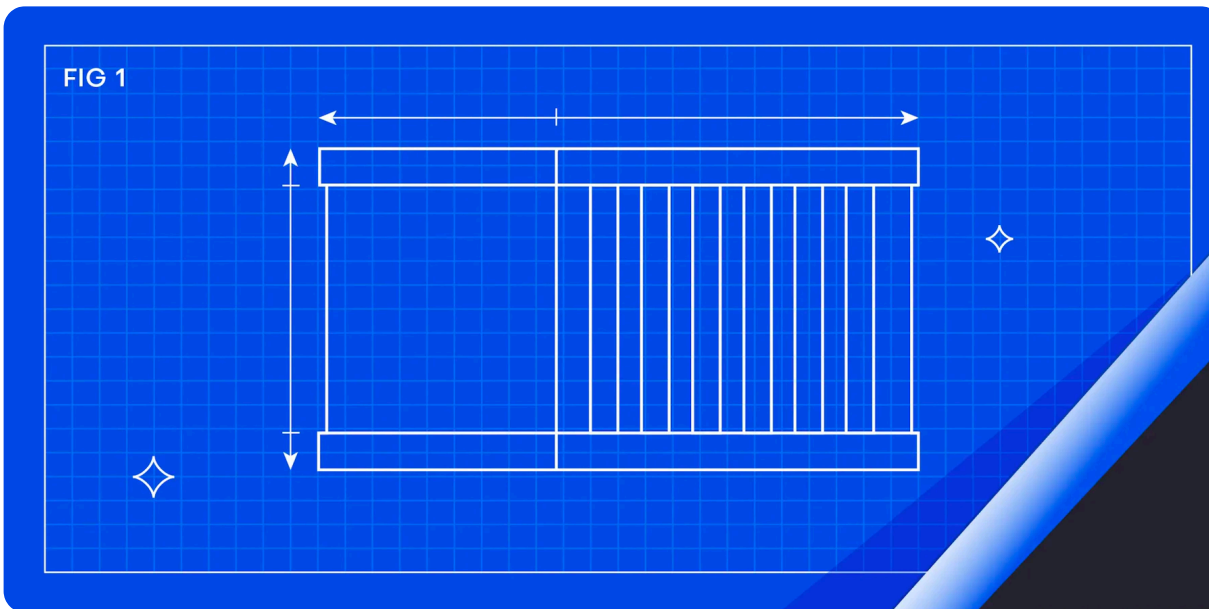


[← All articles](#)

Containerization 101: Transforming Application Development and Deployment



Nicolas Ehrman

February 6, 2025

8 minute read

Containers Cheat Sheet

Kubernetes Cheat Sheet

Main takeaways from Containerization:



Containerization is a cloud resource allocation method which bundles (encapsulates) software applications and their operating system

<https://www.wiz.io/academy/containerization>

1/18

libraries and dependencies into lightweight packages called containers.

- Containerized environment components include host machine hardware, host operating systems, container engines, and containerized applications.
- Containerization supports microservice deployments, legacy migrations, hybrid and multicloud applications, CI/CD pipeline management, and edge computer and IoT workload management.
- Containerization works by using configuration files to specify system environments and create container images for storage and use.
- Popular containerization technologies include Docker, LXC, Windows Server containers, and Kubernetes.

What is containerization?

Containerization is a cloud resource allocation method that bundles (encapsulates) software applications and their operating system libraries and dependencies into lightweight packages called containers. This packaging ensures that the application works the same way no matter where it's deployed, whether that's an on-premises system or a cloud computing platform.

Containers are brought to life by container engines, using container images—pre-made templates containing the app and its environment—to create these neatly packaged units. These engines operate on top of the host machine's operating system, making it easy to build, manage, and run containers.

The beauty of containers lies in their portability. By abstracting the application and its dependencies from the host OS, containers can move seamlessly across platforms and clouds, ensuring consistent performance wherever they go.

Advanced Container Security Best Practices [Cheat Sheet]

This cheat sheet goes beyond the no-brainer container security best practices and explores advanced techniques that you can put into action ASAP. Use this cheat sheet as a quick reference to ensure you have the proper benchmarks in place to secure your container environments.



