

HP云的模型服务自动化实践

惠普DevOps架构师 郑风

Content 目录

01 HP云介绍

02 模型推理平台的需求与设计

03 基于 KServe/Istio/Envoy 的架构实现

- 模型部署
- 模型访问权限控制
- HPA 动态扩展
- 可观测性
- LLM Token 限流和统计
- 金丝雀发布

04 问答环节

Part 01

HP云介绍

AI



HP云介绍



- 惠普云主要架在Amazon上，提供惠普内部项目**所有服务**的部署、监控、运维及管理。

- Kubernetes
- Istio
- Harbor
- Azure Pipeline



AI

- **全方位自动化**

- 基础设施全面实现即代码化 (Infrastructure as Code)
 - Terraform
- 服务自动化部署，项目组可自助完成部署。
 - Helm, Flux2

Part 02

模型推理平台的需求与设计



模型推理平台的需求

- 模型推理需求日益增长，云端部署与管理：
 - 生成式 AI: Llama3、QWen ...
 - 传统机器学习: Scikit-learn、XGBoost
 - 深度学习: TensorFlow Serving、PyTorch ONNX 模型
 - 其他: Hugging Face Transformers
- 模型存储需支持: Hugging Face、S3、PVC、EFS
- 任何项目都能**方便**发布自己的模型推理
- 所有模型推理不用任何额外实现，就**自动拥有** 权限管理，限流，动态扩展，可观测性 等功能

模型推理平台的设计

- KServe 为基础 (不用 Knative)
- 不依赖于 KServe, 自己实现
 - 模型访问权限控制 (Istio)
 - HPA 动态扩展 (Prometheus Adapter)
 - 可观测性
 - LLM Token 限流和统计 (envoyfilter)
 - 金丝雀发布 (Istio)
 - API 限流 (Envoy ratelimit)
 -



