Modeling - 4 - ML Implementation and Operations

SageMaker's Inner Details and Production Variants

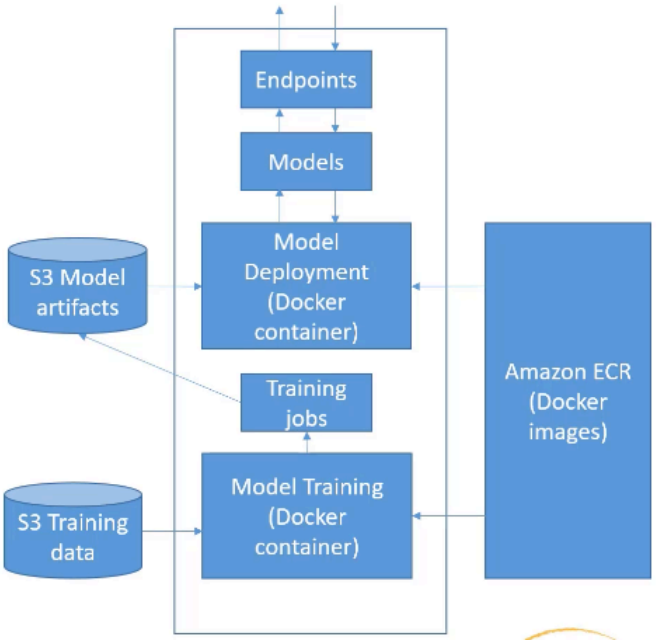
SageMaker + Docker

- All models in SageMaker are hosted in Docker containers
  - Pre-built deep learning
  - Pre-built scikit-learn and Spark ML
  - Pre-built Tensorflow, MXNet, Chainer, PyTorch
    - Distributed training via Horovod or Parameter Servers
  - Your own training and inference code! Or extend a pre-built image.
- This allows you to use any script or algorithm within SageMaker, regardless of runtime or language
  - Containers are isolated, and contain all dependencies and resources needed to run

Notes: **TensorFlow does not get distributed across multiple machines automatically.**
To distribute a training across multiple machines/GPUs, use a framework called **Horovod** or **Parameters Servers**

Using Docker

- Docker containers are created from *images*
- Images are built from a *Dockerfile*
- Images are saved in a *repository*
  - Amazon Elastic Container Registry



Structure of a training container

```
/opt/ml
├── input
│   ├── config
│   │   ├── hyperparameters.json
│   │   └── resourceConfig.json
│   └── data
│       └── <channel_name>
│           └── <input data>
├── model
│
├── code
│   └── <script files>
│
└── output
    └── failure
```

=> **Training code needs to redeployed to "/opt/ml/code"**

Structure of a Deployment Container

```
/opt/ml
└── model
    └── <model files>
```

=**> Inference code should be available under "/opt/ml/model"**

Entire Docker images includes:

Structure of your Docker image

- WORKDIR
  - nginx.conf
  - predictor.py
  - serve/
  - train/
  - wsgi.py

nginx.conf => for running a webserver
predictor.py => implements a flask web server, for making predictions at run time
Server/ this file launches the g-unicorn server which runs multiple instances of flask. And to run the container for inference
Train/ - to run training
Wsgi.py - for serving results (with flask)

Assembling it all in a Dockerfile

```
FROM tensorflow/tensorflow:2.0.0a0

RUN pip install sagemaker-containers

# Copies the training code inside the
container
COPY train.py /opt/ml/code/train.py

# Defines train.py as script entrypoint
ENV SAGEMAKER_PROGRAM train.py
```

$SAGEMAKER_PROGRAM => name entry point of training code
MUST specify this when setting up our own docker container.

Environment variables

- SAGEMAKER_PROGRAM
  - Run a script inside /opt/ml/code
- SAGEMAKER_TRAINING_MODULE
- SAGEMAKER_SERVICE_MODULE
- SM_MODEL_DIR
- SM_CHANNELS / SM_CHANNEL_*
- SM_HPS / SM_HP_*
- SM_USER_ARGS
- ...and many more

Using your own image

- cd dockerfile
- !docker build -t foo .
- from sagemaker.estimator import Estimator

```
estimator = Estimator(image_name='foo',
                      role='SageMakerRole',
                      train_instance_count=1,
                      train_instance_type='local')

estimator.fit()
```

