KubeCon CloudNativeCon India 2026

#KubeCon #CloudNativeCon

The Hidden Cost of ML Data Lifecycles in Kubernetes

Yashasvi Misra
KubeCon + CloudNativeCon India 2026

KubeCon CloudNativeCon India 2026

Hello there 👋

Yashasvi Misra (Yashi)

- 🚀 SWE @Everpure (formerly Pure Storage)
- 💻 Open Source Contributor
- 🌎 Community Builder
- 🌈 Diversity in Tech Advocate

KubeCon CloudNativeCon India 2026

It started with a question I could not answer.

Three weeks into running my ML pipeline on Kubernetes,
my churn model quietly got worse.

The cluster was green. The pipeline had succeeded.
I opened the PVC. I looked at the files.

"Which file actually trained this model?"

I had no idea.

Background: What I Was Building

KubeCon CloudNativeCon India 2026

The System

Goal: build a churn prediction pipeline
Type: personal project, real Kubernetes
Platform: Kubernetes on GKE
Retrain: weekly on fresh input data

My Stack

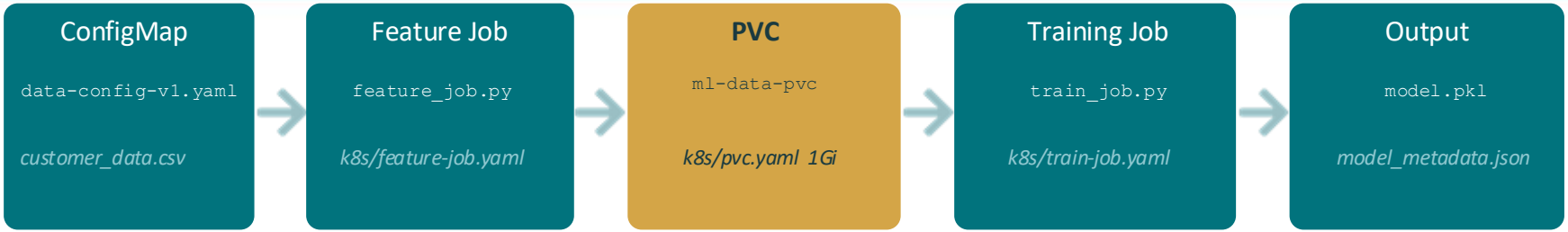
```
No Kubeflow. No Argo. No MLFlow.
Three Kubernetes Jobs:
  feature-job   (feature_job.py)
  train-job     (train_job.py)
  inspect via   (inspect_model.py)
Shared storage: one PVC -> ml-data-pvc
```

Pipeline launched — *Incident* — *Root cause* — *Rebuilt*

I just wanted to keep it simple three jobs, one PVC, everything in plain sight.

The Pipeline I Designed : Intentionally Simple

KubeCon CloudNativeCon India 2026



What I thought at launch:

Simple. Debuggable. The PVC is just fast shared storage between jobs.

feature_job.py writes features_latest.csv and records the version in metadata.txt.

train_job.py reads both. Kubernetes orchestrates everything.

What I did not think about:

train_job.py reads EXPECTED_DATASET_VERSION from the environment.

It reads features from /shared/features_latest.csv without checking whether that file matches the expected version.

The metadata.txt is written but never read.

Week 1: The Pipeline Worked. I Was Happy.

KubeCon CloudNativeCon India 2026

I ran kubectl apply -f k8s/data-config-v1.yaml, then the feature job, then the training job. This is what I saw.

`kubectl logs job/feature-job`

```
ymisra@ymisra--Mac16 kubecon-ml-data-demo % kubectl logs job/feature-job
Reading dataset from /input/customer_data.csv
Dataset version: v1
Feature generation complete
Wrote /shared/features_latest.csv
Wrote /shared/metadata.txt
```

`kubectl logs job/train-job`

```
ymisra@ymisra--Mac16 kubecon-ml-data-demo % kubectl logs job/train-job
Starting training job
Expected dataset version: v1
Reading features from /shared/features_latest.csv
Training complete
Accuracy=1.0

Feature metadata used by training:
features_generated_from=v1
generated_at=2026-06-14T09:03:39.168535Z
feature_file=features_latest.csv


Model written to /shared/model.pkl
Model metadata written to /shared/model_metadata.json
```

- features_generated_from=v1
- expected_dataset_version: v1
- Accuracy = 1.0

Dataset version flows cleanly through the pipeline. This is the baseline. I thought this was the whole story.

