Organizational readiness and change management are crucial components in the transition to cloud computing. Here's how they intersect in the cloud age:

Assessment of Organizational Readiness:

- Before migrating to the cloud, organizations need to assess their readiness. This includes evaluating their current infrastructure, processes, and cultural readiness for cloud adoption.
- Understanding the organization's current state helps in planning for the transition effectively.

Change Management Planning:

- Change management strategies are essential for successful cloud adoption.
- This involves identifying key stakeholders, communicating the benefits of cloud computing, addressing concerns, and creating a roadmap for the transition.

Training and Skill Development:

- Cloud adoption often requires new skills and knowledge.
- Organizational readiness involves assessing the existing skill sets within the workforce and providing training programs to bridge any gaps.
- This ensures that employees are equipped to leverage cloud technologies effectively.

Cultural Shift:

- Moving to the cloud often entails a cultural shift within the organization. This may involve embracing new ways of working, such as collaboration, agility, and innovation.
- Change management strategies should focus on fostering a culture that supports cloud adoption and encourages experimentation and learning.

Managing Resistance to Change:

- Resistance to change is natural, and it's essential to address concerns and objections proactively.
- Change management processes should involve listening to employees' feedback, addressing their concerns, and demonstrating the benefits of cloud computing through pilot projects or proof-of-concepts.

Alignment with Business Objectives:

- Organizational readiness should align with the broader business objectives.
- Cloud adoption should not be seen as an isolated IT initiative but as a strategic enabler for achieving business goals such as scalability, cost optimization, and innovation.

Continuous Improvement:

- Cloud adoption is an ongoing process, and organizational readiness should be viewed as a journey rather than a one-time event.
- Continuous improvement processes should be established to monitor the effectiveness of cloud solutions, address challenges, and optimize processes over time.

In summary, organizational readiness and change management are critical for successful cloud adoption. By assessing readiness, planning for change, addressing cultural shifts, and aligning with business objectives, organizations can navigate the challenges of the cloud age effectively and realize the full potential of cloud computing.

Data security in the cloud is a multifaceted concern that requires careful attention and robust measures to ensure the confidentiality, integrity, and availability of data.

Here are some key aspects of data security in the cloud:

Encryption:

- Encrypting data both in transit and at rest is fundamental to cloud security.
- Data should be encrypted before transmission to the cloud provider's servers and should remain encrypted while stored in the cloud.
- Encryption keys should be managed securely, and strong encryption algorithms should be used to safeguard data from unauthorized access.

Identity and Access Management (IAM):

- Implementing strict controls over who can access cloud resources and data is essential.
- IAM solutions enable organizations to manage user identities, control access permissions, and enforce multi-factor authentication to prevent unauthorized access.

Network Security:

- Cloud environments should be protected by robust network security measures, including firewalls, intrusion detection and prevention systems (IDPS), and virtual private networks (VPNs).
- Network traffic within the cloud environment should be monitored and filtered to detect and prevent malicious activity.

Data Loss Prevention (DLP):

- DLP solutions help organizations prevent the unauthorized transmission or sharing of sensitive data.
- DLP policies can be configured to monitor and enforce rules regarding the handling of sensitive information, such as credit card numbers, personally identifiable information (PII), or intellectual property.

Security Compliance:

- Cloud providers often adhere to industry standards and compliance frameworks to ensure the security and privacy of customer data.
- Organizations should select cloud providers that comply with relevant regulations, such as GDPR, HIPAA, or PCI DSS, and implement additional security controls as necessary to meet their specific compliance requirements.

Security Monitoring and Logging:

- Continuous monitoring of cloud environments is essential for detecting and responding to security incidents in real-time.
- Security information and event management (SIEM) solutions can aggregate and analyze logs from various cloud services and infrastructure components to identify suspicious activities and potential security threats.

Backup and Disaster Recovery:

- **I**mplementing robust backup and disaster recovery solutions is critical for data protection in the cloud.
- Regular backups of data stored in the cloud should be performed to ensure data availability in the event of accidental deletion, corruption, or a security breach.

Security Training and Awareness:

- Educating employees about security best practices and raising awareness about potential threats is essential for mitigating human errors and vulnerabilities.
- Security training programs should cover topics such as phishing awareness, password hygiene, and data handling practices specific to the cloud environment.

Legal issues in cloud computing encompass a range of concerns related to data privacy, security, intellectual property, compliance, jurisdictional differences, and contractual agreements.

Here are some key areas:

Data Privacy and Security:

- Cloud providers must comply with various data protection laws such as GDPR in the EU or CCPA in California.
- Users need assurances that their data is handled securely and in compliance with applicable regulations.
- Data breaches and unauthorized access are significant concerns.

Data Location and Jurisdiction:

- Determining where data physically resides is crucial for legal compliance, especially when different jurisdictions have varying data protection laws.
- It raises questions about which country's laws apply and who has access to the data.

Data Ownership and Control:

- Cloud service agreements need to specify ownership and control of the data stored in the cloud.
- Users should understand their rights regarding their data and how it can be accessed, transferred, or deleted.

Intellectual Property (IP) Rights:

- Users must ensure that their intellectual property rights are protected when using cloud services.
- This includes addressing issues such as licensing agreements, copyright infringement, and protection of trade secrets.

Compliance and Regulatory Requirements:

- Industries such as healthcare and finance have specific regulatory requirements (e.g., HIPAA, PCI DSS).

- Cloud users must ensure that their provider meets these compliance standards and that they themselves remain compliant when using cloud services.

Service Level Agreements (SLAs):

- Cloud service providers offer SLAs detailing the level of service they guarantee, including uptime, data availability, and response times.
- Users need to understand these agreements and their implications for liability and compensation in case of service disruptions.

Vendor Lock-In:

- Users may face challenges migrating data and applications between cloud providers due to proprietary formats, APIs, and dependencies.
- This raises concerns about dependence on a single provider and the associated risks.

Contractual Issues:

- Cloud service contracts must clearly define the rights and responsibilities of both parties, including service scope, pricing, termination clauses, and dispute resolution mechanisms.

Audit and Compliance Monitoring:

- Users need mechanisms to audit and monitor cloud services to ensure compliance with contractual agreements, regulatory requirements, and security standards.

Third-party Services and Subcontractors:

- Cloud providers often engage third-party services or subcontractors to deliver their services.
- Users should be aware of these arrangements and how they impact data security, privacy, and compliance.

GCP stands for Google Cloud Platform, which is a suite of cloud computing services provided by Google.

It offers a wide range of services including infrastructure as a service (IaaS), platform as a service (PaaS), and software as a service (SaaS) offerings.

Some key services and features of Google Cloud Platform include:

Compute:

- Services like Google Compute Engine (virtual machines), Google Kubernetes Engine (container orchestration), and Google App Engine (platform for building and hosting web applications).

