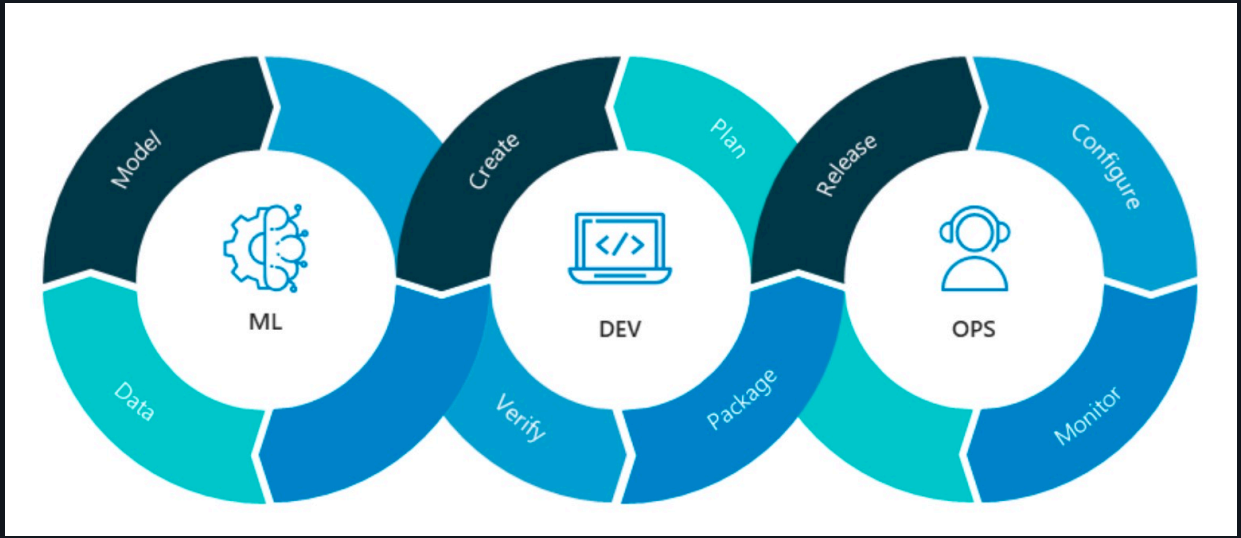
MLOps
Model Deployment & Infrastructure

Sergey Pugachev

Solutions Architect, AWS

What is MLOps?

MLOps, a close relative to DevOps, is a combination of philosophies and practices designed to enable data science and IT teams to rapidly develop, deploy, maintain, and scale out Machine Learning models.



https://blogs.nvidia.com/blog/2020/09/03/what-is-mlops/

AI/ML only brings value **in production**

Pilot/POC

Unclean

PoC

Experimental

Manual

Production

Reproduceable

Reliable

Auditable

Scalable

Consistent

Agenda

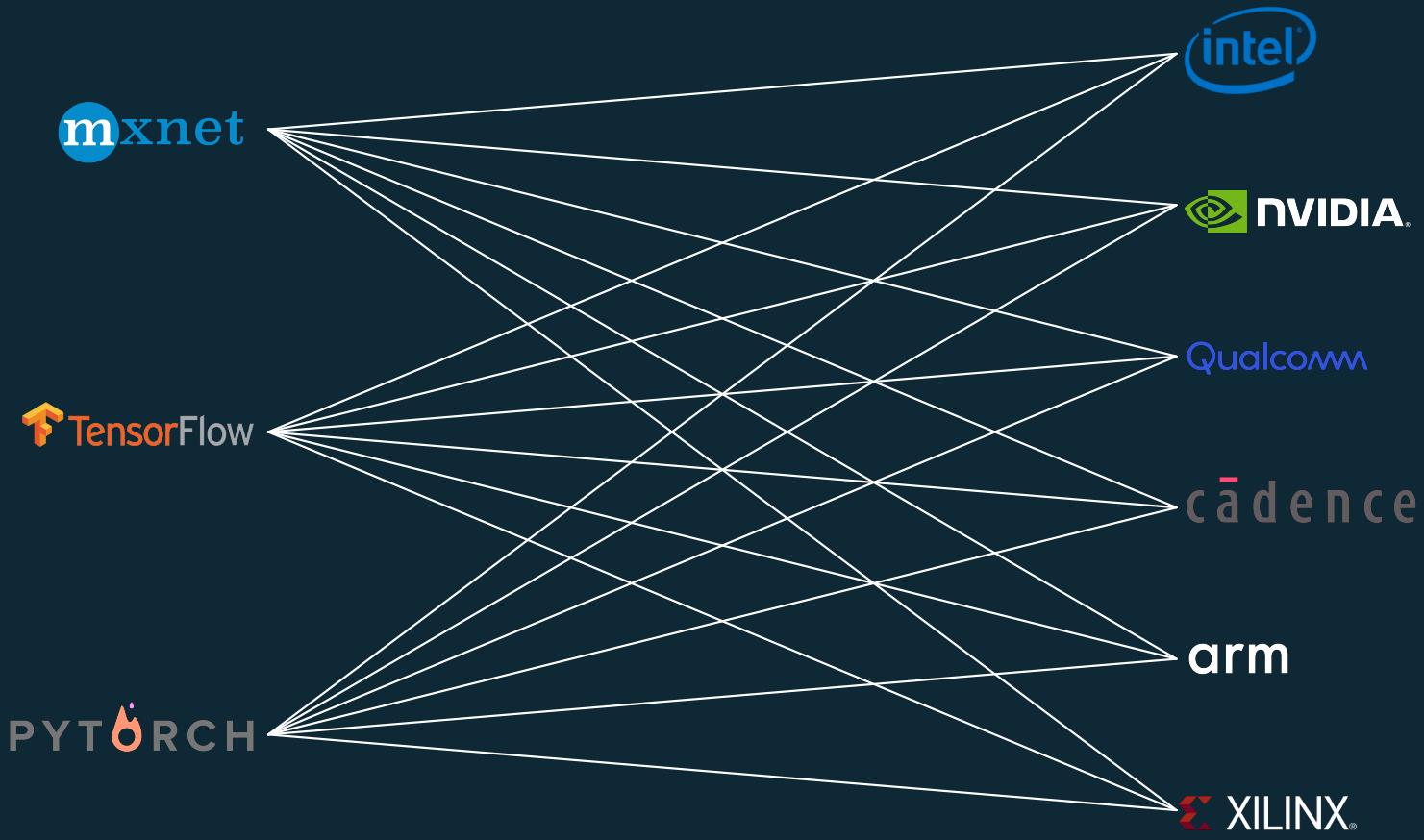
1. What to run?
2. Where to run?
3. How to run?

