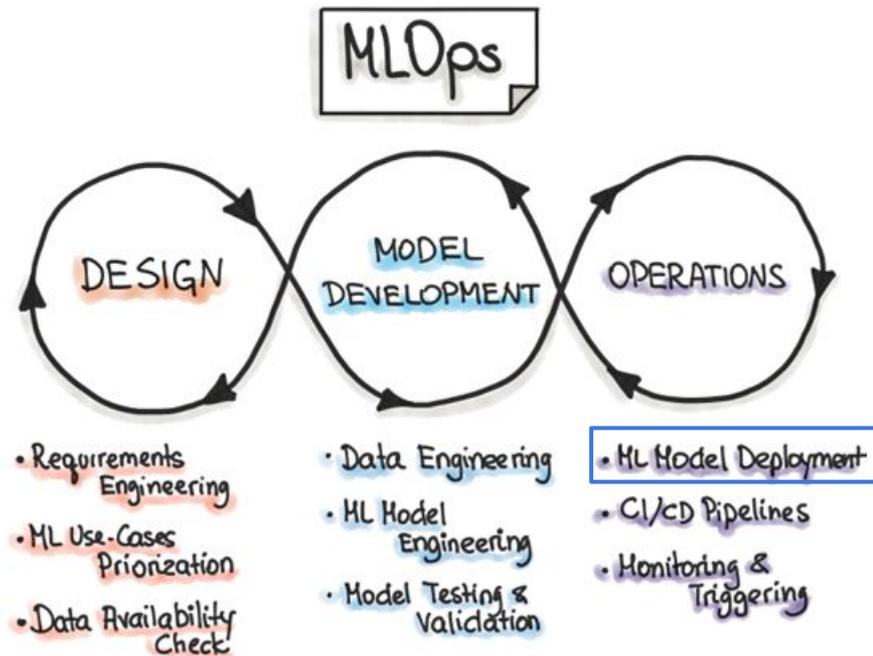

Deployment

02476 Machine Learning Operations
Nicki Skaftø Detlefsen

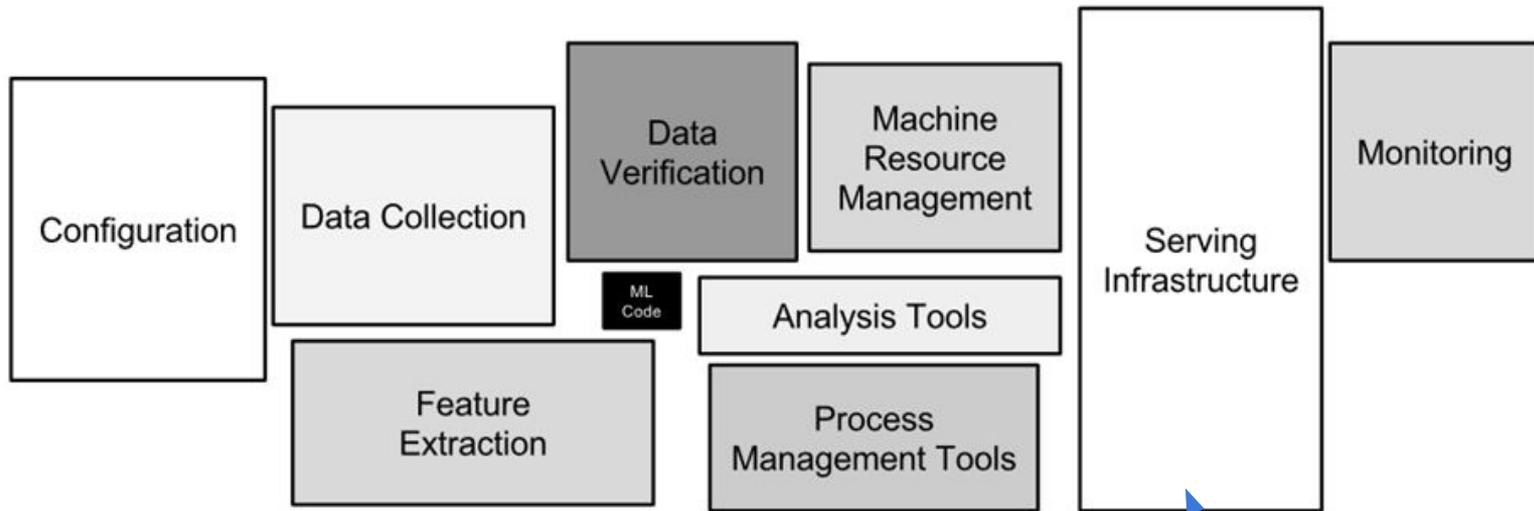
Freeing the model

In a nutshell:

