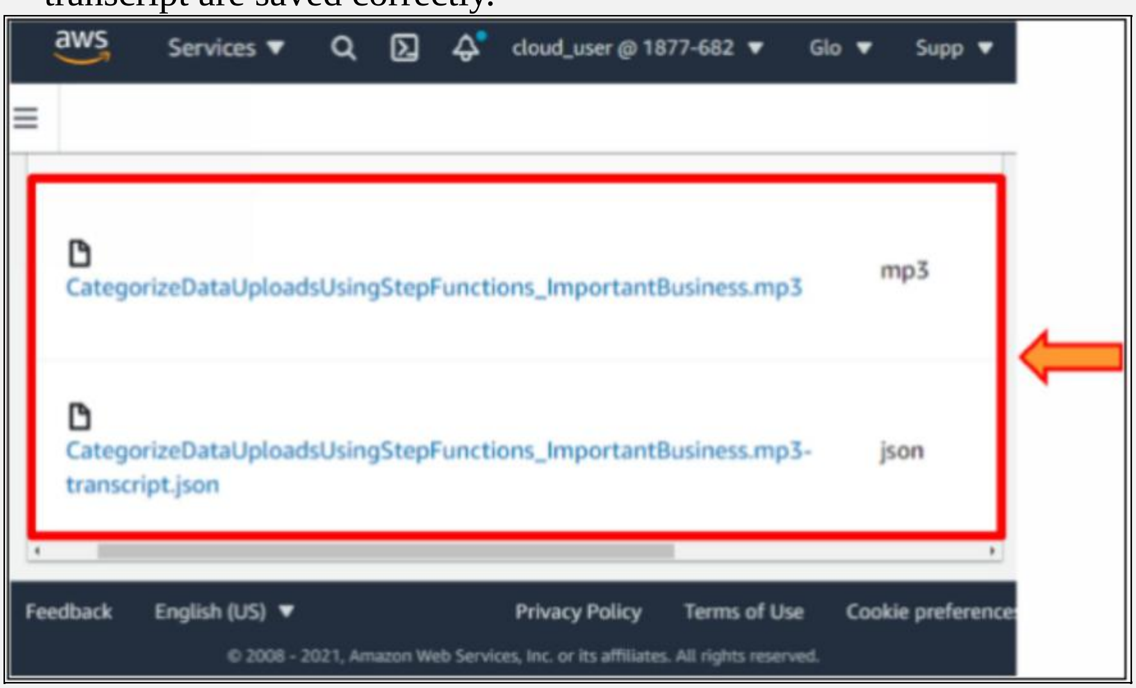


14. Click **Step output** to view the output of the step function.

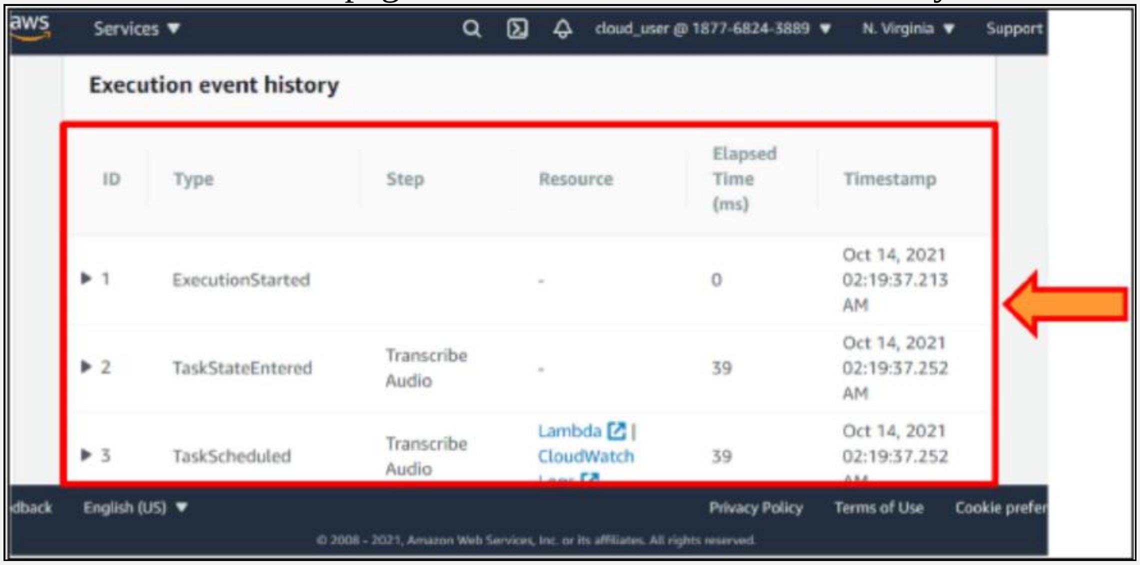
15. Go back to the **S3 Bucket** dashboard.
16. Click on the **important/** folder.



17. Click the **year**, **month**, and **day** folder and confirm the mp3 and transcript are saved correctly.



18. Go back to the **Step Function** tab.
19. Scroll down the page to find the execution event history.



Lab 16: Coordinating AI Services with Step Functions

Certifications	AWS Machine Learning - Specialty
Cloud Provider	AWS
Time Required	1 Hr.
Tags	Lambda, AWS

Lab Prerequisites

- Familiarity with basic AWS Cloud Computing concepts and terminology
- An AWS account with an active subscription

Service Introduction

AWS Lambda

AWS Lambda allows you to run code without creating or managing servers. There is no charge when your code is not executing; you only pay for the compute time you use. You can run code for nearly any application or backend service with Lambda, and you do not have to worry about administration. Upload your code, and Lambda will handle everything necessary to run and grow it with high availability. You may configure your code to be automatically triggered by other AWS services, or you can access it directly from any computer or smartphone app.

AWS Simple Storage S3

Amazon S3 is a type of object storage that allows you to store and recover any quantity of data from any location. It is a low-cost storage solution with business resilience, reliability, efficiency, privacy, and infinite expansion.

Amazon S3 provides a simple web service interface for storing and retrieving any quantity of data from any place at any time. You may quickly create projects that integrate cloud-native storage using this service. Because Amazon S3 is easily customizable and you only pay for what you use, you can start small and scale up as needed without sacrificing performance or dependability.

Amazon S3 is also built to be highly adaptable. Instead of finding out how to store their data, Amazon S3 allows developers to focus on innovation. They can build a simple FTP system or a complex web application like the amazon.com retail website; read the same piece of data a million times or only for emergency disaster recovery; store whatever type and amount of data desired.

AWS Translate

Amazon Translate is a text translation service that uses Neural Machine Translation (MT) to translate text between supported languages. The service, powered by deep learning technologies, delivers high-quality, inexpensive, and configurable language translation, allowing developers to solve business and user-authored content and construct applications that require multilingual support. The service can be accessed using an API, which allows for real-time or batch translation of the text from one language to another.

AWS Transcribe

Conduct a (text-based) content analysis on audio/video content using Automatic Speech Recognition (ASR) technology. Amazon Transcribe is an AWS service that makes converting speech to text simple for customers. Customers may utilize Amazon Transcribe for various commercial applications, such as voice-based customer care call transcription and the production of subtitles for audio/video material.

AWS Comprehend

Amazon Comprehend is a Natural Language Processing (NLP) service that searches the text for meaning and insights using machine learning. You can use Amazon Comprehend to extract key terms, places, people, brands, or events from a library of documents, understand sentiment about products or services, and identify the primary topics. This text could have come from various places, including online pages, social media feeds, emails, and newspapers. You can also input Amazon Comprehend, a collection of text documents. It will pick subjects (or groups of words) that best reflect the data. The output of Amazon Comprehends may be used to evaluate consumer comments better, provide a search experience via search filters, and arrange themes.

AWS Polly

Polly is an Amazon service that converts text into natural-sounding speech. Amazon Polly enables existing apps to speak as a first-class feature and new categories of speech-enabled goods, such as mobile apps, automobiles, gadgets, and appliances. Amazon Polly comes with dozens of lifelike voices and multilingual capabilities, allowing you to pick the perfect voice and distribute your speech-enabled apps across many locations. Amazon Polly is simple to use: transmit the text you want to be translated into speech to the Amazon Polly API. Amazon Polly will return the audio stream to your application. You can play or save it in a regular audio file format like MP3. You can change the speech pace, pitch, or volume using Amazon Polly because it supports Speech Synthesis Markup Language (SSML) tags like prosody. Amazon Polly is a safe service that provides these advantages at low latency and on a large scale. Amazon Polly's generated speech can be cached and replayed at no extra charge. When you sign up for Amazon Polly, you can convert millions of characters per month for free for the first year. Amazon Polly is a cost-effective solution to enable speech synthesis everywhere because of its pay-as-you-go pricing, low cost per request, and lack of constraints on storage and reuse of voice output. You can utilize Amazon Polly to provide high-quality spoken work for your app. This low-cost service provides quick response times and may be used for almost any application, with no constraints on storing and reusing created speech.

Amazon Step Function

AWS Step Functions is a fully managed solution that makes it simple to coordinate the components of distributed applications and microservices using visual workflows. You may easily grow and alter applications by creating them from separate components that execute a specific function. Step Operations is a dependable method for coordinating components and stepping through your application's functions. Step Functions provides a graphical interface for organizing and visualizing your application's components as a series of stages. It makes creating and running multi-step apps a breeze. Step Functions automates the triggering and tracking of each step and retries when problems occur, ensuring that your application runs smoothly. Step Functions records the condition of each step so that you can immediately detect and troubleshoot problems if something goes wrong. You can alter and add stages without writing code, allowing you to grow and innovate your application quickly.

Case Study Enterprise Internet Television Network – Netflix

Business Challenge

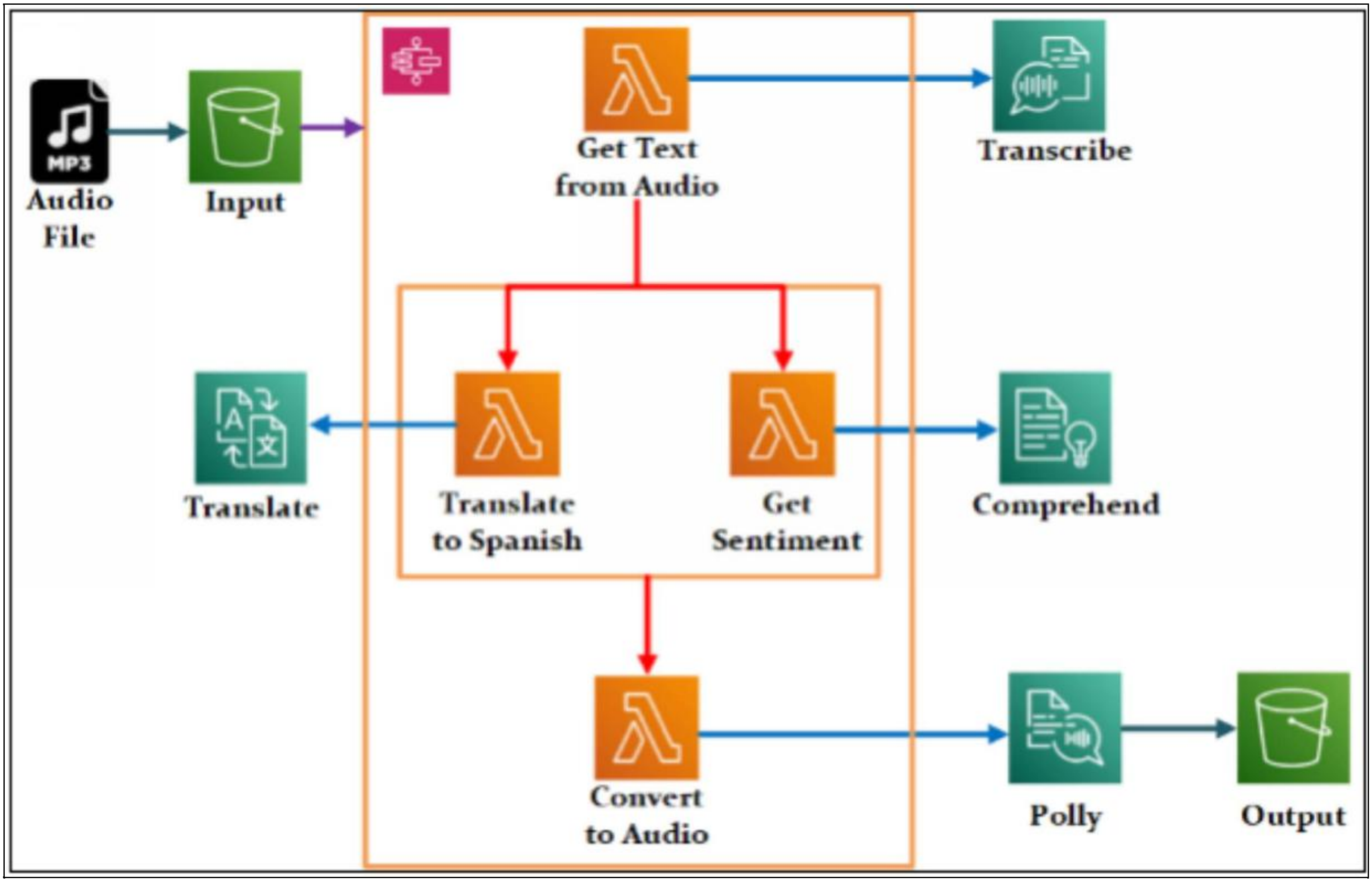
You are a Machine Learning developer at Netflix, developing content on different cloud computing courses. They want their content to be translated into the Spanish language. But there is an issue in translating the whole book of 300 to 500 pages by a content developer. The organization's requirement to complete the book in one week is impossible. They gave you the task of building a Natural Language Processing (NLP) model from scratch using Tensorflow or Pytorch framework. It requires lots of data on the Spanish and English languages. It also needs expensive hardware like GPU to train the model. Building the model from scratch also requires extensive time and cost. Hence, your task is to develop a Neural Machine Translation model with less time and a low budget. How will you automate this task?

Proposed Solution

The solution is simply to design a Machine Learning pipeline using AWS Lambda functions in the background to run the logic of the channel. But you can also use AWS S3 for input and output of data. You also use other Amazon helping services to automate this task. You use AWS Translate, AWS Transcribe, AWS Comprehend, and AWS Polly, as shown in the figure below.

Before deep-diving into the lab, create Lambda Function, Step Function, and S3 Bucket, as discussed in the above lab sections.