[Download now >](#)

Containerization vs. virtual machines (VMs)

[Containers and VMs](#) are two powerful technologies for deploying applications, but they take different approaches.

- **Containerization:** Containers rely on the same shared operating system but keep each app's files and processes separate. This makes containers lightweight, quick to start, and easier to manage. They use fewer resources and are ideal for environments where speed and efficiency are crucial.
- **Virtual machines (VMs):** Virtual machines act like full computers, running multiple operating systems on top of the host machine. While this provides strong isolation, it also makes VMs slower to start, more resource-heavy, and more expensive to maintain due to the extra layers involved.

The components of a containerized environment

Containerization relies on [several interconnected layers](#) that work together to create an efficient environment for running applications. These components include:

- The host machine's hardware infrastructure
- The host operating system
- A container engine
- Containerized applications

Hardware

Think of the host machine's physical hardware as the foundation of your containerized environment. It supplies the compute power, storage, and networking resources needed to keep everything running. Containers can operate on a variety of setups, from a bare metal server to virtualized hosts, depending on the organization's infrastructure. This hardware layer supports the host operating system and the container engine, providing the muscle behind container creation and execution.

Host operating system

The host operating system is the glue holding everything together. It ensures the container engine can talk to the hardware while allocating the resources needed for containerized applications. This smooth communication allows containers to operate seamlessly, bridging the gap between the underlying infrastructure and the application itself.

Container engine

The [container engine](#) is the heart of containerization. It's responsible for creating containers from pre-built templates called [container images](#). The engine manages, runs, and keeps containers efficient, all while sitting atop the host operating system. By providing access to the hardware while maintaining isolation, it ensures containers are lightweight, portable, and secure.

Containerized applications

Containerized applications are self-contained bundles that include all the software, libraries, and dependencies they need to function. These applications are isolated—not just from the host operating system but also from other containers. This isolation ensures consistent performance across environments, eliminating compatibility issues and reducing bugs when moving apps between systems.

How containerization works

Containerization starts by creating the container image: a compact, self-contained executable package that encompasses all the essentials for the software's operation, such as the code, libraries, and configuration.

[Container images](#) are built from a Dockerfile or similar configuration files that specify the application's environment. Once created, these images are stored in a [container registry](#), such as Docker Hub, where they can be downloaded and run on any system with a container engine installed. This engine, such as Docker or Podman, is responsible for container life cycle management, including running, stopping, and managing container instances.

Container engines play a crucial role in the host operating system, leveraging the kernel's features (such as namespaces and cgroups) to isolate the container's processes and manage resources. Each container has its own isolated user space, file system, and network stack, allowing multiple containers to run simultaneously on a single host without interference.

The workings of containerization, from image creation to orchestration, underscore its efficiency and flexibility:

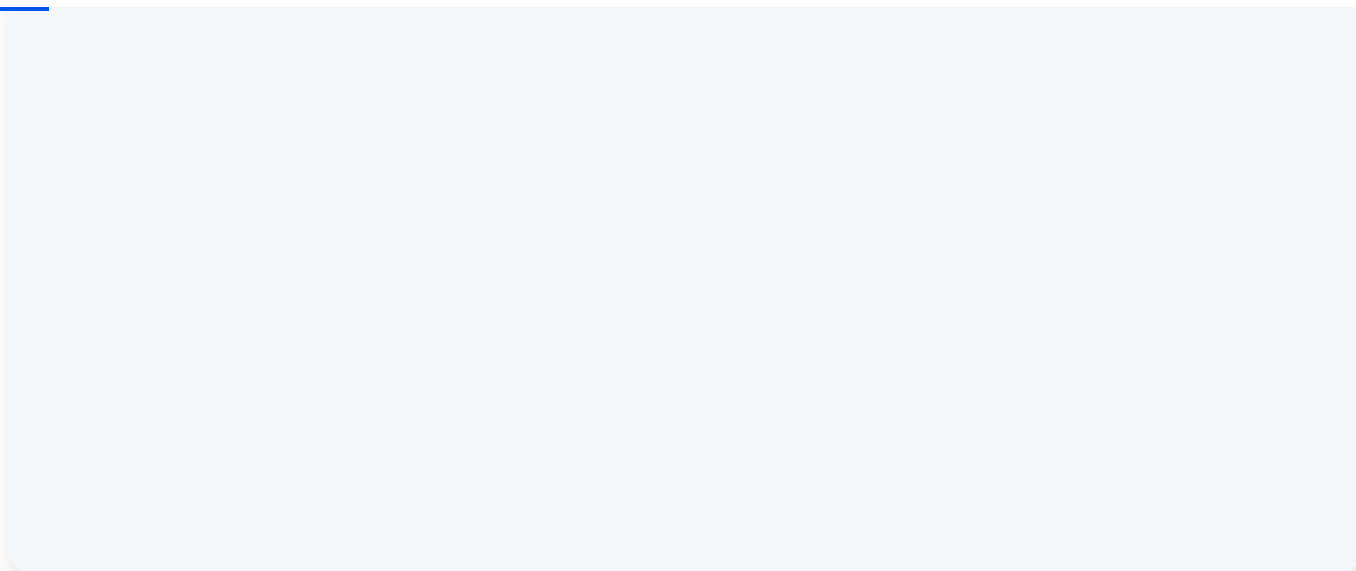
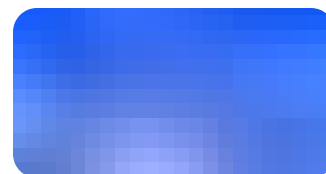


Figure 1: Containerization overview (Source: Docker)

By abstracting away the underlying hardware and operating systems, containerization allows developers to focus on building and deploying applications without worrying about the environment where the application will run.

WIZ BLOG

Kubernetes API limitations in finding non-standard pods and containers



Read more >

Benefits of containerization

The adoption of containerization brings a multitude of benefits for software development and deployment:

- **Improved resource utilization:** Unlike virtual machines, containers share the host's kernel, reducing overhead and enhancing server efficiency. Additionally, this approach minimizes hardware costs and boosts application scalability, making it an ideal method to deploy modern applications that demand efficiency and flexibility.
- **Increased developer productivity:** Containerized applications are encapsulated with their environment into a single container image, promoting consistency across development, testing, and production. Uniformity fosters a DevOps culture, streamlining the development life cycle through continuous integration and delivery (CI/CD).
- **Simplified configuration and testing:** Utilizing configuration files for container settings keeps applications performing consistently across different environments, mitigating bugs and discrepancies.
- **Increased security and fault tolerance:** Containers operate independently, providing fault isolation that secures applications from affecting each other. Isolation is vital for the stability and reliability of complex systems applications and protects the host from any malicious code contained within a compromised container.

Practical applications of containerization

Containers stand out in a variety of use cases:

- Microservice deployments
- Legacy migrations
- Hybrid and multicloud environment applications
- CI/CD pipeline management
- Edge computing and Internet of Things (IoT) workload management

Deploying microservices-based applications

[Microservices](#) break down complex applications into smaller, independently deployable components. Containers are a natural fit for this architecture, offering a lightweight, modular way to manage these pieces. Need to scale one part of your app? Containers make it easy to adjust individual components without disrupting the rest—a huge advantage for flexibility and resource efficiency.

Migrating legacy applications to modern infrastructure

Migrating legacy applications to new platforms can be overwhelming, but containerization makes it manageable. By decoupling applications from their original environments, containers turn them into portable, self-contained units. This allows you to move even monolithic legacy systems to modern infrastructures without compromising functionality or reliability.

Running applications across hybrid or multi-cloud environments

[Hybrid](#) and [multicloud](#) setups often use different cloud providers and systems, which can be tricky to manage because they don't always work well together. Containers make this easier by being portable and consistent, so applications can run smoothly no matter which cloud or platform you're using.

Enabling consistent environments for CI/CD pipelines

DevOps teams know the frustration of those “Well, it worked on my machine” moments. Containers eliminate these inconsistencies by creating uniform environments for testing and deployment in [CI/CD pipelines](#). Containers ensure that what works in development will work in production, streamlining the entire software deployment process.

