GitOps for HPC

Bringing Cloud-Native DevOps Practices to High Performance Computing Environments

Pavan Madduri

Senior Platform Engineer @ W.W.Grainger | CNCF Kubestronaut

Productivity, Performance, and the HPC Pipeline

March 16–20, 2026 · Chicago, IL

About the Speaker

Pavan Madduri

Senior Platform Engineer • W.W. Grainger

KCNA
Kubernetes & Cloud Native Assoc.

CKA
Certified Kubernetes Admin

CKAD
Certified K8s App Developer

CKS
Certified K8s Security Specialist

KCSA
K8s Cluster Security Assoc.

CNCF Kubestronaut • All 5 K8s Certs

🔧 Open Source Contributions

⟶ 13+ PRs across 12 CNCF & ASWF projects

⟶ Contributions to KEDA, Kubernetes, OpenTelemetry, ArgoCD, Crossplane

⟶ HPC-driven improvements that benefit the entire community

🏗️ Platform Engineering at Scale

⟶ 850+ microservices on GitOps platform

⟶ Multi-cluster EKS fleet management with ArgoCD & Crossplane

⟶ Built reusable Helm charts and self-service developer platform

Bridging cloud-native practices with high performance computing

Two Worlds Converging

Can GitOps work at HPC scale?

Traditional HPC

× Manual scripts and schedulers

× Complex dependency management

× Per-admin tribal knowledge

× Long change windows

AI/ML

Cloud-Native

✓ GitOps declarative configs

✓ Kubernetes orchestration and CI/CD

✓ Self-service platforms

✓ Instant rollbacks

AI/ML workloads blur these lines.
We can adopt battle-tested DevOps practices for HPC today.

HPC Deployment Pain Points

Sound familiar?

❌ Manual Deployment Scripts

Custom scripts per cluster, maintained by individual admins.
Different scripts for each system. Tribal knowledge.

⏰ Long Deployment Cycles

Hours or days due to manual processes and change windows.
Stalls innovation and experiment iteration.

🔥 Configuration Drift

What's running doesn't match documentation.

850+ apps: **GitOps reduced deployment failures by 87%**

🔍 Limited Visibility

When something breaks, troubleshooting is painful.
No single source of truth for deployments.

This is where GitOps can help. ↓

Low Sensitivity

What is GitOps?

Git becomes the single source of truth for infrastructure and applications

❌ TRADITIONAL: PUSH

Developer → CI Pipeline → **kubectl apply**

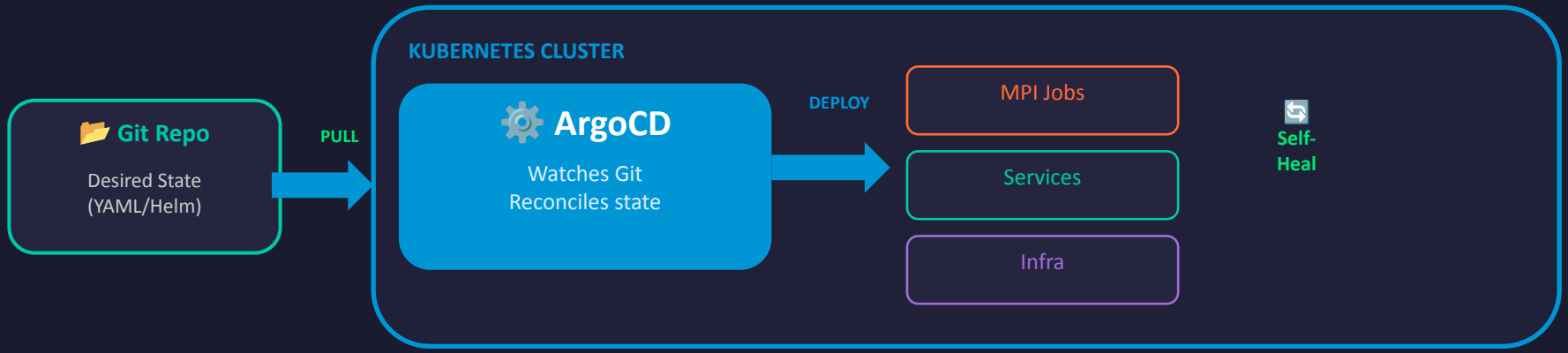
- × CI pushes changes into cluster externally
- × Requires credentials outside the cluster
- × No self-healing if state drifts
- × Audit trail: scattered across CI logs

