Fundamentals of AWS Cloud (2)

What is IAM

AWS Identity and Access Management (IAM)

IAM is a web service that helps you manage access to your AWS resources.

It allows you to create and manage users, groups, and permissions, and to grant and revoke access to AWS resources.

With IAM, you can create AWS users and groups, and assign permissions to them. You can also create policies, which are documents that define permissions for AWS resources.

IAM and SageMaker

1

When you create an Amazon SageMaker notebook instance, you can specify an IAM role that determines what actions **the notebook instance** can perform and what resources it has access to.

2

When you train a machine learning model using SageMaker, you can specify an IAM role that determines what actions the **training job instance** can perform and what resources it has access to.

3

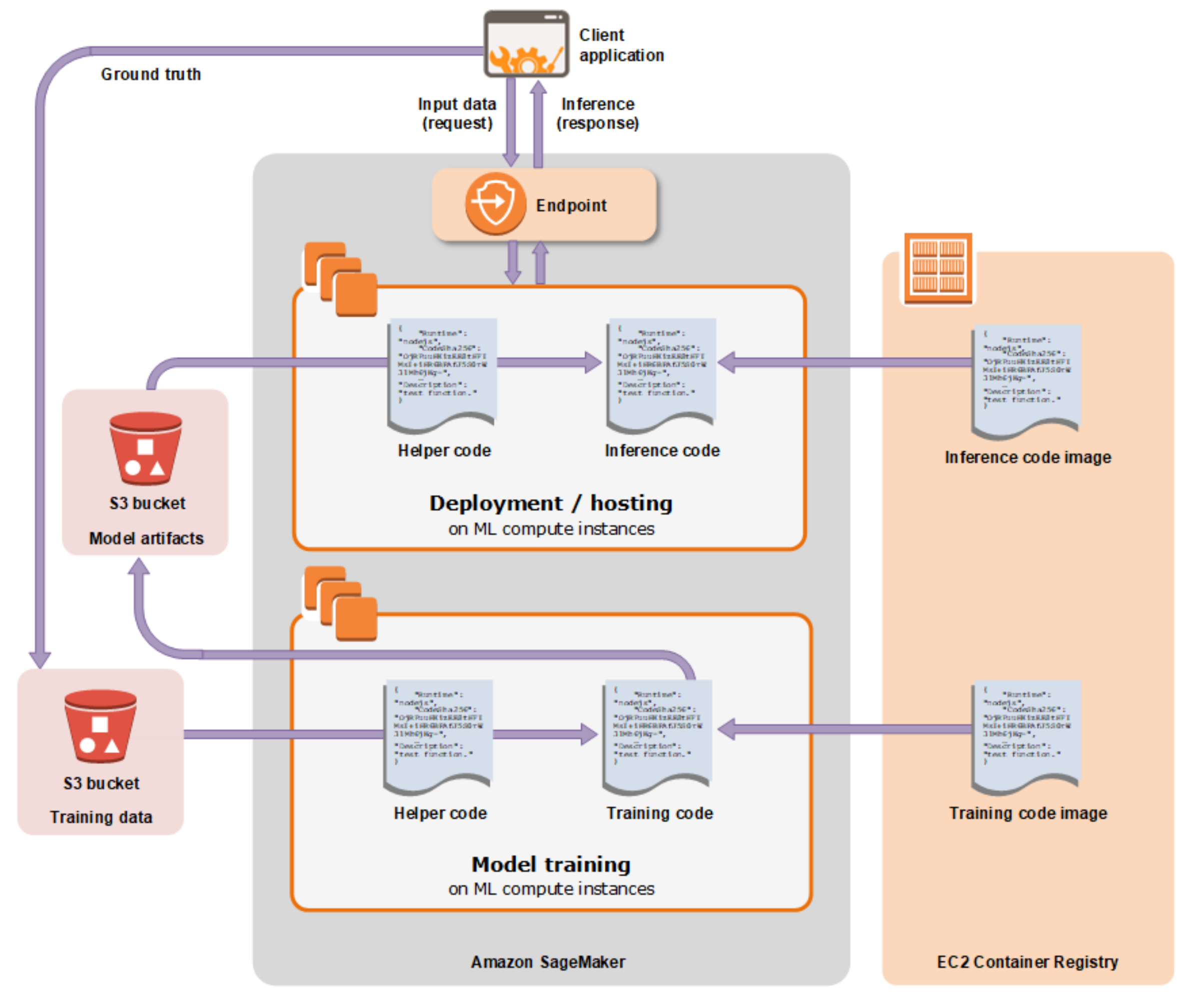
When you deploy a trained model to production using SageMaker, you can specify an IAM role that determines what actions the instance that is hosting the model can perform and what resources it has access to.

