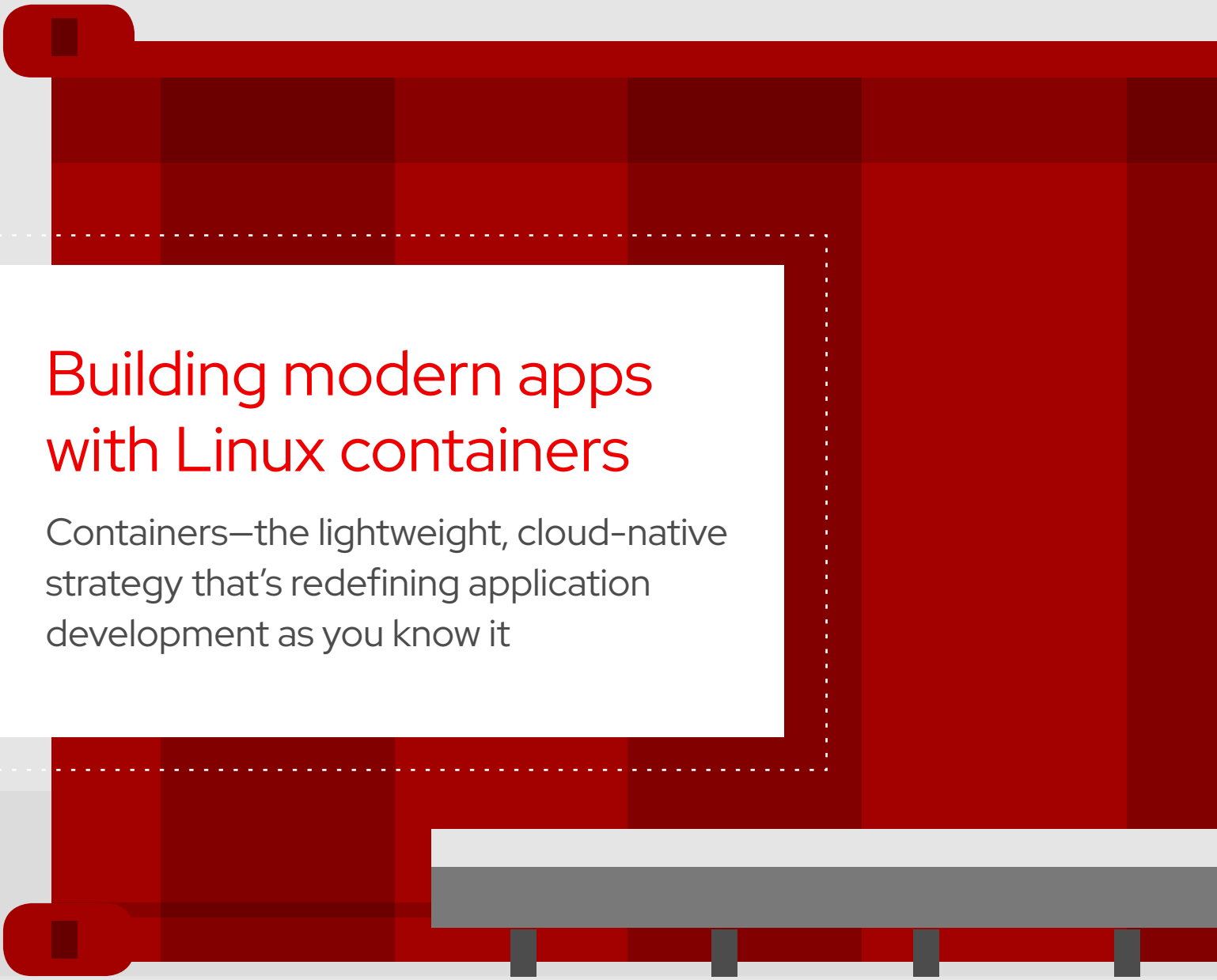
Red Hat



Building modern apps with Linux containers

Containers—the lightweight, cloud-native strategy that's redefining application development as you know it

See what's inside

Introduction

What's inside the box?

The days of monolithic application development are over. And while "digital transformation" may sound like a fancy buzzword that ranks alongside blockchain, agile, and cloud, it's not all hype. With transformation comes unprecedented levels of speed, consistency, and efficiency that are fundamentally changing the way developers do their jobs.

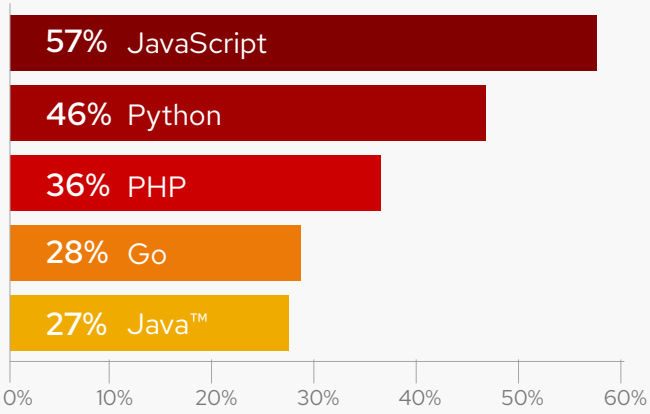
Yes, that includes you.

