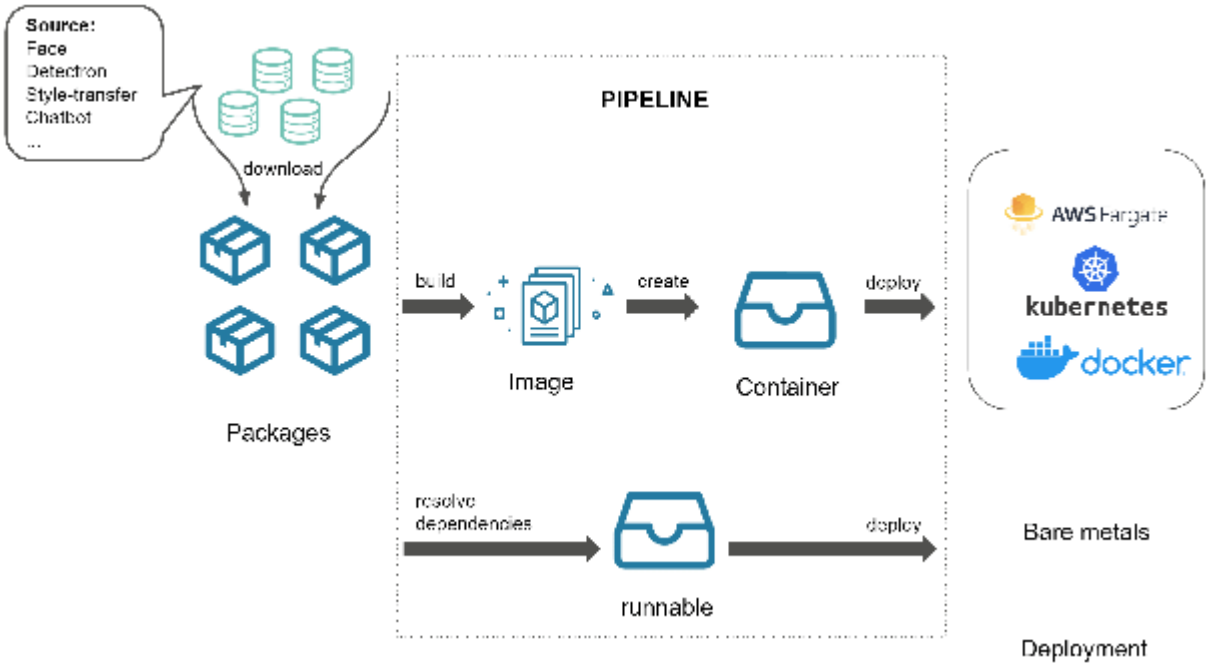
An extensive research on MLOps :-

Machine Learning Models Deployment on Cloud Platforms – AWS SageMaker v/s Docker

1. ML Model Deployment

Machine learning model deployment is the process of integrating a machine learning model into a production environment.



The process typically includes the following steps:

1. **Model development and training**

Data scientists train the model on existing data in a development environment.

2. **Model validation**

Data scientists test the model using new or unseen data to evaluate how well it might perform in production.

3. **Model deployment**

Data scientists and MLOps engineers integrate the model into the production environment.

Some **challenges** that may arise when deploying machine learning models include:

- Knowledge gaps: ML models are typically built and deployed by different teams.
- Infrastructure: Lack of robust infrastructure can slow down the process.
- Scale: Models will likely need to grow over time.

Here are some strategies for deploying machine learning models:

- Rolling deployment: A method for gradually updating a model without any downtime.
- Shadow deployment: A deployment strategy where applications are deployed to a separate environment before the live deployment.
- MLflow: An open source platform for managing the end-to-end machine learning lifecycle.

Here are some steps for deploying a model locally:

1. Set up a virtual environment and install the required packages.
2. Build and save the model.
3. Serve the model locally.
4. Create a folder structure for files.
5. Build a docker image.
6. Spin up a container to run the application.
7. Test the model endpoint.

2. ML Models on Cloud

Machine learning (ML) models are programs that can find patterns or make decisions from a previously unseen dataset.

Cloud ML platforms provide the compute, storage, and services required to train ML models.

