Platform Engineering for LLMs: A Practical Guide to Running Your Own AI Stack

DevOpsDays: AI Chapter

DEVOPSDAYS UKRAINE

#STAND WITH #UKRAINE

Intro

Max Körbächer - Founder, Tech Advisor and Managing Director

@ Liquid Reply

We create platforms, no matter where, for no matter what.

With a focus on Platform Engineering, Internal Developer Platforms & Cloud Native Engineering

- Author of "Platform Engineering for Architects"
- CNCF TAG Environmental Sustainability Initiator & Emiritus
- CNCF Ambassador
- LF Europe Advisory Board

THE LINUX FOUNDATION

AMBASSADOR
CLOUD NATIVE COMPUTING FOUNDATION
SPRING 2023

TAG ENVIRONMENTAL SUSTAINABILITY

Platform vs. Ad-hoc Approach

ca. 67%

Platform Success

Production success rate with structured platforms

ca. 31%

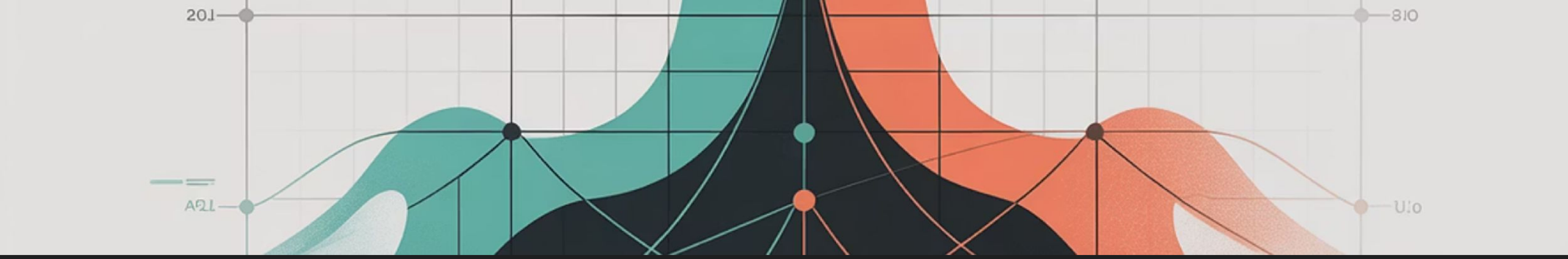
Ad-hoc Success

Production success rate with ad-hoc implementations

70-80%

AI Projects Fail

AI projects don't reach production even at experienced organizations



Why Build Your Own LLM Platform?

Data Privacy & Compliance

Maintain complete control over sensitive data. Keep proprietary information within your security boundaries and meet regulatory requirements like GDPR, HIPAA, and industry-specific regulations.

Cost Control at Scale

Avoid unpredictable API costs that scale linearly with usage. After initial investment, internal infrastructure can be more cost-effective at high volumes, with better visibility into spending.

Customization Flexibility

Fine-tune models on domain-specific data and develop specialized architectures tailored to your use cases. Deploy exactly the model sizes and configurations you need.

Performance Optimization

Minimize latency for real-time applications. Ensure consistent availability without dependency on third-party services, and implement custom optimizations for your specific workloads.

The Platform Engineering Approach

Platform as a Product Mindset

Treat your LLM infrastructure as a product with internal users, clear value propositions, and a roadmap driven by user needs.

Developer Self-Service

Enable ML engineers to deploy, experiment, and iterate without infrastructure bottlenecks or waiting for ops teams.

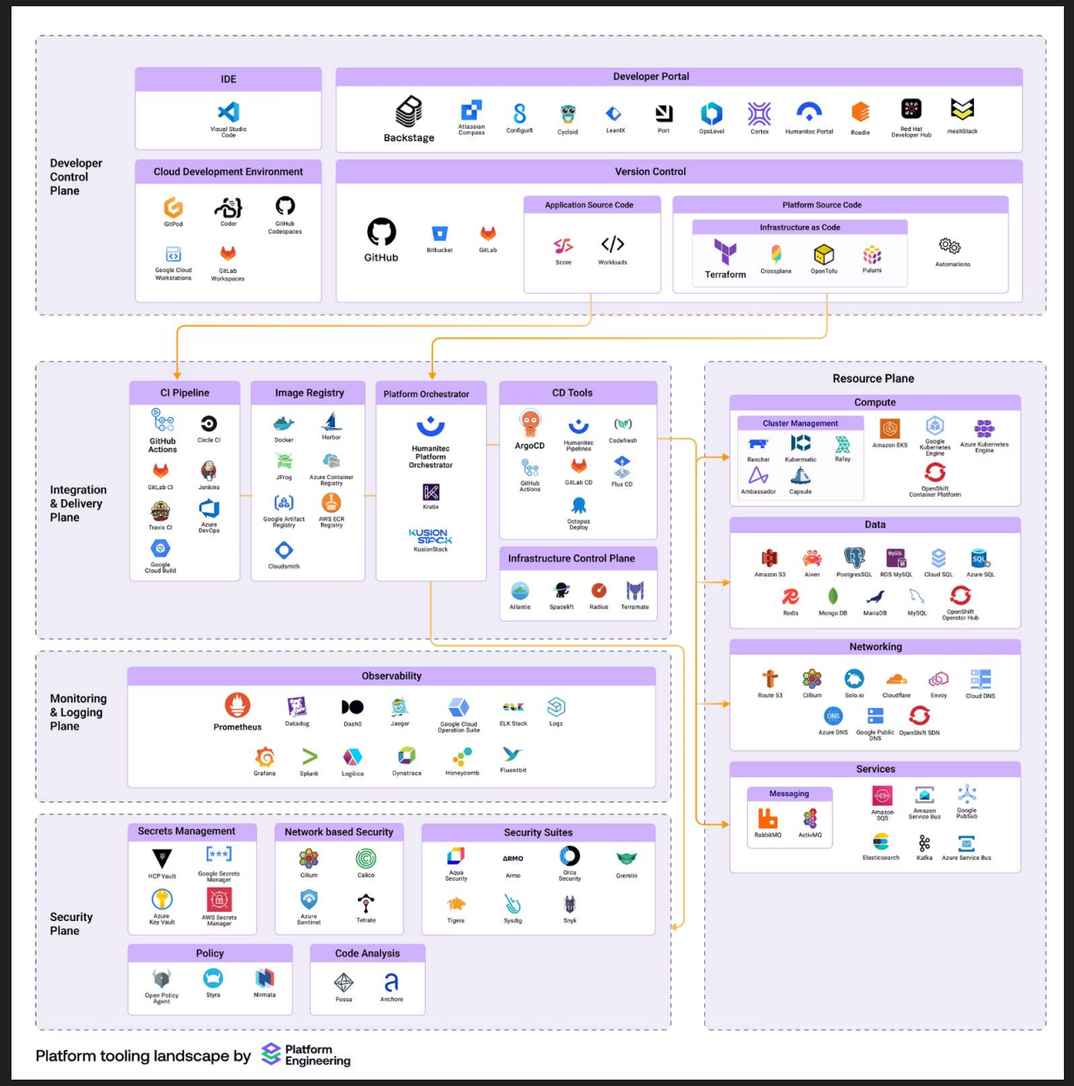
Abstraction of Complexity

Hide infrastructure details behind intuitive APIs and interfaces. Developers shouldn't need to understand GPU optimization to use LLMs effectively.

Standardization & Governance

Establish consistent patterns, security controls, and observability across all AI workloads.

Platform Engineering Reference Architecture



A good platform comes with all batteries included needed to train, manage and serve LLMs, AI, and GenAI.

Platform Hidden Services

Prompt Caching

Up to 85% latency reduction for common queries

Dynamic Batching

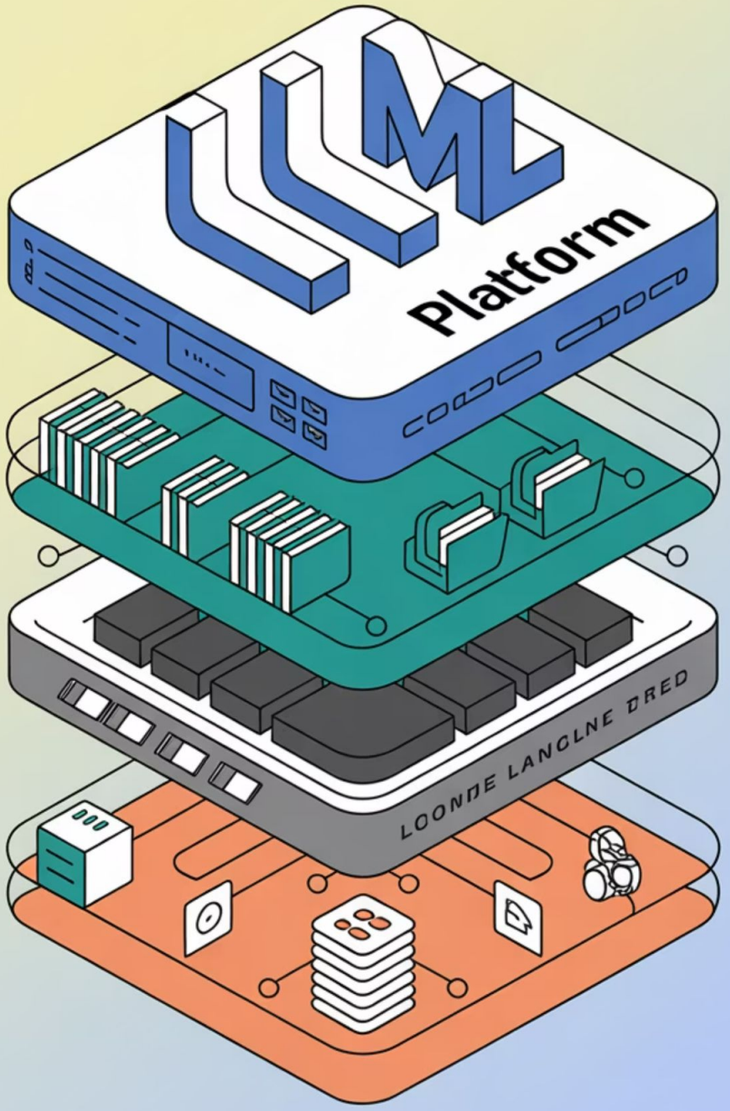
Optimized throughput for high-volume scenarios