Much of the burden and pressure to deliver on the promises made by the business falls on the shoulders of IT developers. In a world where users demand new applications, features, and updates across all their devices in real time, container technology is your secret weapon. Containers let you work smarter by creating consistent development environments to rapidly develop and deliver cloud-native applications that can run anywhere. With containers, you can also deliver microservices that eliminate lengthy regression testing cycles, deploy without disruption, and provide a mechanism for patching or rolling back code on a feature-by-feature basis.

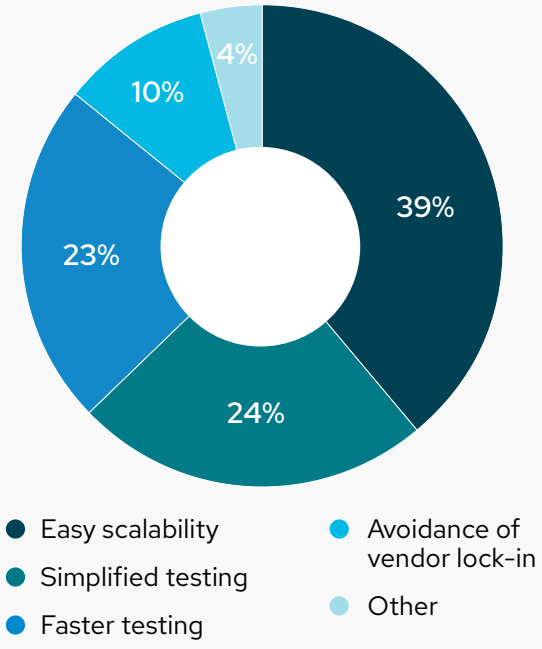
Learn how to deploy containerized applications.
Register for the training course ▸

Although containers are a relatively new technology, developers worldwide are embracing them as valuable and essential tools for modern development. Rapidly maturing container technology and advancements in container orchestration and management tools have allowed container adoption to reach a tipping point, with 49% of developers now using them, according to a 2018 Digital Ocean report.[1]

LANGUAGES USED TO DEVELOP ON CONTAINERS[1]



BENEFITS OF CONTAINERS[1]



1 *Digital Ocean. Currents. June 2018.*

Looking back

The journey is the destination

"Johannes Gutenberg's printing press created a surge in demand for spectacles, as the new practice of reading made Europeans across the continent suddenly realize that they were farsighted; the market demand for spectacles encouraged a growing number of people to produce and experiment with lenses, which led to the invention of the microscope, which shortly thereafter enabled us to perceive that our bodies were made up of microscopic cells."

Steven Johnson
How We Got to Now: Six Innovations That Made the Modern World

Like many other great technological advancements, containers are the culmination of several concepts and technologies that evolved over time. Back in the 1970s and '80s, we started breaking down code into objects, toying with the ideas of abstraction and isolation. We quickly learned that securing parts of our code, while exposing others, helped us maintain greater control over processing and data handling while allowing us the flexibility to integrate with adjacent systems. These advancements led to further layering and abstracting processes and components, and we evolved toward multi-tiered environments and service-oriented architectures (SOAs) that further isolated data layers away from business code and user interfaces. Throughout this time, we also evolved our methodologies from monolithic waterfall development, through the software development life cycle (SDLC), into an era of agile development and scrum, and finally, to DevOps and continuous delivery.

From a business perspective, all these advancements served to lighten the development process to produce code faster, cheaper, and better. From a developer's standpoint, each new iteration of innovation shortened the development cycle and demanded more rigorous adherence to patterns and methodologies. While every advancement was a step in the right direction, the advent of containers brought everything together and provided us with the solution that enabled true flexibility, interoperability, and portability for our code.

Looking specifically at the evolution of Linux® containers, you can see how impactful they've become over the last 15-20 years.

2000
Container technology first appeared as FreeBSD jails, allowing servers to partition into subsystems where a developer could work and not compromise the entire system.

2001
The container concept found its way into Linux via the Linux-VServer project with the goal of running several general-purpose Linux servers on a single box.

2007
Additional technologies combined to make containerization a reality. Specifically, control groups (cgroups), systemd, and kernel/user namespaces added overall control and virtualization capabilities that served as the framework for separating environments.

2008
Docker came onto the scene with its container technology, which added even more concepts and tooling that allowed users to quickly build new layered containers and share them with others.

2012
Microservice architecture evolved as a specialization and refinement of SOA used to build flexible, independent, and deployable software.[4]

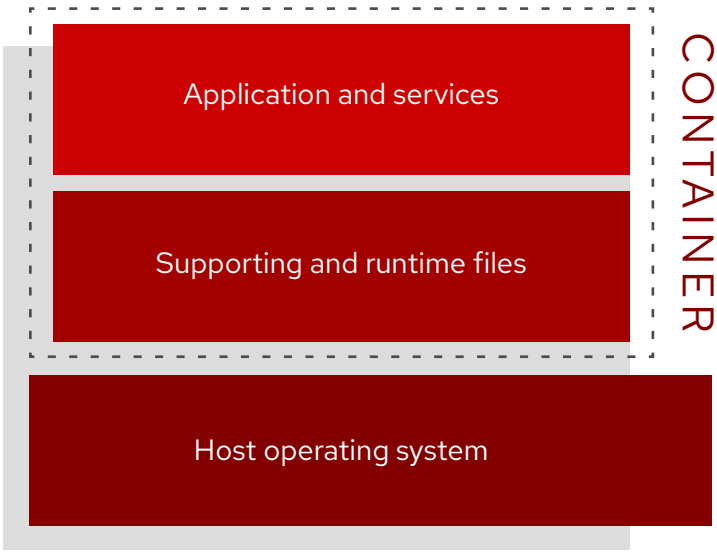
2015
Kubernetes was released as an open source container orchestration system for automating application deployment, scaling, and management.