Demo IAM

Show	Show LabRole
Show	Show policies
Show	Show an instance role and what that means (related to cloud9)

Containers and SageMaker

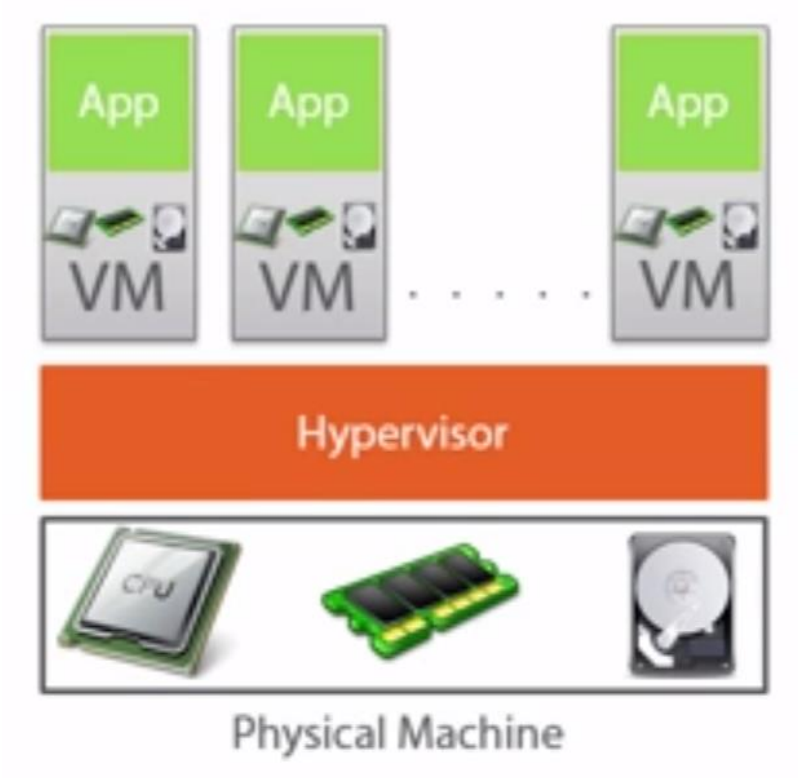
https://docs.aws.amazon.com/sagemaker/latest/dg/how-it-works-training.html



Containers in SageMaker and ECR

What problem virtualization did not solve

- Each VM needs a new OS
- More OS means more resources to be allocated
- More OS means more patching and higher operation cost
- More OS means more things to do fail if something is wrong

