Reference Architecture

Everpure

comet

Everpure FlashBlade and Opik: LLM Observability

Reference architecture for cloud-to-on-premises model migration

Contents

Executive summary

Enterprises that adopted cloud-hosted large language model (LLM) APIs are hitting a cost wall. As usage scales across business units, agentic workflows, and production applications, the economics of per-token cloud pricing become unsustainable. The natural response is to deploy open-weight models on premises using GPU infrastructure, but that introduces a different problem: how do you prove that your on-premises models deliver equivalent output quality, latency, and reliability to the cloud models they replace?

This reference architecture solves that problem by combining Opik, an open source LLM observability platform, with Everpure™ FlashBlade® as the high-throughput storage back end. Together, they provide the instrumentation, evaluation, and data persistence layers required to quantify model equivalence, track cost savings, and maintain observability across both cloud and on-premises inference endpoints. The architecture captures every LLM interaction as a structured trace, runs controlled experiments comparing cloud and local models against ground-truth datasets, and persists all observability data to FlashBlade S3-compatible object storage.

This reference architecture is intended for AI/ML engineers, infrastructure architects, and platform teams planning or actively executing a migration from cloud LLM APIs to on-premises GPU deployments. It provides the full workflow, including architecture, deployment steps, and two companion demo applications that illustrate the cloud-to-on-premises evaluation process in practice.

The problem: Cloud LLM costs at enterprise scale

Cloud-hosted LLMs make prototyping easy, but prototypes that appear successful at pilot scale can become completely unviable when pushed into production across thousands of users and workflows. One enterprise reported exceeding its total annual OpenAI budget by $10M in Q1 alone. The math is simple: per-token pricing means costs scale linearly with usage. The problem is that usage is scaling exponentially as organizations embed LLM calls into customer-facing applications, internal tooling, and multi-step agentic workflows where a single user request can trigger dozens of model invocations.

The engineering response is straightforward: deploy open-weight models (Qwen3, Llama, Mistral, etc.) on premises using purchased GPU infrastructure. A single NVIDIA B200 running Qwen3-32B via vLLM can serve over 10,000 requests per hour at typical enterprise prompt lengths with a fixed infrastructure cost, eliminating per-token charges entirely. The hardware pays for itself within months at enterprise-scale usage.

But deploying the model is not the hard part. The hard part is validating equivalence. Your production workflows were built against GPT-5 or Gemini. Your prompts were tuned for those models, and your users have expectations calibrated to their output quality. Switching to Qwen3-32B without quantifying the accuracy and quality delta leaves you with no way to answer the only question that matters: is this model good enough? And in regulated industries like financial services, healthcare, or legal, that switch carries compliance risk.

This is where LLM observability becomes the control layer for migration. The workflow is concrete:

1. **Log traces** from your existing cloud model to establish a performance baseline.
2. **Curate an evaluation dataset** from production traffic, selecting problematic or representative traces.
3. **Run experiments** comparing your on-premises candidate against the cloud model on that dataset.
4. **Optimize prompts** using the agent optimizer to systematically close any accuracy gaps.
5. **A/B test live** with online evaluation to validate that real-world performance holds under production traffic.
6. **Fully migrate** once online metrics confirm parity.

Without this instrumentation at every stage, you cannot make a defensible decision to migrate.

Opik: LLM observability for the enterprise

Opik is an open source LLM evaluation and observability platform built by Comet. It provides LLM tracing, experimentation, cost tracking, prompt optimization, and online evaluation capabilities purpose-built for production LLM workloads. Here is how each capability maps to the migration use case.

Tracing

Every LLM interaction generates a trace: the input prompt, model output, latency, token counts, tool calls, and metadata. In an agentic workflow, a single user query can trigger multiple LLM calls, tool invocations, and handoffs. Opik captures the full execution tree as a hierarchical trace.

This matters for migration because traces are how you debug regressions. When your on-premises model returns a wrong answer to a customer relationship management (CRM) query, the trace shows you exactly what happened: the SQL the agent generated, the tool output it received, and where it deviated from the expected response.

