KCD Budapest 2025

Model inferencing on Kubernetes with KServe

Peter Cseh – Gabor Lovass

About us

Peter Cseh

Lead Software Engineer

Altair / Siemens

https://www.linkedin.com/in/cseh-peter/

Gabor Lovass

Senior Software Engineer

Altair / Siemens

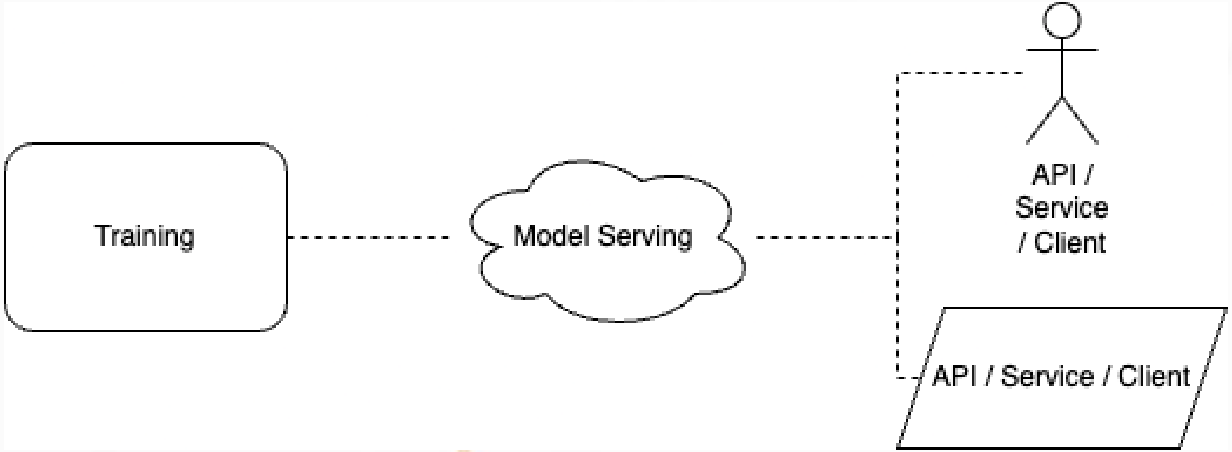
https://www.linkedin.com/in/glovass/

Content

- Model Serving
- Why Kubernetes is a Great Platform for Serving Models?
- KServe
- Serving Runtime and Inference Service
- Inference Service Deployment
- Storage – Modelcar
- LLM serving Runtime
- KServe in action
- Q&A

Model Serving

Model Serving is an integral part of the Machine Learning (ML) lifecycle



Why Kubernetes is a Great Platform for Serving Models?

Deploying model as a microservice is the most common model serving strategy

- **Microservice**
- **Resource management**
- **Reproducibility / Portability**
- **Scalability**
- **Fault-tolerance**

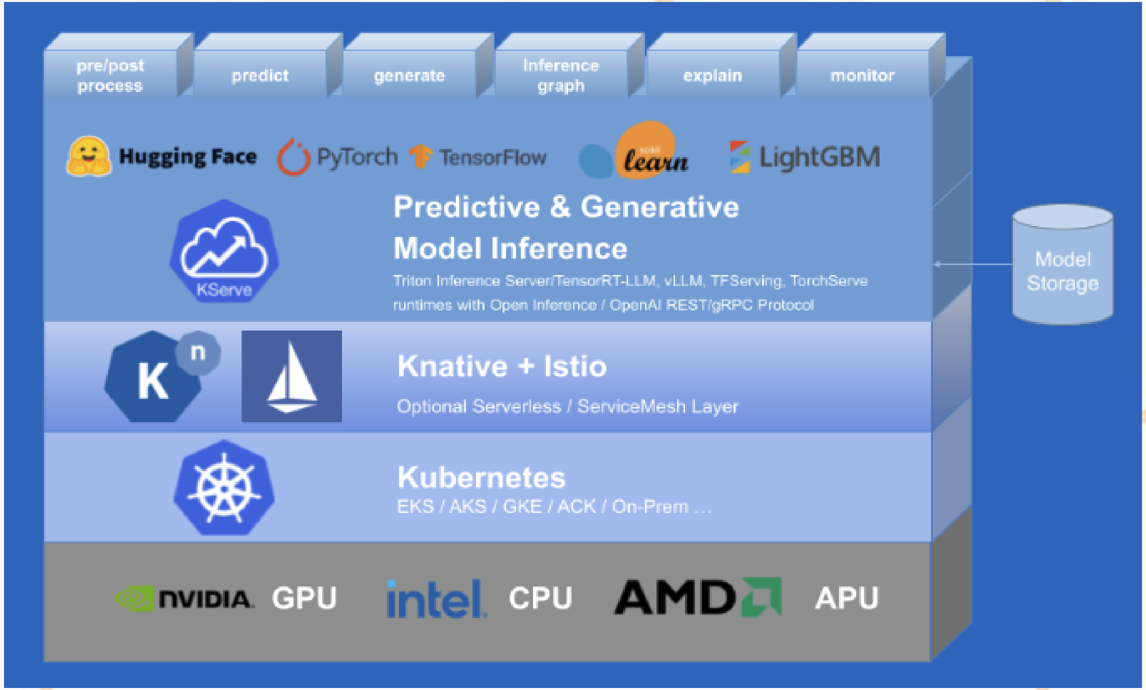
So what?