Storage:

- Google Cloud Storage provides scalable object storage suitable for a wide range of use cases.
- It includes options such as Standard, Nearline, and Coldline storage tiers.

Databases:

- GCP offers managed database services like Cloud SQL (for MySQL, PostgreSQL, and SQL Server), Cloud Spanner (a horizontally scalable relational database), and Firestore (a scalable NoSQL document database).

Networking:

- Google Cloud Platform provides various networking services including Virtual Private Cloud (VPC), Cloud Load Balancing, and Cloud DNS for scalable, reliable, and secure networking solutions.

Big Data and Machine Learning:

- GCP includes services like BigQuery for big data analytics, TensorFlow for machine learning, and AI Platform for building and deploying machine learning models.

Identity and Security:

- Google Cloud IAM (Identity and Access Management) allows users to manage access control and permissions.
- GCP also offers security services like Cloud Identity-Aware Proxy, Cloud Key Management Service, and Cloud Security Command Center.

Management Tools:

- Google Cloud Console provides a web-based interface for managing GCP resources. Additionally, services like Stackdriver offer monitoring, logging, and diagnostics capabilities.

Serverless Computing:

- GCP offers serverless computing options such as Cloud Functions for event-driven functions, Cloud Run for running containers without managing infrastructure, and Firebase for mobile and web application development.

Azure is a cloud computing platform and service provided by Microsoft.

It offers a wide range of cloud services, including infrastructure as a service (IaaS), platform as a service (PaaS), and software as a service (SaaS) offerings,

similar to other major cloud providers like Amazon Web Services (AWS) and Google Cloud Platform (GCP).

Key services and features of Microsoft Azure include:

Compute:

- Azure Virtual Machines (VMs) allow users to run virtualized Windows and Linux servers in the cloud.
- Azure Kubernetes Service (AKS) provides managed Kubernetes clusters for containerized applications, while Azure App Service offers a platform for building and hosting web applications.

Storage:

- Azure Storage provides scalable object, file, and block storage services, including Blob storage, File storage, and Azure Disk Storage.

Databases:

- Azure offers a variety of managed database services, including Azure SQL Database (a fully managed relational database service), Azure Cosmos DB (a globally distributed NoSQL database), and Azure Database for MySQL, PostgreSQL, and MariaDB.

Networking:

- Azure Virtual Network (VNet) enables users to create private networks in the cloud, while Azure Load Balancer and Azure Application Gateway provide load balancing and traffic management capabilities.
- Azure DNS offers domain name system (DNS) hosting.

Big Data and Analytics:

- Azure includes services like Azure Synapse Analytics (formerly SQL Data Warehouse) for data warehousing, Azure HDInsight for big data

analytics, and Azure Databricks for collaborative Apache Spark-based analytics.

Artificial Intelligence (AI) and Machine Learning:

- Azure offers AI services such as Azure Machine Learning, Azure Cognitive Services (pre-built AI models for vision, speech, language, and decision), and Azure Bot Service for building intelligent bots.

Identity and Security:

- Azure Active Directory (Azure AD) provides identity and access management services, while Azure Security Center offers threat protection and security management for Azure resources.

Development Tools:

- Azure DevOps provides a suite of tools for software development, including version control, build automation, and release management. Azure also supports various programming languages, frameworks, and development environments.

Internet of Things (IoT):

- Azure IoT Hub enables bi-directional communication between IoT devices and the cloud, while Azure IoT Central offers a fully managed IoT application platform.

Serverless Computing:

- Azure Functions allows users to run event-driven functions without managing infrastructure, while Azure Logic Apps provides workflow automation and integration capabilities.

Azure architecture layers

Azure architecture can be conceptualized in terms of several layers, each serving a specific purpose and offering a set of services. The typical layers in Azure architecture:

Physical Infrastructure Layer

- At the lowest level, Azure operates a global network of data centers comprising servers, networking equipment, and storage devices.
- These data centers are distributed across multiple regions worldwide, providing the foundation for Azure's cloud services.

Networking Layer

- This layer includes services and features related to networking and connectivity within Azure and between Azure and external networks. Key components include:

Virtual Network (VNet)

- Provides isolated networking environments for Azure resources, allowing users to define subnets, routing, and network security policies.

Azure ExpressRoute

- Offers private, dedicated connectivity between Azure data centers and on-premises infrastructure, bypassing the public internet for enhanced security and performance.

Load Balancers

- Distributes incoming network traffic across multiple VM instances or services to ensure high availability and scalability.

Compute Layer

- The compute layer comprises services for provisioning and managing virtual machines, containers, and serverless computing resources.Key components include:
- **Virtual Machines (VMs)**

- Infrastructure-as-a-Service (IaaS) offering for running applications and workloads on customizable VM instances.
- **Azure Kubernetes Service (AKS):** Managed Kubernetes service for deploying, managing, and scaling containerized applications.
- **Azure App Service:** Platform-as-a-Service (PaaS) offering for building, deploying, and scaling web and mobile applications without managing underlying infrastructure.
- **Azure Functions:** Serverless compute service for running event-driven functions in response to triggers, automatically scaling based on demand.

Storage Layer

- This layer encompasses various storage services for storing and managing data in Azure. Key components include:
  - **Azure Blob Storage:** Scalable object storage for unstructured data, such as documents, images, and videos.
  - **Azure Files:** Managed file shares for cloud-native and legacy applications running on VMs.
  - **Azure Disk Storage:** Persistent block storage for VMs and applications, available in different performance tiers (HDD, SSD, Ultra Disk).

Data Services Layer

- Data services layer includes managed database and analytics services for storing, processing, and analyzing data in Azure. Key components include:
  - **Azure SQL Database**: Fully managed relational database service based on Microsoft SQL Server, offering high availability, scalability, and security.
  - **Azure Cosmos DB:** Globally distributed NoSQL database service for building highly responsive and scalable applications with low-latency data access.
  - **Azure Data Lake Storage:** Scalable and secure data lake storage for big data analytics and machine learning workloads.

Identity and Security Layer

- This layer includes services for managing identity, access control, and security within Azure. Key components include:
    - **Azure Active Directory (AAD):** Identity and access management service for authenticating and authorizing users to access Azure resources.
    - **Azure Key Vault:** Securely store and manage cryptographic keys, secrets, and certificates used by cloud applications and services.
    - **Azure Security Center:** Unified security management and threat protection service for Azure resources, providing advanced threat detection and actionable insights.