Lab Diagram



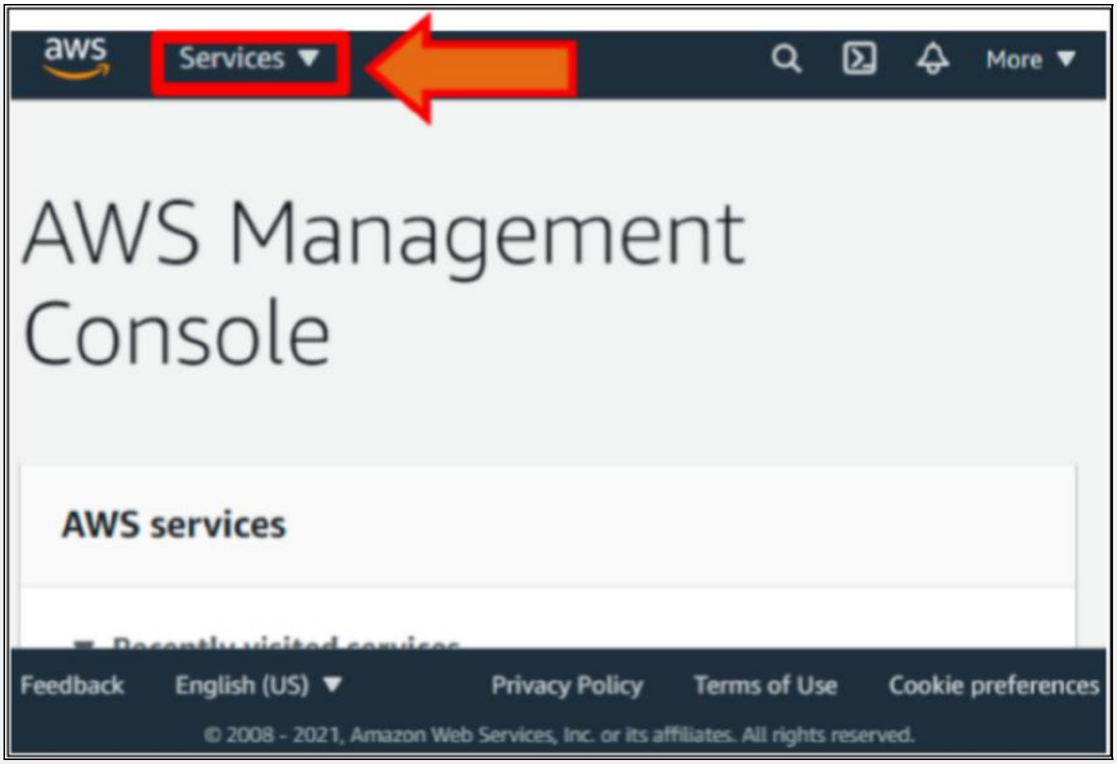
Implementation Steps

1. Create a Translation Pipeline.
2. Upload an Audio File on S3 Bucket
3. Translate Text on AWS Transcribe.

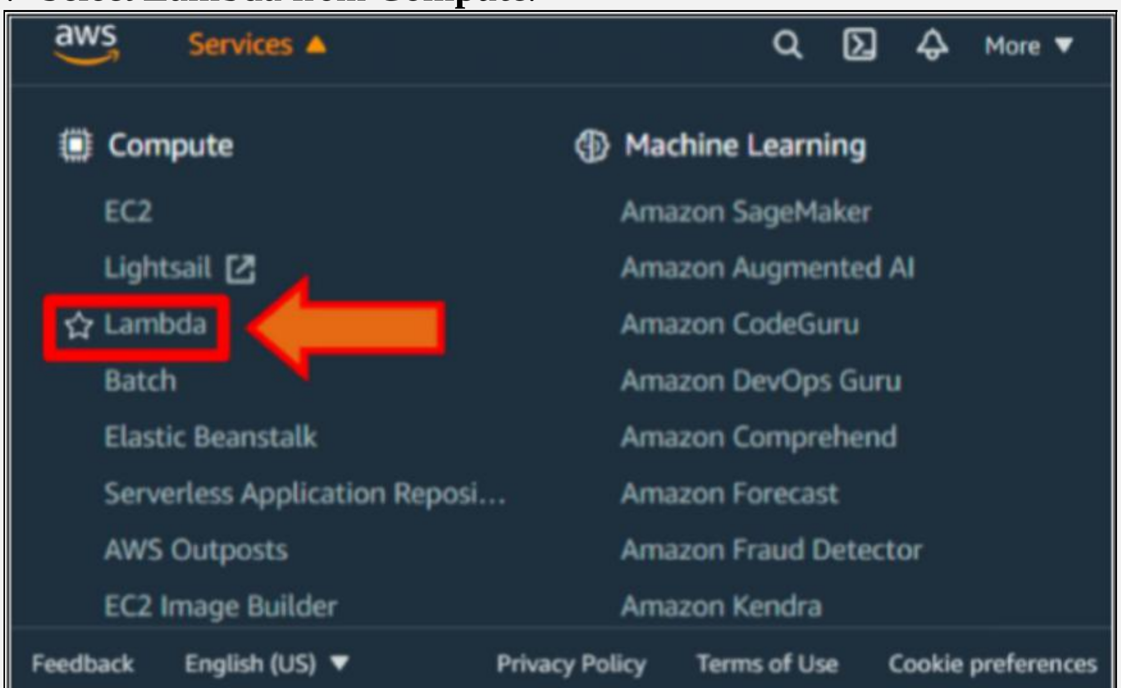
Solution

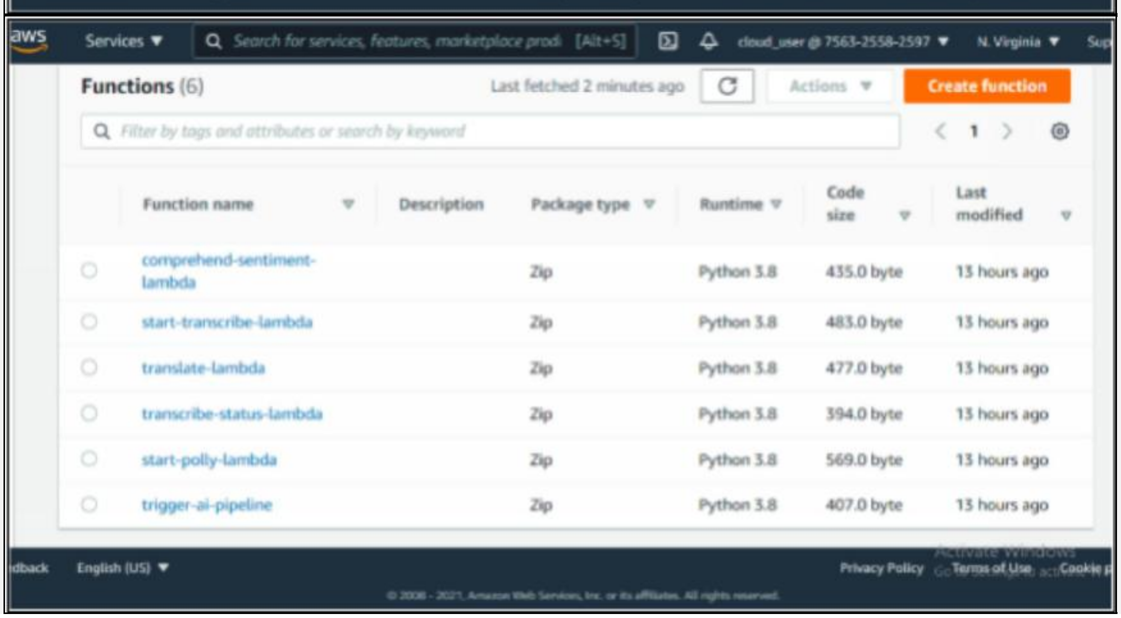
Step 1: Create Translation Pipeline

1. Log in to **AWS Console.**
2. Click on **Services.**

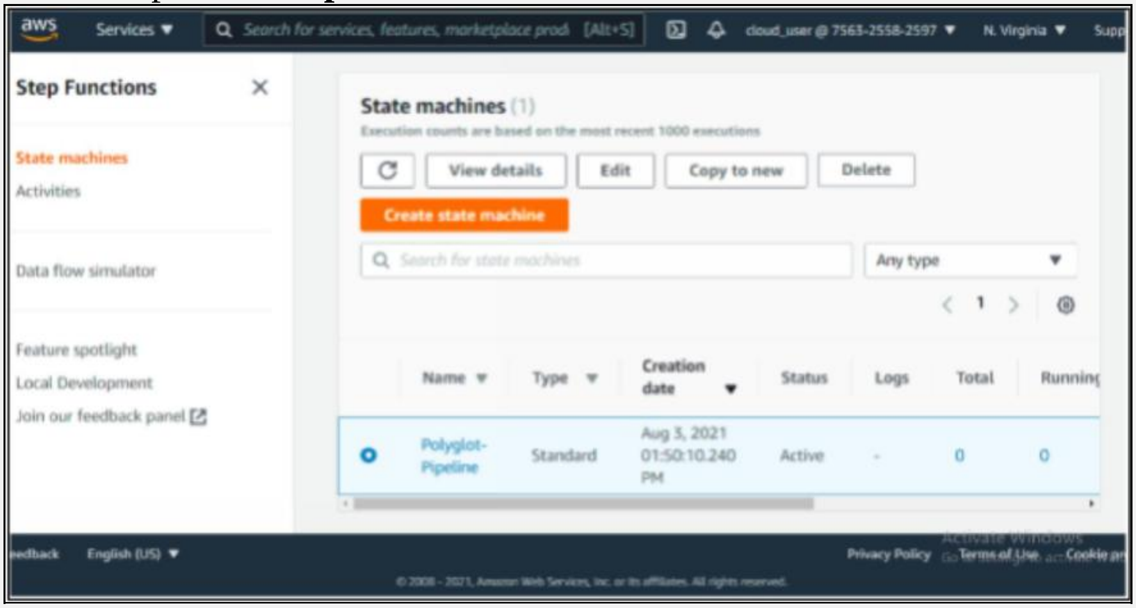


3. Select **Lambda** from **Compute**.

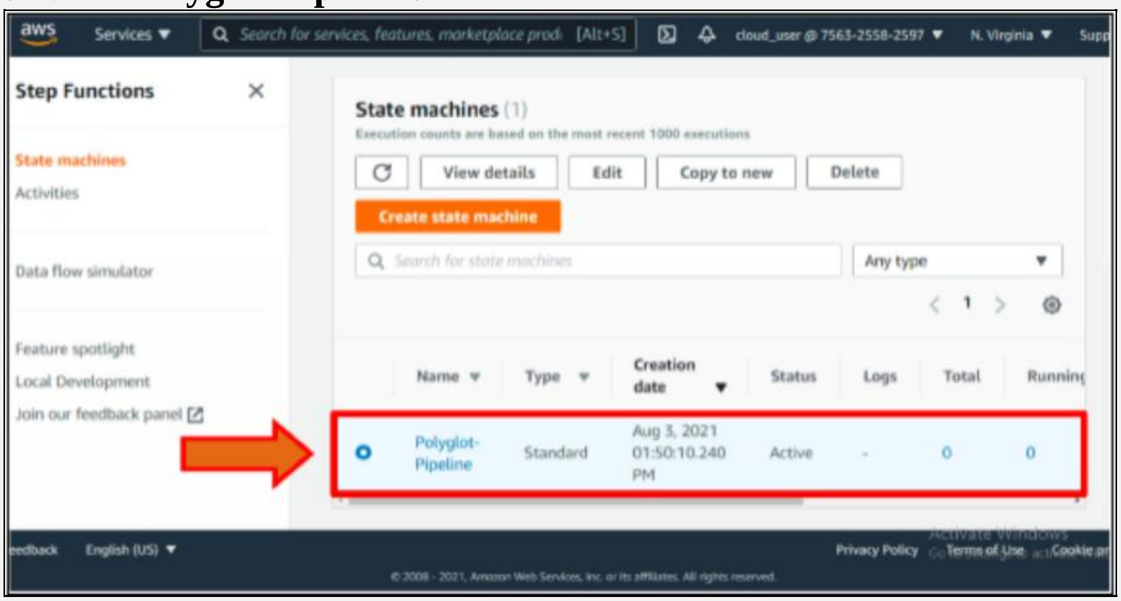




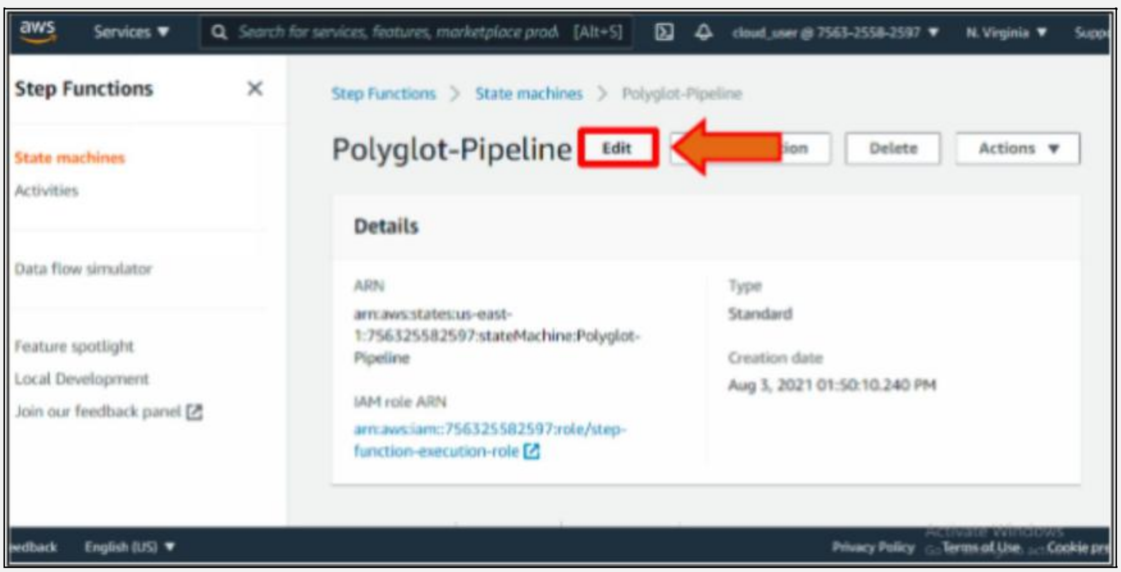
4. Now open the **Step Function** tab.

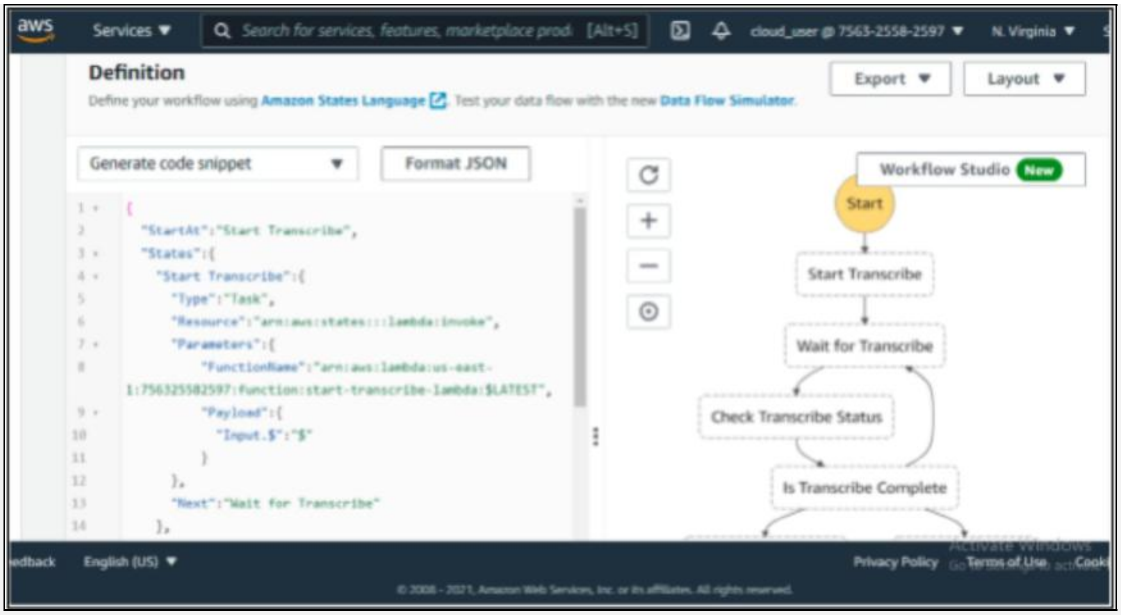


5. Add translation to the pipeline.
6. Click **Polyglot-Pipeline**.

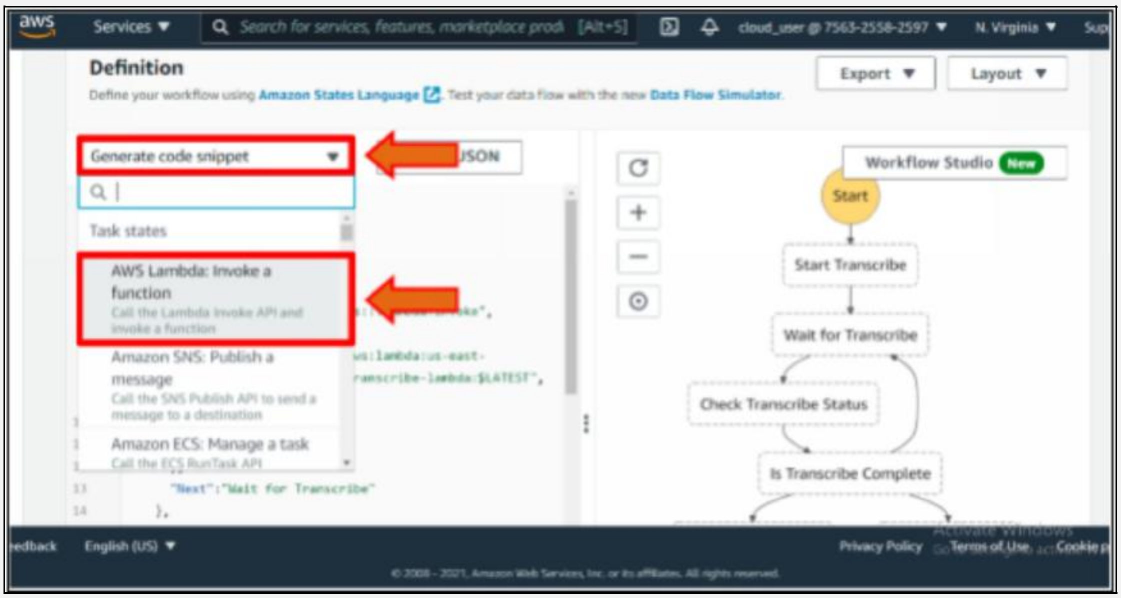


7. Click the **Edit** button.

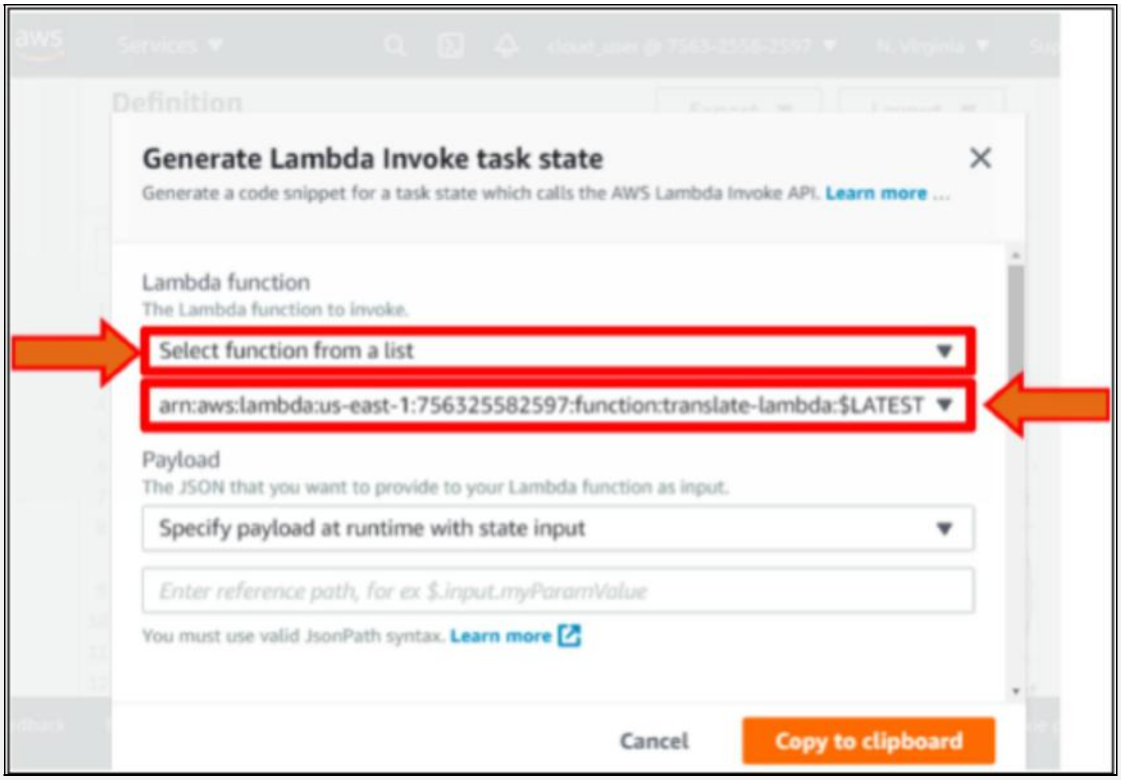




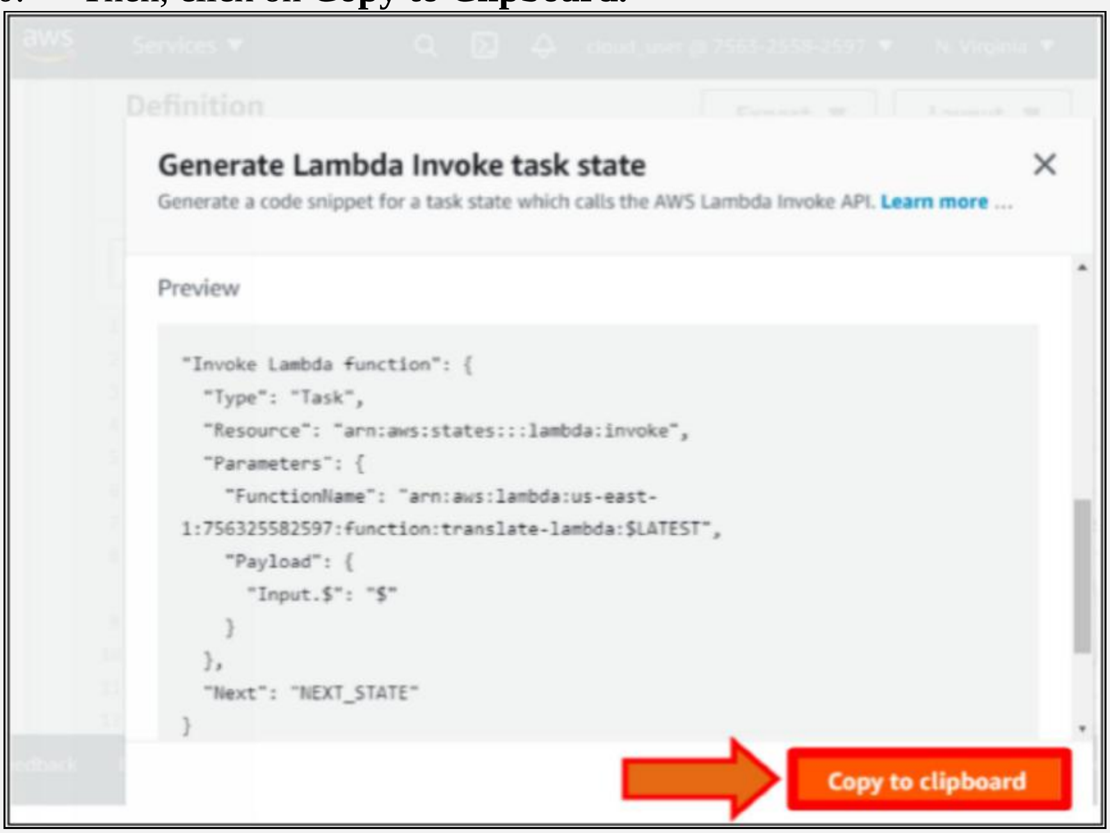
8. From **Generate a code snipped,** select **AWS Lambda: Invoke a function**.



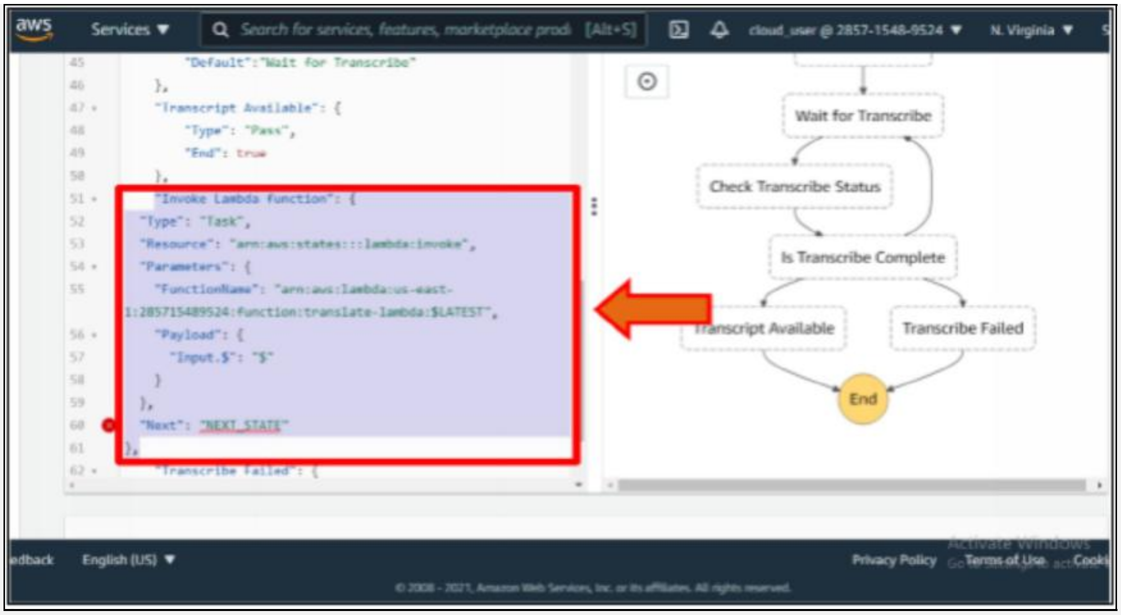
9. From the Lambda function's dropdown menu, select a function from a list and then Select the **translate-lambda** function from the sub-menu.



