portainer.io

How to build your own enterprise Kubernetes platform

From initial need, to operational readiness

Contents

Foreword: Executive Summary

Adopting Kubernetes at an enterprise scale using open-source CNCF tooling is a powerful decision, but it's never simple or quick. Organizations rarely start down this path purely out of curiosity or enthusiasm; more often, they're compelled into action by external vendor requirements or internal mandates to modernize their IT operations. Whichever trigger sets your journey in motion, it's essential to understand exactly what's ahead before you begin.

This eBook outlines the genuine, real-world journey most teams experience when implementing Kubernetes from scratch. From the moment you first realize you need Kubernetes, through foundational learning, detailed exploration of the CNCF ecosystem, intensive architectural decision-making, iterative implementation, stabilization, and finally scaling to multi-cluster management; this guide presents an honest narrative of the full lifecycle.

Each stage involves careful evaluation, detailed planning, constant iteration, frequent setbacks, and incremental successes. It's neither linear nor tidy, but rather a journey shaped by trial and error, learning, collaboration, and perseverance. You should expect your team to spend between 9 and 15 months moving from initial exploration to having a fully operational, production-grade Kubernetes platform. Of course, your mileage may vary, but this is what we have seen in the field.

Through this process, you'll inevitably discover that the complexity of running Kubernetes demands a dedicated Platform Engineering practice. The initial project transforms into an ongoing operational commitment, involving continuous upgrades, tool replacements, and persistent fine-tuning.

While easier, integrated Kubernetes platforms exist, many organizations still deliberately choose the path of building their own from open-source tools, valuing flexibility, control, and avoiding lock-in over immediate convenience.

This eBook is designed to guide, prepare, and support you on your Kubernetes journey, by providing clear expectations, realistic timelines, and honest insights. Embrace the complexity; the reward is an enterprise-ready Kubernetes platform and a team deeply enriched by the experience.

Your Kubernetes Journey Begins Here

CHAPTER 1

Identifying the business need

It usually starts innocently enough. You probably didn't wake up thinking, “today’s a great day to implement Kubernetes!” More likely, your phone buzzed with a message that one of your key software vendors just announced the next release of their product would only be deployable on Kubernetes. Suddenly the project you’d vaguely planned for next year has become urgently necessary right now.

Or maybe the push comes internally. Perhaps your CTO walked into the office one day after a board meeting, saying your infrastructure feels outdated, too costly, or just too slow to adapt to modern cloud-native workloads. Either way, your team now needs answers. What exactly is Kubernetes, how complex is it, what will it take, and is it realistic for your team? Management expects quick answers, but right now, your answers are tentative. You know enough to recognize Kubernetes as powerful but complicated. You’re cautious about promising too much, too fast.

Before making commitments, you want your team to do some research, experiment at home, and likely speak to a few people. You set aside some initial meetings to discuss possible impacts. Engineers voice cautious excitement but also realistic concerns about potential skill gaps, training needs, and infrastructure implications. Your security team immediately starts asking questions about compliance and risks. Budget managers want to understand costs, both immediate and ongoing.

You start drafting emails to peers in other companies who've adopted Kubernetes, asking them about their experiences. You talk to your software vendor to clarify exactly what "Kubernetes-only" means (hoping perhaps they'll allow an extension or alternatives) but their response is clear: the direction is set.

Quickly, it becomes apparent that Kubernetes adoption isn't something you'll simply delegate and forget; it will become an organizational priority. You begin outlining high-level project objectives, scheduling initial internal presentations, and reassuring your team that while it's complex, it's also an exciting and strategic opportunity. Quietly, you start mentally preparing for the journey ahead.

Kubernetes adoption isn't something you'll simply delegate and forget; it will become an organizational priority.

CHAPTER 2

Initial exploration and foundational learning

1–2 months

This initial phase feels both exciting and intimidating. Your engineers dive into Kubernetes for the first time, realizing quickly that containers are just the beginning. A small lab cluster appears on spare hardware or virtual machines, usually set up with tools like kubeadm or Minikube. Early experiments seem promising but often frustrating: simple tasks like deploying a web app take hours initially, as your team learns new concepts such as pods, deployments, services, and ingress controllers.