VS

✅ GITOPS: PULL

Git Repo ← **ArgoCD** (in-cluster)

- ✓ Agent inside cluster pulls from Git
- ✓ No external credentials needed
- ✓ Self-heals: auto-reverts manual changes
- ✓ Audit trail: Git history = deployment log



Key insight: The agent lives inside the cluster and pulls — no external access needed

PILLAR 1

Continuous Delivery

ArgoCD for HPC Deployments

Deployment time: hours → minutes


```yaml
# HPC Job Helm Chart values.yaml (GitOps managed)
replicaCount: 10
image:
  repository: ecr.io/mpi-simulation
  tag: v2.3.0

resources:
  limits:
    nvidia.com/gpu: "2"
    memory: "32Gi"
  requests:
    cpu: "8"
    memory: "16Gi"

jobConfig:
  scheduler: kubernetes
  parallelism: 10
  completions: 100
```

THE PATTERN

→ Define workloads as Helm charts

→ Store configs in Git (templates, quotas)

→ ArgoCD detects changes in seconds

→ Auto rollout with health checks

Impact: 2-4 hour deployments → under 10 minutes | Full audit trail | Instant rollback

PILLAR 2

Infrastructure as Code

Crossplane for Infrastructure

Self-service GPU clusters via Git commits


```yaml
apiVersion: platform.grainger.com/v1alpha1
kind: EKSClaim
metadata:
  name: compute-cluster-a100
  namespace: clusters
spec:
  id: compute-cluster-a100
  parameters:
    region: "us-east-1"
    nodeGroups:
      - instanceType: p4d.24xlarge    # A100 GPUs
        minSize: 0
        maxSize: 10
      - instanceType: c5.4xlarge      # CPU compute
        minSize: 2
        maxSize: 20
```

WHY CROSSPLANE FOR HPC

→ Commit YAML → cluster provisions

→ Delete YAML → auto cleanup

→ Same GitOps for apps AND infra

→ Self-service — no tickets, no waiting

Impact: Manual CloudFormation/Terraform runs → Self-service Git commits

PILLAR 3

Event-Driven Autoscaling

KEDA for Burst Workloads

HPC workloads are bursty — idle most of the time, then sudden spikes

Idle
2 pods
Queue: 0

→

Burst
500 jobs arrive
Queue: 500

→

Scaling
KEDA ⟶ 50 pods
Auto-provisioned

→

Done
0 pods
Scale to zero!

```yaml
apiVersion: keda.sh/v1alpha1
kind: ScaledObject
metadata:
  name: hpc-batch-processor
spec:
  scaleTargetRef:
    name: compute-worker
  minReplicaCount: 0        # Scale to zero!
  maxReplicaCount: 100
  triggers:
    - type: aws-sqs-queue
      metadata:
        queueURL: https://sqs.us-east-2...
        queueLength: "10"   # 1 pod per 10 msgs
```

KEDA HANDLES IT

- ⚡ Monitors queues, metrics, and cron
- ⚡ Scales workers 0 ⟶ 100 on demand
- ⚡ Scale to ZERO when idle
- ⚡ Works with SQS, Kafka, RabbitMQ

Impact: 30% infrastructure cost reduction via scale-to-zero | Instant burst capacity

← Real workload pattern, KEDA matches it

Low Sensitivity

Observability with OpenTelemetry

"You can't manage what you can't measure"

📊 Metrics

- Resource utilization per job
- Queue depth & wait times
- Job completion rates
- Cost per experiment

🔗 Traces

- End-to-end job execution
- Distributed system spans
- Bottleneck identification
- Multi-stage pipelines

📝 Logs

- Correlated with traces
- Structured & searchable
- Error aggregation
- Real-time streaming

Impact: MTTR reduced 40% | Pipeline bottlenecks identified | Unified view across HPC + cloud-native

Lessons from CNCF Contributions

13+ PRs across Kubernetes, KEDA, OpenTelemetry, TiKV, Metal³, and more

Start Small

One Workload at a Time

Don't GitOps everything at once. Start with dev/test environments and non-critical batch jobs. Build confidence, then expand.

Leverage Community

Don't Reinvent the Wheel

ArgoCD, Crossplane, KEDA are production-grade and actively maintained. Contributing back improves the tools for everyone.