aws machine learning

1. What?

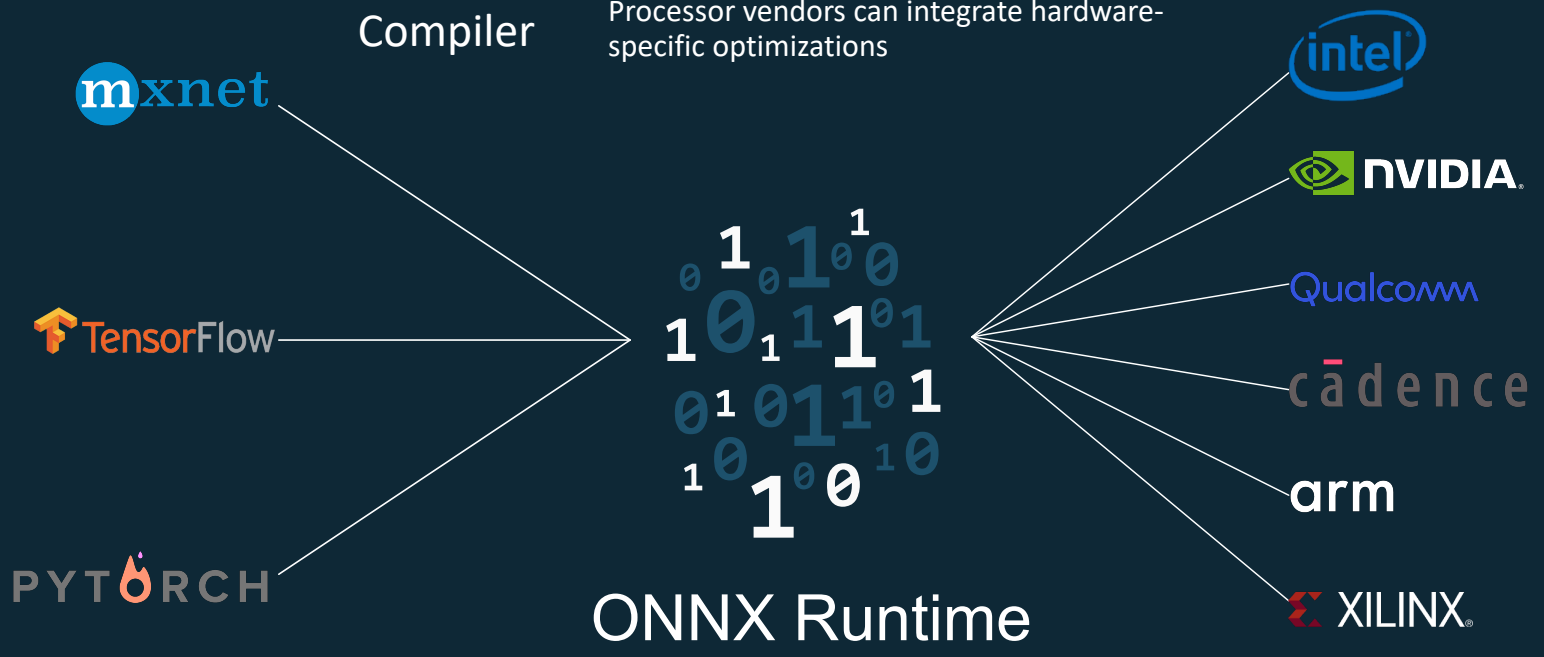

Save to file: pickle and joblib

Problem: Framework optimization is complicated



aws

ONNX runtime: Train once, run anywhere



https://onnx.ai/supported-tools.html

aws

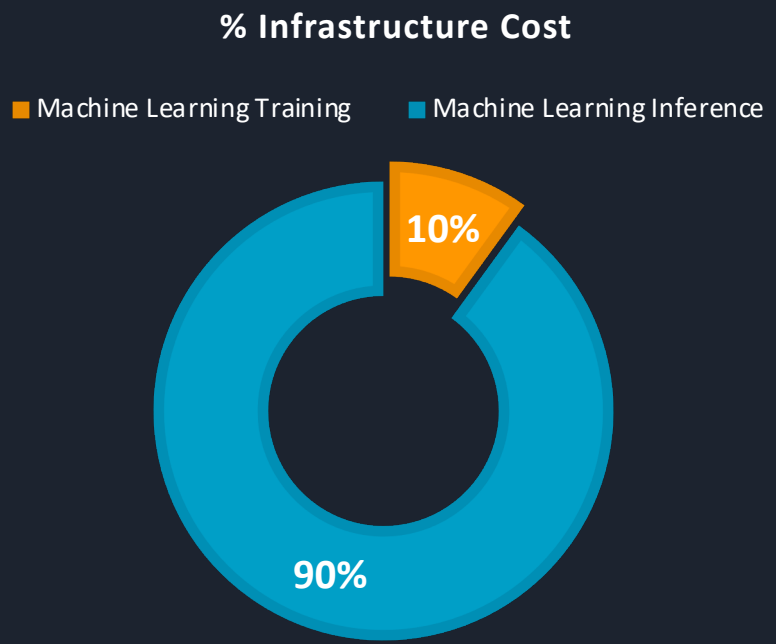
aws machine learning

2. Where?

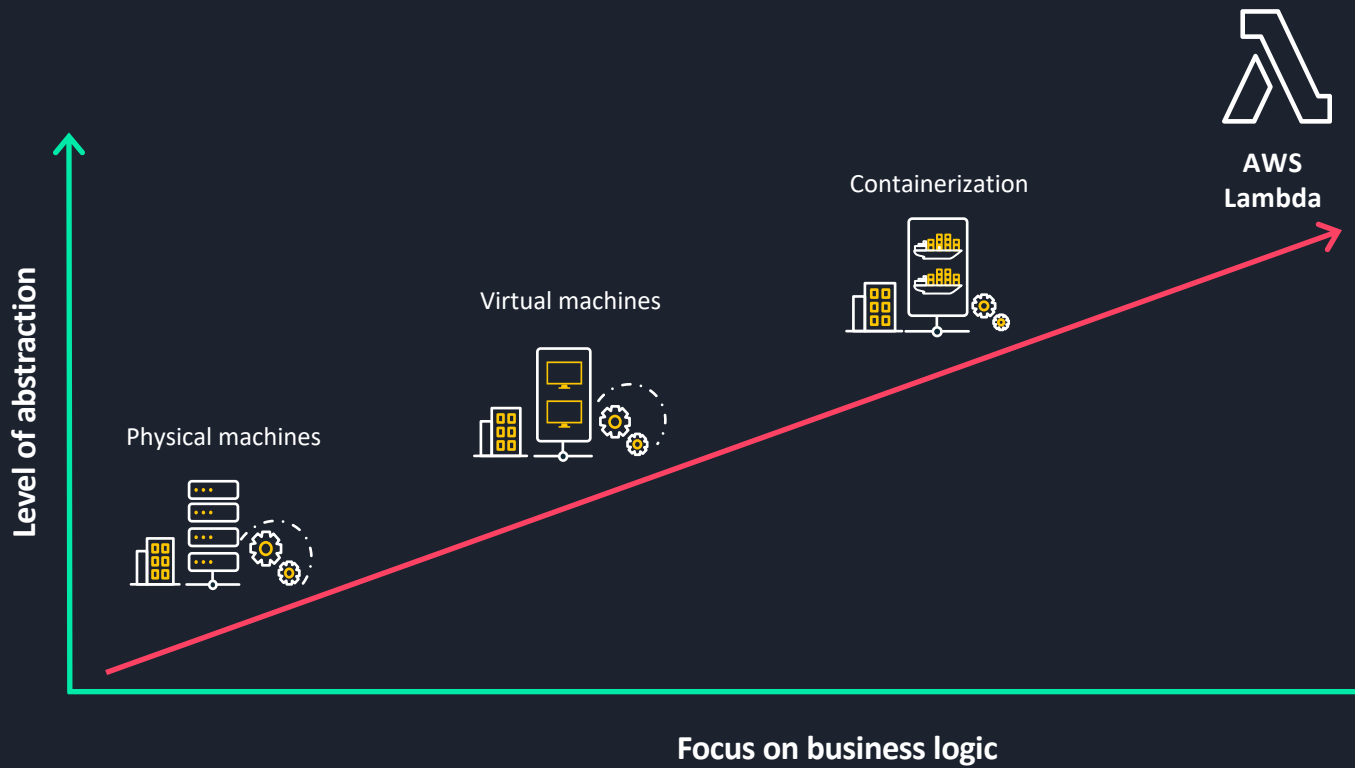
Machine Learning Inference Dominates ML Infra. Costs

