11 Leading Practices When Implementing a Container Strategy




ANDREW O. MELLINGER, WILLIAM NICHOLS, AND JAY PALAT

NOVEMBER 8, 2021

Containers are an application packaging format that help developers and organizations to develop, ship, and run applications. A container "contains" everything that an application needs to run on any system that hosts the specific container technology. Containers can provide a means of basic isolation for services, applications, and components. They will behave like a virtual machine with the benefit of not interfering with processes running around them. Developers use containers to standardize how they compose, package, deploy, and manage applications. Containers provide a manageable means to quickly redeploy a service in a specific configuration, replacing infrastructure with code. They enable reproducibility and ease of archiving configurations, combined with rapid deploy and tear down of services. For organizations, adopting containers can lead to lower costs in development, testing, and deployment. The cost of maintenance over time may also go down substantially with well-maintained containers built using good practices. By isolating processes and enabling multiple applications to run simultaneously, implementation of containers eases the application development lifecycle, increases reliability and security, and makes systems less prone to configuration errors. Containers also make system administration easier in that responsibility for software dependencies are moved to the container developer and away from the system administrator.

While containers are frequently lauded in the latest software development trends, switching from using virtual machines and deploying an organization-wide container strategy remains non-trivial. In this blog post we outline 11 leading practices for organizations looking to adopt and use containers.

Understand the Why

When adopting a new technology, like containers, how will it help you achieve your goals? Containers are used today because they effectively bundle applications, related libraries, dependencies, and configurations in a package that can be deployed across multiple environments. They ease reproducibility and reliability of build-time and run-time software environments. Instead of every application user needing to build up the environment (e.g., libraries, dependencies), the container specification file encapsulates everything to prevent library mismatches. Also, developers can consistently build and run containers on a variety of host environments (e.g., different OSes / different Linux distributions). Containers are lighter than virtual machines, allowing efficient use of hardware and creating higher utilization of existing hardware.

Play to Container Strengths

There are many features of containers that when used intentionally can ease application deployment significantly. Containers provide a system for isolating processes and data without the full virtualization of the whole operating system. Multiple containers can run together and do not share data unless explicitly configured to do so. An individual container can be changed without worrying about negatively impacting other applications or containers. The isolation eases application version changes and means that different versions of applications can be automatically built and tested. Containers are also portable, which allows developers to build on one host and move to another easily. The portability is especially useful for transitioning applications from servers in the cloud to smaller devices at the edge. The ability to reuse containers can lower costs and enable efficient resource use.

Be Aware of the Limitations

As with all technology adoption, container adoption should be driven by purpose, and organizations should not force fit in all scenarios. Containers have limitations. Graphical applications are generally more complex and require cumbersome video forwarding, which can make implementation of containers challenging. Builds can be difficult, especially with the introduction of anything requiring additional environment configurations, such as an enterprise proxy. Furthermore, not all hardware platforms (especially in the embedded space) support containers. Because containers are a relatively new advancement, security mechanisms are still evolving.

Containers are not optimized for monolithic applications, which can be expensive to rewrite or convert into microservices. Overall, as organizations think about adopting containers they should think strategically about where there are significant gains to be made.

Develop a Container Operationalization Process

Business needs, organizational capacity, and containerization technology are constantly changing and will continue to do so. As with modern development and IT practices, delivering containers "early and often" significantly improves an organization's ability to use, evaluate, and evolve the containers and the value provided to the users. Strategies include such aspects as pilot projects, evaluation periods, rollout processes, update cycles, and evolution roadmaps. Organizations must work to ensure that their operationalization remains aligned with the needs of end users, as failing to do so will lead to low adoption and wasted resources. As organizations begin to operationalize containers and related policies, they should evaluate how initial efforts, such as changes to workflows, affect end user productivity. Taking a proactive learning approach will help organizations to iterate on operational strategies and achieve desired outcomes.

Give People Time and Education for Transition

PUBLISHED IN
Secure Development

CITE
Get Citation

SHARE
This post has been shared 3 times.

Get updates on our latest work.

Sign up to have the latest post sent to your inbox weekly.

