O'REILLY®
TensorFlow World

PRESENTED WITH TensorFlow

Advanced model deployments with TensorFlow Serving

Hannes Hapke
@hanneshapke

Most models don't get deployed.

ginablaber
@ginablaber

The story of enterprise Machine Learning: “It took me 3 weeks to develop the model. It’s been >11 months, and it’s still not deployed.” @DineshNirmalIBM #StrataData #strataconf

10:19 AM · Mar 7, 2018 · TweetDeck

Hi, I’m Hannes.

An inefficient model deployment

```python
import json
from flask import Flask
from keras.models import load_model
from utils import preprocess

model = load_model('model.h5')
app = Flask(__name__)

@app.route('/classify', methods=['POST'])
def classify():
    review = request.form["review"]
    preprocessed_review = preprocess(review)
    prediction = model.predict_classes([preprocessed_review])[0]
    return json.dumps({"score": int(prediction)})
```

Simple Deployments

Why Flask is insufficient

- No consistent APIs
- No consistent payloads
- No model versioning
- No mini-batching support
- Inefficient for large models

```
@app.route('/classify', methods=['POST'])
def classify():
    review = request.form["review"]
    preprocessed_review = preprocess(review)
    prediction = model.predict_classes(
        [preprocessed_review])[0]
    return json.dumps({"score": int(prediction)})
```

Image: Martijn Baudoin, Unsplash

TensorFlow Serving

TensorFlow Serving

Production ready Model Serving

- Part of the TensorFlow Extended Ecosystem
- Used internally at Google
- Highly scalable model serving solution
- Works well for large models up to 2GB

TensorFlow 2.0 ready! *

* With small exceptions

Deploy your models in 90s ...

Export your Model

TensorFlow 2.0 Export

- Consistent model export
- Using Protobuf format
- Export of graphs and estimators possible

```
import tensorflow as tf

tf.saved_model.save(
    model,
    export_dir="/tmp/saved_model",
    signatures=None
)
```

Export your Model

- Exported model as Protobuf (Saved_model.pb)
- Variables and checkpoints
- Assets contains additional files, e.g. vocabularies

```
$ tree saved_models/
saved_models/
└── 1555875926
    ├── assets
    │   └── saved_model.json
    ├── saved_model.pb
    └── variables
        ├── checkpoint
        ├── variables.data-00000-of-00001
        └── variables.index
```