Three Weeks Later: One Small Decision I Barely Noticed

I didn't set out to create a mess. I made one small convenience decision. Then another. Here is the exact sequence.

KubeCon CloudNativeCon India 2026

Week 1	`features_v1.csv`	Explicit. Versioned. I knew exactly what this was.
Week 2	`features_latest.csv`	Renamed so train_job.py doesn't need updating each week.
Week 4	`features_latest_backup.csv`	I kept a copy before trying a schema experiment.
Week 6	`features_final.csv`	This had the right schema.
Week 8	`features_final_fixed.csv`	One-off fix that became the permanent source of truth.

Each rename made sense in the moment. None of them felt like a data governance decision. That is exactly the problem.

What I Saw When I Finally Read the Logs

New dataset arrived. I applied data-config-v2.yaml. I did not rerun the feature job. I assumed features were fine. I reran the training job. This is what the logs said.

KubeCon CloudNativeCon India 2026

```
ymisra@ymisra--Mac16 kubecon-ml-data-demo % kubectl logs job/train-job
Starting training job
Expected dataset version: v2
Reading features from /shared/features_latest.csv
Training complete
Accuracy=1.0

Feature metadata used by training:
features_generated_from=v1
generated_at=2026-06-18T19:44:12.748177Z
feature_file=features_latest.csv


Model written to /shared/model.pkl
Model metadata written to /shared/model_metadata.json
```

The log says:

```
Expected dataset version: v2
features_generated_from=v1
```

The pipeline says:

```
Training complete. Accuracy=1.0.
Model written to /shared/model.pkl.
```

What Kubernetes Told Me When I Checked

My first instinct was to check if something had failed. I ran every health check I knew.

KubeCon CloudNativeCon India 2026

```
ymisra@ymisra--Mac16 kubecon-ml-data-demo % kubectl get pods
NAME               READY   STATUS      RESTARTS   AGE
feature-job-jz4sm  0/1     Completed   0          4m13s
pvc-inspector      0/1     Completed   0          4d10h
train-job-rvb25    0/1     Completed   0          88s
ymisra@ymisra--Mac16 kubecon-ml-data-demo % kubectl get jobs
NAME          STATUS     COMPLETIONS   DURATION   AGE
feature-job   Complete   1/1           2s         4m30s
train-job     Complete   1/1           3s         105s
ymisra@ymisra--Mac16 kubecon-ml-data-demo % kubectl get pvc
NAME         STATUS   VOLUME                                     CAPACITY   ACCESS MODES   STORAGECLASS   VOLUMEATTRIBUTESCLASS   AGE
ml-data-pvc  Bound    pvc-120f1246-0386-4ffa-a780-978eff61260d   1Gi        RWO            standard       <unset>                 4d10h
```

Kubernetes reports on compute state not data state.

Completed -> Completed -> 1/1 -> 1/1 -> Bound. Everything it promised to do, it did.

What inspect_model.py Showed Me

I ran inspect_model.py a script I had written specifically to read metadata off the PVC. This is when I understood exactly what had happened.

KubeCon CloudNativeCon India 2026

`kubectl exec pvc-inspector -- python /app/inspect_model.py`

```
ymisra@ymisra-Mac16 kubecon-ml-data-demo % kubectl exec pvc-inspector -- python /app/inspect_model.py
Files on PVC:
- /shared/features_latest.csv: exists
- /shared/metadata.txt: exists
- /shared/model.pkl: exists
- /shared/model_metadata.json: exists

Feature metadata:
features_generated_from=v1
generated_at=2026-06-18T19:44:12.748177Z
feature_file=features_latest.csv


Model metadata:
{
  "expected_dataset_version": "v2",
  "trained_at": "2026-06-18T19:46:58.411191Z",
  "features_file": "features_latest.csv",
  "feature_metadata": "features_generated_from=v1\ngenerated_at=2026-06-18T19:44:12.748177Z\nfeature_file=features_latest.csv\n",
  "accuracy": 1.0
}
```

`kubectl exec pvc-inspector -- find /shared -maxdepth 2`

```
ymisra@ymisra-Mac16 kubecon-ml-data-demo % kubectl exec pvc-inspector -- find /shared -maxdepth 2
/shared
/shared/cache
/shared/features_latest.csv
/shared/model_metadata.json
/shared/metadata.txt
/shared/tmp
/shared/features_final.csv
/shared/features_latest_backup.csv
/shared/model.pkl
/shared/features_final_fixed.csv
/shared/old_models
```

model_metadata.json says expected_dataset_version: v2. Feature metadata says features_generated_from=v1.