4 *Fowler, Martin. "Microservices: a definition of this new architectural term." March 2014.*

Chapter 1: The basics

What is a Linux container?

At its core, a container is simply a new way to abstract one or more processes from the rest of a system. Containers lighten the load, so to speak, allowing you to work on small subsets of code without impacting the overall runtime environment. They also provide a standard way to package and isolate application code, configurations, and dependencies into a single object.



Better, faster, cheaper—you can have all three

The real value of containers is portability. With all the files necessary to run a containerized application, feature, or component in a single, distinct image, Linux containers provide consistency and predictability as they move from development to testing, and finally to production. This makes container deployment much quicker, more reliable, and less expensive than with monolithic development pipelines that rely on replicated development, testing, and production environments. And, you can write and develop containerized code once and deploy it to multiple operating environments without the need for additional development time, lengthy testing cycles, or specific deployment processes.

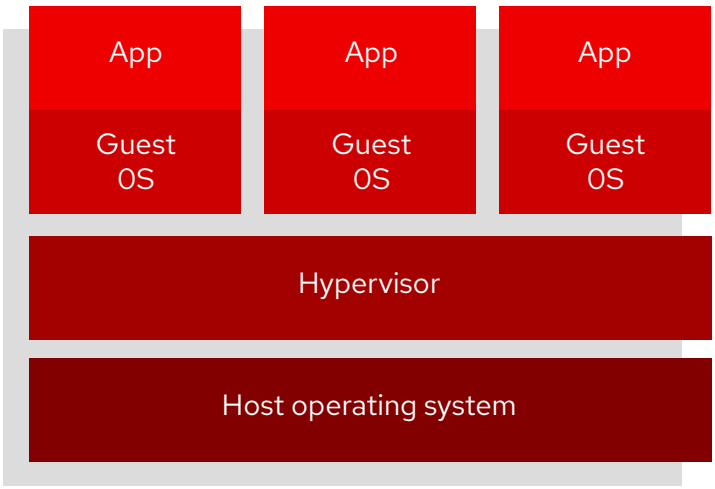
Containers share an operating system (OS) installed on the server and run as resource-isolated processes, ensuring quick, reliable, and consistent deployments, regardless of the environment.

> The real value of containers is **portability**.

Virtualization vs. containers

While containers and virtualization seem similar, they are in fact quite different.

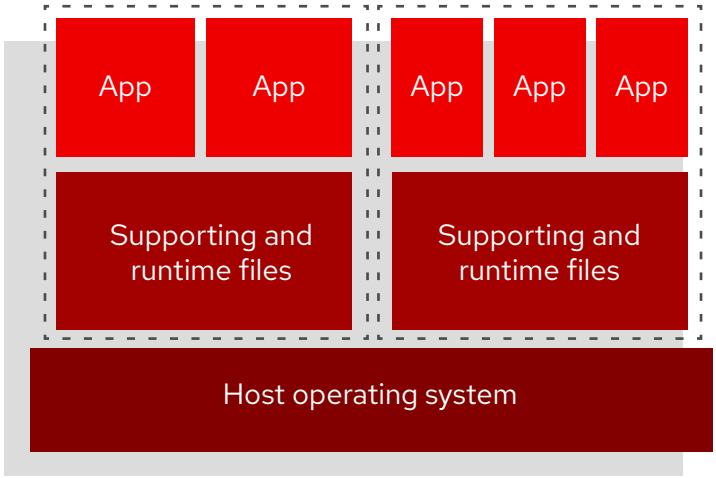
VIRTUALIZATION



Virtualization lets you run multiple separate computers on a single piece of hardware. The OSs and their applications share hardware resources from a single host server. Each virtual machine (VM) requires its own underlying OS. A hypervisor creates and runs the VMs.

VS.

CONTAINERS



A container isolates the application processes from the rest of the system and includes only what's necessary to run a specific application, including some OS files, supporting programs and libraries, and system resources. Containers are lightweight, start much faster than VMs, and use a fraction of the memory of VMs.

Containers + Microservices = The ultimate power couple

Now that you have a general understanding of what a container is, let's explore **the most important use of containers for developers—microservices development**.

Microservices are small, self-contained, single-function applications that communicate through application programming interfaces (APIs). A core principle of microservices architecture is that each microservice handles one, and only one, function and provides a well-defined API that allows communication into and out of the code. Microservices are the ultimate encapsulation mechanism. Because microservices are fully self-contained, making a change to a microservice introduces less risk to the overall application than changing code in a monolithic code structure. Also, microservices are faster and more agile than traditional applications because of their self-contained nature and their independent use of system resources.

Containers and microservices can exist independently and often do. Individually, they serve different purposes. When they are implemented together, they are a powerful tool for creating portable, cloud-native applications.

Think of containers as an enabling technology for microservices. Containers are abstracted away from the host OS and contain all the supporting and runtime files they need to execute the code contained within them. When you deploy a container, it will run regardless of the underlying OS. Containers are portable and can be deployed across multiple clouds and devices without rebuilding or additional testing.