You can **multiple models in SM on LIVE traffic using PRODUCTION VARIANTS**

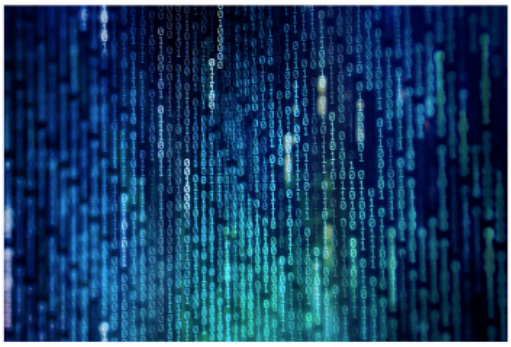
Production Variants

- You can test out multiple models on live traffic using Production Variants
  - Variant Weights tell SageMaker how to distribute traffic among them
  - So, you could roll out a new iteration of your model at say 10% variant weight
  - Once you're confident in its performance, ramp it up to 100%
- This lets you do A/B tests, and to validate performance in real-world settings
  - Offline validation isn't always useful

SageMaker Neo

SageMaker Neo

- Train once, run anywhere
  - Edge devices
    - ARM, Intel, Nvidia processors
    - Embedded in whatever – your car?
- Optimizes code for specific devices
  - Tensorflow, MXNet, PyTorch, ONNX, XGBoost
- Consists of a **compiler** and a **runtime**



=> In application where latency is important (self driving car), we want logic of the mode sitting in the car itself (and not rely on cloud cputing/httos/...)

Neo + AWS IoT Greengrass

- Neo-compiled models can be deployed to an HTTPS endpoint
  - Hosted on C5, M5, M4, P3, or P2 instances
  - Must be same instance type used for compilation
- OR! You can deploy to IoT Greengrass
  - This is how you get the model to an actual edge device
  - Inference at the edge with local data, using model trained in the cloud
  - Uses Lambda inference applications

Greengrass => mechanism to deploy code to an Edge device
- train model in the cloud with SM with training instances
- compiles it with Neo
- deploy it with GreenGrass

SageMaker Security: Encryption at Rest and in Transit

General AWS Security

- Use Identity and Access Management (IAM)
  - Set up user accounts with only the permissions they need
- Use MFA
- Use SSL/TLS when connecting to anything
- Use CloudTrail to log API and user activity
- Use encryption
- Be careful with PII

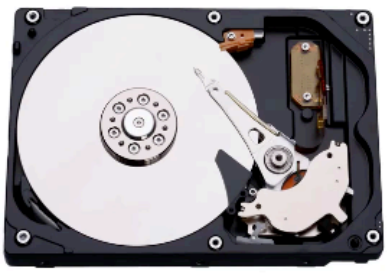
CloudTrail vs CloudWatch

- **CloudTrail** is for AUDITING, leaving a TRAIL of activities a log of what everyone did (API)
- **CloudWatch** is for MONITORING log data and issuing alarms when something goes wrong

Encryption: both at REST and in Transit

Protecting your Data at Rest in SageMaker

- AWS Key Management Service (KMS)
  - Accepted by notebooks and all SageMaker jobs
    - Training, tuning, batch transform, endpoints
    - Notebooks and everything under /opt/ml/ and /tmp can be encrypted with a KMS key
- S3
  - Can use encrypted S3 buckets for training data and hosting models
  - S3 can also use KMS



Protecting Data in Transit in SageMaker

- All traffic supports TLS / SSL
- IAM roles are assigned to SageMaker to give it permissions to access resources
- Inter-node training communication may be optionally encrypted
  - Can increase training time and cost with deep learning
  - AKA inter-container traffic encryption
  - Enabled via console or API when setting up a training or tuning job

=> follow “least access” guideline

SageMaker Security: VPC’s, IAM, Logging and Monitoring

SageMaker + VPC

- Training jobs run in a Virtual Private Cloud (VPC)
- You can use a private VPC for even more security
  - You’ll need to set up S3 VPC endpoints
  - Custom endpoint policies and S3 bucket policies can keep this secure
- Notebooks are Internet-enabled by default
  - This can be a security hole
  - If disabled, your VPC needs an interface endpoint (PrivateLink) or NAT Gateway, and allow outbound connections, for training and hosting to work
- Training and Inference Containers are also Internet-enabled by default
  - Network isolation is an option, but this also prevents S3 access

REMEMBER: If we disable public internet traffic (default) to SageMaker Notebook, need to set up a PrivateLink or NAT Gateway to make sure we can get to the VPC to get the outbound connection opened to allow training and hosting to work

SageMaker + IAM

- User permissions for:
  - CreateTrainingJob
  - CreateModel
  - CreateEndpointConfig
  - CreateTransformJob
  - CreateHyperParameterTuningJob
  - CreateNotebookInstance
  - UpdateNotebookInstance
- Predefined policies:
  - AmazonSageMakerReadOnly
  - AmazonSageMakerFullAccess
  - AdministratorAccess
  - DataScientist