During this foundational period, engineers enrol in introductory Kubernetes courses, often pursuing CKAD or CKA certifications, and spend evenings diving through dense Kubernetes documentation and blogs. It's not uncommon at this stage for someone to suggest the team try Kelsey Hightower's legendary "Kubernetes the Hard Way." This meticulous, step-by-step guide (while deliberately complex and challenging) has become a rite of passage for Kubernetes enthusiasts. By manually bootstrapping clusters without automation, your team gains a fundamental, deeply instructive understanding of Kubernetes internals, invaluable for future troubleshooting and deeper architectural decisions.

You attend introductory webinars, join CNCF community Slack channels, and subscribe to podcasts about Kubernetes best practices. Some team members attend local meetups or online workshops, eager to hear directly from peers who've navigated similar journeys. Throughout this exploratory phase, discussions naturally broaden, as the team starts to see Kubernetes adoption not just as a technical project but as an opportunity to redefine infrastructure operations altogether.

During this exploratory phase, discussions naturally expand. The team, inspired by their research, recommends supplementing Kubernetes adoption with Infrastructure as Code (IaC) and GitOps practices, seeing them as "must-haves" for modern operations. Suddenly, the project's scope expands beyond just Kubernetes, becoming a broader modernization initiative.

By the end of this foundational period, the team has a basic grasp of Kubernetes concepts, a common vocabulary, and a realistic understanding of the steep learning curve ahead through peer discussions and recommendations. The excitement remains, tempered by the realization that this initial learning stage barely scratches the surface. Engineers now begin presenting short summaries of their discoveries to peers, outlining both promising opportunities and areas where more expertise will be essential.

You schedule team check-ins regularly, making sure everyone remains aligned and motivated. You start drafting initial reports back to management, carefully communicating progress, while managing expectations about timelines and complexities. Privately, you're already preparing for the inevitable challenges ahead, hoping your foundational work positions the team well.

By the end of this foundational period, the team has a basic grasp of Kubernetes concepts.

CHAPTER 3

Evaluating the CNCF and open-source ecosystem

2-3 months

Just when confidence starts to build, your engineers hit the infamous "CNCF landscape," a complex map covered in hundreds of open-source projects. You quickly realize Kubernetes itself is only the orchestration core. It doesn't directly provide storage, network policies, security, backups, monitoring, or centralized logging. Now, your team must explore, evaluate, and decide on a dizzying array of solutions.

Engineers run proofs-of-concept (PoCs) for storage solutions like Rook/Ceph or Longhorn, experimenting to see which integrates easiest with your network. The networking team wrestles with Calico and Cilium, while infrastructure teams test OS distributions optimized for Kubernetes; Talos, Flatcar, Bottlerocket, or Ubuntu. The complexity grows exponentially when you start trialling authentication solutions like Dex or Keycloak, struggling to integrate them seamlessly with corporate Active Directory or another OIDC provider.

This stage is marked by extensive trial and error. You inevitably attend industry conferences like KubeCon or regional Kubernetes Days, absorbing as much wisdom as possible from others who've experienced similar challenges. Engineers spend significant time on Stack Overflow, GitHub, CNCF Slack, and community forums, hoping someone has encountered the same obscure networking or storage errors they face at 1 AM. Documentation grows increasingly detailed, recording lessons learned and justifications for tool choices.

You organize internal presentations and workshops, allowing engineers to share what they've learned about each evaluated tool. Debates become lively, even passionate, as strong opinions form around each toolset's pros and cons. You quickly realize tool selection isn't purely technical, factors like community support, maintenance overhead, and team familiarity also heavily influence your decisions.

In this stage, the team also grapples with the realities of combining multiple open-source solutions. Questions about long-term maintenance, potential security vulnerabilities, upgrade paths, and integration complexities constantly surface. Gradually, confidence builds as solutions clarify, but the complexity still feels daunting.