Microservices development with containerized deployment is becoming the norm for enterprise development. The architecture offers unprecedented levels of agility, speed, and resource efficiency for many tasks that you, as a developer, work on daily. Using containers and microservices in a DevOps environment allows developers to deploy each service independently. This practice eliminates the need to merge code changes, greatly improves testing, and helps with fault isolation in both testing and production. Also, parallel developer teams can work on the loosely coupled applications and choose the technology stack best suited for their requirements without enforcing those requirements on other teams.

Reduce, reuse, recycle with container images

If there's one thing developers love, it's the ability to reuse code. With containers, you can create base container images, add them to a repository, and pull them down whenever you're ready to start a new project. Because a base container image is an unchangeable, static file that does not include any executables, it's both consistent and portable, with the ability to run an isolated process on any infrastructure. The image consists of system libraries, system tools, and other platform settings that your applications need to run.

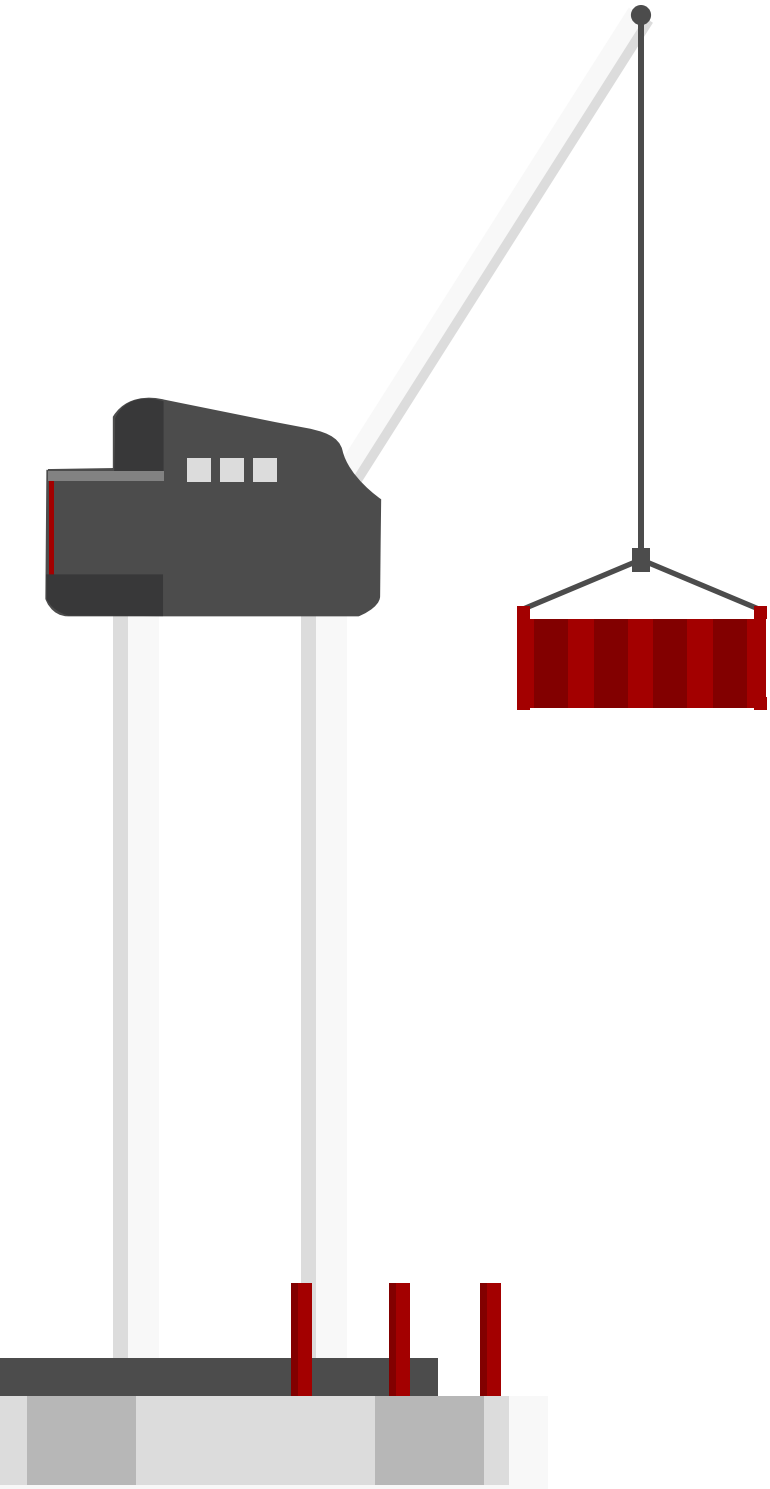
You can create your own container images, or you can choose from available public repositories. Many software vendors, including Red Hat and Microsoft, create publicly available images of their products.

Microservices and containers are a powerful combination, especially when integrated into a DevOps environment.

TOP 5 BENEFITS OF CONTAINER IMAGES

1. Automate build and deployment processes
2. Tag for ease of location and download
3. Simplify vulnerability scanning
4. Enforce coding standards and policies
5. Save time and encourage reuse

Chapter 2: Improve your productivity



Containers and you

Containers provide a streamlined approach to build, test, deploy, and redeploy applications, which makes them a great choice for everything from simple projects to mission-critical applications. There are several benefits to containers for all of IT and the business, but let's take a moment to focus on how they can enhance your day-to-day success as a developer.