Here are some cloud platforms for machine learning:

- Google Cloud AI Platform: Good for large-scale machine learning tasks
- Google Cloud Machine Learning: An AI platform for building, deploying, and managing machine learning models
- BigQuery ML: A cloud-based data warehouse that allows data analysts to build and run models using existing business intelligence tools and spreadsheets
- Amazon SageMaker: A cloud provider that offers various services and features for machine learning
- Microsoft Azure Machine Learning: A cloud provider that offers various services and features for machine learning
- IBM Watson Studio: A cloud provider that offers various services and features for machine learning

Here are some steps to deploy a trained ML model:

1. Persist the model file
2. Create a Dockerfile
3. Build a docker container
4. Create a REST API using framework like Flask, FastAPI, Django etc.
5. Deploy the API on a cloud service like Azure Web Services, AWS Fargate, Google Vertex, etc.

3. Cloud for Machine Learning

Leading ML(Ops) Platforms:-

- AWS SageMaker - Amazon's Machine Learning Platform.
- Databricks - The Big Data Analytics Platform.
- Vertex AI - Google's Machine Learning Platform.
- Qwak - The End to End MLOps Platform.

4. MLOps

Machine learning operations (MLOps) is a set of engineering practices that involve the development and use of machine learning models. MLOps is a combination of the words "**machine learning**" and the continuous development practice of **DevOps** in software.

MLOps aims to simplify the AI lifecycle from start to finish. It involves collaboration between developers, operations, and data science, and can include everything from data pipelines to model production.

The primary benefits of MLOps are **efficiency**, **scalability**, and **risk reduction**. MLOps can add discipline to the development and deployment of machine learning models, making the process more reliable and productive.

The key phases of MLOps are:

- Data gathering
- Data analysis
- Data transformation/preparation
- Model training & development
- Model validation
- Model serving
- Model monitoring
- Model re-training

MLOps engineers focus on ML model deployment and management, while data engineers focus on data infrastructure and pipeline development.

5. AWS – Amazon Web Services

Amazon Web Services (AWS) is a cloud computing platform that offers a variety of services, including:- Servers, Storage, Networking, Remote computing, Email, Mobile development, Security.

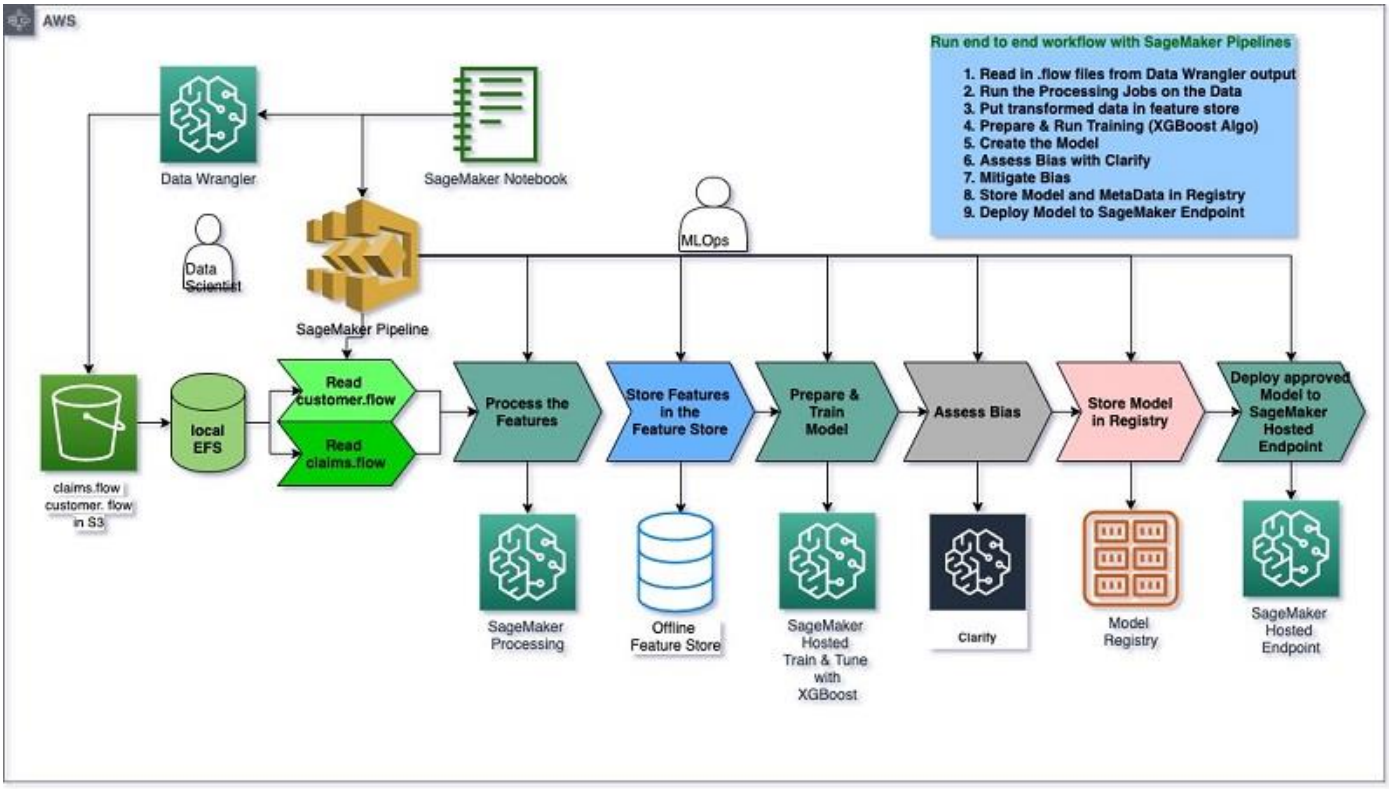
AWS is a subsidiary of Amazon that provides on-demand cloud computing platforms and APIs to individuals, companies, and governments, on a metered, pay-as-you-go basis.

