Introduction to CONTAINERS

eGuide

TECHWELL™

3
Why Choose Containerization?

5
Virtualization or Containerization? Choosing the Right Strategy

6
Deploying, Running Applications in Docker Containers

7
How Docker Enables Agile Software Development

12
Demystifying DevOps: A Day in the Life of a DevOps Tester

13
Exploring Containers: Creating a Dockerfile

14
How Testers Can Use Docker to Shift Left and Automate Deployments

15
Additional Resources

TECHWELL™

INTRODUCTION TO CONTAINERS

eGuide

In the continuing search to accelerate development, test, and production cycles, organizations are looking more and more at the benefits of containerization. Containers help software run reliably when moved from one environment to another. Containers effectively bundle an entire runtime environment, rendering differences in OS distributions and underlying infrastructure non-issues. In addition, containers are lightweight, modular, portable, and require significantly less overhead than server or machine virtualization.

Introduced in 2013, Docker has become the industry standard to build and share containerized apps. This eGuide is filled with information and resources to help you explore the world of containers and whether or not they are right for your process.

In this Containers eGuide

Why Choose Containerization?

In recent years, containers have been adopted by many organizations. Why use containerization? What are the advantages that have spurred its adoption? Let's look at some of the factors that govern the choice of containerization.

Virtualization or Containerization? Choosing the Right Strategy

Virtualization and containerization are top two approaches when it comes to enabling scalability, limiting overhead costs, and standardizing software development, deployment, and management across multiple platforms. Careful selection of one strategy over another helps an IT team become more agile and responsive to ever-changing business needs. But how should they decide which better fits their requirements?

Deploying, Running Applications in Docker Containers

Deploying and running applications in Containers is one of the hottest trends in DevOps and IT today. Docker, a containerization platform that lets users easily package, deploy, and manage their applications within containers, is principally responsible for bringing containers to the mainstream. This article provides information about what containers are, their relationship with DevOps, and the benefits derived from container implementation.

How Docker Enables Agile Software Development

Docker has revolutionized how software is packaged, distributed, and deployed, so it's easy to see why it has become the de facto containerization platform. But have you thought about how Docker actually makes software development, testing, delivery, and deployment more agile? Let's look at how Docker inherently supports several of the founding principles of agile software development.

Demystifying DevOps: A Day in the Life of a DevOps Tester

The idea of working as a test specialist on a team using DevOps can feel intimidating. There are at least two technology stacks you need to be familiar with, add a source code repository like Git, a few test frameworks, and a scripting language, and you start to approach a useful skill set. This article outlines a normal day of testing in DevOps for the author.

Exploring Containers: Creating a Dockerfile

Docker is currently the most popular containerization platform. Let's look at what goes into creating a Dockerfile, which could be used to build a runnable Docker image.

How Testers Can Use Docker to Shift Left and Automate Deployments

There are multiple ways Docker can be to help with testing in continuous delivery and integration (CD and CI). This article outlines some of the ways to help facilitate that process.

Additional Resources

3
Why Choose Containerization?

5
Virtualization or Containerization? Choosing the Right Strategy

6
Deploying, Running Applications in Docker Containers

7
How Docker Enables Agile Software Development

12
Demystifying DevOps: A Day in the Life of a DevOps Tester

13
Exploring Containers: Creating a Dockerfile

14
How Testers Can Use Docker to Shift Left and Automate Deployments

15
Additional Resources

Why Choose Containerization?

By Deepak Vohra

In recent years, containers have been adopted by many organizations. While some other implementations of containers are available, Docker is probably the one you've heard of, as it has become the standard containerization platform.

Why use containerization? What are the advantages that have spurred its adoption? Let's look at some of the factors that govern the choice of containerization.

Lightweight Modules

Docker containers are lightweight units of software that run in isolation on a Docker Engine, which runs on an underlying operating system. Each container has its own networking and file system.

A single Docker container does not fully use an underlying operating system (OS), but makes use of a snapshot or a section of the operating system kernel. In this way, multiple Docker containers may run on the same OS.

A single virtual machine, in contrast, uses a complete operating system per application, and consequently, virtual machines may not be able to use the underlying OS fully, resulting in an underutilized OS system kernel. Infrastructure is better utilized with Docker containers.

Packaging and Dependency Encapsulation

A Docker image packages a complete software solution. A Docker container is an instance of a Docker image, which specifies the software to install, dependencies to install, and commands to run.

A single Docker container encapsulates all the dependencies needed to run an application, thus eliminating the need to install the dependencies separately. Software is easier to install and run using containers.