Be flexible

When it comes to development cycles, there's never enough time. The business has its roadmap, your test team has a backlog of nagging defects, operations teams struggle to keep systems patched and secure, and your customers have their own ideas of what's important. Unfortunately, there are only so many hours in the day, and you have only so many developers working on any given project. When your applications run on a foundation of containers, this allows you, the business, and the operations teams to respond in real time to change logs, defects, security concerns, patch levels, and new customer feature requests. And because you can deploy a single container without impacting the rest of your application architecture, you can make changes on the fly without waiting on adjacent teams to finish their development cycles.

Think bigger

Containers are lightweight and can often start in milliseconds. They do not require an OS boot, and they load only the dependencies that they need. Creating, replicating, or destroying containers can also be accomplished in a matter of seconds. When your customers create seasonal loads, or the business decides to add new teams of users, your operations team can respond by adding resources that your containers will naturally use as needed. Additionally, you can scale out containerized applications to new users—even globally—by simply deploying to new clouds.

Work smarter

Containers allow you to focus on your application logic without worrying about specific OS versions and application configurations. After all, that's why you have IT operations, right? Additionally, containers let you package your code, dependencies, and configurations into a single, encapsulated piece that can be easily version-controlled, tested, and deployed. Combining containers with a service-based architecture also makes it easier to support, test, and enhance your applications.

Achieve (actual) standardization

We all know that a standardized set of environments from development through production is somewhat of a unicorn. Well, get ready for some magic. One of the most powerful benefits of containers is that they standardize local, development, test, QA, and production environments. With this level of predictability, you'll be able to spin up isolated environments and spend less time debugging and diagnosing issues caused by differences in patch levels, operating systems, and applications. Instead, you'll spend your time developing and shipping new functionality. Additionally, new developers on the team can start working much faster without having to spend time installing and configuring their local development environments. They can simply pull down a container image from a repository and start coding.

Write once, run anywhere

The phrase "write once, run anywhere" has historically come with caveats, the biggest of which involved the OS where you developed your application. You could run your application anywhere—if the target system ran the same OS. With containers, your code really can run in virtually any environment, regardless of where it was developed. When you develop inside a container, you can deploy to Linux, Windows, and Mac OS across bare metal, virtual machines, public clouds, private clouds, and hybrid environments. Also, the widespread adoption of the Docker open source project provides a stable way to automate the deployment of applications inside containers, freeing up your time to jump into your next development cycle.

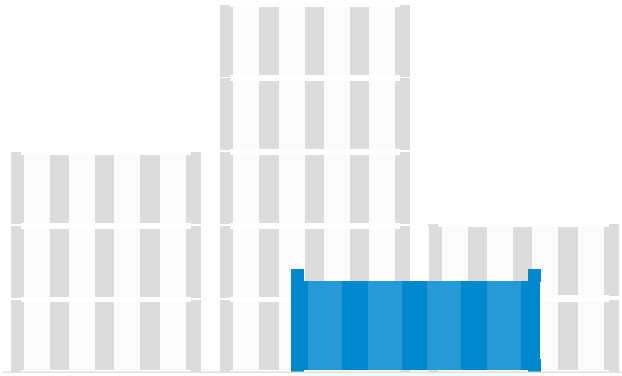
Deliver exceptional application quality

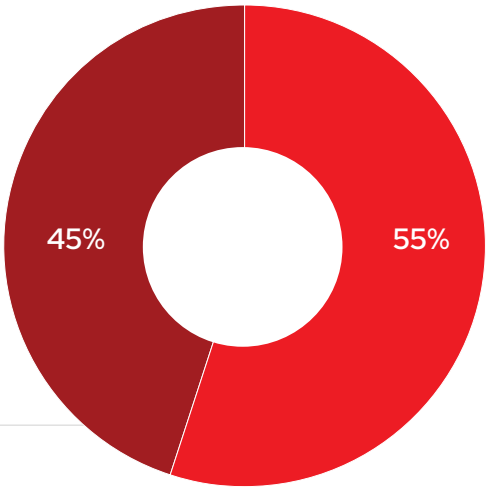
Containers make testing and troubleshooting faster and easier for everyone involved. For you, the ability to contain development to a single application feature means less chance of introducing inadvertent errors in adjacent code. Once you've put your container in version control, QA teams can then test directly against your container image instead of pulling and building the entire application from a continuous integration (CI) server, which simplifies testing and saves time.

Containers can also help QA and support teams identify the root cause of an application problem faster. Because of the API-driven, loose coupling of code, there are fewer interdependencies to sort through when something unexpected happens. Engineers can get to the bottom of an issue quickly and pinpoint the exact container that caused the error. Once the problem is identified, containers make it easy to revert changes. Single containers can be rolled back without affecting the rest of the application. When the container comes back to your desk, you'll spend less time troubleshooting and fixing the issue and more time developing features that your business leaders and customers want. With containers, same-day break-fix becomes a reality, which keeps your customers and your manager happy.