TensorFlow Serving

Minimal installation

- Docker images are available for CPU and GPU hardware
- gRPC and REST endpoints

```
$ docker pull tensorflow/serving

$ docker run -p 8500:8500 \
             -p 8501:8501 \
           --mount type=bind,\
             source=saved_models/,\
             target=/models/my_model \
             -e MODEL_NAME=my_model
             -t tensorflow/serving

$ docker run ...
             -t tensorflow/serving:latest-gpu
```

```
2019-08-22 00:59:24.981442: I tensorflow_serving/model_servers/server.cc:82] Building single
TensorFlow model file config:  model_name: embedding_model model_base_path:
/home/caravel/models/embedding_model/
2019-08-22 00:59:24.984138: I tensorflow_serving/model_servers/server_core.cc:462] Adding/updating
models.
...
2019-08-22 00:59:38.398535: I external/org_tensorflow/tensorflow/cc/saved_model/loader.cc:311]
SavedModel load for tags { serve }; Status: success. Took 13307659 microseconds.
2019-08-22 00:59:38.398722: I tensorflow_serving/servables/tensorflow/saved_model_warmup.cc:103] No
warmup data file found at
/home/caravel/models/embedding_model/1565651848/assets.extra/tf_serving_warmup_requests
2019-08-22 00:59:38.399078: I tensorflow_serving/core/loader_harness.cc:86] Successfully loaded
servable version {name: embedding_model version: 1565651848}
2019-08-22 00:59:38.411807: I tensorflow_serving/model_servers/server.cc:324] Running gRPC
ModelServer at 0.0.0.0:8500 ...
[warn] getaddrinfo: address family for nodename not supported
2019-08-22 00:59:38.420971: I tensorflow_serving/model_servers/server.cc:344] Exporting HTTP/REST
API at:localhost:8501 ...
[evhttp_server.cc : 239] RAW: Entering the event loop ...
```

```
2019-08-22 00:59:24.981442: I tensorflow_serving/model_servers/server.cc:82] Building single
TensorFlow model file config:  model_name: embedding_model model_base_path:
/home/caravel/models/embedding_model/
2019-08-22 00:59:24.984138: I tensorflow_serving/model_servers/server_core.cc:462] Adding/updating
models.
...
2019-08-22 00:59:38.398535: I external/org_tensorflow/tensorflow/cc/saved_model/loader.cc:311]
SavedModel load for tags { serve }; Status: success. Took 13307659 microseconds.
2019-08-22 00:59:38.398722: I tensorflow_serving/servables/tensorflow/saved_model_warmup.cc:103] No
warmup data file found at
/home/caravel/models/embedding_model/1565651848/assets.extra/tf_serving_warmup_requests
2019-08-22 00:59:38.399078: I tensorflow_serving/core/loader_harness.cc:86] Successfully loaded
servable version {name: embedding_model version: 1565651848}
2019-08-22 00:59:38.411807: I tensorflow_serving/model_servers/server.cc:324] Running gRPC
ModelServer at 0.0.0.0:8500 ...
[warn] getaddrinfo: address family for nodename not supported
2019-08-22 00:59:38.420971: I tensorflow_serving/model_servers/server.cc:344] Exporting HTTP/REST
API at:localhost:8501 ...
[evhttp_server.cc : 239] RAW: Entering the event loop ...
```

How to perform predictions?

Preform Model Predictions

Support for gRPC and REST

- TensorFlow Serving supports
  - Remote Procedure Protocol (gRPC)
  - Representational State Transfer (REST)
- Consistent API structures
- Server supports both standards simultaneously
- Default ports:
  - RPC: 8500
  - REST: 8501

Predictions via REST

- Standard HTTP Post requests
- Response is a JSON body with the prediction
- Request from the default or specific model

Default url structure

```
http://{HOST}:{PORT}/v1/models/{MODEL_NAME}
```

Specify model versions

```
http://{HOST}:{PORT}/v1/models/{MODEL_NAME}
  [/versions/{MODEL_VERSION}]:predict
```

```python
import json
from requests import HTTPSession

def rest_request(text, url=None):
    """Example inference of a text classification"""
    if url is None:
        url = 'http://localhost:8501/v1/models/my_model:predict'
    payload = json.dumps({"instances": [text]})
    response = http.request('post', url, payload)
    return response

>> rs = rest_request(text="This is a great movie")
>> print(rs.json())

{'predictions': [[0.389679104, 0.610320866]]}
```

Predictions via gRPC

More sophisticated client-server connections

- Prediction data needs to be converted to the Protobuf format
- Request types have designated types, e.g. float, int, bytes
- Payloads need to be converted to base64
- Connect to the server via gRPC stubs

gRPC vs REST

When to use which API standard

- REST is easy to implement and to debug
- RPC is more network efficient, smaller payloads
- RPC can provide much faster inferences!
- RPC can provide more prediction functionality (more later)

Use Cases

Data Science and DevOps Deployment Cycles

“Remote” Models

Load models from cloud buckets

- Eases deployment workflows for data scientists
- AWS, GCP, Minio supported
- Configure cloud credentials through environment variables

```
$ export AWS_ACCESS_KEY_ID=XXXXX
$ export AWS_SECRET_ACCESS_KEY=XXXXX

# (optional configuration)
$ export AWS_REGION=us-east-1
$ export S3_ENDPOINT=s3.us-east-1.amazonaws.com
$ export S3_USE_HTTPS=1
$ export S3_VERIFY_SSL=1

$ docker run -p 8500:8500 \
             -p 8501:8501 \
             -e MODEL_BASE_PATH=\
              s3://bucketname/model_path/\
             -e MODEL_NAME=my_model \
             -e AWS_ACCESS_KEY_ID=XXXXX \
             ...
             -t tensorflow/serving
```