Intelligent Routing

Model selection based on complexity and cost

Core Architecture

Building blocks for successful LLM platforms



LLM Platform Architecture Overview

1

Compute Layer

- GPU cluster orchestration (Kubernetes, Slurm)
- Hardware acceleration management
- Resource scheduling and allocation
- Network optimization for distributed training

2

Model Management Layer

- **Model versioning and registry**
- **Training pipeline automation**
- **Artifact storage and caching**
- **Metadata tracking and lineage**

3

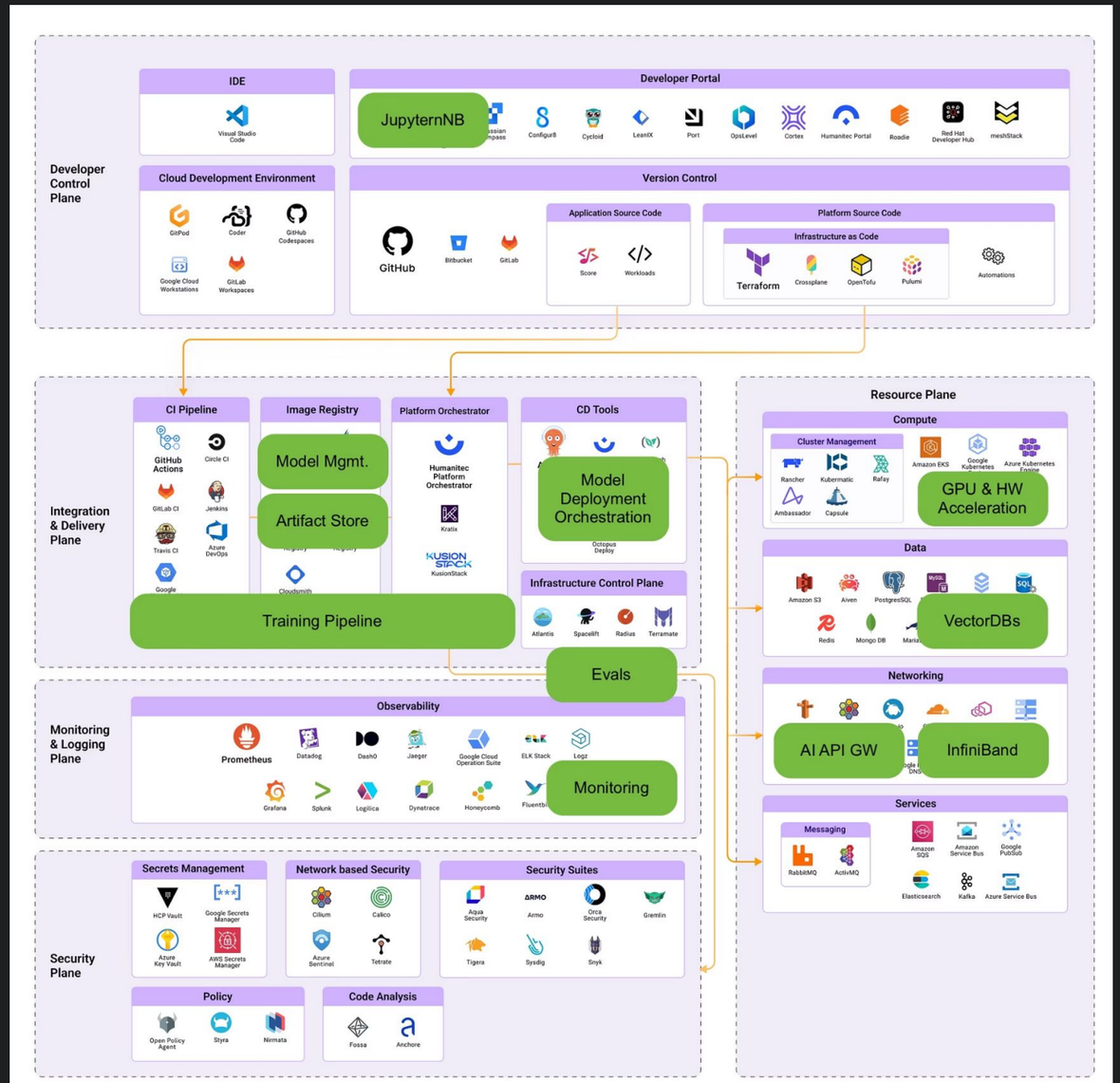
Serving Layer

- Inference API standardization
- Load balancing and scaling
- Request routing and prioritization
- Model deployment orchestration

4

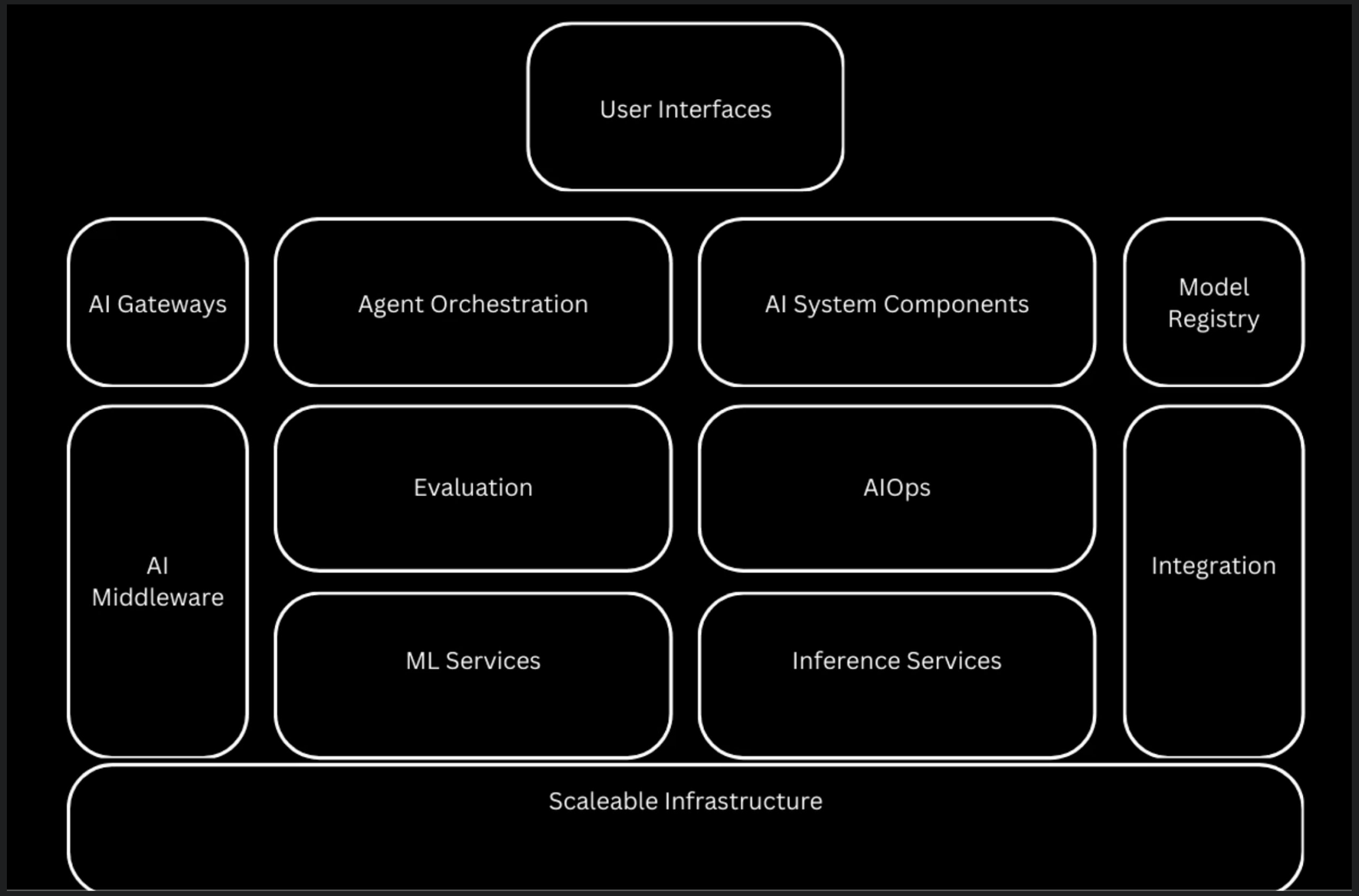
Developer Interface Layer

- Self-service portals and UIs
- SDK/API integration with tools
- Monitoring dashboards
- Documentation and examples