According to a recent IDC study,

55% of IT leaders deployed containers on-site, whereas 45% deployed them in a public cloud.[5]





5 *IDC Study. "Container Infrastructure Market Assessment: Bridging Legacy and Cloud-Native Architectures – User Survey Summary." March 2018.*

Use your favorite tools and languages

As a developer, you know how liberating it can be when you get to choose the tools or languages you want to use on a project. Containers give you the flexibility to do that. Some application runtimes are better suited for certain types of workloads or architectures. For example, vert.x encourages distributed reactive architectures, which can be great for responsive, real-time apps such as the ones required for Internet of Things (IoT) devices. While you could build this type of app in another language, you'd have to reinvent what vert.x provides natively. Because containers are truly agnostic, giving you flexibility and control, you can choose the tool that works for one app without forcing entire application teams to adopt the same tools or languages for their projects.

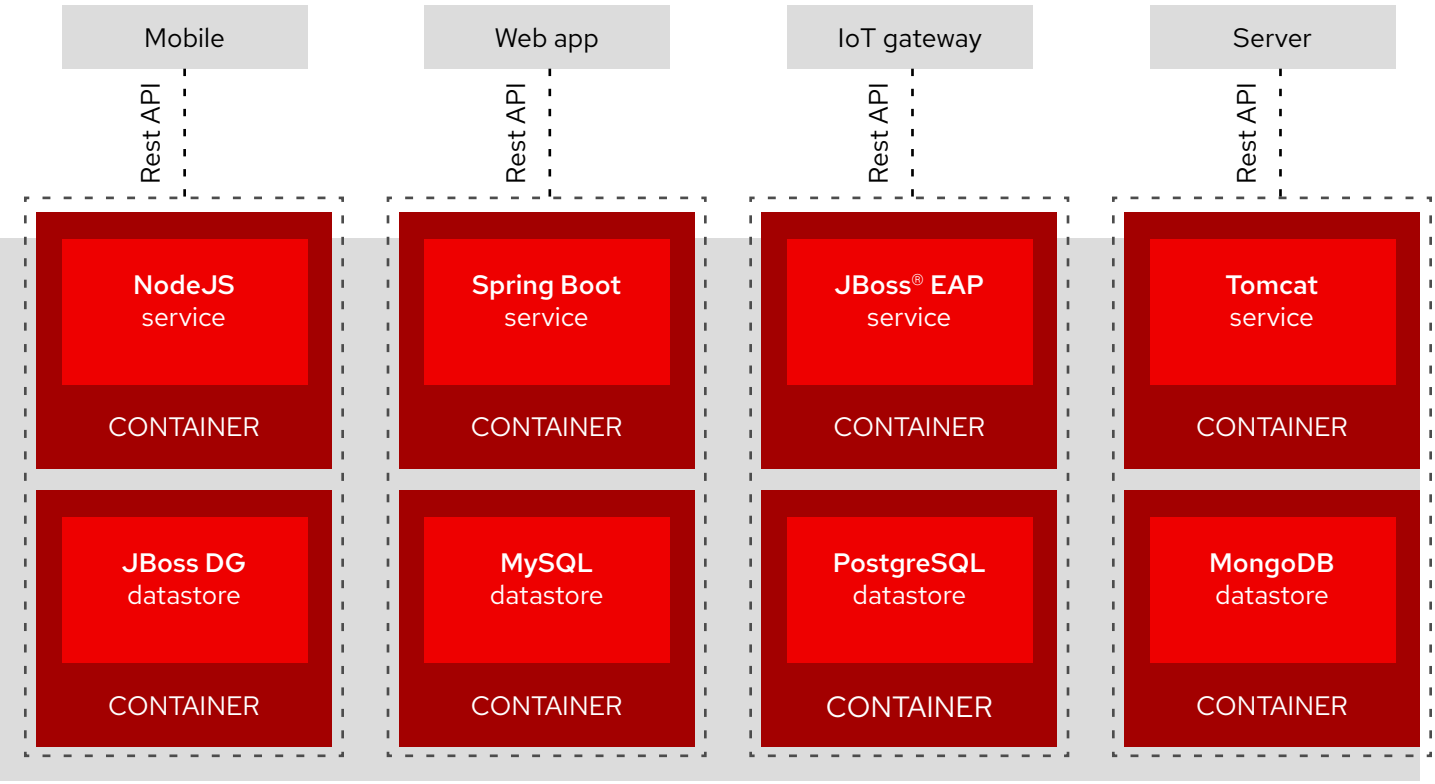


Figure 1: Improve development flexibility and simplify deployment with containers

Increase your personal value

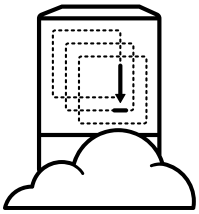
Let's get real on this last benefit. When it comes to your career, if you don't evolve your skills with advancing technology, you end up getting left behind—or worse, getting stuck working on legacy apps until you retire. Opportunities for growth and development are critical to your success. Plus, developers typically enjoy learning new technology. It's a perk of the job.

Microservices, containers, and container orchestration technologies offer an attractive combination of being technically challenging while also being in high demand. Based on a 2018 Red Hat® survey, container usage is expected to increase by 89% in the next two years and will continue the trend of accelerated mass adoption into the future.[6] When you master container-based microservices development, you give yourself new opportunities to grow your career and deliver value to your company.