“Remote” Models

Load models from cloud buckets

- Cost savings
  Limit the polling of the remote buckets

```
...

$ docker run -p 8500:8500 \
             -p 8501:8501 \
             -e MODEL_BASE_PATH=\
              s3://bucketname/model_path/\
             -e MODEL_NAME=my_model \
             -e AWS_ACCESS_KEY_ID=XXXXX \
             ...
             -t tensorflow/serving \
             --file_system_poll_wait_seconds=120
```

Host Multiple Models on one Server

Multi Model Configs

- TensorFlow Serving can load multiple models and their versions
- Production use: One model type per server

```
$ vim /tmp/model_config/model_config_list

model_config_list {
    config {
        name: 'my_model'
        base_path: '/models/my_model/'
    }
    config {
        name: 'another_model'
        base_path: '/models/another_model/'
    }
}
```

Multi Model Configs

- Use `--model_config_file`
- Start the TensorFlow Server with the configuration file instead of the base model

```
$ docker run -p 8500:8500 \
             -p 8501:8501 \
             --mount type=bind,\
             source=/tmp/models,\
             target=/models/my_model \
             --mount type=bind,\
             source=/tmp/model_config,\
             target=/models/model_config \
             -t tensorflow/serving \
             --model_config_file=\
             /models/model_config
```

Model A/B Testing

Model A/B Testing

Compare and test different model versions

- TensorFlow Serving doesn't provide server-side A/B selection
- Istio is a better candidate to route inference traffic
- Simple A/B testing is possible with TF Serving

Model Versions

Load specific model versions

- You can designate specific model versions
- Timestamps are used as the identifier (version folder name)

```
$ vim /tmp/model_config/model_config_list

model_config_list {
    config {
      name: 'my_model'
      base_path: '/models/my_model/'
      model_version_policy {
        specific {
          versions: 1556250435
          versions: 1556251435
        }
      }
    }
    ...
 }
```

Version Labels

Define canonical versions

- You can assign labels to specific versions
- Only available in connection with gRPC (Issue 1413)
- `--allow_version_labels_for_unavailable_models` required flag

```
model_config_list {
  config {
    ...
    model_version_policy {
      specific {
        versions: 1556250435
        versions: 1556251435
      }
    }
    version_labels {
      key: 'stable'
      value: 1556250435
    }
    version_labels {
      key: 'testing'
      value: 1556251435
    }
  ...
  }
}
```

Model Version Configs

- Use `--model_config_file`
- Start the TensorFlow Server with the configuration file instead of the base model

```
$ docker run -p 8500:8500 \
             -p 8501:8501 \
             --mount type=bind,\
             source=/tmp/models,\
             target=/models/my_model \
             --mount type=bind,\
             source=/tmp/model_config,\
             target=/models/model_config \
             -t tensorflow/serving \
             --model_config_file=\
             /models/model_config
```

```python
from random import random
def get_rest_url(model_name, host='localhost', port='8501',
    verb='predict', version=None):
    url = f"http://{host}:{port}/v1/models/{model_name}/"
    if version:
        url += f"versions/{version}"
        url += f":{verb}"
    return url

...
```

```python
"""
submit 10% of all request from this client to version 1
90% of the request should go to the default models
"""

threshold = 0.1
version = 1 if random() < threshold else None
url = get_rest_url(model_name='my_model', version=version)

rs = rest_request(text, url=url)
```

