KUBERNETES COMMUNITY DAYS BEIJING 2025

Automated Model Serving in HP Cloud

John Zheng

Subject

01 **HP Cloud Introduction**

02 **Requirements & Design For AI Inference**

03 **Implement with KServe /Istio /Envoy**

- Model Deployment
- Security And Authentication
- HPA Autoscaling
- Observability
- LLM Token Limit And Summarize
- Canary

04 **Q & A**

KCD
GROWING CLOUD NATIVE TOGETHER
BEIJING

Part 01

HP Cloud Introduction

HP Cloud Introduction

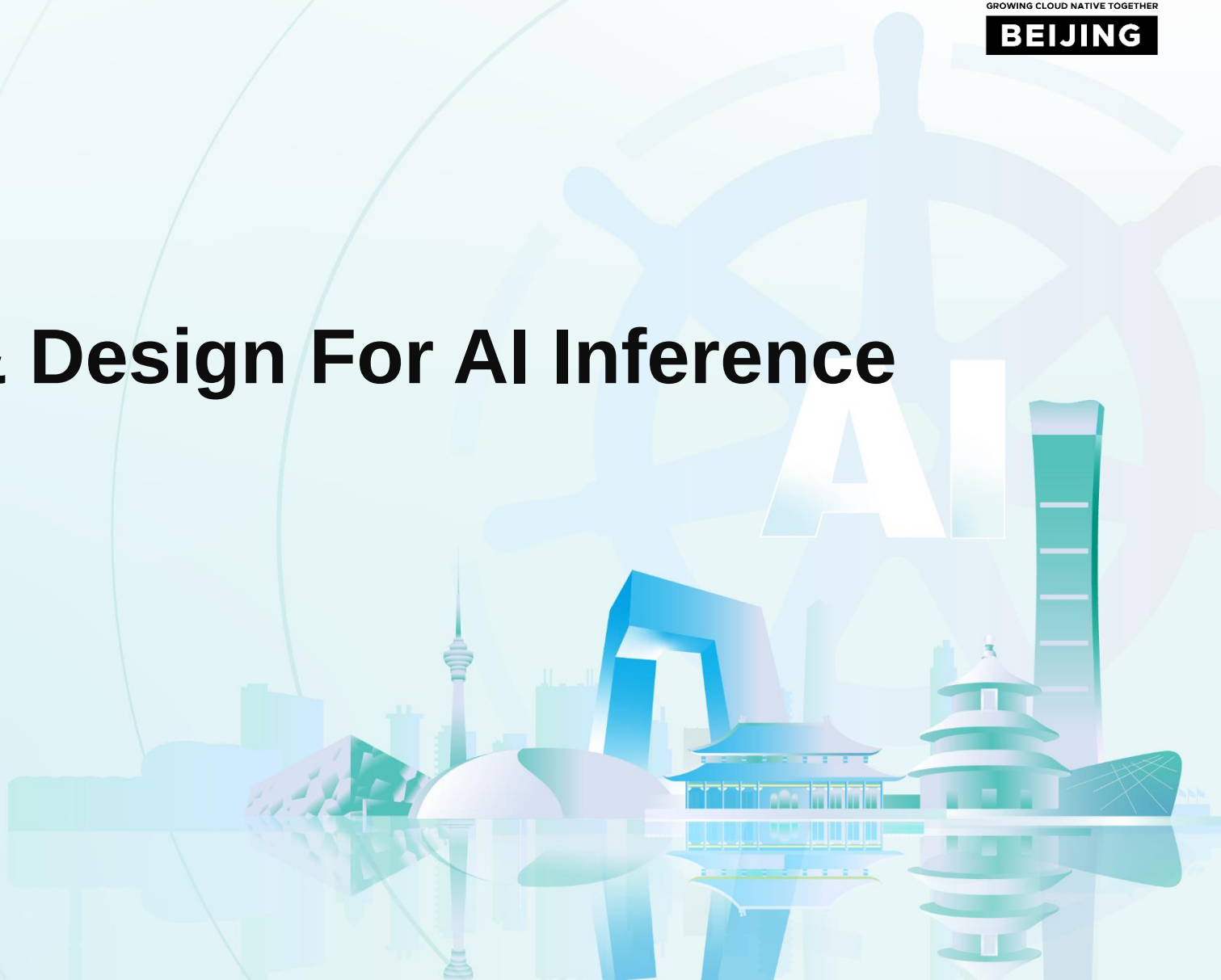
- Built on Amazon, Azure, works for all internal HP project services, providing automatic deployment, monitoring, operations.
  - Kubernetes
  - Istio
  - Harbor
  - Azure Pipeline

- Full Automation
  - Infrastructure as Code
    - Terraform, Terragrunt
  - Continuous Integration/Continuous Deployment
    - Helm, Flux2

KCD
GROWING CLOUD NATIVE TOGETHER
BEIJING

Part 02

Requirements & Design For AI Inference



AI Inference Requirements

- Supported Models:
  - Generative AI Models: LLaMA 3, QWEN, and more
  - Machine Learning Frameworks: Scikit-learn, XGBoost
  - Deep Learning Frameworks: TensorFlow Serving, PyTorch ONNX models
  - Tools: Hugging Face Transformers
- Supported Model Storage:
  - Compatible with Hugging Face, Amazon S3, PVC, EFS, and other storage solutions.