Platform tooling landscape by Platform Engineering

The AI SIX PACK

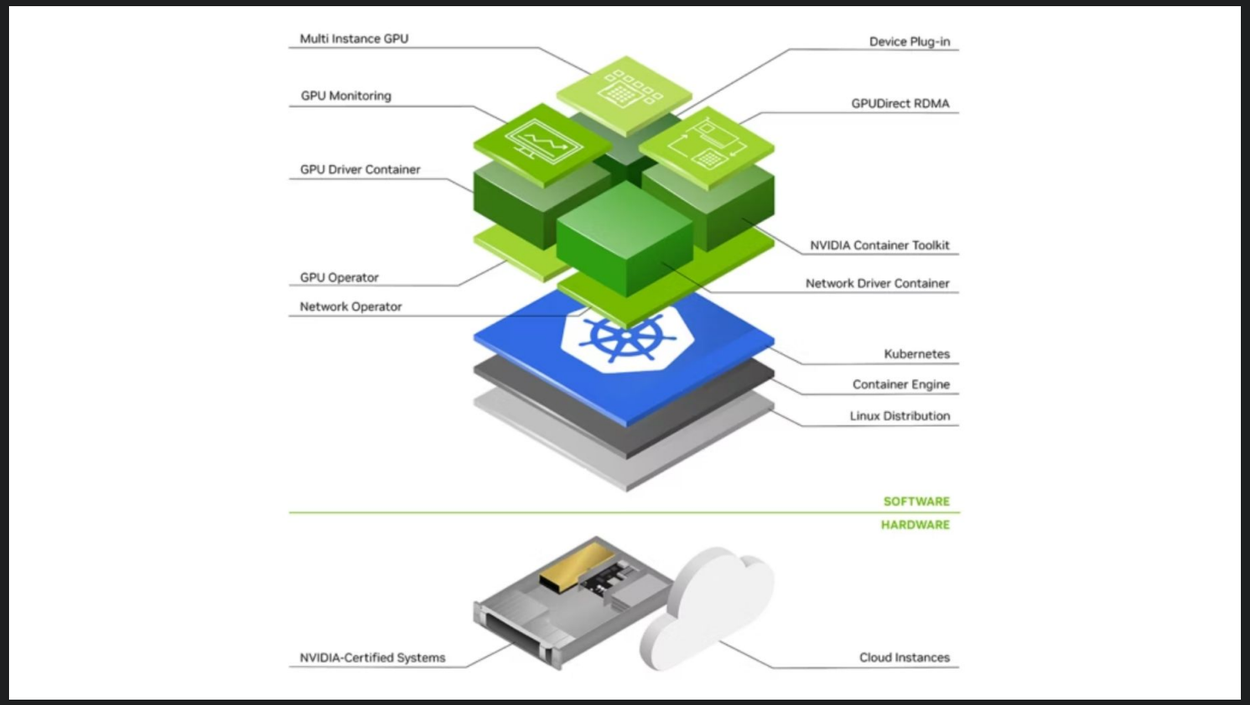


Why K8s?

>90%

Kubernetes Adoption

Enterprises using K8s for AI workloads



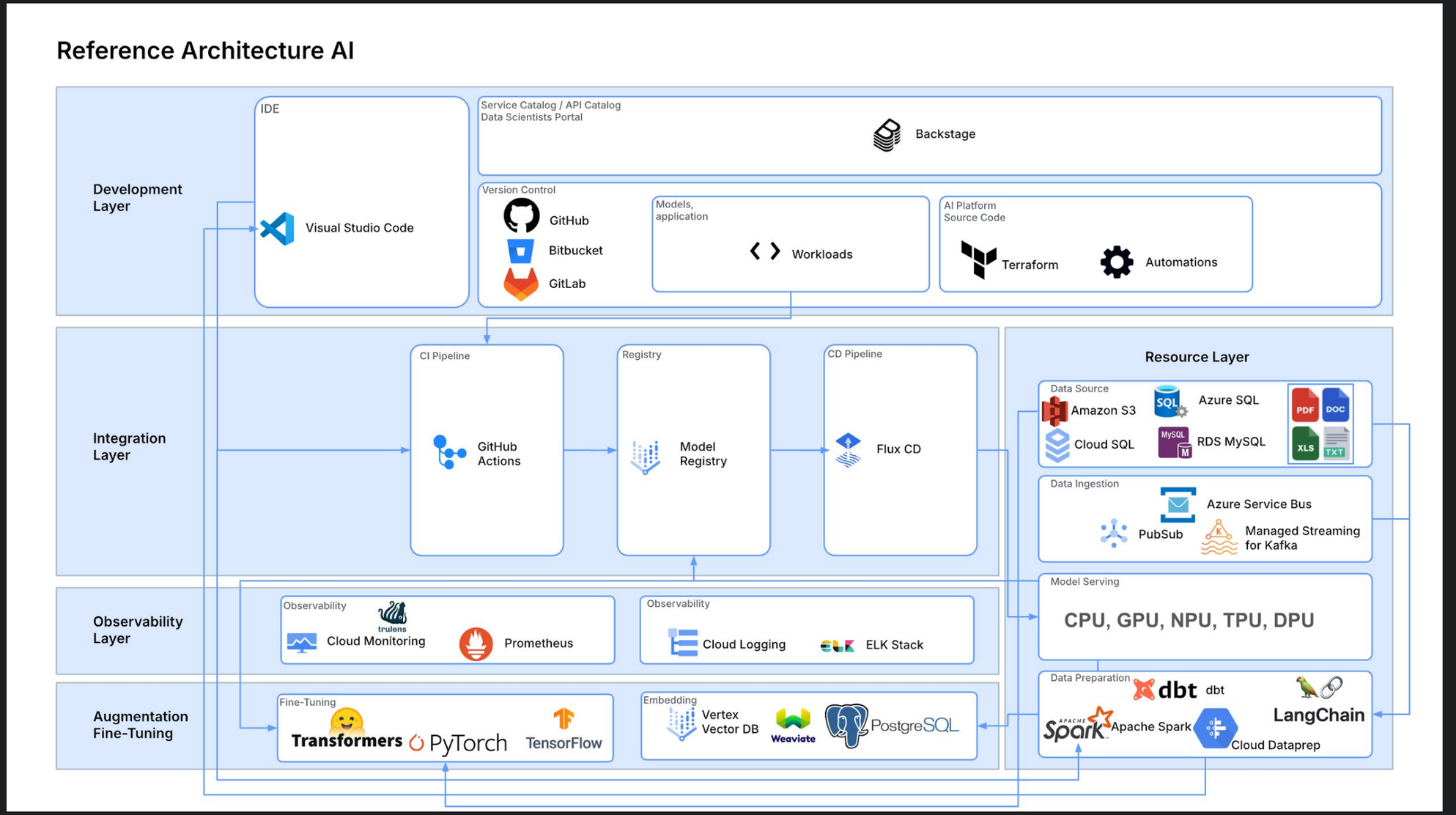
~70%

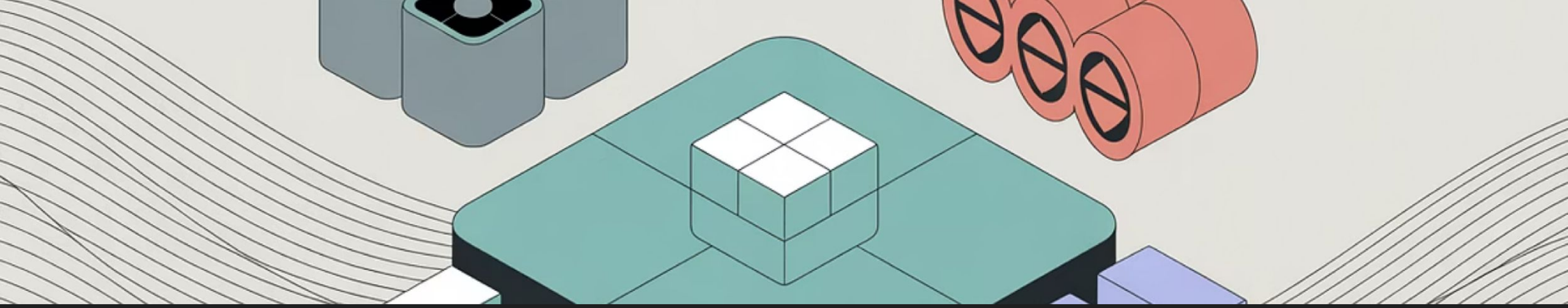
Hybrid Deployments

AI infrastructure spanning multiple environments

1. Easily port services
2. Easily scale services
3. Manage drivers and shared libraries
4. Manage GPU configuration
5. GPU monitoring and telemetry
6. Your non-GPU workloads are already on

