INNOVATE

ONLINE CONFERENCE

MACHINE LEARNING AND AI EDITION

aws | intel

INNOVATE
ONLINE CONFERENCE

MACHINE LEARNING
AND AI EDITION

Scale machine learning from zero to millions of users

Sébastien Stormacq
Developer Advocate
Amazon Web Services

@sebsto

Day 0: Trying to avoid ML altogether

No ML is easier to manage than no ML

INNOVATE
ONLINE CONFERENCE
MACHINE LEARNING
AND AI EDITION

High-level services: Call an API, get the job done

Vision			Speech		Language		Chatbots	Forecasting	Recommendations
AMAZON REKOGNITION IMAGE	AMAZON REKOGNITION VIDEO	AMAZON TEXTRACT	AMAZON POLLY	AMAZON TRANSCRIBE	AMAZON TRANSLATE	AMAZON COMPREHEND & AMAZON COMPREHEND MEDICAL	AMAZON LEX	AMAZON FORECAST	AMAZON PERSONALIZE

Pre-trained AI services that require no ML skills

Ability to easily add intelligence to your existing apps and workflows

Quality and accuracy from continuously learning APIs

High-level != generic

Vision			Speech		Language		Chatbots	Forecasting	Recommendations
AMAZON REKOGNITION IMAGE	AMAZON REKOGNITION VIDEO	AMAZON TEXTRACT	AMAZON POLLY	AMAZON TRANSCRIBE	AMAZON TRANSLATE	AMAZON COMPREHEND & AMAZON COMPREHEND MEDICAL	AMAZON LEX	AMAZON FORECAST	AMAZON PERSONALIZE

Amazon Polly: Custom lexicons, SSML

Amazon Transcribe: Custom vocabulary

Amazon Translate: Custom terminology

Amazon Comprehend: Custom text classification & entity extraction

Amazon Forecast & **Amazon Personalize**: Train on your own data!

Day 1: One user (you)

Breaking out of the sandbox

INNOVATE
ONLINE CONFERENCE
MACHINE LEARNING AND AI EDITION

And so it begins

- You've trained a model on a local machine, using a popular open-source library
- You've measured the model's accuracy, and things look good; now you'd like to deploy it to check its actual behavior, to run A/B tests, etc.
- You've embedded the model in your business application
- You've deployed everything to a single Ubuntu virtual machine in the cloud
- Everything works, you're serving predictions, life is good!

Scorecard

	Single Amazon EC2 instance
Infrastructure effort	C'mon, it's just one instance
ML setup effort	pip install tensorflow
CI/CD integration	Not needed
Build models	DIY
Train models	python train.py
Deploy models (at scale)	python predict.py
Scale/HA inference	Not needed
Optimize costs	Not needed
Security	Not needed

Week 1

INNOVATE
ONLINE CONFERENCE

MACHINE LEARNING
AND AI EDITION

A few instances and models later…

- Life is not that good
- Too much manual work
  - Time-consuming and error-prone
  - Dependency challenges
  - No cost optimization
- Monolithic architecture
  - Deployment challenges
  - Multiple apps can't share the same model
  - Apps and models scale differently

Use AWS-maintained tools
- Deep Learning AMIs
- Deep Learning Containers

Dockerize

Create a prediction service
- Model servers
- Bespoke/custom API (Flask?)

AWS Deep Learning AMIs

Optimized environments on Amazon Linux or Ubuntu

NEW (March 27)
Deep Learning Containers

Conda AMI

For developers who want pre-installed pip packages of DL frameworks in separate virtual environments

Base AMI

For developers who want a clean slate to set up private DL engine repositories or custom builds of DL engines

Demo

Running an Amazon EC2 instance with the Deep Learning AMI
Connecting to Jupyter
Training with the TensorFlow deep learning container

INNOVATE
ONLINE CONFERENCE
MACHINE LEARNING AND AI EDITION

Running a new EC2 instance with the Deep Learning AMI

```
aws ec2 run-instances \
    --image-id ami-02273e0d16172dbd1 \   # Deep Learning AMI in eu-west-1
    --instance-type p3.2xlarge \
    --instance-market-options '{"MarketType":"spot"}' \
    --tag-specifications 'ResourceType=instance,Tags=[{Key=Name,Value=dlami-demo}]' \
    --key-name $KEYPAIR \
    --security-group-ids $SECURITY_GROUP \
    --iam-instance-profile Name=$ROLE
```