Here are some of AWS's services:

- Amazon Simple Storage Service (S3): An open cloud-based storage service that provides scalable object storage for data backup, collection, and analytics.
- Amazon RDS: Handles database administration tasks like patching, backups, and scaling.
- AWS Lambda: A serverless, event-driven compute service that lets you run code for virtually any type of application or backend service without provisioning or managing servers.
- Amazon VPC: Enables you to set up a reasonably isolated section of the AWS Cloud where you can deploy AWS resources at scale in a virtual environment.
- Amazon elastic compute cloud: Provides virtual machines (VM) in the cloud where you can perform operations without the use of on-premises hardware like desktop or physical servers.

6. AWS Sagemaker

Amazon SageMaker is a machine learning platform that can help with machine learning operations (MLOps). MLOps is a set of procedures that machine learning practitioners use to speed up the deployment of machine learning models.



SageMaker can help with:

- Creating, training, and deploying machine learning models
- Providing features like Amazon SageMaker JumpStart
- Providing a large range of ways to develop models
- Helping teams with details and alerts from data to model anomalies
- Reporting bugs
- Automating and standardizing processes

Here are some steps for building an MLOps pipeline in SageMaker:

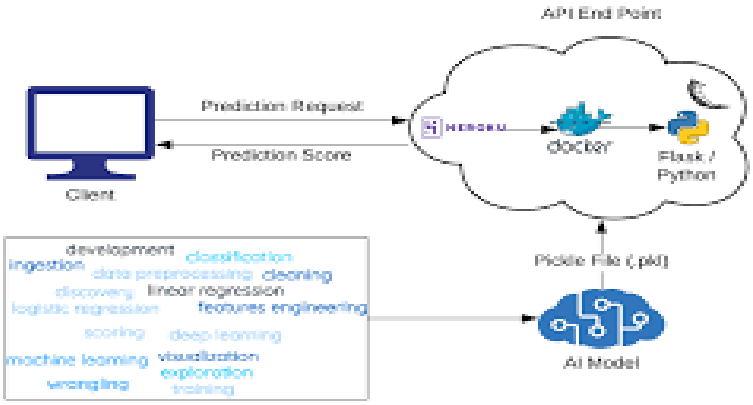
1. Create an Amazon SageMaker Notebook Instance
2. Create a Jupyter Notebook
3. Download, explore, and transform data
4. Train a model
5. Deploy the model
6. Evaluate the model
7. Clean up

To set up a project in Sagemaker Studio, you can:

1. Log in to your AWS account
2. Select Sagemaker from the list of services
3. Select Sagemaker Studio and use Quickstart to create Studio
4. Open Studio with the user you just created

7. Docker

Docker Cloud is a cloud-based platform that helps developers build, test, and deploy applications using Docker containers. It provides a set of tools and services that make it easy to manage and automate the Docker development lifecycle.



Docker Cloud includes the following features:

- Docker Hub: A public registry for storing and sharing Docker images.
- Docker Trusted Registry: A private registry for storing and sharing Docker images within an organization.
- Docker Build: A service for building Docker images.
- Docker Swarm: A service for orchestrating Docker containers.
- Docker Compose: A tool for defining and running multi-container Docker applications.
- Docker Cloud Build: A service for building Docker images in the cloud.
- Docker Cloud Deploy: A service for deploying Docker images to the cloud.

Docker Cloud is available as a free and paid service.

The free service includes Docker Hub, Docker Trusted Registry, and Docker Build.
The paid service includes Docker Swarm, Docker Compose, Docker Cloud Build, and Docker Cloud Deploy.

