Cloud Native LLM Gateway

Author:
- dsun20@bloomberg.net
- yweng14@bloomberg.net

Shared with dev@kubernetes.io for commenter
Shared with wg-serving@kubernetes.io for editing

Background

The field of natural language processing (NLP) has undergone a transformative shift with the emergence of Large Language Models(LLM). These sophisticated AI Models, such as GPT, Gemini, Llama, and others have changed the way businesses interact with and process textual data. In addition to the well-known models, new models from organizations and researchers emerge continuously, including smaller-scale LLMs and customized variations addressing specific industry demands.

Within this dynamic landscape, many companies are opting to engage with multiple LLM providers when selecting models for their applications.

- Mitigate the risk of dependency on a single provider
- Access to specialized capabilities
- Comparison across different models and providers to optimize speed, cost-effectiveness, accuracy, and scalability
- Using multiple providers, a company can stay agile and adapt to changes in technology and market demands
- Redundancy and failover: company can minimize the downtime, ensuring uninterrupted access to the LLM capabilities.

A LLM gateway can act as a critical intermediary with a single unified API, channeling requests to the LLMs providers. It can also perform essential post-processing, and safety checks of the LLM interactions. The primary advantages of deploying a robust LLM gateway includes:

- Streamlined integrations to diverse LLM providers with unified API without needing user to learn the manage each provider's idiosyncrasies
- Improved model accessibility, registering a new LLM model without significant operational overhead with a plugable gateway architecture
- Scalable and efficient operations with built-in resilience and minimized downtime
- Strengthened security measures to ensure that sensitive information is securely controlled before it leaves the customer's environment

- LLM gateway can track performance and operational metrics for each deployed model with centralized LLM monitoring for response accuracy, latency, throughput and cost.

Existing Landscape

By researching the existing landscape for LLM gateway, there are a few interesting projects:

LiteLLM:

https://www.litellm.ai/ is a python-based LLM gateway that is easy to use and stands up locally.

Pros:

- The project is lightweight, and friendly to Python developers.
- Supports a rich set of features.

Cons:

- Any configuration changes requires redeploying the Gateway, such as routing and model configurations.
- Features ~~like~~ SSO, JWT Auth, and guardrail integrations are under the enterprise license.
- Dedicated slack support is under enterprise license.

Kong AI Gateway:

https://konghq.com/products/kong-ai-gateway is an enterprise AI gateway which extends the traditional API gateway by implementing the additional AI plugins such as rate limiting, request/response transformation and prompt templating.

Pros:

- AI features can be implemented via gateway plugins
- Supports a rich set of features.
- APIs are both declarative(YAML) and imperative(REST API)

Cons:

- Advanced features like rate limiting and content safety are under the enterprise license.
- Unified API incompatible with OpenAI schema.
- Limited set of providers supported - no bedrock/sagemaker/google ai studio/vertex ai/databricks/etc.
- Limited set of parameters supported - no tool calling, vision processing, etc. - Link
- Cannot load balance multiple deployments of same model

- Cannot setup fallbacks across providers

Relationship to Kubernetes Ecosystem

- Envoy proxy is commonly used for implementing the API gateway which provides features like load balancing, resilience and rate limiting. Service discovery is a key component of envoy which uses a layered set of dynamic configuration APIs. The layers provide dynamic updates such as host information, backend clusters, listening sockets, HTTP routing, and cryptographic items. These features are critical for implementing the LLM gateway to allow adding new LLM models or configurations without interruptions.
- Projects like Envoy Gateway can be leveraged to implement LLM gateway on top of with its pluggable architecture to implement AI plugins and at the same time enjoy all the benefits gateway API provides.
- The LLM gateway can be deployed in the Kubernetes environment to leverage all the benefits Kubernetes provides but should not be strictly required.

Goals:

- Define the scope of the functionalities of LLM gateway
- Align with the Kubernetes ecosystem for what can be leveraged to build LLM gateway with a pluggable architecture to support the following personas.

Non-Goals:

- Fine-tuned models with LoRA

Personas:

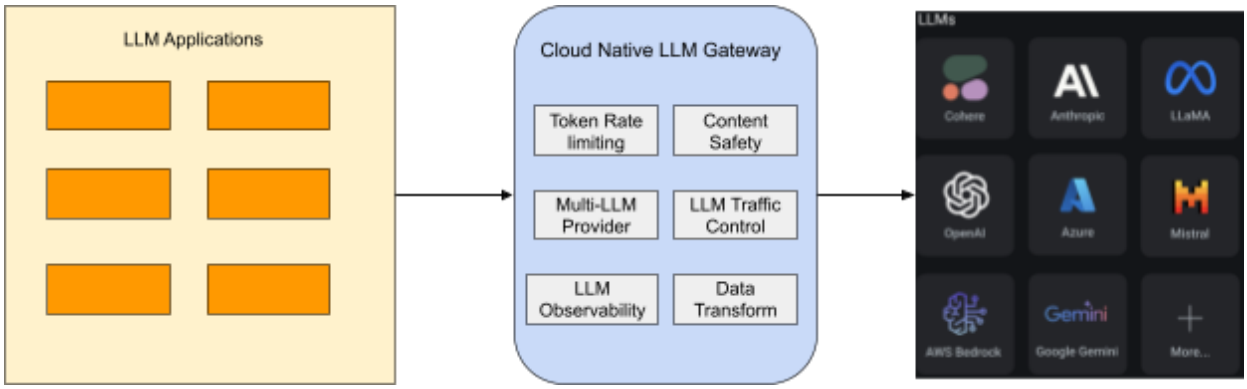
We define 3 main personas that will interact with the LLM Gateway:

1. **Inference platform admin (e.g., a customer's MLOps team)**: wants to manage sets of LLM serving workloads efficiently.
2. **Payments teams:** needs to track per user/tenant model-token usage for billing purposes.
3. **Safety teams:** needs to be able to design, experiment, and enforce new content safety policies uniformly across all

Requirements

Compared to the traditional API gateway such as [envoy gateway](), LLM gateway requires AI specific features like token based rate limiting, prompt data transformation and LLM provider routing. The traditional API gateway has already implemented most of the basic features, implementing the LLM gateway from scratch is going to reinvent a lot of the wheels, hence we are proposing a way to extend the existing API gateway and add the AI specific features on top.

Envoy can be extended by using proxy-wasm sdk to implement the AI plugins such as request transformation, metrics collection.
https://github.com/tetratelabs/proxy-wasm-go-sdk



Multi-LLM Provider

LLM from different providers may have different model architectures and compatibility requirements. LLM gateway can act as a centralized infrastructure component, providing abstraction layers and translation services to facilitate interoperability among various LLM providers.

The core of the LLM gateway is the ability to route LLM requests to various providers with a unified API without needing the end user to learn each provider's idiosyncrasies .

- LLM applications are shielded from LLM provider API specifics, promoting code reusability.
- LLM gateway can give organization a central point of governance and observability over LLM data and usage.

Each provider can have following configuration and can be defined as a custom resource “LLMBackend”:

- Provider name: openai, azure, anthropic, cohere, mistral, kserve
- Supported API endpoints: v1/chat/completions v1/completions
- Auth Configurations: e.g azure managed identity, OpenAI API key
- Model name
- Options: streaming, upstream url/path
- Logging: token statistics or payload

```
kind: LLMBackend
metadata:
  name: kserve-mistral
spec:
  provider: kserve
  model: mistral
  format: openai
  streaming: true
  logging:
    log_token_statistics: true
    log_payload: true
  upstream_url: https://mistral.kserve.io/v1/chat/completions


kind: LLMBackend
metadata:
  name: azure-gpt
spec:
  provider: azure
  model: "gpt-35-turbo"
  format: openai
  streaming: true
  AuthOptions:
    azure_instance:
    azure_deployment_id:
  logging:
    log_token_statistics: true
    log_payload: false
```

Traffic Management

Traffic management involves the strategic routing, filtering and monitoring the request and response between LLM applications and downstream LLM providers/APIs such as OpenAI, Llama, AWS Bedrock or Anthropic.

The LLMRoute resource allows users to configure the LLM routing by matching the HTTP traffic and forwarding to the corresponding LLM backend. It should work the same way as Kubernetes Gateway API that the route configurations are dynamically updated to the LLM gateway without having to restart the gateway.

- Load balancing across multiple model providers
    - Load balancing across multiple instances from different providers for the same model
        - "simple-shuffle", "least-busy", "usage-based-routing", "latency-based-routing"

```
apiVersion: gateway.networking.k8s.io/v1
kind: HTTPRoute
metadata:
   name: mistral-routing
spec:
   rules:
   - matches:
     - path:
        type: prefix
        Value: /mistral
     backendRefs:
     - group: serving.kserve.io/v1alpha1
       kind: LLMBackend
       name: kserve-mistral
     - group: serving.kserve.io/v1alpha1
       kind: LLMBackend
       name: bedrock-mistral

apiVersion: gateway.networking.k8s.io/v1
kind: BackendTrafficPolicy
metadata:
   name: mistral-lb
spec:
   targetRef:
```

```
    group: gateway.networking.k8s.io/v1
    kind: HTTPRoute
    name: mistral-routing
  loadBalancer:
    type: leastRequest   #usage-based-routing, latency-based-routing
```