Model Meta Information

Model Status Information

Obtain model status before your inference

- TensorFlow Serving provides an API to obtain the status of your loaded model

```python
import json
from requests import HTTPSession

def get_model_status(model_name, host='localhost', port='8501', version=None):
    url = f"http://{host}:{port}/v1/models/{model_name}/"
    if version:
        url += f"versions/{version}"
    http = HTTPSession()
    response = http.request('get', url)
    return response
```

```
{
  'model_version_status': [
      {
          'version': '1571698198',
          'state': 'AVAILABLE',
          'status': {
              'error_code': 'OK',
              'error_message': ''
          }
      }
  ]
}
```

Model Meta Information

Obtain model meta information

- TensorFlow Serving provides an API to obtain meta information
- Very useful for model telemetry tracking
- Endpoint provides the model signatures (inputs and outputs)

```python
import json
from requests import HTTPSession

def get_model_metadata(model_name, host='localhost', port='8501', version=None):
    url = f"http://{host}:{port}/v1/models/{model_name}/"
    if version:
        url += f"versions/{version}"
    url += f"/metadata"
    http = HTTPSession()
    response = http.request('get', url)
    return response
```

```
{
   "model_spec": {
       "name": "my_model",
       "signature_name": "",
       "version": "1556583584"
   },
   "metadata": {
       "signature_def": {
           "signature_def": {
               "classification": {
                   "inputs": {
                       "inputs": {
                           "dtype": "DT_STRING",
                           "tensor_shape": {
                               ...
```