Part 03

基于 KServe/Istio/Envoy 的架构实现

- 模型部署
- 模型访问权限控制
- HPA 动态扩展
- 可观察性
- API Rate Limit
- LLM Token 限流和统计
- 金丝雀发布



Part 03 - 01

实现 - 模型部署

AI



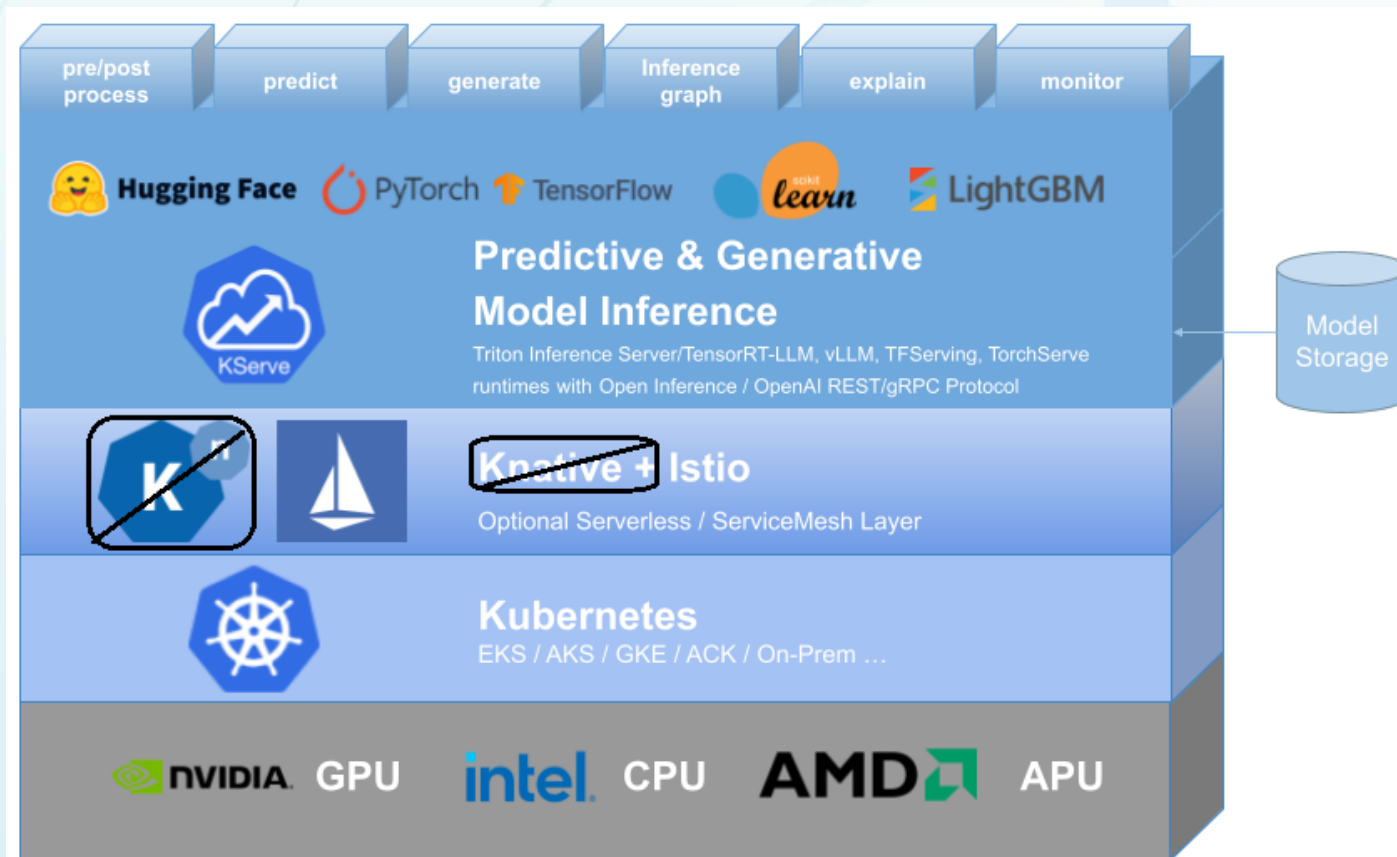
模型部署

KServe: 标准化模型推理平台

- 支持多种模型
- 自动化部署

运行环境: Amazon EKS

无服务架构 Knative 不适合我们, 所以不用

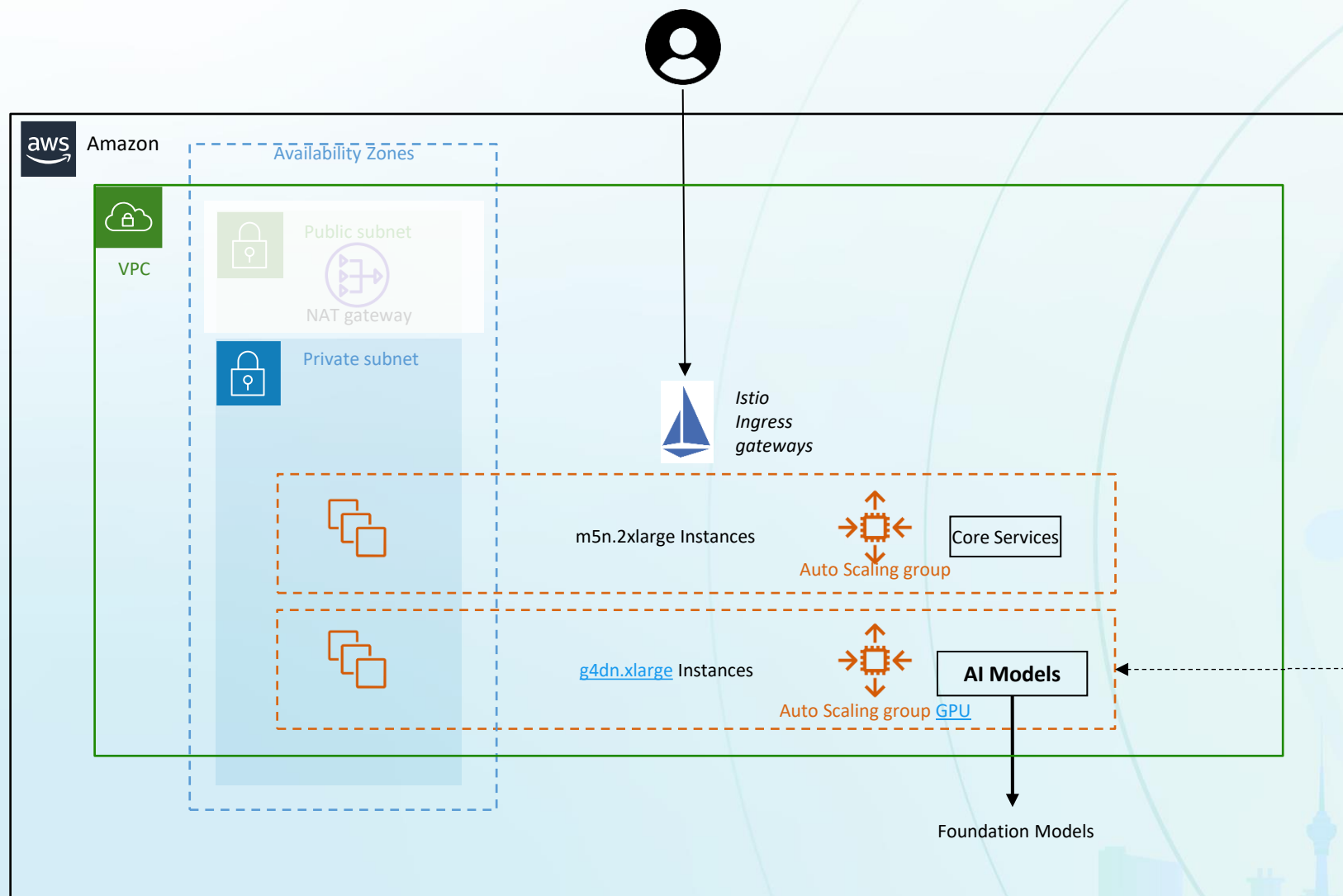


模型部署



- KServe 核心功能：
 - 支持各种推理模型，如： Hugging Face, PyTorch, Scikit-learn
(支持**自定义**推理模型，这样可发布我们自己的模型， 也可以原生使用 vllm 等框架)
 - 支持各种模型存储，如： Hugging Face、S3、PVC、EFS
 - 支持 Model Explainability (模型可解释性)
 - 支持 Multi Model Serving(多模型) 和 Inference Graph(推理 workflow)
- 集成Helm 和 Flux2， 实现自动化部署 [[Sample Code](#)]

支持 GPU Node



Install  [k8s-device-plugin](#)

Add “nodegroup gpu” in **terraform** file:

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

支持 GPU Node

模型如何申请 GPU 资源?



Model owner 在 配置文件定义:

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



```
{{- 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 }}
```



