DATASCIENCEGO
Conference 2019

Deploying ML Models from Scratch

Ido Shlomo - Senior Data Science Manager

BlueVine

About BlueVine

- Fintech startup up based in Redwood City, CA
- Provides working capital (loans) to small & medium sized businesses
- Over $2 BN funded to date
- Data Science challenges:
  - Consume many different semi structured / unstructured data sources
  - Deal with noisy / weak signals
  - Make decisions fast
  - Build models that are stable and accurate

About Me

- Data Science Manager @ BlueVine
- Lead BlueVine's DS team in Redwood City, CA (total of ~20 people across RWC & TLV)
- Team focus:
  - NLP & text mining
  - Anomaly detection
  - Probabilistic ML
  - Response modeling
- Personal interests: Unstructured data and DS Infrastructure.

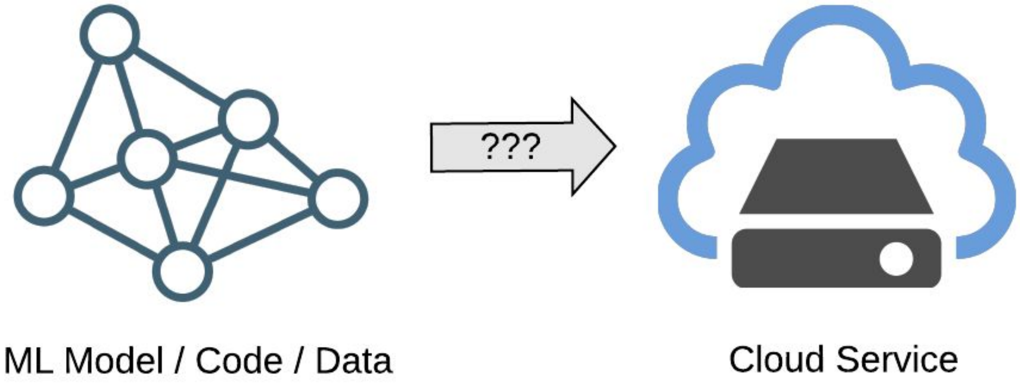
Code & Slides

- Git Repo: https://github.com/ido-sh
- Slides: repo > public_presentations > dsgo_sagemaker_2019.pdf
- Code: repo > public_presentations > sagemaker_tutorial

The Issue

- In a Business context:
  Be able to "deliver" something

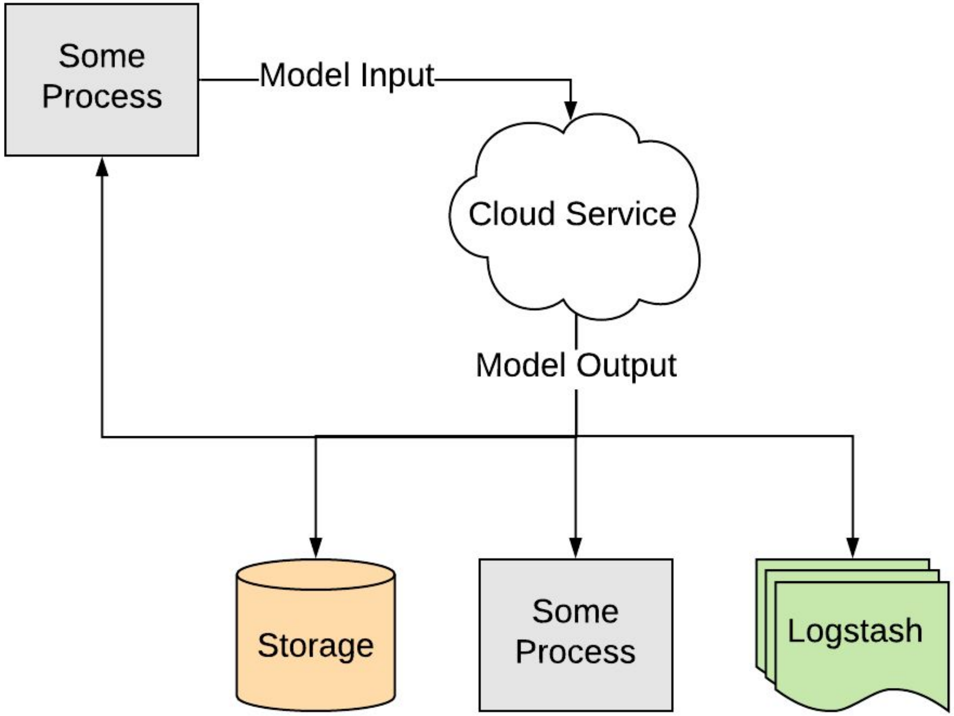
- In Research context:
  Free it from constraints of a local machine



The Desired End Result

Cloud Service:

- Model hosted in the cloud
- Always up (persistence)
- Communicates via API
- Can scale



The Starting Point

Desired Solution

Something that deploys our code as a cloud service

AND

In way that is:

- **Flexible**: Handle any type of Python code or structure
- **Simple**: Requires minimal effort to run
- **Independent**: Can run end-to-end by a Data Scientist with normal skill-set

Some leading deployment paradigms

Solution	Example Setups	Problem
Hand off everything to a team of Engineers		

Some leading deployment paradigms

Solution	Example Setups	Problem
Hand off everything to a team of Engineers	● Data Scientist sends to Engineer: ○ Code & binaries ○ Environment & tests ○ Deployment config ● Engineer: ○ Rewrites code ○ Checks tests ○ Deploys (somehow)	

Some leading deployment paradigms

Solution	Example Setups	Problem
Hand off everything to a team of Engineers	● Data Scientist sends to Engineer: ○ Code & binaries ○ Environment & tests ○ Deployment config ● Engineer: ○ Rewrites code ○ Checks tests ○ Deploys (somehow)	**Not independent**

Some leading deployment paradigms

Solution	Example Setups	Problem
Jointly manage deployment environment with Engineers		