1. Cost to run ML models in production dominates the overall infrastructure costs to develop and run ML applications

2. ML practitioners are creating increasing complex models that are prohibitively more expensive to run in production



We are witnessing a paradigm shift



Serverless

- Continuous scaling
- Fault tolerance built in
- Pay for value
- Zero maintenance
- Focus on business value

Serverless means…

No servers to provision or manage

Scales with usage

Never pay for idle

Availability and fault tolerance built in

Cloud Services

Compute	Virtual Machine	EC2	Elastic Beanstalk		Fargate	AWS Lambda
Databases	MySQL	MySQL on EC2	RDS MySQL	RDS Aurora	Aurora Serverless	DynamoDB
Storage	Storage					S3
Messaging	ESBs		Amazon MQ	Kinesis		SQS / SNS
Analytics	Hadoop	Hadoop on EC2	EMR	Elasticsearch Service		Athena

aws

Optimizing ML performance

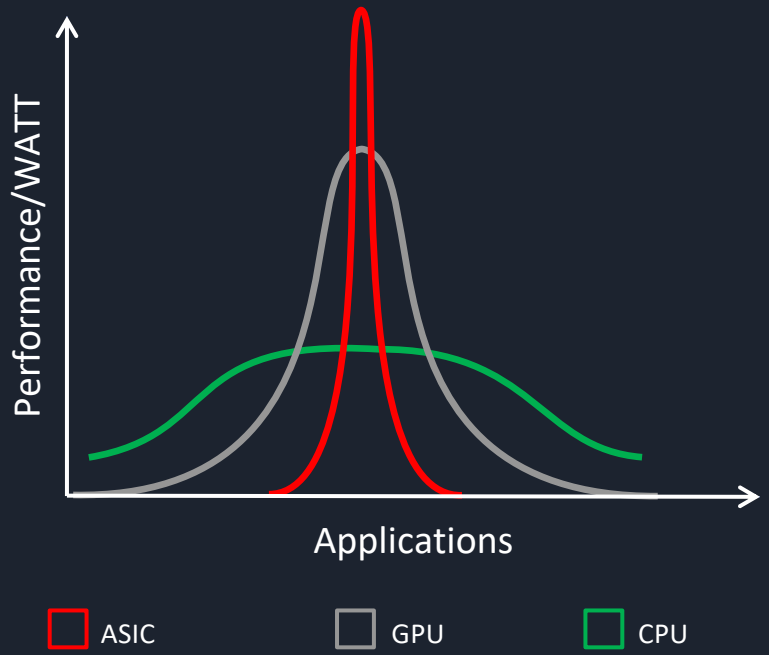
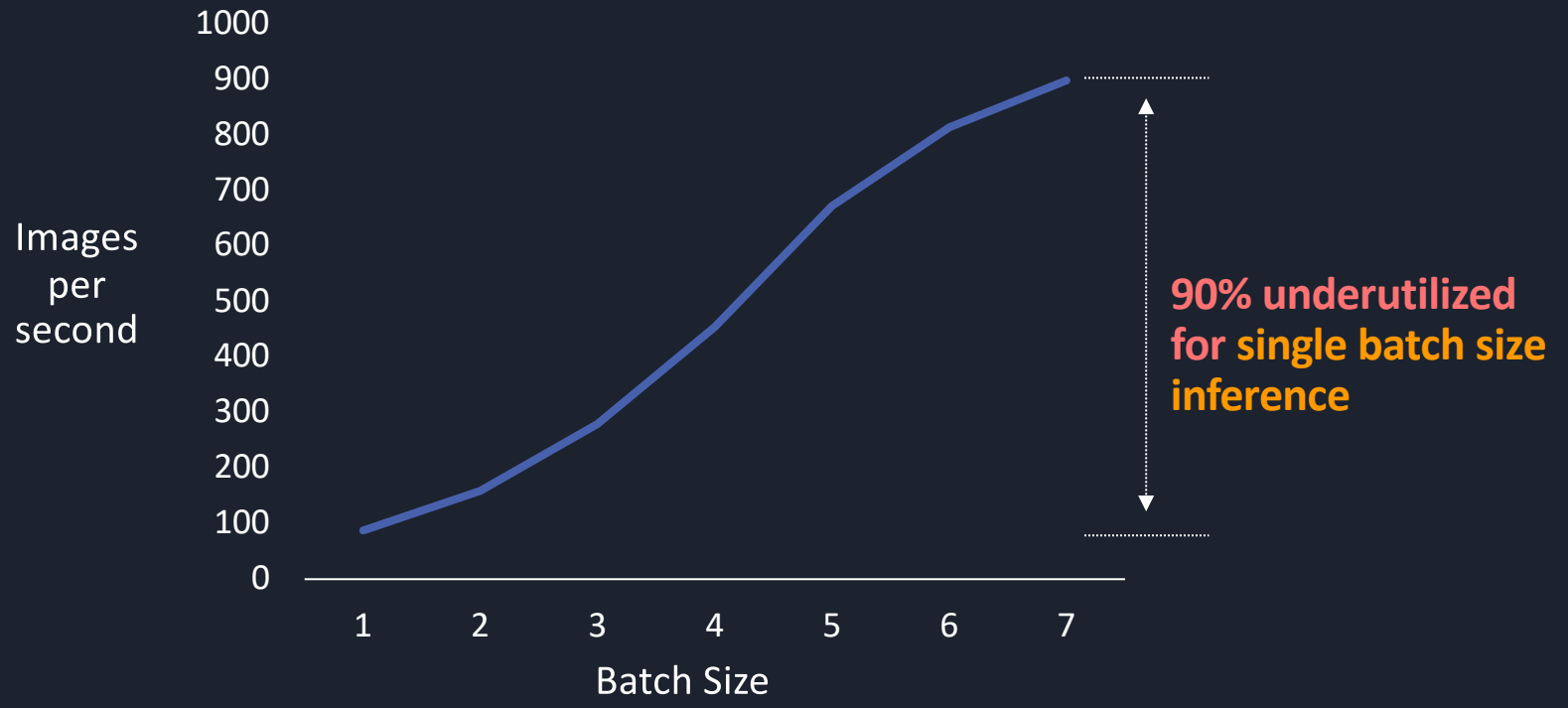