Simple Example Architecture

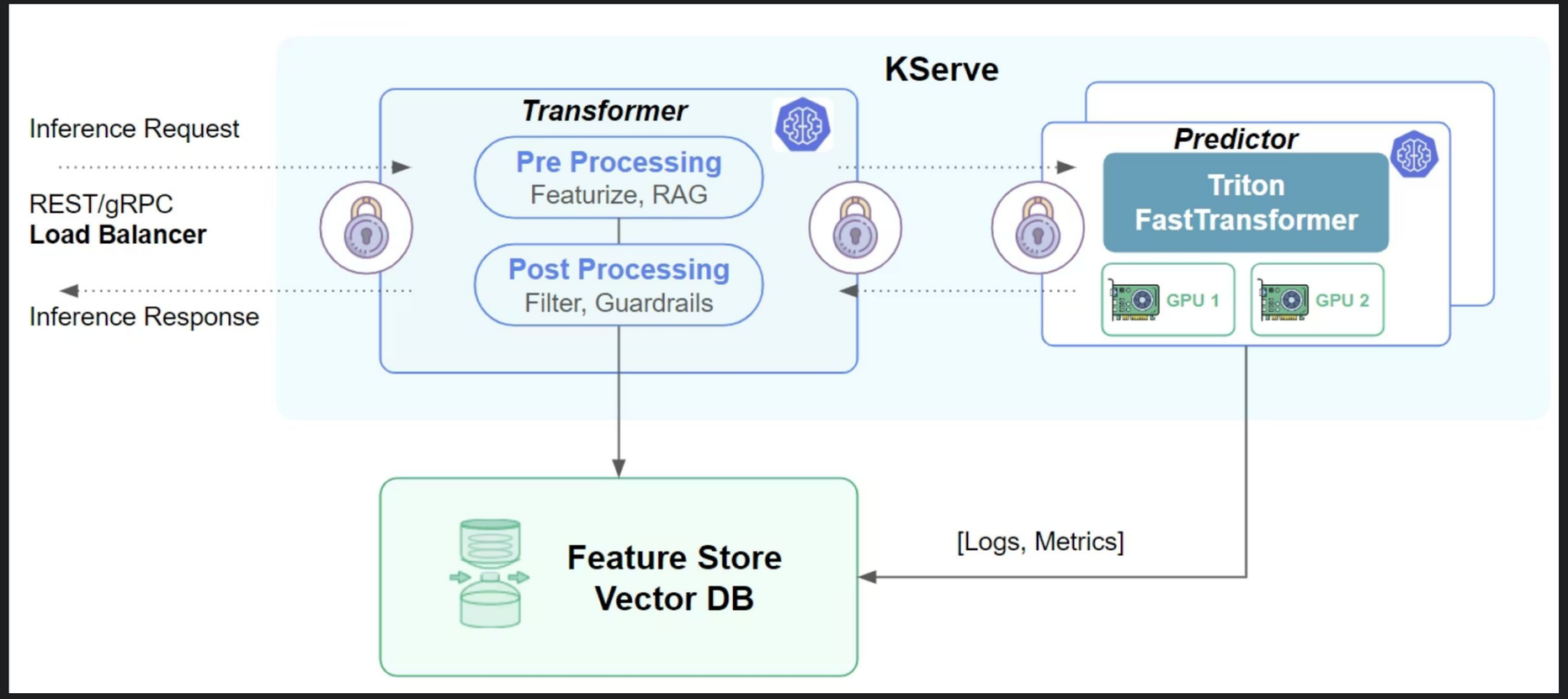


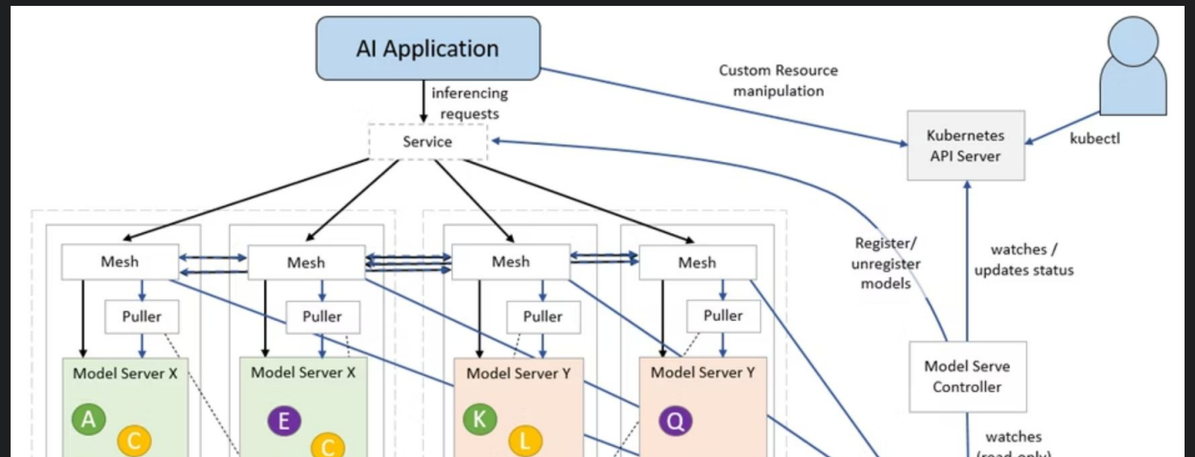


Critical Parts of a LLM Platform

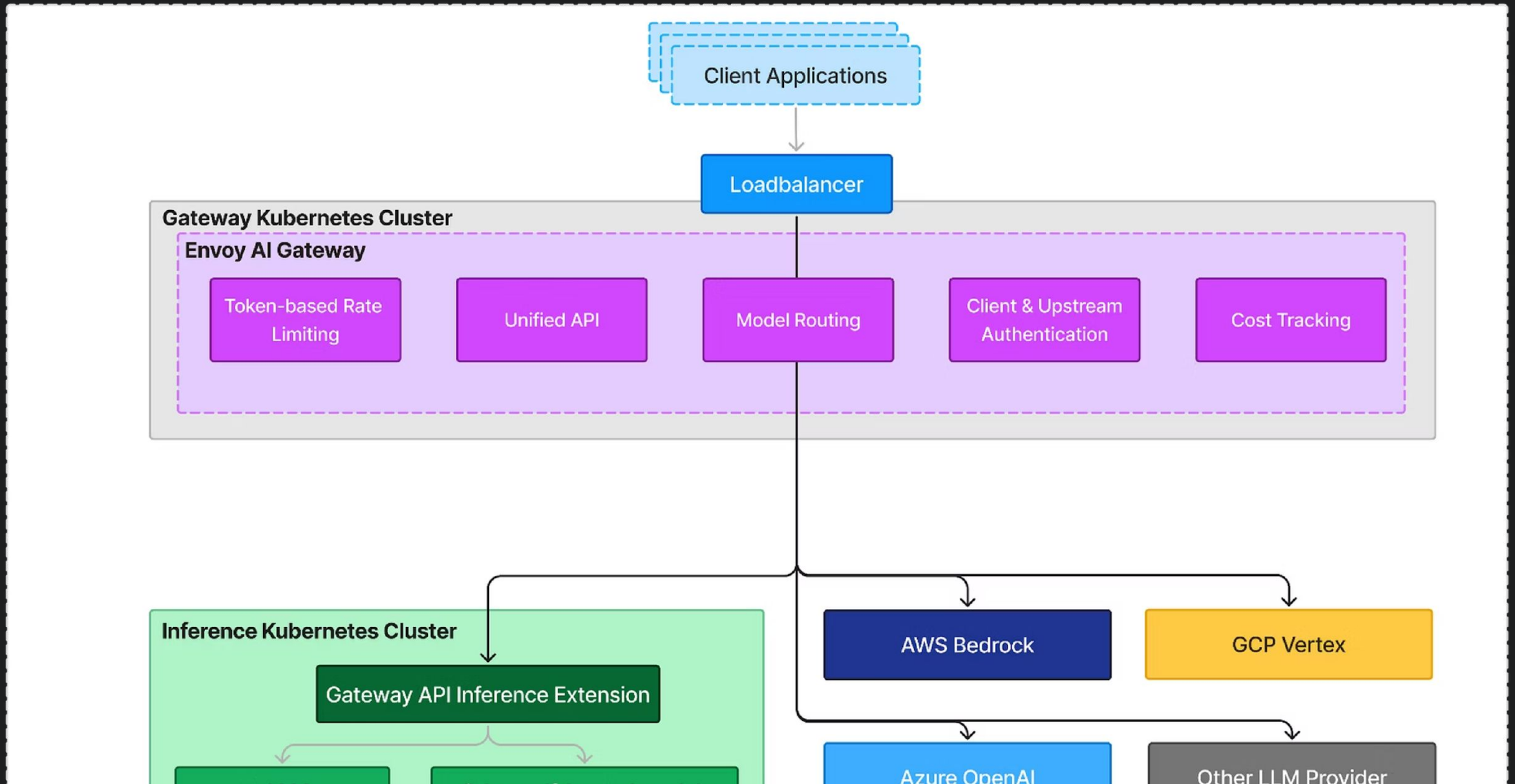
- **Inference Services**
- **AI Gateway**
- **Evals**

vLLM + kserve = infinite scale?

