Accelerate and Standardize Deep Learning Inference with KFServing

Dan Sun, Bloomberg
David Goodwin, NVIDIA

KubeCon | CloudNativeCon
North America 2020
Virtual

Agenda

Accelerate Deep Learning Inference with KFServing

- Dan Sun, Bloomberg

KFServing V2 Inference Protocol

- David Goodwin, NVIDIA

Deep Learning Inference Requirements

As a data scientist or ML engineer

- I want to serve standard deep learning models, like TensorFlow or PyTorch, with minimal efforts and at scale in a unified way.
- I can bring in custom pre/post processing before and after the prediction.
- I can accelerate inference by deploying models on GPUs.
- GPUs are powerful compute resources, but deploying a single model per GPU can under-utilize GPUs. I want an easy way to serve multiple models behind a unified endpoint which can scale to hundreds or thousands of models.
- I want to autoscale based on workload and allow scale to 0 to save resources.
- I want to deploy models with zero downtime and can use multiple deployment strategies like shadow, canary, and blue/green rollouts.

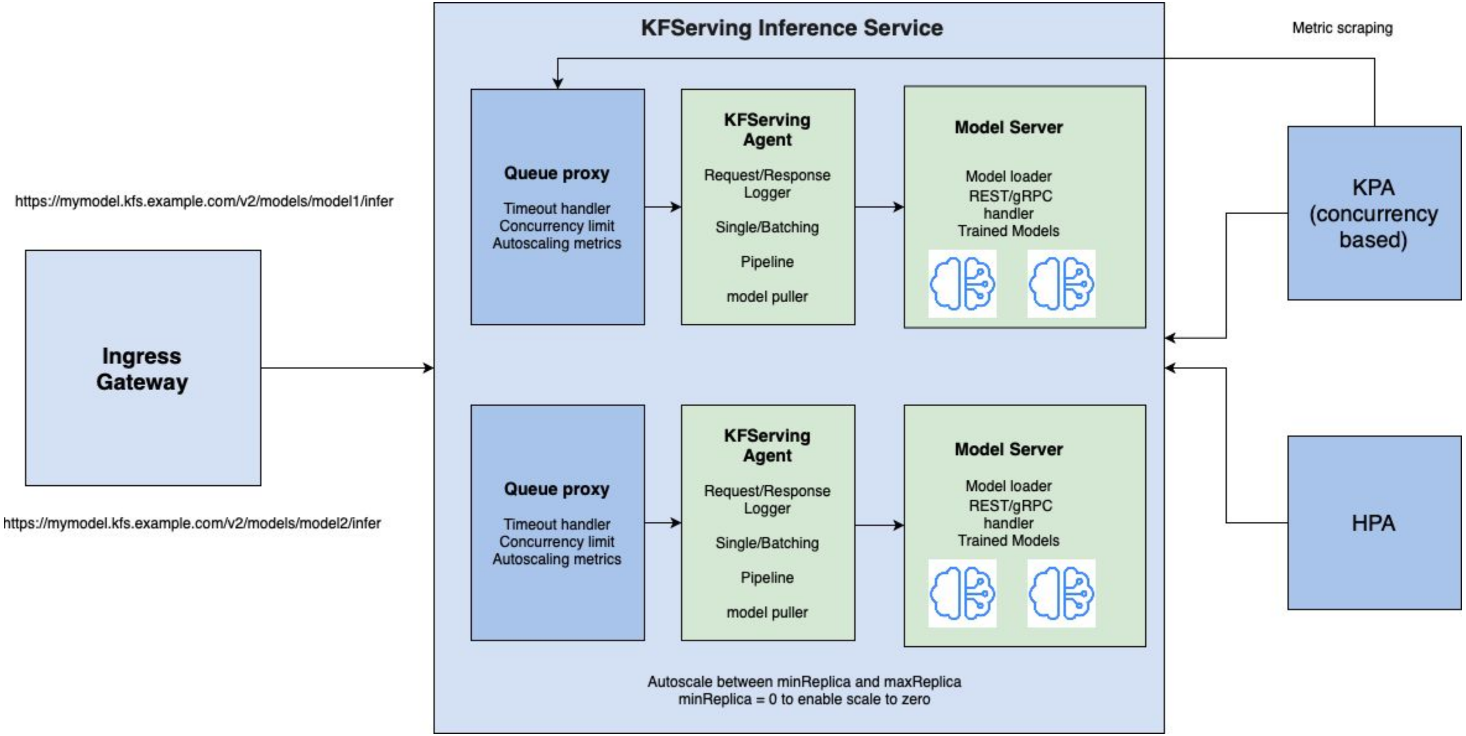
KFServing

- A project founded by Google, NVIDIA, Seldon, Bloomberg, Microsoft, and IBM under Kubeflow.
- Standard deployment across deep learning frameworks on Kubernetes with high performance.
- Create simple intuitive and consistent experience to deploy inference services.
- A complete inference story with feature transformation, prediction, and explanation.
- Serverless inference with GPU Autoscaling to scale down and up from 0!