Some leading deployment paradigms

Solution	Example Setups	Problem
Jointly manage deployment environment with Engineers	- Data Scientist: - Pushes code & binaries to repo / storage - Adheres to preset deployment config (environment, tests) - Engineer: - QA for code repo / storage - Deploys (somehow)	

Some leading deployment paradigms

<table>
<tr><th>Solution</th><th>Example Setups</th><th>Problem</th></tr>
<tr><td>Jointly manage deployment environment with Engineers</td><td>Data Scientist:
Pushes code & binaries to repo / storage
Adheres to preset deployment config (environment, tests)
Engineer:
QA for code repo / storage
Deploys (somehow)</td><td>Semi-independent, Semi-flexible</td></tr>
</table>

Some leading deployment paradigms

Solution	Example Setups	Problem
Develop on a fully managed deployment-capable platform		

Some leading deployment paradigms

Solution	Example Setups	Problem
Develop on a fully managed deployment-capable platform	• DataRobot • Alteryx • ML-Flow • Many others...	

Some leading deployment paradigms

Solution	Example Setups	Problem
Develop on a fully managed deployment-capable platform	● DataRobot ● Alteryx ● Ayasdi ● Many others...	**Not flexible**

Some leading deployment paradigms

Solution	Example Setups	Problem
Build your own docker containers and deploy them		

Some leading deployment paradigms

Solution	Example Setups	Problem

Build your own docker containers and deploy them	• Docker to bundle code and build containers
	• A Docker registry to store them
	• A Docker orchestration tool to deploy them:
	  ○ Kubernetes
	  ○ Docker Swarm
	  ○ Mesos
	  ○ Many others...	

Some leading deployment paradigms

Solution	Example Setups	Problem

Build your own docker containers and deploy them	• Docker to bundle code and build containers
	• A Docker registry to store them
	• A Docker orchestration tool to deploy them:
	  ○ Kubernetes
	  ○ Docker Swarm
	  ○ Mesos
	  ○ Many others...
		Not simple

Amazon SageMaker

Motivation: Get all the pros of docker deployment without writing any docker code

- **Flexible**: Can work with any custom Python code, environment & data files
- **Simple**: Requires relatively basic Python code to run
- **Independent**: Can build docker containers and deploy them, both for cloud training jobs and deployment of cloud services