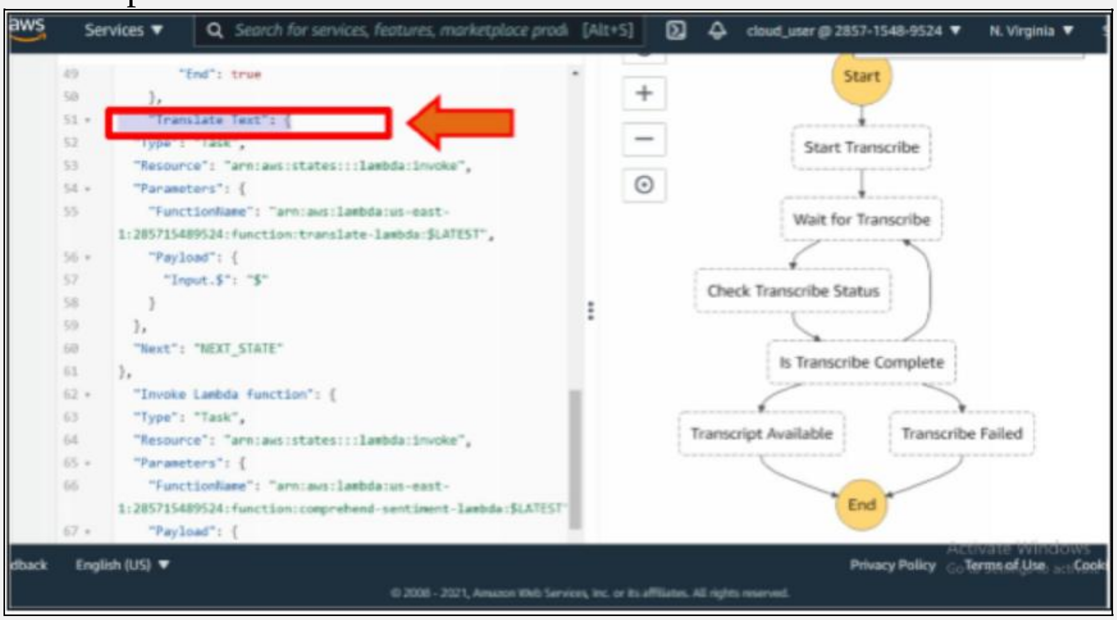
10. Then, click on **Copy to Clipboard.**



11. Paste the code snippet on line 51.

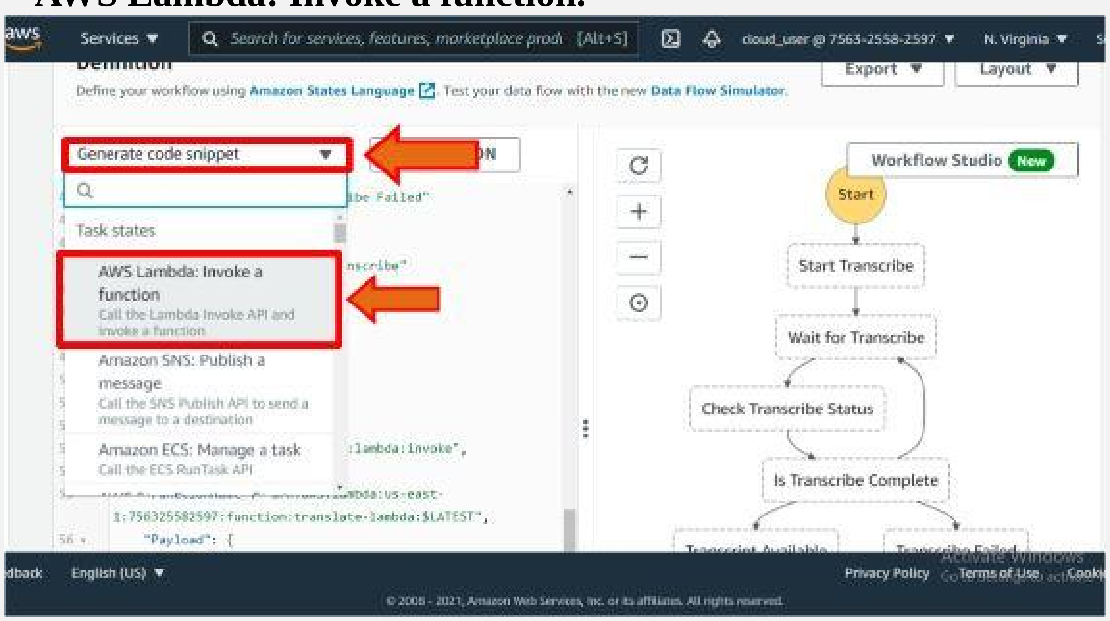


12. Replace **Invoke Lambda Function** with **Translate Text.**



13. Now add Sentiment Analysis to the pipeline.

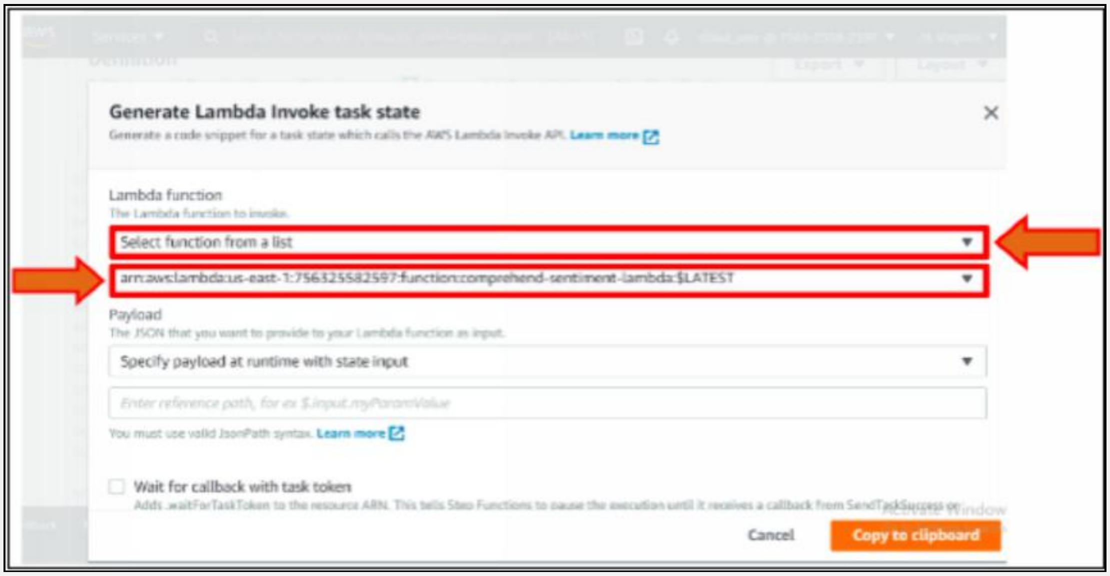
14. In the **Step Function** tab, from **Generate a code snippet,** select **AWS Lambda: Invoke a function.**



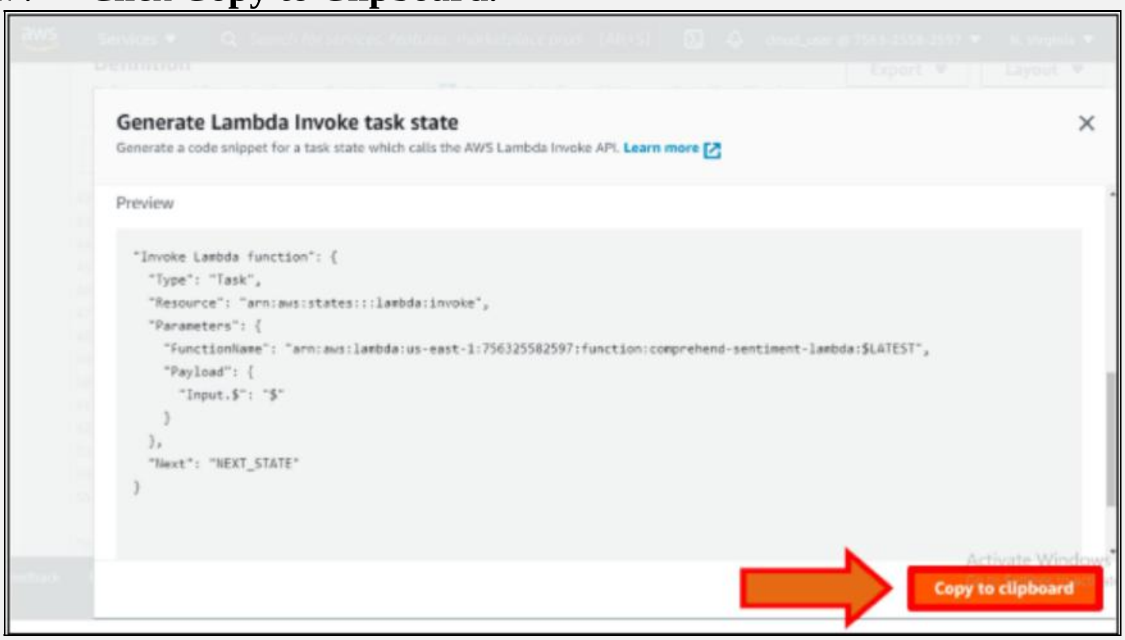
15. From the Lambda Function dropdown menu, select **function from**

a list.

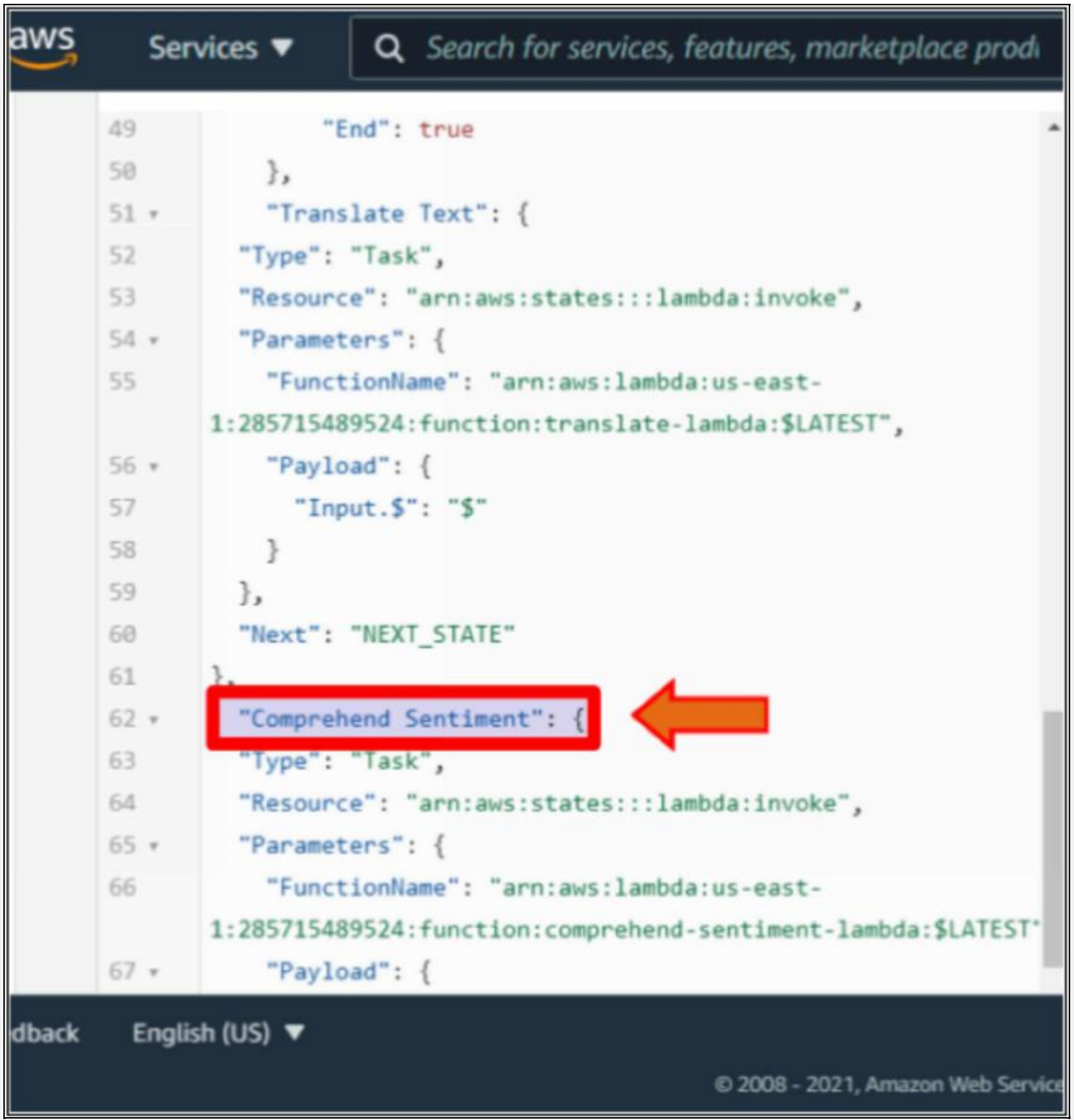
16. Then select the **Comprehend-Sentiment** function from the sub-menu.



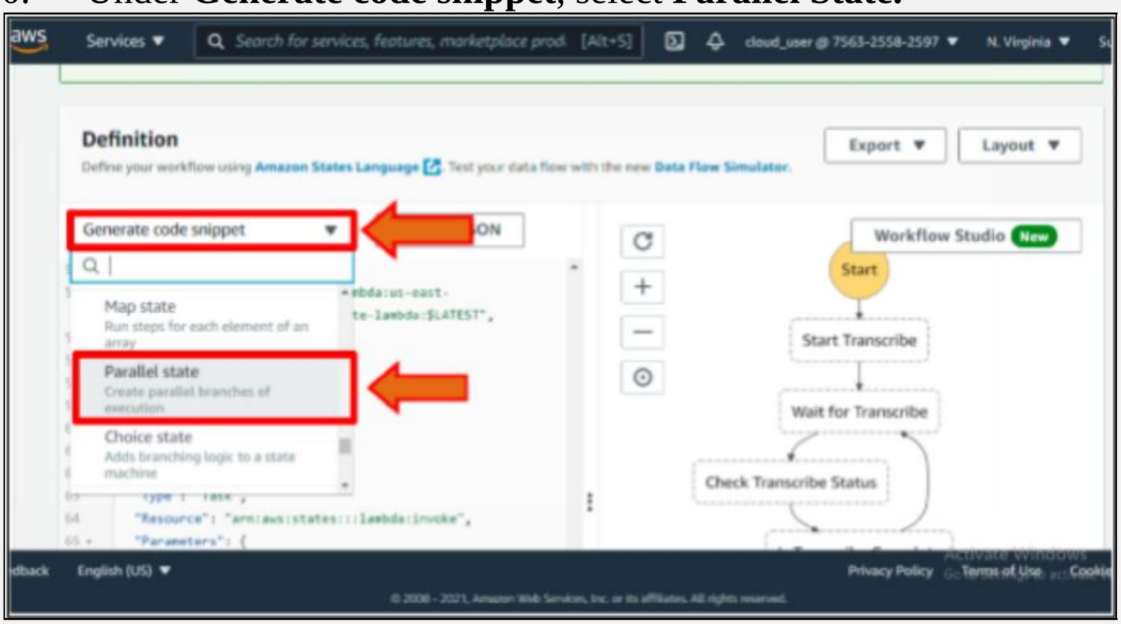
17. Click **Copy to Clipboard.**



18. Paste the **Comprehend-Sentiment** code snippet on line 62.
19. Replace **Invoke Lambda Function** with **Comprehend Sentiment**.