KFServing Design Patterns

- Knative autoscaler based on request volume, scale down and up from 0.
- Extract common model serving features like model pulling, logging, batching, pipeline to KFServing agent sidecar, so that all model servers can benefit from the serving features provided by KFServing.
- Knative immutable deployment and revision management to ensure safe production rollouts.
- Blue/Green, canary rollouts, progressive rollout.

KFServing Architecture



KFServing v1beta1 Release

- Stable v1beta1 API to support standard model serving for TensorFlow, PyTorch, scikit-learn and XGBoost with v2 prediction protocol.
- Provide a custom serving framework to allow users to bring in own custom serving code while benefit all the serving features that KFServing provides.
- Allows a simple data science-friendly interface, while provide flexibility of specifying pod template fields when needed.
- Complete serving story for pre/post processing, inference and explanation.
- Multi-model serving to improve resource utilization.

TFServing & TorchServe

- Flexible, high-performance serving system for TensorFlow
- Saved model format and graphdef
- Written in C++, supports both REST and gRPC
- https://www.tensorflow.org/tfx/guide/serving

TensorFlow

- Flexible and easy way for serving PyTorch models
- Supports serving eager models and JIT saved TorchScript models
- REST Inference and management API
- https://pytorch.org/serve/

PyTorch

KFServing v1beta1 API

```yaml
apiVerson: serving.kubeflow.org/v1beta1
kind: InferenceService
metadata:
  name: flowers
spec:
  predictor:
    tensorflow:
      storageUri:
"gs://kfserving-samples/models/tensorflow/flowers"
      ports:
        containerPort: 9000 #gRPC port
        name: h2c
```

```yaml
apiVerson: serving.kubeflow.org/v1beta1
kind: InferenceService
metadata:
  name: cifar10
spec:
  predictor:
    pytorch:
      storageUri:
"gs://kfserving-samples/models/pytorch/cifar10"
      env:
        name: OMP_NUM_THREADS
        value: "1"
```

NVIDIA Triton Inference Server

- NVIDIA’s highly-optimized model runtime on GPUs
- Supports model repository, versioning
- Dynamic batching
- Concurrent model execution
- Supports TensorFlow, TorchScript, ONNX models
- Written in C++, supports both REST and gRPC
- TensorRT Optimizer can further bring down inference latency

KFServing v1beta1 API: Triton Inference Server

```
apiVerson: serving.kubeflow.org/v1beta1
kind: InferenceService
metadata:
  name: triton-cifar10
spec:
  predictor:
    triton:
      storageUri:
"gs://kfserving-samples/models/torchscript/cifar"
      env:
        name: OMP_NUM_THREADS
        value: "1"
      resources:
        limits:
          nvidia.com/gpu: "1"
          memory: 4Gi
          cpu: 1
```

- OMP_NUM_THREADS is set to 1 to improve inference performance and reduces the resource contention.
- StorageUri is set to the model repository.
- "nvidia.com/gpu" is specified to deploy the model onto GPU and you can also add node affinity or tolerance to schedule to particular node such as T4 GPU.

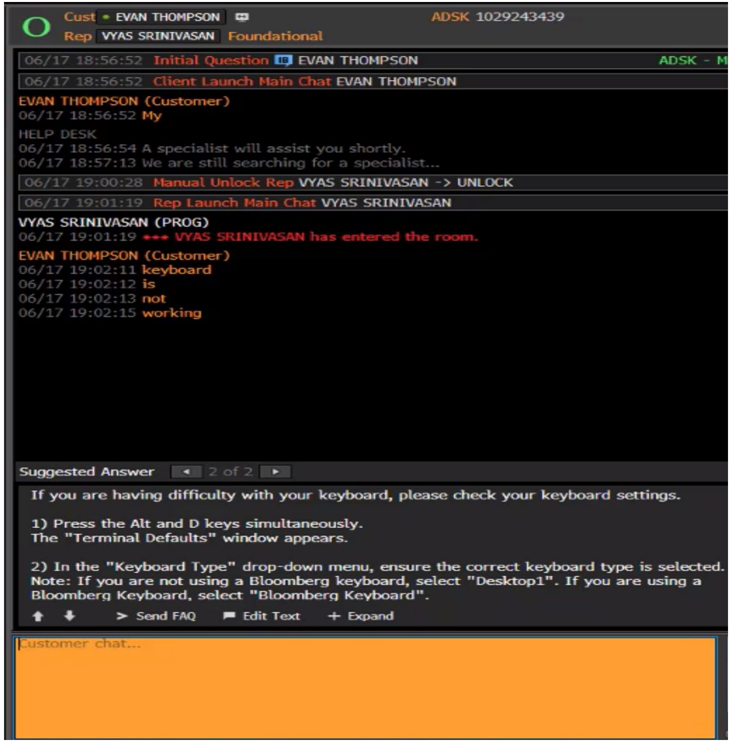
TechAtBloomberg.com

Bloomberg
Engineering

Bloomberg Help Desk Smart Resource

- Customer service reps are pushed content to help answer questions in the Smart Resource window
- All content is curated for accuracy