Make model available to invoke easily and continuously



Easy to get started, hard to get right



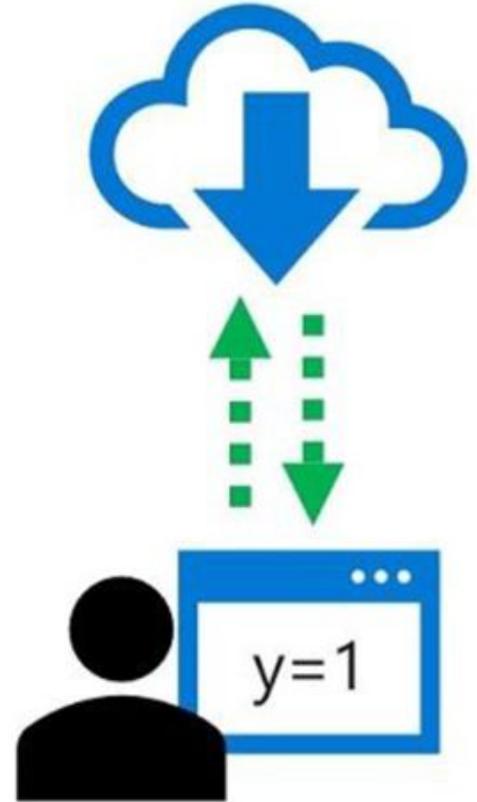
Server infrastructure depends on your use case

This is big for a reason

What to we want to deploy

In ML, *inferencing* refer to the use of a trained model to predict labels for new data on which the model has not been trained.

Around 80% of compute spend in the cloud on machine learning is spend on inference.



Production requirements

1. Portability

Models should be exportable to wide variety of environments, from c++ servers to mobile

2. Performance

We want to optimize common patterns in neural network to improve latency and throughput

Latency



VS

Throughput



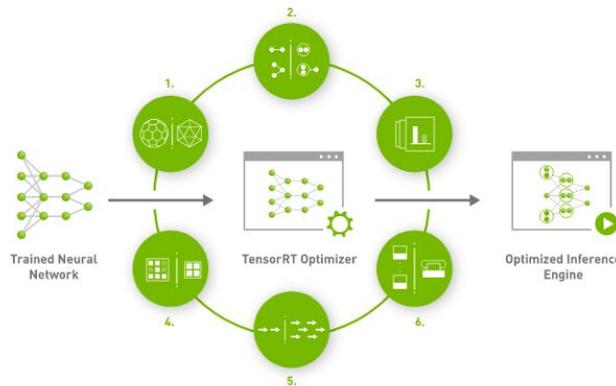
Before you deploy

Start by optimizing your model!