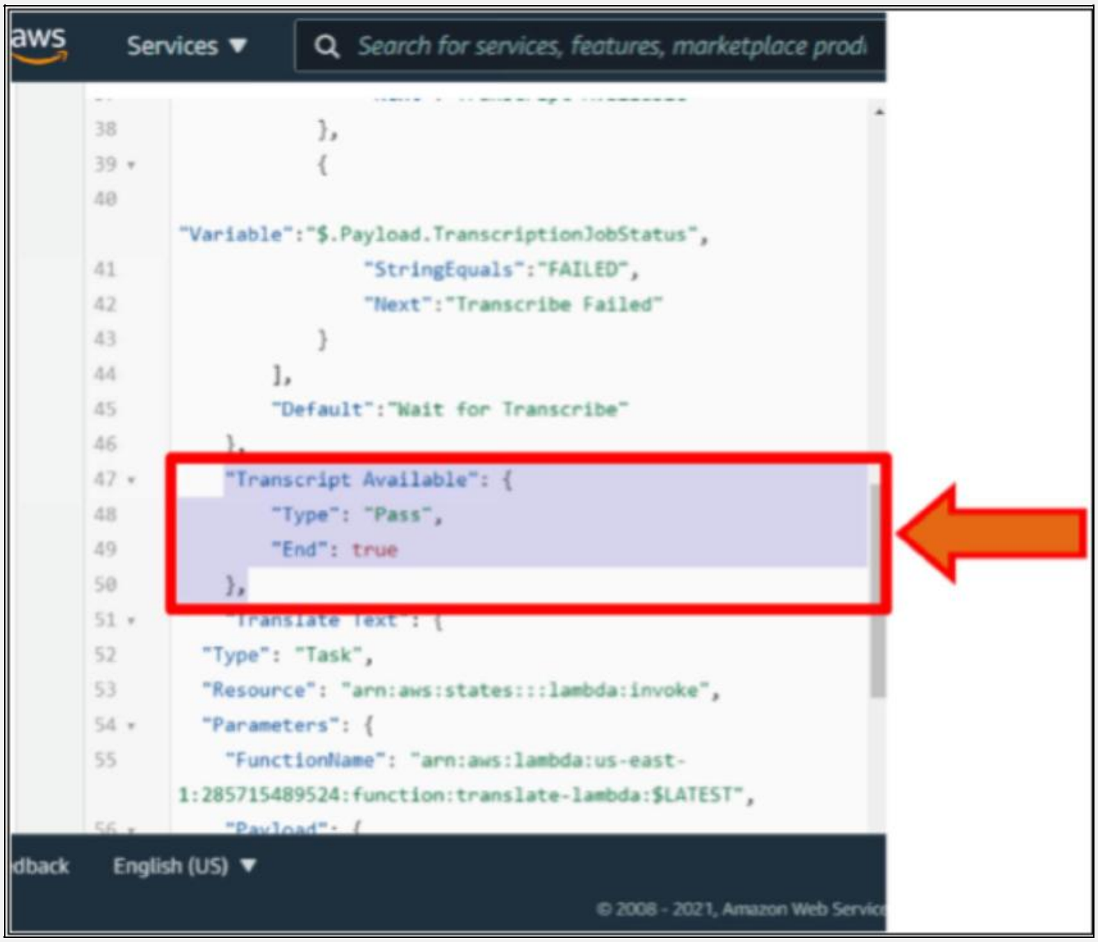


20. Under **Generate code snippet**, select **Parallel State.**

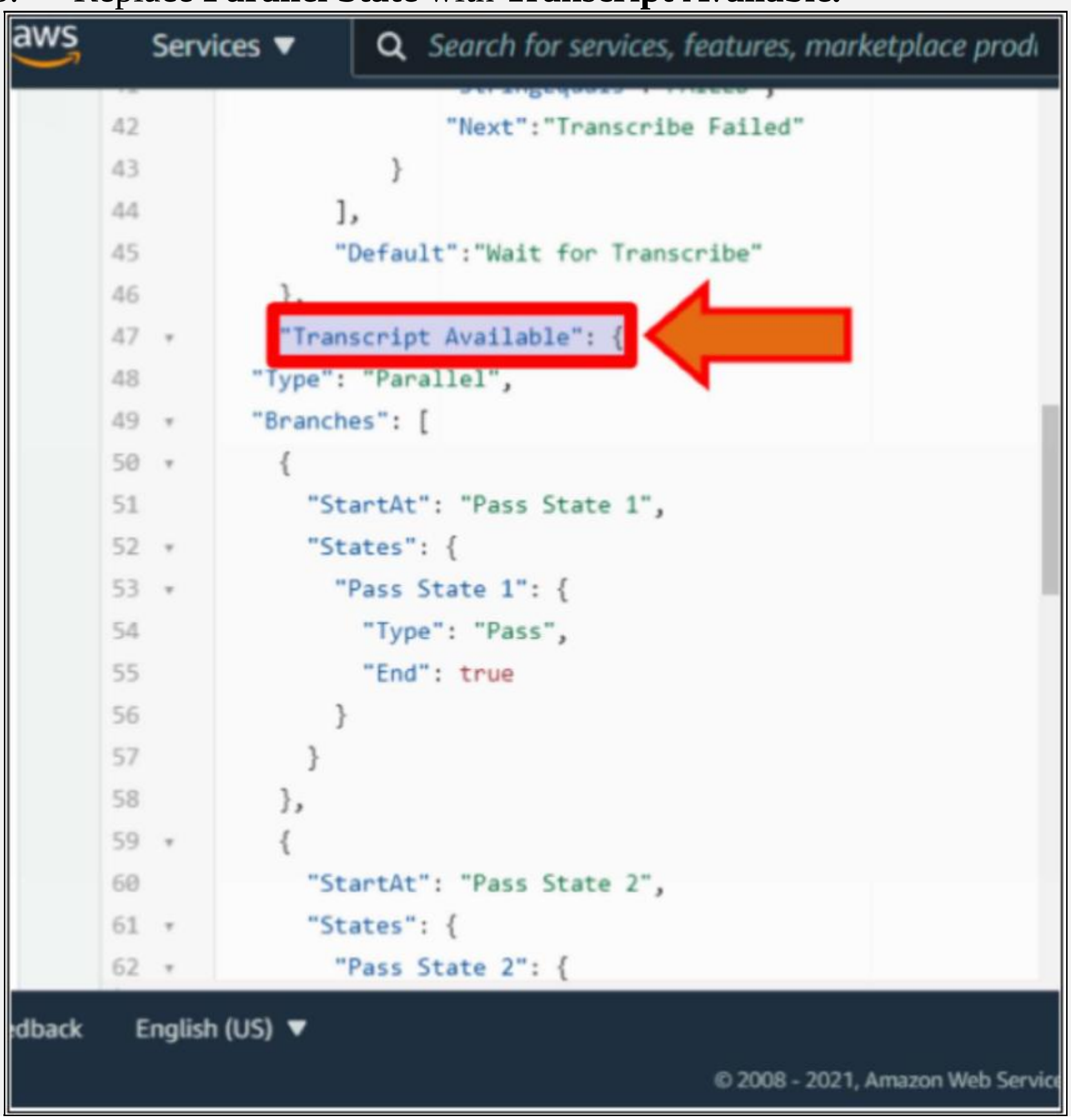


21. Click **Copy to Clipboard.**

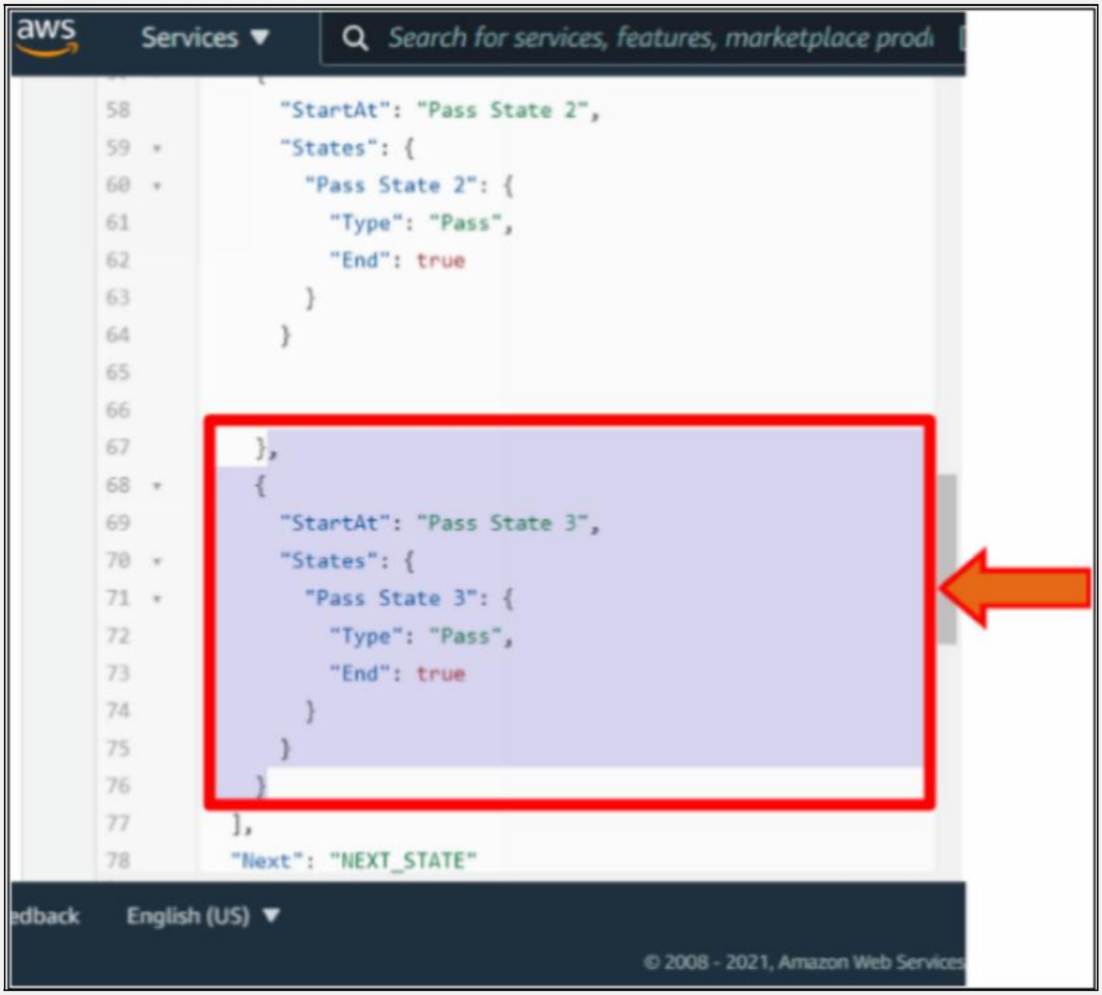
22. Remove the code from lines 47 to 50 and paste the **Parallel State** block code.



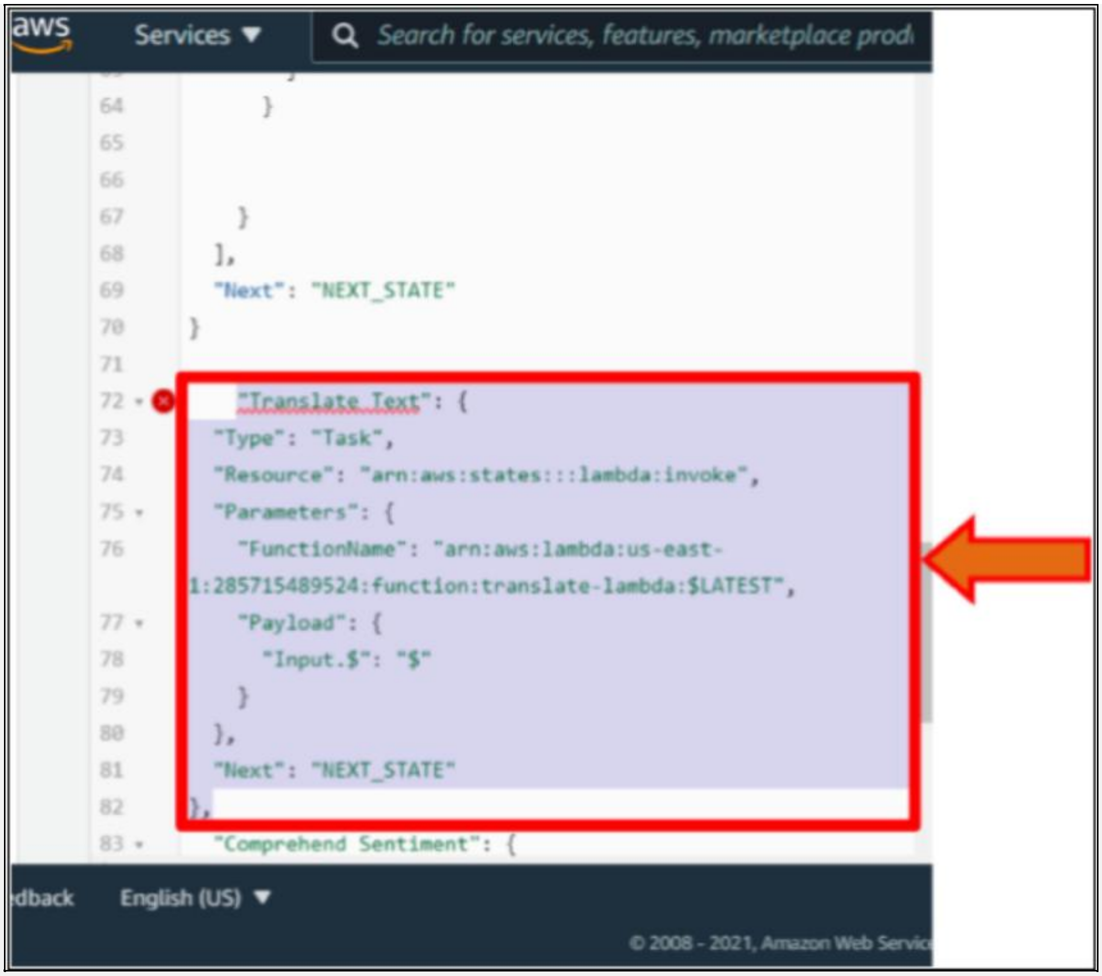
23. Replace **Parallel State** with **Transcript Available.**



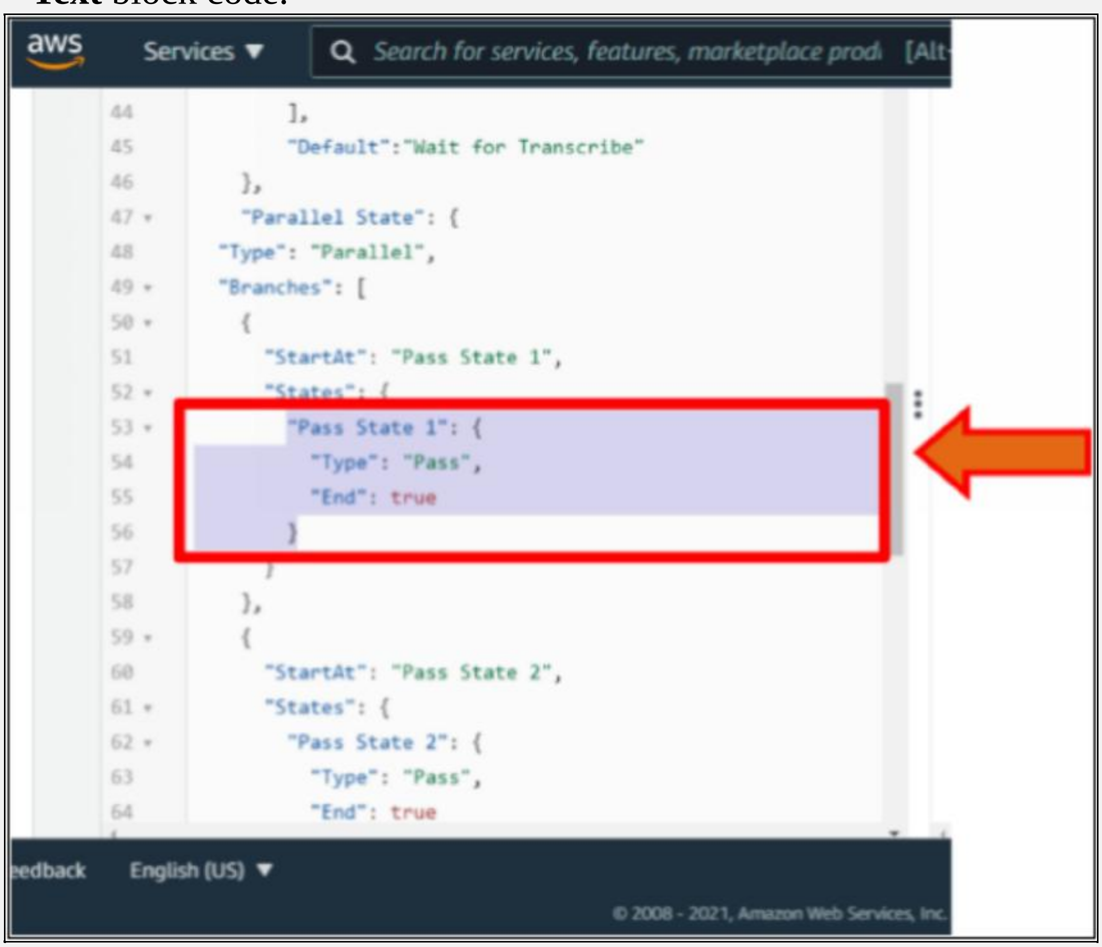
24. Now remove the **Pass State 3** block code.



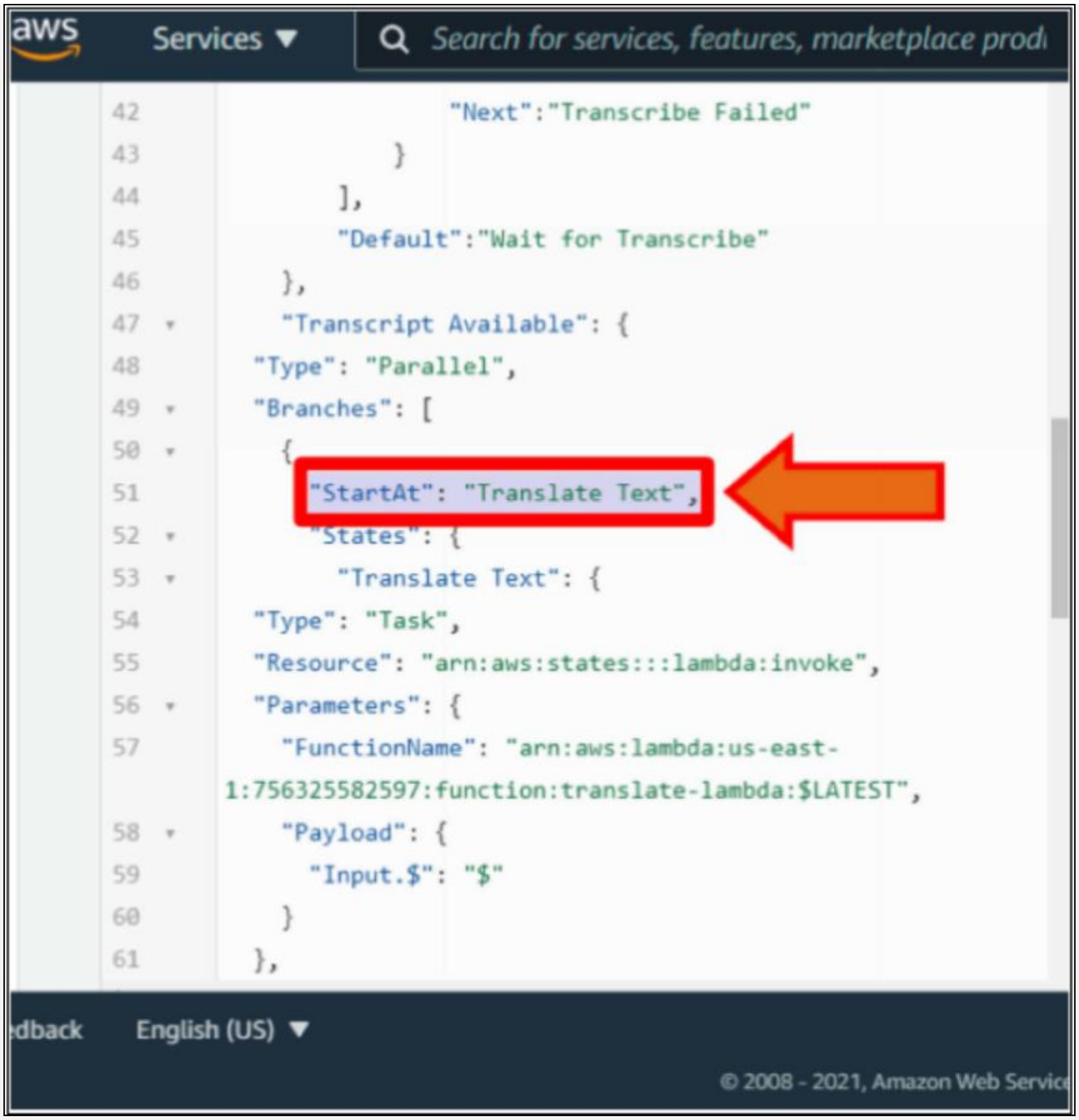
25. Copy the **Translate Text** block code. Remove it from its current position.



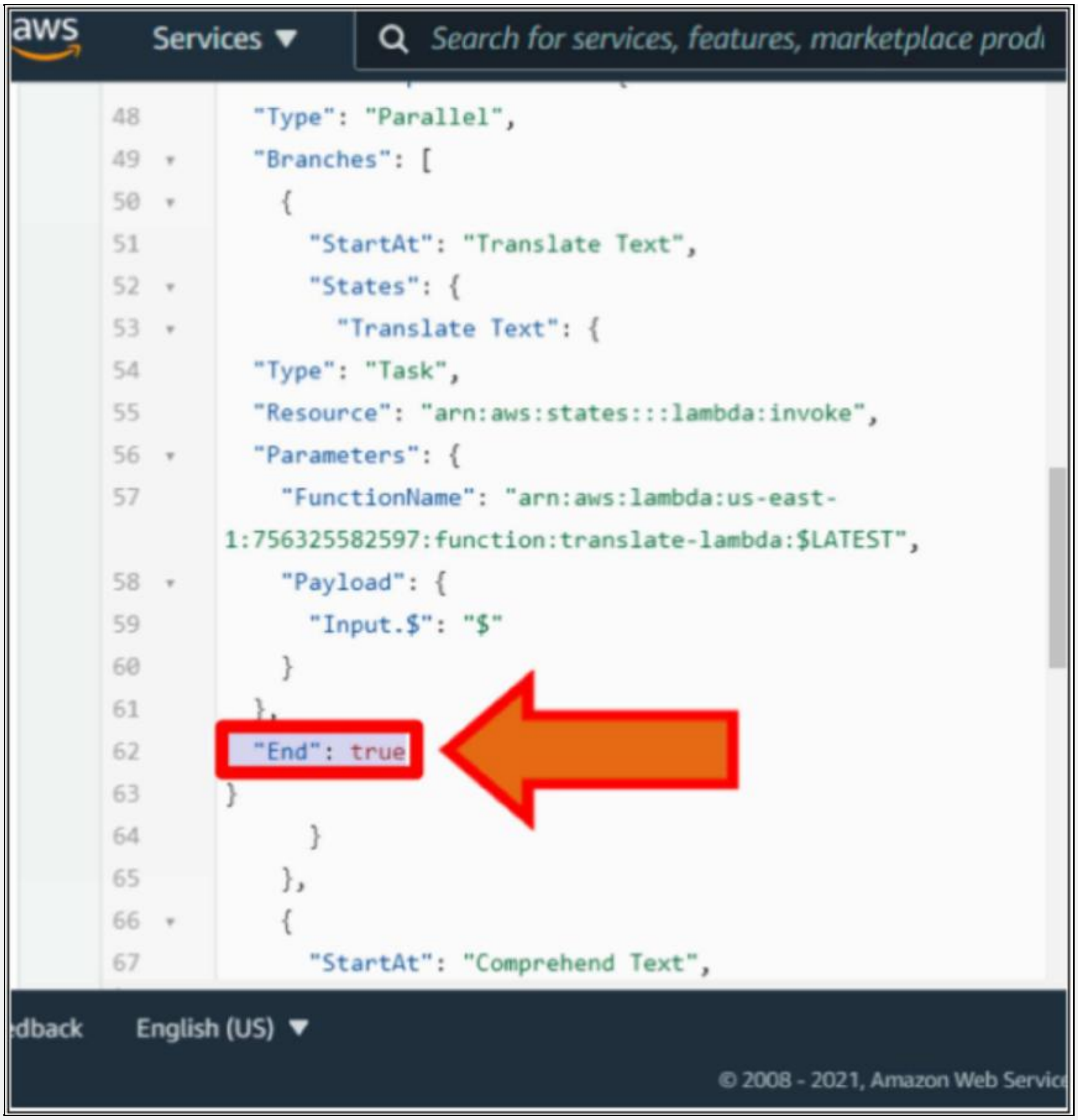
26. Remove **Pass state 1** from lines 53 to 56 and paste the **Translate Text** block code.



