Managing Multi-Tenant Kubernetes Clusters for AEM and HCL Commerce: A Best Practices Study

Harish Govinda Gowda

Engineer,

Cardinal Health International India

Abstract

As enterprise organizations increasingly adopt containerized platforms, the need to efficiently manage multi-tenant Kubernetes clusters becomes paramount—especially when supporting complex applications like Adobe Experience Manager (AEM) and HCL Commerce. This article presents a comprehensive study of best practices for architecting and operating shared Kubernetes environments across multiple business units and application teams. It explores key design patterns for namespace isolation, resource governance, network segmentation, and CI/CD integration, with a focus on maintaining strong security and compliance boundaries. Additionally, it covers critical areas such as observability, autoscaling, secrets management, and disaster recovery, all tailored to the nuanced demands of AEM and HCL Commerce. Drawing from real-world operational lessons, the study emphasizes the importance of standardization, policy-driven automation, and platform engineering to ensure tenant safety, performance stability, and operational agility. The findings offer strategic guidance for platform teams looking to scale multi-tenant Kubernetes clusters without sacrificing control, efficiency, or innovation.

Keywords: Kubernetes, multi-tenancy, AEM, HCL Commerce, namespace isolation, DevOps, CI/CD.

1. Introduction

As digital commerce and content delivery systems evolve, enterprises are increasingly adopting Kubernetes to host large-scale, complex applications such as Adobe Experience Manager (AEM) and HCL Commerce. These platforms are resource-intensive, stateful, and often require high availability and strict uptime guarantees. Running them in isolated clusters has traditionally been the go-to strategy to ensure performance and security. However, this approach often leads to infrastructure sprawl, redundant operational overhead, and underutilized resources. To address these limitations, organizations are exploring multi-tenant Kubernetes clusters as a way to consolidate infrastructure while maintaining workload isolation and governance.

Multi-tenancy allows multiple teams, applications, or business units to share the same Kubernetes infrastructure without interfering with each other. For AEM and HCL Commerce, which have distinct performance profiles and operational requirements, this model introduces unique challenges. These include preventing "noisy neighbor" effects, managing access control across isolated namespaces, and tuning resources appropriately without sacrificing platform efficiency. Despite these complexities, when implemented with best practices, multi-tenancy significantly improves resource utilization, simplifies management, and supports faster time to market for digital services.

This article provides a practical, experience-based study of managing multi-tenant Kubernetes environments that host both AEM and HCL Commerce. Drawing from real-world enterprise deployments, it outlines architectural strategies, tooling decisions, and operational best practices that enable teams to achieve secure, performant, and scalable deployments. It explores how organizations use namespace-level isolation, GitOps pipelines, observability platforms, and Kubernetes-native policies to streamline tenant management while safeguarding critical production workloads.

The goal is to provide platform engineers, DevOps teams, and system architects with a set of actionable recommendations and a blueprint for safely co-hosting AEM and HCL Commerce on shared Kubernetes clusters. By the end of this article, readers will understand the key trade-offs, common pitfalls, and long-term advantages of building a multi-tenant Kubernetes platform that supports high-performance digital commerce and content delivery platforms at scale.

2. Understanding AEM and HCL Commerce Workloads

Successfully managing multi-tenant Kubernetes clusters for AEM and HCL Commerce starts with understanding the nature and operational characteristics of these two platforms. Although both are enterprise-grade systems built to handle large volumes of content, users, and transactions, their architecture and performance demands differ significantly.

Adobe Experience Manager (AEM) is primarily a content management system (CMS) that relies on a tiered architecture: author, publish, and dispatcher. The author tier is where content creators manage digital assets and editorial workflows, typically requiring high memory and persistent storage. The publish tier serves end-user traffic, demanding high throughput, caching, and efficient load balancing. Dispatchers—often implemented as Apache HTTPD modules—add an extra layer of caching and request routing. Because AEM uses Java and stores content in a Java Content Repository (JCR), the platform tends to have large memory footprints and long startup times, which can complicate container orchestration and scaling.

HCL Commerce, on the other hand, is a transactional e-commerce engine focused on shopping cart functionality, product catalogs, pricing, promotions, and order processing. It includes a variety of backend components such as search, transaction servers, caching layers, and databases. Unlike AEM, which has

consistent workload patterns, HCL Commerce experiences more volatility in traffic—especially during promotions or seasonal spikes—requiring more aggressive and responsive scaling strategies.