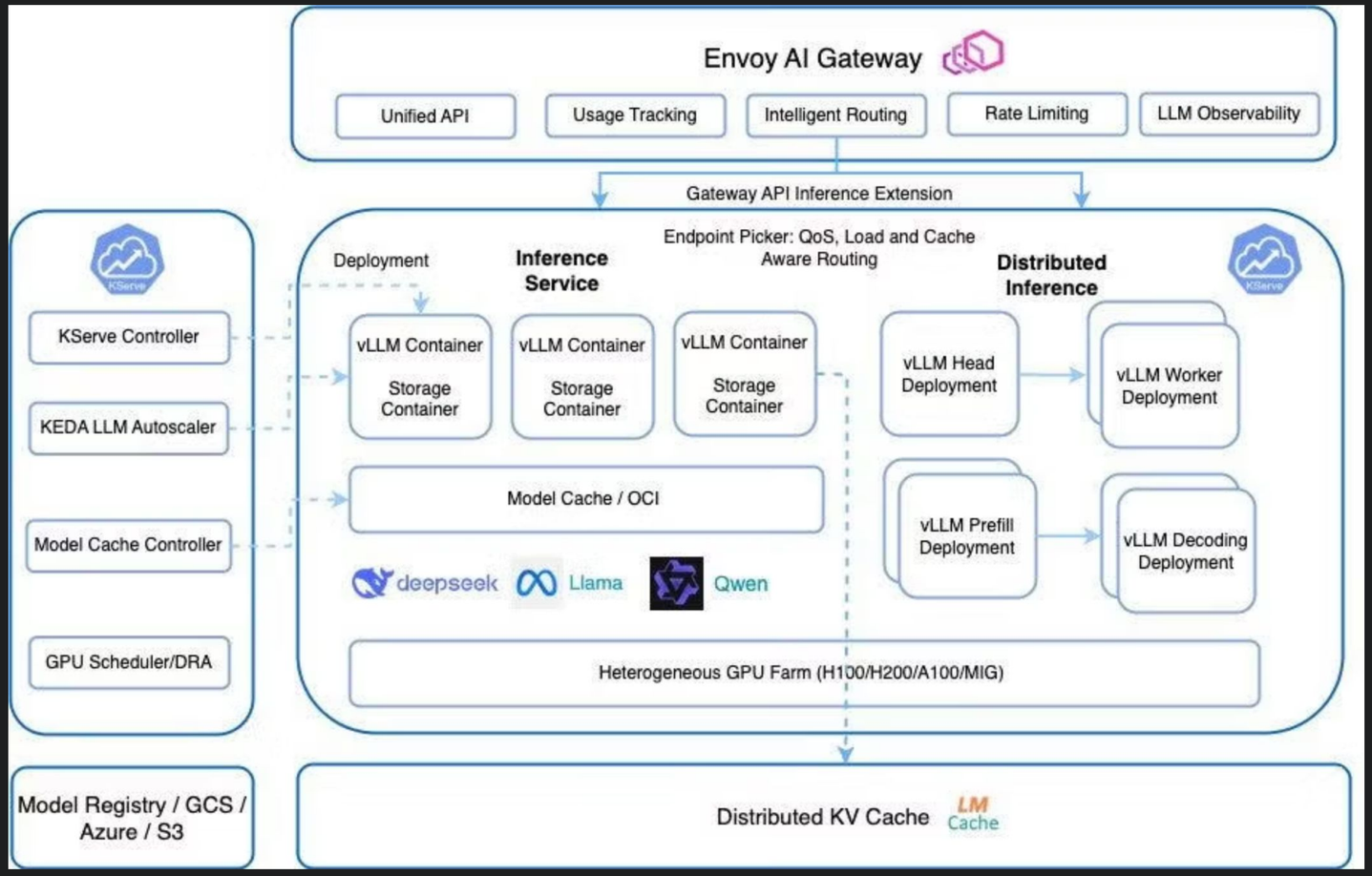


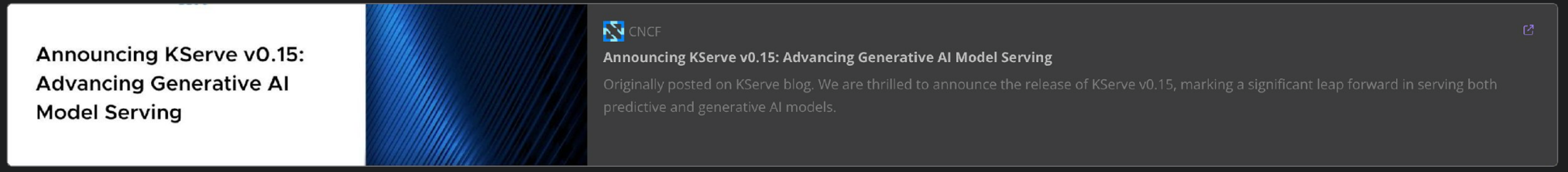


Envoy AI Gateway



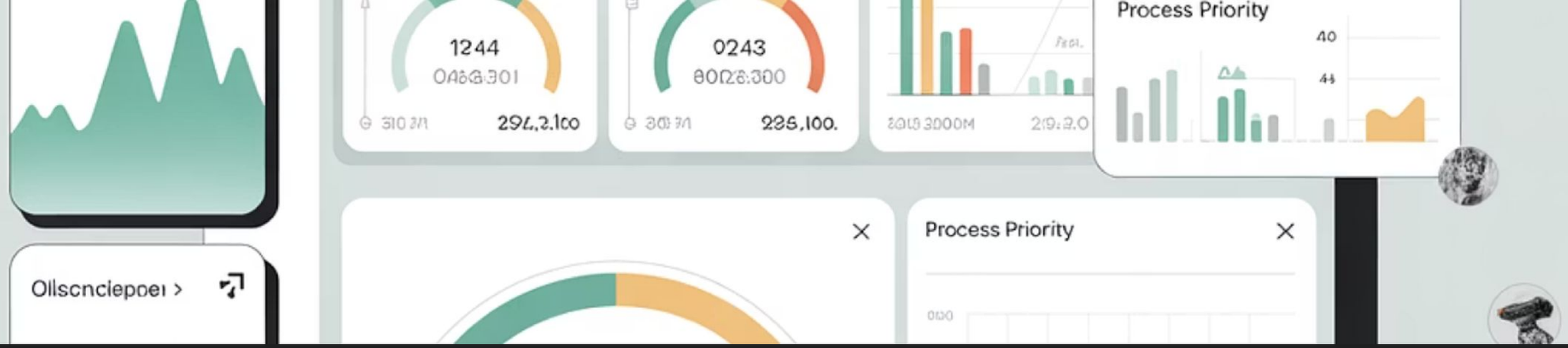
A full picture





Resource Management & Operations

Optimizing performance and controlling costs



GPU Resource Management

vGPU (Virtual GPU)

Sharing of one GPU with multiple VMs or Containers

MIG (Multi-instance GPU)

Partitioning of a physical GPU for multiple independent workloads

GPU Time-Slicing

Slices GPU time across multiple tasks

CUDA

Software to develop, manage and distribute workload

Physical GPU

```
apiVersion: v1
kind: Pod
metadata:
  name: gpu-example
spec:
  containers:
    - name: gpu-example
      image: nvidia/cuda
      resources:
        limits:
          nvidia.com/gpu: 2
  nodeSelector:
    nvidia.com/gpu.product: A100-PCIE-40GB
    nvidia.com/cuda.runtime: 11.4
    nvidia.com/cuda.driver: 470.161.03
```

MIG

```
apiVersion: v1
kind: Pod
metadata:
  name: gpu-example
spec:
  containers:
    - name: gpu-example
      image: nvidia/cuda
      resources:
        limits:
          nvidia.com/mig-1g.5gb: 1
  nodeSelector:
    nvidia.com/gpu.product: A100-PCIE-40GB
    nvidia.com/cuda.runtime: 11.4
    nvidia.com/cuda.driver: 470.161.03
```

Time Slices with GPU

```
apiVersion: v1
kind: Pod
metadata:
  name: gpu-example
spec:
  containers:
    - name: gpu-example
      image: nvidia/cuda
      resources:
        limits:
          nvidia.com/gpu.shared: 1
  nodeSelector:
    nvidia.com/gpu.product: A100-PCIE-40GB
    nvidia.com/cuda.runtime: 11.4
    nvidia.com/cuda.driver: 470.161.03
```

k8s-device-plugin config file

```
version: v1
sharing:
  timeSlicing:
    resources:
    - name: nvidia.com/gpu
      replicas: 10
    ...
```

Time Slices with MIG

```
apiVersion: v1
kind: Pod
metadata:
  name: gpu-example
spec:
  containers:
    - name: gpu-example
      image: nvidia/cuda
      resources:
        limits:
          nvidia.com/mig-1g.5gb.shared: 1
  nodeSelector:
    nvidia.com/gpu.product: A100-PCIE-40GB
    nvidia.com/cuda.runtime: 11.4
    nvidia.com/cuda.driver: 470.161.03
```

k8s-device-plugin config file

```
version: v1
sharing:
  timeSlicing:
    resources:
    - name: nvidia.com/gpu
      replicas: 10
    - name: nvidia.com/mig-1g.5gb
      replicas: 10
    ...
```

Limitations with this approach

Heterogeneous GPU Support

No native support for integrating more than one GPU type per node, restricting hardware configurations.