需要 GPU 的 POD
跑在 GPU Node

HelmRelease Value

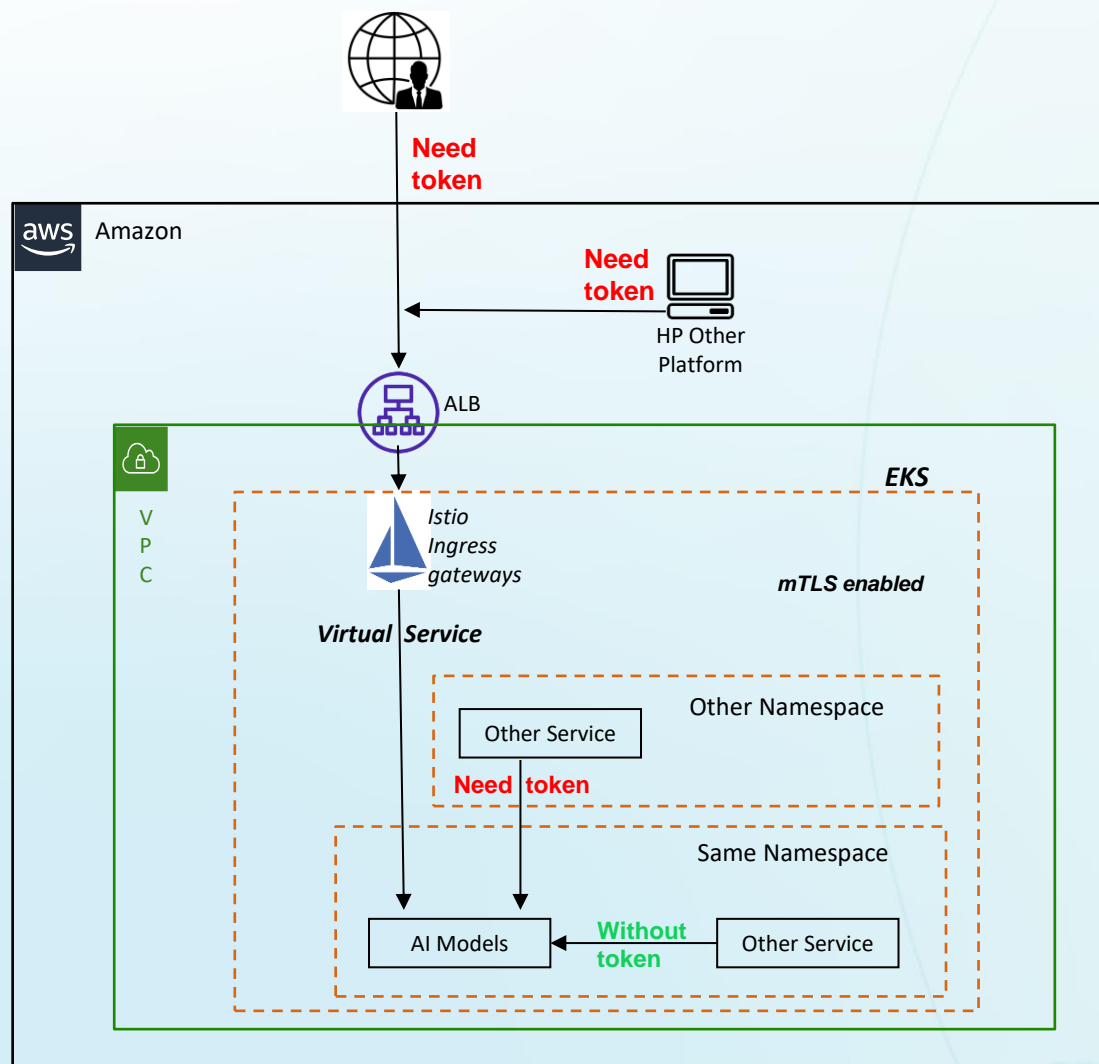
Helm Charts

Part 03 - 02

实现 - 模型服务访问控制 (通过Istio)

AI

模型服务访问控制



```
$ 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

```
{ ... .. }
```

模型服务访问控制

1. 启用 Istio mTLS

[详情链接](#)

2. 通过 Istio Virtual Service 发布模型服务

[详情链接](#)

3. Authentication And Authorization by Istio

- 外部访问需Token
- 同Namespace不需Token
- CUSTOM Authorization 增强保护

[详情链接](#)



AI

Part 03 - 03

实现 - HPA 动态扩展

AI



HPA 动态扩展

需求：支持模型配置 Horizontal Pod Autoscaler (动态扩展)，基于：

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

设计：

- 基于 CPU utilization, Memory utilization 是 HPA 默认支持的
- 使用 Prometheus adapter, 支持 HPA 基于 Custom metrics



Prometheus Adaptor

HPA 动态扩展

- Prometheus adapter 的架构和安装 [详细](#)
- 如何实现 HPA based on "Requests per second"
基于 Istio metrics **istio_requests_total**
[具体例子](#)
- 如何实现 HPA based on "HPA - GPU utilization"
安装 NVIDIA gpu-operator
基于 Nvidia metrics **DCGM_FI_DEV_GPU_UTIL**
[具体例子](#)



AI



HPA 动态扩展

Flux: Install Prometheus adaptor [\[Code\]](#)