Both platforms have complex dependencies and tight integration requirements with third-party systems (e.g., identity providers, CDNs, databases, analytics tools), and they must remain operational during updates or failovers. This makes containerizing and orchestrating them in Kubernetes a non-trivial task. When co-hosted in a single cluster, the resource contention risks increase—especially if one platform experiences load surges that inadvertently affect the other.

Thus, running AEM and HCL Commerce in a shared, multi-tenant Kubernetes environment demands fine-tuned resource allocation, strict namespace isolation, and smart autoscaling strategies. Understanding these platforms at the workload level helps in shaping a resilient and efficient cluster design that supports each platform's unique behavior while maximizing infrastructure efficiency.

3. Multi-Tenancy Architecture Models

When deploying multiple enterprise-grade applications like AEM and HCL Commerce on the same Kubernetes infrastructure, choosing the right multi-tenancy model is essential. Multi-tenancy in Kubernetes can be implemented in various ways, each offering different trade-offs between isolation, operational complexity, and resource efficiency. The two primary approaches are namespace-level isolation within a single cluster and physical isolation using separate clusters. For large-scale environments, namespace-level multi-tenancy is often preferred due to better resource utilization and simplified operations, but it must be reinforced with strict governance mechanisms.

The namespace-per-tenant model is the most common approach to Kubernetes multi-tenancy. In this model, each tenant—typically representing an application, team, or business unit—is assigned its own Kubernetes namespace. Resources such as deployments, services, configmaps, and secrets are scoped to this namespace, providing a soft boundary for isolation. Combined with role-based access control (RBAC), this model ensures that users and automation systems can only interact with their allocated workloads, minimizing the risk of accidental changes across tenants.

Some organizations opt for cluster-per-tenant isolation, especially in highly regulated environments. This provides the highest degree of separation but increases management overhead due to the need to monitor, upgrade, and secure multiple clusters. It may also lead to poor resource utilization if workloads are unevenly distributed. For most enterprises, especially those using AEM and HCL Commerce, shared clusters with logical isolation provide a more sustainable solution when paired with good policies.

Additionally, hybrid models can be used—for example, separating workloads onto different node pools or control plane zones within a shared cluster. Kubernetes features like taints and tolerations, node affinity rules, and pod security policies (or Pod Security Admission in newer versions) are used to enforce additional isolation

between workloads. Integrating service mesh technologies like Istio or Linkerd can enhance network segmentation and observability across tenants, adding another layer of security and traffic control.

Ultimately, the choice of architecture depends on organizational maturity, compliance requirements, and operational scalability. For teams managing both AEM and HCL Commerce in a single cluster, the namespace-per-tenant model supported by automation and robust policy enforcement strikes an optimal balance between control and agility.

4. Namespace Isolation and Policy Enforcement

Namespace isolation is the cornerstone of secure and manageable multi-tenant Kubernetes deployments. In a shared cluster where platforms like AEM and HCL Commerce coexist, strict boundaries must be enforced at the namespace level to prevent unauthorized access, resource contention, and configuration drift. Kubernetes namespaces allow platform teams to logically separate workloads, but achieving effective isolation requires layered controls involving access management, policy enforcement, and resource governance.

Role-Based Access Control (RBAC) plays a central role in defining who can do what within each namespace. With RBAC, teams can ensure that developers, automation pipelines, and support engineers only have access to the resources relevant to their application. For example, AEM teams might have admin rights in the aem-prod namespace but be restricted to read-only access in hcl-prod. Combining RBAC with tools like Kubernetes service accounts and OpenID Connect (OIDC)-backed identity providers ensures that permissions are scoped tightly and traceably.

In addition to access control, policy enforcement engines such as OPA Gatekeeper and Kyverno are essential for maintaining consistent security and compliance across tenants. These tools allow cluster administrators to write and enforce custom policies as code—for instance, prohibiting privileged containers, enforcing naming conventions, or requiring labels for cost tracking. In a multi-tenant environment, policies can be applied globally or scoped to specific namespaces to accommodate different workload requirements while maintaining control.

To further isolate workloads, Kubernetes network policies can be applied to restrict traffic between pods and namespaces. This ensures that services in the HCL Commerce namespace, for example, cannot directly access AEM components unless explicitly allowed. Combined with a service mesh, fine-grained traffic control, encryption, and observability can be added, improving security posture without impacting development velocity.