Complex Constraint Handling

Lack of robust mechanisms for providing complex constraints when requesting a GPU, making advanced scheduling difficult.

Oversubscription Control

Limited control over how oversubscribed GPUs are shared between jobs, potentially leading to unfair resource distribution.

MPS Integration Challenges

Awkward and overly-burdensome support for Multi-Process Service (MPS), adding operational overhead.

Dynamic MIG Provisioning

Inability to dynamically provision Multi-Instance GPU (MIG) devices based on incoming requests, reducing resource elasticity.

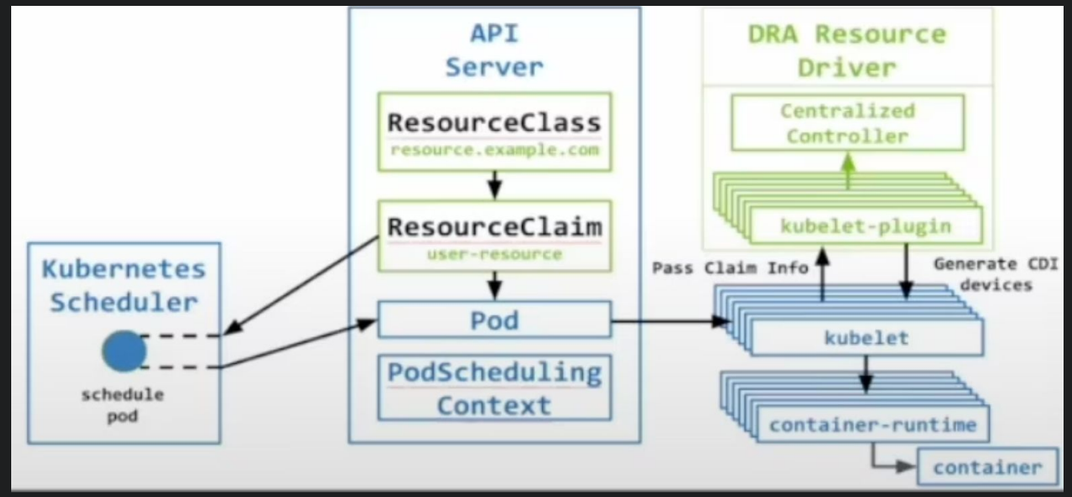
Driver Selection Flexibility

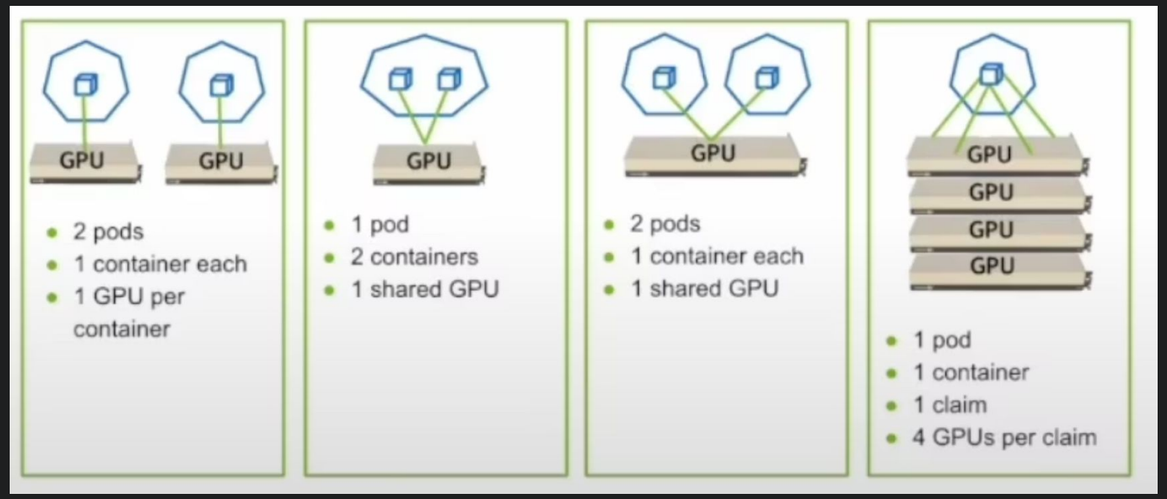
No built-in capability to dynamically choose between NVIDIA and other drivers (e.g., fio) on a per-GPU basis.

DRA - Dynamic Resource Allocation

Allocate **cross-node resources** in Kubernetes

Explicitly **share, partition, and reconfigure** devices **on-the-fly** based on user requests





```yaml
---
apiVersion: resource.k8s.io/v1alpha2
kind: ResourceClaimTemplate
metadata:
  name: unique-gpu
spec:
  spec:
    resourceClassName: gpu.nvidia.com

---
apiVersion: v1
kind: Pod
metadata:
  name: gpu-example
spec:
  containers:
  - name: ctr
    image: nvidia/cuda
    command: ["nvidia-smi" "-L"]
    resources:
      claims:
      - name: gpu0
      - name: gpu1
  resourceClaims:
  - name: gpu0
    source:
      resourceClaimTemplateName: unique-gpu
  - name: gpu1
```

