WHITE PAPER | code-b.dev | May 2026

ML Deployment Architecture Guide

Choosing the Right Stack for Your Team Size, Budget, and Use Case

A practical decision framework with full stack recommendations, real 2026 pricing, and the deployment mistakes most teams make.

Audience: ML Engineers, Data Scientists, CTOs, Engineering Leads

Published: May 2026

Executive Summary

In 2026, the question is no longer whether to deploy machine learning models; it is how. A startup that deploys on AWS SageMaker when it needs rendering burns over $1,400 a month unnecessarily. An enterprise that deploys on Railway when it needs Azure ML exposes itself to compliance risk it cannot afford.

Most deployment guides tell you what tools exist. This white paper tells you which tools are right for your specific situation. We have distilled the 2026 ML deployment landscape into three team archetypes, a 10-question decision framework, full-stack recommendations, real cost data, and the most common stack mismatches we see teams make.

Key Findings

- Teams that choose the wrong deployment tier overspend by an average of 3x.
- Approximately 50% of ML practitioners do not monitor their deployed models (ACM Queue, 2025).
- 87% of ML project failures happen in production, not during training.
- The most common mistake is matching infrastructure complexity to aspirations rather than actual requirements.
- LLM deployment in 2026 requires a fundamentally different architecture from classical ML, yet most teams attempt to use the same patterns for both.

What This Paper Covers

- Why most teams choose the wrong stack and the five questions that reveal the right one
- The three deployment archetypes and how to identify which one you are
- A 10-question decision framework that narrows your options before you open a cloud console
- Full-stack recommendations per archetype: serving, MLOps, hosting, monitoring, and CI/CD
- Classical ML versus LLM deployment: when each architecture applies
- Real 2026 cost comparisons across six deployment paths with a GPU pricing reference
- The most common stack mistakes by team size and a pre-deployment architecture checklist

1. Why Most Teams Choose the Wrong Stack

Stack decisions made at the start of an ML project are sticky. Migrating from SageMaker to self-hosted Kubernetes six months later means re-architecting IAM policies, CI/CD pipelines, monitoring integrations, and team workflows. Getting the initial decision right matters.

The root cause of most wrong-stack decisions is evaluating tools before clearly answering the five questions that determine which tools are even relevant.

The Five Foundation Questions

1. What is your model type and size? A scikit-learn classifier at 2MB has completely different requirements from a fine-tuned 7B LLM needing 14GB of GPU VRAM. Model type is the first filter that eliminates entire categories of deployment options.

2. What are your latency requirements? Real-time inference under 100ms eliminates serverless cold-start options. Batch inference with minutes acceptable opens up spot instances and scale-to-zero architectures that cut costs by 60 to 90 per cent.

3. What is your expected traffic pattern? Steady traffic favours reserved instances. Spiky or unpredictable traffic favours autoscaling platforms. Near-zero with occasional bursts favours scale-to-zero serverless. Most teams overestimate early traffic and over-provision.

4. Who owns and operates the deployment? A solo data scientist handling deployment has no bandwidth for Kubernetes. A dedicated MLOps team can extract real value from that control. Match operational complexity to team capacity.

5. What does your compliance posture require? HIPAA, GDPR, SOC 2, and financial compliance can instantly narrow options to specific clouds and regions or force self-hosting to keep data off third-party infrastructure.

The right deployment stack is the simplest stack that meets all your requirements, not the most powerful or scalable. Complexity has a real cost: engineering time to set up, maintain, and debug when something breaks.

2. The Three Deployment Archetypes

Every ML deployment situation maps to one of three archetypes. Identify yours before evaluating any tool.