Typical evaluations include:

- **Infrastructure automation**
  (Terraform, Cluster API, Crossplane, Sidero Omni)
- **Operating System**
  (Talos, Flatcar, Bottlerocket, Alpine, Ubuntu)
- **Kubernetes Distribution**
  (RKE, K3s, EKS/AKS/GKE, K0s, OKE, Vanilla)
- **Networking tools**
  (Calico, Cilium, Istio, MetalLB, Kube-VIP)
- **Storage**
  (Rook/Ceph, Longhorn, NFS, iSCSI, CloudProvider CSI)
- **Authentication**
  (Dex, Keycloak, OIDC)
- **Observability**
  (Prometheus, Mimir, OpenTelemetary, Grafana, VictoriaMetrics, Loki, ELK, Jaeger)
- **GitOps automation**
  (Flux, Argo CD, Helm, Kustomize)
- **Backup and compliance tools**
  (Velero, Kubescape, kube-bench)

CHAPTER 4

Architectural design and tooling selection

1-2 months

After months of PoCs and evaluations, it's finally decision time. By now, your team is deeply familiar with dozens of tools, and everyone has a clear favourite. Engineers have spent countless hours wrestling with trial installations, writing internal blog posts, and advocating strongly for their preferred solutions. This stage often feels intense, filled with detailed meetings, vigorous debates, and sometimes heated arguments as team members passionately defend their choices.

To keep the selection process structured, you schedule weekly "architecture workshops," each dedicated to a specific topic, networking, storage, security, observability, and GitOps. During these sessions, engineers present detailed evaluation summaries, complete with pros, cons, integration complexities, and potential long-term implications. Sometimes these sessions feel more like courtroom battles than collaborative discussions, but the intense dialogue is vital. You need these passionate conversations to ensure the best possible choices emerge.

Inevitably, the complexity and detail cause decision fatigue. There comes a point where the team agrees that all evaluated tools are reasonably capable and viable, and further analysis might yield diminishing returns. At this stage, a balance is struck between technical ideals and practical realities, such as community adoption, available support channels, and the team's ability to manage tools long-term.

Throughout this process, architects meticulously document every decision made, capturing detailed reasoning and justification. Diagrams become increasingly comprehensive, illustrating exactly how each selected tool integrates into the larger Kubernetes platform. Security and compliance teams closely scrutinize each choice, demanding clear rationales and evidence for why certain tools were chosen over others. Every decision must align clearly with your organization's broader governance frameworks.

By the end of this architectural selection phase, you have a clear, agreed-upon blueprint. Comprehensive implementation plans are outlined, with clear timelines, responsibilities, and documented reasoning behind each choice. As this phase concludes, your team experiences a mix of relief, pride, and nervous anticipation. You're now confident in your choices, but aware that implementation will inevitably uncover complexities your plans cannot fully predict.

By the end of this architectural selection phase, you have a clear, agreed-upon blueprint.

CHAPTER 5

Implementation and iterative configuration

4-6 months

Implementation rarely unfolds as smoothly as architectural diagrams and planning documents suggest. Almost immediately, your team encounters issues never observed during previous PoCs. The networking plugins that seemed ideal suddenly conflict with storage provisioning mechanisms. RBAC permissions, carefully crafted on paper, inadvertently restrict access to essential administrative tasks, causing days of debugging and frustration. Authentication integrations (such as Dex or Keycloak with OIDC) repeatedly fail during initial tests, demanding extensive troubleshooting and endless configuration adjustments.

What follows is an intense cycle of iteration, testing, documentation, and reconfiguration. Engineers frequently revisit community resources, re-watching conference talks from KubeCon, scouring GitHub issue trackers, or seeking advice from CNCF community Slack channels. Sometimes your engineers even directly reach out to tool maintainers, desperate for advice on why integrations fail in subtle yet critical ways. Each resolution seems to lead inevitably to new challenges, creating a complex puzzle where solutions trigger more complexity.

GitOps pipelines, initially set up with Flux or Argo CD, require multiple rebuilds and iterations before finally working smoothly. Helm charts and Kustomize manifests, meticulously crafted and versioned, seem to evolve daily. Monitoring and observability stacks (Prometheus, Grafana, Loki, Jaeger) demand constant tuning, as your team iteratively adjusts alerts and dashboards to filter out noise while still surfacing actionable insights.

Internally, you establish weekly "integration stand-ups," ensuring team alignment, managing morale, and addressing blockers quickly. Engineers become familiar with late-night debugging sessions and weekends spent refining configurations. Yet despite the exhaustion, there's a genuine satisfaction in seeing the platform slowly stabilize.