AI Inference Requirements

- Each project can **easily** onboard model, with *configuration file* or *develop tool UI*
- Model get features automatically, **without any coding** inside models, like Security, Autoscaling, Observability, Token Summarize, Canary … …

AI Inference Design

- KServe as the Foundation (Not Using Knative)
- No Dependency on KServe, Fully Customizable
  - Security and Authentication (Istio)
  - HPA Autoscaling (Prometheus Adapter)
  - Observability
  - LLM Token Limit And Summarize (enovyfilter)
  - Canary (Istio)
  - API Rate Limit (Envoy ratelimit)
  - … …

Part 03

Implementation with KServe/ Istio/ Envoy

- Model Deployment
- Security And Authentication
- HPA Autoscaling
- Observability
- API Rate Limit
- LLM Token Limit And Summarize
- Canary

KCD
GROWING CLOUD NATIVE TOGETHER
BEIJING

Part 03 - 01

Implementation - Model Deployment

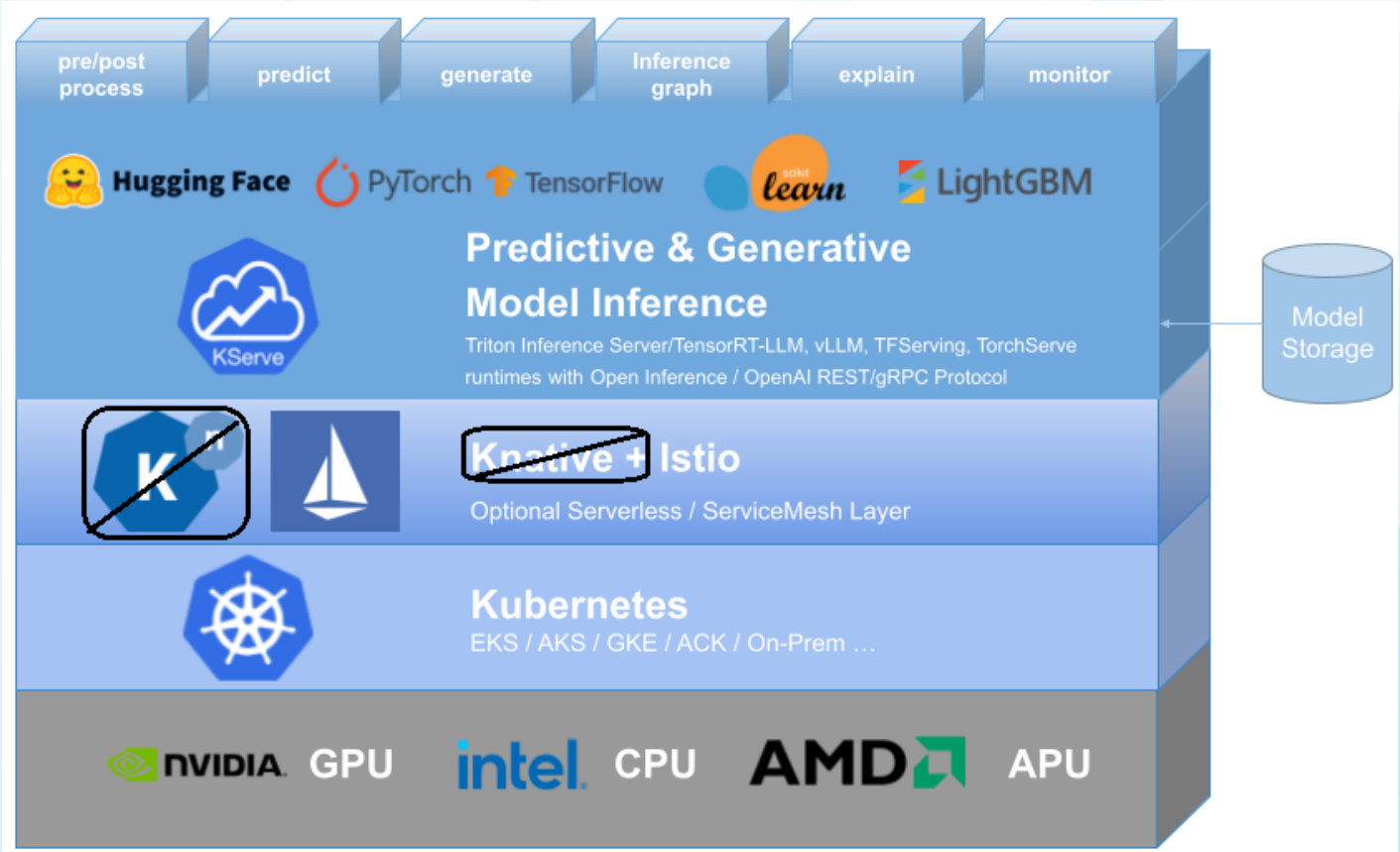
Model Deployment

KServe： Highly scalable and standards based Model Inference Platform

Excluded Features: Knative

Serverless is not suitable for AI inference.