Supports four modes: Tag, Explore, **Train** and **Deploy**

Link: https://aws.amazon.com/sagemaker/

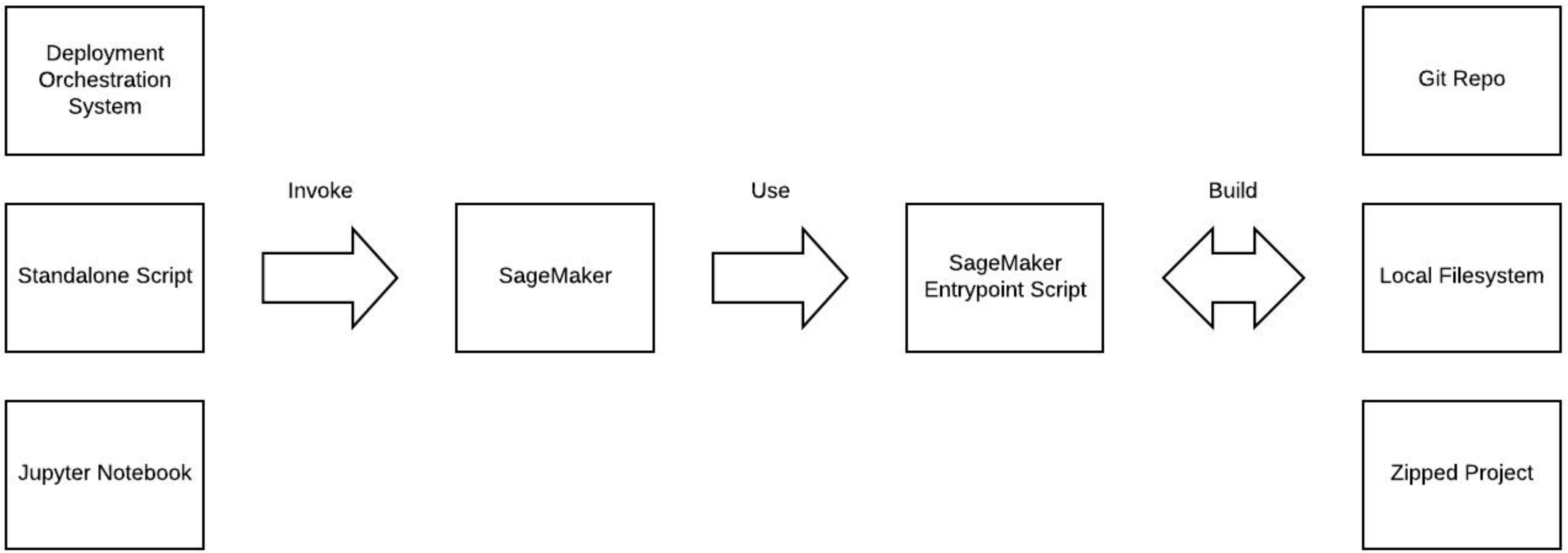
Amazon SageMaker

We’re turning **ALL** of our main Data Science deliverables into cloud services

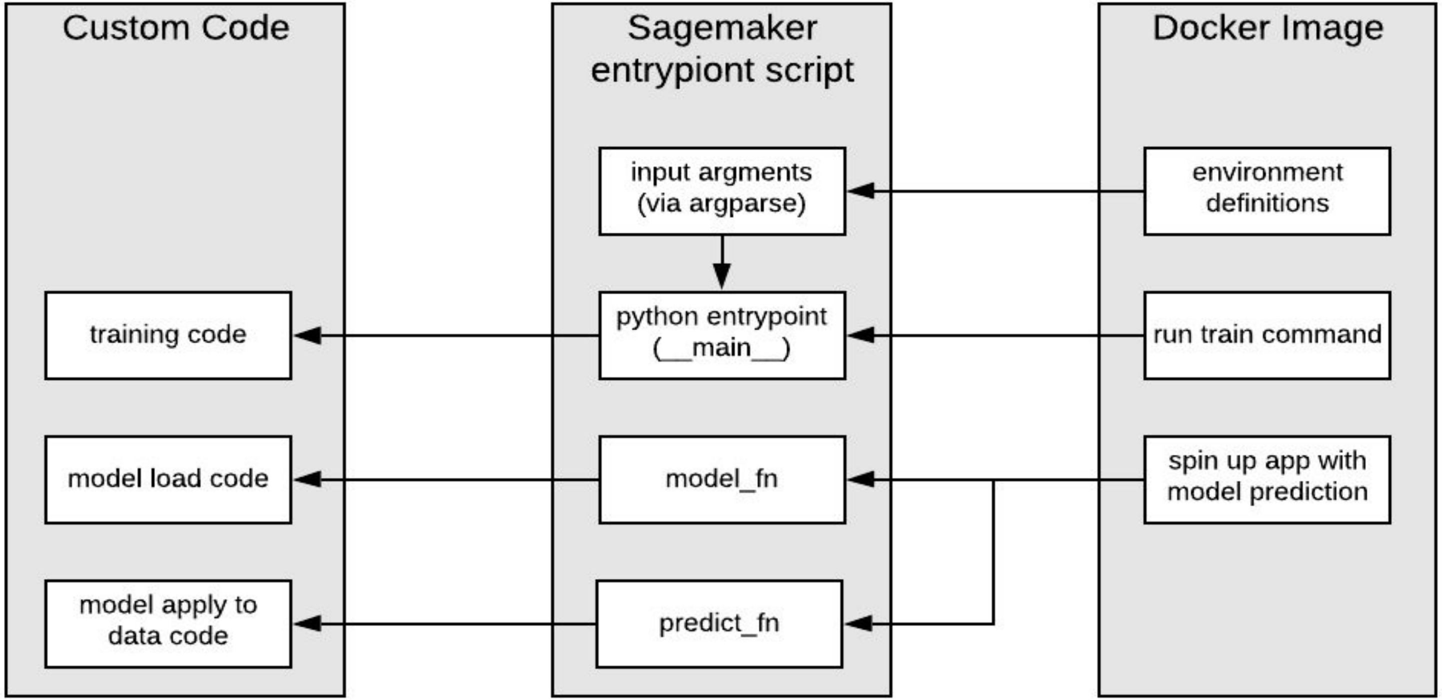
Some notable examples:

- Client Risk score
- Client Fraud score
- Industry classification
- Insights from external sources:
  - Bank data
  - Credit report data
  - Government filing data

Project Structure



Internal Connections



Example > entrypoint.py

```python
import custom_code  # importing whatever custom code you have

# THIS HANDLES TRAINING (DEFAULT SCRIPT INVOKE)
if __name__ == '__main__': ...


# THIS LOADS A TRAINED MODEL
def model_fn(model_dir): ...


# THIS APPLIES MODEL TO INPUT AND RETURNS PREDICTION
def predict_fn(input_data, model): ...
```

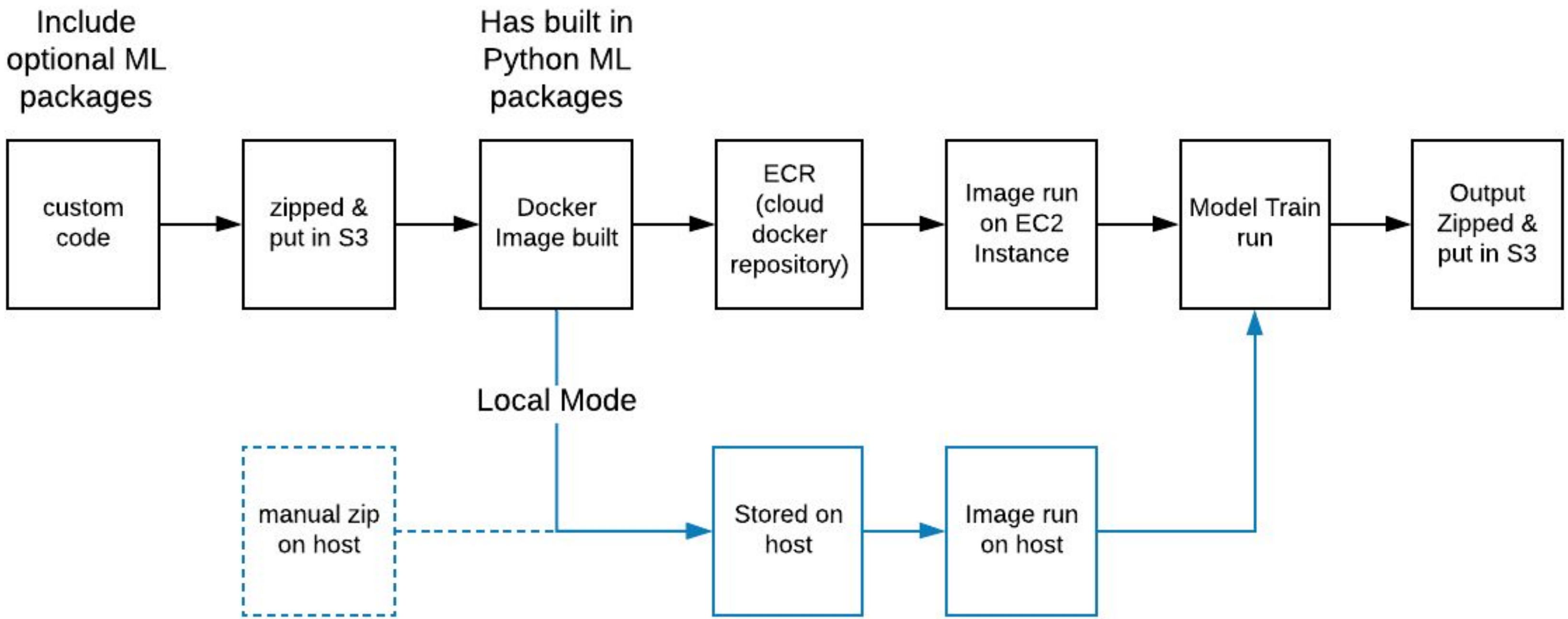
Example > entrypoint.py > __main__

```
# THIS HANDLES TRAINING (DEFAULT SCRIPT INVOKE)
if __name__ == '__main__':

    # parse environment variables
    parser = argparse.ArgumentParser()
    parser.add_argument('--output-data-dir', type=str, default=os.environ['SM_OUTPUT_DATA_DIR'])
    parser.add_argument('--model-dir', type=str, default=os.environ['SM_MODEL_DIR'])
    parser.add_argument('--train', type=str, default=os.environ['SM_CHANNEL_TRAIN'])
    args = parser.parse_args()

    # read training data from train directory
    input_files = [os.path.join(args.train, file) for file in os.listdir(args.train)]
    raw_data = [pd.read_csv(file) for file in input_files]
    train_data = pd.concat(raw_data)

    # fit model to data
    name_comparison_model = custom_code.fit_model(train_data)

    # save model as file
    custom_code.save_model(name_comparison_model, args.model_dir)
```

