DuploCloud

EBOOK

Containerization Best Practices: The Definitive Checklist for Tech Leaders

CONTENTS

INTRODUCTION

Containerization is the future of cloud-native application development. Many organizations, from small businesses to enormous enterprises like **Spotify** and **Major League Baseball**, rely on containers to build and scale their infrastructure to meet the needs of global consumers.

There are numerous benefits to embracing a containerized application architecture:

- Containers are versatile, easy to create, and can scale rapidly.
- They enable developers to ensure applications can run on any device, regardless of configuration or dependencies.
- Developers can build one-off images or coordinate multiple containers through powerful orchestration tools to serve a global user base.

According to Gartner, 95% of organizations will run containerized applications in production by 2028. However, many organizations still rely on virtual machines to manage their legacy applications. Others are looking to the cloud as a new frontier and investigating whether containerized development suits their business.

Whatever your needs, DuploCloud is here to help. We've provided a breakdown of containerization architecture to help you decide which is best for your use case. We've also included a list of best practices for developers looking to get the most out of containerized application development, ensuring security, performance, and longevity for their container clusters.

WHY CONTAINERS?

For many years, cloud development relied on virtual machines (VMs) to migrate legacy applications and ensure compatibility across devices. VMs can emulate specific operating system environments, allowing multiple applications or operations to run within a single virtual machine.

However, there are several drawbacks to using VMs that make them less than ideal for modern cloud-native development environments.

- **VMs are large**, often taking up an ungainly amount of disk space within the network.
- **VMs can take a long time to spin up**, making rapid scalability difficult and time-consuming under larger application loads.
- **VMs require a significant amount of resources** to be deployed effectively.

To meet the scalability and adaptability needs of the modern global cloud architecture, developers began turning to containers. While containers have been in use since the 1970s, in 2013, they became the backbone of cloud-native development with the creation of **Docker**, the most popular containerization program in the world.

Containers address many of the shortcomings of virtual machines:

- **Containers are smaller**, lightweight, and take up fewer resources than VMs.
- **Containers are far quicker to spin up**, making them more versatile under unpredictable workloads.
- **Containers typically only contain a single program**, making it far easier to determine each container's purpose.

Containerized architecture became a necessity with the rise of microservices. Multiple containers can work in concert through orchestration tools like **Kubernetes**, which provides the architecture for highly complex cloud-native applications. However, Kubernetes is very complex, requiring extensive amounts of coding. **Tools such as DuploCloud** enable developers to use Kubernetes constructs with little or no custom coding to simplify deployment, updating, and debugging.

When deciding which containerization architecture your application will rely on, there are **two primary choices:** Docker and Kubernetes.

Both tools are extremely useful in building and maintaining a containerized infrastructure, but they're best used for specific tasks; knowing what you need ahead of time will help you decide which is right for your business. It's also important to remember that these tools are not mutually exclusive, as many organizations that rely on Kubernetes to orchestrate their container infrastructure use Docker to initialize their containers.

Docker vs. Kubernetes

	Docker	Kubernetes
What is it?	The most widely used containerization software in the world.	A powerful open-source container orchestration system.
Benefits	• **Easy to use:** Can be up and running within minutes. • **Easy to share:** Dockerfiles and images can be easily uploaded to platforms like GitHub, allowing anyone in the world to download and run applications on their device. • **Simplified integration:** A series of APIs and SDKs allow Docker to integrate easily into any development environment.	• **Highly scalable:** Automatically deploys additional pods (i.e., extra copies of container clusters) through horizontal scaling, ensuring applications remain reliable under unpredictable workloads. • **Powerful error reduction features:** Offers automated self-healing and rollback features, removing failing clusters and replacing them with new ones to keep applications running smoothly.
Drawbacks	• **Difficult to scale:** While Docker does include a Swarm Mode for managing clusters of containers, it does not offer the same level of automation and orchestration capabilities as Kubernetes.	• **Highly complex:** Managing and optimizing a Kubernetes cluster for maximum efficiency requires a highly specialized knowledge set, which some organizations may not have. • **Resource intensive:** Kubernetes' robust scaling and redundancy capabilities come with a resource cost, requiring a significant investment in cloud networking components.
Best for:	Organizations of any size that need to share and run single or small groups of containers.	Large organizations with the need for extensive container orchestration and scaling capabilities, usually for cloud-native applications that must remain fast and stable on a global scale.

1 One Application Per Container

One of the most important things to remember when building your containerized infrastructure is that **containers are not virtual machines.** Just because they can be configured to run multiple applications at once doesn't mean they should—and doing so means you're missing out on the key benefits of transitioning to a containerized development environment.

Instead, stick to using one application or application component per container. Limiting containers in this way will ultimately benefit your application by:

- Reducing library compatibility issues on a per-container basis.
- Increasing visibility into container health to minimize errors.
- Making horizontal scaling easier.
- Enabling easier reuse of containers.
- Improving project organization, especially when orchestrating multiple containers.
- Minimizing debugging times by limiting the complexity of individual containers.

If your application relies on multiple containers operating simultaneously, you can orchestrate them to work with Docker's Swarm Mode or use Kubernetes to manage larger clusters.

2 Containers Should Be Stateless and Immutable

Containers should not be treated like a traditional server — that is, you won't want to update applications inside a container that is built and running. Instead, you should consider containers to be **stateless** and **immutable.**

Stateless

Instead of storing persistent data within the container, data should be stored outside, either within cloud storage or on external disks.

Designing containers to be stateless ensures that they can be destroyed or rebuilt as needed without worrying about losing crucial information.

Immutable

The container should not be modified while it exists. If you need to update content within the container, destroy it, make your changes, build a new image, and redeploy it.

