Floor 28, Tel Aviv, July 8th, 2019

Train and deploy your Machine Learning workloads with AWS container services

Julien Simon
Global Evangelist, AI & Machine Learning, AWS
@julsimon

Once upon a time…

Your first Machine Learning project

- 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 behaviour, 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!

Score card

	Single 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

Scaling alert!

- More customers, more team members, more models, woohoo!
- 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 CloudFormation, Terraform, etc.
  - Distributed training is a pain to set up
- Prediction
  - Automate deployement with CI/CD
  - Scale with Auto Scaling, Load Balancers, etc.
- Spot, spot, spot

AWS Deep Learning AMIs

Optimized environments on Amazon Linux or Ubuntu

NEW (March 2019) 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

Deep Learning containers

Score card

	More 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 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 / deployments, 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 GPU instances and CPU instances
Running Tensorflow training and prediction

https://gitlab.com/juliensimon/dlcontainers/tree/master/ecs

Demo

Creating an EKS cluster with GPU instances and CPU instances
Running Tensorflow training and prediction

https://gitlab.com/juliensimon/dlcontainers/tree/master/eks

Score card

	EC2	ECS / 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)

What about AWS Fargate?

- AWS Fargate lets you run containers without having to manage servers
- Per-container configuration
  - 0.25 vCPU 0.5GB, 1GB, and 2GB
  - 0.5 vCPU Min. 1GB and Max. 4GB, in 1GB increments
  - 1 vCPU Min. 2GB and Max. 8GB, in 1GB increments
  - 2 vCPU Min. 4GB and Max. 16GB, in 1GB increments
  - 4 vCPU Min. 8GB and Max. 30GB, in 1GB increments
  - $0.04048 / vCPU / hour, $0.004445 / GB / hour
- No GPU available
- AWS Deep Learning containers not officially supported

Demo

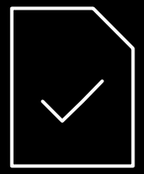
Deploy an Amazon SageMaker model on AWS Fargate

https://gitlab.com/juliensimon/dlnotebooks/blob/master/keras/04-fashion-mnist-sagemaker-advanced/Fashion%20MNIST-SageMaker- Fargate.ipynb

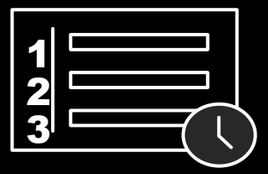
Score card

	EC2	ECS / EKS	Fargate for ECS
Infrastructure effort	Lots	Some (Docker tools)	None
ML setup effort	Some (DL AMI)	Some (DL containers)	Some (DIY containers)
CI/CD integration	No change	No change	No change
Build models	DIY	DIY	DIY
Train models (at scale)	DIY	DIY (Docker tools)	Not compelling
Deploy models (at scale)	DIY (model servers)	DIY (Docker tools)	DIY (Docker tools)
Scale/HA inference	DIY (Auto Scaling, LB)	DIY (Services, pods, etc.)	DIY (Auto Scaling, LB)
Optimize costs	DIY (Spot, RIs, automation)	DIY (Spot, RIs, automation)	DIY (automation)
Security	DIY (IAM, VPC, KMS)	DIY (IAM, VPC, KMS)	DIY (IAM, VPC, KMS)

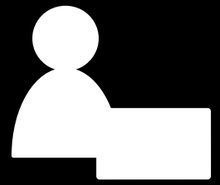
Option 3: go fully managed with Amazon SageMaker



Collect and prepare training data



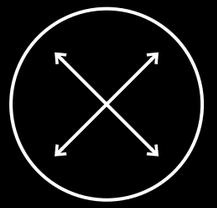
Choose and optimize your ML algorithm



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

intuit

NFL

tinder

KIA

Liberty Mutual.

CONVOY

SIEMENS

DOW JONES

Celgene

SONY

GE Healthcare

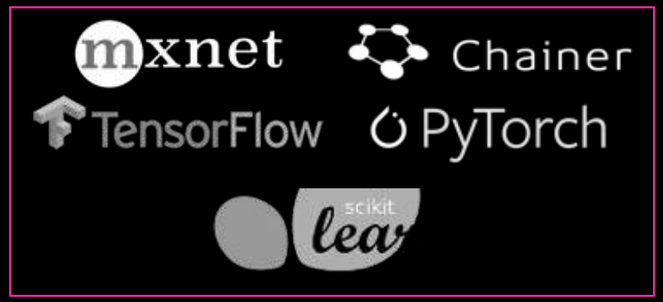
Model options on Amazon SageMaker

Training code

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

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

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(…)
```

Score card

	EC2	ECS / EKS	Fargate for ECS	SageMaker
Infrastructure effort	Lots	Some (Docker tools)	None	None
ML setup effort	Some (DL AMI)	Some (DL containers)	Some (DIY containers)	Minimal
CI/CD integration	No change	No change	No change	Some (SDK, Step Functions)
Build models	DIY	DIY	DIY	17 built-in algorithms
Train models (at scale)	DIY	DIY (Docker tools)	Not compelling	2 LOCs
Deploy models (at scale)	DIY (model servers)	DIY (Docker tools)	DIY (Docker tools)	1 LOCs
Scale/HA inference	DIY (Auto Scaling, LB)	DIY (Services, pods, etc.)	DIY (Auto Scaling, LB)	Built-in
Optimize costs	DIY (Spot, RIs, automation)	DIY (Spot, RIs, automation)	DIY (automation)	On-demand training, Auto Scaling for inference
Security	DIY (IAM, VPC, KMS)	DIY (IAM, VPC, KMS)	DIY (IAM, VPC, KMS)	API parameters

Score card

	EC2	ECS / EKS	Fargate for ECS	SageMaker
Infrastructure effort	Lots	Some (Docker tools)	None	None
ML setup effort	Some (DL AMI)	Some (DL containers)	Some (DIY containers)	Minimal
CI/CD integration	No change	No change	No change	Some (SDK, Step Functions)
Build models	DIY	DIY	DIY	17 built-in algorithms
Train models (at scale)	DIY	DIY (Docker tools)	Not compelling	2 LOCs
Deploy models (at scale)	DIY (model servers)	DIY (Docker tools)	DIY (Docker tools)	1 LOCs
Scale/HA inference	DIY (Auto Scaling, LB)	DIY (Services, pods, etc.)	DIY (Auto Scaling, LB)	Built-in
Optimize costs	DIY (Spot, RIs, automation)	DIY (Spot, RIs, automation)	DIY (automation)	On-demand training, Auto Scaling for inference
Security	DIY (IAM, VPC, KMS)	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: Docker isn't an ML platform.	Same as ECS/EKS, just less infra work. Inference only IMHO.	Learn it in a few hours, forget about servers, focus 100% on ML, enjoy goodies like pipe mode, distributed training, HPO, inference pipelines and more.

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 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 SageMaker, deploy on ECS/EKS… or vice versa
  - Write your own story!

Getting started

http://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 — SageMaker notebooks

© 2019, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Merci!

Julien Simon
Global Evangelist, AI & Machine Learning, AWS
@julsimon