Prediction Optimizations

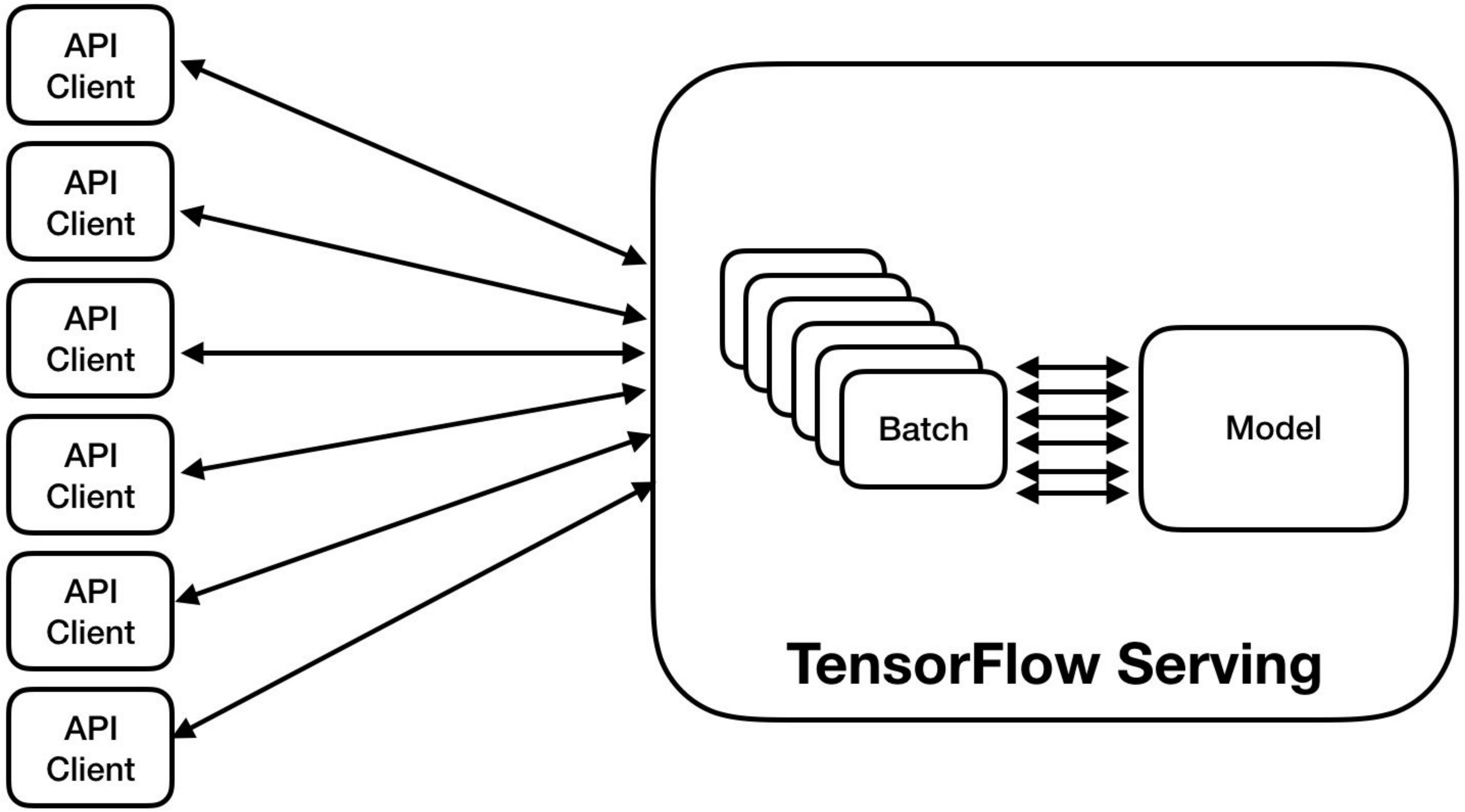
Batching Inferences

Save compute resources with batching

- Most powerful and underrated feature of TensorFlow Serving
- The server can aggregate inference requests and compute them as a block
- Efficient use of CPU or GPU hardware
- Especially relevant if models are memory-consuming

Batching Inferences

Save compute resources with batching



Batching Inferences

Batch configuration

- Max batch size
- Timeout
- Batch threads
- Batch queues

```
$ vim /tmp/batch_config/batching_parameters.txt

max_batch_size { value: 32 }
batch_timeout_micros { value: 1000 }
pad_variable_length_inputs: true
```

Batching Inferences

Batch configuration

- Configuration file can be loaded during start up
- Consider your business case when configuring batching

```
docker run -p 8501:8501 \
           --mount type=bind,\
            source=/path/to/models,\
            target=/models/my_model \
           --mount type=bind,\
            source=/tmp/batch_config/,
            target=/server_config \
           -e MODEL_NAME=my_model \
           -t tensorflow/serving \
           --enable_batching=true \
           --batching_parameters_file=\
            /tmp/batch_config/\
            batching_parameters.txt
```

There is a flag for everything!

Other Server Optimizations

TensorFlow Serving is highly customizable

`--file_system_poll_wait_seconds=1`

Polling for new models. Disabled with -1, loading once with 0

`--tensorflow_session_parallelism=0`

Threads per TensorFlow session.

`--tensorflow_intra_op_parallelism=0`

Number of cores used by TensorFlow Serving.

`--per_process_gpu_memory_fraction`

Fraction that each process occupies of the GPU memory space

```
docker run -p 8500:8500 \
           -p 8501:8501 \
           --mount type=bind,source=/path/to/models,target=/models/my_model \
           -e MODEL_NAME=my_model \
           -t tensorflow/serving \
           --tensorflow_intra_op_parallelism=4 \
           --tensorflow_inter_op_parallelism=4 \
           --file_system_poll_wait_seconds=10 \
           --tensorflow_session_parallelism=2
```

Model Optimizations

NVidia TensorRT

Optimize your inferences even further

- Optimizations of inferences based on NVidia hardware
- Reduces precision of the weights and biases
- Models need to be converted

```
$ saved_model_cli convert --dir saved_models/ \
                          --output_dir trt-savedmodel/ \
                          --tag_set serve tensorrt


$ docker run --runtime=nvidia \
             -p 8501:8501 \
             --mount type=bind,source=/path/to/trt-savedmodel/,target=/models/my_model \
             -e MODEL_NAME=my_model \
             -t tensorflow/serving:latest-gpu
```

