docker

Cracking the code: Effectively managing all of those applications

You Need a Container-First Approach to Software Development

Cracking the code: Effectively managing all of those applications

You Need a Container-First Approach to Software Development

The widespread adoption of microservices, containerization, and orchestration for application development means container technology is virtually synonymous with the cloud-native software architecture. While containers are essential to cloud-native, they are equally beneficial to virtually all aspects of how we package, develop, distribute, manage, operate, and maintain software. This is regardless of whether we are containerizing large applications and their dependencies or containerizing reusable, elemental software and services. Containers are beneficial to legacy applications, infrastructure, development, software deployment, and operating environments.

A container-first approach advocates for the broad application of containers across infrastructure, systems, applications, DevOps, and software development workflows (often referred to as the software development life cycle, or SDLC), security, platform engineering, and more. Businesses investing in containerization enjoy the benefits of standardization, isolation, repeatability, and consistency across their broader organization's use of software.

Putting this into context, a container-first approach is analogous to the API-first approach in modern software design. Just as APIs decouple integrations from underlying databases, containers decouple applications from underlying infrastructure. Using APIs gives users flexibility to switch out languages and databases and containers let developers use the same orchestration tools and swap out "what's in the box", similarly providing re-architecting capabilities behind a common abstraction.

This eBook explores a comprehensive container-first approach that will ultimately help organizations improve development efficiency, scalability, and CI/CD deployments. Further, we examine how broad uses of containers provide standardized interfaces, perform consistently across their use, and aid in securing the application or infrastructure software.

The Standardized "Cargo Container" for Software

Software containers are analogous to the standardized cargo container that revolutionized the mid-20th century shipping industry. Cargo, or shipping containers, brought dramatic efficiency improvements, reduced labor and costs, simplified logistics and increased scalability, which in many ways empowered the globalization of goods transportation and delivery to all parts of the world.

Similar to how a cargo manifest details the contents of a shipping container, a SBOM (software bill of materials) can be attached to a software container to outline all of the installed packages and dependencies. A Dockerfile provides the "packing instructions" on how to create the container image. The resulting Dockerfile, along with the application software and its dependencies, can be version-controlled and tracked. Images can be tagged to help provide an audit trail for changes for one or more variances that can operate in parallel with container images. The Docker image provides consistent and repeatable behavior thanks to the well-defined interface between the container and the underlying infrastructure.

Environment Standardization and Isolation

Most developers, testers, or sysadmins have experienced the 'It worked in my environment' syndrome. Inconsistencies in software, configuration, versions, and other seemingly unrelated elements of the environment introduce change and uncertainty with regard to how software connects to and interacts with other system elements. Errors and variations are introduced due to many factors, from port mapping and configuration changes to patch application and build process variations.

Containers solve this issue by providing an isolated environment for each application or service that behaves the same way in any environment, be it development, testing, or production. This environmental parity helps maintain consistency between development, staging, and production environments, reducing the chances of encountering bugs due to differences in the underlying environments. Developers can use containers on their local machines to immediately become productive by replicating the exact environment in which the application will be tested and run, regardless of the local operating system (OS) or installed libraries. This significantly reduces compatibility issues and improves consistency and reliability across the development process.

The benefits of container isolation extend beyond the elimination of variances and inconsistencies. Each container operates independently of and is isolated from other containers, whether they are multiple instances of the same image, different versions of software operating within their own containers, or software in containers communicating via well-established APIs, ports, or protocols.

The container-first approach benefits from the standardization and consistency containers provide across versions and environments for applications and infrastructure.

Beyond Develop, Build, and Deploy

In a CI/CD pipeline, each code change triggers a process in which the application is built, tested, and prepared for release. Containers are often used at each stage of this pipeline. Instead of needing to pre-configure machines for each type of build and language, these can now be done inside of containers. By leveraging containers, the required maintenance and infrastructure required to run builds is dramatically reduced.

While testing applications, a new container can be spun up for each test run, ensuring the tests run in a clean, consistent environment isolated from external variances every time. Users can run these tests using the exact container image that they just built, which will be the exact same images used in staging and production, giving a greater level of assurance that things are going to work.

Containers and Infrastructure

While defining what is application software and what is infrastructure is imprecise at best, containers provide a boundary that clearly defines and delineates where pieces of software start and stop. Since containers have a light footprint, you can run more applications on less infrastructure. Many companies have seen dramatic reductions in their infrastructure footprint, and therefore costs, by using containers to “do more with less.”

Whether a load balancer is part of an app or the infrastructure becomes less relevant when all components are standardized, isolated, and delivered using containers. Each is created through the same process, which enables predictability and repeatability. And those containers are delivered in the same standardized manner with well-defined contents, interfaces, and isolated dependencies. Additionally, managing different versions and different software dependencies is standardized around the container.

