

# 16 steps for building production-ready Kubernetes clusters

## Overview

Kubernetes is a powerful tool for building highly scalable systems. As a result, many companies have begun, or are planning, to use it to orchestrate production services. However, like most powerful technologies, Kubernetes is complex. How do you know if you have configured your setup correctly and that it is safe to open the network to your services? The following 16 steps can help you prepare your containers and Kubernetes clusters for production traffic.

## Build the foundation

### 1. Use minimal base images

**What:** Containers are application stacks built into a system image. Everything from your business logic to the kernel gets packed inside. Minimal images strip out as much of the operating system (OS) as possible and force you to explicitly add back any components you need.

**Why:** By including only the software you intend to use in your container, you gain performance and security benefits. You have fewer bytes on disk, less network traffic for images being copied, and fewer tools for potential attackers to access.

**How:** [Red Hat® Universal Base Image](#), a component of Red Hat Enterprise Linux® that helps build containers, is a popular choice because it has enterprise support from Red Hat, and it can be used and supported on other platforms.

### 2. Use a registry that offers the best uptime

**What:** Registries are repositories for images, making those images available for download and launch. When you specify your deployment configuration, you will need to specify where to get the image with a path `<registry>/<remote name>:<tag>` :

```
apiVersion: v1
kind: Deployment ...
spec: ...
  containers
  - name: app
    image: docker.io/app-image:version1
```

**Why:** Your cluster needs images to run.

**How:** Red Hat OpenShift® provides a [built-in](#) container image registry that runs as a standard workload on the cluster. Most cloud providers also offer private image registry services: Google offers the Google Container Registry, Amazon Web Services (AWS) provides Amazon Elastic Container Registry (ECR), and Microsoft has the Azure Container Registry.

Do your homework, and choose a private registry that offers the best uptime. Because your cluster will rely on your registry to launch newer versions of your software, any downtime will prevent updates to running services.

### 3. Use ImagePullSecrets to authenticate your registry

**What:** ImagePullSecrets are Kubernetes objects that let your cluster authenticate with your registry, so the registry can be selective about who is able to download your images.

**Why:** If your registry is exposed enough for your cluster to pull images from it, then it is exposed enough to need authentication.

**How:** The [Kubernetes website](#) has a good walkthrough on configuring ImagePullSecrets, using Docker as a sample registry.

## Organizing the cluster

Building and running applications based on microservices architectures can become complex quickly. Much of the benefit of using microservices comes from enforcing separation of duties at a service level, effectively creating abstractions for the various components of your backend. Some good examples include running a database separate from business logic, running separate development and production versions of software, or separating out horizontally scalable processes.

A drawback of having different services perform different duties is that they cannot be treated as equals. Kubernetes gives you many tools to deal with this challenge.

### 4. Isolate environments by using namespaces

**What:** Namespaces are the most basic and powerful grouping mechanism in Kubernetes. They work almost like virtual clusters. Most objects in Kubernetes are, by default, limited to affecting a single namespace at a time.

**Why:** You will need to use namespaces as most objects are namespace scoped. Namespaces provide strong isolation and are perfect for isolating environments with different purposes. For example, production environments and those used strictly for testing. They can also separate different service stacks that support a single application, like keeping your security solution's workloads separate from your own applications. A good rule to follow is to divide namespaces by resource allocation. If two sets of microservices will require different resource pools, place them in separate namespaces.

**How:** Namespaces are part of the metadata of most object types:

```
apiVersion: v1
kind: Deployment
metadata:
  name: example-pod
```

```
namespace: app-pod
```

```
...
```

Note that you should always create your own namespaces instead of relying on the `default` namespace. Kubernetes' defaults typically optimize for development speed, and this approach often means forgoing even the most basic security measures.

## 5. Organize your clusters with labels

**What:** Labels are the most basic and extensible way to organize your cluster. They allow you to create arbitrary key:value pairs that separate your Kubernetes objects. For instance, you might create a label key that separates services that handle sensitive information from those that do not.

**Why:** As mentioned, Kubernetes uses labels for organization, but, more specifically, they are used for selection. As a result, when you want to give a Kubernetes object a reference to a group of objects in some namespace, like telling a network policy which services are allowed to communicate with each other, you use their labels. Because they represent such an open-ended type of organization, do your best to keep things simple, and only create labels where you require the power of selection.

**How:** Labels are a simple spec field you can add to your YAML files:

```
apiVersion: v1
kind: Deployment
metadata:
  name: example-pod
  ...
  matchLabels:
    userexposed: true
    storespii: true
```

## 6. Use annotations to track important system changes

**What:** Annotations are arbitrary key-value metadata you can attach to your pods, much like labels. However, Kubernetes does not read or handle annotations, so the rules around what you can and cannot annotate a pod with are fairly liberal, and they cannot be used for selection.