- How to assist reps provide answers with
  - Higher quality
  - Faster speed



TechAtBloomberg.com

Bloomberg
Engineering

Fine-tuned BERT for Question Similarity

- Data: Categorized and annotated FAQs
  - Within-category annotated questions pairs: similar and not similar
  - Cross-category questions: not similar

- Classification problem
  - Input: two questions
  - Output: similarity score (Similar or not)

- Data mix strategy
  - 50% within-category pairs annotated as “Similar”
  - 25% within-category pairs annotated as “Not Similar”
  - 25% cross-category pairs without annotation

Challengings Serving BERT Models on Production

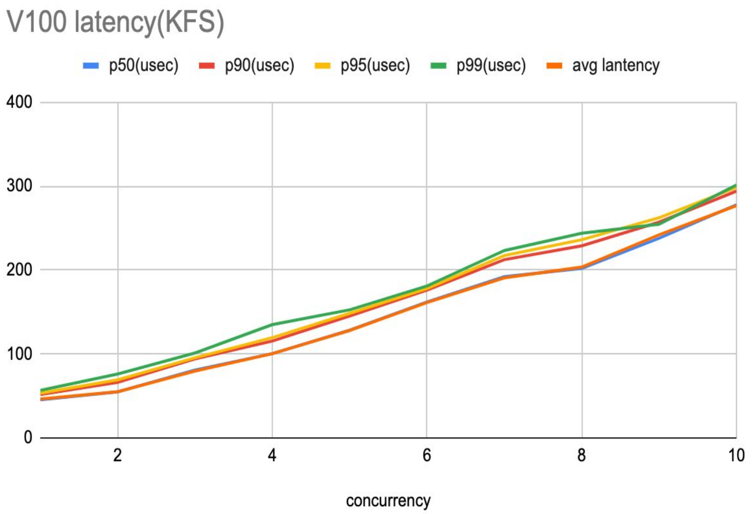
- BERT requires significant compute during inference（100 million parameters).
- Requires pre/post processing before and after the inference.
- Real-time applications, like conversational AI, require low latency.
- Batch evaluation on GPU needs to enable scale down to 0.
- It is much faster on GPU, but how do you better utilize the GPU resources and scale to serve thousands of BERT models?

Deploy BERT Model on KFServing

- Deploy BERT Model on GPU gets 20x speed up.
- Allows bringing custom code for pre/post processing and then calls out to TensorFlow Serving or Triton Inference Server for inference.
- Safe production rollout with Blue/Green and Canary strategy.
- Autoscale based on QPS, scale to 0 after no requests are sent.
- Multi-model serving to improve GPU utilization.

Performance with single Triton pod on GPU

- **SQUAD large 24 layers, fp16, Sequence Length 128 on TESLA V100**



Concurrency	p50(ms)	p90(ms)	p95(ms)	p99(ms)	Throughput
1	45.395	51.736	53.188	56.553	21.6667
2	54.751	66.257	69.182	76.07	36.3333
3	80.942	94.099	95.419	101.189	37.6667
4	100.401	115.389	119.428	134.946	40
5	128.292	145.352	148.42	152.614	39
6	161.971	176.169	178.041	180.996	37
7	192.088	212.405	217.393	223.359	36.6667
8	202.048	228.844	236.175	243.832	39.6667
9	237.9	257.111	262.417	254.646	38
10	277.829	294.093	298.53	301.348	36.6667