Chart for illustrative purposes

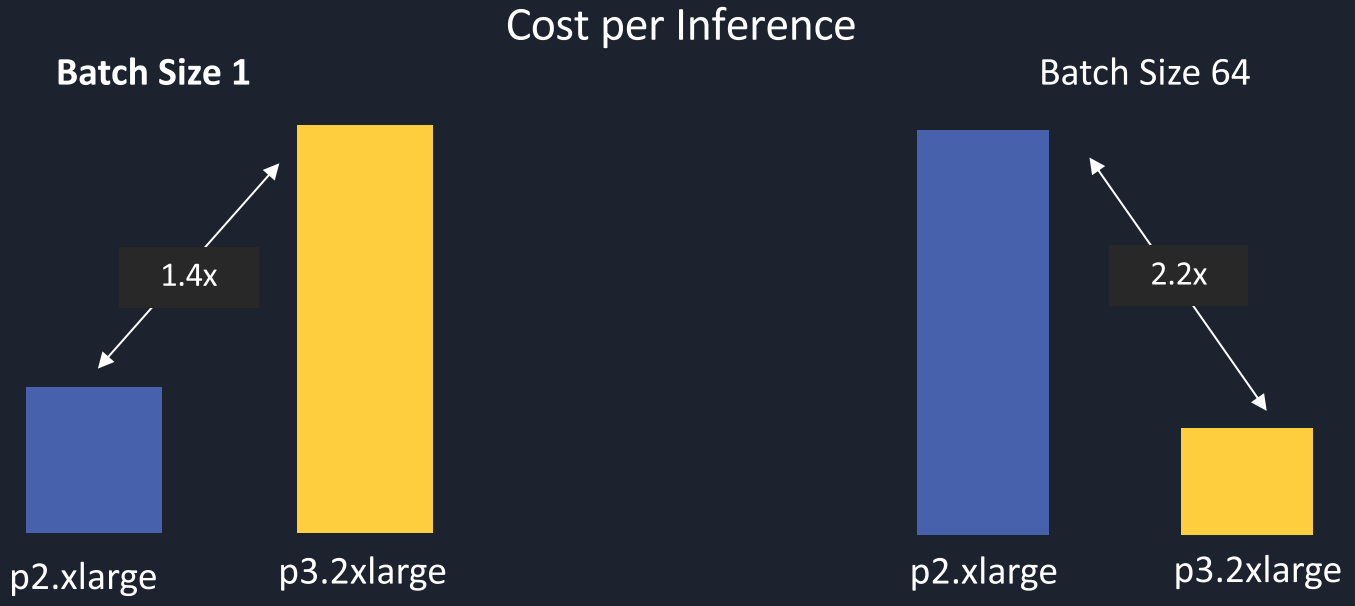
GPU utilization for inference: latency and throughput



Inception-v3 on a p3.2xlarge instance (using a V100 GPU)

How cost effective are GPU instances for inference?

Smaller P2 instances are more effective for real time inference with small batch sizes



Lower numbers are better

Inception-v3 FP32 MXNet

Processors

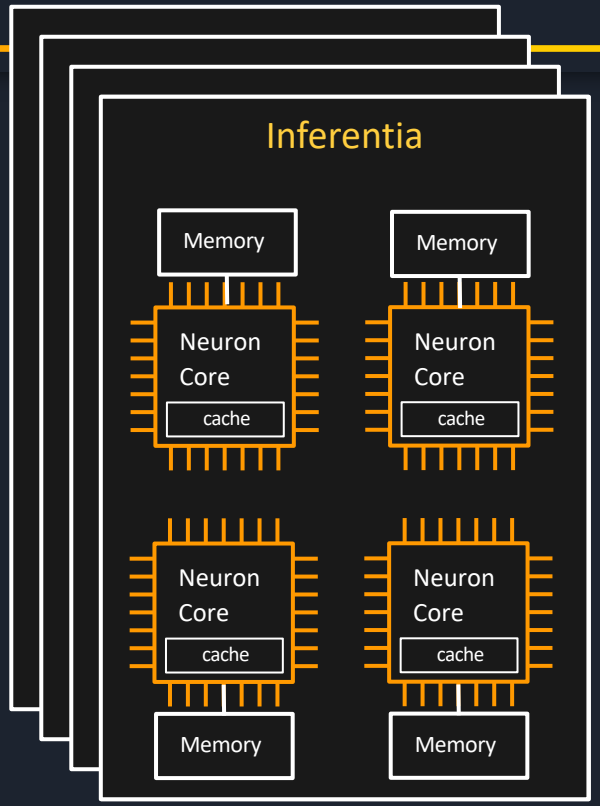
Intel® Xeon Scalable processors

AMD EPYC processors

AWS Graviton processors

AWS Inferentia ASIC

- 4 Neuron Cores
- Up to 128 TOPS
- 2-stage memory hierarchy
  - Large on-chip cache and commodity DRAM
- Supports FP16, BF16, INT8 data types
- Fast chip-to-chip interconnect

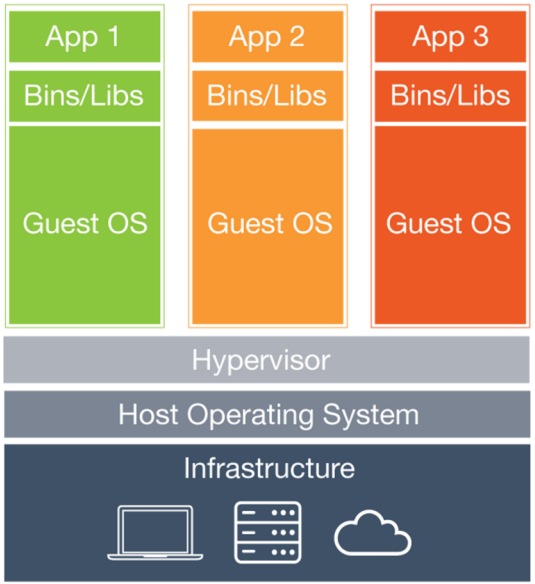


aws machine learning

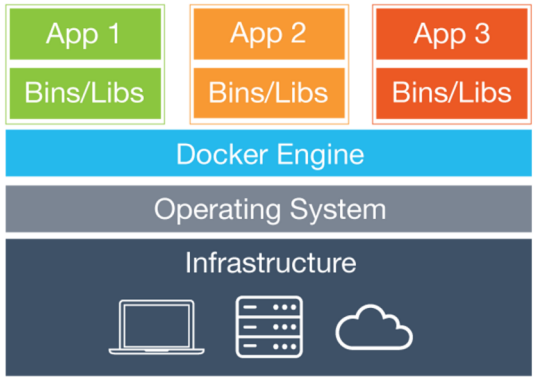
3. How?

