ARC OF AI 2026 · AI NEW AND NOTEWORTHY

Platform Engineering for Modern AI

From Multi-Cloud GPUs to Edge and Intelligent Tooling

Kubernetes | MLOps | Edge Inference | MCP Servers | GPU Orchestration

SPEAKER

Adao Oliveira Junior

AI / Cloud Solutions Architect · Author · Speaker

Platform Engineering for Modern AI

From Multi-Cloud GPUs to Edge and Intelligent Tooling

adao.dev | LinkedIn | GitHub | Slides

THE CHALLENGE

AI Has Escaped the Data Center

Multi-Cloud Training

GPU scarcity forces workloads across AWS, GCP, Azure, and bare-metal

Edge Inference

Latency-sensitive AI runs at the edge — retail, automotive, IoT

Developer Tooling

AI-native IDEs, MCP servers, and agents embedded in the SDLC

Operational Complexity

Model versioning, drift detection, cost optimization at every layer

→ The gap between "we trained a model" and "it runs reliably everywhere" is a platform engineering problem.

FOUNDATION

Why Platform Engineering for AI?

Self-Service Infrastructure

ML teams request GPU clusters, model endpoints, and edge deployments through a unified API

Golden Paths for AI Workloads

Opinionated pipelines: training → evaluation → registry → deployment → monitoring