Secrets management also requires careful handling. Sharing Kubernetes secrets across namespaces can pose a risk if not properly configured. Using tools like HashiCorp Vault or cloud-native secrets managers in conjunction with namespace-scoped injection strategies ensures that sensitive credentials are distributed securely and only to authorized workloads.

By tightly managing access, applying clear policy rules, and enforcing workload isolation through namespaces, platform teams can confidently support multiple high-demand applications in a shared cluster—without compromising security, stability, or compliance.

5. Resource Management and Scaling

Effective resource management is a critical component of multi-tenant Kubernetes operations, especially when hosting heavyweight platforms like Adobe Experience Manager (AEM) and HCL Commerce. Both applications are known for their demanding CPU and memory footprints, largely due to their Java-based architectures and persistent storage needs. Without well-defined resource boundaries and dynamic scaling strategies, workloads can compete for shared resources, leading to instability, degraded performance, or even outages.

Kubernetes provides native mechanisms to manage resource allocation through resource requests and limits, which define the minimum and maximum CPU and memory a container can use. In a multi-tenant setup, platform teams should enforce consistent use of these controls through policies, validating that all workloads declare their resource profiles. AEM's author and publish pods often require high memory with stable CPU, while HCL Commerce pods—particularly during traffic surges—may need more dynamic scaling. Properly tuned values help the scheduler place pods efficiently and prevent one tenant from consuming disproportionate resources.

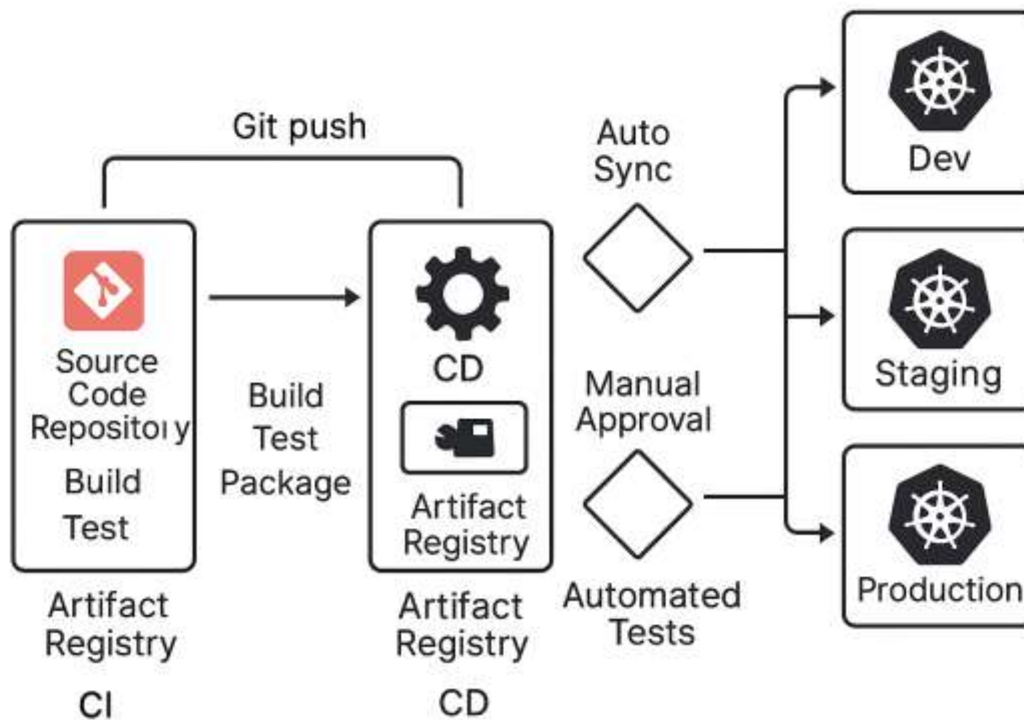
To adapt to changing load conditions, autoscaling is essential. Kubernetes offers Horizontal Pod Autoscaler (HPA) for scaling pods based on metrics such as CPU utilization, and Vertical Pod Autoscaler (VPA) for adjusting resource requests. Cluster Autoscaler can dynamically add or remove nodes to meet overall capacity needs. For multi-tenant clusters, these autoscaling components should be tenant-aware. For instance, AEM might require slower, state-aware scaling, while HCL Commerce could benefit from faster, CPU-based horizontal scaling. Using labels and taints/tolerations, node pools can be reserved for specific applications to isolate their resource consumption and ensure critical workloads are not evicted due to general demand.