Throughout this phase, documentation remains crucially important. Every iteration, failure, and eventual success is carefully recorded, forming detailed runbooks and operational guides. By the end of these intense months, you have a Kubernetes platform that feels robust enough (though not perfect) to cautiously welcome initial production workloads. Your team feels simultaneously drained and proud, aware that they've significantly deepened their collective expertise through this intense implementation journey.

By the end of these intense months, you have a Kubernetes platform that feels robust enough (though not perfect) to cautiously welcome initial production workloads.

CHAPTER 6

Stabilization, documentation, and initial production rollout

1-2 months

After months of relentless troubleshooting, configuration tweaks, and iterative improvements, your Kubernetes platform finally achieves enough stability to cautiously deploy the first critical workloads, likely including the vendor-mandated application that triggered this journey. This phase feels genuinely rewarding, though it also comes with new forms of pressure. Production deployments rarely behave exactly like staging or test environments; subtle differences in workload patterns, security rules, or storage configurations surface immediately.

As the first applications go live, your engineers remain hyper-vigilant, ready to rapidly respond to any issue. Predictably, minor yet urgent problems emerge, storage provisioning doesn't behave consistently, certain network policies prove overly restrictive, or unexpected permission errors arise. Yet your team is ready, armed with extensive documentation built over previous months. They quickly pinpoint root causes, apply fixes, and meticulously document each incident to prevent recurrences.

During this stabilization phase, operational documentation evolves significantly. Engineers spend considerable time writing comprehensive runbooks, troubleshooting guides, and onboarding materials, ensuring that operational knowledge isn't siloed or limited to a few specialists. Internal training sessions and workshops ramp up, helping broader teams (both developers and operations staff) comfortably adopt Kubernetes workflows.

Feedback loops tighten as initial users and developers provide valuable insights. You begin refining observability systems, improving dashboards, tweaking alert thresholds, and optimizing logging solutions like Loki or ELK stacks based on real-world data. Backup and disaster recovery processes (via Velero) are tested extensively, ensuring confidence in recovery capabilities should disaster strike.

By the end of this stage, confidence significantly improves. Your engineers, having successfully handled real-world production issues, feel genuinely accomplished. Management gains clearer visibility into the platform's readiness, becoming increasingly confident in expanding Kubernetes deployments further across your organization.

However, despite the best intentions and repeated emphasis throughout this journey, the reality is that the documentation you've meticulously planned and hoped to build is likely incomplete, scattered, or only half-written. This is a common, if rarely admitted, truth about engineering: everyone recognizes documentation's critical value, but amidst the intense pressures of implementation, debugging, and daily firefighting, maintaining thorough documentation consistently proves far more challenging than initially imagined. Acknowledging this openly at this stage sets the right expectations and underscores the importance of continuously investing time and resources to close the inevitable documentation gaps as soon as practical.

By the end of this stage, confidence significantly improves.

CHAPTER 7

Multi-cluster and centralized management

1–2 months, optional but common

Just as your team finally achieves solid operational stability with your first Kubernetes cluster, a new challenge inevitably arises: managing multiple Kubernetes clusters. Your journey so far focused on building a single robust Kubernetes platform; now the focus shifts to scale, governance, and centralized operations.

Teams quickly realize managing multiple clusters involves complexities around policy enforcement, consistency, upgrades, and governance. You re-enter research and evaluation mode, though with far greater expertise. Conferences like KubeCon become valuable again, this time, your questions center specifically around multi-cluster solutions like Rancher, Open Cluster Management (OCM), Karmada, or Cluster API.

Internal PoCs resume, this time quickly and efficiently, leveraging accumulated knowledge from previous experiences. You evaluate centralized policy enforcement, automated cluster lifecycle management, and consistent configuration enforcement across clusters. Integration complexity resurfaces but feels manageable, drawing on the confidence and resilience gained from previous phases.

Engineers once again present their findings, carefully documenting pros, cons, and recommended solutions. The selection process feels more straightforward due to prior experiences, yet discussions still demand careful consideration of integration complexity, long-term maintainability, and compatibility with existing systems.