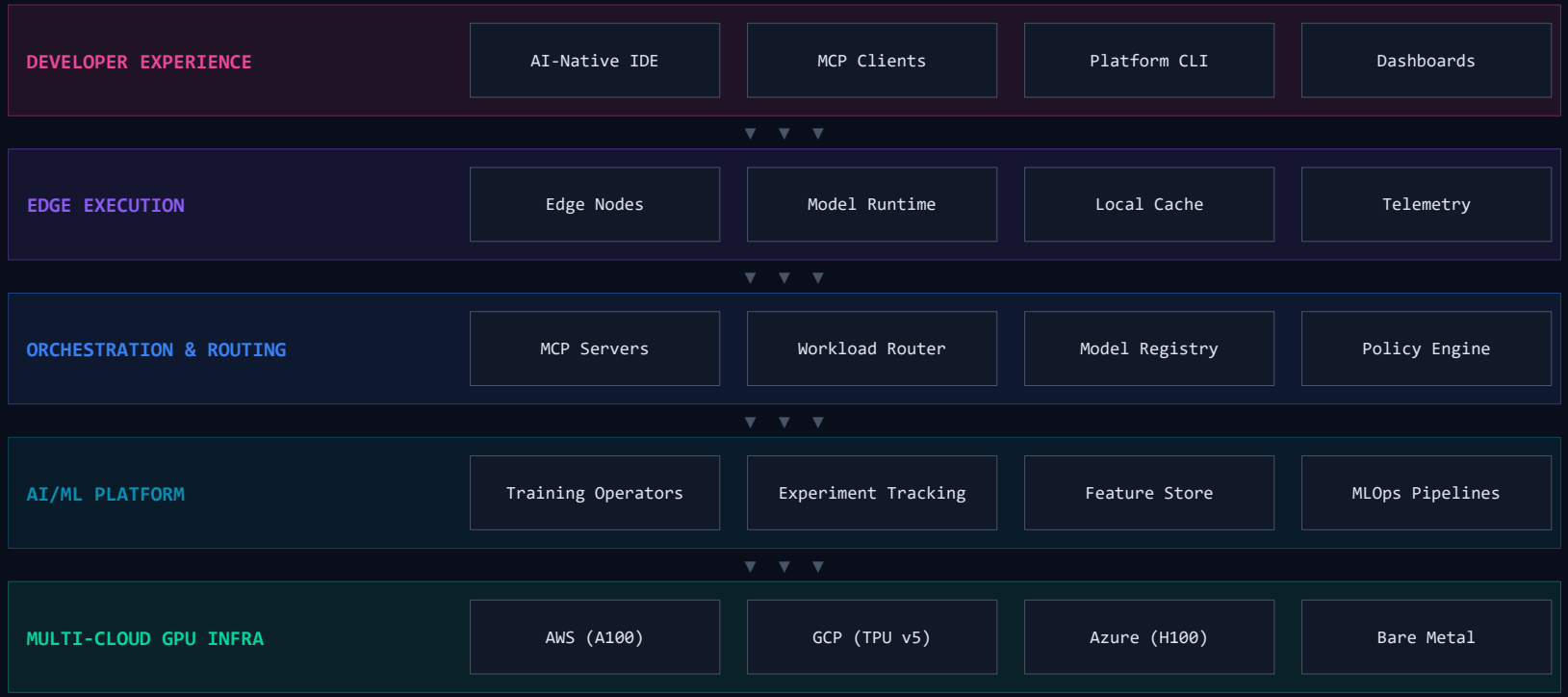
Consistent Abstractions

Same deployment model targeting cloud GPU clusters, edge nodes, or developer laptops

```yaml
# Kubeflow Training Operator — distributed fine-tune
apiVersion: kubeflow.org/v1
kind: PyTorchJob
metadata:
  name: llm-finetune-v3
  namespace: ml-platform
spec:
  pytorchReplicaSpecs:
    Master:
      replicas: 1
      restartPolicy: OnFailure
      template:
        spec:
          containers:
          - name: pytorch
            image: nvcr.io/nvidia/pytorch:24.01-py3
            resources:
              limits:
                nvidia.com/gpu: "8"
          nodeSelector:
            cloud.google.com/gke-accelerator: nvidia-a100-80gb
```