Serving TensorFlow Lite Models

Take advantage of TFLite optimizations

- Experimental option, use with caution
- Quantization with TFLite (int, float16)
- Models need to be converted
- Not all ops are supported

Serving TFLite Models

Steps to convert your model

- Reused your exported model
- Determine your optimization goals
- Convert your model

```
import tensorflow as tf

saved_model_dir = "path_to_saved_model"
converter = \
  tf.lite.TFLiteConverter.from_saved_model(
    saved_model_dir)

converter.optimizations = [
  tf.lite.Optimize.DEFAULT
]
tflite_model = converter.convert()

with open("/tmp/model.tflite", "wb") as f:
  f.write(tflite_model)
```

Serving TFLite Models

Server configuration

- Use `use_tflite_model` flag

```
docker run -p 8501:8501 \
           --mount type=bind,\
            source=/path/to/models,\
            target=/models/my_model \
           -e MODEL_BASE_PATH=/models \
           -e MODEL_NAME=my_model \
           -t tensorflow/serving:latest \
           --use_tflite_model=true
```

Server Monitoring

Monitor your Deployments

Logging with Prometheus

- Prometheus server required
- Prometheus server can pull metrics from TensorFlow Serving
- Rest API must be enabled
- Monitoring configuration required

Prometheus Config

Logs instead of fire

- Create a Prometheus configuration file

```yaml
global:
  scrape_interval:     15s
  evaluation_interval: 15s
  external_labels:
    monitor: 'tf-serving-monitor'

scrape_configs:
  - job_name: 'prometheus'
    scrape_interval: 5s
    metrics_path: /monitoring/prometheus/metrics
    static_configs:
      - targets: ['host.docker.internal:8501']
```

Prometheus Service

Logs instead of fire

- Start your Prometheus instance

```
docker run -p 9090:9090
           -v /tmp/prometheus.yml:\
              /etc/prometheus/prometheus.yml \
           prom/prometheus
```