Iterate Safely

GitOps Enables Experimentation

Everything in Git with version control means rollbacks are trivial. This accelerates innovation. Safe to experiment.

Real-World Impact

Measurable results from applying GitOps to HPC

⚡ **95%**

Faster Deploys

2-4 hours → 10 min

💰 **30%**

Cost Reduction

Scale-to-zero with KEDA

🔍 **40%**

Faster MTTR

OpenTelemetry observability

📋 **100%**

Audit Trail

Every change in Git history

🎯 **The Hidden Benefit**

Developer happiness — teams self-service, experiment safely, and focus on science instead of infrastructure.

Environment Promotion Path

Low Sensitivity

Key Takeaways

1. **GitOps works for HPC**
   ArgoCD
   Start with CD — declare desired state in Git

2. **Infrastructure as Code**
   Crossplane
   Self-service cluster provisioning via K8s CRDs

3. **Smart Autoscaling**
   KEDA
   Scale-to-zero for bursty workloads — 30% savings

4. **Observability is Critical**
   OpenTelemetry
   Unified metrics, traces, logs — 40% faster MTTR

5. **Community Accelerates**
   CNCF Ecosystem
   Don't reinvent — leverage and contribute back

HPC + Cloud-Native convergence is happening. These patterns help you modernize.

Getting Started & Resources

📚 Project Documentation

→ argoproj.github.io/argo-cd

→ crossplane.io/docs

→ keda.sh/docs

→ opentelemetry.io/docs

💬 Community Channels

→ CNCF Slack: #argo-cd, #keda

→ Kubernetes HPC-SIG

→ Crossplane Community Meetings

→ OpenTelemetry Contributors

Happy to discuss your HPC use cases — reach out!

Drift Remediation in Practice

ArgoCD Application — selfHeal reverts manual changes in < 60 s

```yaml
apiVersion: argoproj.io/v1alpha1
kind: Application
metadata:
  name: hpc-workload
  namespace: argocd
spec:
  destination:
    server: https://kubernetes.default.svc
    namespace: hpc-prod
  source:
    repoURL: git@github.com:org/hpc-manifests.git
    path: clusters/prod
  syncPolicy:
    automated:
      prune: true       # ← Deletes resources removed from Git
      selfHeal: true    # ← Reverts manual kubectl edits in <60s
```

✅ **selfHeal: true → 87% reduction in config drift**

🗑️ **prune: true → Removes orphaned resources automatically**

Shift-Left: Formal Verification in CI

GitHub Actions gate — catch drift before it reaches the cluster

```
# .github/workflows/verify.yml — runs on PR to main
- name: Formal Drift Verification
  run: |
    # ← Render manifests exactly as ArgoCD would
    helm template prod ./charts/hpc-app \
      -f values-prod.yaml > rendered.yaml

    # ← Verify no drift from declared state
    kubectl diff -f rendered.yaml --server-side

    # ← Policy constraint check (temporal safety)
    conftest verify rendered.yaml \
      --policy policies/ --fail-on-warn
```

Policy-as-Code: Zero-Trust Governance

Kyverno ClusterPolicy — Enforce mode blocks non-compliant workloads at admission

```yaml
apiVersion: kyverno.io/v1
kind: ClusterPolicy
metadata:
  name: zero-trust-deployment
spec:
  validationFailureAction: Enforce    # ← Block, don't just warn
  rules:
    - name: restrict-registries
      match:
        resources: { kinds: [Pod] }
      validate:
        message: "Images must come from approved ECR"
        pattern:
          spec:
            containers:
              # ← Only trusted private registry allowed
              - image: "709741256416.dkr.ecr.*.amazonaws.com/*"
```

❌ **DENIED**

```
image: docker.io/nginx:latest
image: ghcr.io/unknown/app:v1
```

✅ **ALLOWED**

```
image: 709741256416.dkr.ecr
       .us-east-2.amazonaws.com/app:v2.1
```

⏭ UP NEXT — SESSION 2

The Next Step: From Platform to Practitioner

→ We've seen how GitOps automates the platform — fleet management, infrastructure as code, event-driven scaling, zero-trust security

→ But how do researchers actually package their simulations and deploy onto this platform?