Connecting to Jupyter

On your local machine

```
ssh -L 8000:localhost:8888 ec2-user@INSTANCE_NAME
```

On the EC2 instance

```
jupyter notebook --no-browser --port=8888
```

On your local machine

Open http://localhost:8000

Training with the TensorFlow deep learning container

List of image names: https://docs.aws.amazon.com/dlami/latest/devguide/deep-learning-containers-images.html

On the training machine

```
$(aws ecr get-login --no-include-email --region eu-west-1 --registry-ids 763104351884)

docker pull 763104351884.dkr.ecr.eu-west-1.amazonaws.com/tensorflow-training:1.13-horovod-gpu-py27-cu100-ubuntu16.04

nvidia-docker run -it 763104351884.dkr.ecr.eu-west-1.amazonaws.com/tensorflow-training:1.13-horovod-gpu-py27-cu100-ubuntu16.04
```

In the container

```
git clone https://github.com/fchollet/keras.git

python keras/examples/mnist_cnn.py
```

And then one day…

INNOVATE
ONLINE CONFERENCE
MACHINE LEARNING AND AI EDITION

Scaling alert!

- More customers, more team members, more models, woo-hoo!
- Scalability, high availability & security are now a thing
- Scaling up is a losing proposition—you need to scale out
- Only automation can save you: IaC, CI/CD, and all that good DevOps stuff
- What are your options?

Option 1: Virtual machines

- Definitely possible, but:
  - Why? Seriously, I want to know.
  - Operational and financial issues await if you don't automate extensively.
- Training
  - Build on-demand clusters with AWS CloudFormation, Terraform, etc.
  - Distributed training is a pain to set up
- Prediction
  - Automate deployment with CI/CD
  - Scale with Auto Scaling, Load Balancers, etc.
- Spot, spot, spot

Scorecard

	More Amazon EC2 instances
Infrastructure effort	Lots
ML setup effort	Some (DL AMI)
CI/CD integration	No change
Build models	DIY
Train models	DIY
Deploy models	DIY (model servers)
Scale/HA inference	DIY (Auto Scaling, LB)
Optimize costs	DIY (Spot, automation)
Security	DIY (IAM, VPC, KMS)

Option 2: Docker clusters

- This makes a lot of sense if you're already deploying apps to Docker
  - No change to the dev experience: Same workflows, same CI/CD, etc.
  - Deploy prediction services on the same infrastructure as business apps
- Amazon ECS and Amazon EKS
  - Lots of flexibility: Mixed instance types (including GPUs), placement constraints, etc.
  - Both come with AWS-maintained AMIs that will save you time
- One cluster or many clusters?
  - Build on-demand development and test clusters with AWS CloudFormation, Terraform, etc.
  - Many customers find that running a large single production cluster works better
- Still instance-based and not fully managed
  - Not a hands-off operation: Services/pods, service discovery, etc. are nice but you still have work to do
  - And yes, this matters even if "someone else is taking care of clusters"

Demo

Creating an ECS cluster with 4 GPU instances and 2 CPU instances
Running TensorFlow training and prediction

INNOVATE
ONLINE CONFERENCE
MACHINE LEARNING
AND AI EDITION

Creating an Amazon ECS cluster and adding instances

```
aws ecs create-cluster --cluster-name ecs-demo

# Add 4 p2.xlarge spot instances, ECS-optimized AMI with GPU support, default VPC
aws ec2 run-instances --image-id ami-0638eba79fcfe776e \
       --count 4 \
       --instance-type p2.xlarge \
       --instance-market-options '{"MarketType":"spot"}' \
       --tag-specifications 'ResourceType=instance,Tags=[{Key=Name,Value=ecs-demo}]'
       --key-name $KEYPAIR \
       --security-group-ids $SECURITY_GROUP \
       --iam-instance-profile Name=$ROLE
       --user-data file://user-data.txt

# Add 2 c5.2xlarge, ECS-optimized AMI, default VPC, different subnet
aws ec2 run-instances --image-id ami-09cd8db92c6bf3a84 \
       --count 2 \
       --instance-type c5.2xlarge \
       --instance-market-options '{"MarketType":"spot"}' \
       --subnet $SUBNET_ID \
       . . .
```