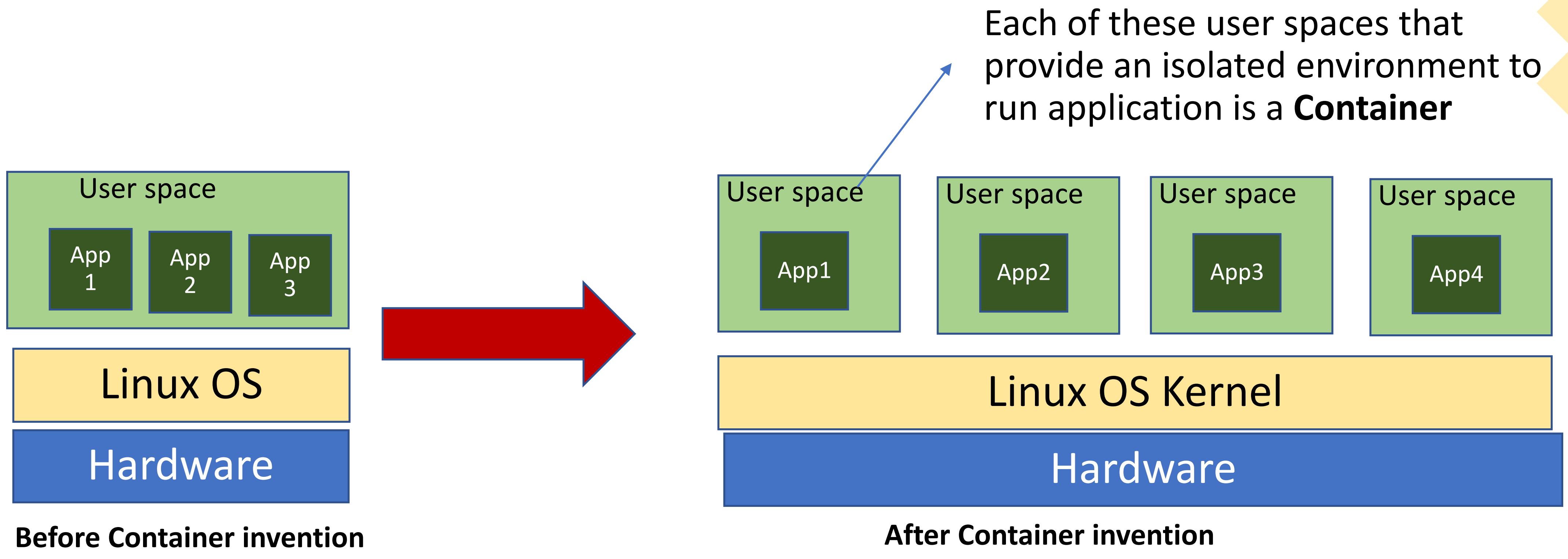


What is a container

- Container is an environment like VM that applications run in it
- But containers are much lighter than VMs
- So each container consume much less CPU, RAM and disk compare to their equivalent VM OS

App
Container

Before and after Containers


Each of these user spaces that provide an isolated environment to run application is a **Container**
User space
App 1
App 2
App 3
Linux OS
Hardware
Before Container invention
User space
App1
User space
App2
User space
App3
User space
App4
Linux OS Kernel
Hardware
After Container invention

Containers footprints

Each Container has a very small footprint

They are not a full OS

Containers share many of files and libraries with underlying OS

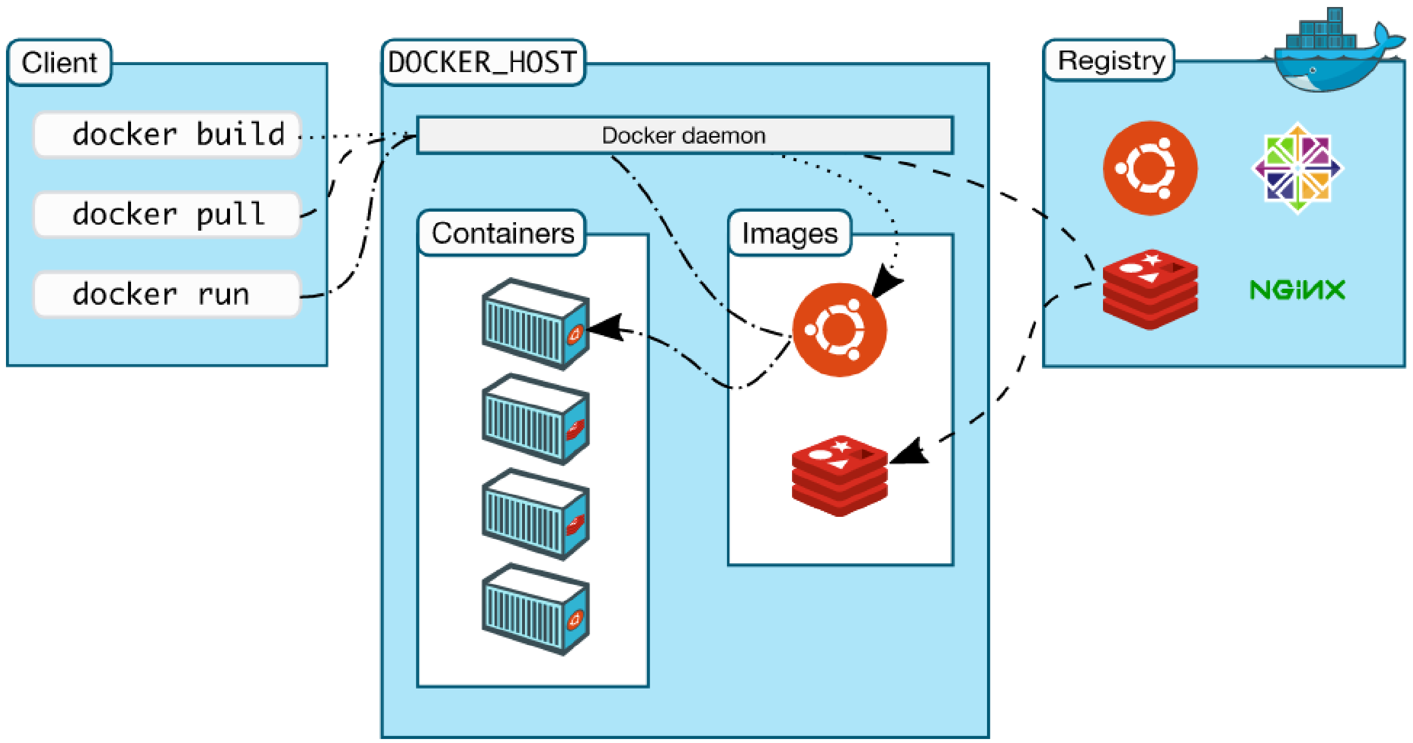
Containers and SageMaker

SageMaker uses Docker containers in the backend to manage training and inference processes.