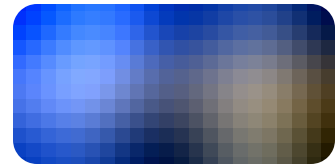
Supporting edge computing and IoT workloads

Edge computing and IoT apps need to respond quickly and run smoothly on devices with limited power and storage. Containers are great for this because they're lightweight and flexible, making it easy for them to work efficiently on these smaller devices. This allows the apps to process data in real-time without slowing down or losing reliability.

WIZ ACADEMY

Containers vs. VMs: What's the difference?

Read more >



Popular containerization technologies

The container technology landscape is rich and varied, with solutions designed to cater to different aspects of the container life cycle—from image creation and management to orchestration and security:

Docker

[Docker](#), which is now considered a synonym for containerization, provides an extensive platform that streamlines the creation, distribution, and execution of applications within containers. The Docker engine is at the heart of Docker's platform, enabling containers to be packaged and run consistently across different environments. Docker also adheres to industry standards set by the Open Container Initiative (OCI), ensuring interoperability and consistency across container technologies.

Docker benefits from an impressive community and ecosystem. It offers extensive documentation, a vast library of container images on Docker Hub, and community support, making it an ideal starting point for those new to containerization.

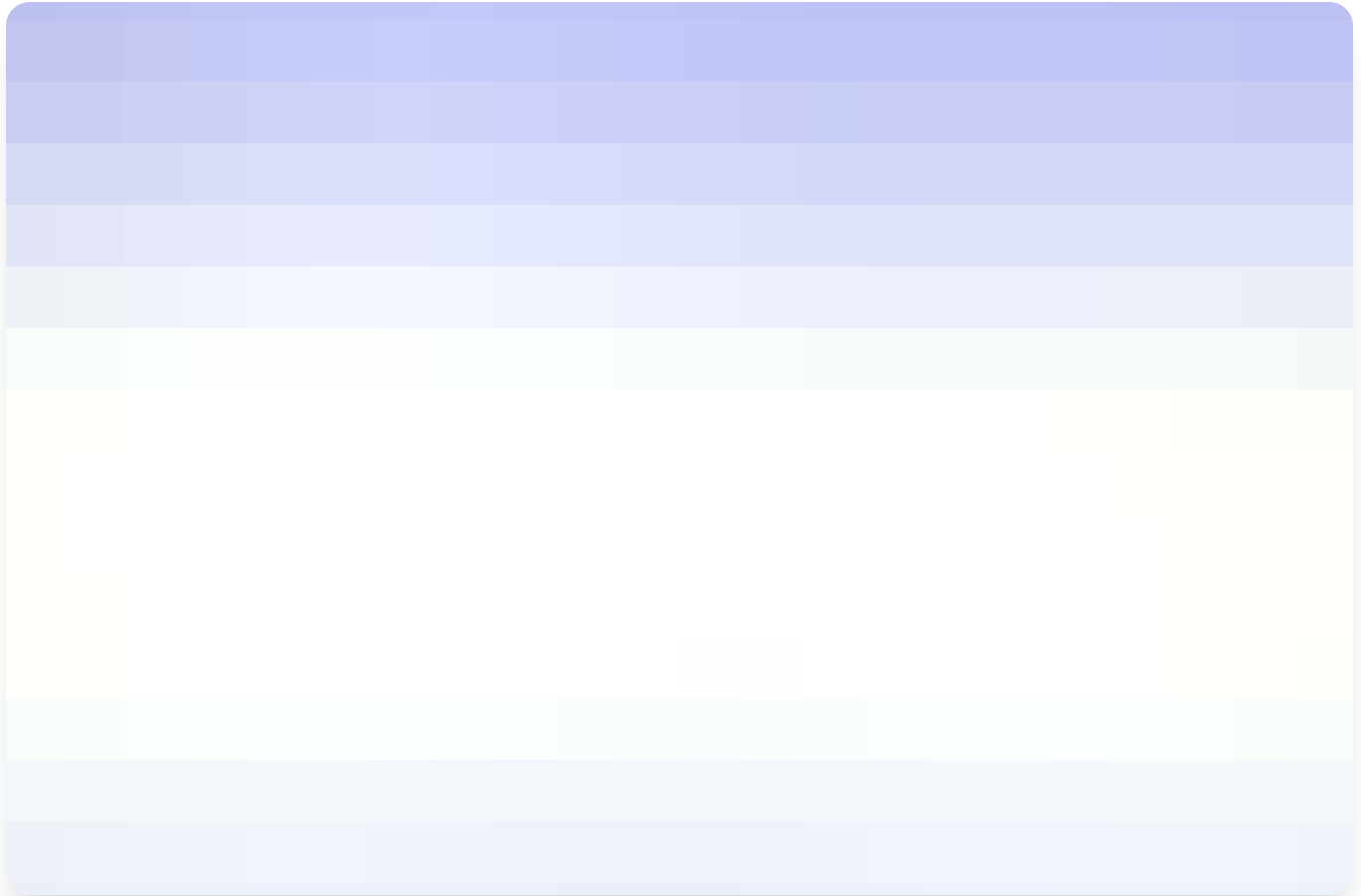


Figure 2: Docker Hub (Source: Docker)

LXC (Linux Containers)

[LXC \(Linux Containers\)](#) is a more traditional containerization technology that predates Docker. The key difference between LXC and Docker lies in their approach to containerization. LXC is more like a lightweight VM, offering a complete Linux system within each container, whereas Docker focuses on application containerization, making it easier to package and ship applications:

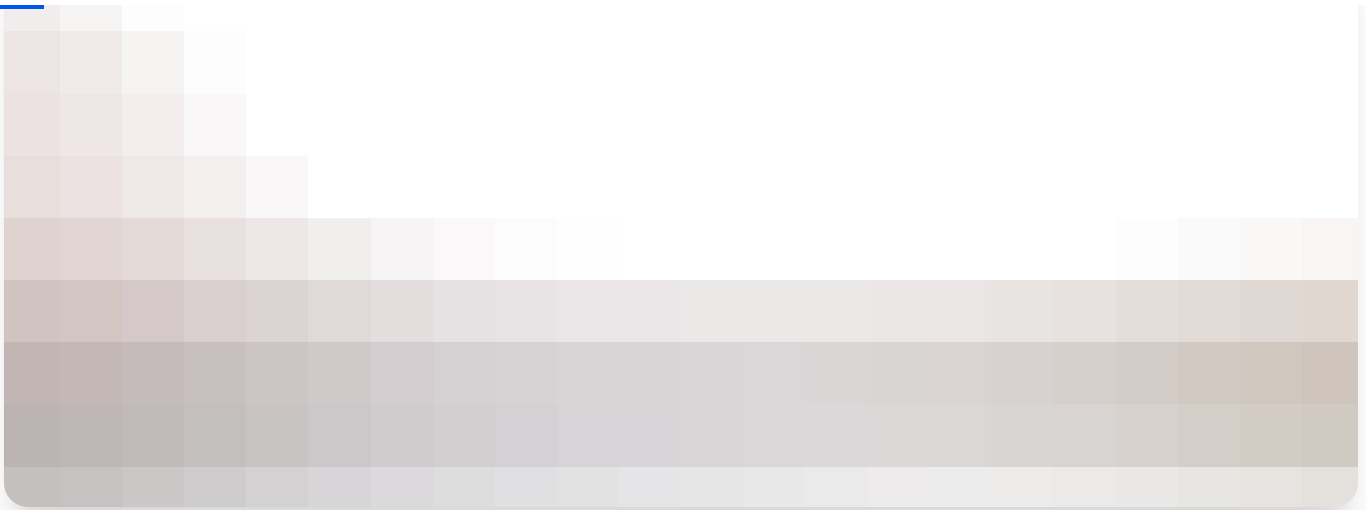
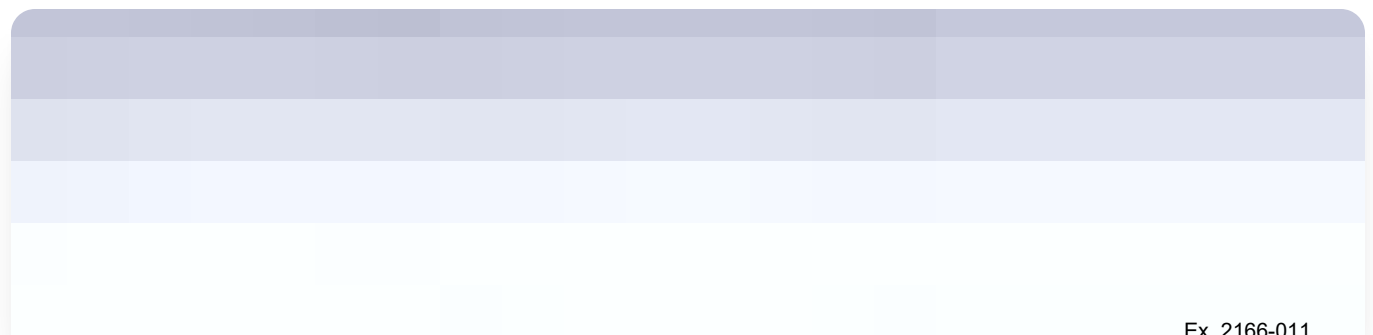