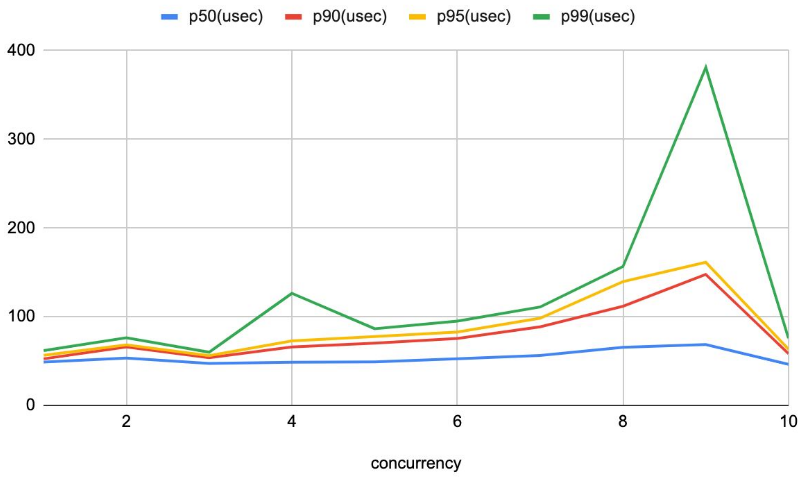
Autoscale on GPUs

```yaml
apiVerson: serving.kubeflow.org/v1beta1
kind: InferenceService
metadata:
  name: triton-bert
spec:
  predictor:
    containerConcurrency: 1
    triton:
      resources:
        limits:
          nvidia.com/gpu: “1”
          cpu: 1
          memory: 8Gi
      storageUri:
“gs://kfserving-examples/models/triton/bert”
```

- Set Container Concurrency to 1 as you can see from previous performance result on a single pod that latency starts to increase when sending concurrent requests and throughput does not increase linearly.

Enable Autoscaling

- Container Concurrency 1



concurrency	p50(ms)	p90(ms)	p95(ms)	p99(ms)	Throughput
1	48.83	52.46	56.436	61.727	18
2	53.413	65.757	68.122	76.23	35.2
3	47.286	53.822	56.118	59.934	63.8
4	48.732	65.929	72.755	77.31	77.2
5	48.976	70.189	77.676	86.478	93.8
6	52.51	75.371	82.646	95.059	106.2
7	56.277	88.548	98.282	110.916	112.8
8	65.387	111.71	139.532	156.64	111.4
9	68.651	147.65	161.35	380.89	103.4
10	46.249	58.326	63.352	75.646	199

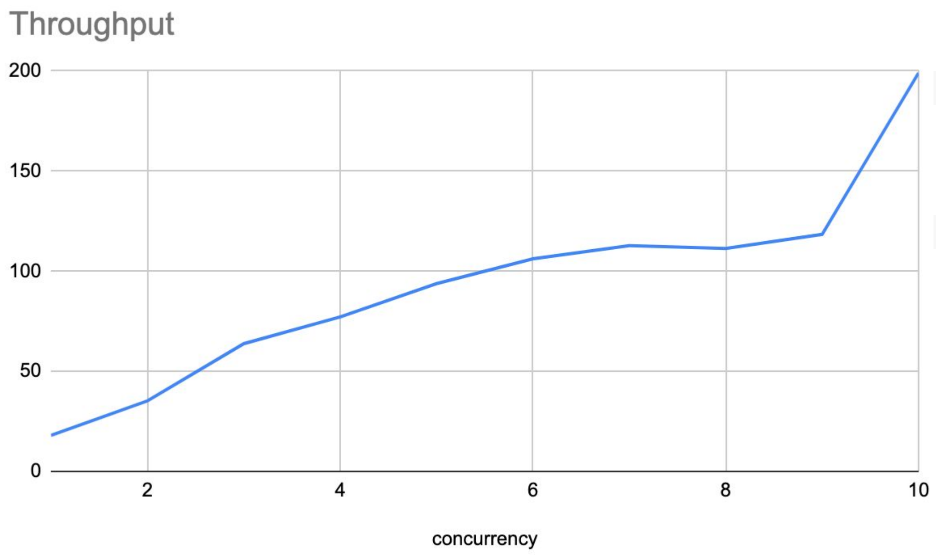
TechAtBloomberg.com

Bloomberg

Engineering

Enable Autoscaling

- **Container Concurrency 1, Min Replica 1**



- It is not exactly linear because of the container cold start up time, on startup InferenceService loads a model from remote storage.
- Model can be cached on PVC so that each pod does not need to load the model individually.

TechAtBloomberg.com

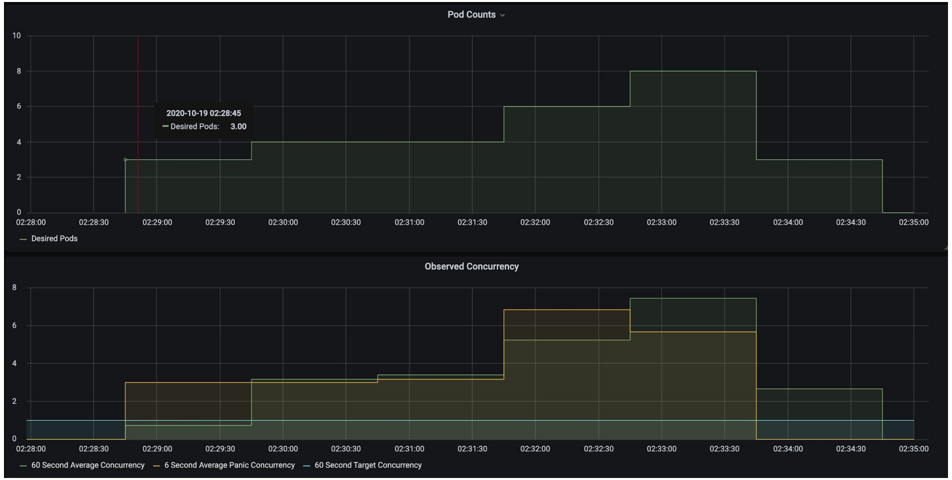
Bloomberg Engineering

How does GPU Autoscaling work ?