While you don't need to use Docker containers explicitly with SageMaker for most use cases, you can use Docker containers to extend and customize SageMaker functionality.

Docker fundamentals

Docker architecture (https://docs.docker.com/get-started/overview/)



Containers in SageMaker

- SageMaker heavily relies on **containers**.
- In the context of Amazon SageMaker, a container is a lightweight, stand-alone, and executable package that contains all the necessary code, libraries, dependencies, and runtime to run a machine learning (ML) model.
- Containers are used to **deploy prebuilt-models** in SageMaker and allow users to run their models in a consistent and reliable environment, regardless of the underlying infrastructure.
- In SageMaker, users can build their own containers or use pre-built containers provided by Amazon.
- Pre-built containers are available for **popular ML algorithms** (as of now, 20+ algorithms) , **ML frameworks** such as TensorFlow, PyTorch, and MXNet, as well as for common data processing and visualization tools like **scikit-learn and Spark**.
- See this link: https://docs.aws.amazon.com/sagemaker/latest/dg/algos.html

Procuring Algorithms in SageMaker

Docker registry for the **Built-in Algorithms** https://docs.aws.amazon.com/sagemaker/latest/dg/sagemaker-algo-docker-registry-paths.html)

Docker Registry for **Built-in frameworks** (see the above link)

Bring your own script and containerize it and push it to ECR

Bring your own container and push it to ECR

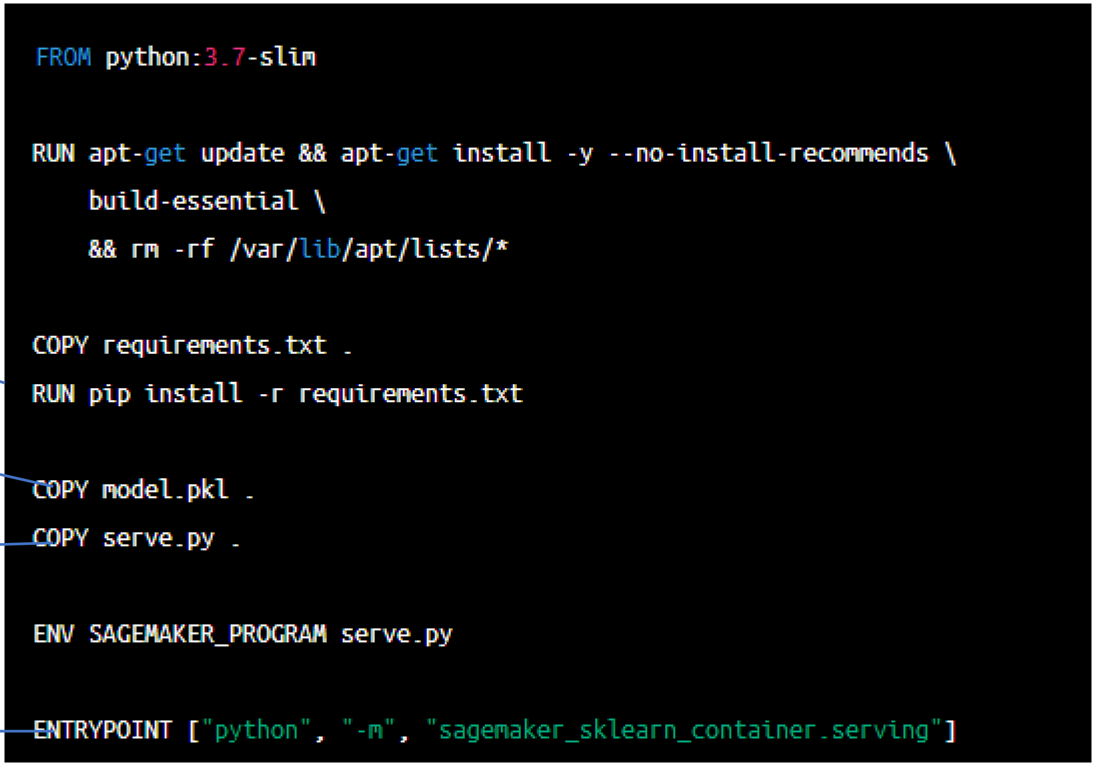
ECR

- **Amazon Elastic Container Registry (ECR)** : If you are familiar with DockerHub, this is the same but, in your account,
- As it is mentioned, the built-in algorithms are stored in public ECR paths, and they are available to you
- You can create a repository in your account by running the following command in Cloud9:
- **aws ecr create-repository --repository-name my-repository**