Runtime： Amazon EKS



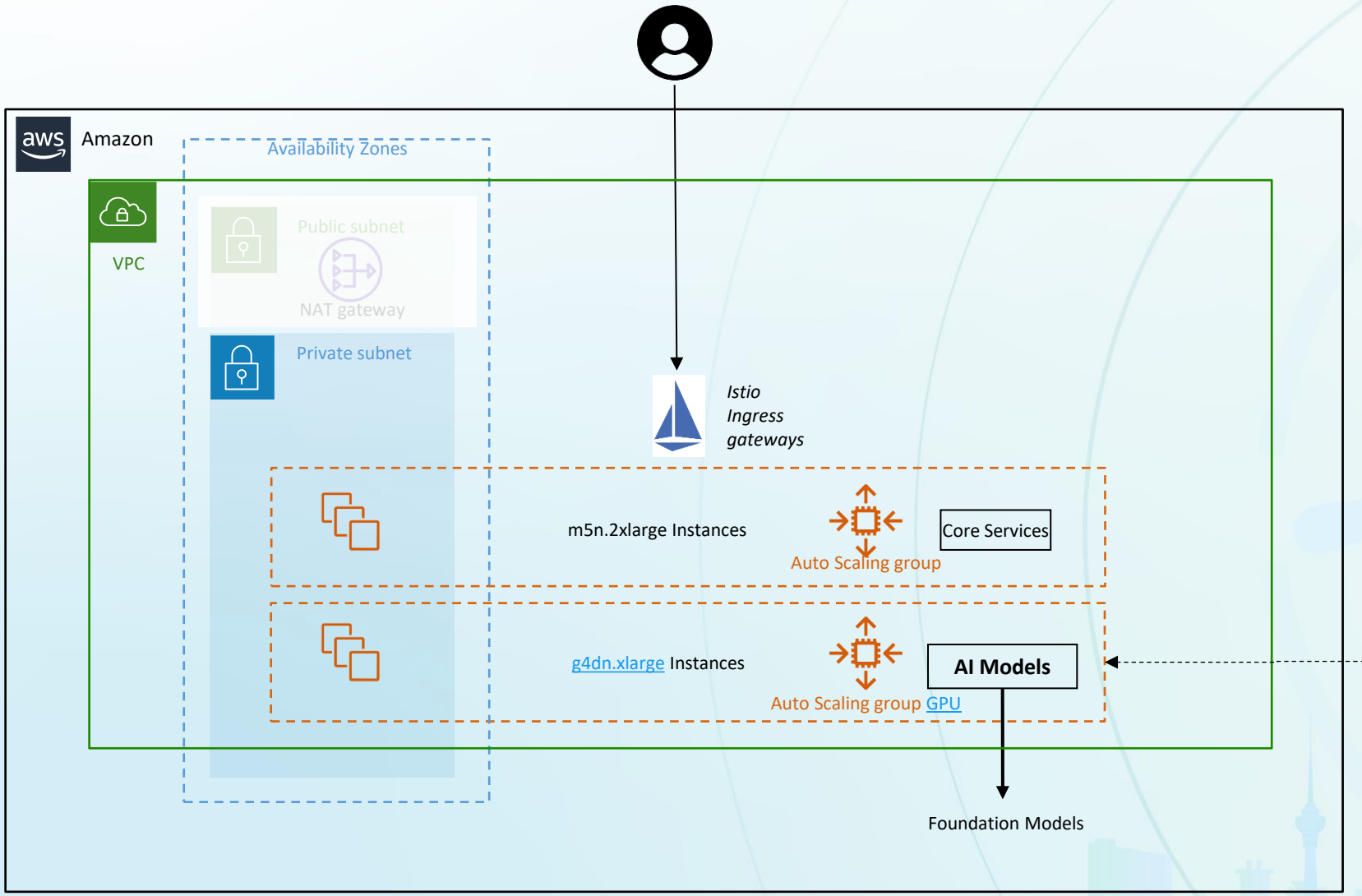
Model Deployment

- KServe Core Capabilities, covered:
  - Multiple Model Types: Hugging Face, Scikit-learn, PyTorch, TensorFlow… …

    (Support **Custom Model**, thus, we can deploy our own **Omen AI model**, or **vLLM**)
  - Multiple Storage Options: Hugging Face, Amazon S3, PVC, EFS, and more… …
  - Model Explainability
  - Multi Model Serving and Inference Graph

- With Helm and Flux2， automated model deploy [Sample Code]

Support GPU Node



Install NVIDIA [k8s-device-plugin]

*Add "nodegroup gpu" in **terragrunt** file:*

```
nodegroup = {
  gpu = {
    node_labels = {
      "node-group" = "gpu",
    }
    node_taints = [
      {
        key    = "nvidia.com/gpu"
        value  = "1"
        effect = "NO_SCHEDULE"
      },
    ]
    instance_type = "g4dn.xlarge"
  },
}
```

Support GPU Node

How AI Model apply GPU Resource?

Enterprise

Model owner defined:

```
resources:
  requestGPU: 1
  limitGPU: 1
```

HelmRelease Value

```
{{- if or .Values.resources.requestGPU .Values.resources.limitGPU }}
tolerations:
  - key: nvidia.com/gpu
    operator: Exists
{{- end }}
```

```
affinity:
  nodeAffinity:
    requiredDuringSchedulingIgnoredDuringExecution:
      nodeSelectorTerms:
      - matchExpressions:
        - key: eks.amazonaws.com/nodegroup
          {{- if or .Values.resources.requestGPU .Values.resources.limitGPU }}
          operator: In
          {{- else }}
          operator: NotIn
          {{- end }}
          values:
          - {{ .Values.clusterName }}-gpu
  {{- end }}
```

Helm Charts

"Pod need GPU"