→ Session 2 answers that: containers, CI/CD, autoscaling, and hardware-level observability for scientific workloads

✅ **Session 1 (This Talk)**

GitOps for HPC
The Platform Layer

⏭ **Session 2 (Up Next)**

DevOps for Scientific Software
The Practitioner Layer

🕐 **Stay in your seats — Session 2 starts in 10 minutes!**

Thank You!

Questions & Discussion

Pavan Madduri

Senior Platform Engineer @ W.W.Grainger | CNCF Kubestronaut

🔗 LinkedIn: linkedin.com/in/pavanmadduri

📝 Blog: pavanmadduri.wordpress.com

💻 GitHub: github.com/pmady

#HPSFcon #GitOps #HPC #CloudNative

Scan for Slides & Research Papers

bit.ly/gitops-hpc-hpsf2026

Security Layers in GitOps HPC

Git RBAC

- 🔒 Branch protection rules
- 🔒 PR reviews required
- 🔒 CODEOWNERS enforcement
- 🔒 Signed commits

K8s RBAC

- 🔒 Namespace isolation
- 🔒 Service account limits
- 🔒 Pod Security Standards
- 🔒 Network policies

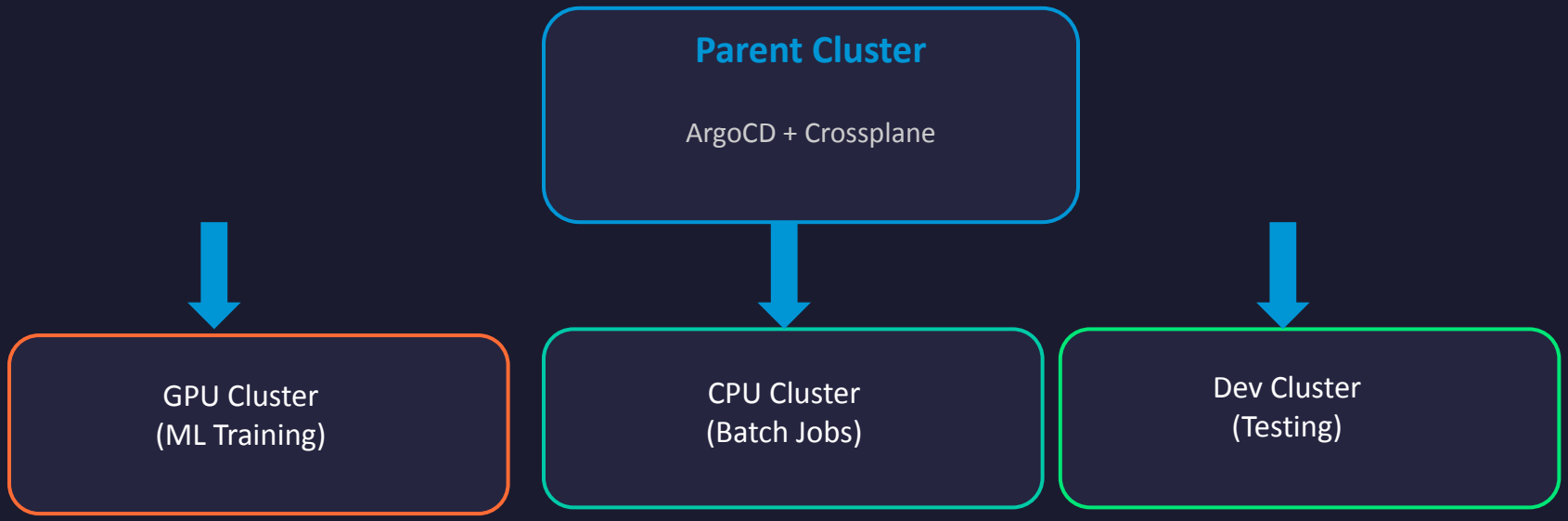
Secrets Mgmt

- 🔒 External Secrets Operator
- 🔒 Vault integration
- 🔒 Auto-rotation
- 🔒 Zero secrets in Git

Policy Engine

- 🔒 Kyverno policies
- 🔒 Image admission ctrl
- 🔒 Resource quotas
- 🔒 Compliance reporting

Hub-and-Spoke Multi-Cluster



Single control plane managing GPU, CPU, and dev clusters across regions

- ✓ Single ArgoCD manages all workload clusters
- ✓ Environment promotion: sandbox → dev → nonprod → prod
- ✓ Disaster recovery and multi-region patterns built-in
- ✓ Consistent policies enforced across all clusters