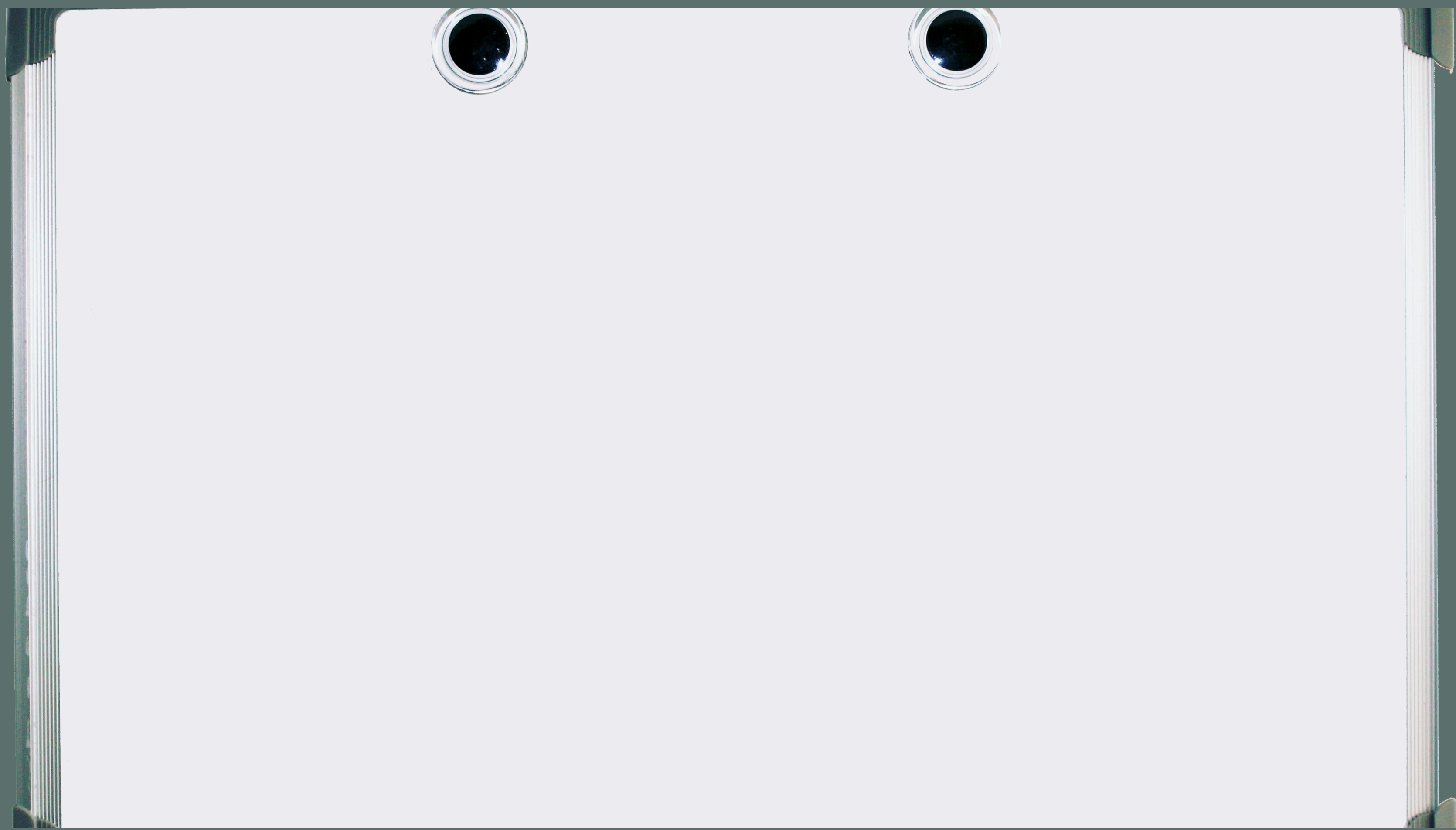
🎨 Systems Design: Multi-Cloud MLOps Pipeline



"Deploy ML models across AWS, GCP, and Azure without vendor lock-in"

Where do you start?

Most people pick one cloud and pray it works
I design for portability from day one.

Here's the thinking process: 👇

📝 Step 1: Why Multi-Cloud?

Why not just pick AWS and be done?

Because:
→ **Vendor lock-in** = pricing power (**they own you**)
→ **Single cloud** = **single point of failure**
→ **Best-of-breed** = **different clouds excel at different things**

