Scale Machine Learning from zero to millions of users

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

Rationale

How to train ML models and deploy them in production, from humble beginnings to world domination

Try to take reasonable and justified steps

Longer, more opinionated version: https://medium.com/@julsimon/scaling-machine-learning-from-0-to-millions-of-users-part-1-a2d36a5e849

Day 1: one user (you)

Breaking out of the sandbox

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 behaviour, to run A/B tests, etc.
- You’ve embedded the model in your business application.
- You’ve deployed everything to a single 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

Week 1

A few instances and models later…

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

Use AWS-maintained tools
- Deep Learning Amazon Machine Images
- Deep Learning containers

Dockerize

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

AWS Deep Learning AMIs and Containers

Optimized environments on Amazon Linux or Ubuntu

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.

Containers
For developers who want pre-installed containers for DL frameworks (TensorFlow, PyTorch, Apache MXNet)

Demo

Running an EC2 instance with the Deep Learning AMI
Connecting to Jupyter
Training with the Tensorflow Deep Learning container

And then one day…

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

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 / 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 a Docker cluster with 4 GPU instances and 2 CPU instances
Running Tensorflow training and prediction

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)

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

Modular service and APIs, going from experimentation to production

intuit · F1 · NFL · tinder · KIA · Liberty Mutual · CONVOY · SIEMENS · DOW JONES · Celgene · SONY · GE Healthcare

aws

Model options on Amazon SageMaker

Training code

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

mxnet, Chainer, TensorFlow, PyTorch, scikit learn, dmlc XGBoost

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.
    https://github.com/aws/sagemaker-python-sdk
  - Spark SDK (Python & Scala)
    https://github.com/aws/sagemaker-spark/tree/master/sagemaker-spark-sdk

- 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(…)
```

Score card

	EC2	ECS / EKS	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)	SDK: 2 LOCs
Deploy models (at scale)	DIY (model servers)	DIY (Docker tools)	SDK: 1 LOCs Kubernetes support
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/Spot training, Auto Scaling for inference
Security	DIY (IAM, VPC, KMS)	DIY (IAM, VPC, KMS)	API parameters

Score card

Flame war in 3, 2, 1…

	EC2	**ECS / EKS**	**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)	SDK: 1 LOCs Kubernetes support
Scale/HA inference	DIY (Auto Scaling, LB)	DIY (Services, pods, etc.)	Built-in
Optimize costs	DIY (Spot, RIs, automation)	DIY (Spot, RIs, automation)	Spot training, 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 if you're able and willing to integrate with the Docker/OSS ecosystem. If not, I'd think twice: Docker isn't an ML platform.	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 »
  - 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 if it makes sense
  - Train on SageMaker, deploy on ECS/EKS… or vice versa
  - Write your own story!

Getting started

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://sagemaker.readthedocs.io/en/stable/amazon_sagemaker_operators_for_kubernetes.html

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

Thank you!

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