Check the repo name inside your account

Dockerfile

- Docker Images are created from a Dockerfile. A Dockerfile can packages our model and create an image from that before pushing that into ECR. The skeleton of that would be something like the following picture:



```
FROM python:3.7-slim

RUN apt-get update && apt-get install -y --no-install-recommends \
    build-essential \
    && rm -rf /var/lib/apt/lists/*

COPY requirements.txt .
RUN pip install -r requirements.txt

COPY model.pkl .
COPY serve.py .

ENV SAGEMAKER_PROGRAM serve.py

ENTRYPOINT ["python", "-m", "sagemaker_sklearn_container.serving"]
```

Dependencies you need to run the model

Copying the model fie (**model.pkl**)

Copying the serving file (**serve.py**)

Whatever you put in front of ENTRYPOINT, it will run when the container starts

Building image and pushing to ECR

- Before you run an algorithm or a model in SageMaker you need to put it inside an image and push that image to ECR. Otherwise, SageMaker is not able to run it.
- For example, we need to run the following command to build an image from the Dockerfile in the previous slide

```
docker build -t my-image .
```

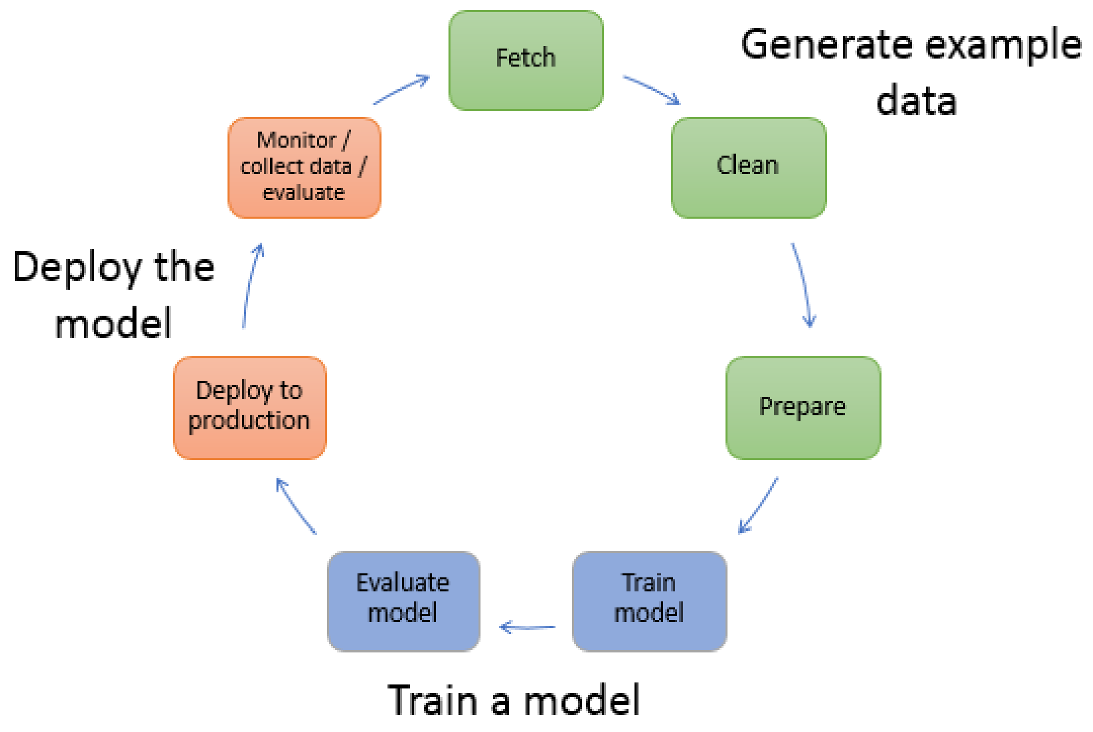
- Then we push that to ECR

```
docker push image_id
```

ML Pipeline and Computing Requirements

A series of steps or stages that are followed in order to build, deploy, and manage an ML model

ML Pipeline



What makes an ML pipeline?