Without traces, debugging a model running locally means reading logs and guessing. Say a user reports that the CRM agent returned the wrong revenue number for Q3. Was it the prompt? Did the model generate bad SQL? Did the retrieval step pull the wrong schema? Did a tool call return an unexpected format that the model misinterpreted? In a multi-step agentic workflow, the failure could originate at any point in the chain. Without a structured trace capturing each step's input, output, and latency, you're left doing a binary search through print statements across services to find the failure point. Traces turn that from a multi-hour debugging session into a click-through in the Opik UI.

Opik also supports human feedback integration. Annotators can attach quality scores, corrections, and labels directly to traces through the UI or API. Users can rate responses with thumbs-up/thumbs-down feedback or numerical scores that are logged as structured feedback on the trace via the UI or API. Opik also lets you curate evaluation datasets directly from production traffic. When you spot a problematic or noteworthy trace, you can add it to a dataset with a few clicks. Over time, this builds a ground-truth evaluation set drawn from real user interactions rather than synthetic examples, giving your experiments a dataset that reflects how your application is actually used.

Experiments: Model-vs.-model evaluation

The experiment capability is the core mechanism that makes migration quantifiable. Given a golden dataset—a set of questions with human-verified ground-truth answers—Opik runs the same prompts against multiple models or configurations, captures structured results, and surfaces them in a comparative dashboard.

The experiment pipeline follows a consistent pattern regardless of use case. You define a dataset of evaluation examples and run them against each model or configuration you want to compare. Opik captures the full response and scores it against a set of metrics you define. These can be LLM-judged metrics like answer correctness and hallucination detection, deterministic metrics like string similarity or format compliance, custom G-Eval metrics tailored to your specific use case, or agent-level metrics that evaluate tool use and trajectory quality. Not every evaluation requires verified ground truth, and in practice many teams find ground-truth datasets difficult to assemble. Opik supports reference-free metrics that assess response quality, coherence, and safety without a known-correct answer, making it possible to start evaluating immediately against production-like inputs. Results are surfaced in a comparative dashboard where you can view scores side by side, filter by metric, and drill into individual examples where models diverge.

FIGURE 1 Opik experiment dashboard showing side-by-side model comparison

The output is not a subjective assessment but a table of scores. For example, "Cloud model scored 0.92 on answer correctness; local model scored 0.85. Local model hallucination rate: 3%. Cloud: 1%." That is a decision you can defend to your VP of engineering or your compliance team.

Cost tracking and trace pricing

Opik tracks token-level pricing per trace, giving teams visibility into the actual cost of each LLM interaction. Aggregated across workflows, this turns trace data into a cost model. You can answer directly: this CRM agent workflow costs $X per thousand invocations on GPT-5 at $15 per million input tokens, and it would cost $Y on Qwen3-32B served locally at your blended GPU infrastructure rate. That comparison, grounded in real production traffic rather than back-of-envelope estimates, is what gives your finance team the number it needs alongside the accuracy scores your engineering team cares about.

Cost visibility bridges the gap between engineering metrics and executive decision-making. Your engineering team cares about answer correctness scores, while your finance team cares about the $3M budget overrun. Opik gives both teams the data they need in the same dashboard.

Prompt optimization

When switching from a frontier model to a smaller on-premises model, prompt engineering is often the lever that closes the remaining accuracy gap. Opik's prompt optimization tooling systematically improves prompt performance against evaluation metrics, replacing manual trial and error with a structured search over prompt variants.

FIGURE 2 Dashboard showing the best prompt from the optimization runs on a curated test dataset

The optimization workflow takes your current system prompt, generates variations using a reasoning model, evaluates each variant against your golden dataset on the metrics you define, and promotes the best-performing prompt. The Opik dashboard surfaces the results of each optimization round, showing how each variant scored so you can identify which prompt performs best on your evaluation criteria. The result is a prompt specifically tuned for your on-premises model's strengths and weaknesses, narrowing the gap to cloud model performance without changing the model itself.