$$\underbrace{\text{Model Serving + Kubernetes}}_{\text{Let's make it simple}} = \text{?}$$

KServe

KServe is highly scalable, standards-based, cloud agnostic **Model Inference Platform** for serving predictive and generative AI models on Kubernetes

- performant, **standardized inference protocol** across ML frameworks
- **serverless** inference workload with **autoscaling** including scale-to-zero on CPU and GPU
- simple and pluggable production serving for **inference, pre/post processing, monitoring** and **explainability**



pre/post process

predict

generate

Inference graph

explain

monitor

Hugging Face PyTorch TensorFlow learn LightGBM

KServe

Predictive & Generative Model Inference

Triton Inference Server/TensorRT-LLM, vLLM, TFServing, TorchServe runtimes with Open Inference / OpenAI REST/gRPC Protocol

Model Storage

Knative + Istio

Optional Serverless / ServiceMesh Layer

Kubernetes

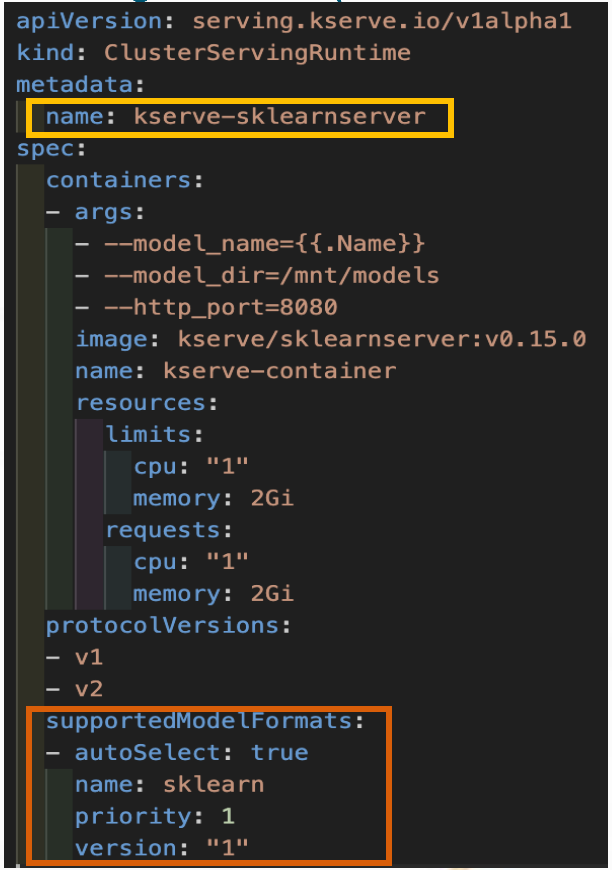
EKS / AKS / GKE / ACK / On-Prem ...

NVIDIA GPU intel CPU AMD APU

Version: 0.15.0

Serving Runtime and Inference Service

Serving Runtime (for KServe admin)