Docker Cloud can be used to **build**, **test**, and **deploy** applications on any cloud provider, including Amazon Web Services (AWS), Microsoft Azure, and Google Cloud Platform (GCP). It can also be used to deploy applications on-premises.

Docker Cloud is a popular choice for developers who want to use Docker to build and deploy applications. It provides a set of tools and services that make it easy to manage and automate the Docker development lifecycle.

8. Docker Cloud for ML Models Deployment

Docker is a containerization platform that allows developers to package their applications into containers. Containers are isolated, lightweight, and portable, which makes them ideal for deploying machine learning (ML) models.

There are several **benefits** to using Docker for ML model deployment:

1. **Reproducibility**: Docker containers can be used to create reproducible environments for ML models. This means that the same model can be deployed on different machines without any changes, which is important for ensuring consistency and accuracy.
2. **Portability**: Docker containers are portable, which means that they can be deployed on any machine that has the Docker platform installed. This makes it easy to deploy ML models to different environments, such as cloud servers or on-premises machines.
3. **Scalability**: Docker containers can be scaled up or down easily, which makes it easy to deploy ML models to meet changing demand. This is important for ML models that are used in production environments, where the demand for predictions can vary significantly over time.

To deploy an ML model using Docker, you will need to create a Dockerfile. A **Dockerfile** is a text file that contains instructions for building a Docker image. The Docker image will contain all of the dependencies that your ML model needs to run, such as the model itself, the Python runtime, and any other necessary libraries.

Docker is a powerful tool for deploying ML models. It provides a number of benefits, such as reproducibility, portability, and scalability. By using Docker, you can easily deploy your ML models to different environments and meet changing demand.

9. Sagemaker v/s Docker Deployment for ML Models

Amazon SageMaker and Docker are both popular tools for machine learning model deployment. However, they have different strengths and weaknesses, making them suitable for different use cases. Here is a comparison of the two tools:-

Factors	Amazon SageMaker	Docker Cloud
Advantages	Easy to use: Amazon SageMaker provides a managed environment for machine learning model deployment, making it easy to get started. Scalable: Amazon SageMaker can scale to meet the needs of even the most demanding machine learning applications. Secure: Amazon SageMaker provides a secure environment for machine learning model deployment, protecting your data and models from unauthorized access.	Flexible: Docker provides a high degree of flexibility for machine learning model deployment. You can customize the deployment process to meet your specific needs. Portable: Docker containers can be deployed on any platform that supports Docker, making it easy to move your machine learning models between different environments. Cost-effective: Docker can be a cost-effective option for machine learning model deployment, especially for small-scale applications.
Disadvantages	Costly: Amazon SageMaker can be expensive, especially for large-scale machine learning applications. Limited flexibility: Amazon SageMaker provides a limited set of options for machine learning model deployment, making it difficult to customize the deployment process to meet your specific needs.	Complex: Docker can be complex to use, especially for those who are not familiar with containerization technology. Not scalable: Docker is not as scalable as Amazon SageMaker, making it difficult to use for large-scale machine learning applications.
Size & Complexity of Model	If you have a large or complex machine learning model, **Amazon SageMaker** may be a better choice because it provides a managed environment that can scale to meet your needs.	
Cost-Effectiveness	Amazon SageMaker can be **expensive**, especially for large-scale machine learning applications.	**Docker** can be a more cost-effective option, especially for small-scale applications.
Technical Expertise	**Amazon SageMaker** provides a managed environment that makes it easier to get started with	Docker can be complex to use, especially for those who are not

	machine learning model deployment.	familiar with containerization technology.
Security	**Amazon SageMaker** provides a secure environment for machine learning model deployment by default.	Docker does not provide a secure environment by default, so you need to take additional steps to secure your Docker containers.

In general, Amazon SageMaker is a good choice for organizations that need a managed, scalable, and secure environment for machine learning model deployment. Docker is a good choice for organizations that need a flexible and portable solution for machine learning model deployment.

Analogies To Understand Model Deployment In Cloud – SageMaker v/s Docker

Let's imagine you're a chef and you've just cooked up a delicious recipe for a magical cake. Now, you want to share this cake with your friends, but you have two options for how you're going to deliver it:- using a magic teleportation machine (AWS SageMaker) or putting it in a special, customizable delivery box (Docker).

1. **AWS SageMaker**: Imagine you have a magical teleportation machine in your kitchen. You just put your cake inside, press a few buttons, and voila! Your cake instantly appears wherever your friends are, ready to be enjoyed. AWS SageMaker is like that teleportation machine but for your machine learning models. You train your model, package it up, and with a few clicks, it's deployed and ready to use in the cloud, without you having to worry about all the logistics of setting up servers and infrastructure.