SageMaker Logging and Monitoring

- CloudWatch can log, monitor and alarm on:
  - Invocations and latency of endpoints
  - Health of instance nodes (CPU, memory, etc)
  - Ground Truth (active workers, how much they are doing)
- CloudTrail records actions from users, roles, and services within SageMaker
  - Log files delivered to S3 for auditing

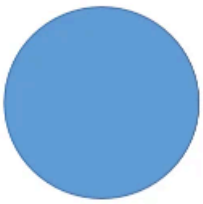
SageMaker Resource Management: Instance Types and Spot Training

Choosing your instance types

- We covered this under "modeling", even though it's an operations concern
- In general, algorithms that rely on deep learning will benefit from GPU instances (P2 or P3) for training
- Inference is usually less demanding and you can often get away with compute instances there (C4, C5)
- GPU instances can be really pricey

Managed Spot Training

- Can use EC2 Spot instances for training
  - Save up to 90% over on-demand instances
- Spot instances can be interrupted!
  - Use checkpoints to S3 so training can resume
- Can increase training time as you need to wait for spot instances to become available



SageMaker Resource Management: Elastic Inference, Automatic Scaling and AZ's

When deploying inference model to the world, we deploy that to a CPU instance
We can also add an "EI" accelerator alongside the CPU instance
Notes: EI can only work with Deeplearning frameworks => Tensorflow, MXnet, Image Classification, Object Detection

Elastic Inference

- Accelerates deep learning inference
  - At fraction of cost of using a GPU instance for inference
- EI accelerators may be added alongside a CPU instance
  - ml.eia1.medium / large / xlarge
- EI accelerators may also be applied to notebooks
- Works with Tensorflow and MXNet pre-built containers
  - ONNX may be used to export models to MXNet
- Works with custom containers built with EI-enabled Tensorflow or MXNet
- Works with Image Classification and Object Detection built-in algorithms

Automatic Scaling

- You set up a scaling policy to define target metrics, min/max capacity, cooldown periods
- Works with CloudWatch
- Dynamically adjusts number of instances for a production variant
- Load test your configuration before using it!

SageMaker and Availability Zones

- SageMaker automatically attempts to distribute instances across availability zones
- But you need more than one instance for this to work!
- Deploy multiple instances for each production endpoint
- Configure VPC's with at least two subnets, each in a different AZ

SageMaker Inference Pipelines

Inference Pipelines

- Linear sequence of 2-5 containers
- Any combination of pre-trained built-in algorithms or your own algorithms in Docker containers
- Combine pre-processing, predictions, post-processing
- Spark ML and scikit-learn containers OK
  - Spark ML can be run with Glue or EMR
  - Serialized into MLeap format
- Can handle both real-time inference and batch transforms

Quiz

Question 1:

Where does SageMaker's automatic scaling get the data it needs to determine how many endpoints you need?

- (●) CloudWatch
- ( ) CloudTrail
- ( ) EC2

CloudWatch is a repository of performance metrics associated with your endpoints, which SageMaker can use to determine if you have the right amount of them.

Your SageMaker inference is based on a Tensorflow or MXNet network. You want it to be fast, but don't want to pay for P2 or P3 inference nodes. What's a good solution?

- ○ Use inference pipelines
- ● Use Elastic Inference
- ○ Use an M4 inference node

EI accelerators can be attached to CPU inference instances to accelerate deep learning inference at a fraction of the cost of using a GPU inference node.

Question 3:

You are deploying SageMaker inside a VPC, but the Internet access from SageMaker notebooks is considered a security hole. How might you address this?

- ● Disable internet access when creating the notebook, and set up a NAT Gateway to allow the outbound connections needed for training and hosting.
- ○ Enable network isolation
- ○ Enable SSL / TLS in SageMaker

Alternatively, PrivateLink endpoints could be used instead of a NAT Gateway.

Question 4:

You want to deploy your trained semantic segmentation model from SageMaker to an embedded ARM device in a car. Which services might you use to accomplish this?

- ○ Lambda and Lex
- ● SageMaker Neo and IoT GreenGrass
- ○ AWS DeepRacer and DeepLens

Neo can compile your model to an ARM processor, and GreenGrass can get it to the device you want.

Question 5:

When constructing your own training container for SageMaker, where should the actual training script files be stored?

- ● /opt/ml/code/
- ○ /opt/ml/model/
- ○ /opt/ml/train/

This is where your code should go. The SAGEMAKER_PROGRAM environment variable will indicate the specific script that should be run.