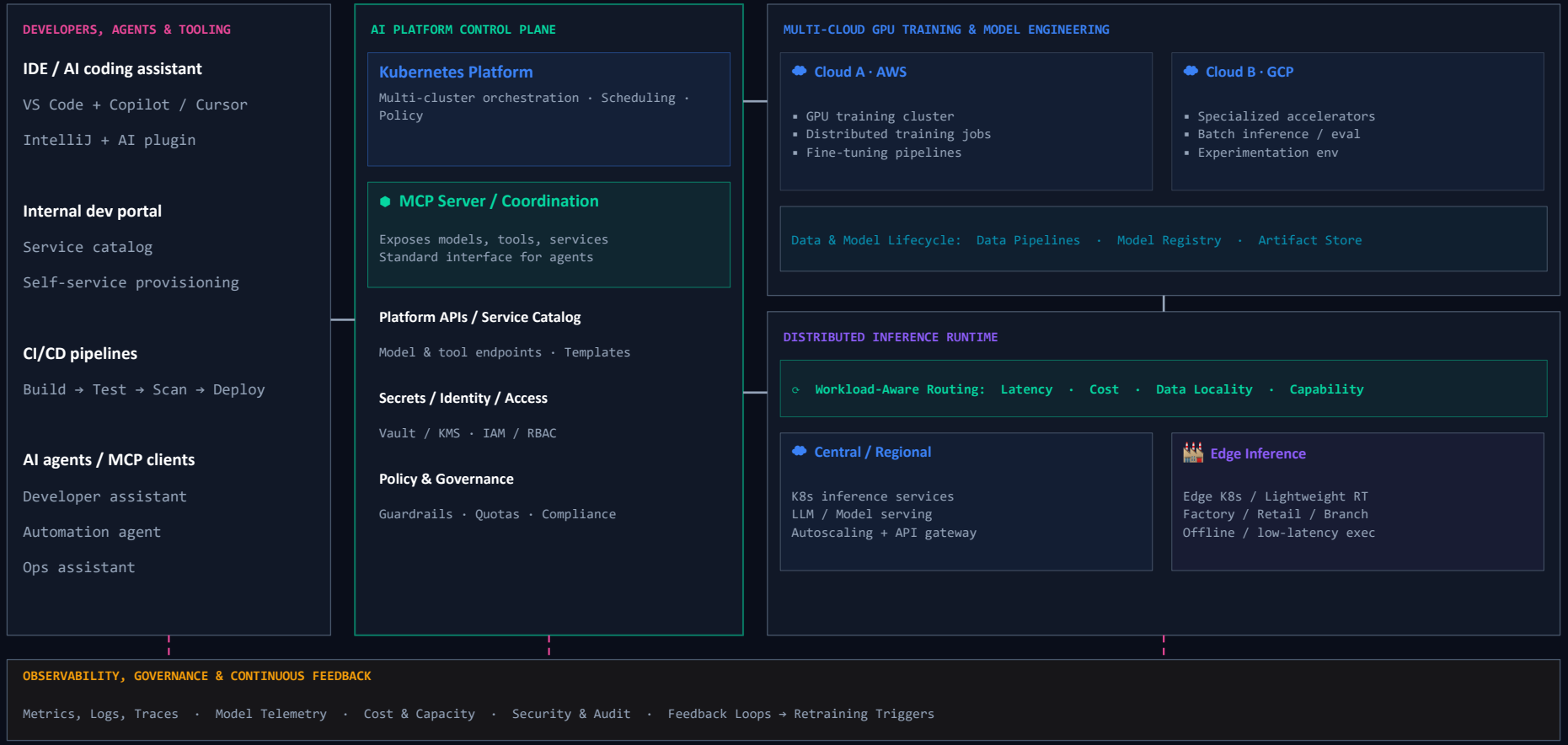
THE BIG PICTURE

Reference Architecture



DETAILED REFERENCE ARCHITECTURE

Platform Engineering for Modern AI



LAYER 1

Multi-Cloud GPU Infrastructure

Kubernetes as Control Plane

Kueue for queuing and quotas. NVIDIA GPU Operator for driver lifecycle. Gang scheduling for distributed training.

Cross-Cloud Resource Pooling

Abstract GPU types behind a unified API. Platform decides placement based on cost, availability, and SLAs.

Cost-Aware Scheduling

Spot/preemptible for training. Reserved for inference. Automatic failover across providers when preempted.

```
# Kueue ClusterQueue
apiVersion: kueue.x-k8s.io/v1beta1
kind: ClusterQueue
spec:
  resourceGroups:
  - flavors:
    - name: aws-a100
      nominalQuota: 64
    - name: gcp-h100
      nominalQuota: 32
```

LAYER 2

AI Ops & MLOps on Kubernetes

Training Operators

KubeFlow, Ray on K8s, PyTorch Elastic — CRDs with auto-scaling

Model Registry

OCI-based storage. Version control, lineage, eval gates

MLOps Pipelines

Argo/Tekton CI/CD. Auto retraining. Blue/green deployments

Observability

GPU util, training loss, inference latency, model drift

Governance

Model cards, audit trails, policy-as-code via OPA

Experiment Tracking

MLflow/W&B integrated. Auto-logged metrics & artifacts

LAYER 3

Edge-First Inference

Why Edge?

Single-digit ms latency. Data sovereignty. Offline capability. Reduced cloud costs for high-volume inference.

Model Lifecycle at Edge

Auto quantization (INT8/INT4). ONNX Runtime / TensorRT. OTA updates via platform.

Edge ↔ Cloud Sync

Telemetry flows to central observability. Performance comparison. Auto promotion or rollback.

```yaml
# KServe — INT8 edge inference service
apiVersion: serving.kserve.io/v1beta1
kind: InferenceService
metadata:
  name: vision-v2-edge
  namespace: edge-us-south
spec:
  predictor:
    canaryTrafficPercent: 10
    model:
      modelFormat:
        name: onnx
      runtime: kserve-onnxruntime
      storageUri: s3://model-registry/vision-v2.1-int8
      resources:
        limits:
          nvidia.com/gpu: "1"
    nodeSelector:
      node-role: edge-gpu
```

EMERGING PATTERN

Workload-Aware Routing

Not every inference request is equal. The routing engine dynamically selects the optimal execution target.