Archetype 1: Solo / Startup

- 1 to 3 engineers; the data scientist owns the deployment
- Speed to ship matters more than infrastructure control
- No dedicated DevOps or MLOps engineer
- Traffic is low or unpredictable; downtime is unfortunate, not catastrophic
- Minimal compliance requirements; classical ML or light LLM use via API

Archetype 2: Growth Team

- 5 to 25 engineers; ML and DevOps are separate roles
- Multiple models in production simultaneously
- Needs proper CI/CD, model versioning, and drift monitoring
- Traffic is growing with some peaks; downtime has a real business cost
- Basic compliance, such as SOC 2, is common; a mix of classical ML and LLMs

Archetype 3: Enterprise

- Dedicated ML platform or MLOps team
- Dozens of models with governance and audit trail requirements
- High traffic with formal SLA requirements; downtime has a significant financial impact
- HIPAA, GDPR, SOC 2, or FedRAMP compliance required
- LLMs, classical ML, and custom model architectures in production simultaneously

Budget alone does not determine your archetype. A Series A startup in healthcare with HIPAA requirements is Archetype 3 regardless of team size. A 200-person company running low-traffic internal analytics models is Archetype 2 regardless of revenue. Match to your actual constraints, not your company stage.

3. The 10-Question Decision Framework

Answer these questions in order. Each answer eliminates options. By question 10, you should have a narrow list of viable stacks rather than a blank canvas of every tool ever written. Write your answers down, the act of committing to paper surfaces assumptions your team has not made explicit.

1. **What is your model type and size?**

- Classical ML (sklearn, XGBoost): any CPU hosting works
- Deep learning under 1GB: CPU or small GPU instance
- LLM (7B to 70B+ parameters): GPU required, eliminates most PaaS options

2. **What is your latency requirement per prediction?**

- Real-time under 100ms: eliminates Lambda and serverless cold-start options
- Near-real-time under 2 seconds: serverless viable with pre-warming
- Batch with minutes acceptable: opens spot instances and scale-to-zero options

3. **Whatis your expected daily request volume?**

- Under 10,000/day: PaaS and free tiers are viable
- 10,000 to 1 million/day: managed containers or cloud ML platforms
- Over 1 million/day: Kubernetes, reserved instances, or custom infrastructure

4. **How many models will you deploy simultaneously?**

- 1 to 2 models: single service, no orchestration needed
- 3 to 10 models: MLflow model registry as a minimum
- 10 or more models: dedicated ML platform such as SageMaker, Vertex AI, or Seldon

5. **Do you have compliance or data residency requirements?**

- HIPAA, GDPR, or FedRAMP: narrows to compliant regions; may require self-hosted
- SOC 2: all major cloud providers qualify
- None: all options remain open

6. **Who owns operations after deployment?**

- Data scientist with no dedicated ops: managed PaaS only, avoid Kubernetes
- Backend engineer with part-time ops: managed containers such as Cloud Run or Fargate
- Dedicated MLOps or DevOps: full range including self-hosted and Kubernetes

7. **Are you already committed to a cloud provider?**

- AWS: SageMaker, ECS/EKS, Lambda with deep native integration
- GCP: Vertex AI and Cloud Run, best for GenAI and RAG pipelines
- Azure: Azure ML, best for Microsoft-stack enterprises
- No commitment: evaluate Google Cloud Run and Render first

8. **Does your traffic have significant peaks and valleys?**

- Flat and predictable: reserved instances save 40 to 60 percent
- Spiky or unpredictable: autoscaling platforms such as Cloud Run or Modal
- Near-zero with occasional bursts: scale-to-zero serverless

9. **Do you need model explainability or bias monitoring?**

- Required in a regulated industry: Fiddler AI, Arize, or SageMaker Clarify
- Nice to have: Evidently AI open-source is sufficient
- Not required: Prometheus and Grafana for infrastructure monitoring are enough

10. **What is your monthly infrastructure budget ceiling?**

- Under $200/month: Render, Railway, Cloud Run, or Hugging Face Spaces
- $200 to $2,000/month: Cloud Run, ECS, or managed ML endpoints

- Over $2,000/month: enterprise ML platforms, Kubernetes, dedicated GPU instances

4. Stack Recommendations by Archetype

These are the patterns that consistently work for each archetype in 2026. Each recommendation covers the key layers: model serving, hosting, MLOps, monitoring, and CI/CD.

Archetype 1: Solo / Startup Stack

Model serving: FastAPI with Docker, self-contained and portable, it works on any hosting provider.