- Autoscale based on GPU metrics can be hard, Knative autoscaler works based on in-flight request concurrency.

- **Target concurrency vs. Observed concurrency:** If the target concurrency is 1 and observed concurrency is 10, then autoscaler scales up to 10 pods to process the load.

- Scale down to minReplica or 0 when there is no traffic.



TechAtBloomberg.com

Bloomberg Engineering

Batch Inference

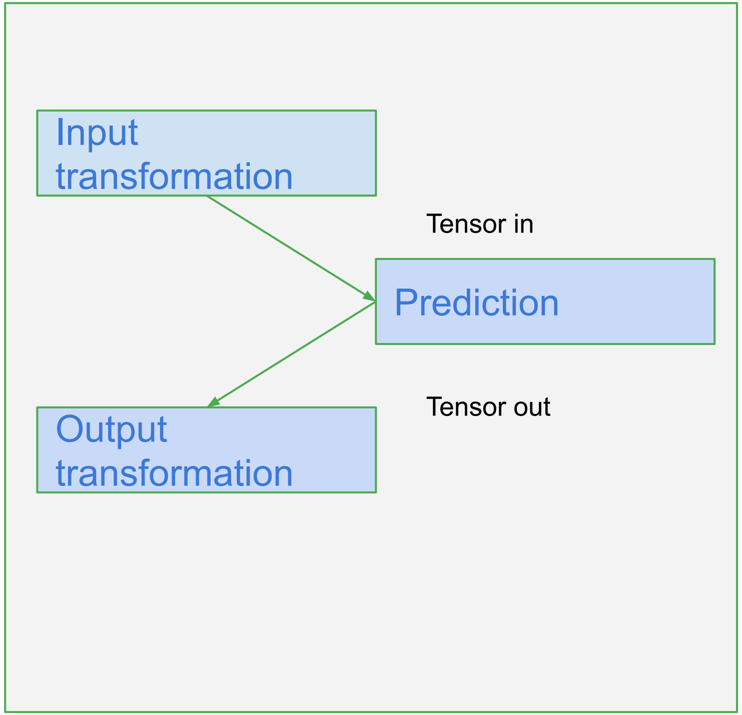
```yaml
apiVerson: serving.kubeflow.org/v1beta1
kind: InferenceService
metadata:
  name: triton-bert
spec:
  predictor:
    batcher:
      maxBatchSize: 16
      maxLatency: 500
    minReplica: 0
    triton:
      resources:
        limits:
          nvidia.com/gpu: "1"
          cpu: 1
          memory: 8Gi
      storageUri:
"gs://kfserving-examples/models/triton/bert"
```

- Server side batching can help increase the throughput and the sidecar agent waits for reaching max batch size or max latency to create the batch.
- We can enable autoscale down to 0 after batch inference is done to save resources and automatically scale up once inference workload starts again.

Inference Service with Transformer

- Often the time you need pre/post processing before and after inference.

- KFServing provides a way to deploy transformers along with predictors, so you can deploy them as a single unit and scale differently with the standardized inference protocol.



Inference Service with Transformer

```yaml
apiVersion: serving.kubeflow.org/v1beta1
kind: InferenceService
metadata:
  name: bert-serving
spec:
  transformer:
    custom:
      containers:
      - image: bert-transformer:v1
        env:
          name: STORAGE_URI
          value: s3://examples/bert_transformer
  predictor:
    triton:
      storageUri: s3://examples/bert
      runtimeVersion: 20.09-py3
      resources:
        limits:
          nvidia.com/gpu: 1
```

Pre/Post Processing

Triton Inference Server

```python
def preprocess(self, inputs: Dict) -> Dict:

    self.doc_tokens =
data_processing.convert_doc_tokens(self.short_paragraph_te
xt)
    self.features =
data_processing.convert_examples_to_features(self.doc_toke
ns, inputs["instances"][0], self.tokenizer, 128, 128, 64)
    return self.features


def postprocess(self, result: Dict)-> Dict:

     (prediction, nbest_json, scores_diff_json) = \
  data_processing.get_predictions(self.doc_tokens,
self.features, start_logits, end_logits, n_best_size,
max_answer_length)

     return {"predictions": prediction, "prob":
nbest_json[0]['probability'] * 100.0
```