Management and Monitoring Layer

- This layer encompasses services for managing, monitoring, and automating Azure resources and workloads. Key components include:
    - **Azure Monitor:** Comprehensive monitoring service for Azure resources, applications, and infrastructure, offering metrics, logs, alerts, and insights for performance and availability monitoring.
    - **Azure Resource Manager (ARM):** Management service for deploying, managing, and organizing Azure resources using declarative templates and APIs.
    - **Azure Automation:** Service for automating repetitive tasks and workflows in Azure using runbooks, DSC configurations, and PowerShell scripts.

Google Cloud Platform (GCP) architecture is designed to provide scalable, reliable, and high-performance cloud services to users around the world. Here's an overview of layers and features of the GCP architecture:

Infrastructure Layer

- **Google Data Centers:**
  - GCP operates a global network of data centers located in various regions and zones around the world.
  - These data centers are interconnected through Google's high-speed, low-latency network infrastructure.
- **Compute Resources:**
  - GCP's infrastructure provides virtualized compute resources in the form of VM instances, containers, and serverless functions.
  - These resources are hosted on Google's Compute Engine, Kubernetes Engine, and App Engine platforms.

Networking Layer

- **Virtual Private Cloud (VPC):**
  - GCP users can create isolated virtual networks to host their cloud resources. VPCs enable fine-grained control over network configuration, routing, and security.
- **Load Balancing:**
  - GCP offers global load balancing services to distribute incoming traffic across multiple instances or regions, ensuring high availability and performance for applications.
- **Interconnect and Peering:**
  - GCP allows users to establish private connectivity between their on-premises networks and GCP resources using dedicated interconnects or VPN tunnels.
  - Additionally, GCP offers peering options for connecting to other networks and cloud providers.

Storage Layer

- **Cloud Storage:**
    - GCP's object storage service provides scalable, durable, and highly available storage for unstructured data.
    - Users can store and retrieve data with low latency and high throughput.
- **Cloud SQL, Bigtable, Spanner:**
    - GCP offers managed database services for various use cases, including relational databases (Cloud SQL), NoSQL databases (Cloud Bigtable), and globally distributed SQL databases (Cloud **Spanner).**

Big Data and Machine Learning Layer

- **BigQuery:**
    - GCP's fully-managed data warehouse service enables users to analyze large datasets using SQL queries with high performance and scalability.
- **Dataflow, Dataproc:**
    - GCP provides managed services for batch and stream processing, such as Cloud Dataflow and Cloud Dataproc, which allow users to run Apache Beam and Apache Spark jobs at scale.
- **AI Platform:**
    - GCP offers a suite of machine learning services for building, training, and deploying models, including TensorFlow, AutoML, and AI Platform Prediction.

Identity and Security Layer

- **Identity and Access Management (IAM):**
    - GCP's IAM service provides centralized access control and permissions management for GCP resources, allowing users to define fine-grained access policies.
- **Encryption:**
    - GCP encrypts data at rest and in transit by default, using strong encryption algorithms and key management services.
- **Security Services:**

- GCP offers a range of security services and features, including DDoS protection, firewall rules, and threat detection, to help users protect their cloud workloads and data.

Management and Monitoring Layer

- **Stackdriver:**
  - GCP's monitoring, logging, and diagnostics suite provides visibility into the performance, availability, and health of GCP resources and applications.
- **Deployment Manager:**
  - GCP offers Deployment Manager, a service for automating the creation, deployment, and management of GCP resources using declarative configuration files.

Developer Tools Layer

- **Cloud SDK, APIs:**
  - GCP provides a set of developer tools, including the Cloud SDK and APIs, for interacting with GCP services programmatically and automating cloud workflows.
- **Development Platforms:**
  - GCP supports various development platforms and frameworks, including Java, Python, Node.js, and Go, enabling developers to build and deploy applications using their preferred languages and tools.

Migration of website to cloud using Docker

Developer develops the application(Frontend & Backend) and pushed to git hub. Git hub will have two repositories Frontend and backend.

These two repositories are going to be cloned into our EC2 instance as front end repository and backend repository. When we are cloning this we will be injecting two files named **Docker** which has six attributes and **.env** file.

To make it accessible to customers this front end and back end we are going to convert these folders or project code into **Docker images** one is called as **client** the other one is called as **server**

In server our backend code will be converted whereas in Client our front end code will be converted. To make it accessible we'll run this Docker image in a **container**. Once it is running in docker environment the application can be accessible by the customers all the clients whoever wants to.

This is what the entire structure is where developer is developing some code he will be pushing into GitHub from GitHub we will be cloning onto our EC2 instance. In EC2 instance we will be installing our Docker and in the docker we will be converting the projects into images and in the docker containers we'll be running the applications. This is how a application running on a local server is been deployed or migrated into a cloud environment

so let's have an Hands-On session on how to migrate a website from local server to Cloud . We are in our Cloud AWS environment where the first thing is you need is to search for an EC2. Just click on EC2 instance . we will launch an EC2 instance I am naming this instance as **Dockerdemo** . Selecting the operating system as **Ubuntu** enhancing my **T2 micro** instance type to **T2 large.** Creating a keypair I am naming it as same **DockerDemoKey**

In network settings I will modify VPC subnet. **Auto-assign public IP** I will enable it and will add security group by clicking **Add security group rule** change **Type** to allowing **All traffic** and in **Source type** to **Anywhere.** In **Configuration storage** enhancing my storage to 30 GB of capacity and now click on **Launch instance**

Once the instance has launched just wait until it turns into running State. Once the instance state gets into running click on **Instance ID** and click on **Connect**. Under SSH will be copying the link to get connected to EC2 instance.

Open the command prompt move to downloads my pem key has got downloaded under downloads . Now continue connecting to EC2 instance . Got connected and could see at certain IP address

Now the first thing will do is cloning my backend repository from GitHub using the command **git clone** https://github.com/procareer3fwd/realgrandebackend.git which is URL of your GitHub repository. Once this has been done will be be cloning frontend from GitHub using command get clone **git clone** https://github.com/procareer3fwd/realgrandefrontend.git. Now you could see that your realgrande back end as well as realgrande front end has got downloaded

One thing before installing of Docker you need to update your ubuntu because as ubuntu is an open-source operating system it's keep on updating its libraries and packages so to be updated we need to update our operating system which is done using command **sudo apt update** .

Next it's time to install your Docker which is done using command **sudo apt -y install docker.io.** This command installs your Docker on your EC2 instance. Once Docker has been installed just cross verify whether Docker has been properly installed using command **sudo docker version**. If you could see **client** and **server** then it indicates that Docker has installed properly.