Hosting: Render or Railway, Git-connected auto-deploys, auto SSL, $50 to $150 per month.

MLOps: MLflow is self-hosted on the same VPS, is free, and provides experiment tracking and a model registry.

Monitoring: Structured JSON logs plus UptimeRobot for uptime checks, sufficient at this stage.

CI/CD: GitHub Actions free tier auto-deploys on every push to main.

LLM option: Replicate or Modal on a pay-per-run basis, do not pay for a dedicated GPU instance until traffic justifies it.

What to avoid at this stage: Kubernetes, SageMaker, dedicated GPU instances, and enterprise monitoring tools. The operational cost in engineering time exceeds their benefit when you have one or two models and low traffic.

Archetype 2 — Growth Team Stack

Model serving: BentoML or TorchServe, multi-model support with versioning built in.

Hosting: Google Cloud Run or AWS Fargate, autoscaling, managed containers, no Kubernetes overhead.

MLOps: With MLflow and a properly maintained model registry, every deployed model must have a version string you can trace back.

Monitoring: Prometheus and Grafana for infrastructure, and Evidently AI for drift detection, both open source.

CI/CD: GitHub Actions building a Docker image, pushing to a registry, and deploying to Cloud Run with canary traffic splitting.

LLM option: vLLM on a GPU instance via Cloud Run GPU or Modal, OpenAI-compatible API, high throughput.

The critical addition versus Archetype 1: drift monitoring. Evidently, AI takes a few hours to integrate and prevents the most expensive class of production ML failure: silent model degradation that goes unnoticed for weeks.

Archetype 3: Enterprise Stack

Model serving: NVIDIA Triton Inference Server or Seldon Core, multi-framework, GPU-optimised, A/B testing built in.

Hosting: AWS SageMaker, Google Vertex AI, or Azure ML, depending on existing cloud commitment.

MLOps: Kubeflow or ZenML combined with MLflow; pipeline orchestration, lineage tracking, and audit trail.

Monitoring: Arize AI or Fiddler — explainability, bias detection, and compliance-level monitoring.

CI/CD: Full GitOps with ArgoCD or Flux, shadow and canary deployments, no manual production changes.

LLM option: vLLM cluster or SGLang on dedicated GPU nodes; AWS Inferentia3 chips for cost efficiency at scale.

Enterprise-specific requirement: model governance. Every deployed model must be traceable to a registered version, a training dataset, and a named human approver. Build this into your MLOps platform from the start. Retrofitting it across 20 production models is extremely expensive.

5. Classical ML vs LLM Deployment

In 2026, a significant share of ML deployment work involves large language models, RAG pipelines, or fine-tuned generative models. These require a fundamentally different architecture from classical ML. Using classical ML patterns for LLM deployment is one of the most common causes of poor production performance and unexpectedly high costs.

Key Architectural Differences

Model size: Classical ML models are kilobytes to hundreds of megabytes. LLMs are 4GB to over 140GB in FP16. A 7B parameter model requires approximately 14GB of GPU VRAM just to load.

GPU requirement: Classical ML runs acceptably on a CPU. LLMs require a GPU for production workloads. Running a 7B model on CPU produces latencies of 30 to 60 seconds per request, unusable for interactive applications.

Response pattern: Classical ML returns a synchronous JSON response in milliseconds. LLMs produce streaming token sequences. You must implement server-sent events or WebSocket streaming so users see tokens as they generate.

Cost model: Classical ML cost is computed in in hours at CPU rates. LLM cost is GPU hours (self-hosted) or per-token pricing (API). At low volumes, the API is almost always cheaper. Above roughly one million tokens per day, self-hosting becomes cost-competitive.

Serving framework: Classical ML: FastAPI, Flask, BentoML, and TorchServe. LLMs: vLLM for high-throughput multi-user production, SGLang for multi-turn and RAG workloads, Ollama for local development only, and llama. cpp for CPU and edge deployment.

Fine-tuning deployment: Classical ML retraining means serializing a new model. LLM LoRA fine-tuning produces small adapter files that attach to the base model at runtime, you can serve multiple fine-tuned variants from one base model, significantly reducing GPU costs.