Run On

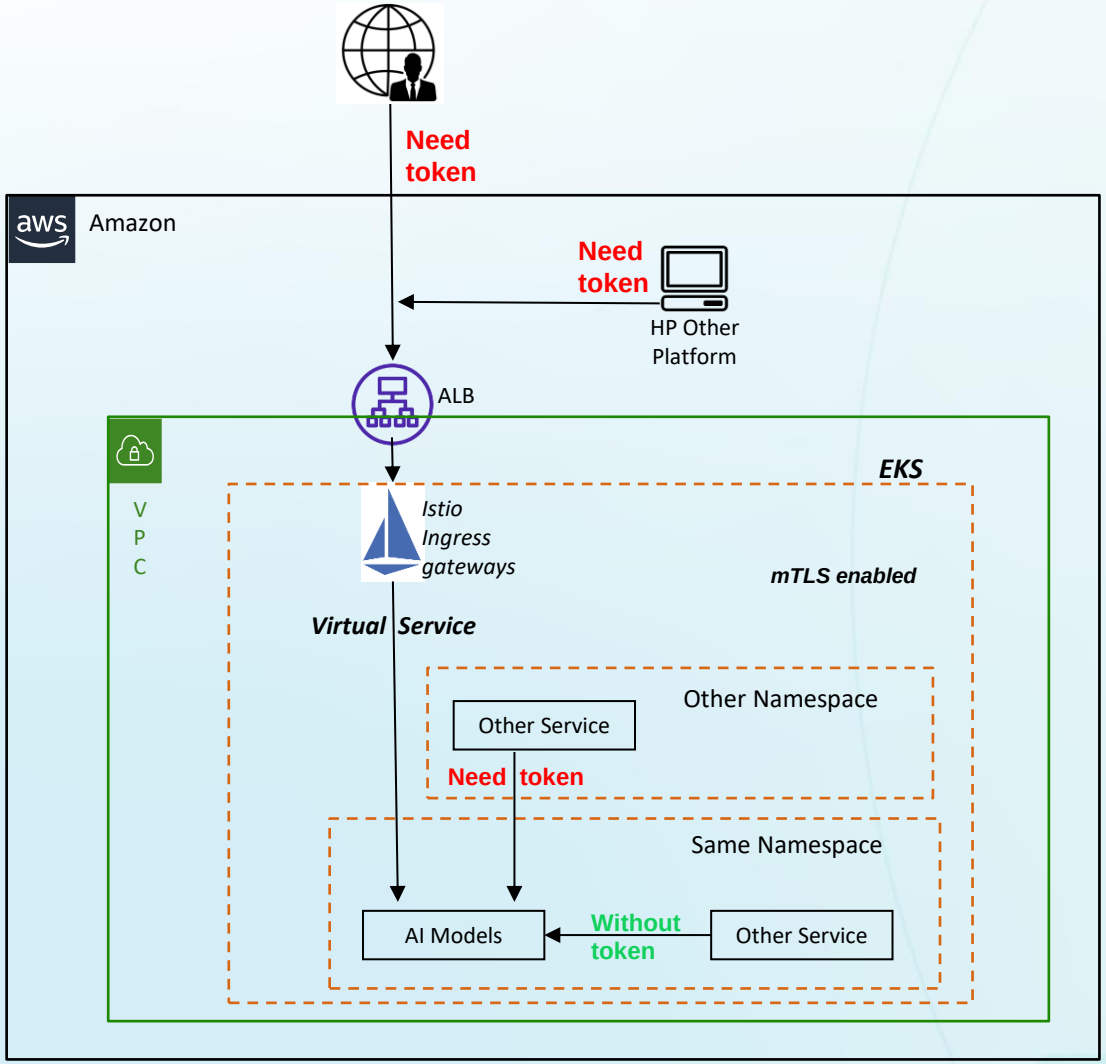
GPU Node

KCD
GROWING CLOUD NATIVE TOGETHER
BEIJING

Part 03 - 02

Implementations - Security And Authentication

Implementations - Security And Authentication



$ curl -X POST -H "Content-Type: application/json" -d payload.json \

https://sample-svc.dev.int.example.com/v2/models/test_model/infer

< HTTP/2 403

RBAC: access denied

$ curl -X POST -H "Content-Type: application/json" -d payload.json \

https://sample-svc.dev.int.example.com/v2/models/test_model/infer \

-H "Authorization: Bearer $token"

< HTTP/1.1 200 OK

{ … ….

Implementations - Security And Authentication

1. Istio mTLS

Details

2. Publish AI Model with Istio Virtual Service

Details

3. Authentication And Authorization by Istio

- Need Token for outside request
- Need Not Token for request in same namespace
- CUSTOM Authorization

Details

KCD
GROWING CLOUD NATIVE TOGETHER
BEIJING

Part 03 - 03

Implementations - HPA Autoscaling

HPA Autoscaling

Requirements: Need Horizontal Pod Autoscaler, based on

- CPU utilization
- Memory utilization
- **Requests per second**
- **GPU utilization**

Design:

- CPU utilization HPA, Memory utilization HPA are supported by default
- **Support Custom metrics HPA, with Prometheus adapter**

Prometheus Adaptor

HPA Autoscaling

- Prometheus adapter – Architecture & Installation Details

- How to implement HPA based on "Requests per second"

  With Istio metrics **istio_requests_total**

  Details

- How to implement HPA based on "GPU utilization"

  Install NVIDIA gpu-operator

  With Nvidia metrics **DCGM_FI_DEV_GPU_UTIL**

  Details

HPA Autoscaling

Flux: Install Prometheus adaptor [Code]

```yaml
---
apiVersion: helm.toolkit.fluxcd.io/v2beta2
kind: HelmRelease
metadata:
  name: prometheus-adapter
  namespace: infra
spec:
  releaseName: prometheus-adapter
  chart:
    spec:
      chart: prometheus-adapter
      version: 4.10.0
      sourceRef:
        kind: HelmRepository
        name: prometheus
  values:
    prometheus:
      url: http://prometheus-server.infra.svc.cluster.local
      port: 80
    rules:
      default: false
      custom:
      - seriesQuery: istio_requests_total{pod!="", namespace!=""}
        resources:
          overrides:
            namespace:
              resource: namespace
            pod:
              resource: pod
        name:
          matches: "istio_requests_total"
          as: "requests_per_second"
        metricsQuery: sum(rate(<<.Series>>{<<.LabelMatchers>>}[2m])) by (<<.GroupBy>>)
      - seriesQuery: '{__name__=~"^DCGM_FI_DEV_GPU_UTIL$", app="nvidia-dcgm-exporter", containe
        resources:
          overrides:
            exported_namespace:
              resource: namespace
            pod:
              resource: pod
        name:
          matches: DCGM_FI_DEV_GPU_UTIL
          as: "gpu_utilization"
        metricsQuery: avg(avg_over_time(<<.Series>>{<<.LabelMatchers>>}[1m])) by (<<.GroupBy>>)
```