Example > entrypoint.py > model_fn / input_fn / predict_fn

```
# THIS LOADS A TRAINED MODEL
def model_fn(model_dir):
    mdl = custom_code.load_model(model_dir)
    return mdl


# THIS APPLIES MODEL TO INPUT AND RETURNS PREDICTION
def predict_fn(input_data, model):
    mdl_output = custom_code.use_model(input_data, model)
    return mdl_output
```

Train Run



Include optional ML packages
Has built in Python ML packages
custom code
zipped & put in S3
Docker Image built
ECR (cloud docker repository)
Image run on EC2 Instance
Model Train run
Output Zipped & put in S3
Local Mode
manual zip on host
Stored on host
Image run on host

Train Run

```
In [1]: import sagemaker
        from boto3 import Session as BotoSession
        from time import sleep

        # create sagemaker session
        boto_session = BotoSession(profile_name='default',
                                   region_name='us-east-1')
        sagemaker_session = sagemaker.Session(boto_session=boto_session)
        sagemaker_role = 'ido-sagemaker-test'
```

Train Run

Example Model:

- Dataset of ~50K book titles from Project Gutenberg (https://github.com/niderhoff/nlp-datasets)
- Train TF-IDF Vectorizer on those titles
- Build model for comparing book titles (cosine similarity)

	book_title
0	a plea for monogamy by wilfrid lay
1	jerusalem the city of herod and saladin by wal...
2	starland by robert stawell ball
3	saratoga national historical park junior range...
4	the bakhtyar nama by anonymous
5	my disillusionment in russia by emma goldman
6	the american missionary vol 37 no 2 february 1...
7	agriculture of the hidatsa indians by gilbert ...
8	the germ growers by robert potter
9	clipped wings by percy f westerman

Train Run

```python
def fit_model(train_data):
    train_text_corpus = [x for x in train_data['book_title'].tolist() if pd.notnull(x)]
    book_title_model = TfidfVectorizer()
    book_title_model = book_title_model.fit(train_text_corpus)
    return book_title_model


def use_model(input_data, model):
    input_1 = input_data['arg1']
    input_2 = input_data['arg2']
    logger.info('input 1: {}, input 2: {}'.format(input_1, input_2))
    score = 1 - cosine(model.transform([input_1]).todense(),
                       model.transform([input_2]).todense())
    return score


def load_model(model_dir):
    mdl = joblib.load(os.path.join(model_dir, "book_title_model.joblib"))
    return mdl
```

__main__

predict_fn

model_fn

Train Run

```
In [2]: #  upload training data to s3
        train_dir = 'data/train'
        project_name = 'sagemaker-dsgo-tutorial'
        train_input = sagemaker_session.upload_data(
            train_dir, key_prefix="{}/{}".format(project_name, train_dir))
        print('location in s3: {}'.format(train_input))

        location in s3: s3://sagemaker-us-east-1-[illegible]/sagemaker-dsgo-tuto
        rial/data/train
```