Containers & Docker

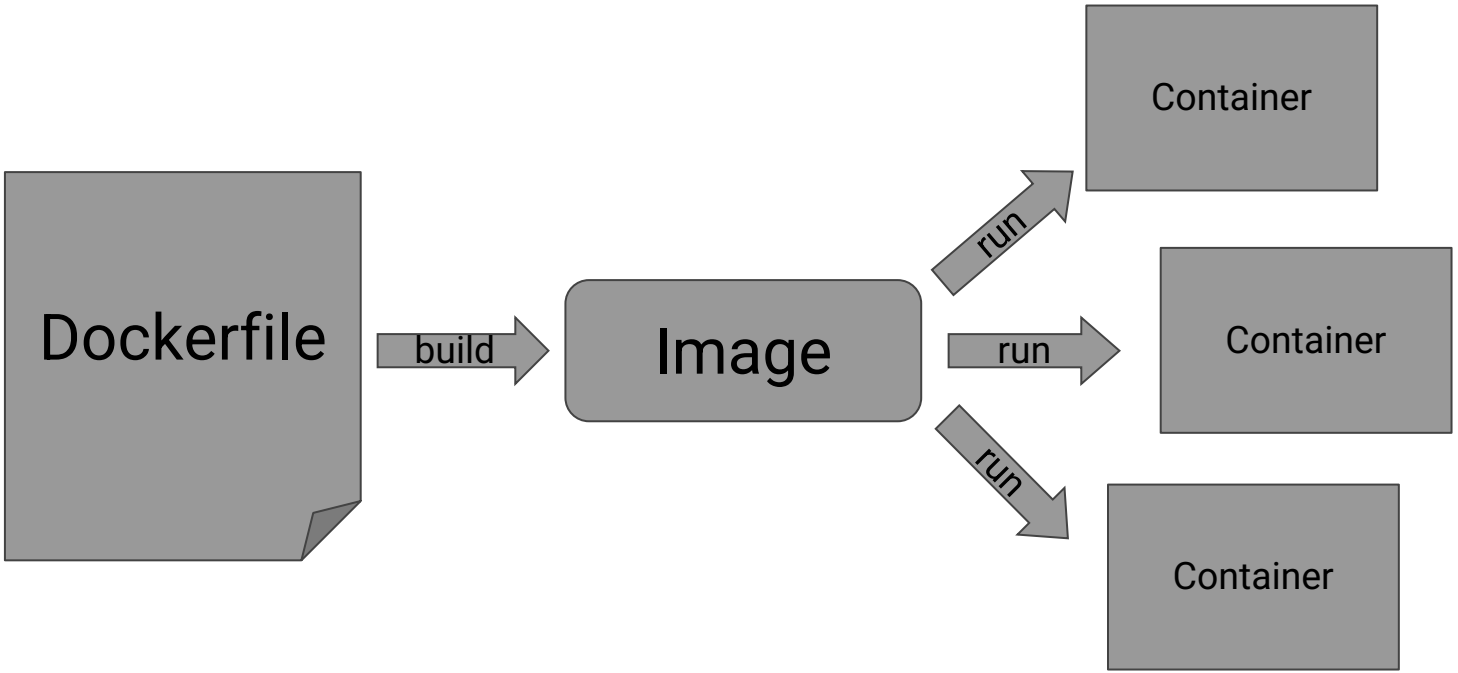
A container is a standard unit of software that packages up code and all its dependencies so the application runs quickly and reliably from one computing environment to another.



Virtual Machines



Containers


Dockerfile
build
Image
run
run
run
Container
Container
Container

Infrastructure as Code / **Terraform**

```
resource "aws_apprunner_service" "example" {
  service_name = "example"

  source_configuration {
    image_repository {
      image_configuration {
        port = "8000"
      }
      image_identifier      = "public.ecr.aws/jg/hello:latest"
      image_repository_type = "ECR_PUBLIC"
    }
  }

  tags = {
    Name = "example-apprunner-service"
  }
}
```

Infrastructure as Code / **CloudFormation**

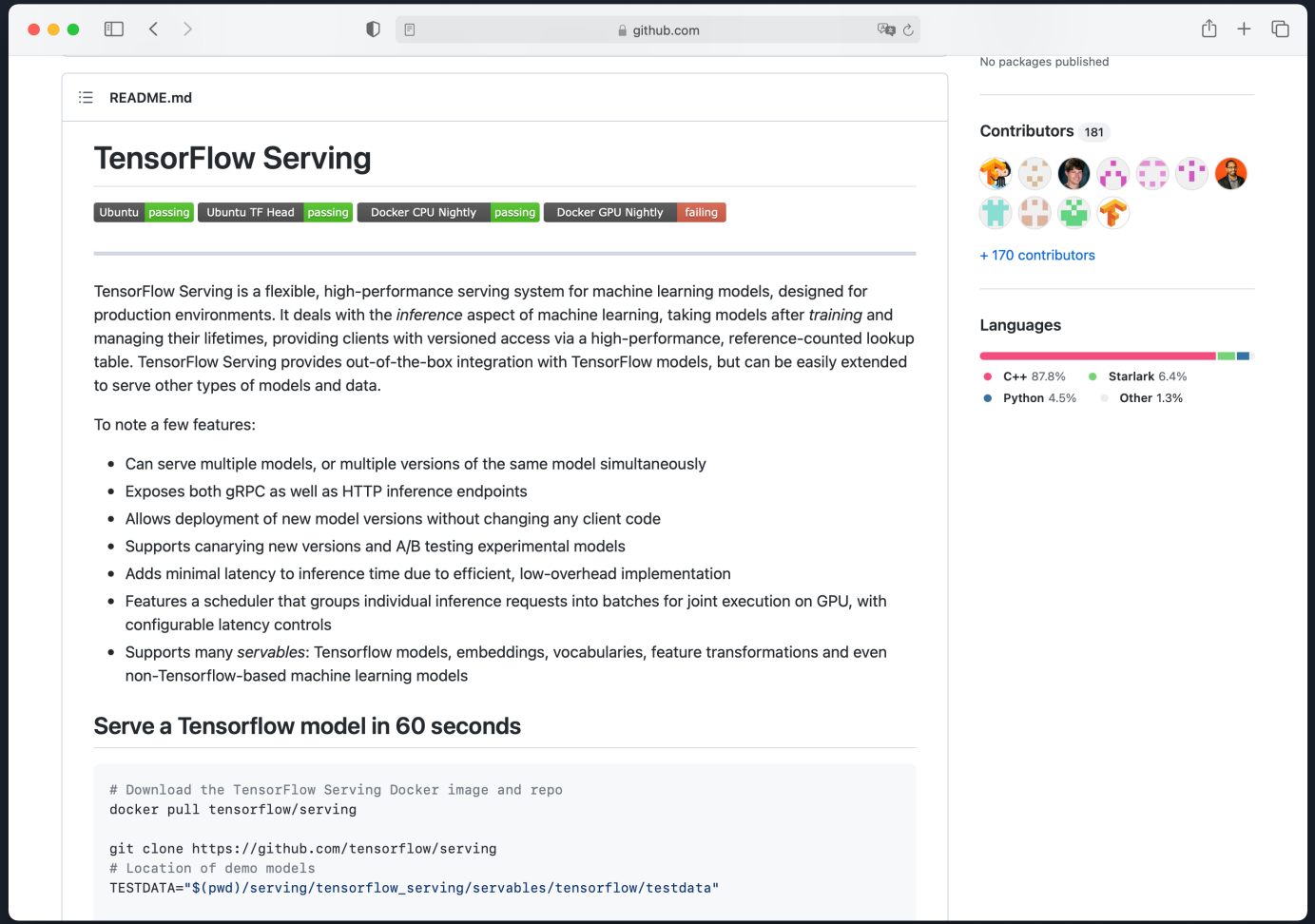
27 lines (27 sloc) | 915 Bytes — Raw | Blame

```yaml
# Copyright Amazon.com Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: Apache-2.0
AWSTemplateFormatVersion: 2010-09-09
Description: CloudFormation template that deploys hello-app-runner app
Parameters:
  ServiceName:
    Type: String
    Default: hello-app-runner
    Description: Name for your App Runner service.
Resources:
  Service:
    Metadata:
      'aws:apprunner:description': 'hello-app-runner example service'
    Type: AWS::AppRunner::Service
    Properties:
      ServiceName: !Ref ServiceName
      SourceConfiguration:
        AutoDeploymentsEnabled: false
        ImageRepository:
          ImageIdentifier: public.ecr.aws/aws-containers/hello-app-runner:latest
          ImageRepositoryType: ECR_PUBLIC
          ImageConfiguration:
            Port: 8000
Outputs:
  Endpoint:
    Description: "The endpoint of the App Runner service."
    Value: !GetAtt Service.ServiceUrl
```