```
---
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", container="nvidia-dcgm-exporter"}'
          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 [\[Code\]](#)

```
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
  minReplicas: 1
  maxReplicas: 10
  metrics:
    - type: Pods
      pods:
        metric:
          name: gpu_utilization
          target:
            averageValue: {{ .Values.autoscaling.targetGPUUtilizationPercentage }}
            type: Value
        {{- end }}
    - type: Pods
      pods:
        metric:
          name: requests_per_second
          target:
            averageValue: {{ .Values.autoscaling.targetRequestRate }}
            type: Value
        {{- end }}
```

Helm Release Values [\[Example\]](#)

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


Part 03 - 04

实现 – 可观测性 (Observability)

AI



可观测性

如何实现 AI 模型的可观测性?

- 日志
 - KServe 日志组件
 - Istio Access log
- Metrics
 - Enable KServe metrics for Prometheus
 - Model metrics (vLLM)
 - GPU Metrics
 - Istio Metrics

AI

可观测性

日志

- KServe 日志组件 [\[详细\]](#)

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.

```
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 [\[详细\]](#)
 - Include API host, method, path, response code, duration

可观测性

Metrics – Enable KServe metrics for Prometheus [\[文档\]](#)

所有推理服务都有相应的 metrics, 如:

- KServe 默认支持的模型定义在 [kserve/config/runtimes](#)

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

- [自定义模型](#) (Custom Model) 可以自己定义在 Helm Chart