Latency

Edge for <10ms, cloud for <200ms, batch for async

Capability

Small models → edge. Large LLMs → cloud GPU

Cost

Budget-aware. Spot for non-critical. Reserved for SLA-bound

Load

Real-time utilization. Queue depth. Auto-spillover

```yaml
# LiteLLM proxy — workload-aware routing
router_settings:
  routing_strategy: latency-based-routing
  num_retries: 3

model_list:
  - model_name: vision-small
    litellm_params:
      model: openai/vision-v2-int8
      api_base: http://kserve.edge-us-south.svc

  - model_name: llm-large
    litellm_params:
      model: openai/llama-3.3-70b
      api_base: http://vllm.gpu-h100.svc

fallbacks:
  - {"vision-small": ["vision-cloud-spot"]}
  - {"llm-large": ["llm-a100"]}
```

THE GLUE

MCP-Based Coordination

Model Context Protocol servers provide the coordination layer — a standardized interface between AI models, services, and developer tools.

Platform-as-MCP

Every platform capability exposed as an MCP tool: deploy, query, retrain, manage

Agent Orchestration

AI agents discover platform tools via MCP. Natural language commands.

Service Mesh for AI

MCP coordinates between models, handles context, manages distributed inference.

```
// MCP Server — Platform Tools
const server = new MCPServer({
  name: "ai-platform",
  tools: [{
    name: "deploy_model",
    description: "Deploy a model to"
      + " cloud or edge targets",
    parameters: {
      model: { type: "string" },
      target: {
        enum: ["cloud", "edge", "auto"]
      },
      routing: { type: "string" }
    },
    handler: async (params) => {
      const result = await
        platform.deploy(params);
      return result.status;
    }
  }]
});
```

PUTTING IT TOGETHER

End-to-End Example Flow

platform-cli

```
$ platform model train --config finetune.yaml
✓ Job scheduled → aws-a100 (64 GPUs)
✓ Training complete — accuracy: 94.2%

$ platform model register vision-v3
✓ Registered → oci://registry/vision-v3
✓ Auto-quantized: INT8 (234MB)

$ platform deploy vision-v3 --target auto
✓ Cloud: live (us-east, eu-west)
✓ Edge: 47/47 nodes (canary 100%)

→ Edge: 78% requests (4.2ms avg)
→ Cloud: 22% requests (89ms avg)
✓ Cost savings: 43% vs cloud-only
```

Developer triggers training

Via CLI, IDE, or MCP agent. Platform selects GPU cluster.

Model registered & optimized

OCI registry. Auto quantization. Eval gates passed.

Deployed to cloud + edge

Routing policy determines targets. Canary rollout.

Intelligent routing active

78% edge at 4ms. 43% cost reduction.

THE GOAL

Developer Experience as the Product

Reduce Friction

Self-service everything. No tickets for GPUs, no manual edge deployments.

Standardize Environments

Same abstractions from laptop to cloud to edge. Dev/staging/prod parity.

Embed Intelligence

MCP servers in the IE. AI agents that know the platform. Natural language deploys.

The metric that matters:

Time from idea → production

Weeks → Hours

LOOKING AHEAD

Emerging Patterns

Edge-First Inference

Default to edge. Fall back to cloud. Inverts the traditional cloud-first architecture for latency-critical AI.

Workload-Aware Routing

Replace static load balancers with AI-aware routers that understand model capabilities and cost constraints.

MCP as Service Mesh for AI

MCP servers become the standard coordination layer. Models discover tools, services discover models.

Platform as Product

Treat the internal AI platform like a product: user research, feedback loops, versioned APIs, developer satisfaction.

SUMMARY

Key Takeaways

01

Design for Portability

Build AI platforms with multi-cloud and edge as first-class targets.

02

Embrace Edge-First Inference

Push inference to the edge by default. Cloud becomes the fallback.

03

MCP as the Coordination Layer

Bridge AI models, platform services, and developer tools.

04

Platform Engineering is the Foundation

Without it, AI ops collapses under operational complexity.

THANK YOU

Questions & Discussion

Platform Engineering for Modern AI
Arc of AI Conference 2026 · Austin, TX