This ensures that containers remain identical when deployed across multiple environments. It also makes rolling back to previous images easier if problems are discovered within more recent versions.

3 Optimize for Build Cache

One of the most significant benefits of using containers is how quickly developers can spin them up as needed, especially in comparison to larger virtual machines. To optimize your build times, ensure you're **leaning on Docker's build cache capabilities** wherever possible.

Docker images are powered by **Dockerfiles,** which describe the instructions for building the container. When building an image, Docker constructs it in layers. Docker can reuse layers from a previous build to speed up build times. Using the command **--cache-from** in your Dockerfile will tell Docker to use the specified image as a source to cache build commands from, helping you speed up future build times.

To get the most out of the build cache, put build steps that change often at the bottom of the Dockerfile. Because Docker builds images in layers, it will use its build cache for earlier build steps that change less frequently, helping to reduce build times more reliably.

4 Reduce Image Size as Much as Possible

Another way to optimize container performance is to **reduce your image size** as much as possible. Doing so will help to reduce download and upload times, enabling you to operate more efficiently.

Smaller images tend to be less complex and rely on fewer dependencies to run. Plus, smaller images tend to have less bloat, reducing the potential attack surface for malicious actors.

Rely on the following techniques to make your container images as small as possible:

- **Use multi-stage builds** to create a cleaner separation between the building of the image and the final output. Multi-stage builds also make Dockerfiles easier to maintain.
- **Use the .dockerignore file** to exclude files not relevant to the build, ensuring your image only contains the files it needs to run.
- **Remove unnecessary tools from the containers.** For example, you don't need a text editor in a database image, so don't include it. Examine what you want your container to accomplish and only have the necessary tools to achieve that task.

5 Maximize Security Posture

Like any aspect of your development pipeline, containers can act as an additional attack surface malicious actors can exploit to gain access to sensitive systems, information, or even the entire Kubernetes cluster. Take the following precautions when constructing your containers to reduce the likelihood of a data breach:

- **Don't use privileged containers, and avoid running container processes as root.** While there may be reasons for doing so, you should avoid these scenarios unless absolutely necessary to limit the likelihood of a malicious actor attaining unrestricted access.
- **Integrate automated vulnerability scanning to reduce the likelihood of attack.** Regular scans will help inform you of potential vulnerabilities, allowing you to plug these security holes as they arise.
- **Automate infrastructural updates** to ensure you're receiving the latest security patches.
- **Remember to keep containers immutable.** Destroy old images, make updates, and then redeploy to reduce the likelihood of introduced vulnerabilities entering the cluster.

6 Build Logging and Monitoring Standards Into Container Architecture

Monitoring systems are crucial to ensure your containers remain healthy, so utilize built-in data logging systems and external platforms regularly to address performance or vulnerability issues.

For example, Docker provides a standardized way to record logs using the **"docker logs"** command. Use this native logging mechanism instead of external ones to get the best insight into individual Docker container performance.

However, getting a complete view of your entire container orchestration will require additional external tools, **especially if you're using Kubernetes.** Monitoring tools like **Prometheus** provide a single pane of glass for viewing container performance, giving you at-a-glance metrics of the entire cluster.

Additionally, configuring **liveness, readiness, and startup probes** into your containers will help you automate the gathering of necessary metrics from each of the pods in your Kubernetes cluster. That way, your monitoring system will scale up and down as pods are created or destroyed, allowing you to restart or fix containers as needed.

7 Use Repositories like Docker Hub and Tag Images

Like any organization's codebase, the quality, security, and longevity of container images and Dockerfiles will benefit from **robust version control and naming conventions.**

Images are usually identified by their name and tag; these let internal developers and external users know what image they're downloading along with other relevant information, such as the version number or date it was created.

Use consistent tagging policies to ensure your images are organized. You'll also provide users with an easy way to find and download a specific version of your image, if necessary. Many developers rely on the **Semantic Versioning Specification** when tagging images, as it provides guidelines that are commonly understood across the globe.

Images should also be uploaded to a centralized repository to provide comprehensive access for anyone who needs to work on or download them and increase the visibility of any changes made to the image with a version history archive. **Docker Hub** is a large public repository like GitHub, which provides quick access to uploaded images and Dockerfiles from anywhere in the world. Organizations that need a private repository can create and host their own **Distribution Registry** on internal servers for increased security and privacy.

SUMMARY

Containers unlock the full potential of cloud-native application development. However, getting the most out of your container cluster requires following best practices that will help you optimize processing speeds while improving container health and security. When building your containers, be sure to keep the following in mind:

- Use one application or component per container.
- Containers should be stateless and immutable to maximize integrity and security.
- Optimize your Dockerfiles to take advantage of the build cache for faster deployment.
- Reduce image size as much as possible to optimize transfer times and reduce potential attack vectors.
- Don't use privileged containers, avoid running processes as root, and run regular scans and updates to maximize your security posture.
- Build logging and monitoring systems into your architecture to keep container orchestration healthy and scale your visibility into cluster performance.
- Implement robust version control and naming conventions to improve usability and reduce errors.

ABOUT DUPLOCLOUD

Manually configuring and building containers is an error-prone, time-consuming process, especially if you want your infrastructure to operate on a global scale.

That's why we developed DuploCloud: a DevOps automation platform that seamlessly orchestrates container configuration, along with other crucial elements of the DevOps pipeline, with a centralized no-code/low-code platform. Plus, DuploCloud provides 24/7 performance and security monitoring and reporting, ensuring that your infrastructure remains secure and compliant, no matter how much it scales to meet user demand.

Learn more about how DuploCloud speeds up your organization's deployment times by a factor of ten. **Contact us today** for a personalized one-on-one walkthrough, and see DuploCloud in action for yourself.

DuploCloud

duplocloud.com