```
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 }}
```

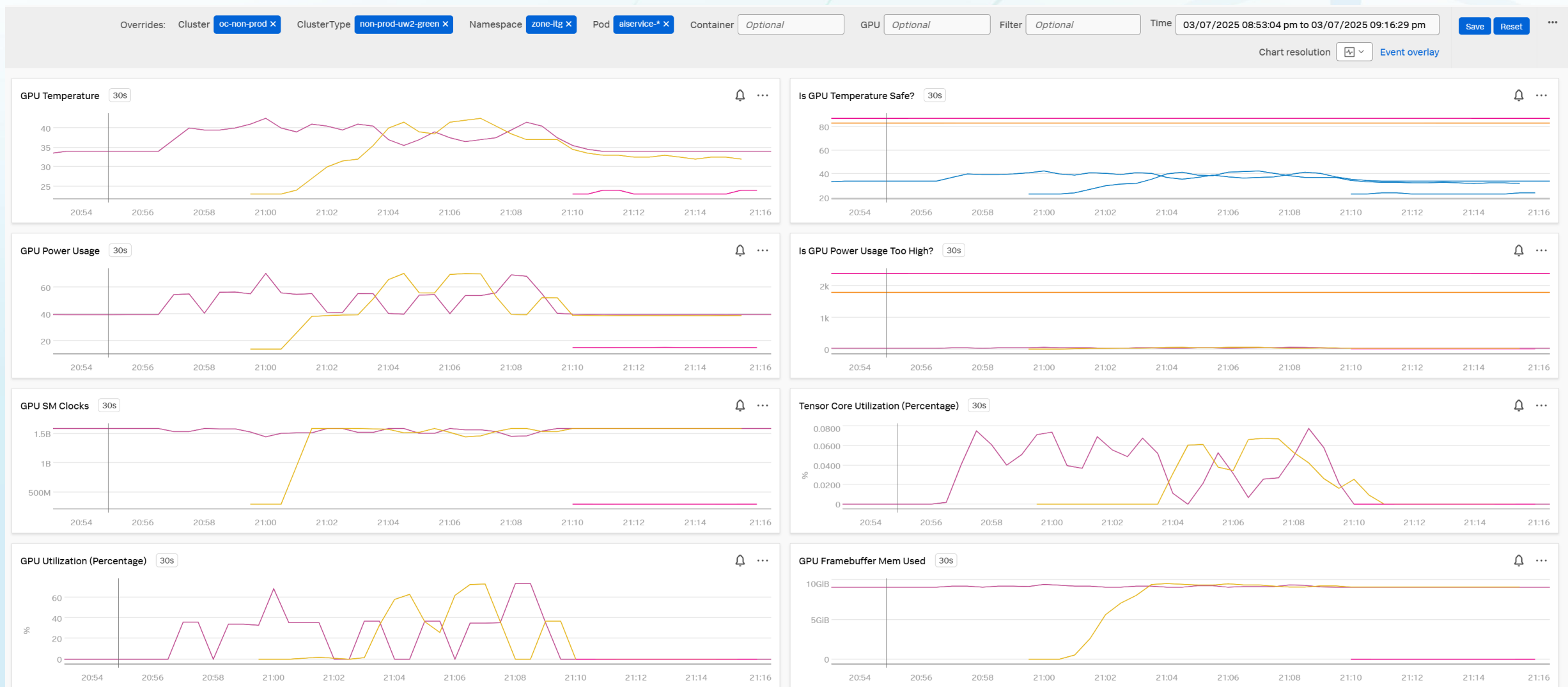
可观测性

例子：Custom Model 干问的 metrics [\[配置\]](#)



可观测性

例子：GPU metrics



Part 03 - 05

实现 – API Rate Limit

AI

API Rate Limit

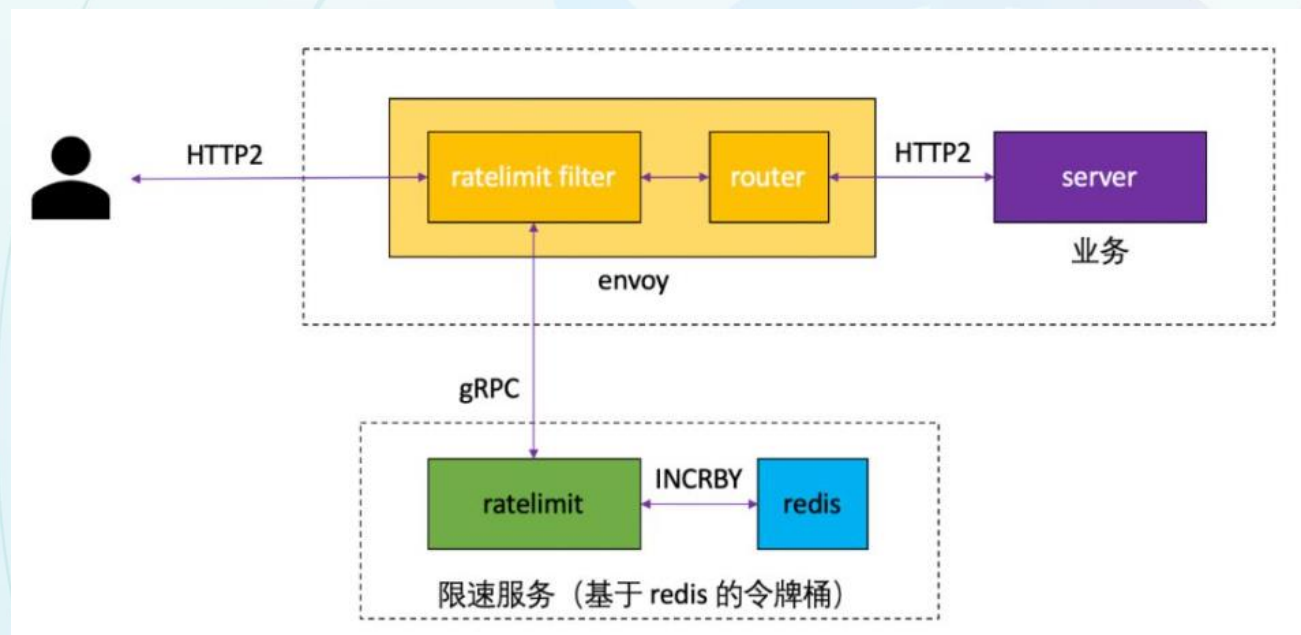
目的： API 级别的限流

- 每一个 IP 限制每分钟的 API 访问量
- 每一个 tenant 限制每分钟的 API 访问量

配置：

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