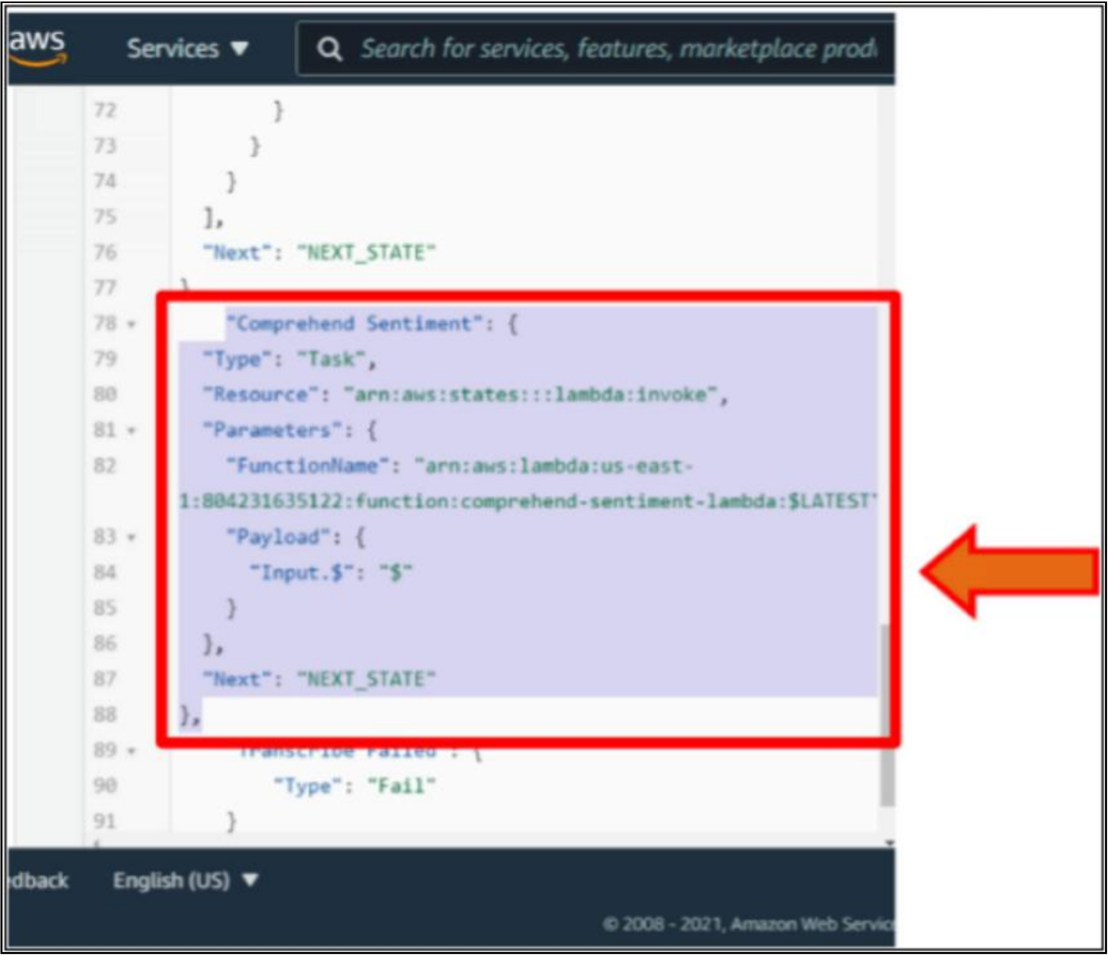
27. For this block, replace **StartAt** with **Translate Text.**



28. In line 62, replace **Next: NEXT_STATE** with **End**: true.

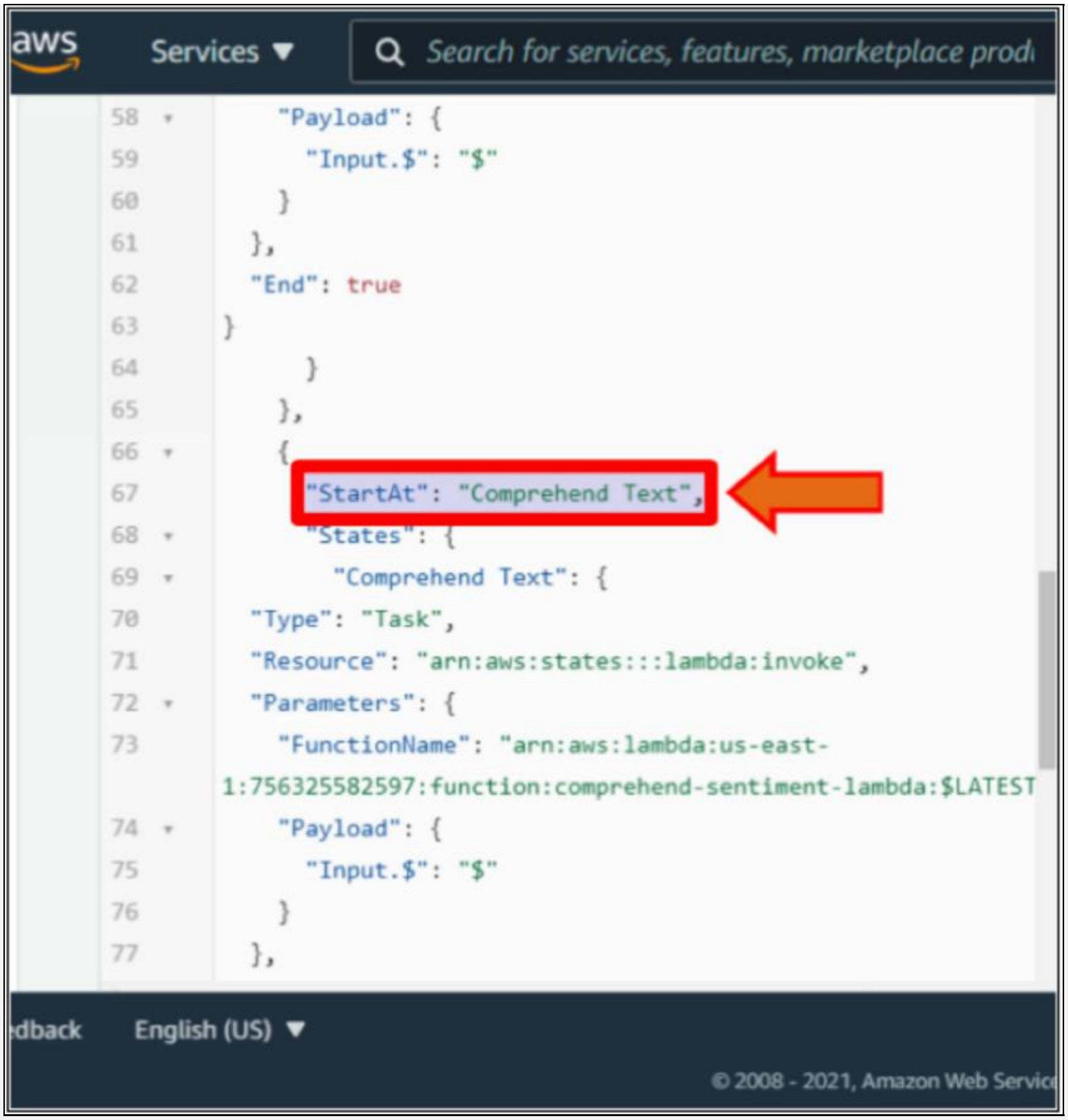


29. Copy the **Comprehend Sentiment** block code. Remove it from its current position.

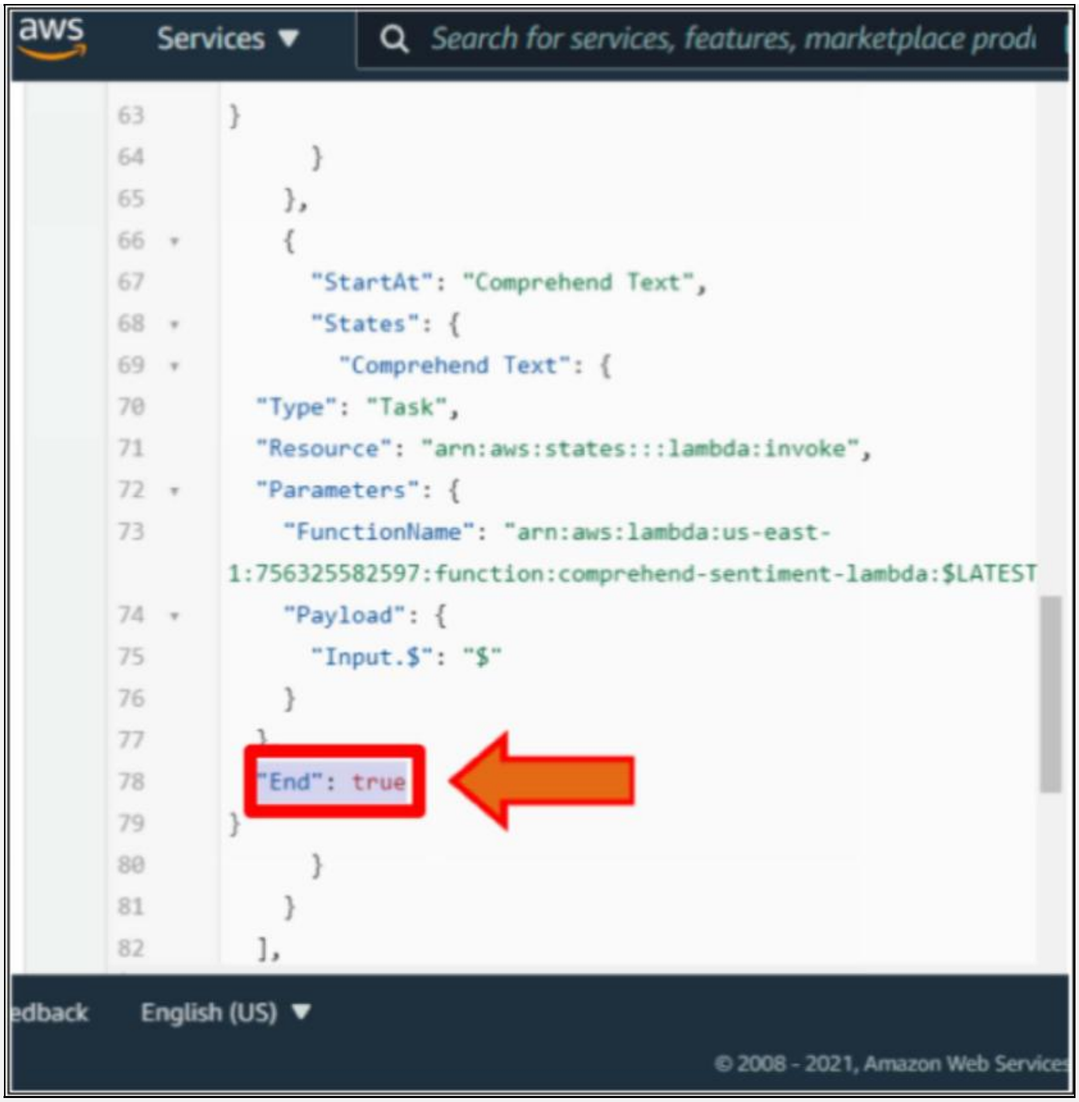


30. Remove **Pass State 2** from lines 69 to 72. Paste the **Comprehend Sentiment** block code.

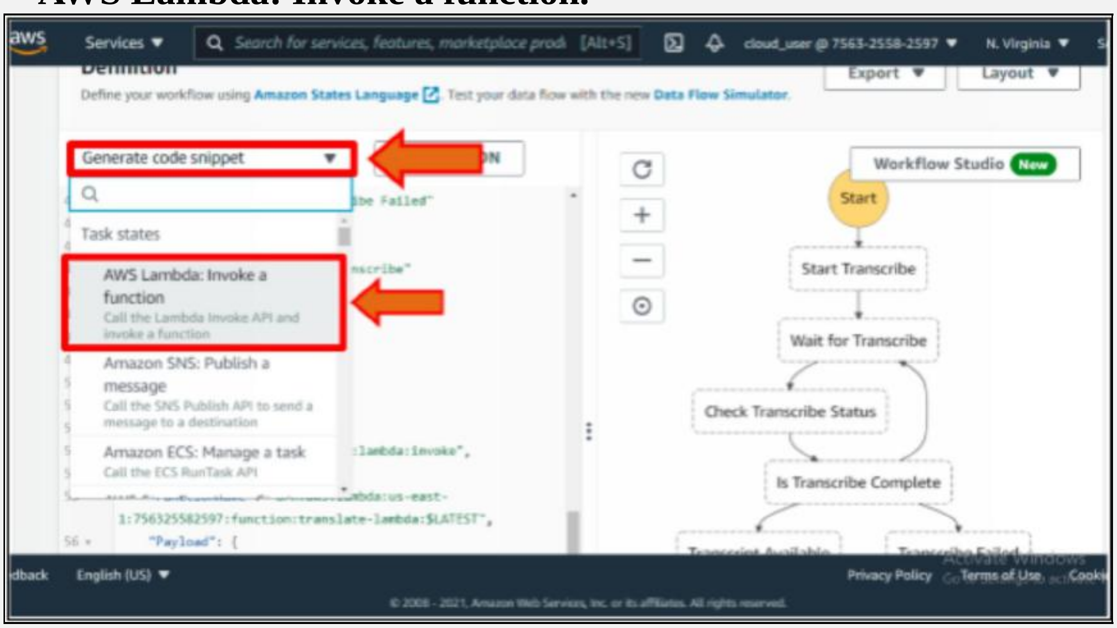
31. Replace **StartAt** with **Comprehend Sentiment**.



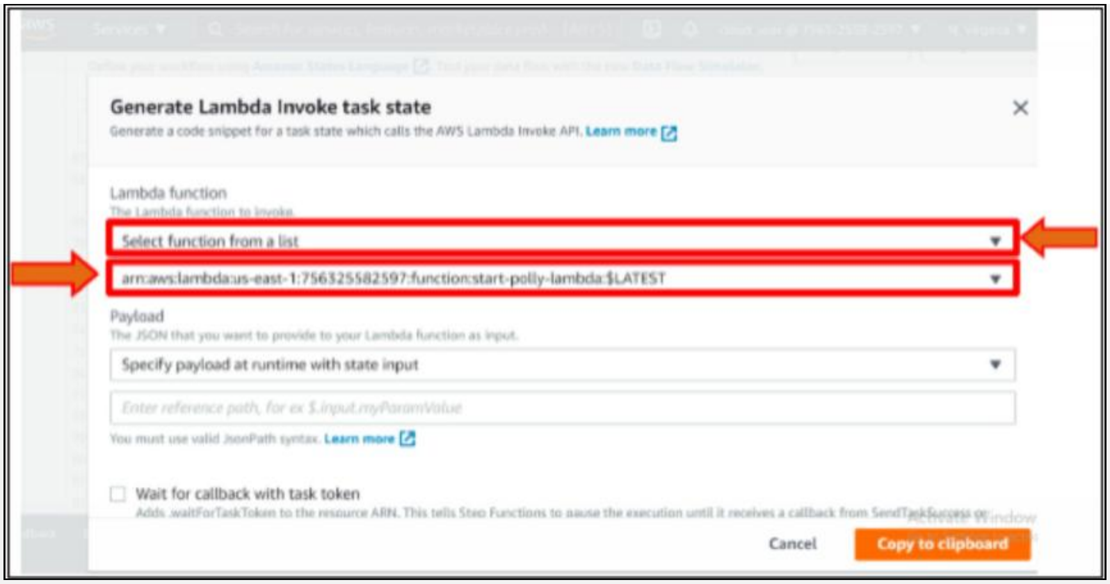
32. In line 78, replace **Next: NEXT_STATE** with **End**: true.