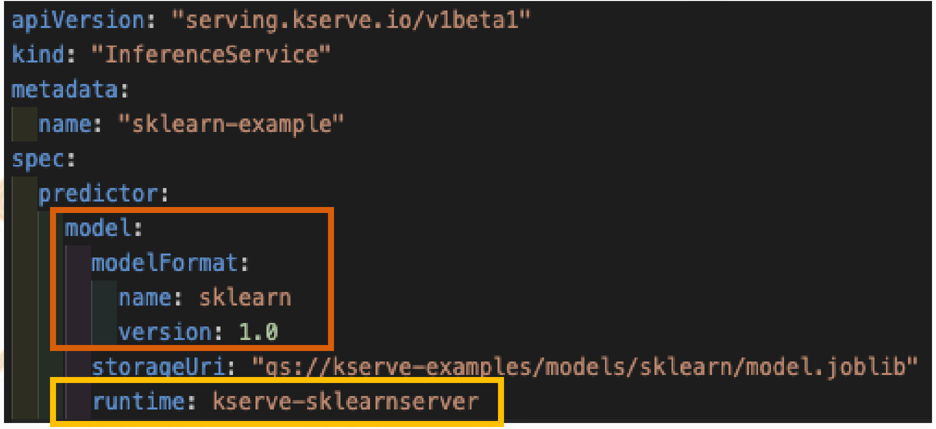


```
apiVersion: serving.kserve.io/v1alpha1
kind: ClusterServingRuntime
metadata:
  name: kserve-sklearnserver
spec:
  containers:
  - args:
    - --model_name={{.Name}}
    - --model_dir=/mnt/models
    - --http_port=8080
    image: kserve/sklearnserver:v0.15.0
    name: kserve-container
    resources:
      limits:
        cpu: "1"
        memory: 2Gi
      requests:
        cpu: "1"
        memory: 2Gi
  protocolVersions:
  - v1
  - v2
  supportedModelFormats:
  - autoSelect: true
    name: sklearn
    priority: 1
    version: "1"
```

Serving Runtime defines the templates for pods that can serve one or more model formats

Inference Service allows to specify the model formats and version for a trained model

InferenceService (for user)