Log Predictions

Tell TF Serving to log predictions

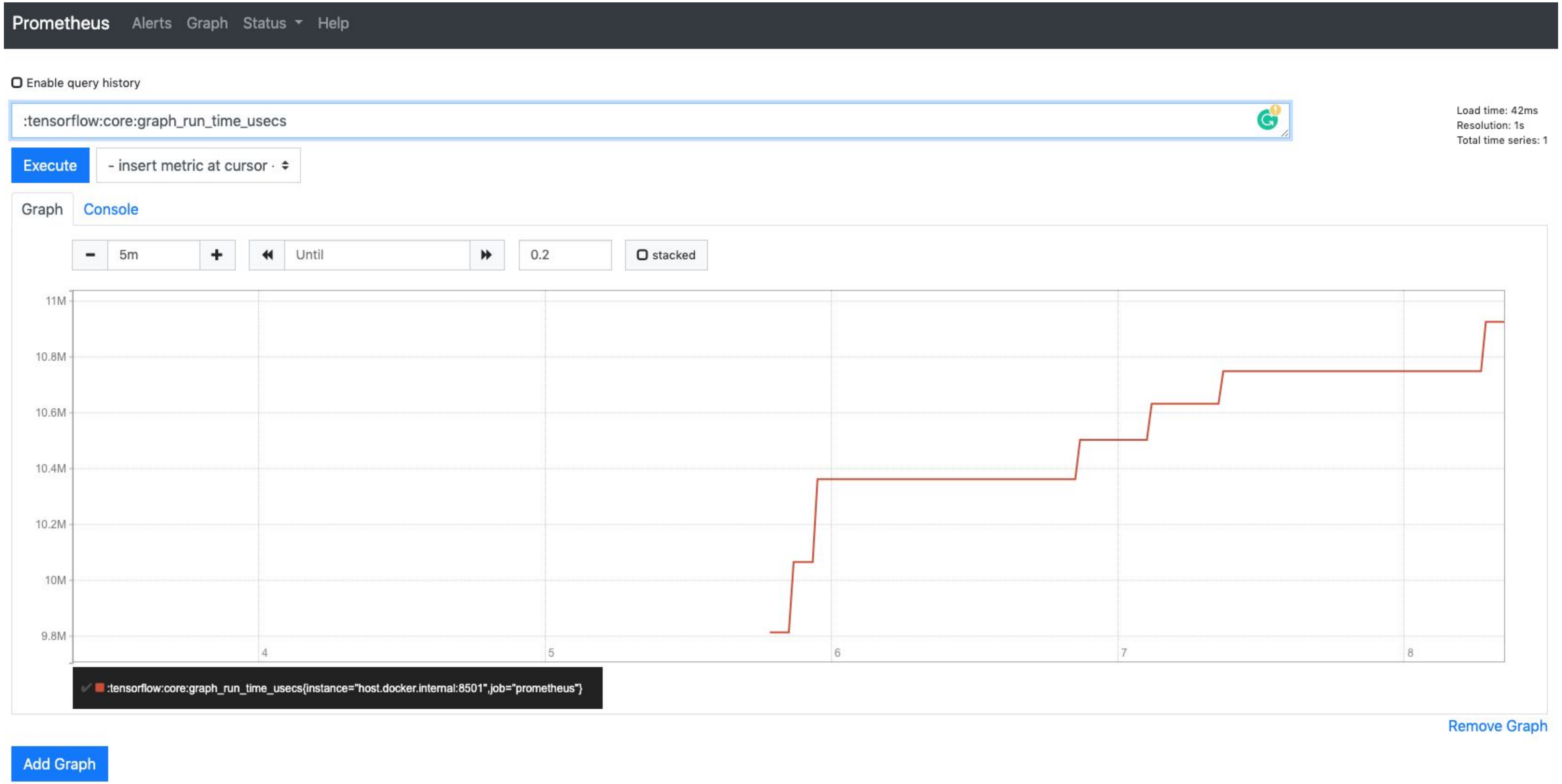
- Create a configuration file for TF Serving
- Load the additional configuration

```
$ vim monitoring_config.txt

prometheus_config {
  enable: true,
  path: "/monitoring/prometheus/metrics"
}

docker run -p 8501:8501 \
    --mount type=bind,source=`pwd`,\
      target=/models/my_model \
    --mount type=bind,source=`pwd`/configurations,\
      target=/models/model_config \
    -t tensorflow/serving \
    --monitoring_config_file=\
      /models/model_config/monitoring_config.txt
```

Monitor your Deployments



Scaling your Models

Scale with Kubernetes

Beyond a single server

- Consider Kubeflow for scalable deployment
- Easy deployment possible without Kubeflow

```yaml
apiVersion: apps/v1
kind: Deployment
spec:
  ...
  template:
    spec:
      containers:
        - args:
            - --rest_api_port=8501
            - --model_name=intent
            - --model_base_path=gs://models/my_model
          command:
            - /usr/bin/tensorflow_model_server
          env:
            - name: GOOGLE_APPLICATION_CREDENTIALS
              value: /secret/gcp/service_acc.json
          image: tensorflow/serving
```

Managed Deployments

Ease your Deployments

- All major cloud providers have model deployment offerings
  - GCP allows you to reuse SavedModel instances