6 *Dawson, Margaret. "Red Hat Global Customer Tech Outlook 2019: Automation, cloud & security lead funding priorities." Red Hat Blog. Dec. 18, 2018.*

Chapter 3: Containers in the wild

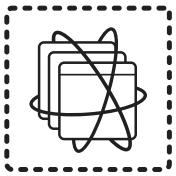
If you haven't already, you'll most likely be asked to deliver your code in containers in the near future. As businesses undergo digital transformation exercises, leadership teams learn how containers allow IT organizations to better use resources, both human and computer, to reduce costs, improve efficiency, and deliver value. As you're planning your projects, keep these use cases in mind, and bring the idea of containers to the table during your next team meeting.



Lift and shift

While there is much talk about cloud-native applications, most enterprise applications still tend to be monolithic. Lift and shift refers to the process of migrating existing applications to a more modern, cloud-native architecture. Existing applications are simply migrated to the cloud with the least amount of code changes as possible. Typically, this process is brief and preferred by business leaders because they can use their existing mission-critical applications to drive better value and performance without heavily investing in a massive redesign.

While lift and shift is simple and fast, it is typically seen as a stopgap solution because it doesn't fully take advantage of cloud-specific tools. If you lift and shift one of your applications, you will want to re-evaluate at some point to make sure it is still delivering value and to make recommendations for refactoring or replacement.



Refactor

Refactoring has been a staple of application modernization since the first application became obsolete. In this case, we're specifically talking about refactoring an application for containers. While refactoring is much more intensive than a lift-and-shift migration, it allows the application to take full advantage of the benefits of containerized and cloud-native environments.

In the past, refactoring efforts were often overlooked due to time commitment and complexity. With containers, refactoring becomes an incremental process. To start, you modify only the most critical parts of an application and lift and shift the rest. Then, you can refactor the rest over time, allowing you to keep business running while making additional improvements as your schedule allows. You'll often see this process called "strangling the monolith" in books and blogs related to microservices.

Of course, you can still refactor the old way, where you modify the complete application before releasing it for use. There are advantages and disadvantages to both strategies. The leading benefit of a gradual refactor is time. The drawback is a tendency to never fully migrate to a cloud-native platform, which introduces management complexity and risk.

New application development

As you've seen, containers are a powerful and flexible tool for developing new cloud-native applications. Developing from scratch allows you to unlock the full benefits of containers. Containers provide an ideal platform for microservices, hybrid applications, automation of repetitive jobs and tasks, and future-forward applications such as artificial intelligence (AI) and machine learning (ML).

Microservices

Containers go hand in hand with microservices architectures. Distributed applications and microservices can be easily isolated, deployed, and scaled using individual containers as building blocks.

Hybrid applications

Containers let you standardize how code is deployed, making it easy to build workflows for applications that run between on-premise and cloud environments.

Repetitive jobs and tasks

Repetitive jobs and tasks, such as batch processing and extract, transform, and load (ETL) jobs, can easily be developed on containers to start jobs quickly. They can then be automated for ease of operation and scaled dynamically to meet demand.

Artificial intelligence and machine learning

Containers offer a new way to build and deploy portable cloud applications that incorporate AI and ML. These containers can quickly scale the AI and ML models to accommodate the processing needs of advanced training algorithms. Additionally, AI and ML containers can be deployed close to data sources to improve performance and shorten training cycles.

Chapter 4: Considerations and challenges

Make life easier—don't skip this chapter

We have talked a lot about the impact and power of containers, but like any other technology, they present challenges that you'll have to consider. These challenges can become critical if not properly addressed, particularly as the number of enterprise containers deployed into production increases. You'll want to keep an eye on the continued evolution of technology and how well your organization is adopting, managing, and maintaining your container-based architecture.

Things to consider before you get started ...

By this point, you probably have a few ideas in mind for starting a new project or refactoring a legacy application. There is no better way to gain experience and knowledge with containers than to get in there and start coding.

However, you should consider a few things before you start. Creating a containerized application presents some challenges that don't exist in a traditional environment. The following information is a summary of considerations that could impact your development efforts. Many of the ideas in this section were inspired by Project Atomic. After you finish reading this e-book, you'll want to check it out.

Determine your data strategy

While most cloud-native applications are stateless, many applications require persistent data storage. Containers have immutable storage—your data will be lost once the container spins down. You'll need to keep this in mind and design your applications in a way that will allow data to persist regardless of the state of the container.

If your application data needs to be preserved after a container terminates, you can assign a storage volume to your container. These assigned volumes will persist regardless of the state of the container. Developers should make sure their applications are designed for writing to a shared datastore. For enterprise applications, tools such as Red Hat OpenShift® Container Storage provide software-defined storage specifically built for container environments. OpenShift Container Storage gives your data a permanent place to live—even when containers spin up and down—and easily scales across bare-metal, virtual, container, and cloud deployments to further improve the portability of your containers without constraining them to your storage architecture.

Get your containers communicating

Distributed application components need to communicate with one another to accomplish workflows. Container technologies encourage developers to make interconnection points explicit and provide a mechanism to communicate between containers using APIs. That's great for container-to-container communication, but what about your databases?

Traditional databases typically communicate using a socket over a network. Because a container namespace changes as its state changes, this type of legacy communication mechanism doesn't work. In a containerized application, you'll need a container orchestration platform, such as Kubernetes or Red Hat OpenShift Container Platform, that facilitates network communication between your containers, databases, and other network resources.