- Pruning
- Quantization
- Compile to low language
- Device optimizations*

to increase throughput, reduce memory and reduce energy consumption

Assume we have such model

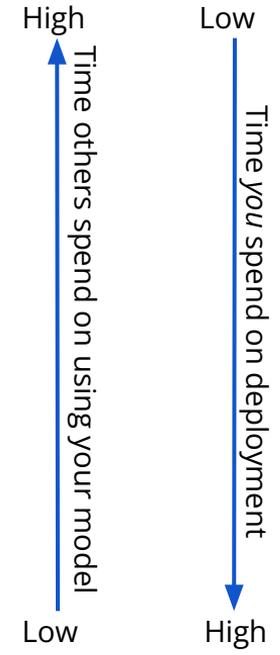


*TensorRT

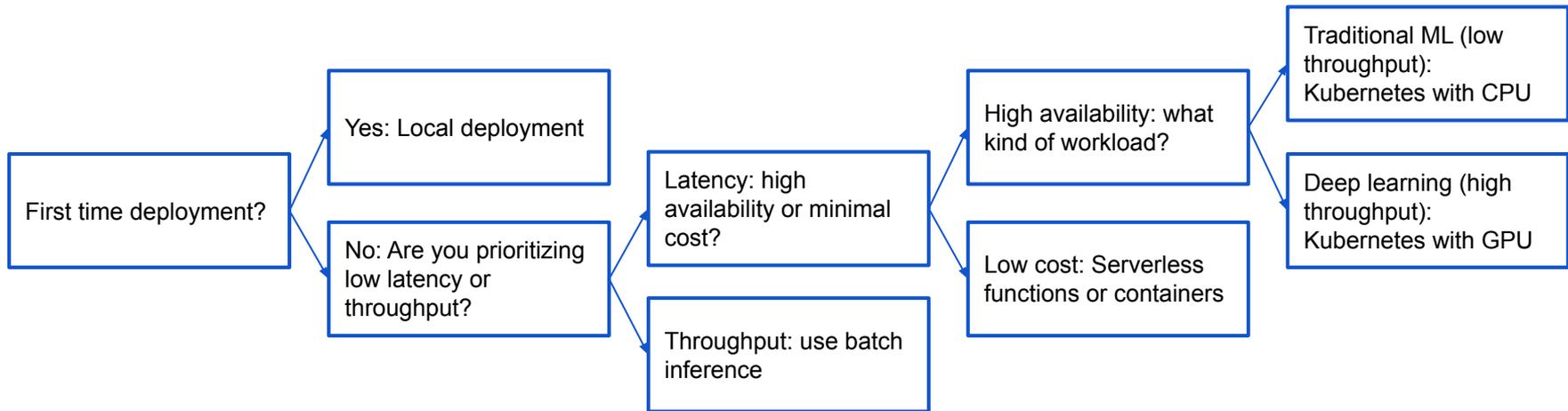
- 1. Weight & Activation Precision Calibration**
Maximizes throughput by quantizing models to INT8 while preserving accuracy
- 2. Layer & Tensor Fusion**
Optimizes use of GPU memory and bandwidth by fusing nodes in a kernel
- 3. Kernel Auto-Tuning**
Selects best data layers and algorithms based on target GPU platform
- 4. Dynamic Tensor Memory**
Minimizes memory footprint and re-uses memory for tensors efficiently
- 5. Multi-Stream Execution**
Scalable design to process multiple input streams in parallel
- 6. Time Fusion**
Optimizes recurrent neural networks over time steps with dynamically generated kernels

Many levels of deployment

1. Github repository + link to model weights
 - a. Easy to "deploy"
 - b. Pain in the *** to use
2. Deploy on local computer/cluster
 - a. Fairly easy getting up and running, just requires people can access from outside
 - b. Can be fairly easy to use
 - c. Does not scale at all
3. Deploy to cloud service
 - a. Can be a pain to setup
 - b. Easy to use and scales to ∞ (and beyond!)