- **Data preparation:** This step involves collecting and preprocessing the data that will be used to train the ML model. This may include tasks such as collecting data from various sources, cleaning and formatting the data, and splitting the data into training and validation sets.
- **Model training:** In this step, the ML model is trained using the prepared data. This typically involves selecting an ML algorithm, tuning its hyperparameters, and training the model using a training dataset.
- **Model evaluation**: After the model has been trained, it is evaluated using a validation dataset to determine how well it performs. This may involve calculating various performance metrics such as accuracy or AUC.
- **Model deployment**: If the model performs well on the validation dataset, it can be deployed for use in a production environment. This may involve creating an API or deploying the model to a serverless platform such as Amazon SageMaker.
- **Model monitoring and maintenance**: After the model has been deployed, it is important to monitor its performance and make any necessary updates or adjustments. This may involve retraining the model on new data or adjusting its hyperparameters to improve performance.

How the ML pipeline are streamlined?

01

You can use **a Jupyter notebook** to call the steps one after another

02

You can use a **built-in SageMaker pipeline** service and create all the steps and orchestrate the orders

03

You can use a service like **Step Function** to run the steps in order

04

You can use any **open source orchestration** tools like MLflow

Computing Resources in ML pipeline

- **Data preparation:** For data preparation tasks, you may want to use a smaller instance with sufficient CPU and memory to handle the data processing requirements. Examples of suitable instances for data preparation tasks include the ml.m5.large or ml.c5.xlarge instances.

- **Model training:** For model training, you will typically need more compute resources, such as a larger instance with **more CPU and memory**. You may also want to consider using a **GPU-powered instance** if your ML algorithm requires it. Examples of suitable instances for model training tasks include the ml.p3.2xlarge or ml.c5.18xlarge instances.

- **Model evaluation**: For model evaluation, you will typically need a computing instance with similar resources to the one used for model training.

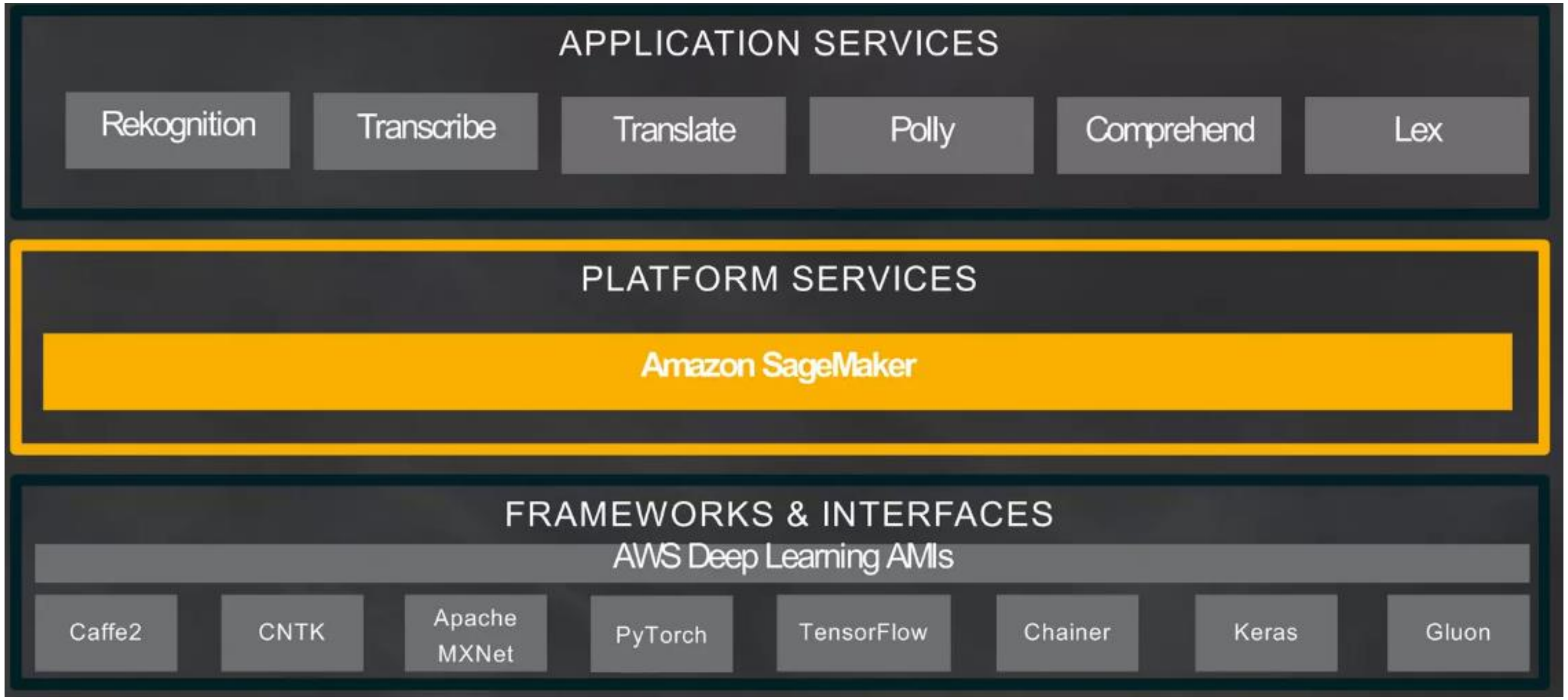
- **Model deployment:** For model deployment, you will need to select a computing instance that is suitable for serving the model and handling the expected workload. The recommended instance will depend on the specific requirements of your project, such as the number of concurrent requests and the size of the payloads.