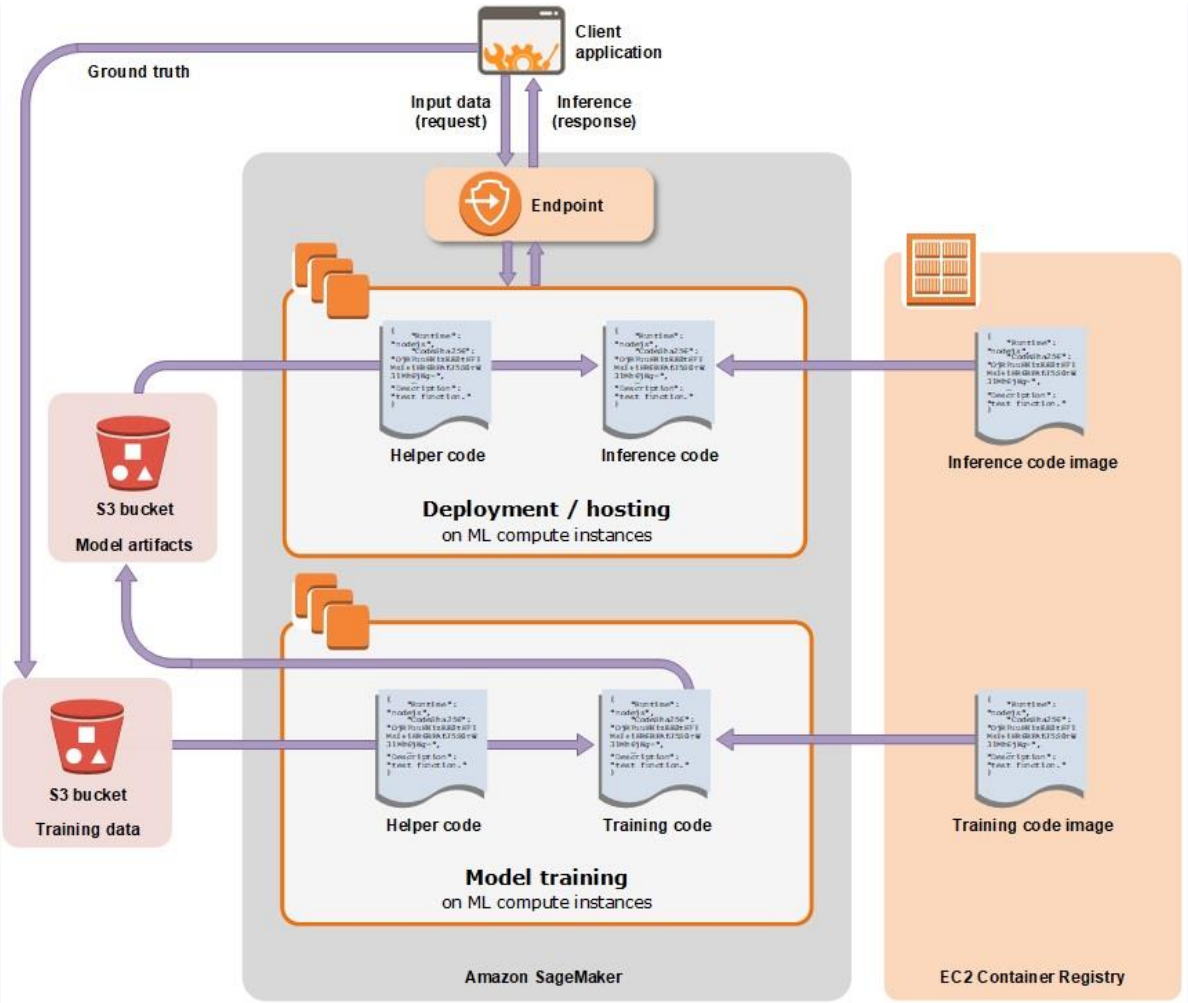
2. **Docker**: Now, let's say you have this special delivery box that you can customize however you want. You carefully place your cake inside, add some decorations, and seal it up. This box keeps your cake safe and ensures it arrives exactly as you intended, no matter where it's going. Docker works similarly for machine learning models. You package your model and all its dependencies into a Docker container, which is like that customizable delivery box. Then you can easily move this container around, whether it's to deploy it in the cloud, on your own servers, or even on your friend's computer. It's like having a portable, self-contained version of your model that you can transport anywhere.