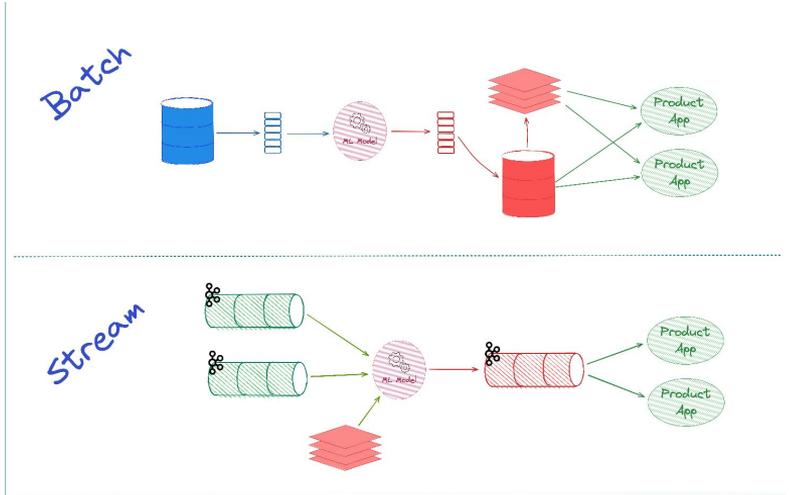


Choosing the right service

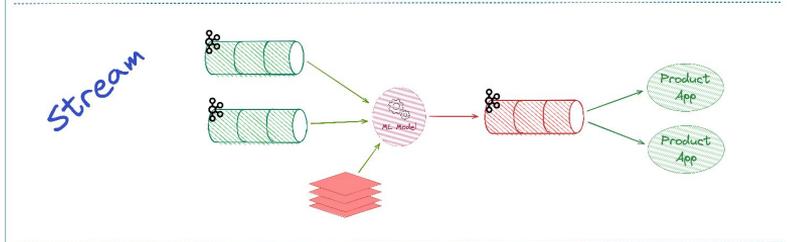


Different kind of deployments

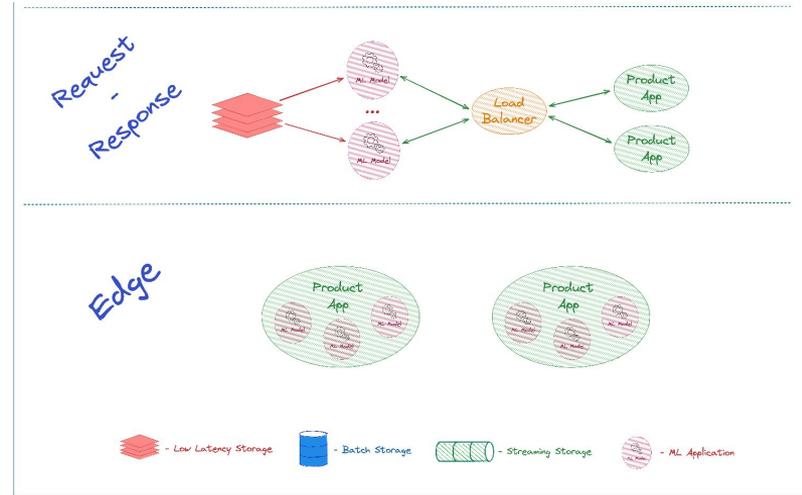
Scheduled run



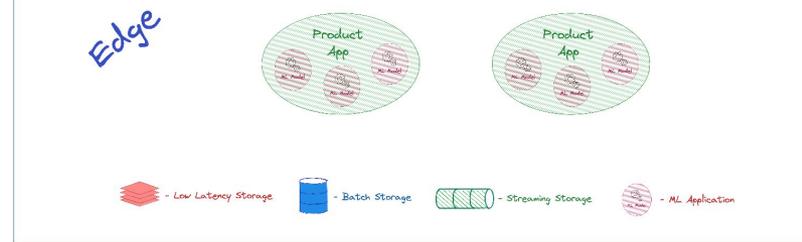
Continues run



On demand run



Embedded run

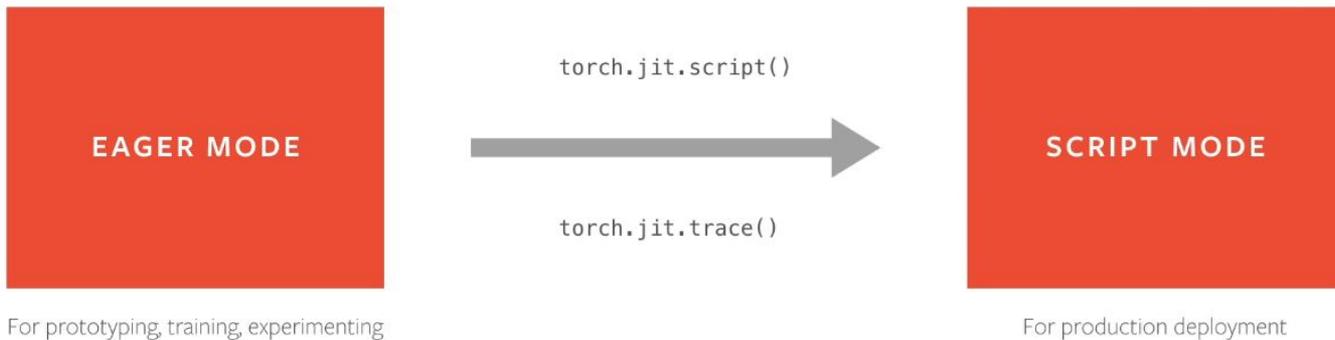


Deployment of Pytorch applications

Pytorch is a dynamic framework (uses a dynamic graph)

- Great for development
- In practise a lot of performance is lost to the JIT compiler

Solution: convert to script mode



`torch.jit.script` serialize the model, but what does that mean?