Train Run

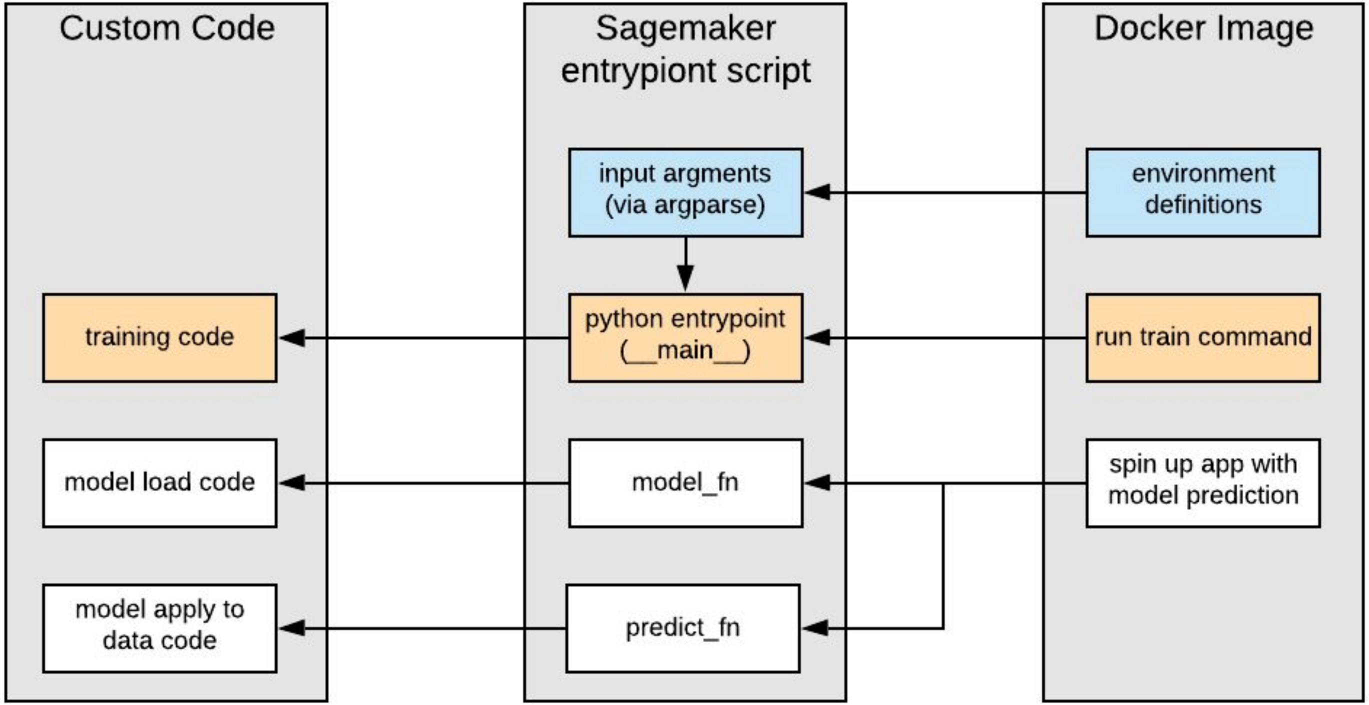
In [3]:
```
from sagemaker.sklearn.estimator import SKLearn

# config model training
cloud_model = SKLearn(
    entry_point='sagemaker_entry_point.py',
    source_dir='.',
    train_instance_type='ml.c4.xlarge',
    train_instance_count=1,
    role=sagemaker_role
)
```

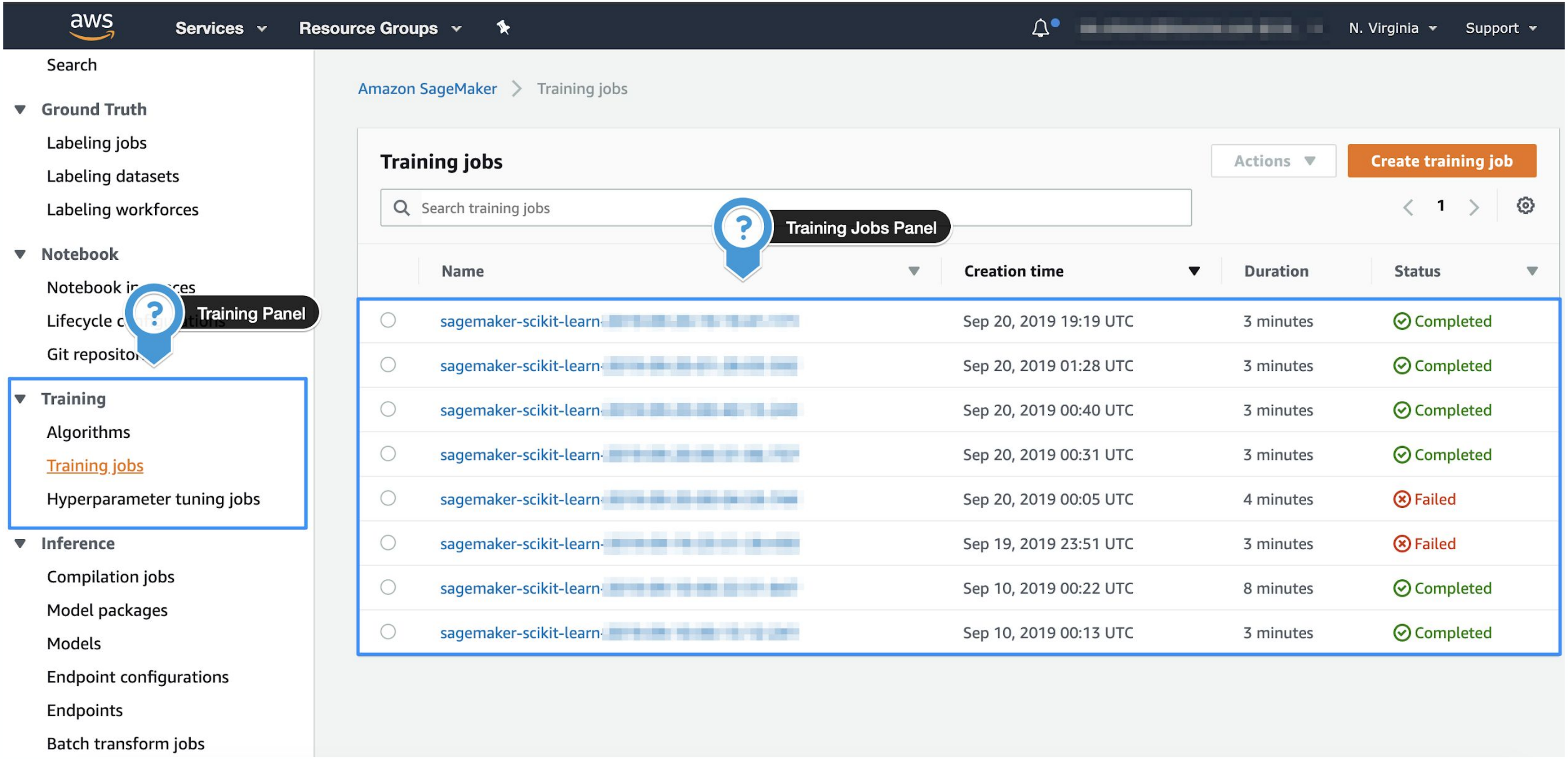
Train Run

```
In [4]: # run model training (data has to be from s3)
        cloud_model.fit({'train': train_input})
```