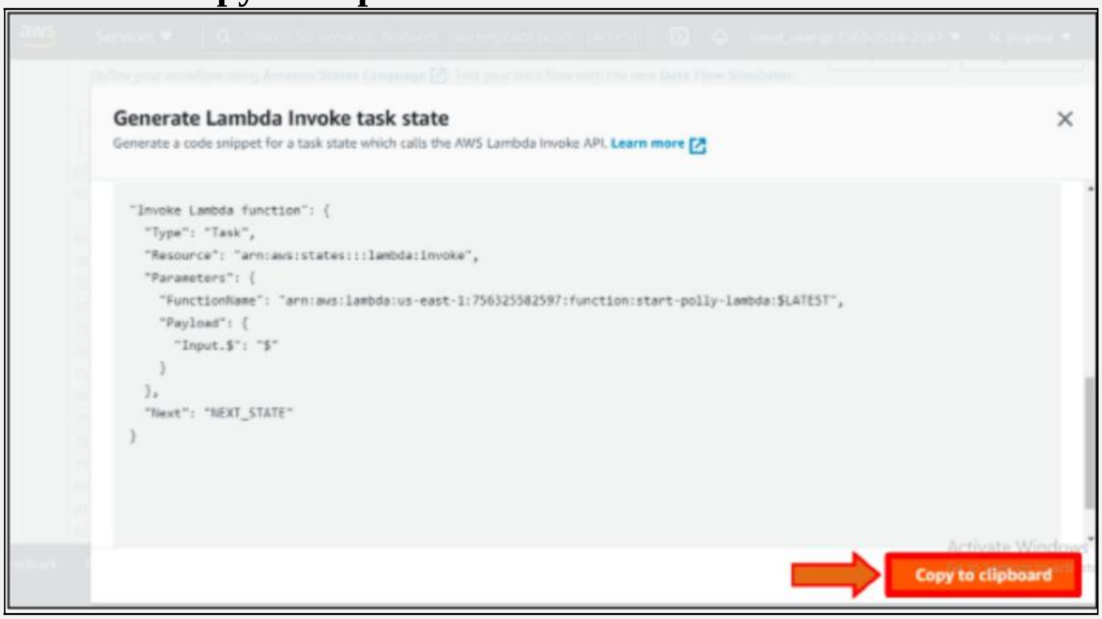
33. Now convert the Translated Text to Audio.
34. In the **Step Function** tab, from **Generate a code snippet**, select **AWS Lambda: Invoke a function.**



35. From the Lambda function dropdown menu, select **Start-Polly** Lambda Function.

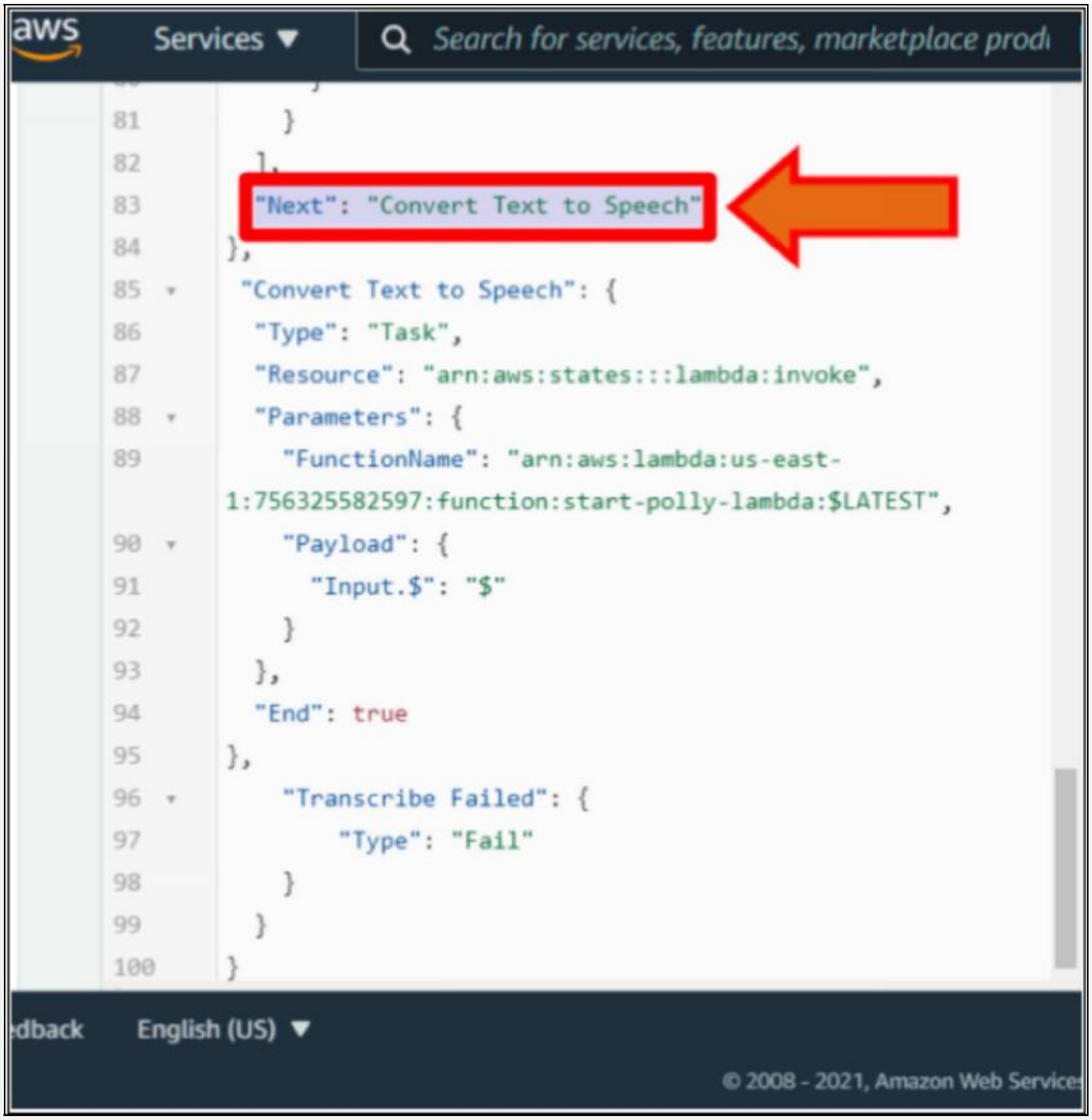


36. Click **Copy to Clipboard.**

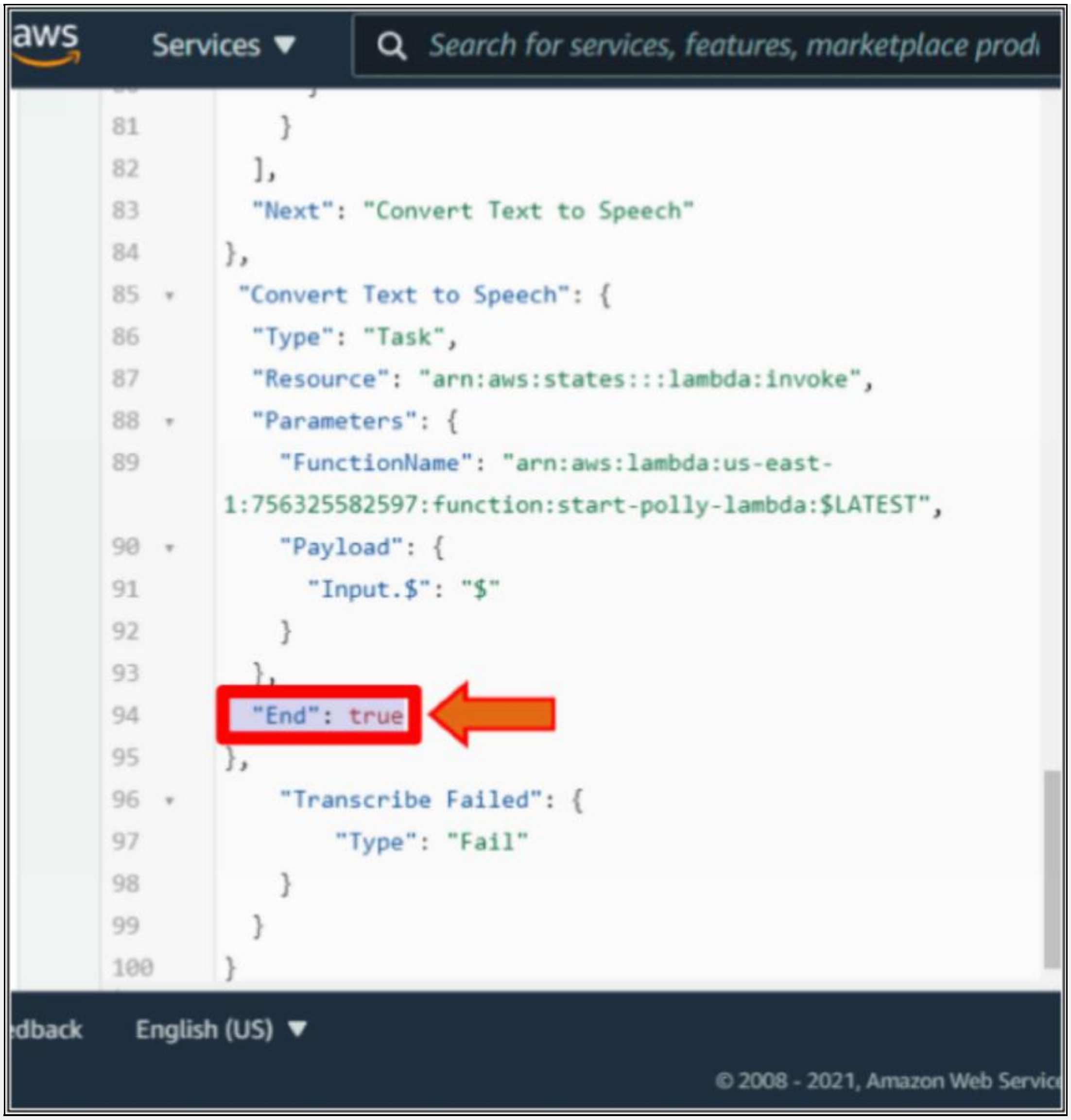


37. Paste the **Start-Polly** code block on line 85.
38. Replace **Invoke Lambda Function** with **Convert Text to Speech**.

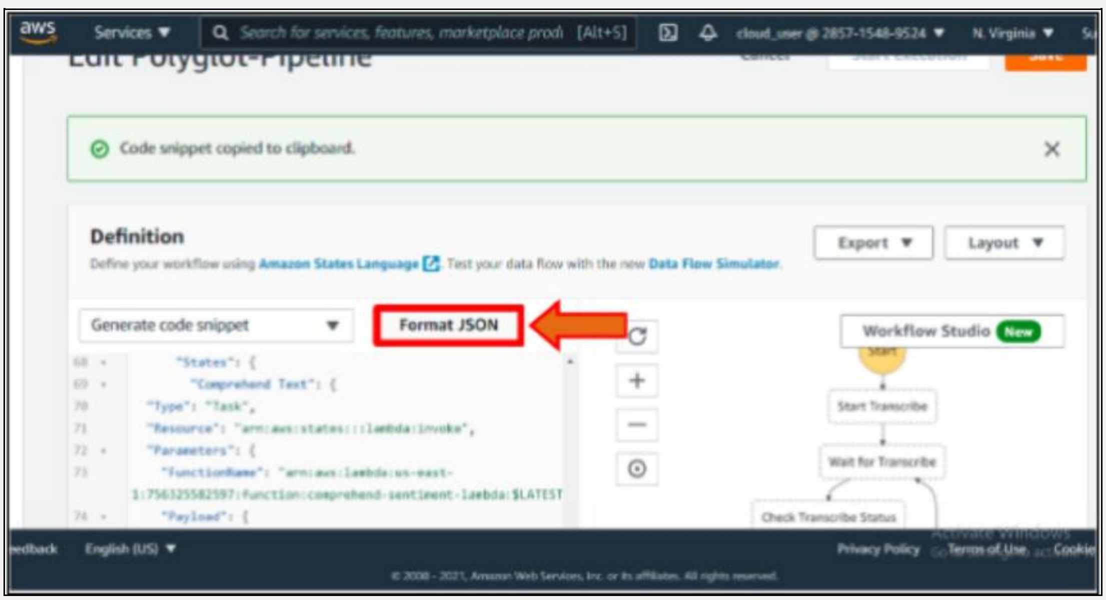
39. Replace the **Next: NEXT_STATE** with **Next: Convert Text to Speech**.



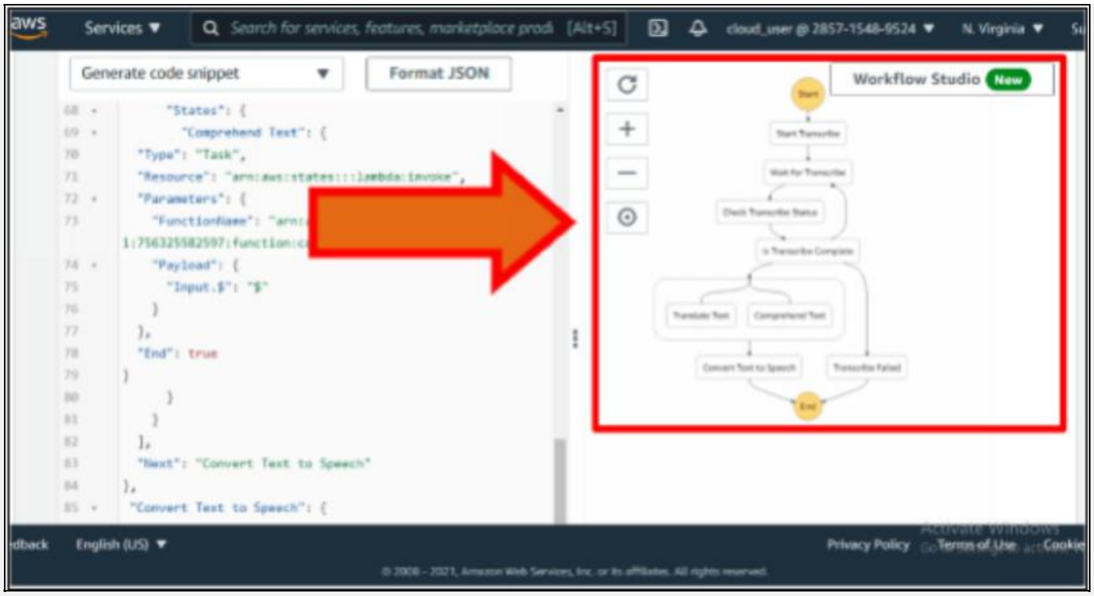
40. In the end, the block replaces **Next: NEXT_STATE** with **End:** true.



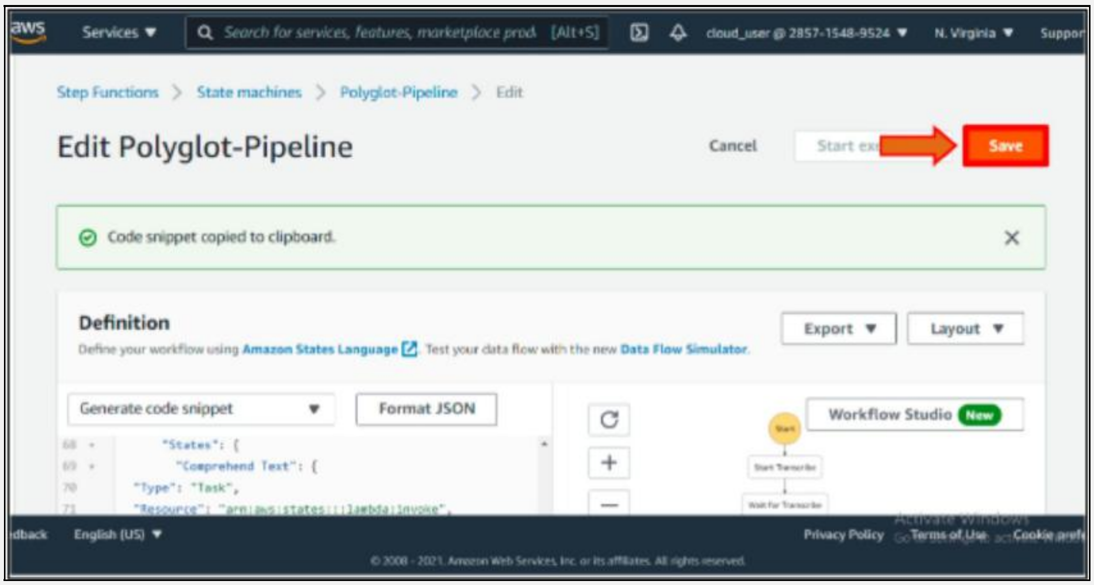
41. Click **Format JSON.**



42. On the right-hand side of the screen, you will observe the change in the block diagram.



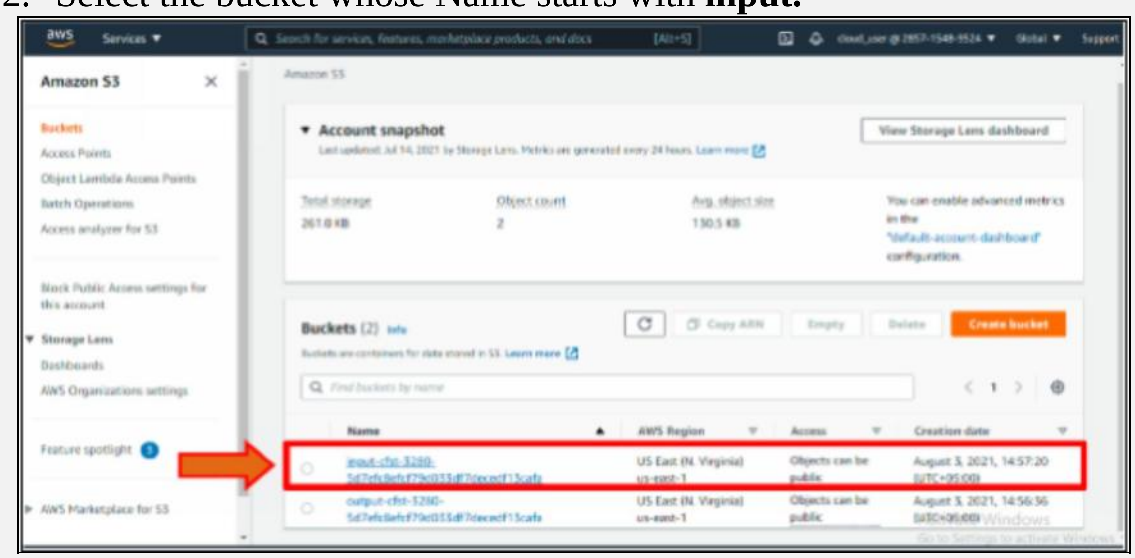
43. Click the **Save** button.



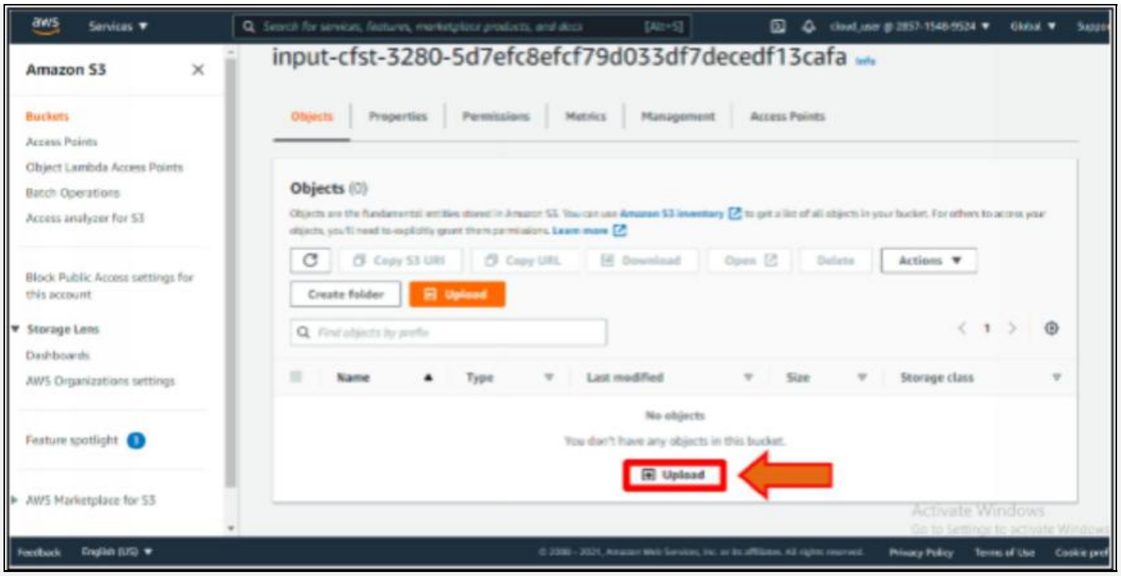
44. After that, upload the Audio file and watch the magic.