Implementation is quicker, yet still involves familiar iterative cycles; tests, troubleshooting, adjustments, and refinement. Documentation expands further, detailing multi-cluster governance strategies, configuration management approaches, and centralized operational workflows. Training sessions expand to include managing and operating multiple clusters, emphasizing consistency and governance practices.

By the end of this stage, your organization possesses a mature Kubernetes platform strategy, capable of confidently managing multiple clusters through a centralized framework. This expanded operational capability positions your team strongly for future growth and innovation, with robust governance structures underpinning your organization's Kubernetes deployments.

By the end of this stage, your organization possesses a mature Kubernetes platform strategy, capable of confidently managing multiple clusters through a centralized framework.

CHAPTER 8

The ongoing reality of platform operations

9-15 months

Having successfully navigated the initial adoption journey (identifying the need, exploring, evaluating, architecting, implementing, stabilizing, and even scaling to multiple clusters) your team now transitions from initial deployment into long-term, day-to-day operational reality. At first, you hope this phase will feel stable, predictable, and relatively straightforward compared to the intense prior months. But you quickly learn that operating Kubernetes platforms, especially at scale, introduces an entirely new set of ongoing responsibilities and complexities.

Routine platform operations quickly become prominent. Regular tasks include Kubernetes version upgrades, security patching, and tool updates for your chosen CNCF stack. Engineers learn that even minor version updates occasionally bring unexpected challenges, as subtle changes in Kubernetes APIs, container runtimes, or third-party tooling trigger incompatibilities and regressions. Your team establishes clearly defined maintenance windows, carefully balancing stability with necessary progress, performing cautious and methodical upgrades to minimize disruption.

Simultaneously, previously selected CNCF tools continue evolving rapidly, demanding continuous attention. Projects that initially seemed ideal might lose maintainers, community support, or simply become less suitable over time. The team regularly re-evaluates critical components, considering replacements or migrations when tools become deprecated or no longer match organizational needs. Lifecycling decisions demand careful analysis, often revisiting earlier PoC processes, though more quickly and efficiently given your prior experiences.

Troubleshooting remains a constant part of platform operations. Even with robust monitoring (Prometheus, Grafana), comprehensive logging (Loki, ELK), and distributed tracing (Jaeger, OpenTelemetry), subtle performance issues or unexpected outages inevitably occur. Engineers grow increasingly skilled at rapidly diagnosing problems, leveraging their extensive documentation and internal runbooks. Incident retrospectives become routine, each incident feeding back into improved operational processes and documentation.

Over time, the workload and complexity of managing Kubernetes clusters becomes clearly unsustainable for engineers with broader roles or responsibilities. You realize managing this sophisticated platform demands a dedicated team with specialized skills and a defined focus. At this point, your organization begins establishing a dedicated Platform Engineering practice, shifting responsibility away from generalist infrastructure teams towards specialists focused exclusively on Kubernetes operations and continuous improvement.

Continued...

Platform Engineers become responsible not only for ongoing operational tasks but also for continuously improving the developer experience. They implement self-service capabilities, refine deployment automation, enhance observability tooling, and maintain clear communication with development teams. Your organization quickly appreciates how Platform Engineering contributes directly to developer productivity and operational resilience, recognizing it as a strategic function critical to modern software delivery.

Finally, the shift towards dedicated platform engineering marks a new maturity in your Kubernetes journey. Your platform now feels genuinely enterprise-grade, reliable, secure, scalable, and continuously evolving. Your team, deeply enriched by experiences and accumulated expertise, stands ready to manage the complexity that Kubernetes inevitably brings, viewing it no longer as a daunting challenge but as a powerful, enabling platform for innovation.

Closing Statement

This is the common journey most organizations experience when electing to adopt Kubernetes. It's complex, demanding, and occasionally exhausting. While integrated, pre-built Kubernetes platforms exist to significantly simplify adoption, many organizations deliberately choose the open-source path described here. They value flexibility, control, and avoiding vendor lock-in, consciously embracing the complexity as a worthwhile trade-off.

Your Kubernetes journey, challenging as it may be, positions your organization strongly for innovation and sustained growth.

Welcome to your Kubernetes future.