TechAtBloomberg.com

Bloomberg

Engineering

Improve GPU/Resource Utilization

- There are common use cases where you want to serve many models for different categories or personalization.
- Schedule single model onto an InferenceService can be expensive and utilization is usually low for serving a single model on a GPU.
- TFServing, TorchServer, Triton Inference Server all allow co-locating multiple models on the same GPU in the container, KFServing adds a TrainedModel CR to enable scheduling models on to the InferenceService at scale.
- All models assigned to the same inference service CR can be accessed under the same URL.

Multi-Model Serving

- Decouple trained model and inference service so you can deploy multiple models on the inference service.
- Each pod can host multiple models under memory constraints; inference can be executed in parallel.
- Provide health check for each model endpoint and reflect model status in TrainedModel CR status.
- Auto-sharding when the given inference service instance is at memory capacity.

Trained Model CR

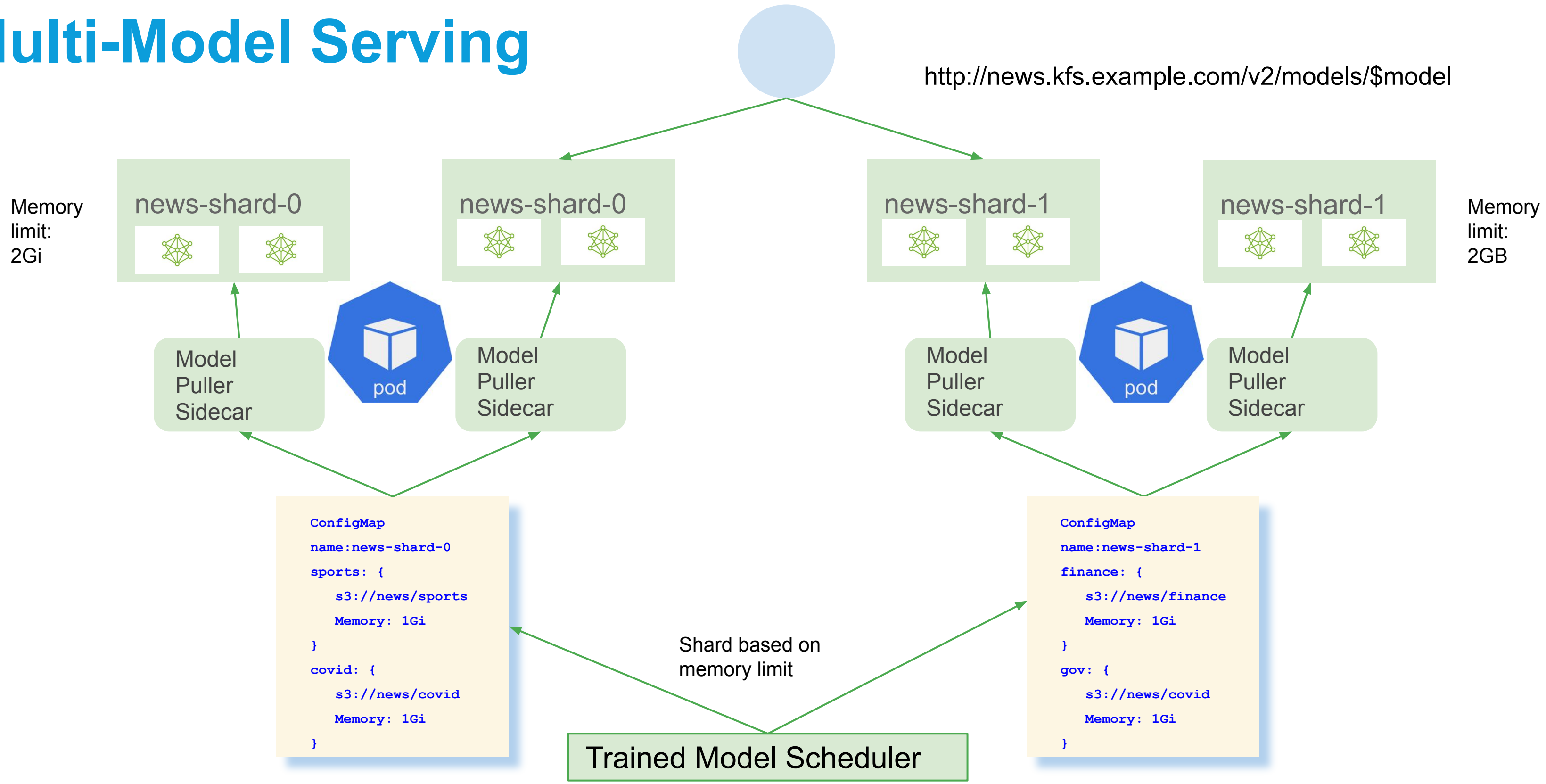
```yaml
apiVerson: serving.kubeflow.org/v1alpha1
kind: TrainedModel
metadata:
  name: sports-news
spec:
  inferenceService: news-category-service
  model:
     storageUri: s3://news-category/sports
     framework: pytorch
     resources:
       memory: 1Gi
```