Helm Chart: HPA template [Code]

```yaml
apiVersion: autoscaling/v2
kind: HorizontalPodAutoscaler
metadata:
  name: {{ .Release.Name }}-predictor
  namespace: {{ .Release.Namespace }}
  labels:
    app: {{ .Release.Name }}
spec:
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: {{ .Release.Name }}-predictor
  {{- with .Values.autoscaling.targetGPUUtilizationPercentage }}
  - type: Pods
    pods:
      metric:
        name: gpu_utilization
      target:
        averageValue: {{ . }}
        type: Value
  {{- end }}
  {{- with .Values.autoscaling.targetRequestRate }}
  - type: Pods
    pods:
      metric:
        name: requests_per_second
      target:
        averageValue: {{ . }}
        type: Value
  {{- end }}
```

Helm Release Values [Example]

```yaml
replicaCount: 1
autoscaling:
  maxReplicaCount: 10
  targetCPUUtilizationPercentage: 80
  targetMemoryUtilizationPercentage: 70
  targetGPUUtilizationPercentage: 75
  targetRequestRate: 100
```

KCD
GROWING CLOUD NATIVE TOGETHER
BEIJING

Part 03 - 04

Implementation - Observability

Observability

Bring excellent observability to AI Models

- Logs
  - KServe Logger Component
  - Istio Access log
- Metrics
  - Enable KServe metrics for Prometheus
  - Model metrics (vLLM)
  - GPU Metrics
  - Istio Metrics

Observability

Logs

- KServe Logger Component [Details]

  With **KServe Inference Logger**, **all predict header/ body of requests/ response** can be sent to your “message handle” service, for save to S3, database, or just print out.

```yaml
apiVersion: serving.kserve.io/v1beta1
kind: InferenceService
metadata:
  name: sklearn-iris
spec:
  predictor:
    logger:
      mode: all
      url: http://message-dumper.default/
    model:
      modelFormat:
        name: sklearn
      storageUri: gs://kfserving-examples/models/sklearn/1.0/model
```

Configure

```
{  "instances": [    [6.8,  2.8,  4.8,  1.4],    [6.0,
Received Request:
x-request-id: e4123d01-5d29-9ab8-8f4a-76761d62d18b
x-b3-traceid: 4933d0bdf218ca0c3b514339c0f9fd9f
x-b3-spanid: 2d576fcb7dd00f52
x-b3-flags: Not provided
Payload:
{"predictions":[1,1]}
```

Logs in message service

- Istio Access log [Details]
  - Include API host, method, path, response code, duration … …

Observability

Metrics – Enable KServe metrics for Prometheus [Doc]

Each AI Model has its metrics, 如:

- All supported serving runtimes support metrics by default kserve/config/runtimes

```yaml
metadata:
  name: kserve-torchserve
spec:
  annotations:
    prometheus.kserve.io/port: '8082'
    prometheus.kserve.io/path: "/metrics"
```

- Custom Model metric can also be support in Helm Chart

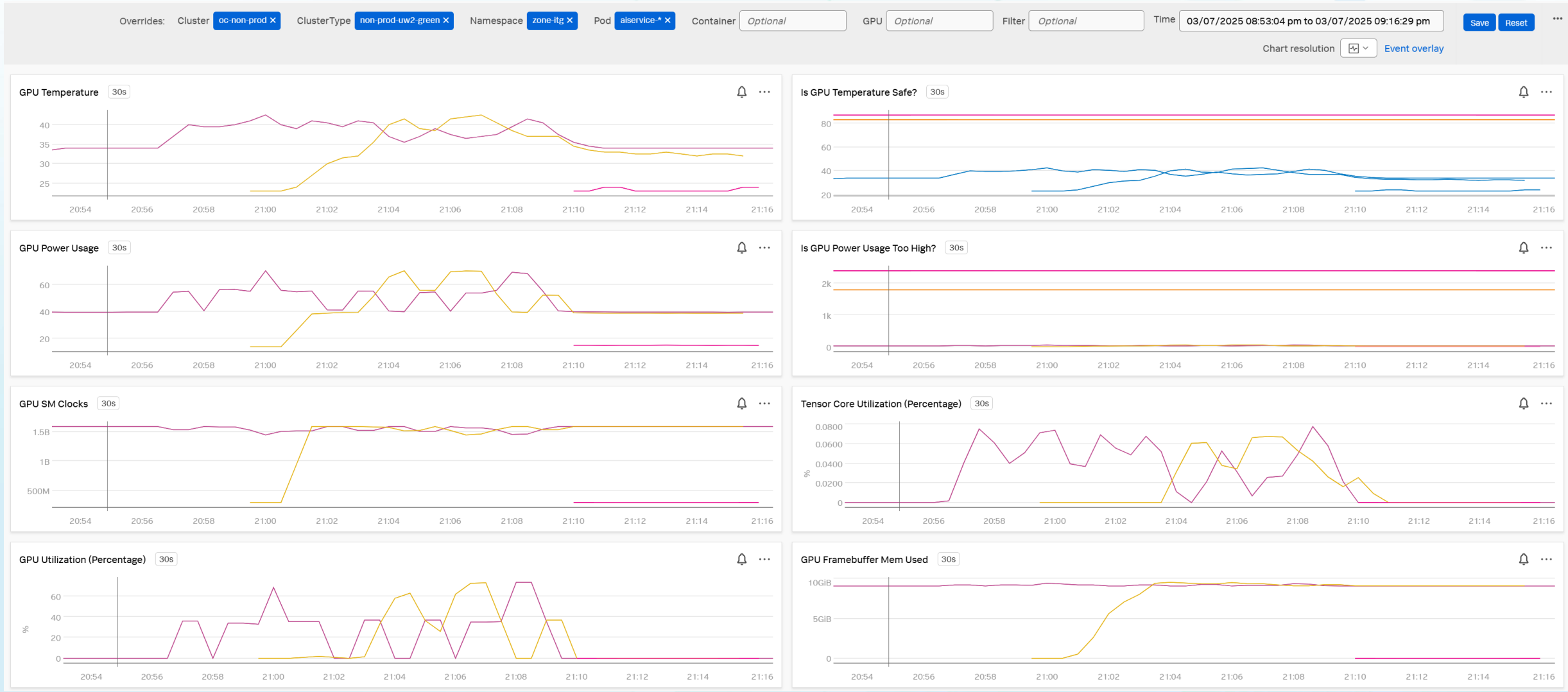
```yaml
apiVersion: serving.kserve.io/v1beta1
kind: InferenceService
metadata:
  annotations:
    {{- if and .Values.metrics .Values.metrics.enabled }}
    prometheus.io/scrape: 'true'
    prometheus.io/port: {{ .Values.metrics.port | default 8082 | quote }}
    prometheus.io/path: {{ .Values.metrics.path | default "/metrics" }}
    {{- end }}
  name: {{ .Release.Name }}
```