Online evaluation

Offline experiments tell you whether an on-premises model can match cloud performance on a static dataset. Online evaluation tells you whether it holds up under live traffic. Opik lets you define scoring rules that automatically evaluate production traces using LLM-as-a-Judge metrics. You configure a metric (hallucination detection, answer relevance, moderation, or a custom scorer tailored to your use case), set a sampling rate, and Opik scores incoming traces continuously without manual review.

This is the A/B testing layer of the migration workflow. Run both your cloud and on-premises models in parallel, apply the same online scoring rules to both, and monitor the results in real time through the Opik dashboard. If the on-premises model's scores hold steady against the cloud baseline under production conditions, you have the evidence to fully cut over. If scores drift, you catch it immediately rather than learning from user complaints.

Everpure FlashBlade: Infrastructure for LLM observability

Everpure FlashBlade is a unified, fast file and object storage platform. In this architecture, it serves a specific role: the S3-compatible object storage back end for ClickHouse, which is Opik's analytics database.

Opik stores all trace data, experiment results, feedback scores, and observability metadata in ClickHouse. ClickHouse natively supports S3 as a storage back end, and FlashBlade speaks S3 natively as a first-class protocol. There is no translation layer, no gateway process proxying requests, and no additional service to deploy, monitor, and scale. ClickHouse writes directly to FlashBlade via standard S3 API calls.

Compare this to architectures that rely on lightweight S3-compatible gateways: those introduce an additional process in the data path that must be sized, maintained, and troubleshot when it becomes the bottleneck. FlashBlade eliminates that layer entirely, reducing both operational complexity and tail latency on every write and read.

As trace volumes grow to thousands of traces per minute across multiple models, agents, and workflows, the storage back end must sustain high write throughput for continuous trace ingestion and fast analytical reads for dashboard queries. The scale-out architecture of FlashBlade, designed for unstructured data workloads at petabyte scale, handles this natively. Write throughput scales linearly with blade count, and read performance remains consistent as data volumes grow. A single-node S3 gateway becomes the ceiling on your observability pipeline. FlashBlade removes that ceiling.

FlashBlade sits within a broader Everpure AI infrastructure stack. For organizations running GPU inference on premises, the stack also offers the Everpure Key-Value Accelerator (KVA) for inference optimization workloads and Portworx® for Kubernetes-native persistent storage. This reference architecture focuses specifically on the role of FlashBlade as the observability data tier, but the same infrastructure often supports the model serving and training pipelines as well.