Subscribe

Get our RSS feed

MORE IN SECURE DEVELOPMENT

Detection and Repair: The Cost of Remediation
MARCH 3, 2025 • BY DAVID SVOBODA

Measurement Challenges in Software Assurance and Supply Chain Risk Management
MAY 20, 2024 • BY NANCY R. MEAD, CAROL WOODY, SCOTT HISSAM

What Recent Vulnerabilities Mean to Rust
APRIL 29, 2024 • BY DAVID SVOBODA

The SEI SBOM Framework: Informing Third-Party Software Management in Your Supply Chain
NOVEMBER 6, 2023 • BY CHRISTOPHER J. ALBERTS, MICHAEL S. BANDOR, CHARLES M. WALLEN, CAROL WOODY

Rust Vulnerability Analysis and Maturity Challenges
JANUARY 23, 2023 • BY GARRET WASSERMANN, DAVID SVOBODA

IPR2025-01084
Orca Security EX1046 Page 1

Education, training and planning can significantly reduce development time and transition risk. Container-focused deployments can be subtly different from bare-metal or virtual machine focused deployments. For developers who have never used them before, it takes a bit of time to get used to developing in a container environment. While perhaps slower than desired at first while developers are getting used to new workflows, containers can prevent many down-stream development issues (e.g., library mismatches) and in fact speed development in the long run. Consider also that there may be different stakeholders involved in building and deploying containers, and the training they need might vary as well.

Invest in Image Design and Container Execution Strategy

Image development requires significant time for design, development, and testing. Pursue best practices such as good base image selection, container hierarchies, dependency version management, package selection minimalism, layer management practices, cache cleaning, reproducibility, and documentation. When a container is run from an image, there are many options such as temporary containers, mounting volumes, and user accounts. A good image design process and system architecture process considers these options.

Maintenance Is a Continuous Process

Platforms, libraries, and tools will constantly fix defects and security issues, and any container deployment strategy must be prepared to integrate updates. At first glance it seems attractive to use automatic update features of the underlying operating system on container start, but that leads to increased startup times while reducing reproducibility and stability. Images should be rebuilt cleanly on a periodic basis incorporating vetted versions, patches and updates. Teams should frequently remove unnecessary or disused packages and assets as part of their maintenance process, test changes, and redeploy. One should expect to do this on a regular basis and allocate resources and budget appropriately. As images can quickly build up, a good image management strategy should be developed with versioning and removal. As new images are redeployed all existing containers should be restarted using the new images, which reinforces the idea of transient containers. When hierarchies of containers are used, remember to rebuild all dependent containers as appropriate.

Consider Security from the Start

Containers are not inherently secure; there are still concerns that must be addressed proactively. Many consider the isolation of containers to support their overall security. The level of real isolation provided by a containerized environment should be thought of as isolation of resources as opposed to a primary security mechanism and should be treated in addition to other security measures, not as a replacement for other methods. At the same time, isolation can be a weakness. For example, if the container runtime is not secured correctly and gets compromised, it can be another entry point for malicious activities. Container hardening should be integrated into the build process well before deployment.

Thinking about security considerations proactively and early can help reduce risk. Scanning individual images for potential vulnerabilities is and should be a standard practice in any new environment. When creating a container, be mindful of where that container will exist. Container networks exist as user-defined bridges and namespaces which provide basic isolation by controlling the flow of traffic across virtual network adapters. Existing security systems can be leveraged within an individual container and pulled down with images during the build process and should be considered as part of your deployment. Most importantly, defining and identifying attack surfaces clearly will allow engineers, developers, and organizations to look ahead and head off potential threats. Understanding what containers and services exist within which namespace, which containers can and cannot communicate with each other, which services are exposed to the outside world, and where threats exist are all good examples of what to examine.

Architect Your System With Containers in Mind

Organizations should be prepared to develop a capability to continuously evolve their system architecture as new business needs are encountered, new technologies are developed, and systems change. Container strengths can have significant impacts on how the system is decomposed into components, their responsibilities, connections, and lifecycle—and to take advantage of those strengths, the system architectures need to evolve. Conversely, containers have some weaknesses that need to be mitigated by changes to the system architecture. As with any technology change, it is best to deal with change in increments; therefore, having a strong organizational capability to plan, organize, and deploy incremental system changes is critical to any change while maintaining continuity of operations. Switches to container deployments are consequently easier because of their finer-grained architectures, recomposibility, and ease of deployment. As an example, containers emphasize process isolation as opposed to machine isolation, which leads to architectures with finer-grained decomposition. In newer systems, each container has a smaller set of responsibilities compared with classic architectures, and many newer systems are switching to micro-architectures. The dynamic nature of these more cohesive and decoupled services increases the need for container orchestration pieces, which can become a central need in container architectures. All of these changes require that organizations are prepared to develop a process to evolve their architecture as responsibilities are reallocated over time to take advantage of the newer capabilities. This environment is growing and changing and will continue to do so. Organizations must be prepared for continuous evolution and growth of their system architectures.