Defining the training task

```
"containerDefinitions": [{

    "command": [
      "git clone https://github.com/fchollet/keras.git && python keras/examples/mnist_cnn.py"],

    "entryPoint": [ "sh","-c"],

    "name": "TFconsole",

    "image": "763104351884.dkr.ecr.eu-west-1.amazonaws.com/tensorflow-training:1.13-horovod-gpu-py36-
cu100-ubuntu16.04",

    "memory": 4096,

    "cpu": 256,

    "resourceRequirements" : [ {"type" : "GPU", "value" : "1"} ],

    . . .
```

Defining the inference task

```
"containerDefinitions": [{

    "command": [
      "git clone -b r1.13 https://github.com/tensorflow/serving.git && tensorflow_model_server
--port=8500 --rest_api_port=8501 --model_name=<MODEL_NAME> --model_base_path=<MODEL_PATH>"],

    "entryPoint": [ "sh","-c"],

    "name": "TFinference",

    "image": "763104351884.dkr.ecr.eu-west-1.amazonaws.com/tensorflow-inference:1.13-cpu-py36-
ubuntu16.04",

    "memory": 4096,

    "cpu": 256,

    "portMappings": [{ "hostPort": 8500, "protocol": "tcp", "containerPort": 8500},
                     { "hostPort": 8501, "protocol": "tcp", "containerPort": 8501},

. . .
```

Running training and inference on the cluster

```
# Create task definitions for training and inference
aws ecs register-task-definition --cli-input-json file://training.json
aws ecs register-task-definition --cli-input-json file://inference.json


# Run 4 training tasks (the GPU requirement is in the task definition)
aws ecs run-task --cluster ecs-demo --task-definition training:1 --count 4

# Create inference service, starting with 1 initial task
# Run it on c5 instance, and spread tasks evenly
aws ecs create-service --cluster ecs-demo \
 --service-name inference-cpu \
 --task-definition inference:1 \
 --desired-count 1 \
 --placement-constraints type="memberOf",expression="attribute:ecs.instance-type =~ c5.*" \
 --placement-strategy field="instanceId",type="spread"

# Scale inference service to 2 tasks
aws ecs update-service --cluster ecs-demo --service inference-cpu --desired-count 2
```

Scorecard

	Amazon EC2	Amazon ECS / Amazon EKS
Infrastructure effort	Lots	Some (Docker tools)
ML setup effort	Some (DL AMI)	Some (DL containers)
CI/CD integration	No change	No change
Build models	DIY	DIY
Train models (at scale)	DIY	DIY (Docker tools)
Deploy models (at scale)	DIY (model servers)	DIY (Docker tools)
Scale/HA inference	DIY (Auto Scaling, LB)	DIY (Services, pods, etc.)
Optimize costs	DIY (Spot, RIs, automation)	DIY (Spot, RIs, automation)
Security	DIY (IAM, VPC, KMS)	DIY (IAM, VPC, KMS)

Option 3: Go fully managed with Amazon SageMaker

Collect and prepare training data

Choose and optimize your ML algorithm

Set up and manage environments for training

Train and tune ML models

Deploy models in production

Scale and manage the production environment

Same service and APIs from experimentation to production

Model options on Amazon SageMaker

Training code

Factorization Machines
Linear Learner
Principal Component Analysis
K-Means Clustering
XGBoost
And more

Chainer
TensorFlow
PyTorch
scikit learn

Built-in Algorithms (17)

No ML coding required
No infrastructure work required
Distributed training
Pipe mode

Built-in Frameworks

Bring your own code: Script mode
Open-source containers
No infrastructure work required
Distributed training
Pipe mode

Bring Your Own Container

Full control, run anything!
R, C++, etc.
No infrastructure work required

The Amazon SageMaker API

- Python SDK orchestrating all Amazon SageMaker activity
  - High-level objects for algorithm selection, training, deploying, automatic model tuning, etc.
  - Spark SDK (Python & Scala)

- AWS SDK
  - For scripting and automation
  - CLI: '*aws sagemaker*'
  - Language SDKs: boto3, etc.