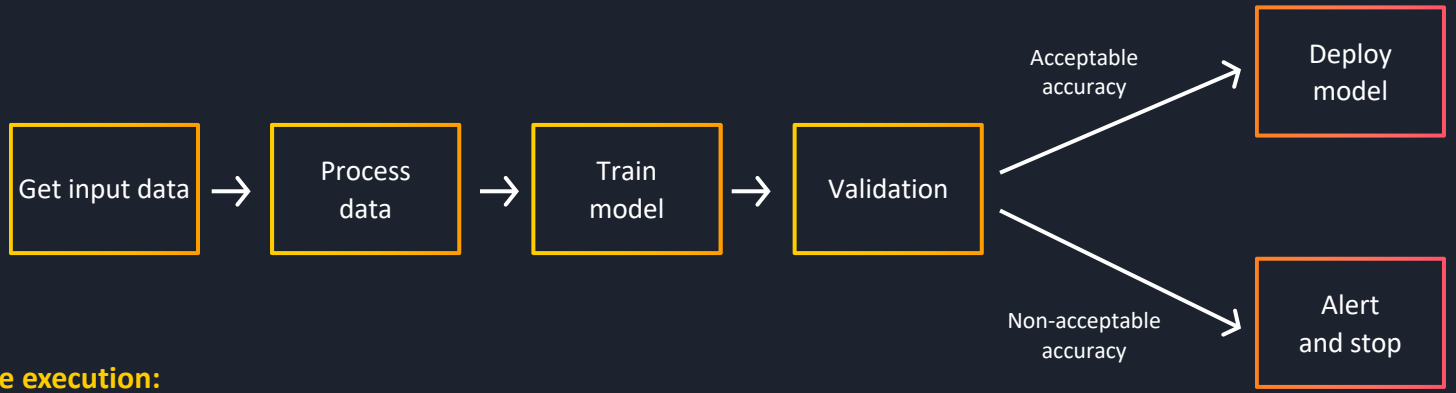
Infrastructure as Code / **CDK**

30 lines (24 sloc) | 841 Bytes — Raw | Blame

```
import { App, CfnOutput, Stack, StackProps } from 'aws-cdk-lib';
import * as apprunner from 'aws-cdk-lib/aws-apprunner';
import { Construct } from 'constructs';

export class AppRunnerStack extends Stack {
  constructor(scope: Construct, id: string, props: StackProps = {}) {
    super(scope, id, props);

    const service = new apprunner.CfnService(this, 'service', {
      serviceName: 'hello-apprunner',
      sourceConfiguration: {
        autoDeploymentsEnabled: false,
        imageRepository: {
          imageIdentifier: 'public.ecr.aws/aws-containers/hello-app-runner:latest',
          imageRepositoryType: 'ECR_PUBLIC',
        },
      },
    });

    new CfnOutput(this, 'url', {
      value: 'https://' + service.attrServiceUrl,
    });
  }
}

const app = new App();

new AppRunnerStack(app, 'hello-apprunner');

app.synth();
```


github.com
README.md
TensorFlow Serving
Ubuntu passing
Ubuntu TF Head passing
Docker CPU Nightly passing
Docker GPU Nightly failing
TensorFlow Serving is a flexible, high-performance serving system for machine learning models, designed for production environments. It deals with the inference aspect of machine learning, taking models after training and managing their lifetimes, providing clients with versioned access via a high-performance, reference-counted lookup table. TensorFlow Serving provides out-of-the-box integration with TensorFlow models, but can be easily extended to serve other types of models and data.
To note a few features:
Can serve multiple models, or multiple versions of the same model simultaneously
Exposes both gRPC as well as HTTP inference endpoints
Allows deployment of new model versions without changing any client code
Supports canarying new versions and A/B testing experimental models
Adds minimal latency to inference time due to efficient, low-overhead implementation
Features a scheduler that groups individual inference requests into batches for joint execution on GPU, with configurable latency controls
Supports many servables: Tensorflow models, embeddings, vocabularies, feature transformations and even non-Tensorflow-based machine learning models
Serve a Tensorflow model in 60 seconds
Download the TensorFlow Serving Docker image and repo
docker pull tensorflow/serving
git clone https://github.com/tensorflow/serving
# Location of demo models
TESTDATA="$(pwd)/serving/tensorflow_serving/servables/tensorflow/testdata"
No packages published
Contributors 181
+ 170 contributors
Languages
C++ 87.8%
Starlark 6.4%
Python 4.5%
Other 1.3%

ML Pipelines



Start pipeline execution:

- Manually
- New data uploads
- On schedule
- Upon code check-in

Creating and managing ML workflows

STITCH TOGETHER CODE RECIPES

Data prep — Model training — Model deployment

Compose and manage workflows

Create your ML workflows using the Python. Define the parameters and steps

Visualize the workflows

Include steps such as data transformations, training, debugging, and optimizing models

```python
# Example processing step for feature engineering

step_process = ProcessingStep(
    name="PreprocessAbaloneData",
    processor=sklearn_processor,
    outputs=[
        ProcessingOutput(output_name="train", source="/opt/ml/processing/train"),
        ProcessingOutput(output_name="validation", source="/opt/ml/processing/validation"),
        ProcessingOutput(output_name="test", source="/opt/ml/processing/test"),
    ],
    code=os.path.join(BASE_DIR, "preprocess.py"),
    job_arguments=["--input-data", input_data],
)

# Example training step

step_train = TrainingStep(
    name="TrainAbaloneModel",
    estimator=xgb_train,
    inputs={
        "train": TrainingInput(
            s3_data=step_process.properties.ProcessingOutputConfig.Outputs[
                "train"
            ].S3Output.S3Uri,
            content_type="text/csv",
        ),
        "validation": TrainingInput(
            s3_data=step_process.properties.ProcessingOutputConfig.Outputs[
                "validation"
            ].S3Output.S3Uri,
            content_type="text/csv",
        ),
    },
)
```