如何实现： [[详细](#)]



Part 03 - 06

实现 – LLM Token 限流和统计

AI

LLM Token 限流和统计



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 } }, { "index": 1, "message": { "role": "assistant", "reasoning_content": null, "content": "2 + 2 = 4", "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 代码

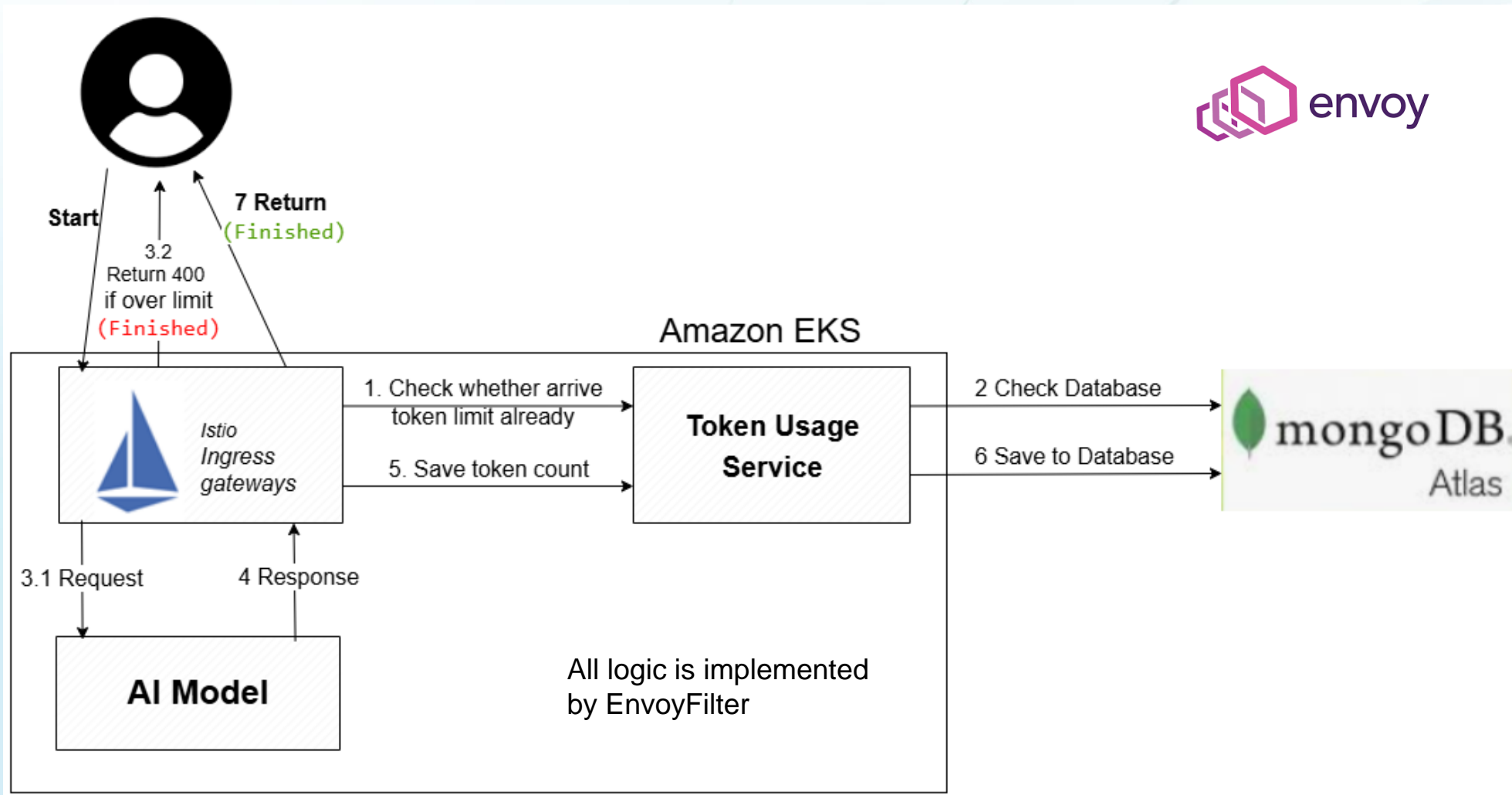
modelToken.tenantLimitation

