NINE · CLOUD NAVIGATORS

Getting Started with Kubernetes

A practical checklist for engineering teams

Kubernetes is the industry-standard platform for running containerised workloads in production. But getting there takes deliberate preparation. This checklist walks through the key questions every engineering team should answer before – and while – deploying their application on Kubernetes. Work through these in order; each builds on the previous.

1 Have you automated your container builds and pushes?

Kubernetes fetches container images from a registry, so a reliable, automated pipeline is essential. If you haven't yet automated your build and push process, start here: commit to a Git repository, trigger a build, produce an OCI-compliant image, and push it to a registry (private or public).

Working backwards from the image push: consider automated testing, code review gates, and any other workflow steps that need to be in place. A solid CI pipeline is the foundation for everything that follows.

---

2 Are you scanning your container images for vulnerabilities?

A small image is a more secure image, but size alone is not enough. Container images can carry known vulnerabilities (CVEs) in base layers or dependencies. Integrate image scanning into your CI pipeline using tools such as **Trivy** or **Grype**. Many container registries (including our private registry) offer built-in scanning.

Aim to fail the pipeline on critical or high-severity findings, and update base images regularly. Reducing the number of layers in your images also reduces the attack surface.

---

3 How lightweight is your container?

Your container image will be pulled by various systems throughout your pipeline: testing, integration deployments, the CI/CD system, and more. To optimise startup time and bandwidth costs, remove everything you don't need: use multi-stage builds, use minimal base images (such as distroless or Alpine), and avoid bundling build tools into your production image.

Leaner images also mean fewer packages with potential vulnerabilities: security and efficiency go hand in hand.

---

4 Have you written your Kubernetes configuration?

If you haven't yet written your Kubernetes manifests, consider the available templating options. **Helm charts** are the de-facto standard for packaging Kubernetes applications, as they simplify the deployment of complex applications and are widely used across the ecosystem. Familiarise yourself with Helm even if you start with plain YAML manifests.

Kubernetes uses a declarative configuration model. There are several tools to help generate and manage configuration at scale: **Helm**, **Kustomize** (built into kubectl), and more complex approaches like **cdk8s** or **Pulumi** for teams preferring to write infrastructure in code. Whatever you choose, treat your Kubernetes configuration as a first-class part of your application: version it, review it, and test it.

---

5 Do you know your application's performance profile?

To get the most out of Kubernetes you need to understand how your application performs under load: at minimum, the CPU and RAM profile. This lets you properly configure resource requests and limits, which Kubernetes uses for scheduling and self-healing.

Without proper resource configuration, your application might not restart when there's a problem, or enter `CrashLoopBackOff` if Kubernetes sees normal behaviour as exceeding resource limits. It is highly unlikely you'll get this right on the first try: monitor actual usage and iterate. Kubernetes v1.27+ supports in-place vertical scaling without pod restarts (currently in beta), making it easier to adjust resources over time.

---

6 Have you considered how your application scales?

One of the most powerful aspects of Kubernetes is horizontal scaling. Once you know the performance profile of your containers, configure a **HorizontalPodAutoscaler (HPA)** to scale your application based on current traffic or custom metrics. For clusters with variable workloads, **node autoscaling** ensures that the underlying infrastructure scales accordingly, adding and removing nodes without manual intervention.

Gone are the days of fearing high-traffic peaks. Let the cluster work for you.

---

7 Have you considered how your application responds to restarts?

Kubernetes will start, stop, and restart your containers regularly: that is a feature, not a bug. It will restart your application if it detects a failure, move it to a different node during maintenance, or reschedule it during scale-up/down operations. Make sure your application handles restarts gracefully: avoid running database schema migrations on every startup, implement proper liveness and readiness probes, and ensure fast start times so the scheduler can make the most of Kubernetes' dynamic capabilities.

If your application uses sidecars (e.g. service mesh proxies, log shippers), note that **native sidecar containers** reached General Availability in Kubernetes v1.32, and they now have proper lifecycle management and shut down correctly when the main container stops.

---

8 Have you considered the availability of your application?

Physical and virtual machines fail, and there is no way around it. But you can configure your workloads to eliminate single points of failure. Running multiple replicas spread across different nodes (using **Pod Topology Spread Constraints** or Pod Anti-Affinity) significantly improves availability. There is little point in running 10 replicas of a container on a single node if you have a multi-node cluster.

For mission-critical workloads, consider running across multiple availability zones, a feature offered by managed Kubernetes providers on multi-zone infrastructure.

---

9 Have you considered how your application responds to unavailable services?

Unavailability is always a possibility, regardless of the infrastructure, service provider, or architecture you choose, and it is especially important in cloud environments. Design your application to be resilient in such scenarios: implement retry logic with exponential backoff, circuit breakers, and graceful degradation. Ensuring your application remains accessible and does not lose data during a dependency outage should be a primary concern during the design phase.

---

10 Have you set up GitOps for your deployments?

A mature Kubernetes deployment workflow goes beyond CI/CD pipelines that push changes directly to clusters. **GitOps** tools such as **ArgoCD** or **Flux** treat Git as the single source of truth for cluster state. Changes are made via pull requests, automatically reconciled against the live cluster, and any drift from the desired state is detected and corrected. This gives you a fully auditable, version-controlled deployment history and simplifies rollbacks to any previous state.

GitOps is not optional for production: it is the pattern that makes continuous deployment safe and predictable.

---

11 Have you considered how to run the services your application depends on?

There is a lot to think about when running your application in Kubernetes. Regardless of whether you are new to Kubernetes or experienced, it may be a good idea to partner with an experienced managed services provider who can offer auxiliary services at a guaranteed level of quality. If you need direct contact with certified engineers, managed observability, GitOps setup, solution architecture support, or automated cluster upgrades, a provider like us – with CKA-certified engineers and years of experience running Kubernetes in production – provides significant value and lets you focus on shipping your product.

Nine Internet Solutions AG · Badenerstrasse 47 · 8004 Zurich · Switzerland
info@nine.ch · +41 44 637 40 00 · nine.ch