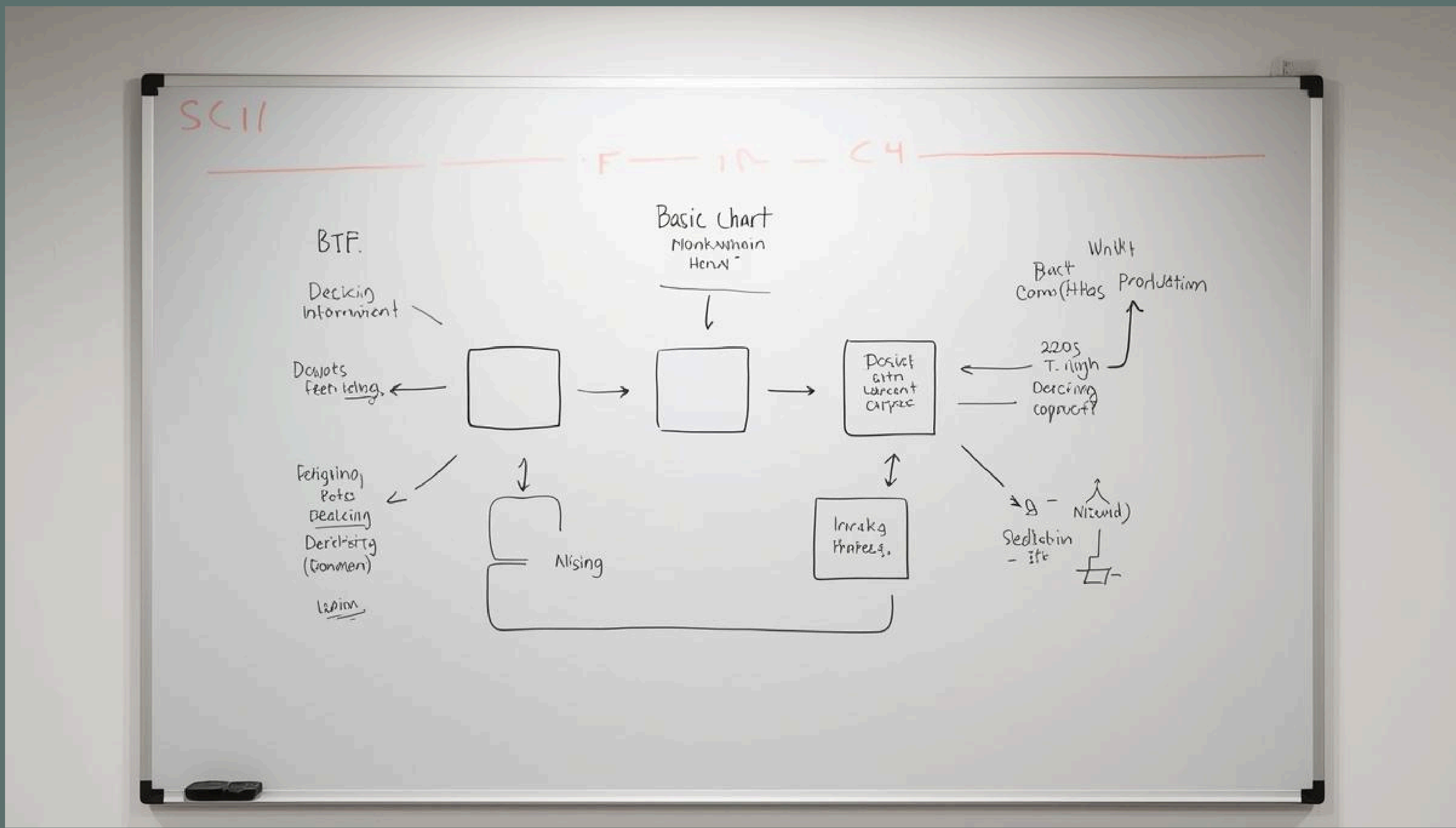
The real question:

"How do we deploy models anywhere **WITHOUT** rewriting everything?"

This question **determines your architecture.**

Wrong approach = locked in forever.
Right approach = deploy anywhere in hours.

🔄 Step 2: Map the MLOps Flow



Training → Registry → Deploy → Monitor

Fill in the [?]:

→ Where do models train? (Which cloud? Both?)

→ Where do artifacts live? (One registry? Three?)

→ How do we deploy? (Cloud-native? Containers?)

→ How do we monitor? (One dashboard? Three?)

This is the **skeleton.**

Now add the **organs** that make it **cloud-agnostic**.

🧩 Step 3: Core Components

Training (**Kubernetes**, **Kubeflow**, or **cloud-native**)
↓
Model Registry (**MLflow**, **W&B**, **cloud-native**)
↓
Deployment (**SageMaker**, **Vertex**, **Azure ML**, or **K8s**)
↓
Monitoring (**Prometheus**, **CloudWatch**, **unified dashboard**)

But this is **TOO** simple.

What's missing?
→ How do we **move models** between clouds?
→ How do we **handle cloud-specific** APIs?
→ What if a **cloud goes down**?
→ How do we **track costs** across three clouds?

Add the **layers** that multi-cloud requires.

📦 Step 4: Multi-Cloud Artifact Registry

Cloud-Agnostic Model Registry:

Option 1: MLflow (self-hosted)
→ Works anywhere
→ Cloud-agnostic
→ You manage infrastructure

Option 2: Weights & Biases
→ SaaS (no infrastructure)
→ Multi-cloud native
→ Unified tracking

Option 3: Cloud-native + sync
→ AWS S3 ↔ GCP GCS ↔ Azure Blob
→ Sync artifacts across clouds
→ More complex, full control

Questions to answer:

→ **Single source of truth**? (Yes—one registry)
→ **Versioning strategy**? (Semantic versioning)
→ **Metadata**? (Training config, metrics, lineage)

This is where your models live.
Get this wrong = can't deploy anywhere.