Now check whether you do have any images in your Docker or not using command **sudo docker images. D**on't have any docker images . To have docker image we need to inject **.env** and **Dockerfile** So the first thing is to get into your backend directory by typing **cd realgrandebackend/** and in backend you need to inject a **.env .** First create .env file by using command **nano .env** and paste these lines

MONGODBURL="mongodb+srv://fsd04.2hxrdca.mongodb.net/realgrande?retryWrites=true&w=majority"
DBUSERNAME=procareer3
DBPASSWORD=ISobjBDohsFqEAqg
FRONTENDURI="http:// 3.82.247.96"

and do remember you need to change your public IP address over here which you will get into from your EC2 instance. Get your public IP from EC2 instance which is **DockerDemo** for me here . After pasting in .env file please type ctrl+ X hit Y and hit enter to save and quit

So I was talking about the docker file just type command **ls** to see it. It is already present in your real grande backend as well as frontend this is the file. Now type **cat Dockerfile**. Here we do have six attributes all the attribute names will be in upper case

1. **FROM node** it means that to run your react applications you require some sort of environment where operating system as well as the middle Ware should be present.node which has nodejs in it with the operating system system , Alpine is the operating system and nodejs is the environment or server on which your application is going to run.So in order to just use this what I'm doing is **From node.**

   what Docker file will do is it will search its local machine where node image is present if it is not then it will get into dockerhub and it pulls this particular image dockerhub which has its own operating system named Alpine and above that at the middleware which is running that is nodejs
2. The second attribute creating my **own workspace** where all the files are going to be copied under this workspace
3. Then I will be installing my **npm** (node package manager)
4. In the fourth command I'm **exposing** this back end on the port number 5,000 and will be running
5. I will be starting this particular application using **npm start**

These are the six attributes which are going to be mentioned in my Docker file

Now how to build the image presently we don't have any images in our Docker how to build an image use command **sudo docker build -t backend .**

It means should build based on a Docker file which is present in the current directory hit enter you can see that it is pulling an image from dockerhub named node pull complete you can see this is step one of six let us wait till it completes all the Steps

Step two it has created its own working directory / app step two it is creating it has created yeah it has Exposed on port number and it has started its image

Let's see content image running in a container using command **sudo docker ps** .Presently not running. To make that image to run in this container I'll be using the command

sudo docker run -d -p 2001:5000 backend

-d means in detachable mode **-P** the port on which it has to be exposed I can say it as 2001 colon 5000 and which image you want to run in the container the name of the image is backend as which we had created just now the backend successfully tagged backend

Just check it out whether that is running in a container using command **sudo docker ps** . yes it has its **own container ID** and the name of the image which is running in the container is **backend** it is been exposed on port number 2001 and 5,000 is a port which is internal if I want to access then I need to use 2001 port number. It is Port binding concept

So let's check whether we can connect to our backend through a browser paste the public IP **3.82.247.96:2001/api** hit enter . Yeah we can see Json file it means that we could connect to our backend repository

Let's redirect to our frontend now to convert our front end into an image. Type **cd** and type **cd realgrandefrontend.** So first thing is we have to check whether we do have Docker file type **ls.** yes we do it also . Now type **cat Dockerfile** we see has the same six attributes but this will be exposed on port number 3,000 the frontend

Let's create a .env file in our front end and paste **REACT_APP_BACKEND_URL=http://3.82.247.96:2001/api** this line . I have changed you can see that I have been given 2001 my back end is been exposed on port number 2001 so if you are exposing any other Port then you need to mention over here . ctrl+ X and Y hit enter

once the .env file is been done then you can build your images .So Docker build

sudo docker build -t frontend .

This time your node has been already downloaded in the image node so it has not taken too much time it will check its local repository. node image is already present in its local repository. So it will not again get into the docker Hub and keep on downloading the node image because in our previous back end itself we had downloaded our node image we had pulled it from the docker Hub once this has been done then just run this frontend image and check whether you can accessyour front end or not by typing **sudo docker images.** Now my image has been successfully created . If you want you can check it it through images . You can see **frontend** image and **node** image and **backend** back end and **frontend** we had created node we had pulled it from the docker Hub

Just run it by using command **sudo docker run -d -p 80:3000 frontend** and port on which you are going to expose is 80:3000 is a port and which image you want to run , you want to run the frontend image hit enter. Just check it out whether that has been running using **Sudo docker ps** . You can see frontend has been running with a **container ID** and it is exposed on port number **80**

let's get back to our browser copy the public IP address **3.82.247.96:2001**hit enter as at is the default Port I need not to mention my port number you can see you can access your front end you can retrieve the data just like can't all these data are being retrivied fromthe mongodb database.You can login, you can sign up everything

Steps to Migrate a website from local server to Cloud

==========

1) Launch an EC2 Instance on AWS console
2) Open the command prompt and Connect to EC2 Instance

3) clone the git repos

backend

==========

tinyurl.com/cs1bekmit

git clone https://github.com/procareer3fwd/realgrandebackend.git

frontend

==========

tinyurl.com/cs1fekmit

git clone https://github.com/procareer3fwd/realgrandefrontend.git

4) Update the Ubuntu

sudo apt update

5) Install the Docker

sudo apt -y install docker.io

6) Check the docker images

sudo docker images

7) change the directory to backend

cd realgrandebackend/

8) create an .env file

nano .env

9) Copy Paste the lines in .env file

MONGODBURL="mongodb+srv://fsd04.2hxrdca.mongodb.net/realgrande?retryWrites=true&w=majority"

DBUSERNAME=procareer3

DBPASSWORD=ISobjBDohsFqEAqg

FRONTENDURI="http://3.82.247.96"

10) Build the docker image for backend

sudo docker build -t backend_server .

11) Check for any images running in the container

sudo docker ps

12) Now run the backend_server docker image in the container

sudo docker run -d -p 2001:5000 backend_server

13) Now try to access the backend_server on the browser

<EC2_PUBLIC_IP_ADDRESS>:2001/api

Eg: **3.82.247.96:2001/api** // 3.82.247.96 is the public ip of my EC2 instance

14) Now change the directory to frontend

ubuntu@ip-172-31-1-17:~/realgrandebackend$ cd

ubuntu@ip-172-31-1-17:~$ cd realgrandefrontend/

ubuntu@ip-172-31-1-17:~/realgrandefrontend$

15) Now create a .env file

nano .env

16) Copy paste the line in .env file

REACT_APP_BACKEND_URL="http:// 3.82.247.96 :2001/api"

17) Build the docker image for frontend

sudo docker build -t frontend .