Figure 3: LXC, Docker, and VM architecture (Source: Oracle Forums)

LXC is preferable for scenarios requiring full Linux system containers rather than just application containers. It offers a solution closer to virtual machines but with the efficiency of containers. LXC maintains a dedicated community and documentation, though it is smaller than Docker's expansive ecosystem.

Windows Server containers

[Windows Server containers](#) offer containerization technology integrated with the Windows Server operating system. This enables Windows-based applications to be containerized and managed similarly to Linux-based containers. While Docker focuses on Linux containers, it also supports Windows containers, providing a cross-platform solution for containerization. Windows Server containers, however, are specifically optimized for the Windows environment and provide a path to modernizing legacy Windows applications through containerization.



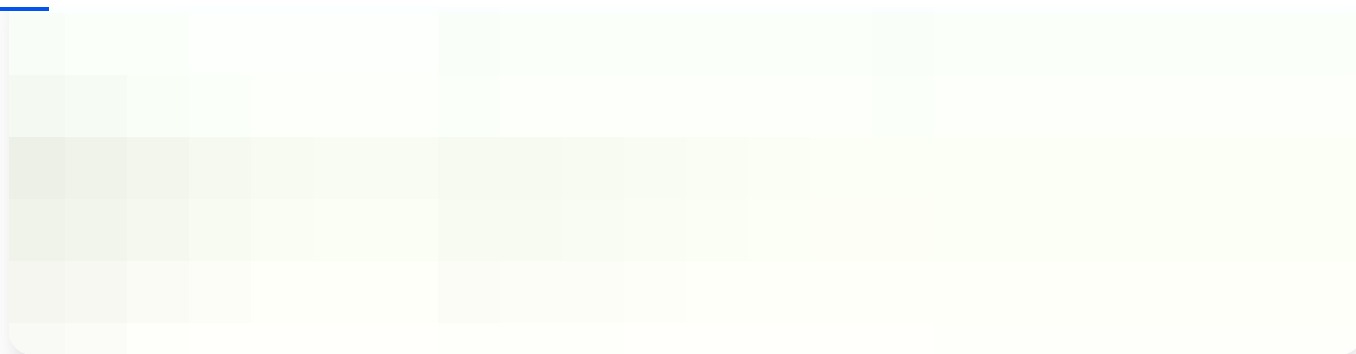


Figure 4: Windows containers in Windows Admin Center (Source: Microsoft)

Kubernetes

Kubernetes (often abbreviated as K8s) is an open-source tool that helps manage containers by automating their deployment, scaling, and organization. It groups containers into easy-to-manage units, making applications run more efficiently. Kubernetes is perfect for handling complex systems across different environments, which is why it's popular for large-scale applications.