Associated with the DRA Driver and installed by the cluster admin

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: gpu-example
spec:
  containers:
    - name: ctr0
      resources:
        claims:
        - name: gpu
    - name: ctr1
      resources:
        claims:
        - name: gpu
resourceClaims:
  - name: gpu
    source:
      resourceClaimName: unique-gpu
```

Shared access to same underlying GPU

Measuring the Impact of Your Platform

AI Serving Signals

Token Consumption

Daily processing volume

Guardrail Success

Protection execution rate

PII Leakage

Sensitive data exposure rate

AI Performance Metrics

Inference Latency

Average response time

Availability

System uptime for AI services

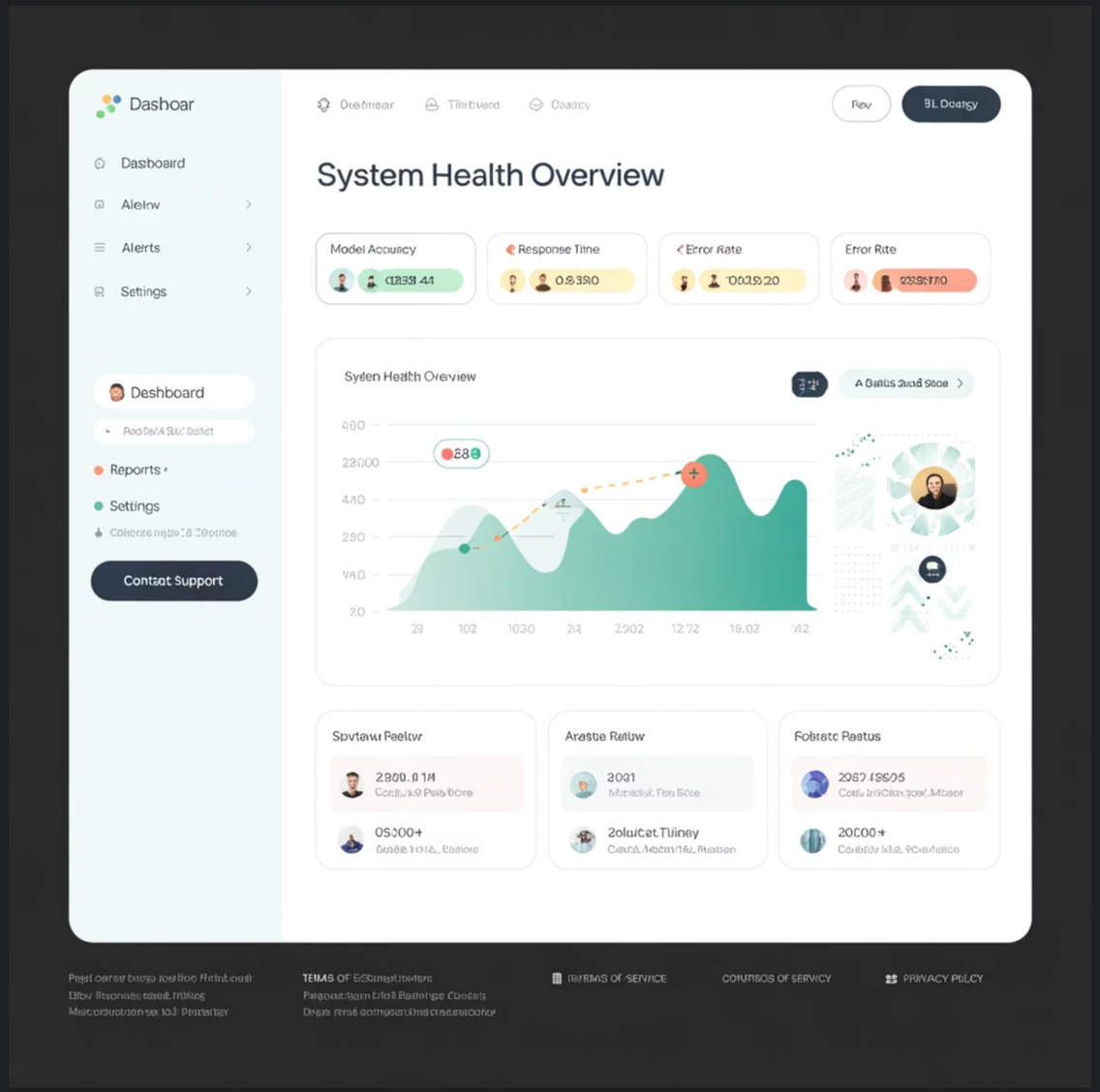
Throughput Gain

vs. non-optimized deployment

Monitoring & Observability

AI-specific requirements:

- 66% of developers cite "almost right but not quite" as biggest frustration
- OpenTelemetry for vendor-neutral monitoring
- Model-specific metrics beyond traditional infrastructure



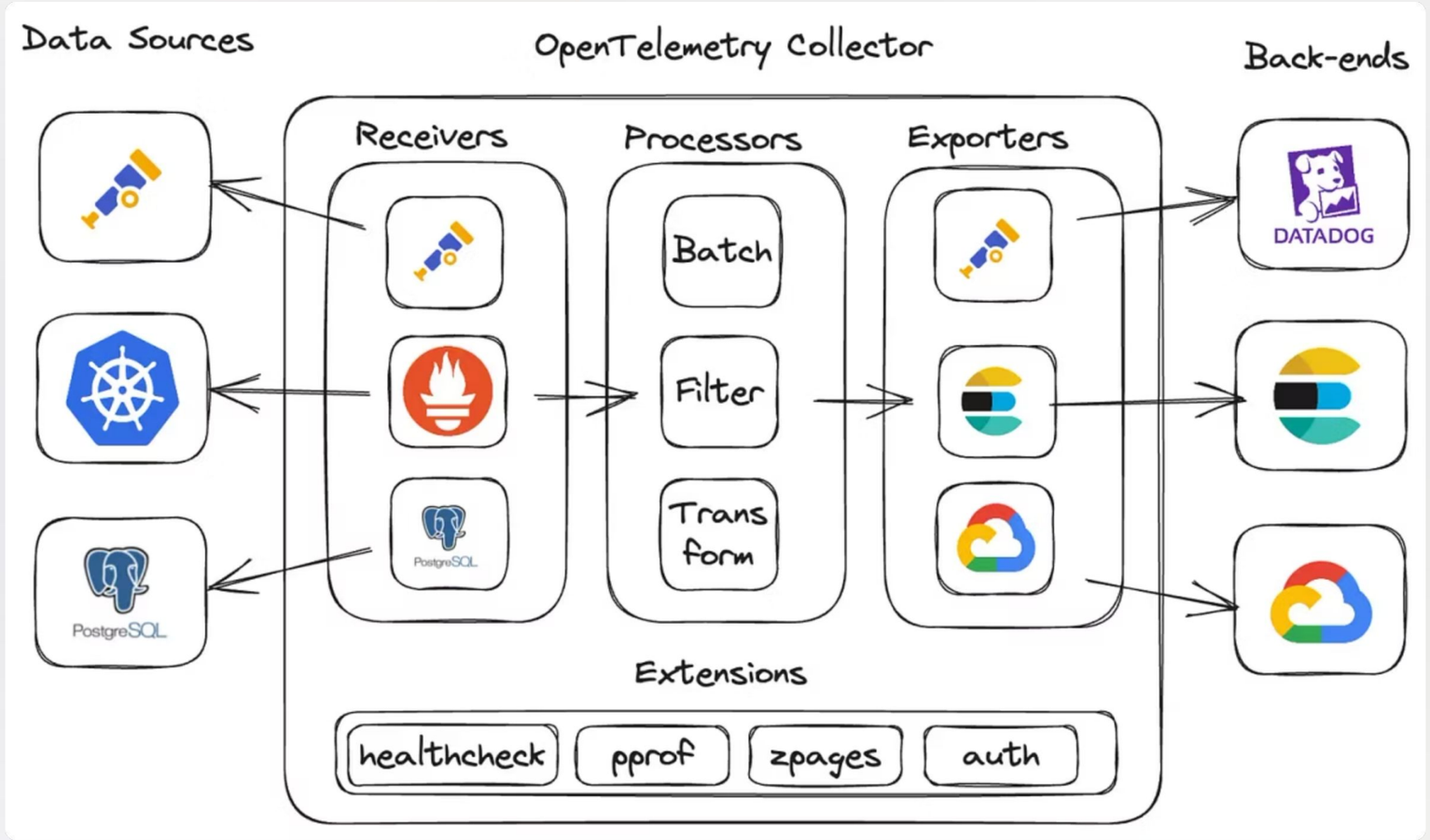
OTEL & CO

Traces

- Request Metadata
  - Temperature
  - top_p
  - Model Name or Version
  - Prompt Details
- Response Metadata
  - Tokens
  - Cost
  - Response Details

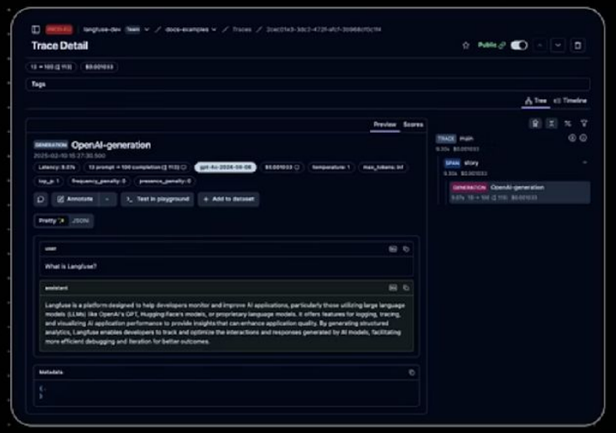
Metrics

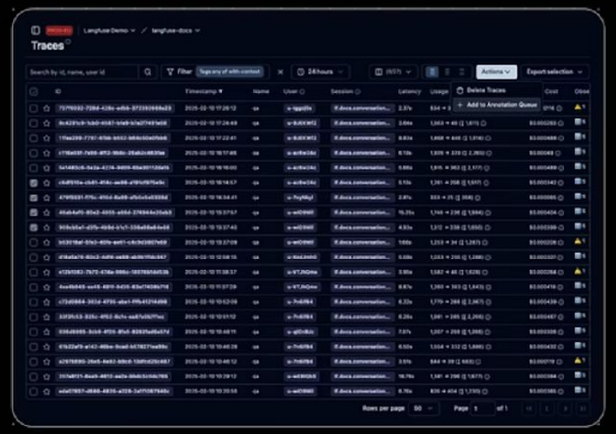
- Request Volume
- Request Duration
- Costs and Tokens Counters

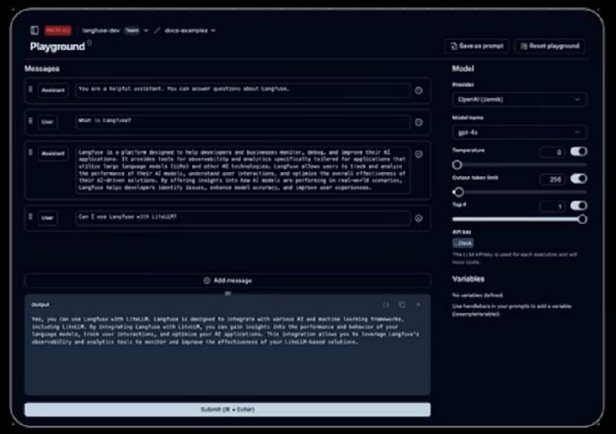


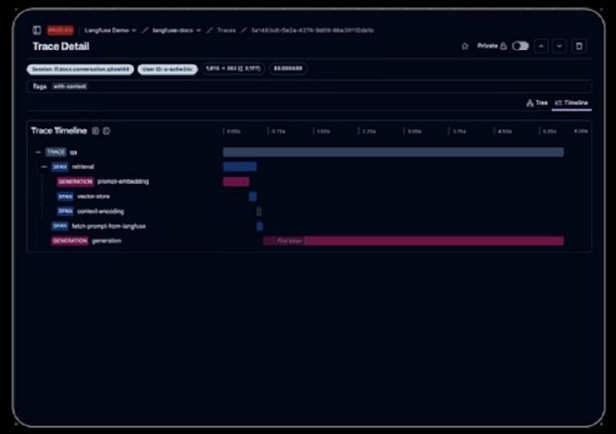
Langfuse

traces, evals, metrics, prompt management, and playground...









...to debug and improve your LLM application together.