2026 LLM tooling update: HuggingFace Text Generation Inference (TGI) was placed into official maintenance mode in December 2025. For new production LLM deployments, use vLLM or SGLang. Existing TGI deployments continue to work, but new features and model support will arrive elsewhere first.

6. Real 2026 Cost Comparison

The following estimates are for a medium-traffic classical ML API serving approximately 100,000 requests per day with a sub-200ms latency requirement. Costs are 2026 estimates and include compute, basic storage, and egress. Engineering time and third-party tooling licences are excluded.

Deployment Path	Platform	Est. Monthly Cost	Ops Effort	Best For
Self-hosted VPS	Hetzner CX31 + Docker + Nginx	$15 – $35	High	Solo dev, budget-first
PaaS managed	Render / Railway	$50 – $150	Low	Startups, fast shipping
Serverless containers	Google Cloud Run	$40 – $200	Low-Medium	Variable traffic scale-to-zero
Managed containers	AWS Fargate / ECS	$150 – $400	Medium	AWS teams, growth stage
Managed ML platform	AWS SageMaker RT Endpoint	$400 – $1,800	Medium	Enterprise AWS, multi-model
Kubernetes self-managed	EKS / GKE + custom tooling	$300 – $800	Very High	Large teams, full control

GPU Pricing Reference (On-Demand, May 2026)

GPU	VRAM	AWS ($/hr)	GCP ($/hr)	Hetzner ($/hr)	Models That Fit
T4	16 GB	~$0.53	~$0.35	~$0.22	Small DL, INT8 7B LLM
L4	24 GB	—	~$0.70	—	7B FP16 - best price/perf on GCP
A10G	24 GB	~$1.50	—	—	7B FP16, 13B INT8 LLMs
A100 40GB	40 GB	~$3.50	~$2.90	~$1.90	13B to 30B FP16 LLMs
H100 80GB	80 GB	~$12 – $16	~$10	~$5.50	70B+ LLMs, maximum throughput

Cost Optimisation Levers

1. Scale to zero during idle time. Cloud Run, Modal, and Railway do not charge when no requests come in. An always-on SageMaker endpoint at $1.50/hour costs $1,080/month even at zero traffic.

2. Quantise your model. INT8 quantisation cuts GPU VRAM by 50%. INT4 cuts it by 75% with a small accuracy trade-off. This is often the difference between needing an H100 and an A10G.

3. Cache repeated requests. 20 to 40 per cent of requests in most production systems are identical. A Redis cache layer eliminates that share of inference cost with no model changes.

4. Use spot instances for batch jobs. Spot instances cost 60 to 90 per cent less. Nightly scoring jobs tolerate interruption. Real-time APIs cannot use spot, but batch jobs absolutely should.

5. Commit to reserved instances for stable load. AWS 3-year reserved Inferentia3 instances cut SageMaker inference costs by approximately 58% versus on-demand. Commit once traffic is predictable and stable.

7. Common Stack Mistakes by Team Size

These are the patterns that most consistently cause production failures and budget overruns. The failure modes differ almost entirely between small and large teams.

Mistakes Startups Make

Deploying with enterprise-grade infrastructure from day one. Kubernetes, SageMaker, and enterprise monitoring tools require dedicated engineering time to operate. A two-person team spending 40 per cent of its bandwidth on ML infrastructure is not building a product. Start with Render or Cloud Run.

No model versioning from the first deployment. Retrofitting model versioning after six models are in production is painful. Even a basic MLflow setup from model one gives you traceable predictions and a clean rollback.

Using Lambda or serverless for real-time inference with large models. AWS Lambda cold starts of 2 to 10 seconds are unusable for user-facing predictions. Reserve serverless for lightweight, infrequent inference only.

Mistakes Growth Teams Make

Adding Kubernetes before it is operationally justified. Kubernetes is the right answer when you have multiple models with different resource profiles and a team with the skills to operate it. Cloud Run provides autoscaling with zero operational overhead for the majority of use cases.