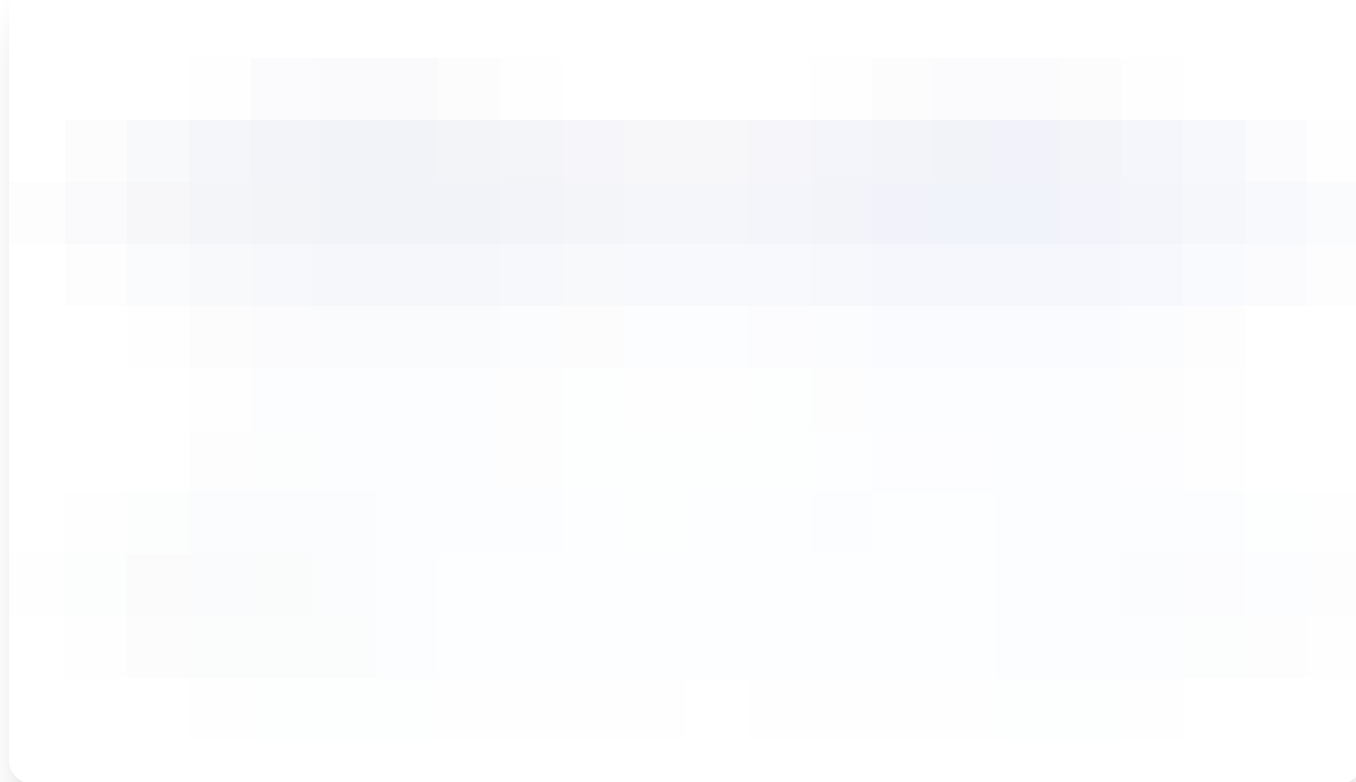


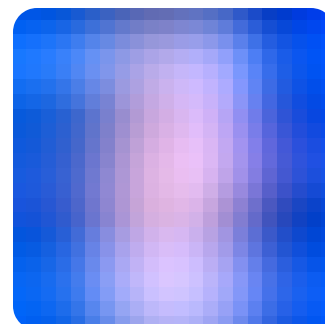
Figure 5: Kubernetes Dashboard UI (Source: Kubernetes)

When choosing the right container technology, consider your project requirements, target operating system, application, and desired level of isolation and efficiency.

The Container Security Buyer's Guide

Ensure the container security solutions you're considering can deploy the full set of capabilities to secure the entire CI/CD lifecycle.

[Download Guide >](#)



Containerization security: risks and strategies

Because containerization has become increasingly prevalent in software development and deployment, understanding and addressing its security implications is paramount. Though containers offer numerous benefits, they also introduce specific security challenges that must be managed to protect applications and data effectively.