In addition to scaling logic, resource quotas and limit ranges should be applied at the namespace level. This prevents any one tenant from consuming excess cluster resources, enforces fair usage, and provides predictability for capacity planning. Developers can be given soft boundaries with warnings or hard enforcement depending on SLA requirements.

Special attention must also be paid to garbage collection, image caching, and persistent storage performance, all of which can become bottlenecks in multi-tenant environments. AEM's reliance on large content repositories, for example, can strain I/O operations, so using high-performance storage classes and tuning pod lifecycles can make a significant difference.

By combining quotas, autoscaling, dedicated node pools, and continuous monitoring, organizations can confidently run diverse enterprise workloads in a shared Kubernetes cluster while minimizing risk and maximizing efficiency.

6. CI/CD Integration and Environment Promotion



A CI/CD pipeline

Integrating robust CI/CD pipelines into a multi-tenant Kubernetes environment is key to enabling fast, safe, and consistent deployments for applications like AEM and HCL Commerce. However, multi-tenancy introduces additional complexities: pipelines must be isolated per tenant, secrets must be scoped correctly, and promotion between environments (e.g., dev → staging → prod) must honor the unique resource and compliance requirements of each application.

Modern deployment strategies leverage GitOps principles, using tools like Argo CD or Flux to synchronize Git repositories with Kubernetes clusters. In a multi-tenant setup, each application can be assigned its own Git repository or folder structure within a shared repo. Git becomes the single source of truth for manifests, Helm charts, and Kustomize overlays, enabling automated, auditable, and version-controlled deployments. This structure also makes it easier to roll back changes, enforce approvals, and trace the origin of misconfigurations.

For AEM and HCL Commerce, which often require customized deployment logic, Helm charts are commonly used to encapsulate complex Kubernetes templates with reusable parameters. Multi-tenancy best practices include enforcing chart validation during pull requests, organizing Helm values per environment, and templating resources to avoid hardcoding environment-specific settings. This enables consistent deployment pipelines across dev, test, and production environments, even if each namespace or cluster differs slightly in configuration.

Secret management is another critical area. In multi-tenant clusters, it is unsafe to share cluster-wide secrets. Instead, secrets should be injected per namespace, with automation tools like Vault Agents, Sealed Secrets, or External Secrets Operator handling distribution and rotation. Pipelines should only have access to secrets relevant to their namespace and should use short-lived credentials wherever possible.

Promotion between environments must also follow a strict workflow. Many organizations implement promotion via Git merges, tagging, or automated pull requests, ensuring that every environment change is traceable. Additionally, progressive delivery techniques such as canary deployments or blue/green rollouts can be layered in to reduce risk during updates.

Finally, CI/CD workflows should integrate with observability and alerting systems, allowing teams to monitor deployment health in real time and roll back automatically on failure. This full lifecycle integration is what transforms multi-tenant Kubernetes clusters from risky shared environments into resilient, autonomous platforms.

7. Observability and Monitoring

In a multi-tenant Kubernetes environment hosting complex enterprise applications like AEM and HCL Commerce, observability is not just a convenience—it's a necessity. The ability to monitor, trace, and log tenant-specific activity is critical for troubleshooting, performance tuning, and ensuring compliance with SLAs. Because tenants share the same underlying infrastructure, poor observability can lead to blind spots that delay incident resolution, misattribute root causes, or expose sensitive data across namespaces.

A robust observability stack begins with metrics collection. Tools like Prometheus are widely used for scraping and storing metrics across clusters. In multi-tenant clusters, Prometheus must be configured to support label-based scoping, allowing teams to view metrics for their own namespaces while restricting access to others. This ensures that AEM engineers, for instance, can analyze CPU usage and request latencies specific to their services without seeing data from HCL Commerce workloads. To visualize these metrics, Grafana dashboards can be deployed per tenant with access restrictions, providing intuitive, real-time insights into application health, performance trends, and capacity planning needs.

For logging, solutions like Fluent Bit, Fluentd, or Logstash are commonly used to collect logs from pods and send them to backends such as Elasticsearch, Loki, or cloud-native logging platforms. Multi-tenancy introduces complexity in log aggregation, so log collectors should include namespace labels or custom metadata in each log entry. This allows centralized storage while preserving strict tenant-level access controls. Logs from AEM author nodes or HCL Commerce search pods can then be filtered, searched, and alerted on without cross-contamination between workloads.