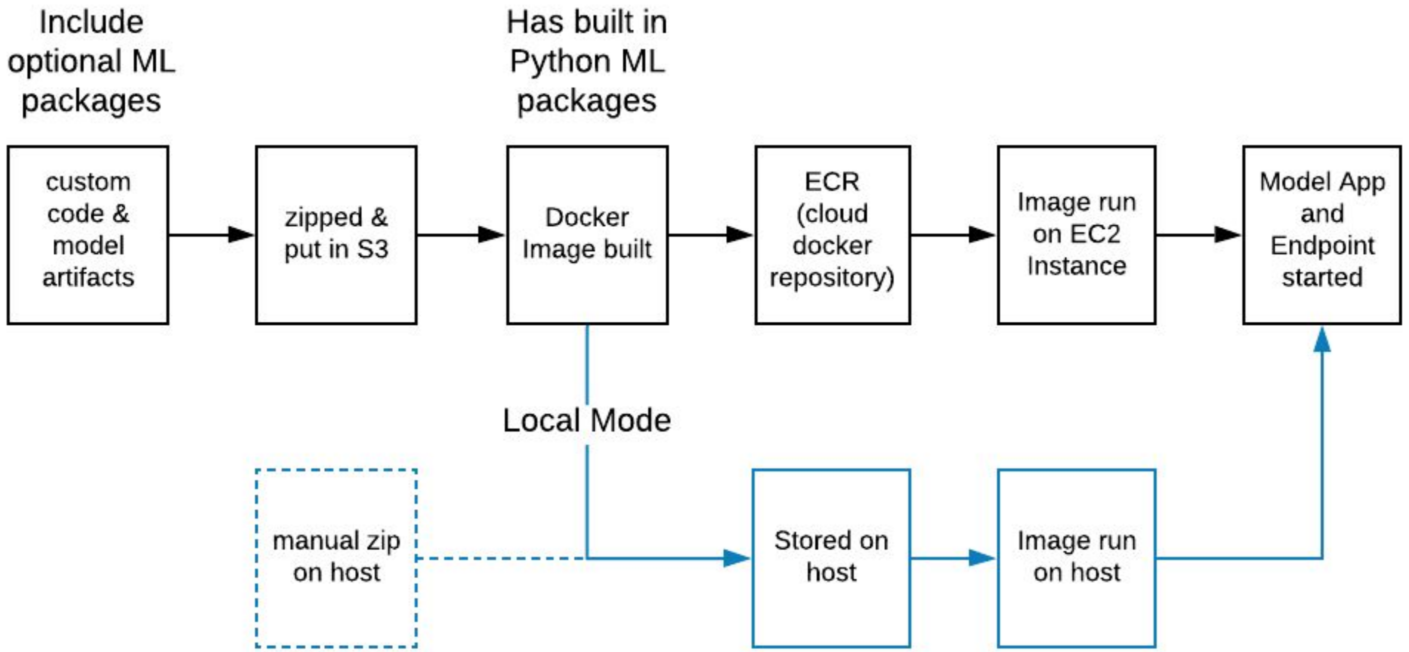
```
Training Env:

{
    "input_config_dir": "/opt/ml/input/config",
    "job_name": "sagemaker-scikit-learn-[redacted]",
    "module_dir": "s3://sagemaker-us-east-[redacted]/sagemaker-scikit
-learn-[redacted]/source/sourcedir.tar.gz",
    "user_entry_point": "sagemaker_entry_point.py",
    "is_master": true,
    "input_dir": "/opt/ml/input",
    "log_level": 20,
    "input_data_config": {
        "train": {
            "S3DistributionType": "FullyReplicated",
            "RecordWrapperType": "None",
            "TrainingInputMode": "File"
        }
    },
    "output_dir": "/opt/ml/output",
```

Internal Connections




aws
Services
Resource Groups
N. Virginia
Support
Search
Ground Truth
Labeling jobs
Labeling datasets
Labeling workforces
Notebook
Notebook instances
Lifecycle configurations
Git repositories
Training Panel
Training
Algorithms
Training jobs
Hyperparameter tuning jobs
Inference
Compilation jobs
Model packages
Models
Endpoint configurations
Endpoints
Batch transform jobs
Amazon SageMaker > Training jobs
Training jobs
Actions
Create training job
Search training jobs
Training Jobs Panel
1
Name
Creation time
Duration
Status
sagemaker-scikit-learn-
Sep 20, 2019 19:19 UTC
3 minutes
Completed
sagemaker-scikit-learn-
Sep 20, 2019 01:28 UTC
3 minutes
Completed
sagemaker-scikit-learn-
Sep 20, 2019 00:40 UTC
3 minutes
Completed
sagemaker-scikit-learn-
Sep 20, 2019 00:31 UTC
3 minutes
Completed
sagemaker-scikit-learn-
Sep 20, 2019 00:05 UTC
4 minutes
Failed
sagemaker-scikit-learn-
Sep 19, 2019 23:51 UTC
3 minutes
Failed
sagemaker-scikit-learn-
Sep 10, 2019 00:22 UTC
8 minutes
Completed
sagemaker-scikit-learn-
Sep 10, 2019 00:13 UTC
3 minutes
Completed

Deploy Run


Include optional ML packages
Has built in Python ML packages
custom code & model artifacts
zipped & put in S3
Docker Image built
ECR (cloud docker repository)
Image run on EC2 Instance
Model App and Endpoint started
Local Mode
manual zip on host
Stored on host
Image run on host

Deploy Run

In [5]:

```python
# from model trained on cloud via sagemaker
cloud_predictor = cloud_model.deploy(initial_instance_count=1,
                                     instance_type="ml.m4.xlarge")
```

```
-------------------------------------------------------------------------
-------------!
```