18) Check for any images running in the container

sudo docker ps

19) Now run the backend_server docker image in the container

sudo docker run -d -p 80:3000 frontend

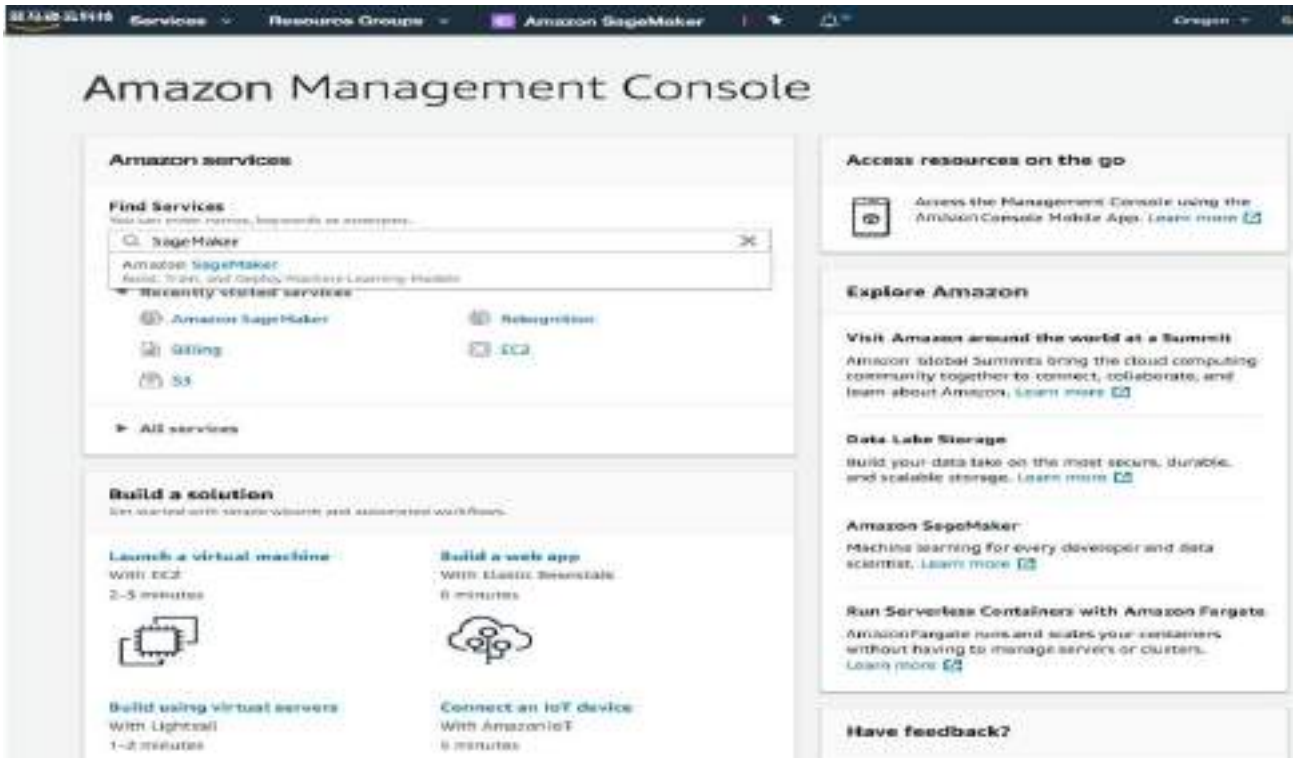
20) Now try to access the frontend on the browser

<EC2_PUBLIC_IP_ADDRESS>

Eg: **3.82.247.96** //This is public IP of my EC2 instance

Step 1. Enter the SageMaker console

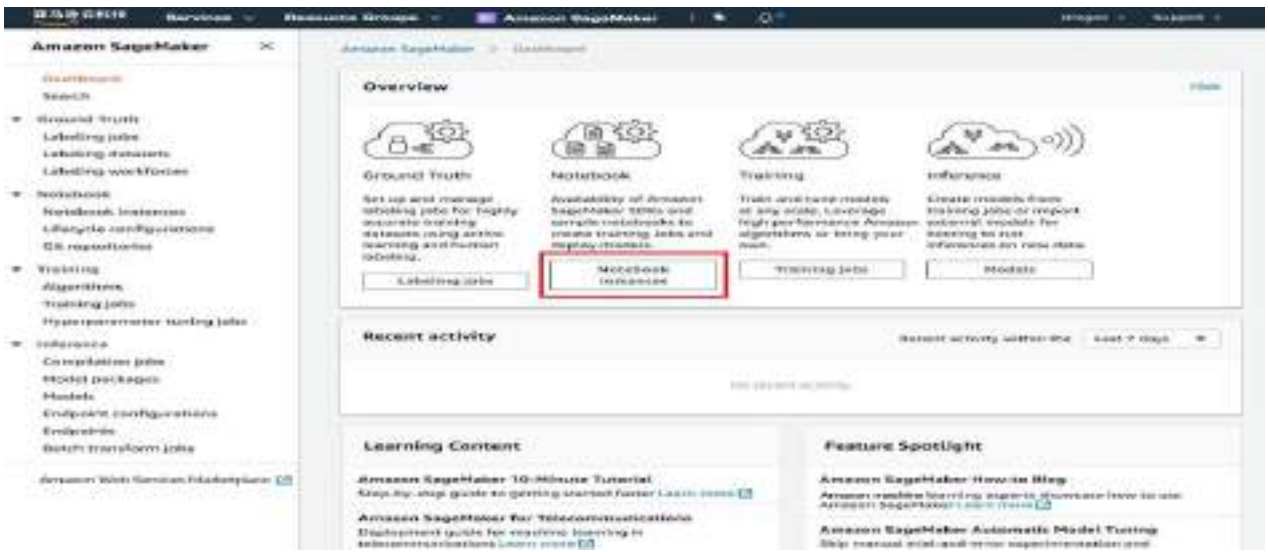
Navigate to the Amazon SageMaker console.Open the **Amazon Web Services Management Console**. Next, begin typing SageMaker in the search bar and select Amazon SageMaker to open the service console.