Distributed tracing is equally important, especially for diagnosing latency issues in microservices architectures. Using tools like Jaeger or OpenTelemetry, developers can trace requests across services and identify

bottlenecks. Each tenant's applications should emit context-rich traces that include environment and namespace tags, allowing the platform team to isolate and debug issues specific to a tenant's namespace.

Another key aspect of observability is alerting and incident routing. Alerts should be tenant-scoped and routed to the appropriate on-call teams using tools like Alertmanager or cloud-based incident platforms. Defining alerting rules per namespace ensures that AEM teams are not overwhelmed with irrelevant HCL Commerce alerts and vice versa.

By structuring observability systems around namespace boundaries, metadata labels, and access controls, platform teams can achieve strong visibility while respecting multi-tenant boundaries. This promotes faster resolution of production issues, improves trust in the platform, and provides the foundation for advanced operations like autoscaling, anomaly detection, and automated remediation.

8. Security and Compliance Considerations

Security in a multi-tenant Kubernetes environment is fundamentally about trust boundaries: ensuring that each tenant is isolated, protected, and governed in accordance with enterprise and regulatory policies. Hosting both AEM and HCL Commerce in a shared cluster raises the stakes, as vulnerabilities or misconfigurations can have widespread consequences—potentially exposing data, allowing privilege escalation, or enabling lateral movement between namespaces.

The first line of defense is access control, enforced through Kubernetes RBAC (Role-Based Access Control) and integrated with identity providers like LDAP, Active Directory, or OIDC. Each team should have access only to their own namespace(s), with roles that restrict actions to the minimum necessary—for example, allowing HCL Commerce developers to manage deployments in hcl-staging, but not to view or edit resources in aem-prod. Service accounts used by automation tools should follow the same principle of least privilege.

At the network level, Kubernetes NetworkPolicies should be used to restrict pod-to-pod and namespace-to-namespace communication. For example, AEM's publish pods should not be able to access HCL Commerce's transaction services unless explicitly allowed. These policies can be extended with a service mesh like Istio to enable mTLS encryption between services, fine-grained traffic control, and advanced security policies.

Secrets management is another critical area. Secrets should be encrypted at rest using provider-native tools (e.g., AWS KMS, GCP KMS), and access should be scoped to the namespace. Tools like HashiCorp Vault, External Secrets Operator, or Sealed Secrets can manage secret rotation, audit access, and inject secrets dynamically into pods. Avoiding the use of cluster-wide secrets is key in a multi-tenant setup to prevent unintended leakage.

On the compliance side, logging and auditing are essential. Kubernetes audit logs must be enabled and stored securely, with a trail of all user and service account activity. Tools like Falco, KubeAudit, or cloud-native security platforms can provide real-time alerting for suspicious activity. Additionally, compliance frameworks

such as PCI-DSS, HIPAA, or ISO 27001 may require detailed controls, including workload isolation, encryption, vulnerability scanning, and access audits.

To enforce security policies consistently, use policy-as-code frameworks like OPA Gatekeeper or Kyverno. These tools can block deployments that violate security standards—for instance, containers running as root, missing resource limits, or using unapproved base images. Policies can be tested and version-controlled alongside application code to ensure consistency across environments.

Ultimately, strong security and compliance in a multi-tenant Kubernetes cluster come from layered defenses: identity and access management, network segmentation, secrets protection, policy enforcement, and continuous auditing. By proactively addressing these areas, platform teams can confidently scale their environments while meeting both operational and regulatory expectations.

9. Operational Lessons Learned

Operating multi-tenant Kubernetes clusters for enterprise-grade platforms like AEM and HCL Commerce uncovers a number of lessons that aren't always apparent during initial design. These lessons often stem from real-world incidents, scaling pressures, or evolving team dynamics. One of the most consistent realizations is the need to treat the cluster itself as a product—not just an environment. This mindset shift encourages platform teams to prioritize stability, usability, documentation, and regular feedback loops with tenant teams.

A key operational insight is the importance of clear boundaries and ownership. Without well-defined namespaces, RBAC roles, and resource quotas, teams often step on each other's toes, leading to deployment delays, configuration conflicts, or unintentional outages. For example, one tenant might mistakenly push resource-hungry changes that exhaust node capacity, causing unrelated services to evict or crash. Establishing strong guardrails and automation to prevent such cross-tenant interference is essential.