Deploy Run

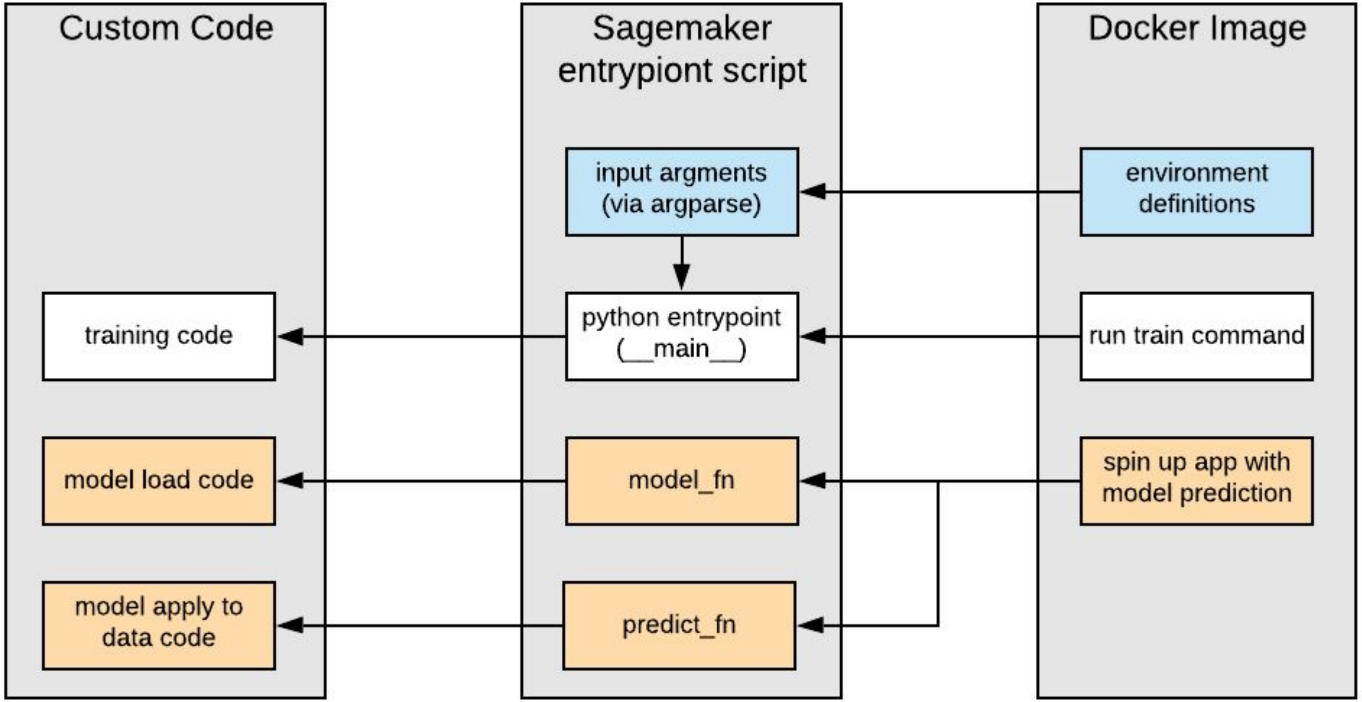
```
In [11]: book1 = 'tale of two cities'
         book2 = 'tale by two cities'
         result = cloud_predictor.predict({'arg1': book1, 'arg2': book2})

         print("\nRESULT --> {} VS {}: {}".format(book1, book2, result))
```

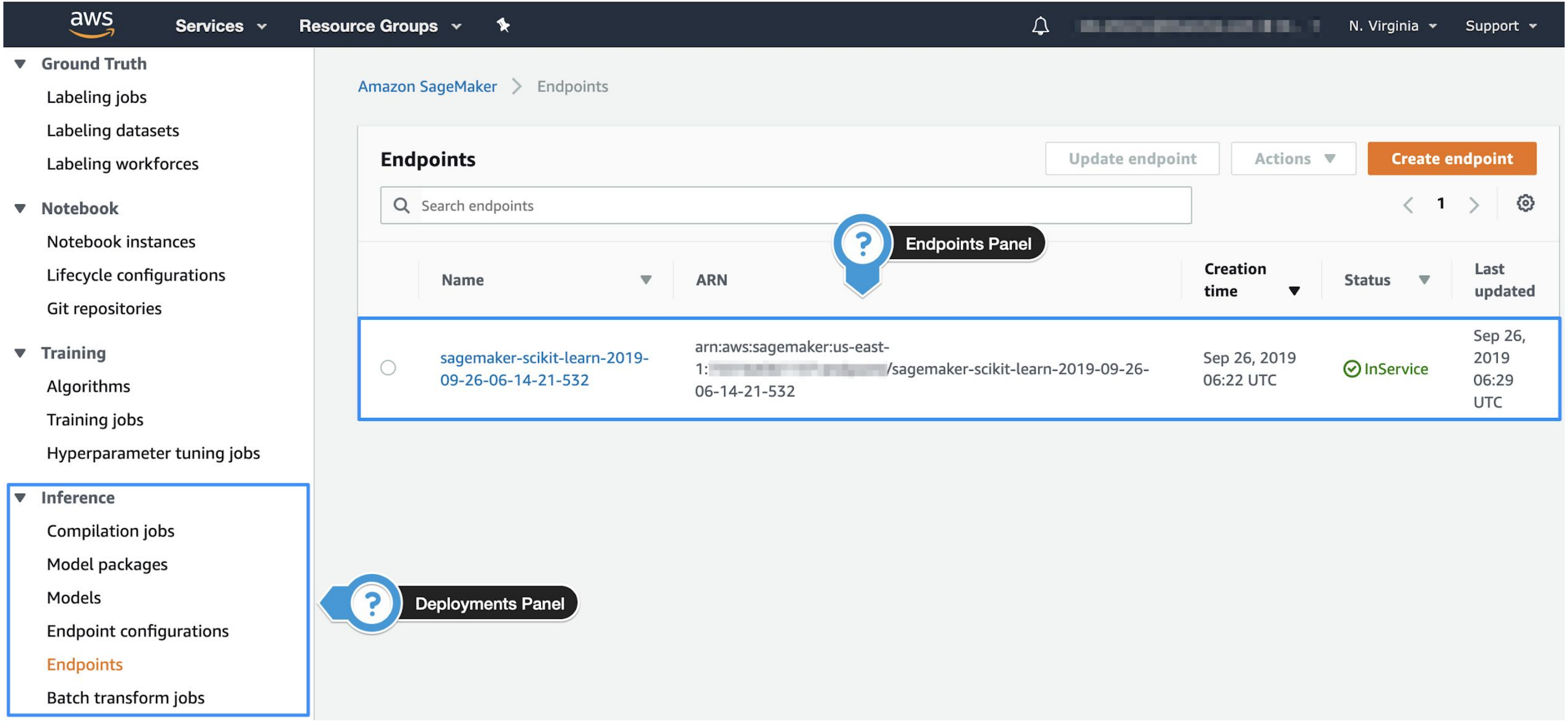
```
RESULT --> tale of two cities VS tale by two cities: 0.9797165873399724
```

```
In [12]: book1 = 'tale of two cities'
         book2 = 'tale of two towns'
         result = cloud_predictor.predict({'arg1': book1, 'arg2': book2})

         print("\nRESULT --> {} VS {}: {}".format(book1, book2, result))
```

```
RESULT --> tale of two cities VS tale of two towns: 0.57483165082027
```