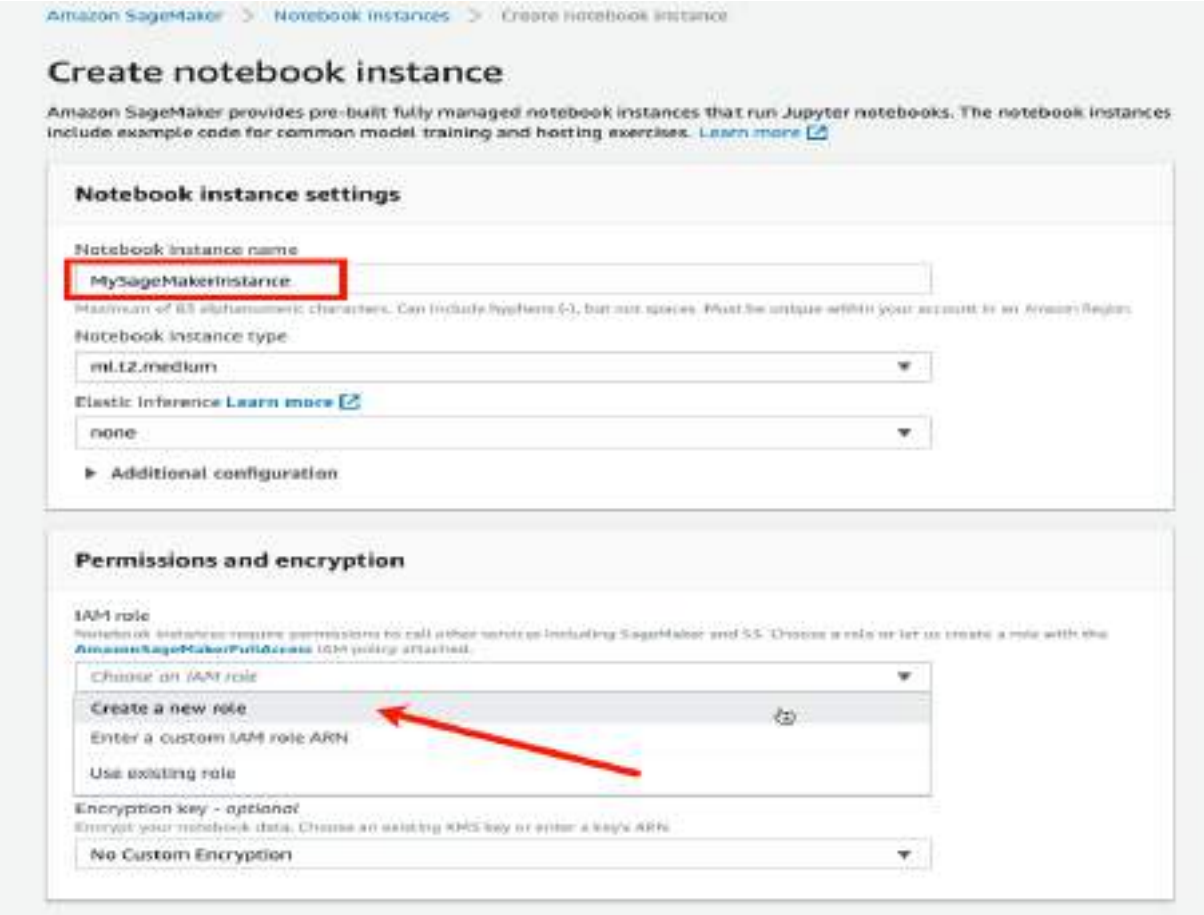
Step 2. Create a SageMaker notebook instance

In this step, you will create a SageMaker notebook instance.

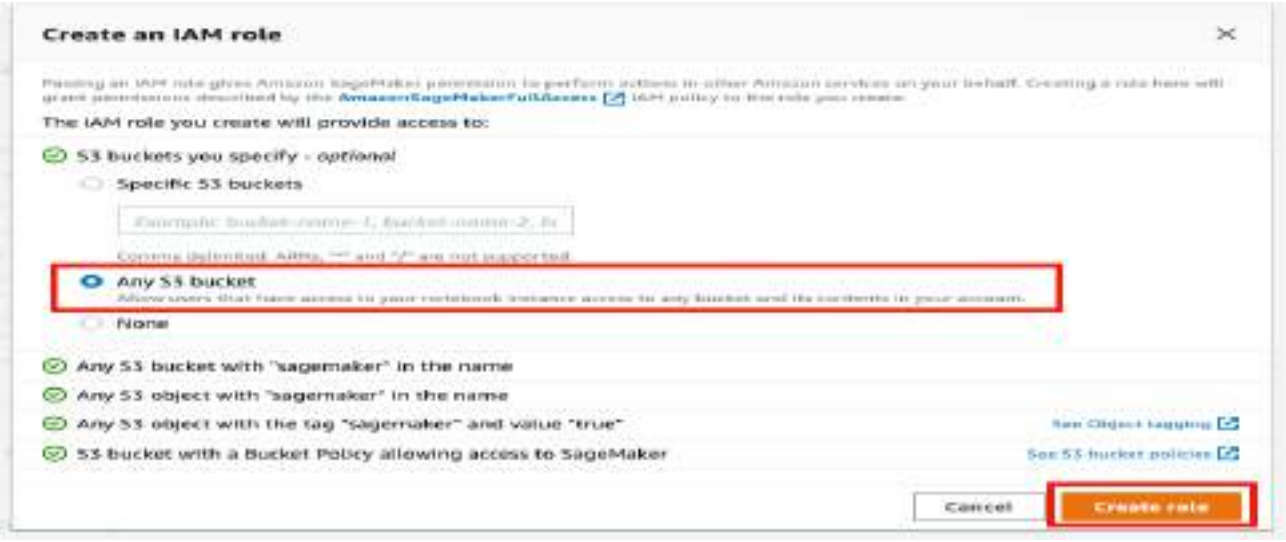
2a. From the Amazon SageMaker dashboard, select Create notebook Instance.