Another lesson revolves around standardization versus flexibility. While it's tempting to offer tenants total control over how they structure deployments, this quickly becomes unsustainable at scale. Instead, platform teams benefit from offering standardized templates—Helm charts, GitOps workflows, and namespace configurations—with just enough flexibility for customization. This balance accelerates onboarding, reduces errors, and streamlines support.

Observability maturity also emerges as a key differentiator between operational success and firefighting. Clusters with clear, tenant-scoped dashboards and logs experience significantly faster resolution times during incidents. However, observability must be carefully scoped so that one team's metrics or logs don't overwhelm another's view. Platform teams often learn to preconfigure dashboards and alerts per namespace, reducing manual configuration overhead and ensuring consistency.

Disaster recovery drills and failover testing further reinforce the need for automated, reproducible environments. Many teams discover during outages that their backup or redeployment processes are incomplete,

poorly documented, or slow. Maintaining automated CI/CD pipelines that can recreate environments from scratch—even in new clusters—is one of the most impactful lessons learned, particularly during region-wide cloud outages.

Finally, operational friction often arises around communication and support expectations. Shared environments require shared responsibility, which means clear SLOs, incident response playbooks, and escalation paths. Regular tenant reviews, documentation updates, and retrospectives help prevent repeating mistakes and foster a culture of continuous improvement.

These operational lessons are foundational to building a robust, maintainable multi-tenant Kubernetes platform. By learning from failure, automating repeatable tasks, and enforcing consistency with empathy, teams can scale both technology and trust across organizational boundaries.

10. Future Trends and Strategic Guidance

As organizations continue to adopt Kubernetes at scale, the demand for effective multi-tenancy solutions—especially for platforms like AEM and HCL Commerce—will only increase. Looking ahead, several trends and strategic shifts are shaping the future of multi-tenant Kubernetes management, providing opportunities for improvement, automation, and innovation.

One major trend is the move toward platform engineering and internal developer platforms (IDPs). These platforms abstract the complexity of Kubernetes and provide developers with self-service capabilities for deploying applications, provisioning namespaces, or scaling resources. Tools like Backstage, Port, and Kratix are gaining popularity as they help organizations standardize tenant onboarding and enforce best practices behind intuitive interfaces. For AEM and HCL Commerce teams, this means faster time-to-market and reduced cognitive load.

Another significant evolution is in the area of policy-driven automation. The increasing maturity of tools like OPA Gatekeeper, Kyverno, and even AI-powered controllers will enable clusters to enforce dynamic policies based on context, workload type, or usage trends. This allows for more intelligent scaling decisions, automated compliance enforcement, and proactive security posture management—all crucial in a multi-tenant setting where different workloads have different risks and requirements.

Cloud providers and Kubernetes distributions are also making strides in multi-cluster and fleet management, making it easier to manage multiple shared clusters as a unified control plane. Platforms like Google Anthos, Azure Arc, and Amazon EKS Anywhere provide centralized policy, visibility, and identity management across environments. For organizations running AEM in one region and HCL Commerce in another—or across dev/test/prod environments—this improves operational consistency and reduces manual toil.

In the future, runtime security and workload identity will play a more central role. As multi-tenant clusters become more sophisticated, teams must adopt zero-trust principles that authenticate and authorize every

interaction—between services, pipelines, and users. Integrating identity-aware proxies, service meshes, and workload identity with tenant scoping ensures that every communication is verified and logged.

Strategically, organizations must also rethink how they measure success in shared environments. Metrics like cost efficiency, deployment velocity, and tenant satisfaction are becoming more important than uptime alone. Governance models must evolve to balance freedom with safety, enabling teams to innovate without jeopardizing others.

In summary, the future of multi-tenant Kubernetes is guided by automation, platform abstraction, policy enforcement, and user-centric design. Teams that invest in scalable architecture, developer experience, and security now will be well-positioned to support increasingly diverse workloads—at scale, securely, and with agility.

11. Conclusion and Strategic Recommendations

Managing multi-tenant Kubernetes clusters for enterprise platforms such as Adobe Experience Manager (AEM) and HCL Commerce requires a deliberate and well-architected approach. These systems are not only resource-intensive but also mission-critical, meaning any operational oversight in a shared environment can have widespread consequences. However, with the right strategies in place, multi-tenancy becomes a powerful tool for cost efficiency, infrastructure consolidation, and consistent DevOps practices across teams.