**Why:** They help you track certain important features of your containerized applications, like version numbers or dates and times of first bring up. Annotations, in the context of Kubernetes alone, are a fairly powerless construct, but they can be an asset to your developers and operations teams when used to track important system changes.

**How:** Annotations are a spec field similar to labels:

```
apiVersion: v1
kind: Pod
metadata:
  name: example-pod
```

```
...  
annotations:  
  version: four  
  launchdate: tuesday
```

## Providing security for your cluster

Now that you have a cluster set up and organized the way you want, your next step is incorporating security. In this section, we detail some important steps you should take to protect your cluster against security threats.

### 7. Implement access control using Kubernetes role-based access control (RBAC)

**What:** RBAC allows you to control who can view or modify different aspects of your cluster.

**Why:** If you want to follow the principle of least privilege, then you need to have RBAC set up to limit what your cluster users and what your deployments are able to do.

**How:** If you are setting up your own cluster (i.e., not using a managed Kubernetes service), make sure you are using “`--authorization-mode=Node,RBAC`” to launch your Kubernetes application programming interface (API) server. If you are using a managed Kubernetes instance, you can check that it is set up to use RBAC by querying the command used to start the Kubernetes API server. The only generic way to check is to look for “`--authorization-mode...`” in the output of `kubectl cluster-info dump`.

Once RBAC is turned on, you need to change the default permissions to suit your needs. The Kubernetes project site provides documentation on how to set up [RBAC Roles and RoleBindings](#).

### 8. Prevent risky behavior and configurations using Open Policy Agent (OPA) Gatekeeper

**What:** Gatekeeper provides a Kubernetes admission controller built around the OPA engine to integrate OPA and the Kubernetes API service. OPA offers an open source service that can evaluate inputs against user-defined policies and mark the input as passing or failing, making OPA very useful for Kubernetes cluster security compliance, as well as for practical resource configuration management. The Gatekeeper controller constantly monitors existing cluster objects to detect policy violations.

**Why:** Building and enforcing security policies is one of the core pillars of protecting containerized applications in Kubernetes. Providing standard policies that can be enforced across environments ensures consistent security and compliance.

**How:** Users can write policies using the OPA custom programming language, Rego. Rego has a simple syntax and a small set of functions and operators, optimized for query evaluation. You can learn more about using OPA Gatekeeper by reading our two-part introductory blog series on OPA Gatekeeper. Read [part one](#) and [part two](#).

### 9. Implement network control and firewalls using network policies

**What:** Network policies are objects that allow you to explicitly state which traffic is permitted. With these policies in place, Kubernetes will block all other nonconforming traffic.

**Why:** Limiting network traffic in your cluster is a basic and important security measure. Kubernetes by default enables open communication between all services. Leaving this “default open” configuration in place means an Internet-connected service is just one hop away from a database storing sensitive information.

**How:** The Cloud Native Computing Foundation (CNCF) published a [blog](#) that will get you started.

## 10. Use Kubernetes Secrets to store and manage necessary sensitive information

**What:** Secrets are how you store sensitive data in Kubernetes, including passwords, certificates, and tokens.

**Why:** Your services may need to authenticate one another, other third-party services, or your users, whether you’re implementing transport layer security (TLS) or restricting access.

**How:** The Kubernetes project offers a [guide to Secrets](#). One key piece of advice: avoid loading secrets as environment variables, because having secret data in your environment is not a good general security practice. Instead, mount secrets into read-only volumes in your container. Get more information in this [Using Secrets](#) article.

## 11. Use an image scanner to identify and remediate image vulnerabilities

**What:** Scanners inspect the components installed in your images, including everything from the OS to your application stack. Scanners are useful for finding vulnerabilities in the versions of software that your image contains.

**Why:** Vulnerabilities are discovered in popular open source packages all the time. Some notable examples are Heartbleed and Shellshock. You will want to know where such vulnerabilities reside in your system, so you know what images may need updating.

**How:** Scanners are a fairly common bit of infrastructure. Leading container registries have a scanner offering. If you want to host something yourself, the open source Clair project is a popular choice.

## Keeping your cluster stable

Kubernetes represents a tall stack. You have your applications, running on baked-in kernels, running in virtual machines (VMs) or on bare metal, accompanied by Kubernetes’ own services sharing hardware. With all of these elements, plenty of things can go wrong, both in the physical and virtual realms, so it is very important to reduce risk in your development cycle wherever possible. The ecosystem around Kubernetes has developed the following set of best practices to help reduce risk.

## 12. Follow continuous integration and continuous delivery (CI/CD) methodologies

**What:** CI/CD is a process philosophy. It is the belief that every modification committed to your codebase should add incremental value and be production-ready. So if something in your codebase changes, you probably want to launch a new version of your service, either to run tests or to update your exposed instances.