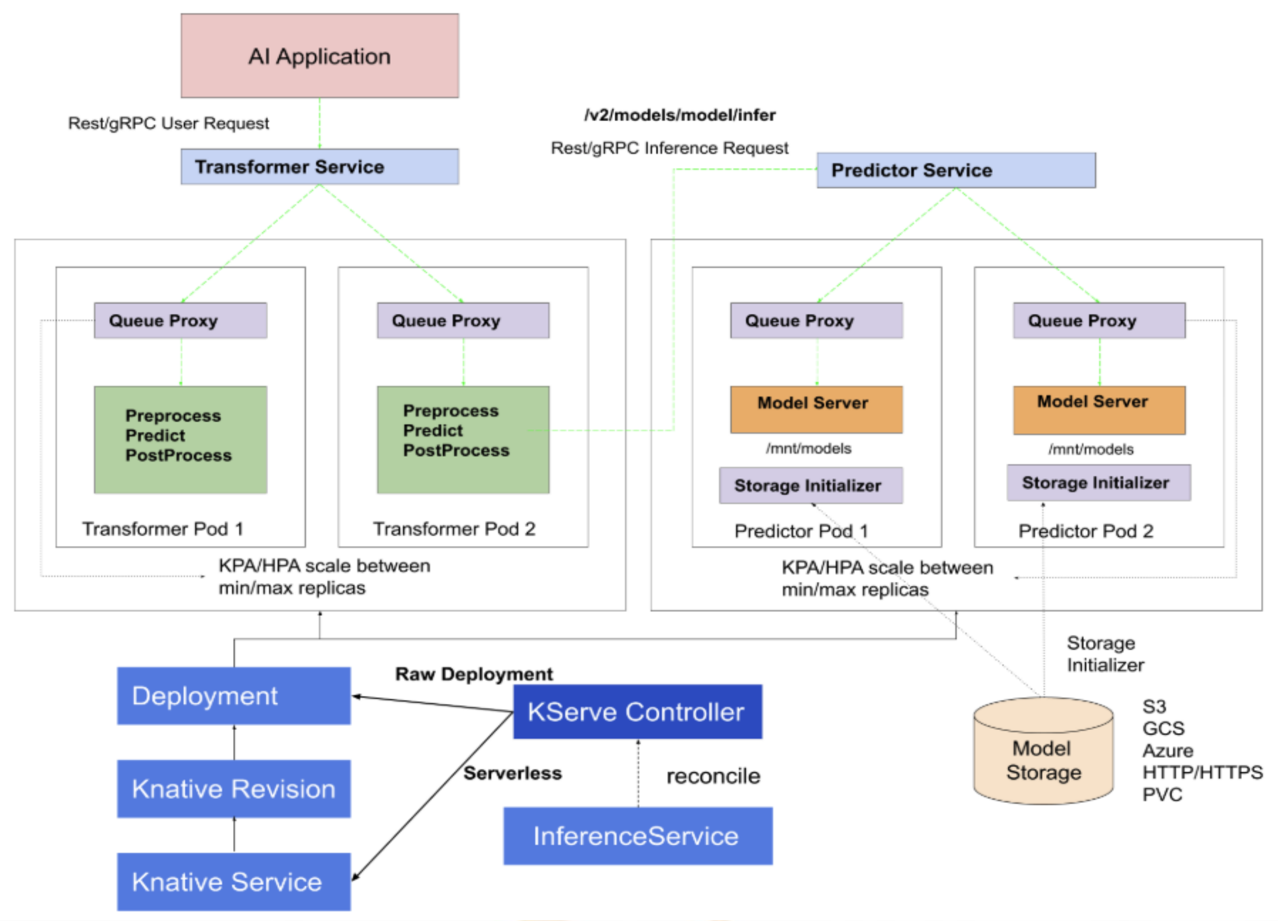


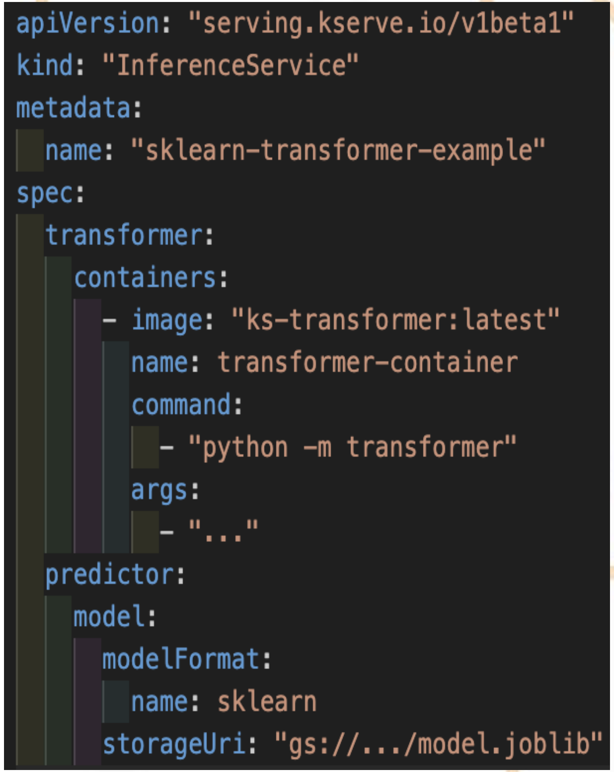
```
apiVersion: "serving.kserve.io/v1beta1"
kind: "InferenceService"
metadata:
  name: "sklearn-example"
spec:
  predictor:
    model:
      modelFormat:
        name: sklearn
        version: 1.0
      storageUri: "gs://kserve-examples/models/sklearn/model.joblib"
      runtime: kserve-sklearnserver
```

Serving Runtime Support Matrix (0.15.0)

Serving Runtime / Model Format	Scikit-learn	Tensorflow	PyTorch	LightGBM	Paddle	PMML	Spark MLlib	XGBoost	ONNX	MLFlow	LLM
KServe (0.15.0)	●			●	●	●	●	●			●
KServe - Triton (23.05-py3)		●	●						●		
KServe - TorchServe (0.9.0)			●								
KServe - MLServer (1.5.0)	●			●				●		●	
KServe - TFServing (2.6.2)		●									