☁️ Step 5: Cloud-Specific Deployment Layers

AWS Deployment:
→ SageMaker (managed inference)
→ EKS (Kubernetes)
→ Lambda (serverless)

GCP Deployment:
→ Vertex AI (managed)
→ GKE (Kubernetes)
→ Cloud Run (serverless)

Azure Deployment:
→ Azure ML (managed)
→ AKS (Kubernetes)
→ Azure Functions (serverless)

The trick: **Abstraction layer**

Container-based deployment (**Docker**):
→ **Build once**
→ **Deploy** to SageMaker, Vertex, Azure ML
→ Or deploy to **K8s** (EKS, GKE, AKS)

Kubernetes = your portability layer.
Cloud APIs change. Containers don't.

📊 Step 6: Unified Monitoring Across Clouds

The Problem:

→ AWS CloudWatch (only sees AWS)
→ GCP Cloud Monitoring (only sees GCP)
→ Azure Monitor (only sees Azure)

You need: ONE dashboard, THREE clouds

Solution: Cloud-agnostic monitoring

Prometheus + Grafana:

→ Scrapes metrics from all clouds
→ Unified dashboards
→ Consistent alerting

Track:

→ Inference latency (per cloud)
→ Model drift (unified across clouds)
→ Error rates (compare clouds)
→ Costs (which cloud is expensive?)

Without unified monitoring = flying blind across three clouds.

You can't optimize what you can't see.

🔧 Step 7: CI/CD for Multi-Cloud

GitHub Actions / GitLab CI:

Push code → Run tests (unit, integration)
↓
Build container
↓
Push to registry (MLflow/W&B/ECR/GCR/ACR)
↓
Deploy to: [AWS | GCP | Azure | All Three]

Key: Infrastructure as Code (Terraform)

One config file = deploy to any cloud:
→ terraform apply -var="cloud=aws"
→ terraform apply -var="cloud=gcp"
→ terraform apply -var="cloud=azure"

Secrets management:
→ HashiCorp Vault (cloud-agnostic)
→ Or cloud-native (AWS Secrets Manager, GCP Secret Manager)

CI/CD = your deployment automation.

Get this right = deploy anywhere in minutes.

⚠️ Step 8: Design for Cloud-Specific Failures

Where does multi-cloud break?

Cloud-specific quirks:
→ AWS: IAM permissions (complex, service-specific)
→ GCP: Service accounts (different auth model)
→ Azure: Resource groups (deployment scope)

Network failures:
→ Cross-cloud latency (AWS → GCP = 50-100ms)
→ Egress costs (moving data OUT = expensive)

Cost surprises:
→ AWS: Data transfer out ($0.09/GB)
→ GCP: Networking egress ($0.08/GB)
→ Azure: Bandwidth ($0.087/GB)

Then design defenses:
→ Cloud-agnostic abstractions (Terraform, K8s)
→ Fallback deployments (if AWS fails, route to GCP)
→ Cost monitoring (alert on >$X per cloud)
→ Test BEFORE production (staging on each cloud)

Plan for cloud failures.

💰 Step 9: Multi-Cloud Cost Optimization

Why multi-cloud saves money:

Spot instances / Preemptible VMs:
→ AWS Spot: 70-90% discount
→ GCP Preemptible: 80% discount
→ Azure Spot: 90% discount

Training workloads:
→ Train on cheapest cloud at that moment
→ Auto-switch if prices change

Inference workloads:
→ Route traffic to lowest-cost region
→ Or lowest latency (user experience > cost)

Storage tiering:
→ Hot data: Where it's accessed most
→ Cold data: Cheapest cloud (GCS Nearline, S3 Glacier)

Tools: CloudHealth, Kubecost

Track spend per cloud, per model, per team.

Multi-cloud without cost tracking = budget nightmare

✅ The Complete Multi-Cloud MLOps System

Training (K8s or cloud-native)
↓
Model Registry (MLflow / W&B - cloud-agnostic)
↓
CI/CD (GitHub Actions / Terraform)
↓
Deploy to: AWS | GCP | Azure
↓
Unified Monitoring (Prometheus + Grafana)
↓
Cost Tracking (per cloud, per model)

This is production **multi-cloud MLOps.**

Not "pick **CLOUD** and hope."

6 layers. Each critical.

The difference: Systems thinking.

Most teams:
→ Pick one cloud
→ Use cloud-native tools
→ Get locked in
→ Pay whatever the cloud charges

1% teams:
→ Design for portability from day one
→ Use cloud-agnostic tools (K8s, Terraform, MLflow)
→ Deploy anywhere in hours
→ Negotiate from strength (can switch clouds)

Vendor lock-in isn't technical.
It's architectural.

Design for portability, or
pay the lock-in tax forever.

Next in series: Petabyte-Scale Data Pipeline 👇

#SystemsDesign #MLOps #MultiCloud #CloudArchitecture