“The more widely companies use containers, the more likely they are to call security their top challenge with containers.”

[CNCF Annual Survey](#)

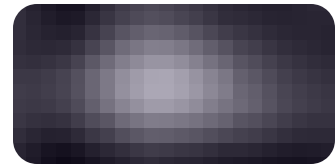
Common security risks in containerized environments include:

- **Image vulnerabilities:** Containers are only as secure as their base images, and vulnerable images can introduce security risks to the environment.
- **Misconfigurations:** Incorrectly configured containers or container orchestration tools can expose applications to security threats.
- **Runtime security:** Containers share the host's kernel, which can lead to potential escape vulnerabilities if a container is compromised. [Container Runtime Security Explained ->](#)
- **Network security:** Containers often communicate with each other and with external services, necessitating robust network security policies to prevent unauthorized access.

To mitigate these risks, several strategies for [securing containerized applications](#) can be employed:

- **Secure the build pipeline:** Implement security checks and vulnerability scanning within the CI/CD pipeline to ensure container images are secure before deployment.
- **Use trusted base images:** Only use container images from trusted registries and maintain them by installing patches and updates to reduce the risk of vulnerabilities.
- **Leverage isolation and segmentation:** Use network policies and other mechanisms to isolate containers from each other and segment them from the broader network, limiting the potential spread of malicious activity.
- **Implement continuous security assessments:** Continuously scan container images for vulnerabilities and monitor running containers for suspicious activity, employing tools designed for container environments.

Container Security Best Practices

[Read more >](#)

Simplify and secure containerized environments with Wiz

Containerization has fundamentally reshaped software development and deployment, offering a pathway to more efficient, scalable, and consistent application delivery. As we look to the future, container technology continues to evolve, promising even greater advancements in [orchestration](#), security, and performance optimization. The ongoing development of this ecosystem suggests a future where containerization both simplifies and also accelerates the pace of software innovation.

When it comes to navigating the complexities of containerized deployments, [Wiz](#) is a pivotal ally. With our unified cloud security platform, we offer prevention, detection, and response capabilities that empower you to build and run secure, efficient applications in the cloud.

Wiz's approach to [container and Kubernetes security](#) allows teams to rapidly build containerized applications without compromising on risk. Our platform's comprehensive coverage—from managing vulnerabilities, secrets, and misconfigurations across clouds and workloads to continuous monitoring for suspicious activity—ensures that containerized environments are holistically secured from build time to runtime.

To see for yourself how Wiz can help you keep your containerized applications and Kubernetes environments secure from development to production, contact

us today to [get a personal demo](#).

What's running in your containers?

Learn why CISOs at the fastest growing companies use Wiz to uncover blind spots in their containerized environments.

[Get a demo >](#)



Explore more on this topic

Introduction to Containers and Containerization

- [Containerization 101: Transforming Application Development and Deployment >](#)

- [What is a Container Engine? >](#)
- [Containers vs. VMs: What's the difference? >](#)
- [Container Architecture: A Security Review >](#)
- [Linux Container Security: Benefits, Risks, & Best Practices >](#)
- [Container Runtimes Explained >](#)
- [Container Images Demystified: Structure, Security, and Best Practices >](#)

PLATFORM

[Wiz CNAPP](#)[Wiz Code](#)[Wiz Cloud](#)[Wiz Defend](#)[Integrations](#)[Environments](#)[Documentation](#)

LEARN

[Customer stories](#)[Train Your Team For Cloud](#)[Blog](#)[CloudSec Academy](#)[Resources Center](#)[Cloud threat landscape](#)[Cloud Security Assessment](#)[Vulnerability Database](#)

COMPANY

[About Wiz](#)[Join the team](#)[Newsroom](#)[Events](#)[Contact us](#)[Trust Center](#)[Our partners](#)



[Status](#) [Privacy Policy](#) [Your California Privacy Rights](#) [Terms of Use](#) [Modern Slavery Statement](#)

[Cookie Settings](#)

© 2025 Wiz, Inc.