Observability

Custom Model - Qwen metrics exposed to Prometheus [Configure]

Observability

GPU metrics got from GPU Operator



KCD
GROWING CLOUD NATIVE TOGETHER
BEIJING

Part 03 - 05

Implementation - API Rate Limit

API Rate Limit

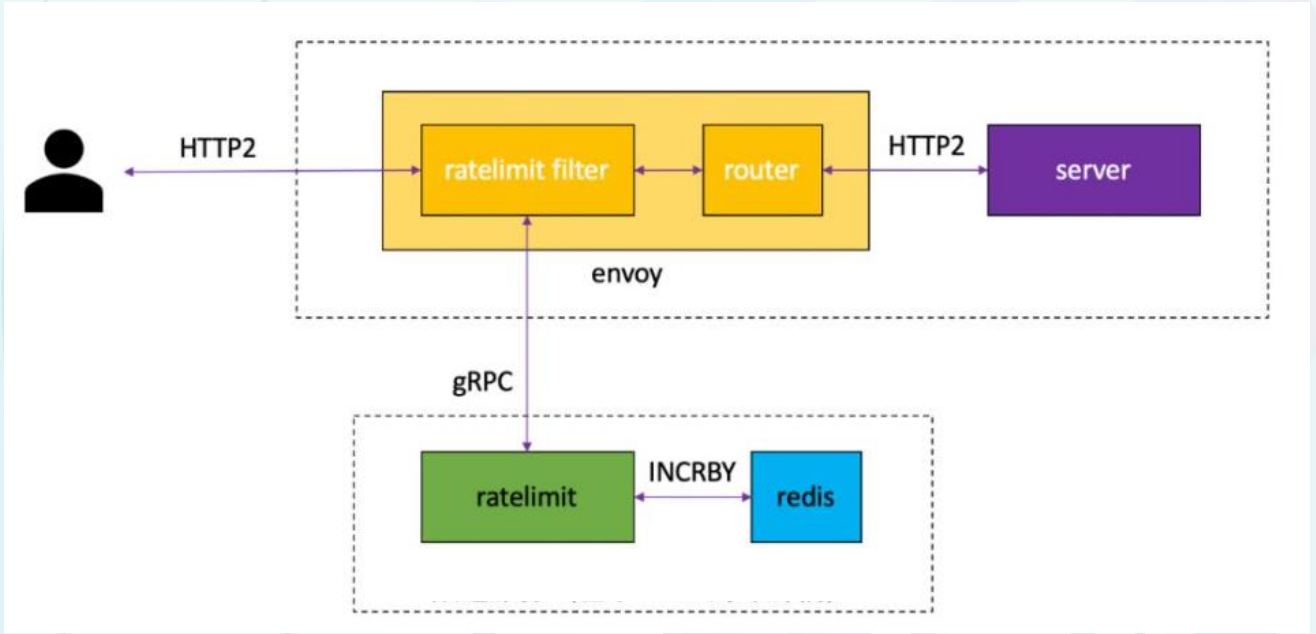
Purpose: Rate limit on API level, no more than number of requests per minute.

- Limit per client IP
- Limit per tenant

User Configure easily:

```
ratelimitIPPerMinute: 10
ratelimitEachSAPerMinute: 100
```

How to implement: [Details]



KCD
GROWING CLOUD NATIVE TOGETHER
BEIJING

Part 03 - 06

LLM Token Limit And Summarize

LLM Token Limit And Summarize

Invoke API: (under token limitation)

curl https://qwen-predictor-default.api.sandbox-uw2.sample.io/v1/chat/completions -H "Content-Type: application/json" -H 'Authorization: Bearer **$token**' -d '{ "model": "Qwen/Qwen2.5-14B-Instruct-GPTQ-Int4", "messages": [{"role": "system", "content": "You are a helpful assistant."}, {"role": "user", "content": "**1 + 1 =?**"}] }'

{"id":"chatcmpl-3634b846-bf56-9c98-ae57-8252df864ff7","object":"chat.completion","created":1740653385,"model":"Qwen/Qwen2.5-14B-Instruct-GPTQ-Int4","choices":[{"index":0,"message":{"role":"assistant","reasoning_content":null,"content":"**1 + 1 = 2**","tool_calls":[]},"logprobs":null,"finish_reason":"stop","stop_reason":null}],"usage":{"**prompt_tokens":25**,"**total_tokens":33**,"**completion_tokens":8**,"prompt_tokens_details":null},"prompt_logprobs":null}

Invoke API: (Over token limitation)

curl https://qwen-predictor-default.api.sandbox-uw2.sample.io/v1/chat/completions -H "Content-Type: application/json" -H 'Authorization: Bearer **$token**' -d '{ "model": "Qwen/Qwen2.5-14B-Instruct-GPTQ-Int4", "messages": [{"role": "system", "content": "You are a helpful assistant."}, {"role": "user", "content": "**1 + 1 =?**"}] }'

< HTTP/2 400
Usage over limit -- serviceAccount + IP.