Training and deploying

```
tf_estimator = TensorFlow(entry_point='mnist_keras_tf.py',
                          role=role,
                          train_instance_count=1,
                          train_instance_type='ml.c5.2xlarge',
                          framework_version='1.12',
                          py_version='py3',
                          script_mode=True,
                          hyperparameters={
                                    'epochs': 10,
                                    'learning-rate': 0.01})

tf_estimator.fit(data)

# HTTPS endpoint backed by a single instance
tf_endpoint = tf_estimator.deploy(initial_instance_count=1,instance_type=ml.t3.xlarge)

tf_endpoint.predict(…)
```

Training and deploying, at any scale

```
tf_estimator = TensorFlow(entry_point= my_crazy_cnn.py',
                          role=role,
                          train_instance_count=8,
                          train_instance_type='ml.p3.16xlarge',    # Total of 64 GPUs
                          framework_version='1.12',
                          py_version='py3',
                          script_mode=True,
                          hyperparameters={
                                    'epochs': 200,
                                    'learning-rate': 0.01})

tf_estimator.fit(data)

# HTTPS endpoint backed by 16 multi-AZ load-balanced instances
tf_endpoint = tf_estimator.deploy(initial_instance_count=16, instance_type=ml.p3.2xlarge)

tf_endpoint.predict(…)
```

Demo

https://gitlab.com/juliensimon/dlnotebooks/blob/master/sagemaker/08-Image-classification-advanced.ipynb

INNOVATE
ONLINE CONFERENCE

MACHINE LEARNING
AND AI EDITION

Scorecard

	Amazon EC2	Amazon ECS / Amazon EKS	Amazon SageMaker
Infrastructure effort	Maximal	Some (Docker tools)	None
ML setup effort	Some (DL AMI)	Some (DL containers)	Minimal
CI/CD integration	No change	No change	Some (SDK, Step Functions)
Build models	DIY	DIY	17 built-in algorithms
Train models (at scale)	DIY	DIY (Docker tools)	2 LOCs
Deploy models (at scale)	DIY (model servers)	DIY (Docker tools)	1 LOCs
Scale/HA inference	DIY (Auto Scaling, LB)	DIY (Services, pods, etc.)	Built-in
Optimize costs	DIY (Spot, RIs, automation)	DIY (Spot, RIs, automation)	On-demand training, Spot Auto Scaling for inference
Security	DIY (IAM, VPC, KMS)	DIY (IAM, VPC, KMS)	API parameters

Scorecard

	Amazon EC2	Amazon ECS / Amazon EKS	Amazon SageMaker
Infrastructure effort	Maximal	Some (Docker tools)	None
ML setup effort	Some (DL AMI)	Some (DL containers)	Minimal
CI/CD integration	No change	No change	Some (SDK, Step Functions)
Build models	DIY	DIY	17 built-in algorithms
Train models (at scale)	DIY	DIY (Docker tools)	2 LOCs
Deploy models (at scale)	DIY (model servers)	DIY (Docker tools)	1 LOCs
Scale/HA inference	DIY (Auto Scaling, LB)	DIY (Services, pods, etc.)	Built-in
Optimize costs	DIY (Spot, RIs, automation)	DIY (Spot, RIs, automation)	On-demand training, Spot, Auto Scaling for inference,
Security	DIY (IAM, VPC, KMS)	DIY (IAM, VPC, KMS)	API parameters
Personal opinion	Small scale only, unless you have strong DevOps skills and enjoy exercising them	Reasonable choice if you're a Docker shop and know how to use the rich Docker ecosystem; if not, I'd think twice	Learn it in a few hours, forget about servers, focus 100% on ML, enjoy goodies like Pipe mode, distributed training, HPO, and inference pipelines

Conclusion

- Whatever works for you at this time is fine
  - Don't over-engineer, and don't "plan for the future"
  - Fight "we've always done it like this," NIH, and Hype-Driven Development
  - Optimize for current business conditions, pay attention to TCO
- Models and data matter, not infrastructure
  - When conditions change, move fast: Smash and rebuild
  - … which is what cloud is all about!
  - "100% of our time spent on ML" shall be the whole of the Law
- Mix and match
  - Train on Amazon SageMaker, deploy on ECS/EKS… or vice versa
  - Write your own story!

Getting started

https://aws.amazon.com/free

https://aws.ai

https://aws.amazon.com/machine-learning/amis/
https://aws.amazon.com/machine-learning/containers/

https://aws.amazon.com/sagemaker
https://github.com/aws/sagemaker-python-sdk
https://github.com/awslabs/amazon-sagemaker-examples

https://medium.com/@julsimon
https://gitlab.com/juliensimon/dlcontainers DL AMI / container demos
https://gitlab.com/juliensimon/dlnotebooks Amazon SageMaker notebooks

Thank you!

Sébastien Stormacq
Developer Advocate
Amazon Web Services

@sebsto

INNOVATE
ONLINE CONFERENCE
MACHINE LEARNING AND AI EDITION