SageMaker supported Instance types

- Depending on the pipeline stage, SageMaker provides different instance types:
- https://aws.amazon.com/sagemaker/pricing/ (see On-Demand Pricing table)

SageMaker Architecture

Amazon ML Stack



SageMaker Architecture

- Amazon SageMaker is a **set of tools and services** that allow data scientists and ML Engineers to build and train machine learning models.
- It includes a development environment for writing and testing ML code, a training environment for building and training models, and a deployment environment for deploying trained models to production.
- SageMaker has APIs to enable ML engineers to go through ML pipeline stages (more about this later).

SageMaker Features (Console Demo)

- It has services for different tasks in pipeline
- Notebook instance
- Training
- Processing
- Hosting
- Marketplace
- And many more features available through SageMaker Studio that we will learn later

SageMaker APIs

BOTO3 PYTHON SDK

SAGEMAKER PYTHON SDK

Example: SageMaker APIs using boto3

- **Create a training job:** To create a training job, you can use the CreateTrainingJob API. This API allows you to specify the location of your training data, the type of training algorithm you want to use, and other parameters such as the number of training instances and the type of instances to use.

- **Create a model:** To create a model using a trained model artifact, you can use the CreateModel API. This API allows you to specify the S3 location of the trained model artifact and the container image for the inference code.

- **Create a transform job**: To create a transform job, you can use the CreateTransformJob API. This API allows you to specify the location of your input data, the location to store the transformed data, and the type of transform algorithm to use.

- **Create an endpoint**: To create an endpoint for hosting a model, you can use the CreateEndpoint API. This API allows you to specify the name and configuration of the endpoint, as well as the model to deploy to the endpoint.

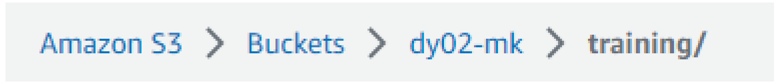
https://boto3.amazonaws.com/v1/documentation/api/latest/reference/services/sagemaker.html

Similar but easier to use API: **using SageMaker Python SDK**

- **create_training_job**: This API creates a training job on Amazon SageMaker to train a machine learning model.
- **create_model:** This API creates a model in Amazon SageMaker using a trained model artifact.
- **create_endpoint**: This API creates an endpoint in Amazon SageMaker that can be used to deploy a machine learning model and serve predictions.
- **create_transform_job**: This API creates a transform job on Amazon SageMaker to apply pre-processing or post-processing to a large dataset.
- **create_hyper_parameter_tuning_job:** This API creates a hyperparameter tuning job on Amazon SageMaker to find the best combination of hyperparameters for a machine learning model.
- **create_experiment:** This API creates an experiment in Amazon SageMaker to track and compare the results of multiple training jobs.
- **create_notebook_instance:** This API creates a Jupyter notebook instance in Amazon SageMaker that can be used for data exploration, prototyping, and model development.