[Envoyfilter Code]

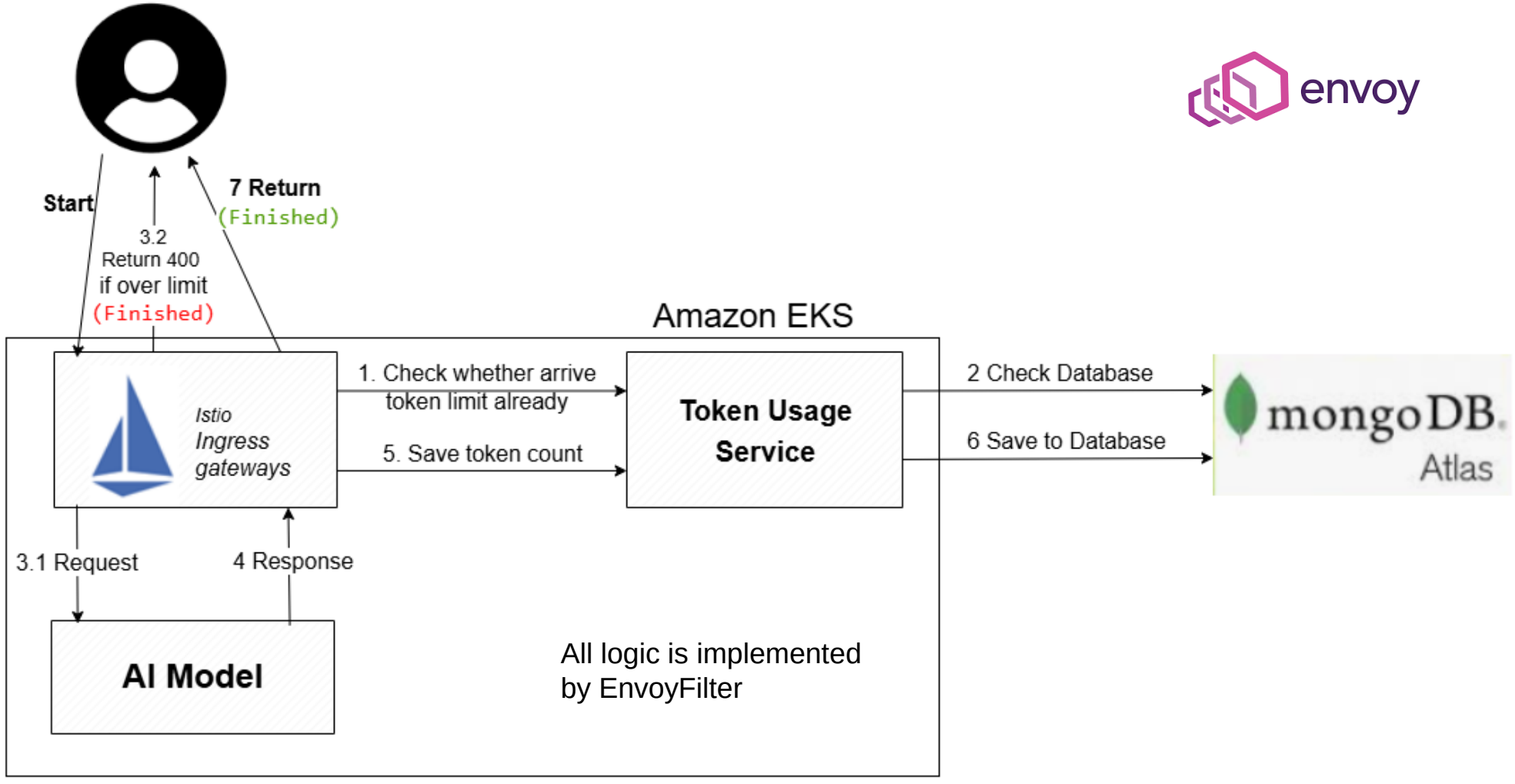
modelToken.tenantLimitation

```
_id: ObjectId('67b9e35ea2876d5f86c7c29b')
sa_limit : 12000
ip_limit : 400
tenant_id : "tenant01"
```

modelToken.sa_ip_usage_daily

```
_id: ObjectId('67c042e4317d13162b21de16')
serviceAccount : "console@zone-prod.hpoc-sa.com"
clientIP : "15.65.196.24"
date : "2025-02-27"
usage : 421
```

LLM Token Limit And Summarize



Envoyfilter Code

KCD
GROWING CLOUD NATIVE TOGETHER
BEIJING

Part 03 - 07

Implementation - Canary

Canary Deployment for Model

Requirements

- Implement a canary deployment strategy based on
  - model versions
  - Image versions