- Fallback, timeout and retries across multiple model providers
    - Fallback to a different provider if the call fails after a number of retries

```
apiVersion: gateway.networking.k8s.io/v1
kind: HTTPRoute
metadata:
  name:  mistral-routing
spec:
  rules:
  - matches:
    - path:
       type: prefix
       value: /mistral
    backendRefs:
    - group: serving.kserve.io/v1alpha1
      kind: LLMBackend
      name: kserve-mistral

apiVersion: gateway.networking.k8s.io/v1
kind: BackendTrafficPolicy
metadata:
  name:  mistral-retry
spec:
  targetRef:
    group: gateway.networking.k8s.io/v1
    kind: HTTPRoute
    name: mistral-routing
  retry:
    numRetries: 5
    retryOn:
      httpStatusCode:
      - 429
    fallback:
      backendRefs:
      - group: serving.kserve.io/v1alpha1
        kind: LLMRoute
```

```
        name: bedrock-mistral
```

- Advanced routing strategies:
    - Rate limiting aware
        - filter out the deployment if tpm or rpm is exceeded
        - Route to the deployment with lowest tpm or rpm

Rate Limiting

Rate limiting is a feature that allows the user to limit the number of tokens in the incoming LLM requests to a predefined value to prevent DDoS attacks and prevent the LLM providers from getting overloaded. The LLM gateway can support both request based rate limiting and token based rate limiting.

For token based rate limiting LLM gateway uses the following statistics to calculate the token metrics.

- Total Tokens: The total number of tokens including both prompt and generated completion
- Prompt token: The tokens provided by the user as input to the LLM
- Completion tokens: The tokens generated by the LLM in response to the prompt

The configured rate limiting:

```
apiVersion: gateway.networking.k8s.io/v1
kind: BackendTrafficPolicy
metadata:
    name: rate-limiting-mistral
spec:
   targetRef:
     group: serving.kserve.io/v1alpha1
     kind: LLMRoute
     name: mistral-routing
   rateLimit:
     type: Global
     global:
       rules:
       - clientSelectors:
         - headers:
           - name: x-claim-name
             value: John Doe
         limit:
           tokens: 300
```

```
            unit: Minute
```

When rate limiting is enabled, LLM gateway sends additional headers back to the LLM application indicating the allowed limits, how many requests are available and how long it will take until the quota is restored.

```
X-RateLimit-Limit-30-azure: 1000
X-RateLimit-Remaining-30-azure: 950
```

If any of the limits configured have been reached, LLM gateway returns an `HTTP/1.1 429` status code to the client with the following JSON body:

```
{ "message": "API rate limit exceeded for provider azure, cohere" }
```

Metrics

The LLM gateway should produce standard LLM latency, token metrics as documented in [External] Standardizing Large Model Server Metrics in Kubernetes

total_tokens{reporter="llm-gateway", provider="mistral", model="mistral-tiny", temperature="0.2"}

LLM Gateways in the Ecosystem

A number of groups have already begun either implementing LLM Gateways by extending the Gateway API or have their own internal gateways. They are listed below for reference. If you know of others, please add them to the list.

- Gloo AI Gateway
- Kong AI Gateway
- LiteLLM
- Cloudflare AI Gateway
- IBM AI Gateway
- MLFlow AI Gateway
- AI Gateway
- PortKey AI Gateway

- Google AI Gateway
- Azure GenAI Gateway
- AWS AI Gateway (more a use case)

Open Questions:

Is the goal of this proposal to define a set of standards that are resources for building an LLM Gateway API, of which there can be many compliant controller implementations, similar to the Gateway API for Ingress, or is it to extend an existing gateway implementation, such as Envoy Gateway, to support the features defined in this doc?

These are not necessarily mutually exclusive goals. However, I think we should be very clear about what we are proposing and in what order, as they could be concurrent or sequential efforts if we decide both make sense. Personally, I think we should drive them in parallel.

Creating an initial prototype implementation using Envoy Proxy would:
- help ground our thinking for what a more general standard might look like
- come together much faster than designing and building consensus around a standard; thus, allowing us to meet *some* user's needs sooner, assuming there's actually a strong signal for this at the moment.

Designing an LLM Gateway API standard would:
- open the door for more implementations as Envoy only serves a subset of users
- allow for continued requirements gathering from users and input from the relevant SIGs (network, others?)

Supposing we decide to put energy into the "standard setting" route, then we should liaise with the Gateway API SIG. They are about to embark on a post-mortem to gather lessons learned from building the Gateway API and then synthesize those into an initial "Best practices for how to build an official, yet out-of-tree, 'Kubernetes Certified Extension'." We'd be remiss not to tap such wisdom before barreling ahead.

Meeting Notes:

June 26, 2024
- Initial discussion in the Envoy gateway community meeting
  - Envoy Gateway Community Meeting
  - Extend envoy gateway to implement AI specific features such as token based rate limiting, payload transformation, prompt control. Suggested to create a repository for common AI plugins.

        - https://gateway.envoyproxy.io/contributions/design/envoy-extension-policy/
        - https://gateway.envoyproxy.io/latest/tasks/extensibility/ext-proc/
    - Advanced routing features such as fallback, latency based routing can be implemented in envoy gateway itself.