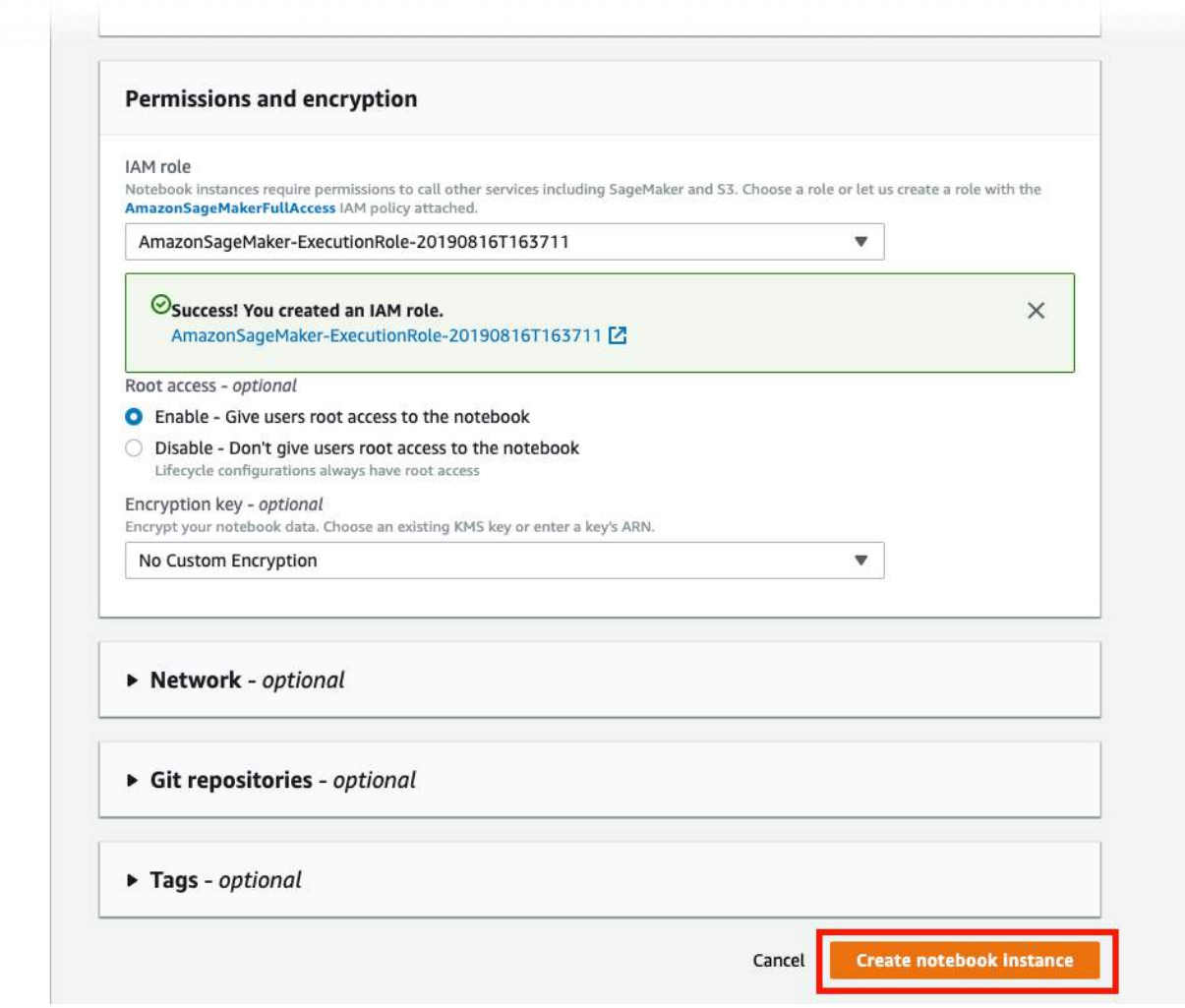
2b. On the **Create notebook instance** pane, in **Notebook instance name** field type *MySageMakerInstance*. So that your new instance can securely access S3 and other Amazon Web Services services, SageMaker can create a new IAM role with the right permissions and assign it to your instance for you. Instruct SageMaker to create this IAM role by selecting **Create a new role** from the **IAM role** drop down.



2c. On the **Create an IAM role** box, select **Any S3 bucket**. This will allow your SageMaker instance to access all the S3 buckets in your account. Now, select **Create role**.



2d. Notice that SageMaker made a role called *AmazonSageMaker-ExecutionRole-**** for you. On the **Create notebook instance** panel, you can also optionally place your instance in VPC, set a lifecycle configuration, and an encryption key. Here we leave the rest of the fields with the default options. Choose **Create notebook instance.**