Design

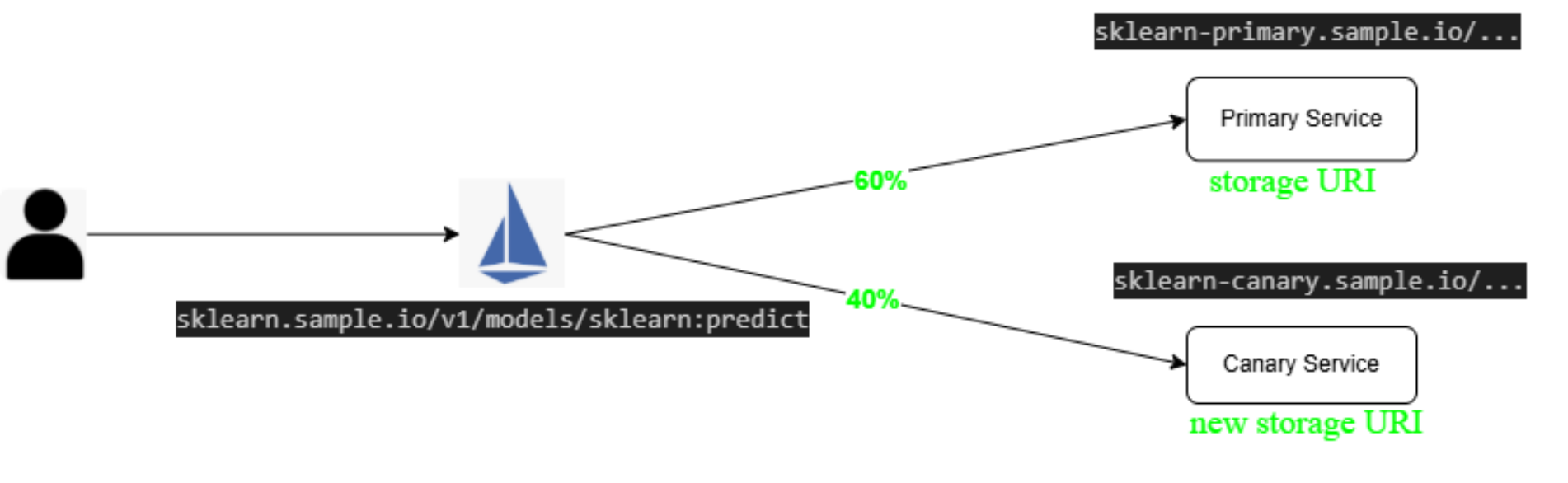
- Create two inference services: one for the primary model and another for the canary model.
- Both services share the same configuration, except `storage Uri` and `container image tag`.
- Use Istio Virtual Service to split traffic between the two inference services.

Canary Deployment for Model

Easy Configure:

```yaml
values:
  primaryLoad: 60   # What percentage of traffic will be forwarded to the primary model?
  inferenceservice:
    predictor:
      model:
        modelFormat:
          name: sklearn
        storageUri: "gs://kfserving-examples/models/sklearn/1.0/model"        # Primary model
        newStorageUri: "gs://kfserving-examples/models/sklearn/1.0/model-2"   # Canary model
```

Achieve:



Canary Deployment for Model

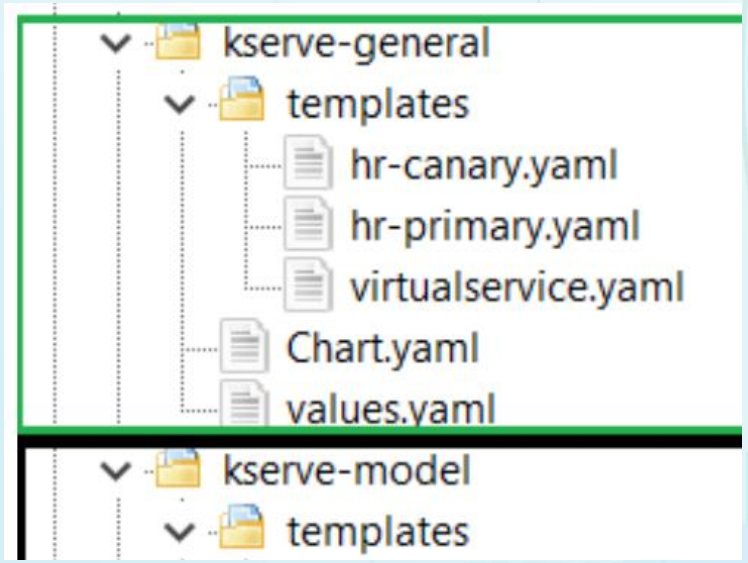
Helm Chart Design： Create kserve-general Chart
[Details]

Traffic Split： Istio Virtual Service
[Details]

Changes of Previous Chart:
[Details]

Flux:
[Details]

Test Result:
[Details]




```yaml
apiVersion: networking.istio.io/v1
kind: VirtualService
metadata:
  name: sklearn-predictor
  namespace: project
spec:
  gateways:
  - istio-system/apigee-gateway
  hosts:
  - sklearn-predictor-project.int.dev-us.sample.io
  http:
  - match:
    - uri:
        regex: ^/.+$
    name: sklearn-predictor
    route:
    - destination:
        host: sklearn-primary-predictor
        port:
          number: 80
      weight: 60  #{{ .Values.primaryLoad }}
    - destination:
        host: sklearn-canary-predictor
        port:
          number: 80
      weight: 40 # {{ sub 100 .Values.primaryLoad }}
```

Summary

Summary

- For AI Model Inference Platform, it is important that model can **obtain features automatically without any coding**, like security, HPA, observability, LLM Token summary, canary release ... We achieved this goal.
- Our solution is **flexible** and can be **easily customized** according to needs.
- Use Helm and Flux to achieve **full automation**. Users can **easily publish** various models through configuration.

Welcome everyone to refer to and implement in the same way [Sample]

Q & A

Email: john.zheng@hp.com
WeChat: johnzhengaz
GitHub: johnzheng1975