Architecture overview

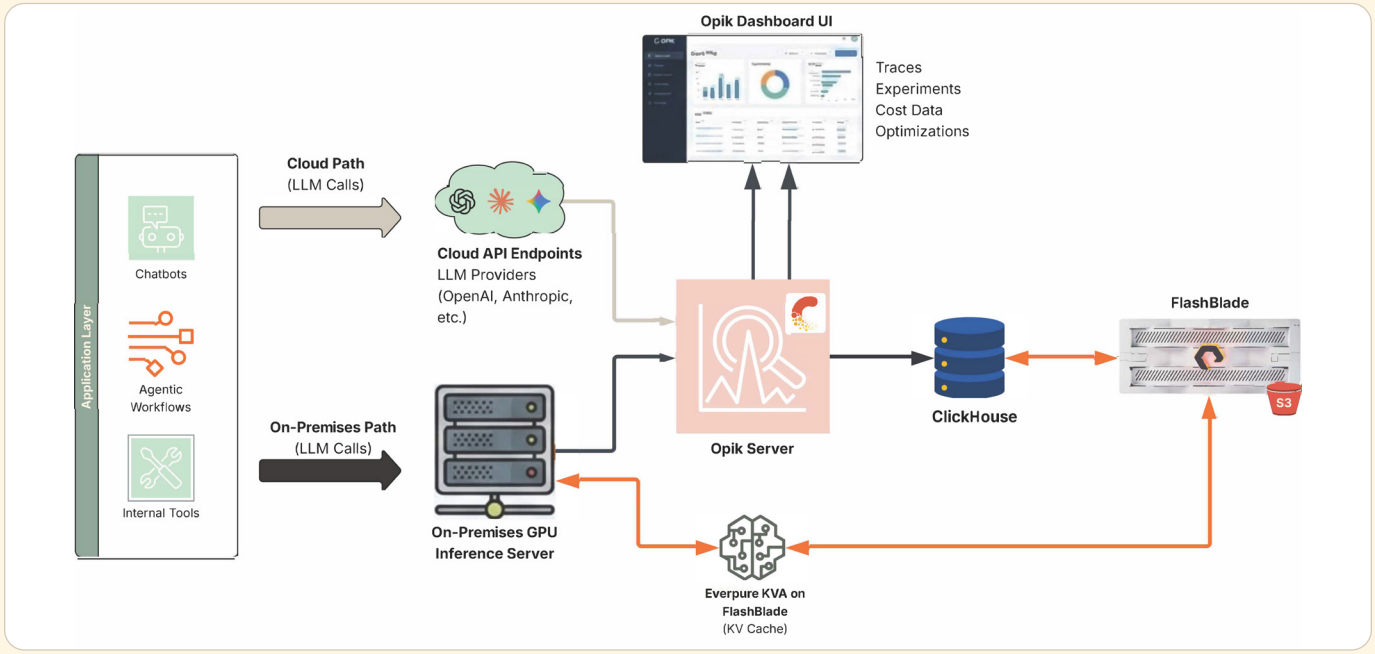

FIGURE 3 Integrated architecture showing cloud and on-premises inference paths feeding into a unified Opik observability pipeline, with ClickHouse persisting trace data to FlashBlade via S3 and Everpure KVA accelerating on-premises inference through storage-backed key-value caching

The integrated architecture consists of the following components:

- **Application layer**: any application that makes LLM calls, including customer-facing chatbots, multi-step agentic workflows, internal productivity tools, retrieval-augmented generation (RAG) pipelines, or batch processing jobs. The application is instrumented with the Opik SDK, which captures traces regardless of whether the underlying call routes to a cloud API or a local model. The SDK integration is lightweight, typically a decorator or context manager wrapping each LLM call.
- **GPU inference cluster**: on-premises models are served via vLLM on GPU nodes. The vLLM server exposes an OpenAI-compatible API on a configurable port, making it a drop-in replacement for cloud endpoints from the application's perspective.
- **Everpure KVA**: Everpure KVA integrates with vLLM to persist and reuse key-value caches on FlashBlade, eliminating redundant prefill computation when the same or overlapping prompts appear across requests. In this architecture, KVA means your GPUs spend more time generating tokens and less time recomputing context they've already seen.
- **Cloud LLM API endpoints**: OpenAI, Anthropic, or other cloud providers. These remain available for comparison and as fallback during migration.
- **Opik server**: the observability platform, deployed via Helm chart on Kubernetes or Docker Compose for development. It receives traces from the Opik SDK, stores them in ClickHouse, and serves the dashboard UI.
- **ClickHouse**: the columnar analytics database that stores all trace data, experiment results, and metrics. It is deployed as part of the Opik Helm chart using the Altinity Kubernetes Operator for ClickHouse.
- **FlashBlade**: provides the S3-compatible endpoint that Opik's back end writes to and that ClickHouse uses for backup persistence. It replaces the bundled MinIO in production deployments.

The data flows as follows:

Application makes LLM call (cloud or local) → Opik SDK captures trace with @opik.track decorator → trace stored in ClickHouse → ClickHouse persists/backs up to FlashBlade via S3 → Opik dashboard surfaces analytics, experiments, and cost data.

The same architecture captures traces from both cloud and on-premises models, tagged by provider, model, and deployment type. This enables side-by-side comparison without separate infrastructure.