Establish an Orchestrator

Orchestrating containers is the best way to accomplish complex tasks. Orchestration platforms can allow for consistent automation for many tasks handled manually and such platforms have costs in terms of complexity and support. Kubernetes, a popular orchestrator, can be provided by many cloud vendors as well as on-premise infrastructure software vendors such as VMWare or Red Hat. The cost and maintenance of these infrastructures should be heavily weighed. They often require a high amount of care and feeding. Once able to accomplish more complex orchestration, organizations will find scaling a deployed application, internal build or quality control process, or externally facing service to be easier to manage in the long term. Effective orchestration mechanisms mean that organizations can automate scaling as part of the infrastructure as code stack. Strong automation leads to ease of updates with a collection of containers working in tandem and new assets being spun up on demand from existing configurations. Configuration also can allow management of and network-level coordination between containers.

Set Policy (and Infrastructure) to Encourage Adoption

Individuals' behaviors are guided (implicitly or explicitly) by underlying structures. Adoption must start with a purpose, whether that is a service or part of a larger project. Investment is needed during spin up to ensure proper experience is gained by project members. The chosen project must also have a clearly defined success metric. There will need to be some level of acceptance that development staff will have to make. If organizations want their developers and engineers to adopt and use containers, they must consider the enabling incentives and infrastructure. One way of spurring adoption is to set organizational policies and/or requirements that promote examining and using containers for new projects or refactors. Organizations might also foster conversations between employees at other organizations that have successfully transitioned to containers to understand pain points encountered and key lessons learned. There is also a need for organizations to understand the business model implications of switching to containers. Most importantly for leaders, remember that change is hard and takes time. Making time to listen to concerns, integrating ideas into strategic plans, and transparent decision making can all help to improve change

management.

Final Thoughts

While "microservices" is a trending topic in software today, making the switch is non-trivial. Having an idea of how containers and microservices are related, coupled with an understanding of the strengths and weaknesses of a containerized architecture, can help you to make informed decisions about how software is deployed and operated and maintained in your computing environments. Even though adopting containers may involve getting past individual, team, and organizational inertia, containers have the potential to tremendously simplify debugging, development, and deployment processes.

Some questions to consider as you adopt a containerized workflow:

- *What paradigms will we follow when building and deploying containers?*
- *How will we provide guidance on container creation?*
- *How will we keep each container as optimized as possible?*
- *What strategies will support long-term storage needs?*
- *How might we build from small and functional base images?*
- *What guidelines are needed to ensure that projects are easily rebuilt?*
- *What processes are needed to keep images up to date?*
- *What are you going to do to scan your images before build and deployment?*

ADDITIONAL RESOURCES

Read the SEI Blog post *Virtualization via Virtual Machines*.

WRITTEN BY



Andrew O. Mellinger

DIGITAL LIBRARY PUBLICATIONS

SEND A MESSAGE



William Nichols

AUTHOR PAGE

SEND A MESSAGE



Jay Palat

DIGITAL LIBRARY PUBLICATIONS

SEND A MESSAGE

MORE BY THE AUTHORS

The Myth of Machine Learning Non-Reproducibility and Randomness for Acquisitions and Testing, Evaluation, Verification, and Validation

JANUARY 13, 2025 • BY ANDREW O. MELLINGER, DANIEL JUSTICE, MARISSA CONNOR, SHANNON GALLAGHER, TYLER BROOKS

Measuring AI Accuracy with the AI Robustness (AIR) Tool

SEPTEMBER 30, 2024 • BY MICHAEL D. KONRAD, NICHOLAS TESTA, LINDA PARKER GATES, CRISANNE NOLAN, DAVID JAMES SHEPARD, JULIE B. COHEN, ANDREW O. MELLINGER, SUZANNE MILLER, MELISSA LUDWICK

Cost-Effective AI Infrastructure: 5 Lessons Learned

MAY 13, 2024 • BY WILLIAM NICHOLS, BRYAN BROWN

The Need for Quantum Software Architecture

AUGUST 28, 2023 • BY DANIEL JUSTICE, ANDREW O. MELLINGER

Play it Again Sam! or How I Learned to Love Large Language Models

FEBRUARY 13, 2023 • BY JAY PALAT

Get updates on our latest work.

Each week, our researchers write about the latest in software engineering, cybersecurity and artificial intelligence. Sign up to get the latest post sent to your inbox the day it's published.

Subscribe

Get our RSS feed

Report a Vulnerability to CERT/CC | Subscribe to SEI Bulletin | Request Permission to Use SEI Materials

Carnegie Mellon University
Software Engineering Institute
4500 Fifth Avenue
Pittsburgh, PA 15213-2612
412-268-5800

Contact Us

© 2025 Carnegie Mellon University