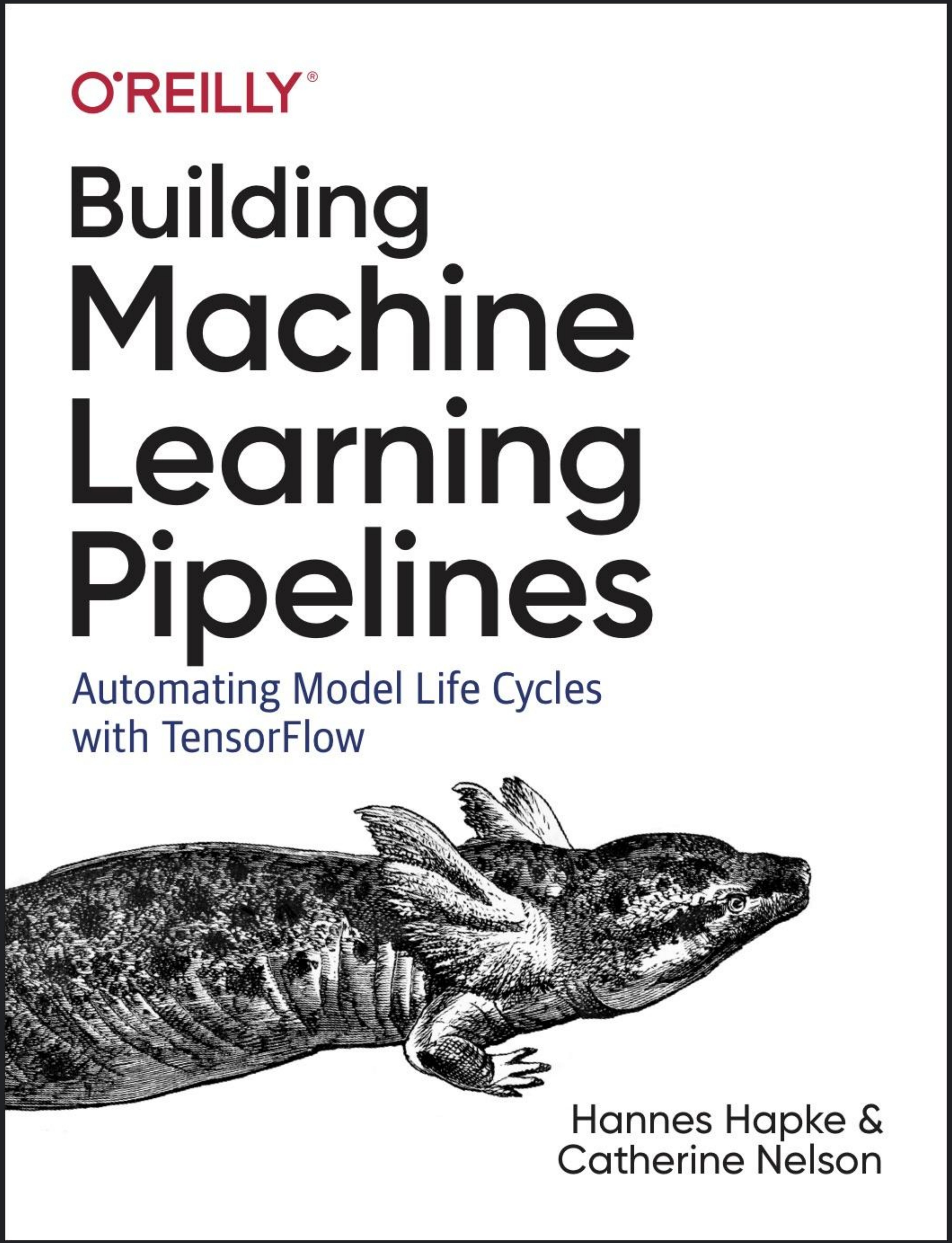
- DKube - One Convergence
  - Managed Kubeflow Service

O'Reilly Publication

Machine Learning Pipelines

- Early Release available at learning.oreilly.com
- Print Release: Summer 2020

We would like to hear your use cases of ML Pipelines!



O'REILLY TensorFlow World PRESENTED WITH TensorFlow

Thank You!

bit.ly/tf-world-tf-serving

Please rate this talk!

@hanneshapke