Deployment guide: Installing Opik with FlashBlade

Prerequisites

FlashBlade S3 configuration

- A FlashBlade array with an S3 endpoint accessible from your Kubernetes cluster (for example, http://192.168.xxx.xxx:80)
- An S3 bucket created on FlashBlade (for example, opik-data)
- An access key and secret key with read/write permissions on the bucket

Kubernetes cluster

- Kubernetes 1.24+ with Helm 3.x installed
- Sufficient resources for the Opik stack: ClickHouse (recommend 8GB RAM, 50GiB storage), MySQL, Redis, and the Opik back-end/front-end pods
- Network connectivity from the cluster to the FlashBlade S3 endpoint

GPU infrastructure (for on-premises inference)

- One or more NVIDIA GPUs with sufficient VRAM for your target model (for example, 64GB for Qwen3-32B at FP16)
- vLLM installed and configured with tool-calling support

Opik installation via Helm chart

Clone the Opik repository and install via Helm:

```
git clone https://github.com/comet-ml/opik.git
cd opik/deployment/helm_chart/opik
helm dependency update
helm install opik . -f values.yaml -f values-flashblade.yaml
```

The values-flashblade.yaml override file configures FlashBlade as the S3 back end. Create it with the following content.

Configuring FlashBlade as the S3 back end

The key configuration change is disabling the bundled MinIO and pointing Opik's back end to your FlashBlade S3 endpoint.
Create a values-flashblade.yaml file:

```
# Disable bundled MinIO --- use FlashBlade instead
minio:
 enabled: false
component:
 backend:
  env:
   # FlashBlade S3 configuration
   S3_BUCKET: "opik-data"                                    # Your FlashBlade bucket name
   S3_REGION: "us-east-1"                                    # Required but can be any value for S3-compatible storage
   S3_ENDPOINT_URL: "http://<FLASHBLADE_S3_ENDPOINT>"        # e.g., http://192.168.xxx.xxx:80
   AWS_ACCESS_KEY_ID: "<FLASHBLADE_ACCESS_KEY>"              # Your FlashBlade S3 access key
   AWS_SECRET_ACCESS_KEY: "<FLASHBLADE_SECRET_KEY>"          # Your FlashBlade S3 secret key
```

Replace the placeholder values:

- <FLASHBLADE_S3_ENDPOINT>: your FlashBlade S3-compatible endpoint (IP or FQDN with port)
- <FLASHBLADE_ACCESS_KEY>: the access key ID generated on FlashBlade for S3 access
- <FLASHBLADE_SECRET_KEY>: the corresponding secret access key

Configuring ClickHouse backups to FlashBlade

ClickHouse supports native S3 backups using the BACKUP TO S3() coommand. Enable automated backups in your values override:

```
clickhouse:
 backup:
  enabled: true
  bucketURL: "http://<FLASHBLADE_S3_ENDPOINT>"
  secretName: "clickhouse-backup-secret"
  schedule: "0 0 * * *"    # Daily at midnight (adjust as needed)
  successfulJobsHistoryLimit: 1
  command:
   - /bin/bash
   - '-cx'
   - |-
    export backupname=backup$(date +'%Y%m%d%H%M')
    echo "BACKUP ALL EXCEPT DATABASE system TO S3('http://<FLASHBLADE_S3_ENDPOINT>/<BUCKET_NAME>/clickhouse-backups/${backupname}/', '$ACCESS_KEY', '$SECRET_KEY');" | \
    clickhouse-client -h chi-opik-clickhouse-cluster-0-0 --send_timeout 600000 --receive_timeout 600000 --port 9000
```

Create the Kubernetes secret containing FlashBlade credentials:

```
kubectl create secret generic clickhouse-backup-secret \
 --from-literal=access_key_id=<FLASHBLADE_ACCESS_KEY> \
 --from-literal=access_key_secret=<FLASHBLADE_SECRET_KEY>
```

Verifying the integration

After deployment, verify each layer of the stack:

1. Confirm the Opik pods are running:

```
kubectl get pods | grep opik
```

You should see pods for opik-backend, opik-frontend, opik-python-backend, ClickHouse, MySQL, Redis, and ZooKeeper.