**Why:** Following CI/CD helps your engineering team keep quality in mind in their day-to-day work. If something breaks, fixing it becomes an immediate priority for the whole team, because every change thereafter, relying on the broken commit, will also be broken.

**How:** Owing to the rise of cloud-deployed software, CI/CD is a popular methodology. As a result, you can choose from many great offerings, from managed to self-hosted. If you are part of a small team, managed offerings can help you save time and effort.

### 13. Use canary methodologies for rolling out updates

**What:** A canary deployment is a way of bringing service changes from a commit in your codebase to your users. You bring up a new instance running your latest version, and you migrate your users to the new instance slowly, gaining confidence in your updates over time, instead of swapping over all at once.

**Why:** No matter how extensive your unit and integration tests are, they can never completely simulate running in production—there is always the chance that something will not function as intended. Using canary methodologies limits your users' exposure to these issues.

**How:** Kubernetes, as extensible as it is, provides many routes to incrementally roll out service updates. The most straightforward approach is to create a separate deployment that shares a load balancer with currently running instances. The idea is that you scale up the new deployment while scaling down the old until all running instances are of the new version.

### 14. Implement monitoring and integrate it with security information and event management (SIEM)

**What:** Monitoring means tracking and recording what your services are doing.

**Why:** No matter how great your developers and security experts are, things will go wrong. When they do, you will need to know what happened to ensure that it does not happen again.

**How:** There are two steps to successfully monitor a service—the code needs to be instrumented, and the output of that instrumentation needs to be fed somewhere for storage, retrieval, and analysis. How you perform instrumentation is largely dependent on your toolchain, but a quick web search should give you a starting place. For storing the output, consider a managed SIEM technology, like Splunk or Sumo Logic, unless you have specialized knowledge or need.

## Advanced topics

Once your clusters reach a certain size, enforcing your best practices manually becomes impossible, and, as a result, the safety and stability of your systems will be challenged. After you cross this threshold, consider the following topics:

### 15. Manage interservice communication using a service mesh

**What:** A service mesh is a way to manage your interservice communications, effectively creating a virtual network that you use when implementing your services.

**Why:** Using a service mesh can alleviate some of the more tedious aspects of managing a cluster, such as ensuring that communications are properly encrypted.

**How:** Depending on your choice of service mesh, getting up and running can vary wildly in complexity, and your configuration process will largely depend on your workloads.

A word of warning: If you expect to need a service mesh in the future, set it up sooner rather than later. Incrementally changing communication styles within a cluster can be challenging.

## 16. Use admission controllers to unlock advanced features in Kubernetes

**What:** Admission controllers are a great catch-all tool for managing what is going into your cluster. They allow you to set up webhooks that Kubernetes will consult during bring up. There are two types of admission controllers: mutating and validating. Mutating admission controllers alter the configuration of the deployment before it is launched. Validating admission controllers get permission from your webhooks that a given deployment is allowed to be launched.

**Why:** Admission controller use cases are broad and numerous—they provide a great way to iteratively improve your cluster’s stability with home-grown logic and restrictions.

**How:** To learn more, [read this guide](#) on how to get started with admission controllers.

### Implementing Kubernetes-native security with Red Hat




Security platforms purpose-built to protect Kubernetes offer powerful security and operational advantages. Kubernetes-native security applies controls at the Kubernetes layer, ensuring consistency, automation, and scale. Organizations successfully deploy security as code with security that is built in, not bolted on.

Download this whitepaper—[Kubernetes-native security: What is it and why it matters](#)—to find out more about the key features and benefits of Kubernetes-native security. Learn how it is different from existing container security approaches and how it delivers protections that are purpose-built for Kubernetes environments.



### About Red Hat

Red Hat is the world’s leading provider of enterprise open source software solutions, using a community-powered approach to deliver reliable and high-performing Linux, hybrid cloud, container, and Kubernetes technologies. Red Hat helps customers develop cloud-native applications, integrate existing and new IT applications, and automate and manage complex environments. [A trusted adviser to the Fortune 500](#), Red Hat provides [award-winning](#) support, training, and consulting services that bring the benefits of open innovation to any industry. Red Hat is a connective hub in a global network of enterprises, partners, and communities, helping organizations grow, transform, and prepare for the digital future.

 facebook.com/redhatinc  
 @RedHat  
 linkedin.com/company/red-hat

**North America**  
 1 888 REDHAT1  
 www.redhat.com

**Europe, Middle East,  
and Africa**  
 00800 7334 2835  
 europe@redhat.com

**Asia Pacific**  
 +65 6490 4200  
 apac@redhat.com

**Latin America**  
 +54 11 4329 7300  
 info-latam@redhat.com