Step 2: Upload an Audio File on S3 Bucket

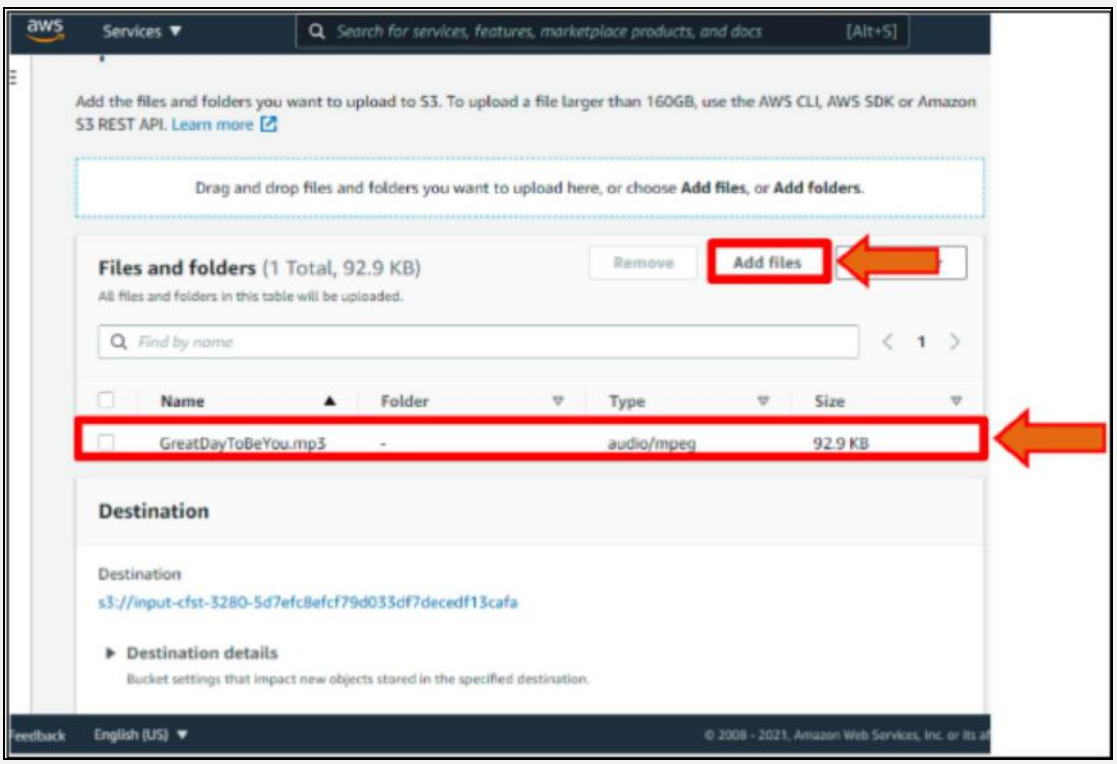
1. Go to the **S3** tab.
2. Select the bucket whose Name starts with **input.**



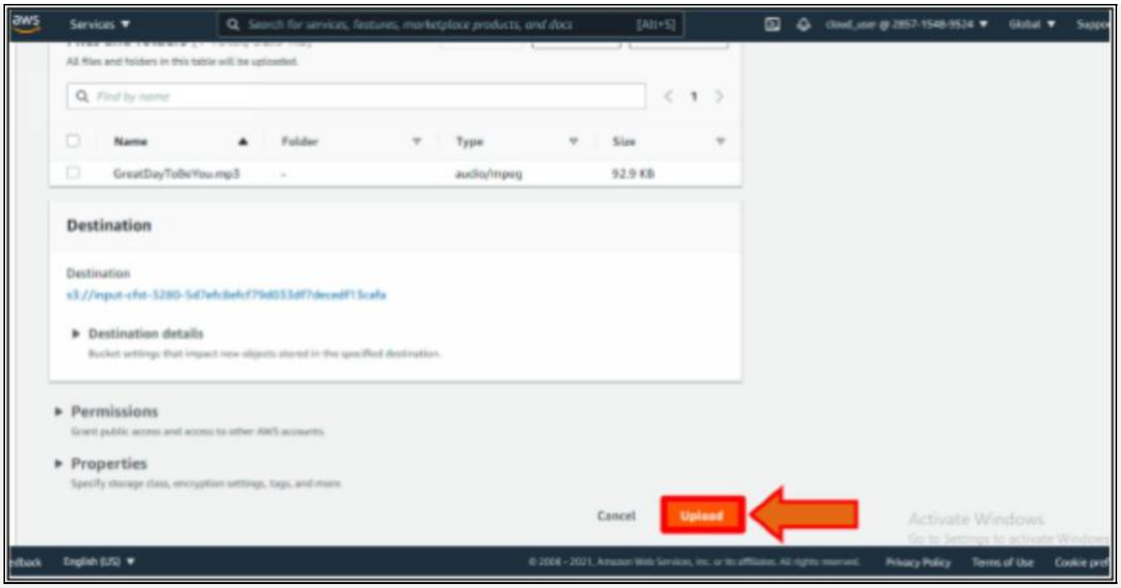
3. Click the **Upload** button.



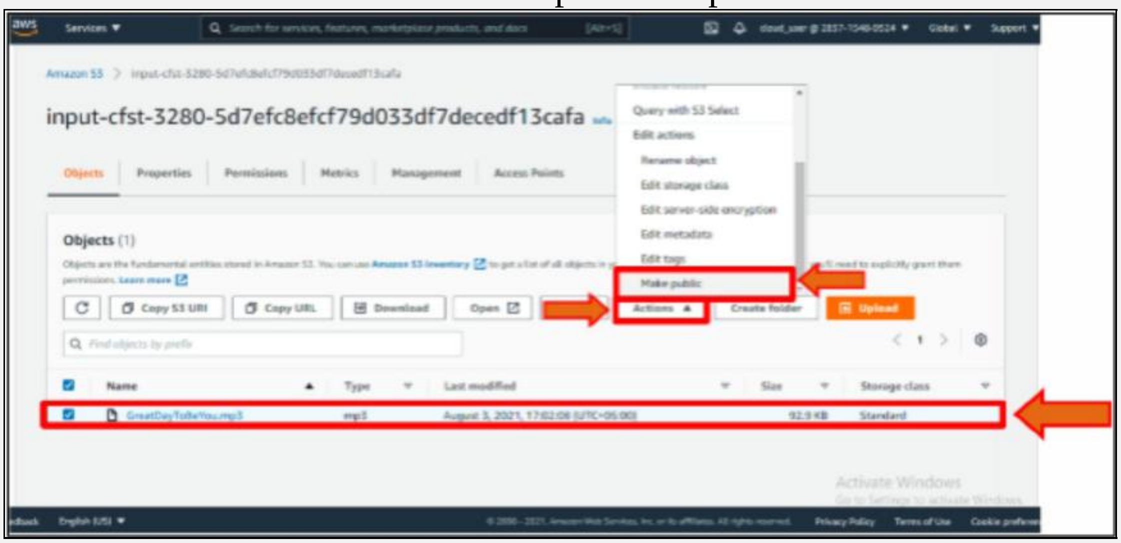
4. Click the **Add files** button.
5. Select the **GreatDayToBeYou.mp3** file and upload it. The files are provided in the below Github link: https://github.com/linuxacademy/content-aws-mls-c01/tree/master/CoordinateAIServicesWithStepFunctions



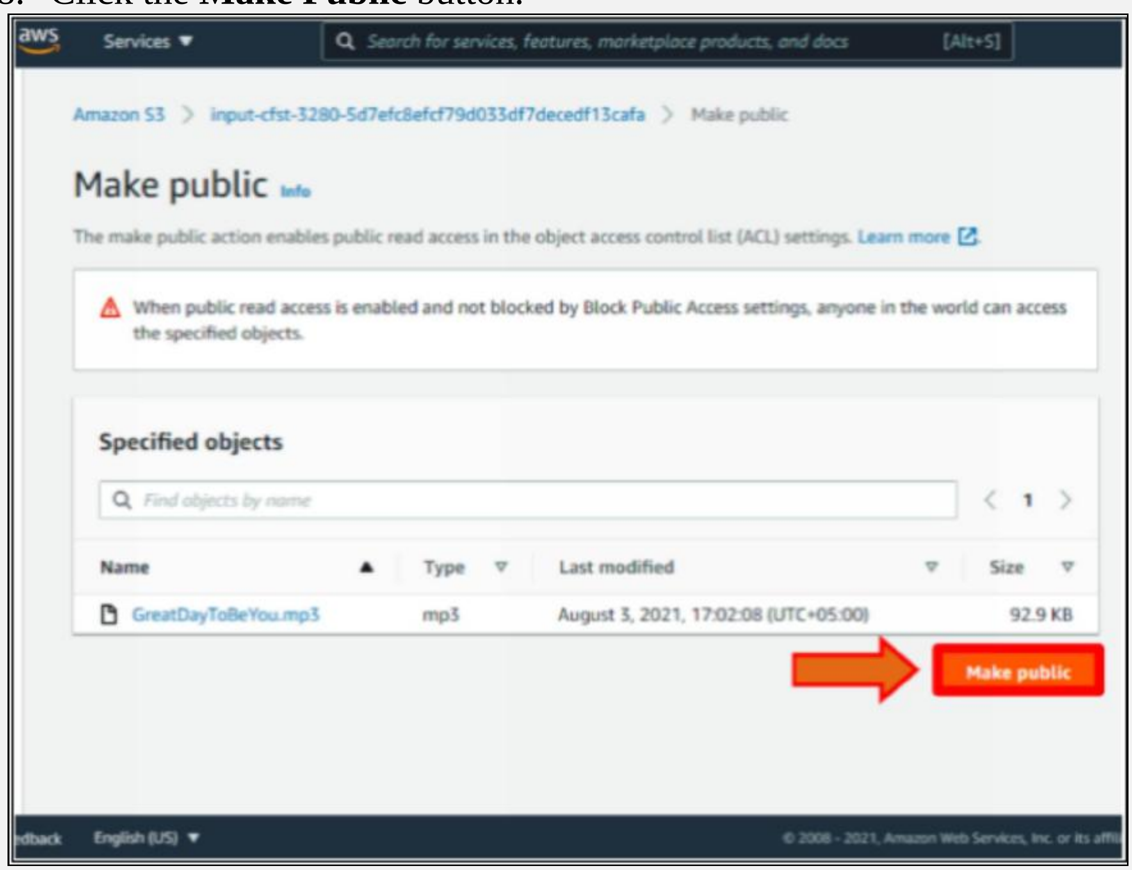
6. Click on the **Upload** button.



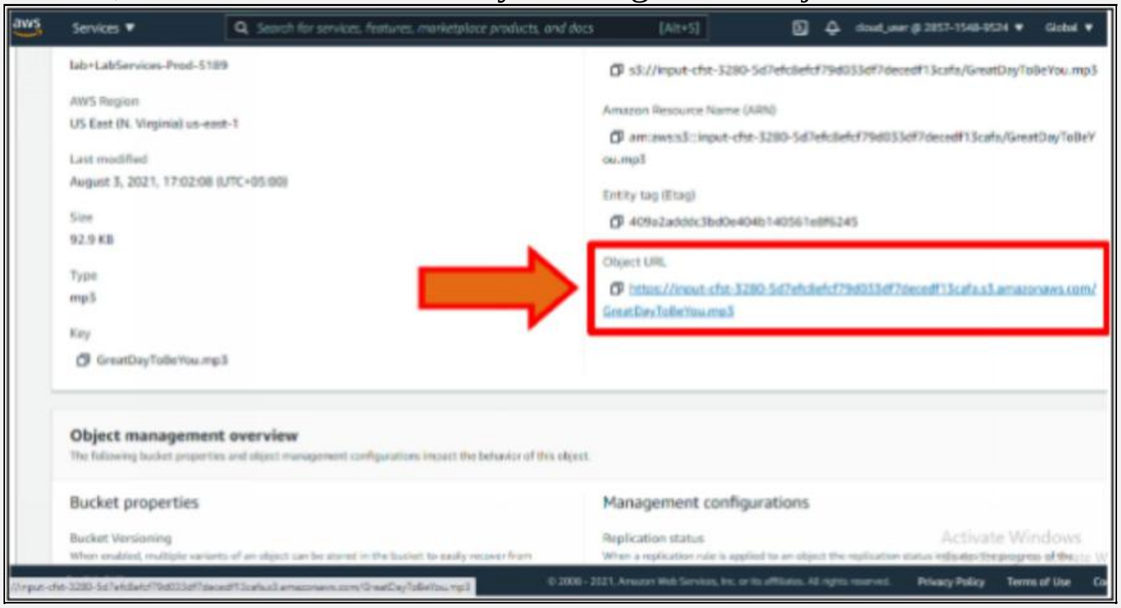
7. Select **Make Public** from the dropdown to publicize the file.



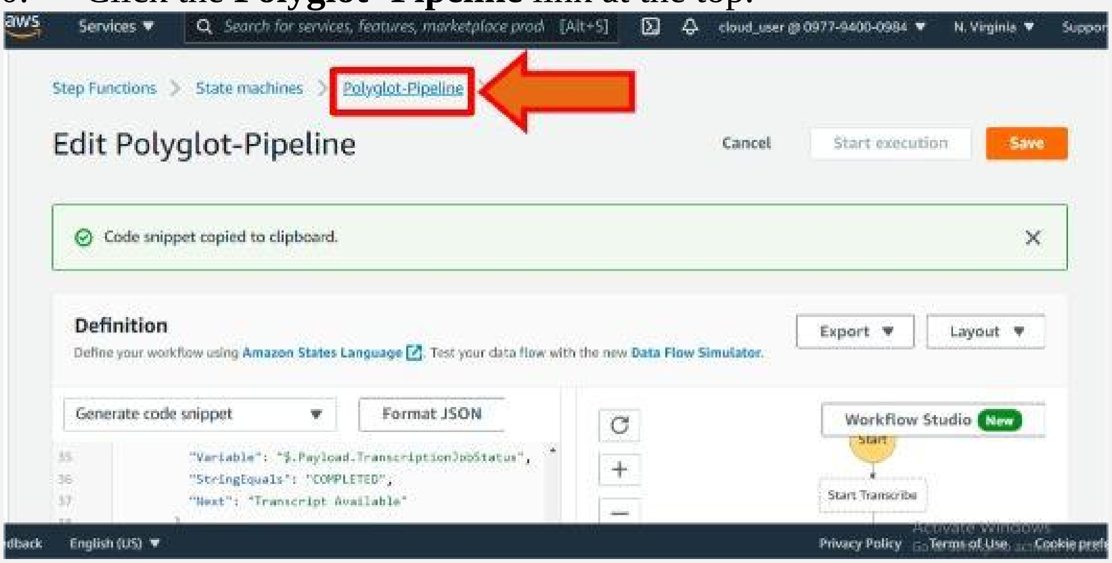
8. Click the **Make Public** button.



9. Also, listen to the file audio by clicking on the object URL.

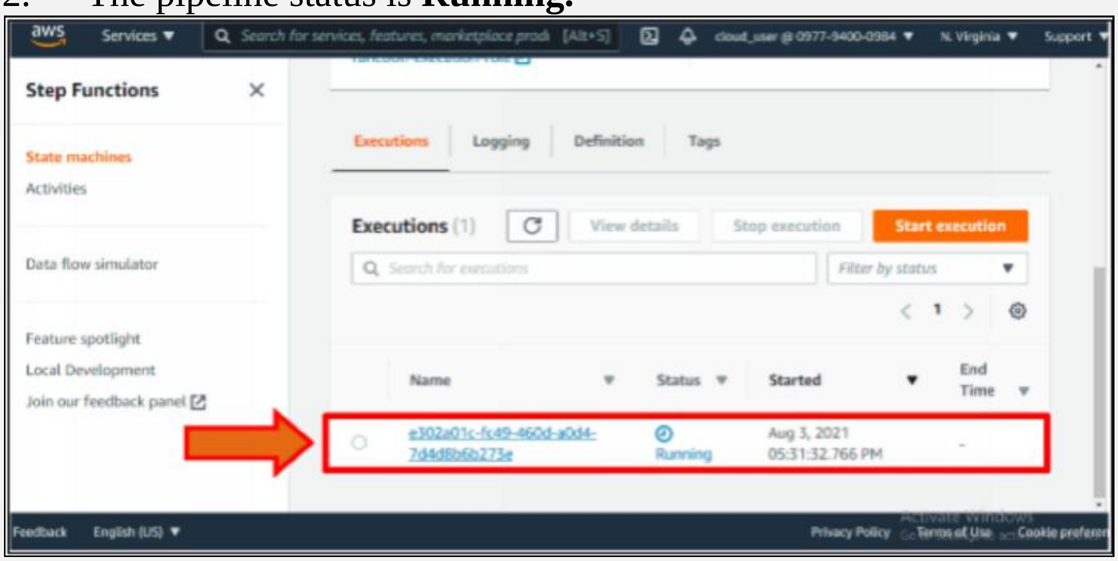


10. Click the **Polyglot- Pipeline** link at the top.



11. Click the link under **Name**.

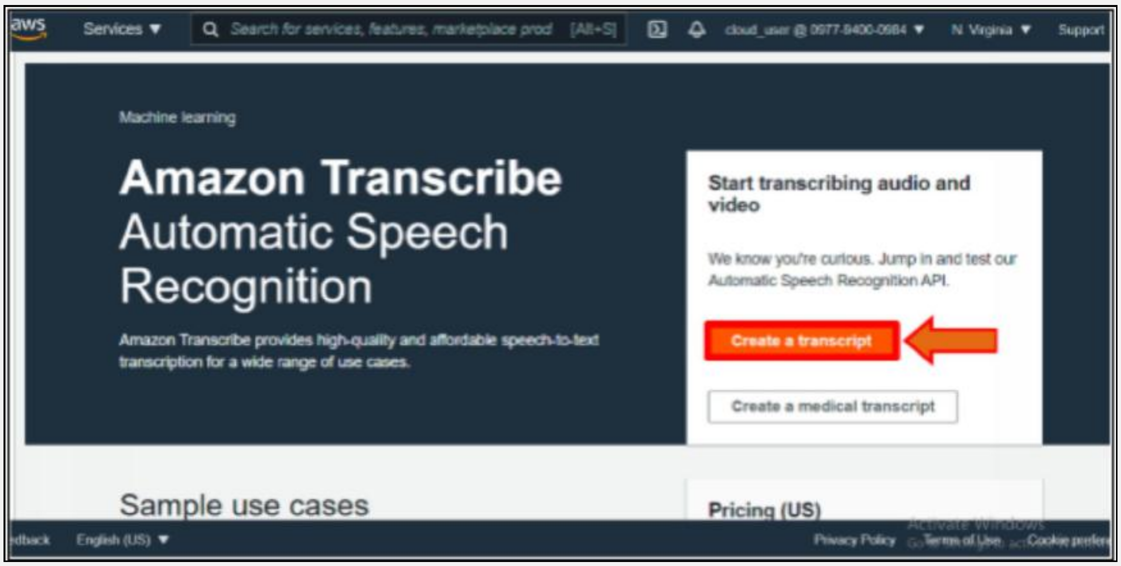
12. The pipeline status is **Running.**



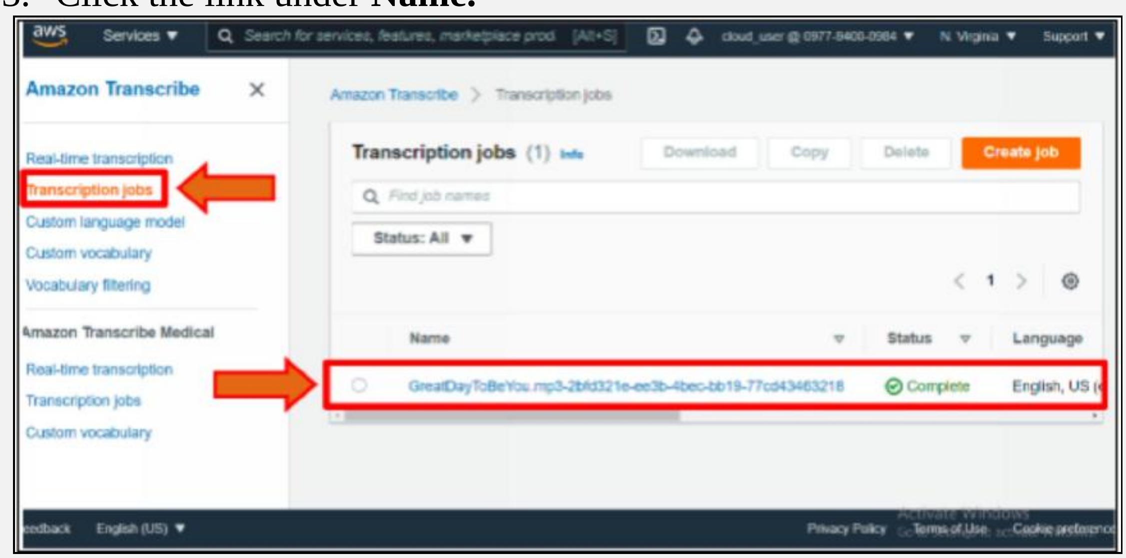
13. View the **Step Input** and **Step Output** tabs to view the pipeline's progress.