No drift monitoring until something breaks. Teams add monitoring after a model silently degrades and causes a business incident. Evidently, AI takes a few hours to integrate and prevents the most expensive class of production ML failures.

Deploying LLMs with classical ML patterns. Running a 7B model behind Flask on a CPU server produces 30 to 60 second latencies. LLMs need GPU infrastructure, streaming responses, and a framework designed for token generation, such as vLLM.

Mistakes Enterprises Make

Underinvesting in model governance tooling. Large organisations with many production models often cannot answer who approved the current model and when. Invest in a model registry and approval workflow before you have 20 models in production.

Manual deployment gates that slow iteration. A three-week model approval process kills the ability to retrain in response to drift. Automate the validation workflow. Keep human sign-off only where regulation genuinely requires it.

Cloud lock-in without awareness. Choosing SageMaker because it is there, without evaluating the cost and lock-in implications, is a decision that becomes increasingly expensive to reverse. Evaluate the build-versus-buy question explicitly every 18 months.

8. Pre-Deployment Architecture Checklist

Validate your stack decisions against this checklist before provisioning any infrastructure. Items in the first two groups are blockers. Items in later groups represent risks you are consciously accepting if you proceed without them.

Requirements, Blockers

- Model type confirmed and hosting choice matches its compute requirements
- A latency budget is defined as a specific number, benchmarked on representative hardware, before choosing hosting
- Compliance requirements reviewed, and the chosen platform confirmed to be compliant
- Full pipeline serialized: preprocessing steps and model saved as a single object, not the model alone
- Smoke test passing: known input produces expected output inside the container environment

Infrastructure - Blockers

- Health endpoint: GET /health returns 200 with model version
- API versioned: endpoints at /v1/predict, not /predict
- Input validation active: field types, presence, and ranges validated before the model receives the request
- Secrets in environment variables: no API keys or credentials in code or Docker images
- Rollback procedure documented and tested in a non-production environment

Operational Readiness - High Priority

- Model registered with a version string that appears in every API response
- Prediction logging: every prediction recorded with input hash, output, confidence, latency, and model version
- Latency alert: fires if P99 latency exceeds the budget for more than 3 consecutive minutes
- Error rate alert: fires if 5xx rate exceeds 5% over a 5-minute window
- CI/CD pipeline active: no manual deployments to production

Drift and Quality Monitoring

- Baseline distribution captured at launch for future drift comparison
- Drift monitoring scheduled: daily or weekly job comparing distributions against the baseline
- Retraining trigger defined: team knows exactly what condition initiates a retrain
- Business metric wired: at least one downstream metric is tracked and comparable across model versions

9. Next Steps

You now have a complete framework for ML deployment architecture decisions. The next step is applying it to your current project.

Immediate Actions

- Answer the 10 framework questions for your current project and write down the answers. This surfaces assumptions that the team has not made explicit.
- Identify your archetype and compare the recommended stack against what you have today.
- Run the pre-deployment checklist against any existing production deployments. The most common findings: prediction logging is absent, drift monitoring is absent, and rollback has never been tested.
- If deploying an LLM, confirm your serving framework: vLLM for multi-user production, SGLang for multi-turn and RAG workloads, and Ollama for local development only.
- If your framework output points in multiple directions, make the trade-off explicit. Cost, simplicity, compliance, and team capacity trade off against each other. Naming the trade-off you are making is more useful than searching for a solution that avoids it.

The Full Technical Guide

This white paper covers architecture decisions. The complete technical implementation guide, step-by-step code, Docker configuration, CI/CD setup, monitoring integration, LLM deployment, and the full tools comparison - is available free on code-b.dev.

code-b.dev/blog/deploy-machine-learning-model

- Step-by-step deployment guide with copy-paste code: Flask, FastAPI, and Docker
- LLM deployment walkthrough: vLLM setup, RAG pipeline architecture, LoRA fine-tune serving
- Complete tools comparison: 14 tools, 8 evaluation dimensions, 2026 pricing
- Monitoring setup guide: Evidently AI, Prometheus with Grafana, structured prediction logging