TechAtBloomberg.com

Bloomberg

Engineering

Multi-Model Serving



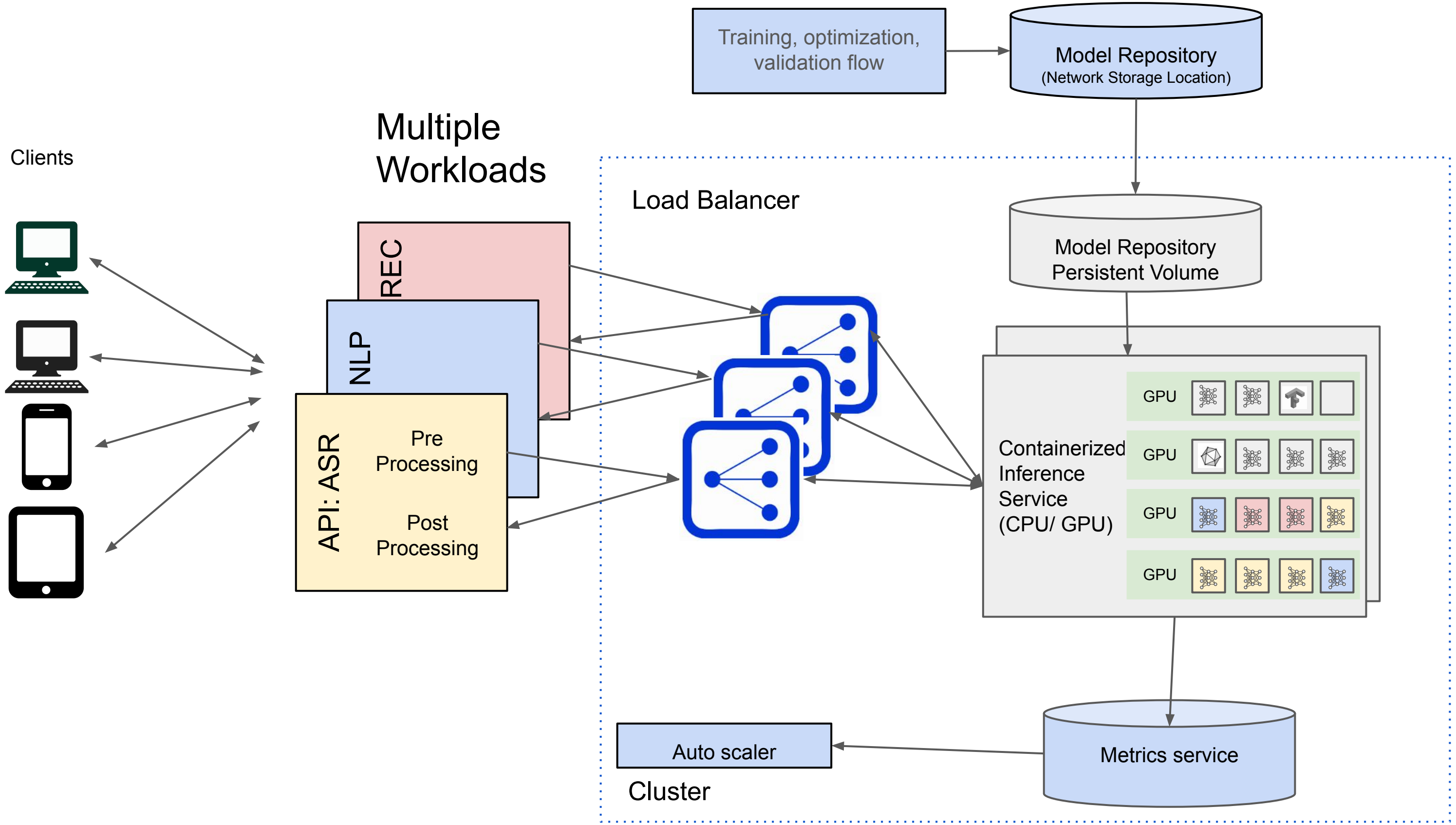
Scalability for Multi-Model Serving

- For a single Istio Ingress gateway, we have limits about running ~2K services, but we want to scale to 100K models.
- Can we support 100K trained model CR? We are bound to the # CR limit on etcd.
- Even we can deploy 100K models on 2K services, we may still hit the limit for the number of virtual services we can create on the gateway for routing the models.
- Phase 2 MMS is to find out these limits.