Historically, software delivery, installation, and configuration centered around the host. The environment, software dependencies and the drift in variances between hosts are problems containers easily solve. The same container, carrying its dependencies, and operating instructions, can now easily be deployed across various environments, including physical or virtual hosts, without those host-centric concerns. This is true whether we are deploying a container in development or production and also holds true across different operating environments.

Trading Cognitive Load for Innovation Focus

Changing business, competitive, and operating conditions increasingly put a premium on software developer productivity. Expand the same thinking to the full spectrum of software-related functions and the ways in which operations, cloud infrastructure, platform engineering, and more hinder or aid in achieving business outcomes quickly becomes apparent.

Using consistent containerized environments — rather than setting up each developer's laptop, configuring all the different versions of software and tools, and working through out-of-date dependencies and READMEs — quickly increases productivity. Now, developers only need to clone the code repository locally and run a single command to quickly start focusing on their actual work.

Onboarding developers is significantly faster and code is created and delivered in a consistent environment from development into test through to production.

Development teams frequently must work on multiple applications and across different versions of the same applications. Tearing down one project environment to spin up or debug another is a productivity killer, at least in a pre-container environment. In containerized development, test, and delivery environments, developers can fluidly shift between environments and technical dependencies from Project A to B.

Other parts of the development workflow that benefit significantly from containerization include CI/CD pipelines - the heart of DevOps workflows. Developers and DevOps engineers struggle to find the time and resources required to set up and maintain build machines across multiple environments (Java, Ruby, Node, etc.). They waste valuable time managing the entire fleet of build processes for different environments.

With containers, developers no longer need pre-configured environments designed to run specific jobs, they only need the ability to run containers. Each job can start a container specifically designed to perform the required task. This dramatically reduces the CI/CD environment's footprint, allowing the same resources to be reused for any number of different workloads.

There is a complementary relationship between containers and emerging disciplines such as platform engineering. Whether creating cloud-based infrastructure and platforms or implementing automation for deployment, scaling, and monitoring, platform engineering teams benefit by using the same container technology and processes to deliver their respective components of the software stack layer.

Doing so also eases software delivery from test and development onto the platforms created and maintained by platform engineering, eliminating outdated dependencies and environment inconsistencies.

Securing The Software Supply Chain

Containers can play a vital role in securing the software supply chain. Once software is wrapped up in a container image, it's easy to validate that you're getting exactly what is expected wherever we transport that container. It is equally easy to detect tampering. A SBOM can be attached to a container image at build-time to accurately reflect how the image was created. By ingesting the SBOM into a system of record, we can continuously monitor for both current and new vulnerabilities for any installed packages or dependencies. With the addition of signed workflows, we can validate who or where a container came from, giving a clear attribution path for its contents.

Container technology also helps security professionals implement aspects of isolation, micro-segmentation, and other techniques critical to cybersecurity and the zero-trust framework. While isolation and segmentation approaches are not new, they are increasingly used in smaller and smaller components of networks, infrastructure, and applications to reduce attack surfaces and isolate inevitable breaches. Containers add a degree of isolation from external environment factors and provide narrow interfaces to secure and minimize external dependencies. While containers do not fully implement isolation and segmentation, they will greatly increase security through the use of trusted base images, running containers using least privileges, performing vulnerability scanning, and utilizing runtime protection technologies.

Summary

Containerization and a container-first software development strategy delivers numerous benefits across development life cycles, operational tools, infrastructure and platforms, applications, and security. Standardization, consistent performance, and security are just three of these benefits that your business can take advantage of to reduce costs and boost developer productivity so you can deliver solutions to your customers faster.

You should invest in containers if you want:

Consistent & reliable software performance: Standardization reduces issues that slow down collaboration, like "It works on my machine," and ensures applications will perform across the development process.

Reduction in bugs: Environmental parity helps to maintain consistency between development, staging, and production environments. This reduces the chances of encountering bugs due to differences in underlying environments.

Faster developer onboarding: Developers can use containers on their local machines to immediately become productive by replicating the exact environment in which the application will be tested and run, regardless of the local operating system (OS) or installed libraries.

Easier transition to platform engineering practices: Container-first emphasizes the same consistency and standardization values as platform engineering teams.

Enhanced application security: Besides working in tandem with SBOMs to secure software supply chains, containers also help security professionals implement aspects of isolation, micro-segmentation and other zero-trust techniques and reduce the defensible attack surface.

Most organizations are already using Docker or similar technologies, positioning them to expand standardization, productivity and security across their business and improve the efficiency of their software development practices.

Use Docker to optimize your development process.
Get started today!