- Serialization essentially encodes all modules methods, submodules, parameters, and attributes into a byte stream
- This makes the encoded model independent of python!
- This is basically just "pickling" and "unpickling".

Other options for Pytorch

TensorRT



ONNX



ONNX

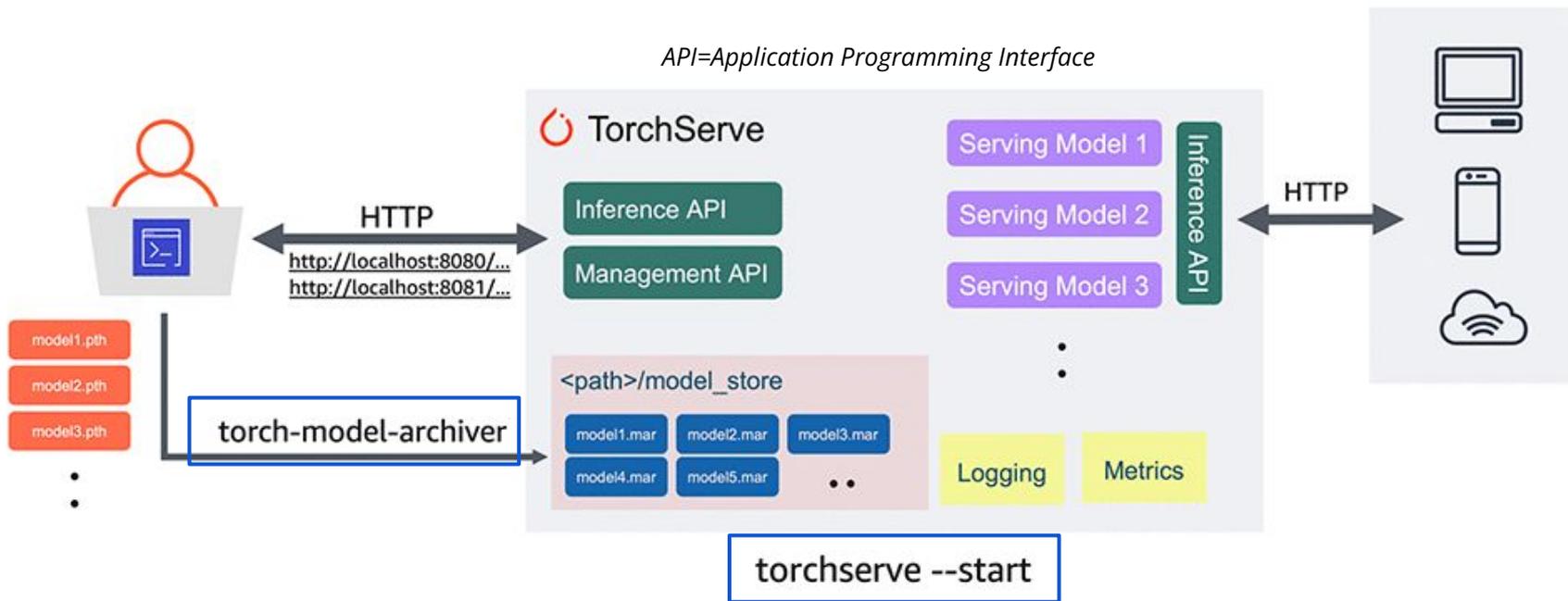
GLOW



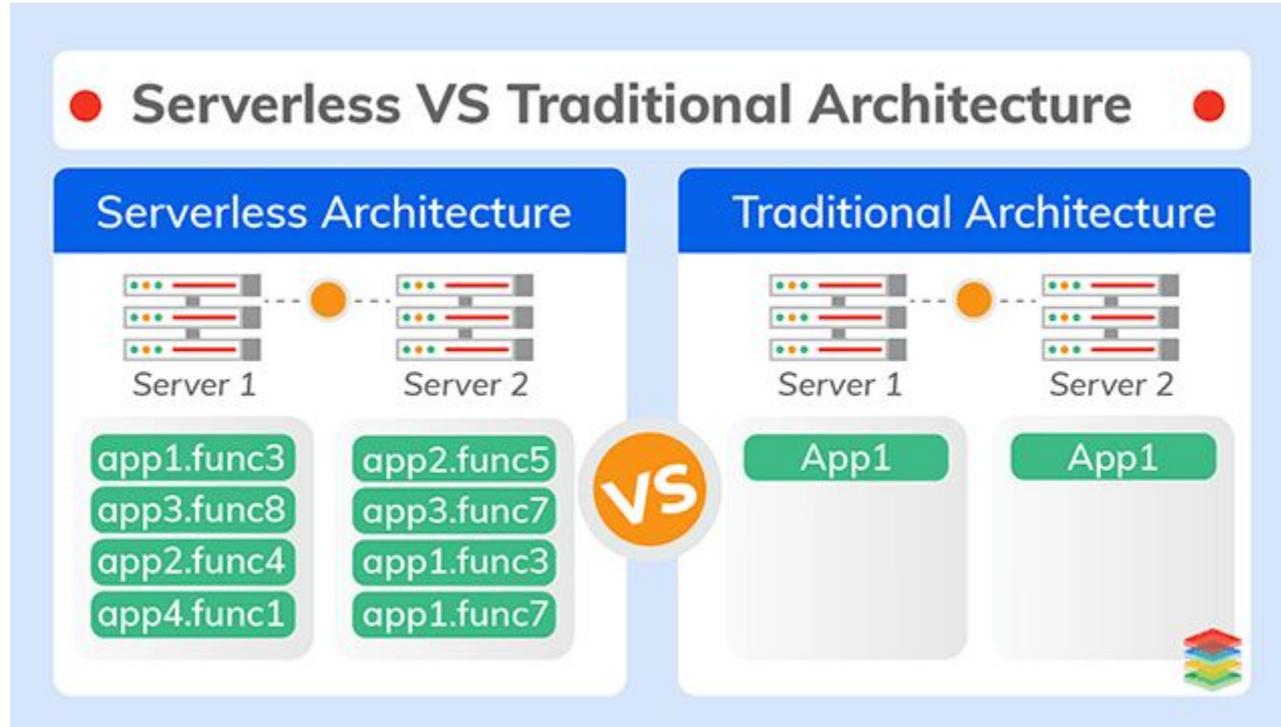
GLOW

Torchserve

You can also use [FastAPI](#) or [Flask](#)



Cloud deployment



GCP Functions

Simple two script deployment

- dependencies
- single script

Limited in expressiveness

The screenshot shows the Google Cloud Functions console interface. At the top, there are navigation options: 'Cloud Functions', 'Function details', 'EDIT', 'DELETE', and 'COPY'. Below this, the function name 'function-1' is displayed with a green checkmark, and a dropdown menu shows 'Version 11, deployed at Jan 13, 2022, 4:32:16 P...'