Step 3: Translate Text on AWS Transcribe

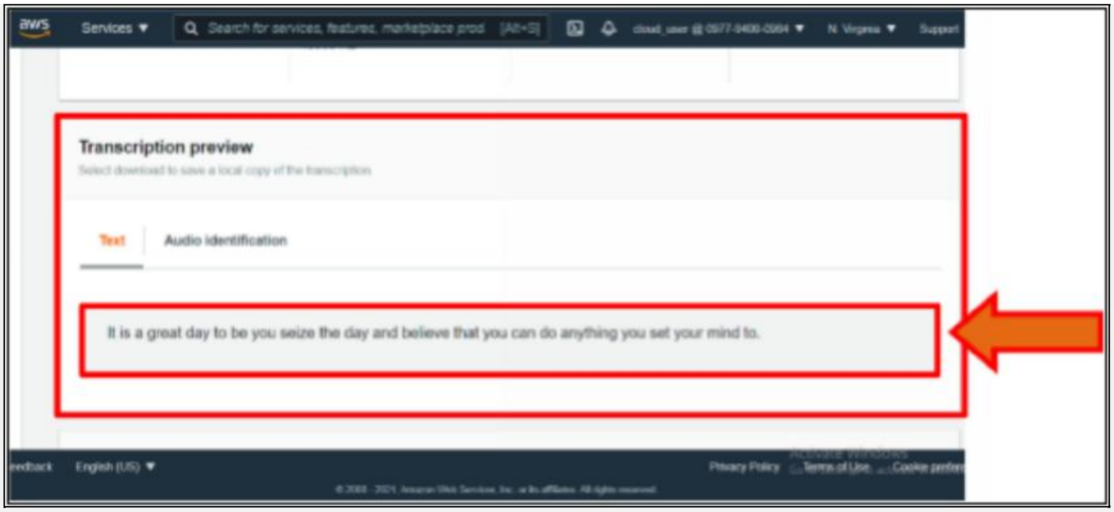
1. Now, go to **Amazon Transcribe.** Click **Launch Amazon Transcribe.**



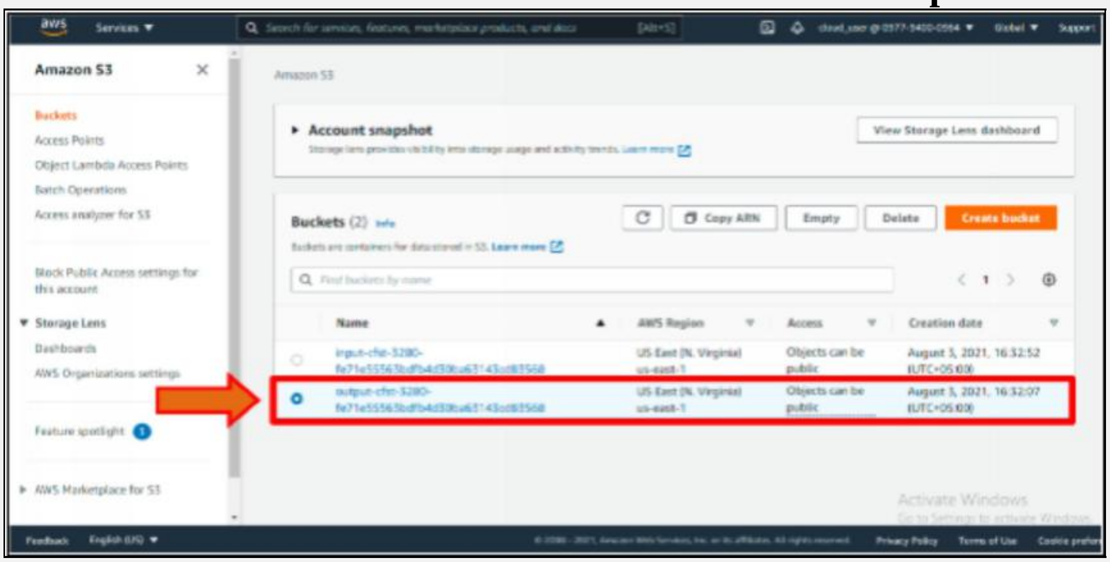
2. In the left-hand menu, click **Transcription Jobs**.
3. Click the link under **Name.**



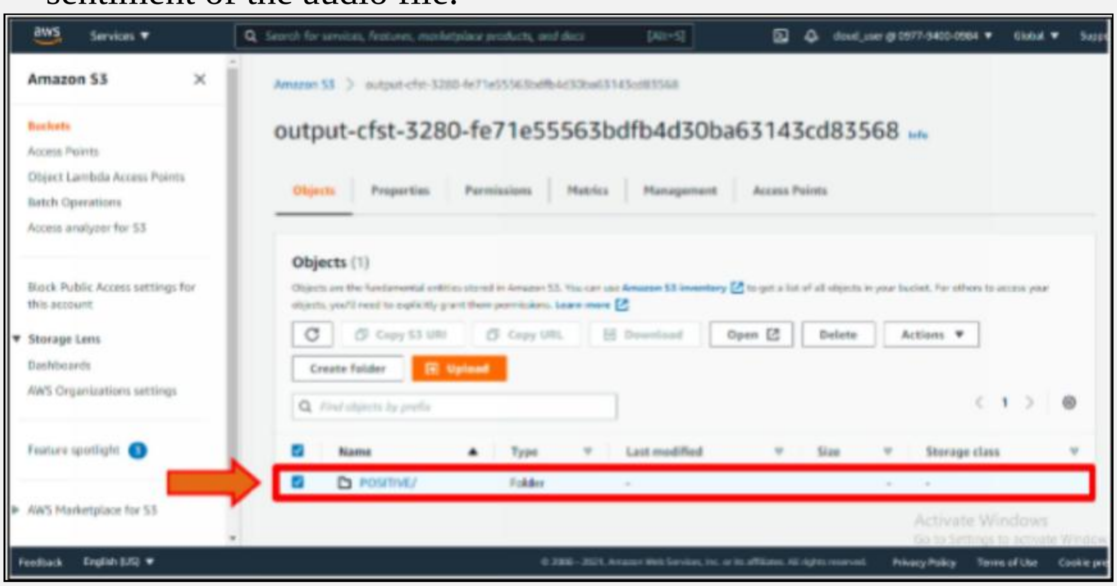
4. Under Transcription preview, view the translated text.



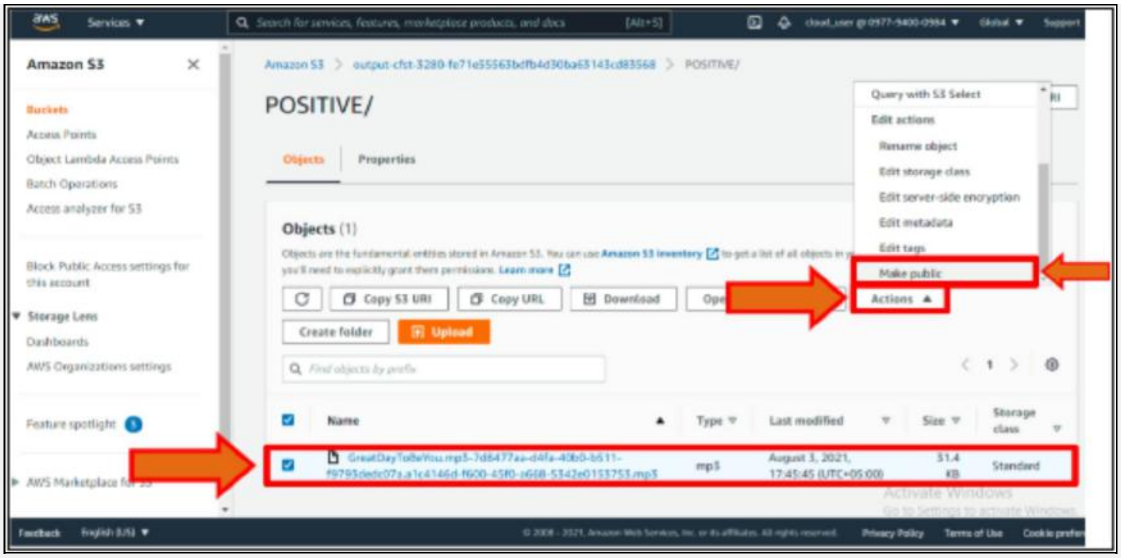
5. Go back to the **AWS S3** tab.
6. Now select the bucket name that starts with the word **Output.**



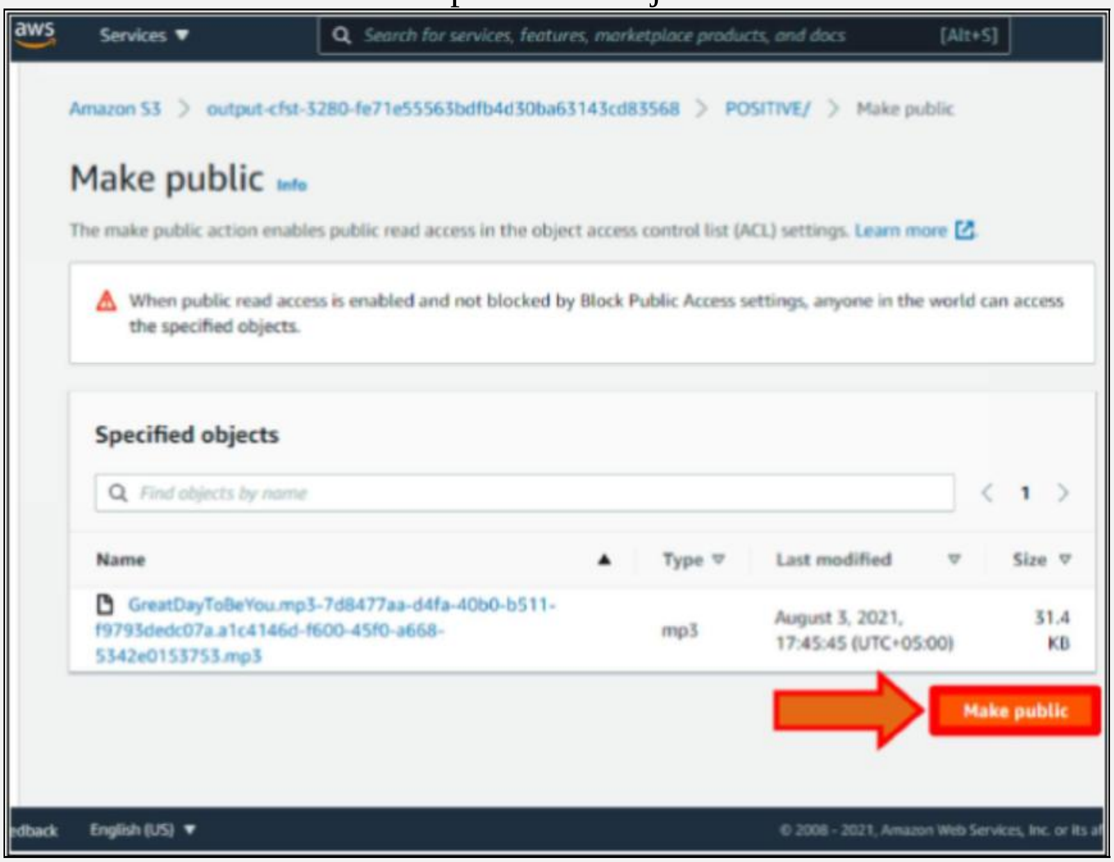
7. Click on the folder in the **Output** bucket, which should reflect the sentiment of the audio file.



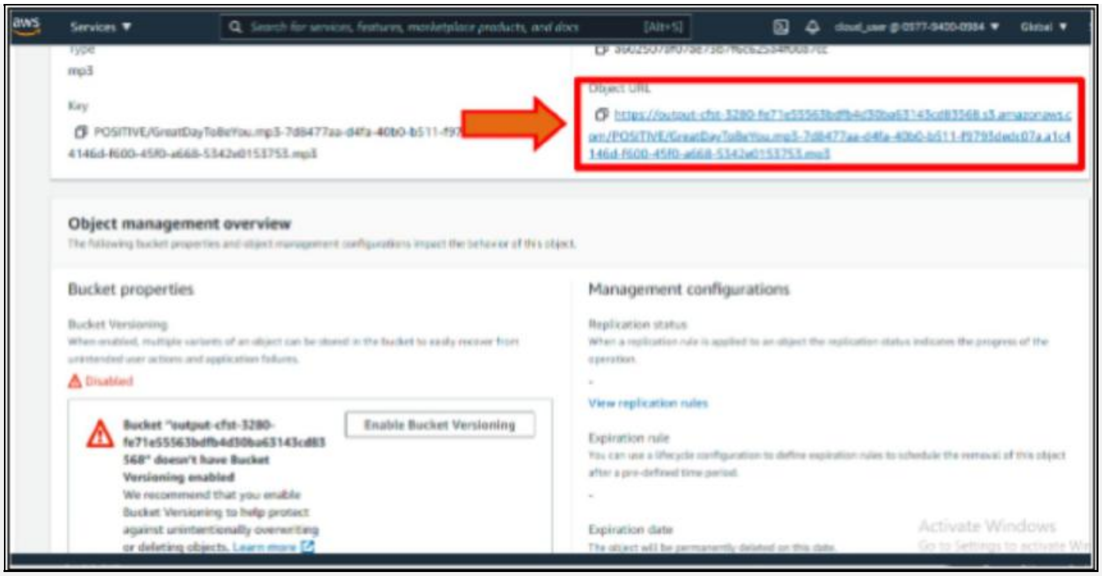
8. Click on **Actions** to make the file public.



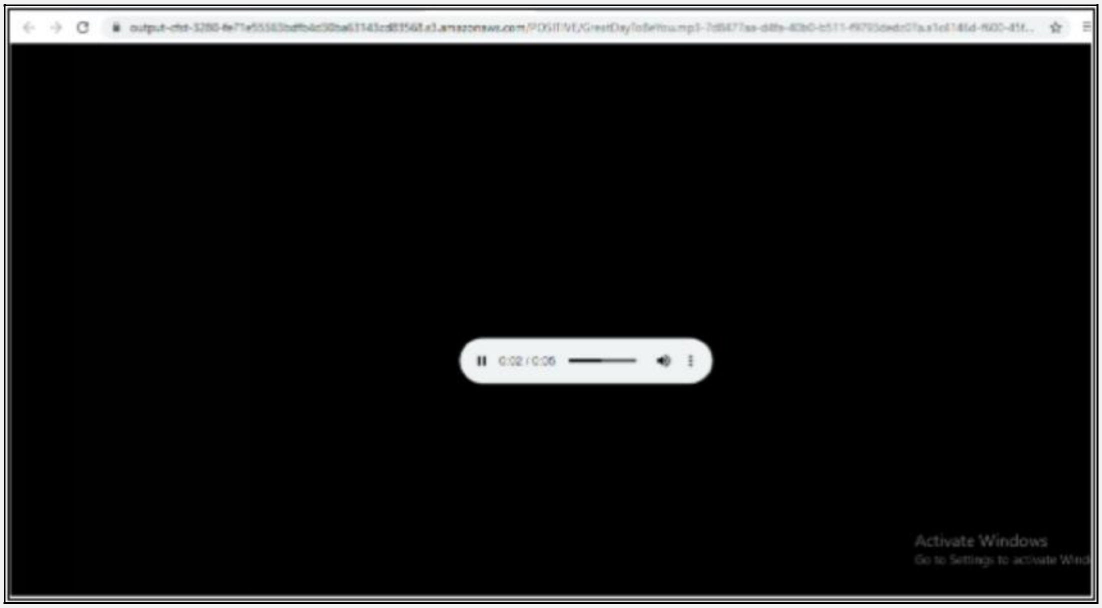
9. Click **Make Public** to the public the object.



10. Click on the object URL to listen to the translated audio in the Spanish Language.



11. The output translates audio.



Github Link: https://github.com/linuxacademy/content-aws-mls-c01/tree/master/CoordinateAIServicesWithStepFunctions

Note: If there is trouble executing the code, you also take help from the below-given code.

Code:

```
{
 "StartAt":"Start Transcribe",
 "States":{
  "Start Transcribe":{
   "Type":"Task",
   "Resource":"arn:aws:states:::lambda:invoke",
   "Parameters":{
                                        "FunctionName":"arn:aws:lambda:us-east-
1:756325582597:function:start-transcribe-lambda:$LATEST",
     "Payload":{
      "Input.$":"$"
     }
   },
   "Next":"Wait for Transcribe"
  },
  "Wait for Transcribe":{
    "Type":"Wait",
    "Seconds": 45,
    "Next":"Check Transcribe Status"
  },
  "Check Transcribe Status": {
   "Type":"Task",
   "Resource":"arn:aws:states:::lambda:invoke",
   "Parameters":{
                                        "FunctionName":"arn:aws:lambda:us-east-
1:756325582597:function:transcribe-status-lambda:$LATEST",
     "Payload":{
      "Input.$":"$"
     }
   },
    "Next": "Is Transcribe Complete"
  },
  "Is Transcribe Complete":{
    "Type":"Choice",
    "Choices":[
      {
        "Variable":"$.Payload.TranscriptionJobStatus",
        "StringEquals":"COMPLETED",
        "Next":"Transcript Available"
      },
```

```
        {
          "Variable":"$.Payload.TranscriptionJobStatus",
          "StringEquals":"FAILED",
          "Next":"Transcribe Failed"
        }
      ],
      "Default":"Wait for Transcribe"
    },
    "Transcript Available": {
  "Type": "Parallel",
  "Branches": [
    {
      "StartAt": "Translate Text",
      "States": {
        "Translate Text": {
  "Type": "Task",
  "Resource": "arn:aws:states:::lambda:invoke",
  "Parameters": {
                    "FunctionName":     "arn:aws:lambda:us-east-
1:756325582597:function:translate-lambda:$LATEST",
    "Payload": {
      "Input.$": "$"
    }
  },
  "End": true
}
      }
    },
    {
      "StartAt": "Comprehend Text",
      "States": {
        "Comprehend Text": {
  "Type": "Task",
  "Resource": "arn:aws:states:::lambda:invoke",
  "Parameters": {
                    "FunctionName":     "arn:aws:lambda:us-east-
1:756325582597:function:comprehend-sentiment-lambda:$LATEST",
    "Payload": {
      "Input.$": "$"
    }
  },
  "End": true
}
      }
    }
```

```
  ],
  "Next": "Convert Text to Speech"
},
"Convert Text to Speech": {
  "Type": "Task",
  "Resource": "arn:aws:states:::lambda:invoke",
  "Parameters": {
    "FunctionName": "arn:aws:lambda:us-east-1:756325582597:function:start-polly-lambda:$LATEST",
    "Payload": {
      "Input.$": "$"
    }
  },
  "End": true
},
    "Transcribe Failed": {
        "Type": "Fail"
    }
  }
}
```

About Our Products

Other products from IPSpecialist LTD regarding CSP technology are:

- AWS Certified Cloud Practitioner Study guide
- AWS Certified SysOps Admin - Associate Study guide
- AWS Certified Solution Architect - Associate Study guide
- AWS Certified Developer Associate Study guide
- AWS Certified Advanced Networking – Specialty Study guide
- AWS Certified Security – Specialty Study guide
- AWS Certified Big Data – Specialty Study guide
- AWS Certified Database – Specialty Study guide
- AWS Certified Machine Learning – Specialty Study guide
- Microsoft Certified: Azure Fundamentals
- Microsoft Certified: Azure Administrator
- Microsoft Certified: Azure Solution Architect
- Microsoft Certified: Azure DevOps Engineer
- Microsoft Certified: Azure Developer Associate

Microsoft Certified: Azure Security Engineer

Microsoft Certified: Azure Data Fundamentals

Microsoft Certified: Azure AI Fundamentals

Microsoft Certified: Azure Database Administrator Associate

Google Certified: Associate Cloud Engineer

Google Certified: Professional Cloud Developer

Microsoft Certified: Azure Data Engineer Associate

	Microsoft Certified: Azure Data Scientist
	Ansible Certified: Advanced Automation

Oracle Certified: OCI Foundations Associate

Oracle Certified: OCI Developer Associate

Oracle Certified: OCI Architect Associate

Oracle Certified: OCI Operations Associate

Kubernetes Certified: Application Developer

Other Network & Security related products from IPSpecialist LTD are:

- CCNA Routing & Switching Study Guide
- CCNA Security Second Edition Study Guide
- CCNA Service Provider Study Guide
- CCDA Study Guide
- CCDP Study Guide
- CCNP Route Study Guide
- CCNP Switch Study Guide
- CCNP Troubleshoot Study Guide
- CCNP Security SENSS Study Guide
- CCNP Security SIMOS Study Guide
- CCNP Security SITCS Study Guide
- CCNP Security SISAS Study Guide
- CompTIA Network+ Study Guide
- Certified Blockchain Expert (CBEv2) Study Guide
- EC-Council CEH v10 Second Edition Study Guide
- Certified Blockchain Expert v2 Study Guide