So, whether you choose AWS SageMaker or Docker, you're basically picking the **best way** to deliver your magical cake (or machine learning model) to your friends (or users) in the most efficient and convenient way possible.

ML Model Deployment Process in SageMaker Using An Example

1. Creating a Notebook Instance
2. Loading Datasets, EDA & Train validation split
3. Dataset upload in S3 bucket
4. Training Process
5. Model Deployment and Endpoint creation

What is deploying an ML model?



After having a robust ML model and data science team agreed to rely on it, we need to deploy it in a production environment. This could be a simple task by running the ML model on batch data and perhaps writing a creating a scheduled cron job for scheduling the process of running the model every while in the future. However, sometimes we might empower our software application with the created model to make predictions on streaming data. Therefore, we might call the ML model using simple RESTful APIs. By this time, we might need to retrain your model on the updated data to avoid overfitting. As a result, having updated versions of the ML model run on production. It is necessary to handle model versioning, smoothly transition from one model to the next, and even plan to roll back to the previous model in case any failures happen, and maybe run multiple versions of models in parallel to perform A/B or canary experiments.

How to deploy machine learning models into production?

Based on use case, we perform various deployment approaches. For instance, AWS SageMaker provides **real-time inference workloads** where we have real-time, interactive, low latency requirements. Further, we can use SageMaker batch transform to get predictions for an entire dataset. Also, there are more deployment types like; Serverless Inference, Asynchronous Inference, and SageMaker Edge Manager for edge deployment.

How to maintain a deployed model?

Monitoring a model is the fundamental principle to maintaining ML model. AWS SageMaker's real-time monitoring ensures the quality of machine learning models in production. We can set alerts when anomalies and data drift hit the model. AWS model monitor services allow us to take quick actions to maintain and upgrade the quality of the running model.

Introducing the use case for deploying a model

In this project, we use Amazon SageMaker to create a machine learning model that **forecasts flight delays for US domestic flights** on the US Department of Transportation flight data to train the model that **predicts flight delays**.

We will go through the process of preparing raw data for use with machine learning algorithms. Then we will use a built-in SageMaker algorithm to train a model using the prepared data. Finally, we will use SageMaker to host the trained model and learn how to make real-time predictions using the model.

1. Create

First, navigate to the AWS SageMaker (https://aws.amazon.com/sagemaker/) from the console.

From the left list, choose Notebook → Notebook instances.

Choose the default configurations and hit Create notebook instance. The Sagemaker notebook environment is similar to a Jupyter notebook. Therefore, it should be easy to follow along.

2. Train the model to learn from the data

Once uploaded the US flight delay data to S3 bucket and then, load it with pandas to convert CSV data into a DataFrame (df) that is easy to manipulate. To do so, we need to provide pandas with data types (dtypes) for each feature.

Now that the training data is available on S3 in a format that SageMaker can use, we're ready to train the model. The Amazon SageMaker linear learner algorithm provides a solution for both classification and regression problems. It scales to large data sets and is sufficient for demonstrating the use of built-in SageMaker algorithms using Python.

SageMaker algorithms are available via container images. Each region that supports SageMaker has its own copy of the images. We will begin by **retrieving the URI** of the container image.

In [12]:
```
from sagemaker import image_uris

container = image_uris.retrieve('linear-learner', boto3.Session().region_name)
container
```

Out[12]:
```
'174872318107.dkr.ecr.us-west-2.amazonaws.com/linear-learner:1'
```

From the image URI, we can see the images are stored in **Elastic Container Registry** (ECR). We can also make our own images to use with SageMaker and store them in ECR.

In [13]:
```
import sagemaker

# Store the output in the S3 bucket
s3_output = 's3://{}/{}/output'.format(bucket, prefix)

# Retrieve the notebooks role and use it for model training (has access to S3, SageMaker)
role = sagemaker.get_execution_role()

session = sagemaker.Session()

# Use the Estimator interface to configure a training job
linear = sagemaker.estimator.Estimator(container,
                                       role,
                                       instance_count=1,
                                       instance_type='ml.m4.xlarge',
                                       output_path=s3_output,
                                       sagemaker_session=session,
                                       input_mode='Pipe' # stream training data from S3
                                       )
```

NOTE: Pipe mode is used for faster training and allows data to be streamed directly from S3 as opposed to File mode, which downloads the entire data files before any training.

We must now configure the hyperparameters for the linear-learner algorithm. We'll only configure the required **hyperparameters**.

In [14]:
```
linear.set_hyperparameters(feature_dim=len(train.columns) - 1, # minus 1 for the target
                           predictor_type='regressor')
```

The training job can now **begin training the model**. We can monitor the progress of the training job as it is output below the following cell. Blue logs represent logs coming from the training container and summarize the progress of the training process.