. A horizontal menu below the function name includes 'METRICS', 'DETAILS', 'SOURCE' (which is selected), 'VARIABLES', 'TRIGGER', 'PERMISSIONS', 'LOGS', and 'TESTING'. Under the 'SOURCE' tab, the runtime is set to 'Python 3.9' and the entry point is 'knn_classifier'. A file explorer shows two files: 'main.py' and 'requirements.txt'. The main code editor displays the following Python code:

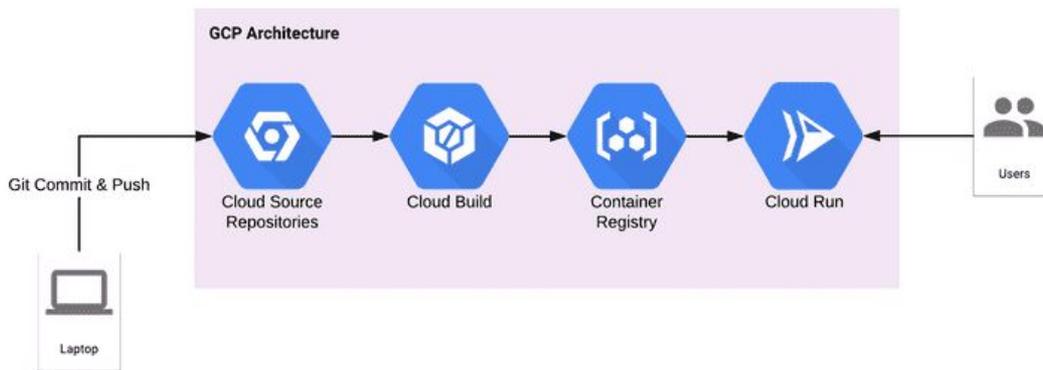
```

1 from google.cloud import storage
2 import pickle
3 client = storage.Client()
4 bucket = client.get_bucket("dtumlops")
5 blob = bucket.get_blob("model.pkl")
6 pickle_in = blob.download_as_string()
7 my_model = pickle.loads(pickle_in)
8
9
10 def knn_classifier(request):
11     """ will to stuff to your request """
12     request_json = request.get_json()
13     if request_json and 'input_data' in request_json:
14         data = request_json['input_data']
15         input_data = list(map(int, data.split(',')))
16         prediction = my_model.predict([input_data])
17         return f'Belongs to class: {prediction}'
18     else:
19         return 'No input data received'
20