KFServing Inference Protocol - Version 2 (aka KFServing V2 Dataplane)

David Goodwin, NVIDIA

Protocol Domain - Inference Service



KFServing Inference Protocol - Version 2

- Why a standard protocol?
  - Inference clients can talk to multiple servers, increase portability
  - Inference servers expand client base, increase utility
  - Clients and servers operate seamlessly on platforms that have standardized around this protocol.
- Requirements
  - Support both ease-of-use and high performance
  - Extensible with both standard and server-specific customization
  - GRPC and HTTP/REST

Core Protocol and Extensions

- Core protocol required for all conforming servers
  - Server Live and Ready
  - Server Metadata
  - Model Ready
  - Model Metadata
  - Inference
- Extensions are optional
  - Currently no standard extensions
  - Inference server implementations can provide their own extensions

Live and Ready

- Server Health
  - Indicate that server is live and ready to receive requests
  - Directly use for livenessProbe and readinessProbe
- Model Ready - is specific model ready to receive inference requests

- HTTP/REST returns 200 or 40x status code

```
$ curl -v infer.com/v2/health/live
…
< HTTP/1.1 200 OK
```

- GRPC has dedicated endpoints, for example:

```
rpc ServerLive(ServerLiveRequest) returns (ServerLiveResponse) {}
```

Metadata

- Server Metadata - name, version, extensions
- Model Metadata - name, versions, inputs, outputs

```
$ curl infer.com/v2/models/resnet50
{
  "name" : "resnet50",
  "versions" : [ "1" ],
  "platform" : "tensorflow_graphdef",
  "inputs" : [ {
        "name" : "input",
        "shape" : [ -1, 224, 224, 3 ],
        "datatype" : "FP32"
     } ],
  "outputs" : [ {
        "name" : "resnet50/predictions/Softmax",
        "shape" : [ -1, 1000 ],
        "datatype" : "FP32"
     } ]
}
```

Inference

- Send input tensors to specific model and get back output tensors

```
POST /v2/models/resnet50/infer
{
  "inputs": [ {
    "name" : "input",
    "shape" : [1, 224, 224, 3],
    "datatype" : "FP32",
    "data" : [75.0,87.0,86.0 … ]
  ]
}
```


```
{
  "model_name" : "resnet50",
  "model_version" : "1",
  "outputs" : [ {
    "name" : "resnet50/predictions/Softmax",
    "datatype" : "BYTES",
    "shape" : [1,1],
    "data" : ["0.826413:504:COFFEE MUG"] }
  ]
}
```

Triton Inference Server

- Multi-framework, multi-model, CPU and GPU
- Implements KFServing Inference Protocol as well as extensions

- Per-model statistics endpoint
- Model repository management (load / unload models)
- Support sequences of related inference request (stateful inference)
- Communicate tensors via system and GPU shared memory
- Communicate tensors using binary data (HTTP/REST)

High-Performance HTTP/JSON

- HTTP/REST easy to use, but poor performance encoding tensors with JSON
- Triton *binary data* extension resolves

```
POST /v2/models/resnet50/infer
{
  "inputs": [ {
    "name" : "input",
    "shape" : [1, 224, 224, 3],
    "datatype" : "FP32",
    "data" : [75.0,87.0,86.0 … ]
  ]
}
```

Encode and decode 150k+ FP numbers to send a single small image

```
POST /v2/models/resnet50/infer
  'Inference-Header-Content-Length': 123
{
  "inputs": [ {
    "name" : "input",
    "shape" : [1, 224, 224, 3],
    "datatype" : "FP32",
    "parameters":{"binary_data_size":602112}
  ]
}<602112 bytes of binary data>
```

Extension allows "raw" data to be sent after JSON header. Eliminates encode and decode overhead

- For 128 requests, local network, provides ~ 17x speedup

Inference Protocol Reference

- KFServing Inference Protocol - Version 2

  https://github.com/kubeflow/kfserving/tree/master/docs/predict-api/v2

- Triton Inference Server implements core protocol plus many extensions

  https://github.com/triton-inference-server/server/blob/master/docs/inference_protocols.md

KFServing Reference

- **KFServing v0.5.0 with v1beta1 API and inference v2 protocol**

  RFC:https://docs.google.com/document/u/1/d/1ktiO7gWohq19C_rixXH0T_D91TjkrQELlQjlkvSefVc

- **Alpha version of Multi-Model Serving.**
- **GitHub:** https://github.com/kubeflow/kfserving
- **Examples:** https://github.com/kubeflow/kfserving/tree/master/docs/samples
- **Open community and we love your contributions!**