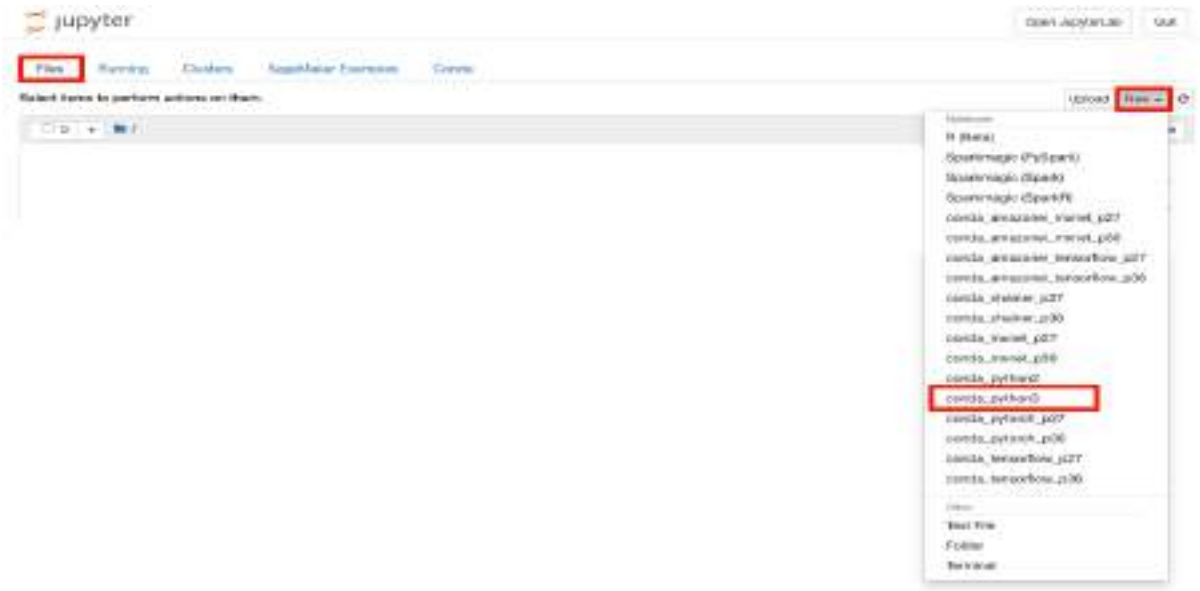


Step 3. Prepare the data

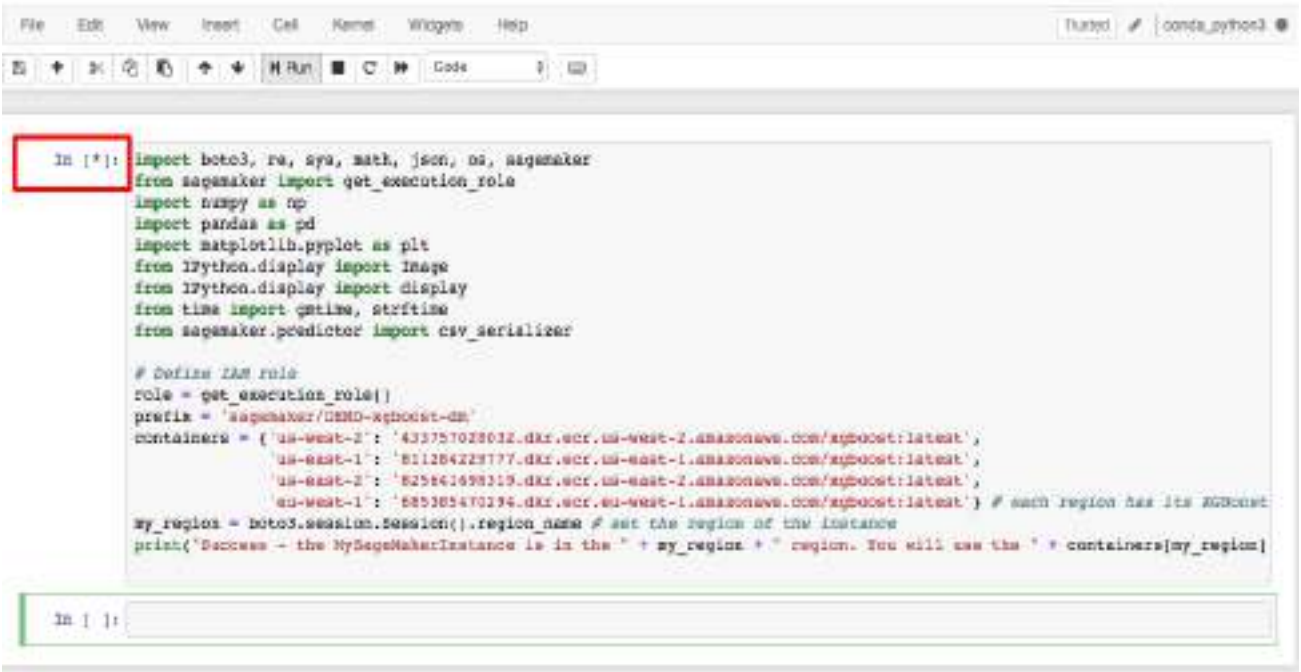
In this step you will use your Amazon SageMaker notebook to preprocess the data you need to train your machine learning model.

3a. On the **Notebook instances** pane, after *MySageMakerInstance* has transitioned from **Pending** to **InService,** select **Open** from the **Actions** column.

3b. After Jupyter opens up *MySageMakerInstance*, in the **Files** tab, click on **New**, and then choose **conda_python3**.



3c. To prepare the data, train the machine learning model, and deploy it, you will need to import some libraries and define a few environment variables in your Jupyter notebook environment. Copy the following code into the code cell in your instance and select **Run**.



3d. You will need a S3 bucket to store your training data once you have processed it. Copy and **Run** the following code to create a S3 bucket into the next code cell in your notebook.

Because S3 bucket names must be globally unique and have some restrictions, change the ***your_s3_bucket_name*** string in the code to a unique string. In your notebook, select **Run**.

If you don't receive a success message after running the code, change the bucket name and try again

3e. Next, you need to download the data to your SageMaker instance and load it into a dataframe. Copy and **Run** the following code:

3f. Then, shuffle the data and split it into training data and test data. The **training data** (70% of customers) will be used during an iterative cycle called gradient optimization to learn model parameters and infer the class label from input features with the least possible error. The **test data** (30%) will be used to evaluate the performance of the model. Copy then paste the following code and select **Run** to shuffle and split the data:

Step 4. Train the model from the data

In this step, you will train your machine learning model with the training dataset.

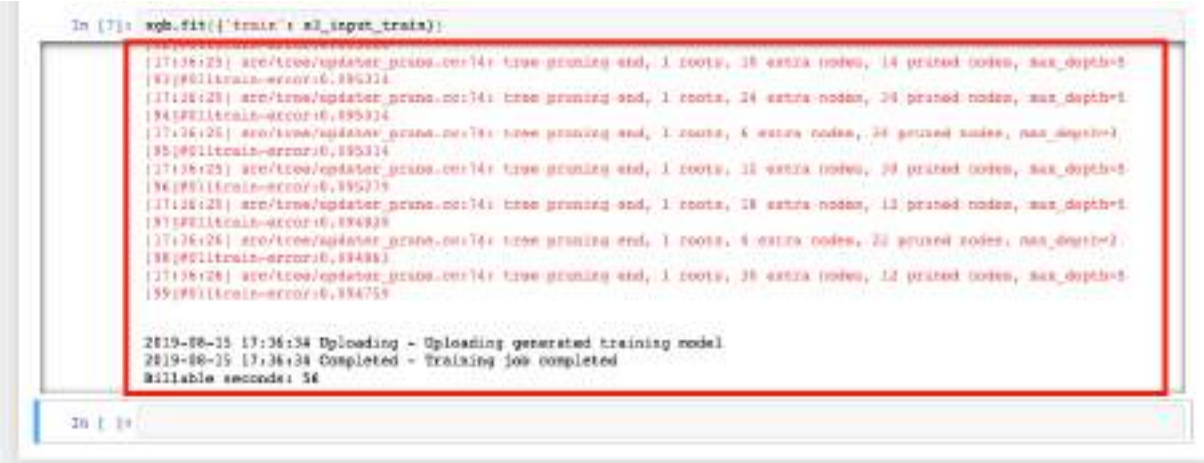
4a. To use a SageMaker pre-built XGBoost model, you will need to reformat the header and first column of the training data and load the data from the S3 bucket. Copy then paste the following code into a new code cell and select **Run** to reformat and load the data:

```
sess = sagemaker.Session()

xgb = sagemaker.estimator.Estimator(containers[my_region],role,
train_instance_count=1,
train_instance_type='ml.m4.xlarge',output_path='s3://{}/{}/output'.format(buck
et_name, prefix),sagemaker_session=sess)

xgb.set_hyperparameters(max_depth=5,eta=0.2,gamma=4,min_child_weight=6,subsamp
le=0.8,silent=0,objective='binary:logistic',num_round=100)
```

4c. With the data loaded and XGBoost estimator set up, train the model using gradient optimization on a ml.m4.xlarge instance by pasting the code and selecting **Run**.



Step 5. Deploy the model

In this step, you will deploy the trained model to an endpoint, reformat then load the CSV data, then run the model to create predictions.

5a. To deploy the model on a server and create an endpoint you can access, copy, paste, and **Run** the following code:

```
In [9]: xgb_predictor = xgb.deploy(initial_instance_count=1,instance_type='ml.m4.xlarge')

INFO:sagemaker:Creating model with name: xgboost-2018-07-13-14-29-39-425
INFO:sagemaker:Creating endpoint with name xgboost-2018-07-13-14-25-03-272

--------------------------------------------------------------!
```

5b. To predict whether customers in the test data enrolled for the bank product or not, copy, paste, and select **Run** the follow code:

```
In [20]: test_data_array = test_data.drop(['y_no', 'y_yes'], axis=1).values #load the data into an array
         xgb_predictor.content_type = 'text/csv' # set the data type for an inference
         xgb_predictor.serializer = csv_serializer # set the serializer type
         predictions = xgb_predictor.predict(test_data_array).decode('utf-8') # predict!
         predictions_array = np.fromstring(predictions[1:], sep=',') # and turn the prediction into an array
         print(predictions_array.shape)

(12357,)
```