Task orchestration options

Open source party options

MLflow

Open source platform for the ML lifecycle

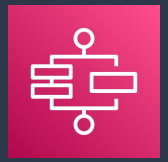
Apache Airflow

Platform to author, schedule and monitor workflows

Kubeflow

ML toolkit for Kubernetes

Native AWS options



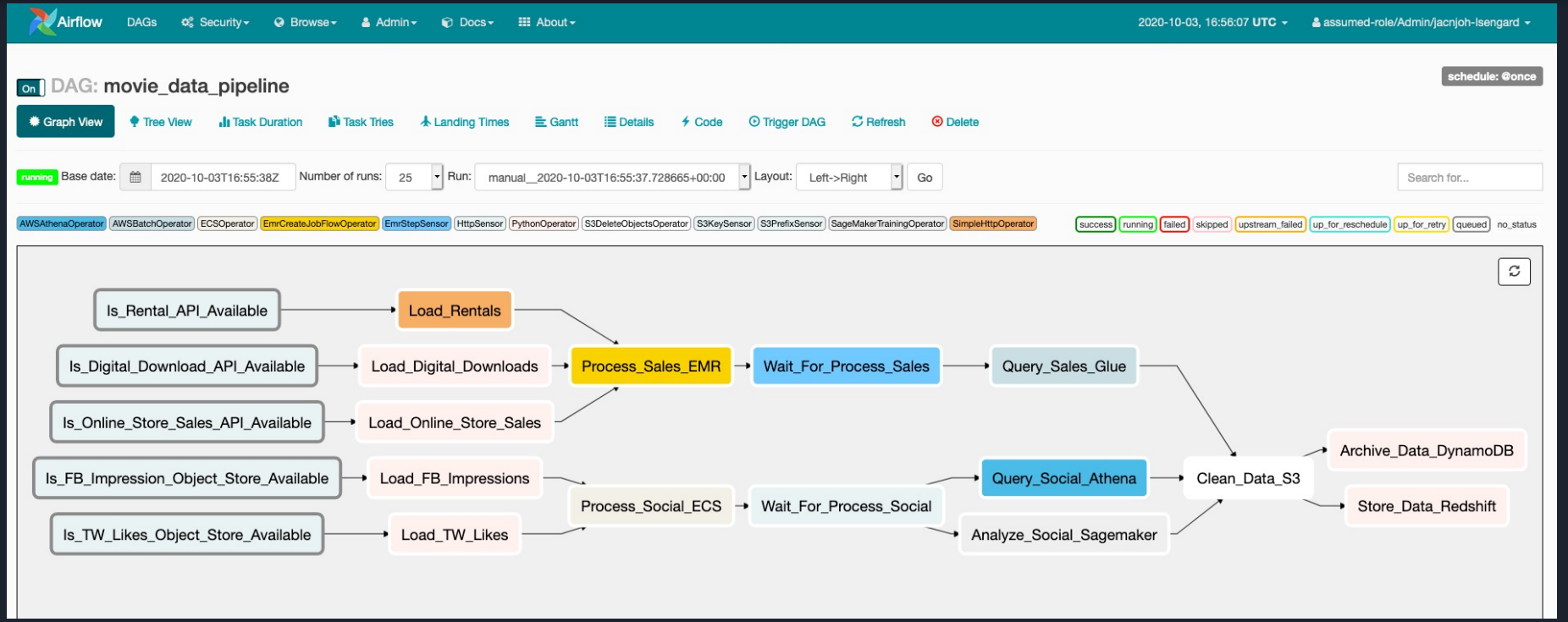
AWS Step Functions

Serverless pipeline orchestration

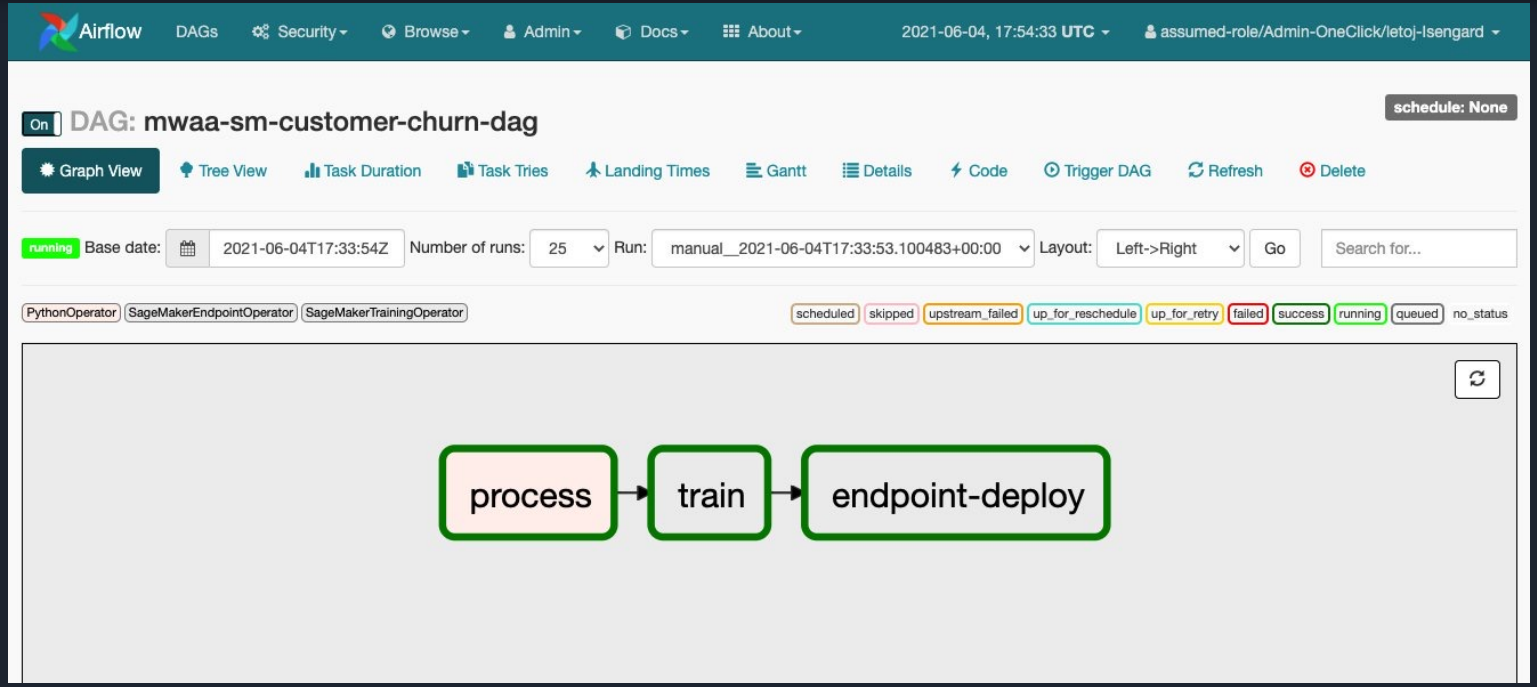
Amazon SageMaker Pipelines

Managed ML pipelines in SageMaker Studio

Apache Airflow



Apache Airflow



https://aws.amazon.com/blogs/machine-learning/orchestrate-xgboost-ml-pipelines-with-amazon-managed-workflows-for-apache-airflow/

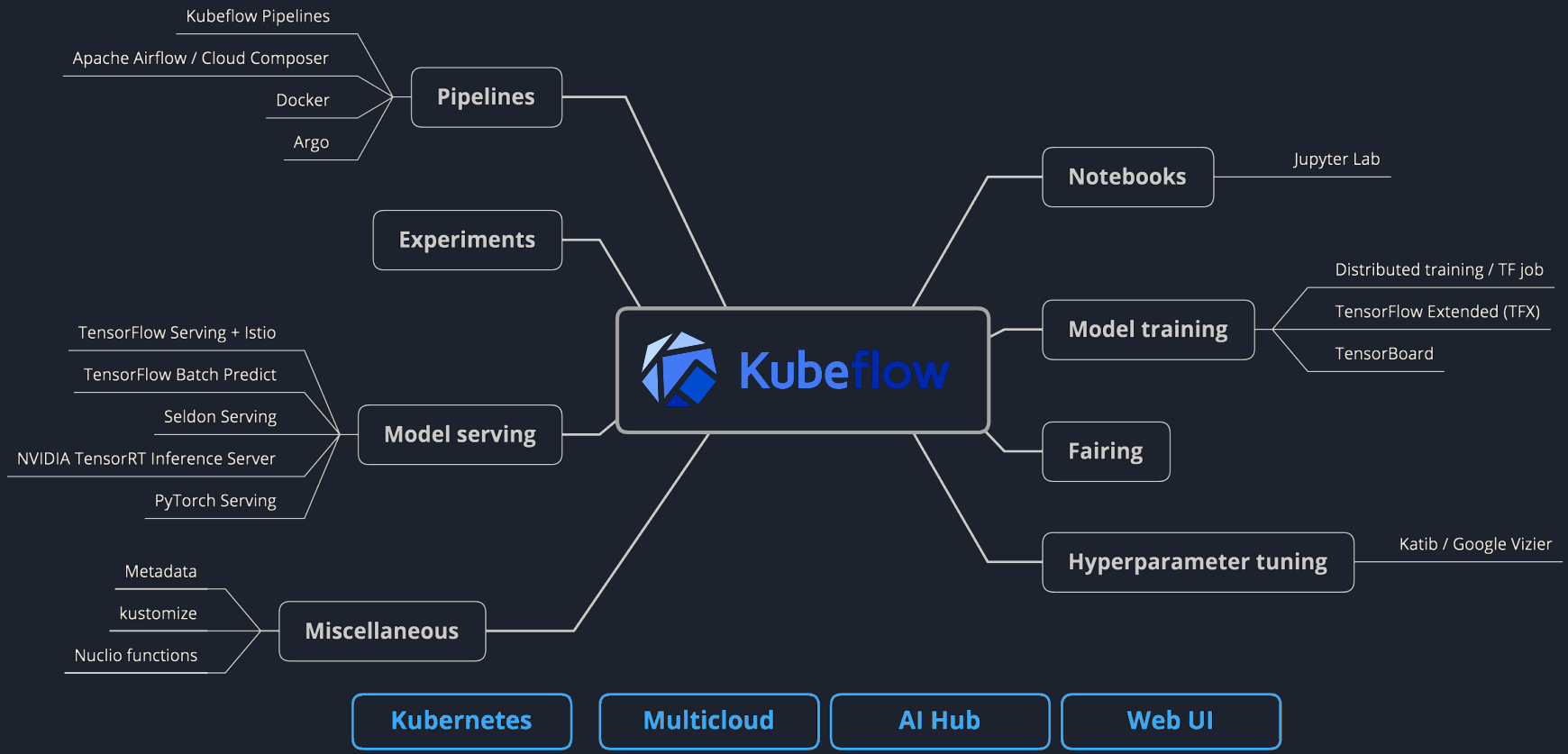
Monitoring Pipeline

Machine Learning lifecycle does not end after you have deployed your Machine Learning model. Part of the lifecycle is about tracking the model performance in production.

There are multiple sources of errors that could degrade your ML model performance, such as:

Model Drift

Data Drift



Version 1.1 20190807 @MichalBrys

aws machine learning

The AWS ML Stack

Broadest and most complete set of Machine Learning capabilities

AI SERVICES

VISION	SPEECH		TEXT			SEARCH	CHATBOTS	PERSONALIZATION	FORECASTING	FRAUD	DEVELOPMENT	CONTACT CENTERS
Amazon Rekognition	Amazon Polly	Amazon Transcribe *+Medical*	Amazon Comprehend *+Medical*	Amazon Translate	Amazon Textract	Amazon Kendra	Amazon Lex	Amazon Personalize	Amazon Forecast	Amazon Fraud Detector	Amazon CodeGuru	Contact Lens *For Amazon Connect*

ML SERVICES

Amazon SageMaker

Ground Truth

ML Marketplace

SageMaker Studio IDE

Built-in algorithms | Notebooks | Experiments | Model training & tuning | Debugger | Autopilot | Model hosting | Model Monitor

Neo

Augmented AI

ML FRAMEWORKS & INFRASTRUCTURE

TensorFlow | mxnet | PYTORCH

GLUON | Keras | DeepGraphLibrary

Deep Learning AMIs & Containers | GPUs & CPUs | Elastic Inference | Inferentia | FPGA


Questions?
DevOps & MLOps

> Thank you!