```yaml
exporters:
  otlphttp/langfuse:
    endpoint: "https://cloud.langfuse.com/api/public/otel" # EU data region
    # endpoint: "https://us.cloud.langfuse.com/api/public/otel" # US data region
    headers:
      Authorization: "Basic ${AUTH_STRING}" # Previously encoded API keys
```

Think Product and Developer Experience

Making AI accessible to engineering teams

Models as First-Class Citizens - same but different

Development

Create model architectures

Training

Run on specialized hardware

Evaluation

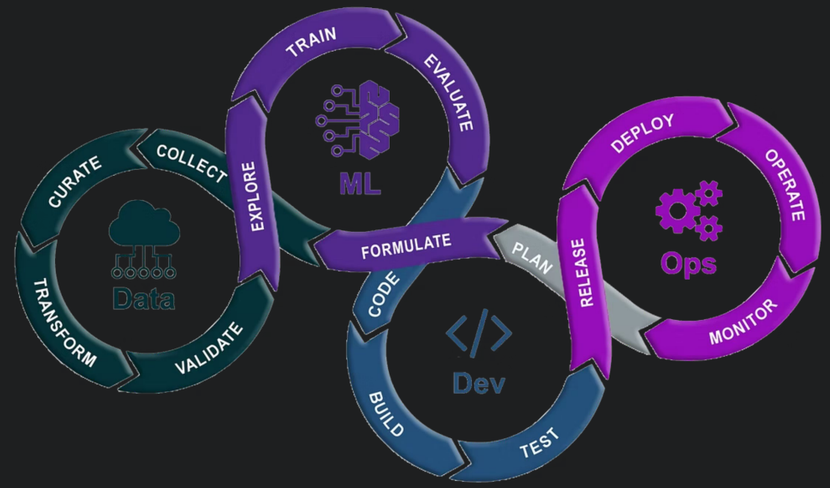
Test against metrics

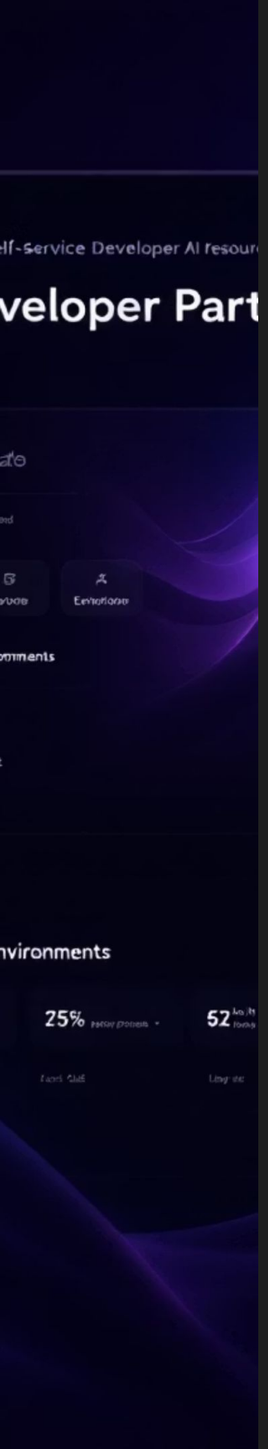
Registry

Store with metadata

Deployment

Serve to production





Democratizing AI Development

Self-Service Portal

- GPU quota management
- On-demand environments
- Cost visibility
- Simple Inferencing and Dependency Management

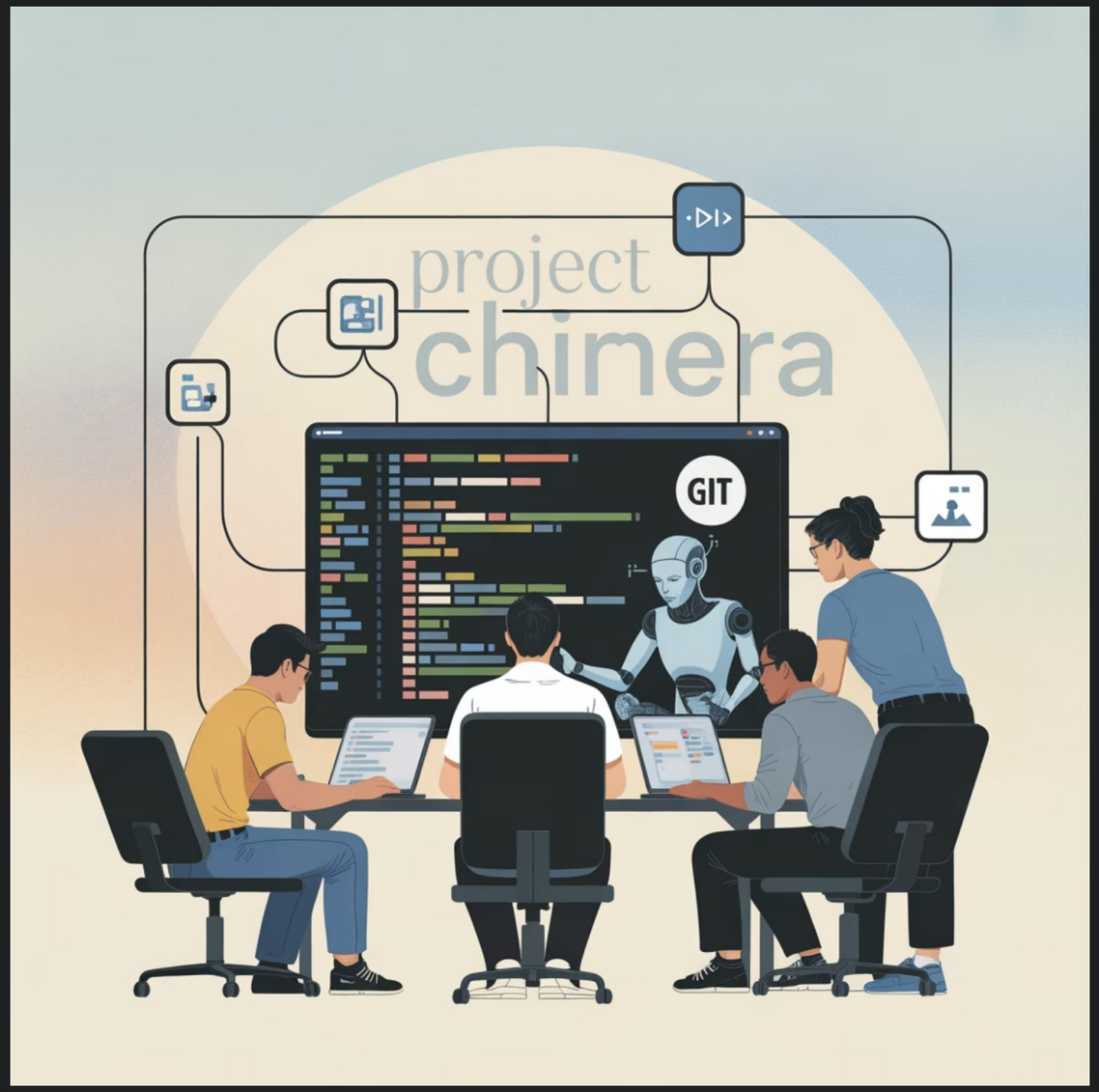
Environment Templates

- Pre-configured ML stacks
- Notebook environments
- Training clusters

Tool Integration

- Popular ML frameworks
- Experiment tracking
- Model management

Workflow Integration



Modern Dev Practices

- Git-native workflows for AI model iterations
- CI/CD with automated AI-powered testing
- Shared workspaces with centralized prompt libraries
- Version control for both code and models

Closing

Platforms for LLMs Enable ISO 42001 & EU AI Act

AI-ready Internal Developer Platforms unlock compliance with emerging AI governance standards.

2023

Standard Release

First international standard for AI management systems.

100%

Governance

Complete oversight of AI assets and processes.

42001

Certification

Demonstrates commitment to responsible AI practices.

Common Pitfalls and Solutions

Over-Engineering from Day One

Underestimating Resource Requirements

Neglecting Model Governance

Poor Developer Adoption

Key Takeaways

Platform engineering transforms LLM deployment from artisanal craft to systematic practice

Balance Control & Flexibility

Create standardized paths that make the right way the easy way, while allowing for customization when necessary.

Start Simple, Iterate Often

Begin with core capabilities that unblock key workflows. Add sophistication based on actual user needs and feedback.

Developer Experience Drives Adoption

The most technically impressive platform is worthless if developers avoid using it. Invest in intuitive interfaces and documentation.

Efficient Resource Management is Critical

GPU utilization directly impacts both cost-effectiveness and user satisfaction. Optimization pays dividends at scale.