```

GCP Run

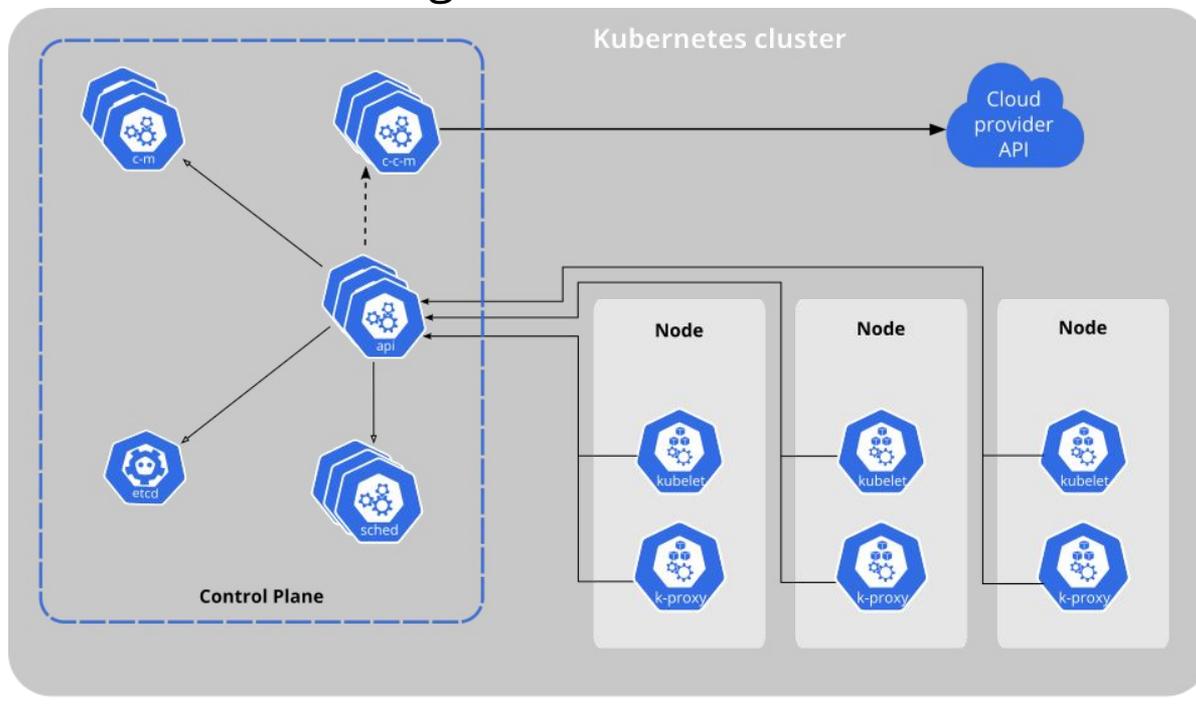
If you work with containers Run offers more expressive interface (because containers are more expressive)



Run is still serverless deployment

Kubernetes (not part of the course, yet)

If you want to be in charge of the cluster



- API server 
- Cloud controller manager (optional) 
- Controller manager 
- etcd (persistence store) 
- kubelet 
- kube-proxy 
- Scheduler 
- Control plane 
- Node 

Meme of the day

**MY COWORKERS
WATCHING ME DEPLOY A
"SMALL FIX" ON A FRIDAY**