In [15]:
```
%%time

import time

# Set the content_type to text/csv (default is application/x-recordio-protobuf)
train_channel = sagemaker.session.TrainingInput(s3_train_data, content_type='text/csv')
test_channel = sagemaker.session.TrainingInput(s3_test_data, content_type='text/csv')

name = f"flight-delays-{int(time.time())}"

linear.fit({'train': train_channel, 'test': test_channel}, job_name=name) # validation channel is also supported
```

```
2022-05-21 13:17:30 Starting - Starting the training job...
2022-05-21 13:17:56 Starting - Preparing the instances for trainingProfilerReport-1653139050: InProgress
.........
2022-05-21 13:19:16 Downloading - Downloading input data...
2022-05-21 13:19:53 Training - Downloading the training image.....Docker entrypoint called with argument(s): train
Running default environment configuration script

2022-05-21 13:20:54 Training - Training image download completed. Training in progress.[05/21/2022 13:20:46 INFO 139
721559922496] Reading default configuration from /opt/amazon/lib/python3.7/site-packages/algorithm/resources/default
-input.json: {'mini_batch_size': '1000', 'epochs': '15', 'feature_dim': 'auto', 'use_bias': 'true', 'binary_classifi
er_model_selection_criteria': 'accuracy', 'f_beta': '1.0', 'target_recall': '0.8', 'target_precision': '0.8', 'num_m
odels': 'auto', 'num_calibration_samples': '10000000', 'init_method': 'uniform', 'init_scale': '0.07', 'init_sigma':
'0.01', 'init_bias': '0.0', 'optimizer': 'auto', 'loss': 'auto', 'margin': '1.0', 'quantile': '0.5', 'loss_insensiti
vity': '0.01', 'huber_delta': '1.0', 'num_classes': '1', 'accuracy_top_k': '3', 'wd': 'auto', 'l1': 'auto', 'momentu
m': 'auto', 'learning_rate': 'auto', 'beta_1': 'auto', 'beta_2': 'auto', 'bias_lr_mult': 'auto', 'bias_wd_mult': 'au
to', 'use_lr_scheduler': 'true', 'lr_scheduler_step': 'auto', 'lr_scheduler_factor': 'auto', 'lr_scheduler_minimum_l
r': 'auto', 'positive_example_weight_mult': '1.0', 'balance_multiclass_weights': 'false', 'normalize_data': 'true',
'normalize_label': 'auto', 'unbias_data': 'auto', 'unbias_label': 'auto', 'num_point_for_scaler': '10000', '_kvstore
': 'auto', '_num_gpus': 'auto', '_num_kv_servers': 'auto', '_log_level': 'info', '_tuning_objective_metric': '', 'ea
```

3. Deploy the model

SageMaker can host models through its hosting services. The model is accessible to clients through a SageMaker endpoint. The hosted model, endpoint configuration resource, and endpoint are all created with a single function called **deploy()**.

SageMaker Studio Notebooks are one-click Jupyter notebooks that can be spun up quickly. The underlying compute resources are fully elastic, and the notebooks can be easily shared with others, enabling seamless collaboration.

In [16]:
```
%%time

linear_predictor = linear.deploy(initial_instance_count=1,
                                 instance_type='ml.t3.medium')
```

```
-------!CPU times: user 134 ms, sys: 3.34 ms, total: 138 ms
Wall time: 3min 31s
```

The Endpoint is accessible over HTTPS, but when using the SageMaker Python SDK, we can use the predict function to abstract the complexity of HTTPS requests. We first configure how the predictor is to serialize requests and deserialize responses. We'll use CSV to **serialize requests** and JSON to **deserialize responses**.

In [17]:
```
from sagemaker.serializers import CSVSerializer
from sagemaker.deserializers import JSONDeserializer

linear_predictor.serializer = CSVSerializer()
linear_predictor.deserializer = JSONDeserializer()
```

Then, use the model endpoint to make **predictions** using a sample vector from the test data.

In [18]:
```
test_vector = test.iloc[0][1:] # use all but the first column (first column is actual delay)
actual_delay = test.iloc[0][0]

response = linear_predictor.predict(test_vector)

predicted_delay = response['predictions'][0]['score']
print(f'Predicted {predicted_delay}, actual {actual_delay}, error {actual_delay-predicted_delay}')
```

```
Predicted -12.270792007446289, actual -20.0, error -7.729207992553711
```

Conclusion

We used Python to prepare data, use SageMaker to train and deploy a model on the AWS cloud, and make predictions with a model hosted in SageMaker.