This article has explored the full spectrum of best practices—starting with tenancy models, namespace isolation, and policy enforcement, all the way through CI/CD integration, observability, and long-term scaling considerations. A consistent theme across all areas is the value of standardization and automation. The more repeatable and enforceable your platform governance becomes, the more resilient your shared environment will be—even under the pressure of rapid releases or unpredictable user loads.

Strategically, organizations should prioritize building an internal developer platform (IDP) layer atop Kubernetes, giving application teams self-service access to resources while abstracting away complexity. Security and compliance must remain foundational, not bolted on—embedding enforcement directly into CI/CD pipelines, Helm templates, and policy controllers. Similarly, observability should be tenant-aware, offering scoped dashboards, logs, and alerts that align with application ownership.

From an operational standpoint, clear ownership models, disaster recovery drills, and postmortem culture help reduce chaos and foster shared accountability. No multi-tenant platform is static—teams evolve, requirements change, and workloads grow—so your platform must remain agile and modular. Investing in multi-cluster strategy, fleet management, and cross-region redundancy will prepare your organization for even greater scale and complexity.

In conclusion, managing AEM and HCL Commerce in a multi-tenant Kubernetes environment is entirely achievable with thoughtful architecture, strong governance, and a commitment to platform excellence.

Organizations that build resilient, developer-friendly platforms grounded in policy, observability, and automation will not only streamline operations but also enable innovation across teams—safely and at scale.

References

1. Eiermann, A., Renner, M., Großmann, M., & Krieger, U.R. (2017). On a Fog Computing Platform Built on ARM Architectures by Docker Container Technology. *International Conference on Innovations for Community Services*.
2. Beltre, A., Saha, P., & Govindaraju, M. (2019). KubeSphere: An Approach to Multi-Tenant Fair Scheduling for Kubernetes Clusters. *2019 IEEE Cloud Summit*, 14-20.
3. Zakai, I., Varner, M.E., & Gerber, R.B. (2017). Concerted transfer of multiple protons in acid-water clusters: [(HCl)(H2O)]2 and [(HF)(H2O)]4. *Physical chemistry chemical physics : PCCP, 19 31*, 20641-20646 .
4. Charmant, J.P., Haddow, M.F., Mistry, R.N., Norman, N.C., Orpen, A.G., & Pringle, P.G. (2008). A simple entry into nido-C2B10 clusters: HCl promoted cleavage of the C-C bond in ortho-carboranyl diphosphines. *Dalton transactions, 11*, 1409-11 .
5. Bresnahan, C.G., David, R., Milet, A., & Kumar, R. (2019). Ion Pairing in HCl-Water Clusters: From Electronic Structure Investigations to Multiconfigurational Force-Field Development. *The journal of physical chemistry. A*.
6. Uras-Aytemiz, N., Balcı, F.M., & Devlin, J.P. (2019). Can sulfur-containing molecules solvate/ionize HCl? Solid state solvation of HCl on/in methanethiol clusters/nanoparticles. *The Journal of chemical physics, 151 19*, 194309 .
7. Huneycutt, A.J., Stickland, R.J., Hellberg, F., & Saykally, R.J. (2002). Infrared cavity ringdown spectroscopy of acid – water clusters : HCl – H 2 O , DCl – D 2 O , and DCl – „ D 2 O ... 2.
8. Samanta, A.K., Wang, Y., Mancini, J.S., Bowman, J.M., & Reisler, H. (2016). Energetics and Predissociation Dynamics of Small Water, HCl, and Mixed HCl-Water Clusters. *Chemical reviews, 116 9*, 4913-36 .
9. Zischang, J., Skvortsov, D., Choi, M.Y., Mata, R.A., Suhm, M.A., & Vilesov, A.F. (2015). Helium nanodroplet study of the hydrogen-bonded OH vibrations in HCl-H_2O clusters. *The journal of physical chemistry. A, 119 11*, 2636-43 .
10. Lin, W., & Paesani, F. (2015). Infrared spectra of HCl(H2O)n clusters from semiempirical Born-Oppenheimer molecular dynamics simulations. *The journal of physical chemistry. A, 119 19*, 4450-6 .
11. Mancini, J.S., & Bowman, J.M. (2014). Effects of Zero-Point Delocalization on the Vibrational Frequencies of Mixed HCl and Water Clusters. *The journal of physical chemistry letters, 5 13*, 2247-53 .