Nobody knows.

Which file trained this?

The Gap I Had Built Without Realising

The more I looked at this, the more I understood it wasn't a bug. It was a design gap. I had two lifecycles and I had only managed one of them.

KubeCon CloudNativeCon India 2026

Kubernetes managed compute

- When did the pod run?
- Did the job succeed?
- Were resources within limits?
- Is the PVC mounted?

All four: Kubernetes answered correctly.

Nobody managed data

- Which dataset trained this model?
- Are features in sync with dataset?
- When were features last generated?
- Can we reproduce this run?

All four: nobody could answer.

Kubernetes successfully orchestrated every container. It never promised to orchestrate the lifecycle of your data.

What It Actually Cost Us

This is what the gap between building and understanding actually looks like.

KubeCon CloudNativeCon India 2026

11 days

looking in the wrong place

I kept reading model code that was perfectly fine. The problem was upstream, in a metadata file I had written myself.

I stopped

trusting my own pipeline

Every output felt like a question mark. That feeling when you are not sure if your own system is lying to you is the worst part.

I could not

answer a basic question

Which data trained this model? I built the whole thing and I could not answer that. That is what made me take this seriously.

I rebuilt

from scratch to understand

Not because the code was broken. Because I did not trust what I had built anymore. The rebuild was how I finally understood the gap.

Four Things I Had Been Doing Wrong : Without Knowing It

KubeCon CloudNativeCon India 2026

Looking back at my own code, each failure has a name. I was running all four.

Forever PVC

ml-data-pvc with no ownership, no expiry, no audit trail. It grew because nothing ever cleaned it.

features_latest.csv

A filename that describes intention, not content. It stopped being 'latest' the moment I forgot to rerun the feature job.

No version contract

feature_job.py writes version to metadata.txt. train_job.py reads EXPECTED_DATASET_VERSION. Neither checks the other.

Silent success

The mismatch was recorded in model_metadata.json. Nothing read it at training time. The job succeeded on wrong data.

What I Changed in the Rebuild

The rebuild took three weeks. The insight took one afternoon.
Here is what it looked like after.

KubeCon CloudNativeCon India 2026

```
ymisra@ymisra--Mac16 kubecon-ml-data-demo % kubectl exec pvc-inspector -- find /shared/features /shared/models -type f
/shared/features/metadata_v2_20241201.json
/shared/features/features_v2_20241201.csv
/shared/models/model_metadata_v2_20241201.json
/shared/models/model_v2_20241201.pkl
ymisra@ymisra--Mac16 kubecon-ml-data-demo % kubectl exec pvc-inspector -- cat /shared/models/model_metadata_v2_20241201.json
{"expected_dataset_version":"v2","features_version":"v2","trained_at":"2024-12-01T14:22:31Z","features_file":"features_v2_20241201.csv","accuracy":0.94}
```

What changed:

Versioned filenames with dates.
Metadata JSON alongside every artifact.
Version check fails training on mismatch.

What did not change:

The Kubernetes Jobs. The training code.
The cluster config. The compute layer.
Only the data discipline changed.

Three Rules I Now Treat as Non-Negotiable

KubeCon CloudNativeCon India 2026

I could have read these in a talk. I learned them from an incident. Here they are.

1

Data lives outside Kubernetes.

A PVC is scratch space between jobs not a data store. Object storage (S3, GCS) is the data plane. Do not confuse the two.

2

Every artifact carries its version always.

In the filename. In a metadata JSON. In both. No 'latest'. No 'final'. If you cannot say what version it is, it does not have a version.

3

A version mismatch must stop the pipeline not log a warning.

If train_job.py reads features_generated_from=v1 and EXPECTED_DATASET_VERSION is v2, it fails. Loudly. Before training. Every time.

Models Are Temporary.
Data Decisions Compound.

I built a healthy Kubernetes pipeline.

It successfully trained wrong models for three weeks.

Kubernetes did its job.
I hadn't done mine.

Thank You !

Let's Connect

Yashasvi Misra