2. Verify FlashBlade connectivity from the back-end pod:

```
kubectl exec -it deployment/opik-backend -- curl -s http://<FLASHBLADE_S3_ENDPOINT>
```

3. Send a test trace via the Opik SDK:

```
import opik
opik.configure(use_local=True)
@opik.track(name="test_trace", project_name="Integration Test")
def test_function():
  return "FlashBlade integration verified"
test_function()
```

4. Confirm that data appears in the Opik UI at http://<OPIK_FRONTEND_URL>:5173. Navigate to the “Integration Test” project and verify the trace is visible.

5. Verify backup execution (if backup is enabled):

```
# Check the backup CronJob status
kubectl get cronjobs | grep clickhouse-backup
# Check completed backup jobs
kubectl get jobs | grep clickhouse-backup
```

Production hardening

Backup strategy: The default configuration runs daily ClickHouse backups to FlashBlade. For production, consider more frequent backups (for example, every 15 minutes during initial deployment) and test the restore procedure:

```
# Restore from a specific backup
kubectl exec -it chi-opik-clickhouse-cluster-0-0 -- clickhouse-client
```

```
RESTORE ALL FROM S3(
 'http://<FLASHBLADE_S3_ENDPOINT>/<BUCKET_NAME>/clickhouse-backups/<backup_name>/',
 '<ACCESS_KEY>',
 '<SECRET_KEY>'
);
```

Retention policies: Configure ClickHouse time-to-live (TTL) policies on trace tables to automatically expire old data. The default Opik configuration sets 30-day TTLs on system log tables (error_log, latency_log, metric_log, and query_metric_log). Adjust trace retention based on your compliance requirements.

Scaling considerations: For high-volume deployments (10,000+ traces per minute), increase ClickHouse resources and consider multi-shard configurations by adjusting clickhouse.shardsCount and clickhouse.replicasCount in the Helm values. FlashBlade scales horizontally by adding blades—no reconfiguration of the Opik stack is required.

Secrets management: The previous example uses inline credentials for clarity. In production, use Kubernetes Secrets or a secrets manager (such as HashiCorp Vault or AWS Secrets Manager) to manage FlashBlade access keys. Reference them via envFrom in the Helm values:

```
component:
 backend:
  envFrom:
   - secretRef:
     name: flashblade-s3-credentials
```

Conclusion

Migrating from cloud-hosted LLMs to on-premises inference is an infrastructure decision that requires engineering rigor. The cost savings are real: eliminating per-token pricing at enterprise scale can reclaim millions of dollars annually. But those savings only materialize if the on-premises models deliver equivalent quality, and proving equivalence requires instrumentation, not intuition.

Opik provides the observability and evaluation layer that makes this migration quantifiable:

- **Traces capture** every interaction for debugging and audit.
- **Experiments** compare cloud and on-premises models against ground-truth datasets with structured metrics.
- **Prompt optimization** closes accuracy gaps without changing models.
- **Cost tracking** connects engineering metrics to business outcomes.

Companion demonstrations, a CRM agent, and a financial services agent are available for teams evaluating this architecture. Both implement the full workflow described in this reference architecture, including side-by-side cloud vs. on-premises comparison, experiment evaluation, and prompt optimization.

FlashBlade provides the storage infrastructure that this observability layer demands. As trace volumes scale with production traffic, FlashBlade S3-compatible object storage delivers the write throughput for continuous ingestion and the read performance for real-time dashboards, without becoming the bottleneck in your observability pipeline.

Stop guessing whether your local model is good enough. Measure it.

See how Everpure FlashBlade and KVA fit into your AI deployment.

Connect with Your Account Team to Scope the Architecture for Your Environment

ID18120-02-en-04/26