Internal Connections




aws
Services
Resource Groups
N. Virginia
Support
Ground Truth
Labeling jobs
Labeling datasets
Labeling workforces
Notebook
Notebook instances
Lifecycle configurations
Git repositories
Training
Algorithms
Training jobs
Hyperparameter tuning jobs
Inference
Compilation jobs
Model packages
Models
Endpoint configurations
Endpoints
Batch transform jobs
Amazon SageMaker > Endpoints
Endpoints
Update endpoint
Actions
Create endpoint
Search endpoints
1
Endpoints Panel
Name
ARN
Creation time
Status
Last updated
sagemaker-scikit-learn-2019-09-26-06-14-21-532
arn:aws:sagemaker:us-east-1:/sagemaker-scikit-learn-2019-09-26-06-14-21-532
Sep 26, 2019 06:22 UTC
InService
Sep 26, 2019 06:29 UTC
Deployments Panel

Filter events

My custom logs

	Time (UTC +00:00)	Message
	2019-09-26	
▸	06:33:12	10.32.0.1 - - [26/Sep/2019:06:33:12 +0000] "GET /ping HTTP/1.1" 200 0 "-" "AHC/2.0"
▸	06:33:17	10.32.0.1 - - [26/Sep/2019:06:33:17 +0000] "GET /ping HTTP/1.1" 200 0 "-" "AHC/2.0"
▾	06:33:19	2019-09-26 06:33:19,207 INFO - custom_code - input 1: tale of two cities, input 2: tale by two cities

```
2019-09-26 06:33:19,207 INFO - custom_code - input 1: tale of two cities, input 2: tale by two cities
```

▾	06:33:19	10.32.0.1 - - [26/Sep/2019:06:33:19 +0000] "POST /invocations HTTP/1.1" 200 136 "-" "AHC/2.0"

```
10.32.0.1 - - [26/Sep/2019:06:33:19 +0000] "POST /invocations HTTP/1.1" 200 136 "-" "AHC/2.0"
```

▸	06:33:22	10.32.0.1 - - [26/Sep/2019:06:33:22 +0000] "GET /ping HTTP/1.1" 200 0 "-" "AHC/2.0"
▸	06:33:27	10.32.0.1 - - [26/Sep/2019:06:33:27 +0000] "GET /ping HTTP/1.1" 200 0 "-" "AHC/2.0"
▸	06:33:32	10.32.0.1 - - [26/Sep/2019:06:33:32 +0000] "GET /ping HTTP/1.1" 200 0 "-" "AHC/2.0"
▸	06:33:37	10.32.0.1 - - [26/Sep/2019:06:33:37 +0000] "GET /ping HTTP/1.1" 200 0 "-" "AHC/2.0"
▸	06:33:42	10.32.0.1 - - [26/Sep/2019:06:33:42 +0000] "GET /ping HTTP/1.1" 200 0 "-" "AHC/2.0"
▸	06:33:47	10.32.0.1 - - [26/Sep/2019:06:33:47 +0000] "GET /ping HTTP/1.1" 200 0 "-" "AHC/2.0"
▸	06:33:52	10.32.0.1 - - [26/Sep/2019:06:33:52 +0000] "GET /ping HTTP/1.1" 200 0 "-" "AHC/2.0"

Other Train / Deploy modes

Local Mode

- Uses docker on local machine instead of on EC2 instance in the cloud
- Train: Trained on local machine
- Deploy: Deploy an endpoint on local machine
- Does **NOT** mean offline
- Useful for debugging (avoids spin up latency & cloud computing costs)

External Model Deployment: Run deployment cycle using model trained outside SageMaker.

[Additional code examples in my repo]

Conclusion

Key Takeaways:

- To make your models “actionable”, you need to be able to deploy them
- Having a flexible, simple and independent deployment mechanism is hugely empowering
- Amazon SageMaker is one such mechanism

Some SageMaker caveats:

- Not the easiest to debug
- Local mode is not 100% offline
- Vendor lock in (Amazon)

Code & Slides

- Git Repo: https://github.com/ido-sh
- Slides: repo > public_presentations > dsgo_sagemaker_2019.pdf
- Code: repo > public_presentations > sagemaker_demo

BlueVine

DATASCIENCEGO
Conference 2019

We’re Hiring!!!

https://jobs.lever.co/bluevine

- Junior Data Scientist
- Data Scientist
- Senior Data Scientist

Contact Me

- Git: https://github.com/ido-sh
- Email: ido.shlomo@bluevine.com

Thanks!