Inference Service Deployment

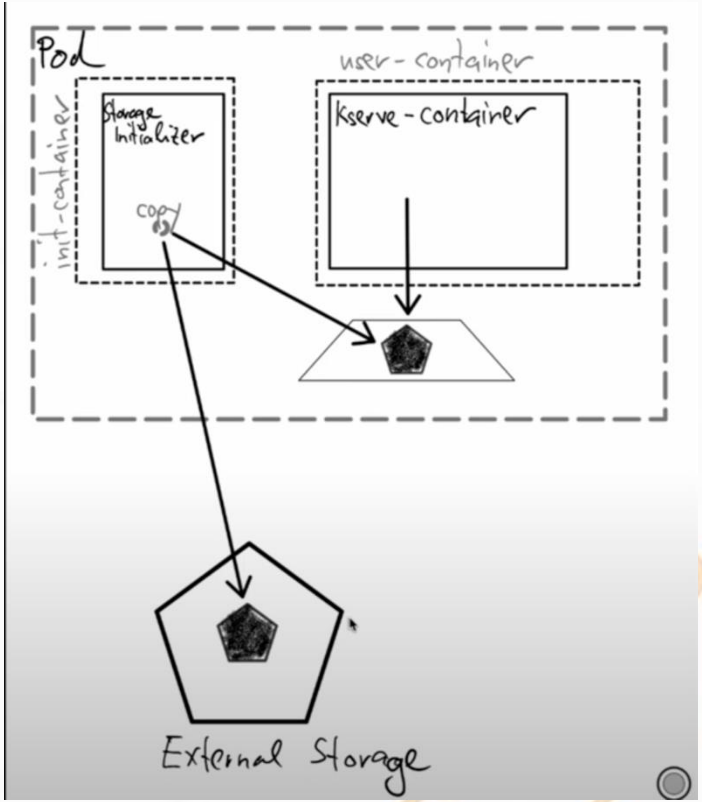


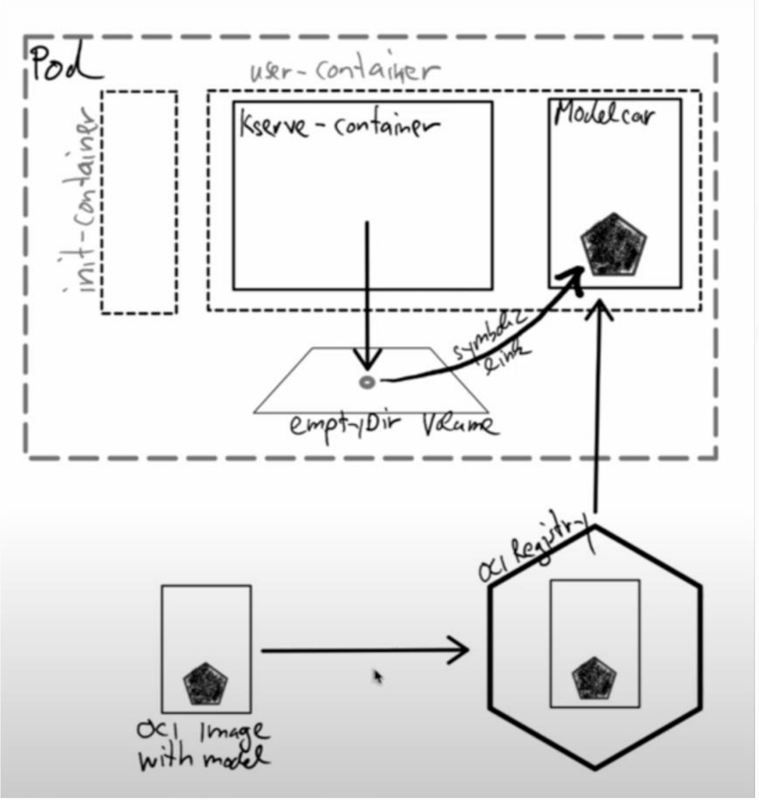


```yaml
apiVersion: "serving.kserve.io/v1beta1"
kind: "InferenceService"
metadata:
  name: "sklearn-transformer-example"
spec:
  transformer:
    containers:
      - image: "ks-transformer:latest"
        name: transformer-container
        command:
          - "python -m transformer"
        args:
          - "..."
  predictor:
    model:
      modelFormat:
        name: sklearn
      storageUri: "gs://.../model.joblib"
```

Storage - Modelcar

- KServe's **traditional** approach vs **Modelcar**





LLM Serving Runtime

- The Hugging Face serving runtime implements two backends namely **Hugging Face** and **vLLM** that can serve Hugging Face models out of the box.
- Serve Hugging Face runtime by default uses **vLLM** backend
- The Hugging Face runtime supports the following ML tasks:
  - Text Generation
  - Text2Text Generation
  - Fill Mask
  - Token Classification
  - Sequence Classification (Text Classification)

DEMO

Demo 1 – Error detection

- Solve the following task with an Inference Service
- Create an alerting application that detects anomaly in logs to help detect issues in production clusters as soon as possible to avoid outages
- Include a transformer that preprocesses the request data and transforms the result
- Deploy it as an Inference Service

Demo 2 – Error detection

- Improve the previous solution and use an LLM
- Store the model with the Modelcar (OCI) approach that improve scalability
- Deploy the model as an Inference Service

Q&A

Thank you