Demo: Using SageMaker **CreateTrainingJob()** API

- In AWS Academy – Virginia
- Upload the **training_data.csv** to a folder in S3 called training



Amazon S3 > Buckets > dy02-mk > training/

- upload a training job script (**training.py**) to the cloud9 but before running the code, update the code and environment as shown in the next two slides.
- Go to SageMaker training in console and observe there is no training job

Changes in Cloud9

- Make sure the **AWS setting temporary credential is disabled**
- Assign **lab role** to the instance
- Run aws sts get-caller-identity
- Run aws configure and just set the region
- Go to next slide before starting the training job

Changes in the training.py

- Set the job name (line 7)
- Set the bucket and folder names (line 8 and 11)
- Set the role ARN (line 14)
- Make sure the URL of the algorithm is correct (use this path: https://docs.aws.amazon.com/sagemaker/latest/dg/sagemaker-algo-docker-registry-paths.html) (line 20)
- After applying the above changes, you should be able to run the job and see that in the SageMaker Training console

Mandatory Assignment

- Create a dummy lambda function with Python 3.9 runtime and LabRole assigned to it. After creation click on test and select Configure test event, from the **template-optional**, select S3Put. It shows you the JSON event that is passed from S3 to Lambda service when a put event happens on S3.

- Review the JSON file and as you see, there are entries like **bucket name** and **object key**. The assignment is you write a Lambda function in Python that **prints** the bucket name and object name when you upload an object in S3 bucket.

- In S3, learn how to create an event to call a lambda function

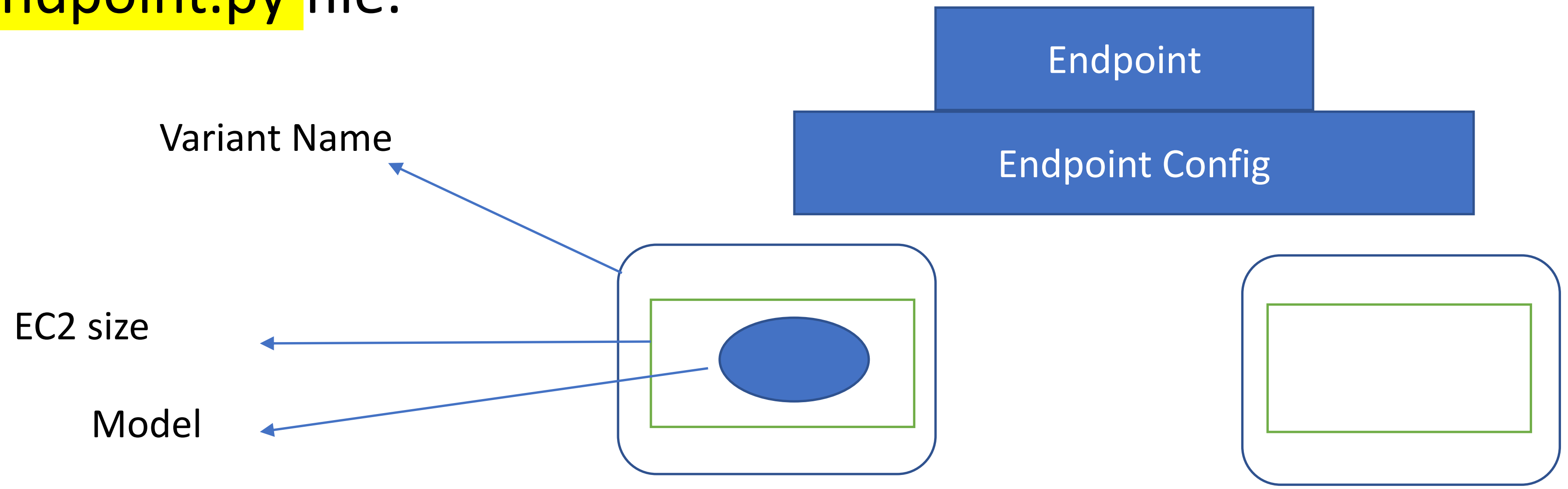
- The Lambda function you will write may have some code snippets like what I have shown you in the next slide (those are just some parts of the final code you will write, so it needs modifications)

Sample code

```
def lambda_handler(event, context):
    ……
    # Get the S3 object key (object name)
    object_key = event['Records'][0]['s3']['object']['key’]
    # Print the object key
    print(object_key)
```

Assignment option (1): create a model

- Use the **model.tar.gz** model artifact file that has been created by the training job earlier and create a SageMaker model.
- I have provided a skeleton code to use as a starting point, but that code will not run as-is, you need to modify it.
- The file name is model.py
- (**Optional**)I have also provided another skeleton to deploy the code by create_endpoint.py file.



Assignment option (2)

- Provision a Cloud9 instance
- Create 3 HTML files that they link to each other
- Creating an image locally by nginx
- Push the image to ECR