Synchronize and standardize

Some containerized applications require the host and container to be synchronized on certain attributes for uniform behavior. For example, consider a centralized log server that receives data from multiple containers deployed across various geographical regions. The log timestamps and information would be almost useless to the operations team if each container reported a different time than the host without also reporting the server location. By synchronizing and standardizing on a set of environmental attributes, you can ensure that the data communicated back to the central datastore is accurate, relevant, and usable.

Capture all the logs

Every application should log appropriate information that makes troubleshooting easier. If your application logs actions, errors, and warnings to some sort of log mechanism, you will want to consider how to allow users to obtain, review, and possibly retain those logs. Because your containers are separated by namespace and cannot directly access bare-metal components, including local hard drives, you'll need to rethink your logging strategy.

The easiest way to collect logs is to use a tool that's built for the job. Container orchestration platforms such as OpenShift Container Storage automatically collect container log data. You can then save that data to persistent storage or follow the recommended guideline of sending logs to standard out or standard error so they can be viewed on the central management console.

Enhance security

Storing sensitive data, including access credentials, is essential for a containerized application so that containers can communicate without repeatedly challenging the user. However, storing credentials can be tricky and can open up your application to potential security risks. The most common way to pass sensitive data in a container is through environment variables that are not exposed publicly. Container orchestration platforms such as Kubernetes and OpenShift Container Platform provide native mechanisms to secure environment variables and pass sensitive data across a containerized application.

Challenges

Keep ahead of evolving technology

The evolution and expansion of the container ecosystem are moving extremely fast. This ecosystem includes tools to help create, deploy, configure, automate, and manage containers. The open source projects that support this ecosystem are extremely active. While this is an encouraging sign about the future of the technology, it can be difficult to maintain the skills needed to deliver applications using these solutions. As a developer, you'll want to stay current with ongoing open source projects, enroll in training courses, and keep your skills sharp so you can continue evolving at or ahead of the pace of container technology.

Embrace DevOps culture

As was stated at the beginning of this e-book, containers are fundamentally changing the way applications are developed. In response, application teams must adapt by shifting their processes and culture. To effectively use containers, organizations must have a healthy DevOps culture without the traditional separation that formed under the monolithic code model. To create a fast, reliable, consistent, and security-focused workflow from development to deployment, organizations must follow and automate DevOps principles. As a developer, you'll become part of the overall process from development, through deployment, and beyond—to customer satisfaction. Your new role includes taking responsibility for your containers and ensuring that any application logic you create runs in production. With the ability to release fixes in real time, you'll be expected to quickly debug, fix, and deploy code to keep your part of the application working at its optimum potential.

Stay on top of security

Security is always a challenge for application developers, regardless of whether you are working with containers. While containers offer the security benefit of application isolation, the proliferation of containers in an organization poses a new breed of security risk.

Containers typically require applications to be broken into smaller microservices, which results in increased data traffic and complex access control rules. As the number of containers increases, so does the potential to create loose access controls between containers. Without proper adherence to security and access protocols, you could introduce vulnerabilities into your production environment. Also, many organizations use container image repositories, but they need to verify that those images meet their organization's security and compliance requirements. Be sure you understand the risks involved with containers, and always adhere to access control policies.

Manage and monitor

Container monitoring can become increasingly difficult as the number of deployed containers rise. When you deploy hybrid cloud solutions, where containers run on both public and private clouds, the management complexity increases considerably. Implementing a consistent datastore to accumulate, analyze, and act on the events being generated from all containers and applications can be challenging. Additionally, once an event has been captured, it can be difficult to pinpoint where errors occur. Fortunately, emerging technologies such as Istio, Prometheus, and Jaeger can help alleviate some of these challenges.

Final thoughts

Containerization is the methodology of choice in just about every modernized enterprise development team, with 91% of cloud developers and development managers now employing containers in some capacity on-premise.[7] Containers allow you to work smarter and more efficiently, with consistent development environments for rapid development and delivery of cloud-native applications that can run anywhere.

The purpose of this e-book is to bring you up to speed on the basics of containers and help you understand the impact and potential they can have for you and your organization. Many resources are available to help you take the knowledge that you have gained here and apply it to your current projects. Soon, you'll separate your apps from your architecture, elevate your organization, and accelerate your own career with the most modern, agile, and future-forward development practice—containerization.

There is no better time to start than now. Good luck on your journey to becoming a cloud-native application developer.

Learn more

Do you want to get a head start learning how to deploy containerized applications? Take a free online course from Red Hat to learn about the concepts of containerization—and see it in action. You'll see how to containerize applications and services, test them using Docker, and deploy them on a Kubernetes cluster using Red Hat OpenShift Container Platform. You will also learn how to build and deploy an application from source code using the Source-to-Image facility of OpenShift Container Platform.

Learn more and register ▸

Explore additional resources

Udemy
Stack Overflow
Docker
AWS
Google
Red Hat

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 integrate new and existing IT applications, develop cloud-native applications, standardize on our industry-leading operating system, and automate, secure, and manage complex environments. Award-winning support, training, and consulting services make Red Hat a trusted adviser to the Fortune 500. As a strategic partner to cloud providers, system integrators, application vendors, customers, and open source communities, Red Hat can help organizations prepare for the digital future.

7 *Evans Data Corporation. "Cloud Development Survey 2017, Volume 2."*