ML Model Deployment Process In Docker Using An Example

Steps to deploy a machine learning model with Flask and Docker:-

We will use **Docker as a container** to deploy the **Flask app**. A container is similar to a virtual machine except that it does not have its own resources. Rather, it shares them with the host machine. This helps to deploy the app easily by installing all the dependencies.

A self-contained and efficient software package called a **Docker container image** encompasses all the necessary components to execute an application, including code, runtime, system tools, system libraries, and configurations.

1. Create a machine learning model

In this step, we will create a ML model that will be used for deployment.

```
# Data Manipulation libraries

import pandas as pd
from sklearn.model_selection  import train_test_split
from sklearn.neural_network import MLPRegressor
import joblib
```

```
df = pd.read_csv('house_price.csv')  # Load the dataset
```

```

```

```
x_train = df[['location', 'bedroom_count', 'locality', 'carpet_area']]

y_train = df[['price']]
```

```

```

```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()

scaler.fit(x_train)
```

```

```

```
x_train_scaled = scaler.transform(x_train)

x_train_scaled = pd.DataFrame(x_train_scaled, columns=x_train.columns)

X_train, X_test, Y_train, Y_test = train_test_split(df_x_scaled, df_y, test_size = 0.33, random_state = 5)
```

```

```

```
mlp = MLPRegressor(hidden_layer_sizes=(60), max_iter=1000)
```

```
mlp.fit(X_train, Y_train)

Y_predict = mlp.predict(X_test)
```

```
#Saving the machine learning model to a file
joblib.dump(mlp, "model/price_model.pkl")
```

We load the dataset named "**house_price.csv**" and perform basic preprocessing steps like scaling and splitting the dataset into training and testing sets. We then pass the training set into the MLP regressor for training. Finally, we save the trained model using the joblib library in pickle format.

2. **Create a REST API with Flask framework**

Now, we will create a Flask API that will be called later for **inferencing**.

```
#importing necessary libraries

from flask import Flask, jsonify, request
import pandas as pd
import joblib
```

```
app = Flask(name)
```

```
@app.route("/predict", methods=['POST'])

def do_prediction():

json = request.get_json()

#loading saved model here in this python file

model = joblib.load('model/rf_model.pkl')

#creating data frame of JSON data

df = pd.DataFrame(json, index=[0])
```

```
from sklearn.preprocessing import StandardScaler
#performing preprocessing steps
scaler = StandardScaler()
scaler.fit(df)

x_scaled = scaler.transform(df)

x_scaled = pd.DataFrame(x_scaled, columns=df.columns)
y_predict = model.predict(x_scaled)
```

```
res= {&amp;quot;Predicted Price of House&amp;quot; : y_predict[0]}
return jsonify(res)
```

```
if name == "main":
app.run(host='0.0.0.0')
```

Next, we will create a requirements.txt file such that all the **dependencies** can be installed in a single command. Below is the code for the same:

```
pip freeze> requirements.txt
```

Once you have created the Flask API, the code above has to be pushed to the GitHub repository so that it can be cloned when we **dockerize** the entire code.

3. Create a Docker Image

After creating the service, the initial step will involve defining the Docker image, which is critical for the process. Defining the Dockerfile contents is essential to include all the necessary dependencies and copy the application's content into the container.

Below is the content to be included in the Docker file:

```
#Using the base image with Python 3.10
FROM python:3.10
```

```
#Set our working directory as app

WORKDIR /app

#Installing Python packages through requirements.txt file

RUN pip install -r requirements.txt
```

```
# Copy the model's directory and server.py files
```

```
ADD ./models ./models

ADD server.py server.py
```

```
#Exposing port 5000 from the container
EXPOSE 5000
#Starting the Python application
CMD ["gunicorn", "--bind", "0.0.0.0:5000", "server:app"]
```

4. Run Docker

Here, we will build a Docker container for the service. Note that it is necessary to install Docker before this step is executed.

```
#Cloning the repo from Github in your local
git clone https://github.com/harsha89/ml-model-tutorial.git

#Building the docker image
docker build -t ml-model
```

Once we achieve this, we can push this code to the Docker repository which can be used to run the application from anywhere.

Conclusion

We've discussed why ML models need to be deployed to production and how to do so using Docker and Flask. Without deployment, trained models cannot be used for inference for real-time data. To deploy any service to production, two key factors are important, i.e., scalability and portability. Microservices architecture helps introduce these two factors and allows each service to run in a container, such as Docker, as an independent service.