```
_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 限流和统计



[Envoyfilter 代码](#)

Part 03 - 07

实现 - 金丝雀发布 (Canary)

AI

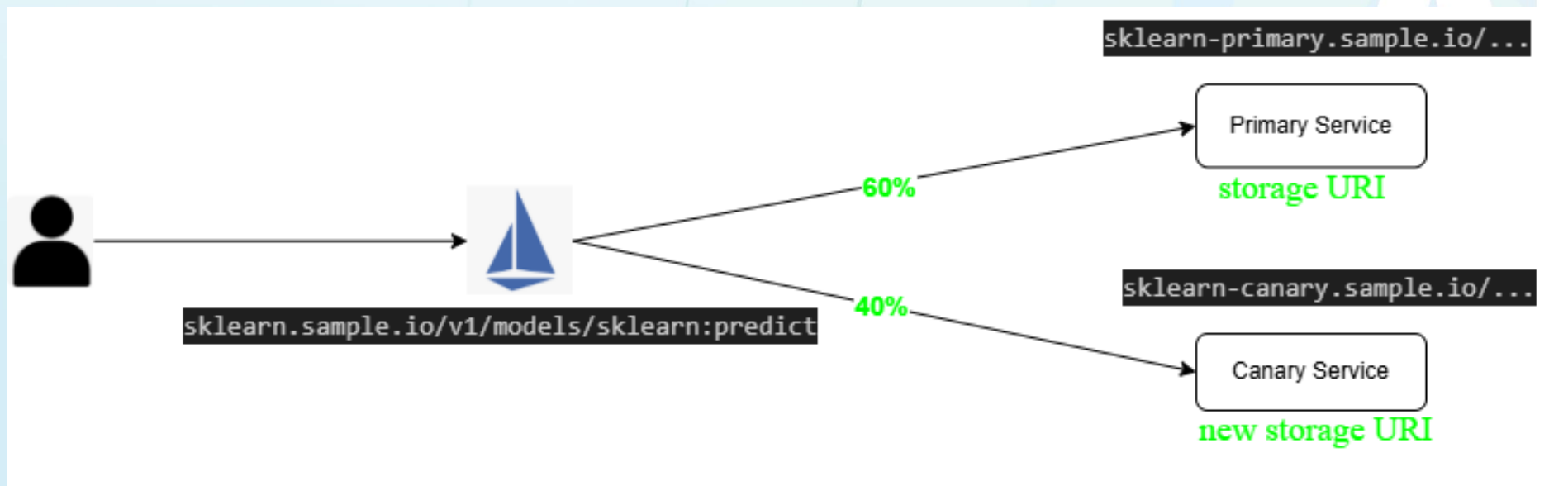
金丝雀发布 (Canary)

需求：与传统的金丝雀发布只注重 container image 不同，模型的金丝雀发布更要注重模型的版本。

设计：创建两个 inference service。

他们所有值，对象都一样，除了 `storage Uri` 和 `container image tag`。

两个 inference service 的流量分配用 Istio virtual service 来实现。



金丝雀发布 (Canary)

Helm Chart 设计: 建立新的 kserve-general Chart

[[详细](#)]

分流的实现: Istio Virtual Service

[[详细](#)]

原 Chart 改动: 兼容直接调用

[[详细](#)]

Flux调用

[[详细](#)]

测试

[[详细](#)]

```
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 }}
```

总结

AI



总结



- 实现统一的模型推理平台，除了选择 KServe 这样的工具发布模型，更要让各个模型自动获得模型访问权限控制，HPA 动态扩展，可观测性，LLM Token 限流和统计，金丝雀发布，API 限流。我们集成 Istio, Envoy, Prometheus Adaptor, GPU Operate, 很好地实现了这些功能。
- 由于是自己实现，非常灵活，可以根据自己的需求方便地定制。
- 利用 Helm, Flux, 实现全方位自动化。用户通过配置，方便地发布各种模型。

AI

实现简单，架构清晰。欢迎大家参考，同样地实现 [[代码](#)]

问答

Email: john.zheng@hp.com

微信: johnzhengaz

GitHub: johnzheng1975