3
Why Choose Containerization?

5
Virtualization or Containerization? Choosing the Right Strategy

6
Deploying, Running Applications in Docker Containers

7
How Docker Enables Agile Software Development

12
Demystifying DevOps: A Day in the Life of a DevOps Tester

13
Exploring Containers: Creating a Dockerfile

14
How Testers Can Use Docker to Shift Left and Automate Deployments

15
Additional Resources

Containerization provides several benefits and has fewer drawbacks compared to other forms of application and services deployment.

Developer Speed and Efficiency

Developer speed and efficiency is greatly improved with containerization. Software is easier to package, install, run, and manage.

Platform Independence

Docker containers are platform-independent and run the same on all supported platforms of Linux and Windows distributions. Isolating the software from the environment makes software portable across local and cloud platforms.

Microservices Architecture

Monolithic architecture has its limitations in terms of its scope, extensibility, and scalability. Modern applications are typically composed of multiple services that need to be deployed and managed independently but still be able to interact with each other, and a monolithic architecture cannot provide independently deployed services.

Instead, a monolithic architecture requires that all applications or services be deployed as a single unit. With dependencies between services, individual services cannot be scaled or otherwise managed independently in a monolithic architecture.

Containerization has made the microservices architecture feasible with its modular structure. Multiple container-based services may be deployed, scaled, and managed independently of each other and still be able to interact.

Agility

Agile software is easier to improve through the employment of continuous refactoring. Containerization, with its support for microservices, decouples the different components of an application and makes it easier to modify or refactor individual components of an application. Code for one service may be iterated, tested, and redeployed independent of another service.

Scalability

Containers and microservices are easier to scale as compared to monolithic software. As standalone containers are rarely used other than for a small-scale, single-service deployment, container orchestration becomes essential. Several tools and platforms are available for container orchestration to manage the containers' provisioning, scheduling, networking, and scaling.

Containerization provides several benefits and has fewer drawbacks compared to other forms of application and services deployment. Virtualizing pieces of an operating system through containerization improves resource utilization, making your development and testing more efficient.

TECHWELL™

3 Why Choose Containerization?
5 Virtualization or Containerization? Choosing the Right Strategy
6 Deploying, Running Applications in Docker Containers
7 How Docker Enables Agile Software Development
12 Demystifying DevOps: A Day in the Life of a DevOps Tester
13 Exploring Containers: Creating a Dockerfile
14 How Testers Can Use Docker to Shift Left and Automate Deployments
15 Additional Resources

Virtualization or Containerization? Choosing the Right Strategy

By Kunal Chauhan

Virtualization and containerization are top two approaches when it comes to enabling scalability, limiting overhead costs, and standardizing software development, deployment, and management across multiple platforms. These abilities are invaluable for both PaaS (platform as a service) products and configuration of reproducible and portable cross-platform development environments.

Careful selection of one strategy over another helps an IT team become more agile and responsive to ever-changing business needs. But how should they decide which better fits their requirements?

If the main requirements are development environment isolation and being able to quickly create different VM (virtual machine) images, it is recommended to use lightweight containers using Docker, an open source development platform. This containerization includes the application and its dependencies.

In such cases, it is better to look for tools that reuse the host operating system kernel with containers. These run as isolated processes on the host operating system and are not tied to any specific infrastructure—essentially, they can run on any computer.

Docker’s containers are faster; use less CPU, RAM, and space; and don’t add an overhead of maintaining a guest operating system in parallel to an application and its libraries. The containers are configured via files called Dockerfiles that communicate with each other on a private network.

If your top requirement is virtual machines with full operating system access and control over each guest operating system library, you could leverage a solution like Vagrant, an open source software product for building and maintaining virtual development environments independent of the host operating system.

Each environment runs on its own VM and is configured by a config file called a Vagrantfile, which contains parameters to enable VM configuration. For instance, you can set up automatic synchronization of folders, redirect port traffic, and run custom commands upon VM provisioning.

Vagrant VMs are also beneficial in cross-development scenarios where development teams are geographically spread out and may not necessarily have access to the same operating systems and versions. Users are able to create a baseline VM image with an application and its dependencies. No matter which operating system is being used, the VM image will be mounted as a separate operating system, providing every team with uniform development environments.

One thing to be watchful for in this approach is that each VM includes not just an application, its codebase, and installed libraries, but the entire guest operating system as well, which could be large and require regular maintenance, patches, updates, and security fixes.

Virtualization and containerization each has its pros and cons that need to be individually weighed against a given team’s composition, the nature of the application under development, and the technology stack utilized. Thinking through the decision will bring the benefits of a scalable and sustainable development solution that works for your specific needs.

3
Why Choose Containerization?

5
Virtualization or Containerization? Choosing the Right Strategy

6
Deploying, Running Applications in Docker Containers

7
How Docker Enables Agile Software Development

12
Demystifying DevOps: A Day in the Life of a DevOps Tester

13
Exploring Containers: Creating a Dockerfile

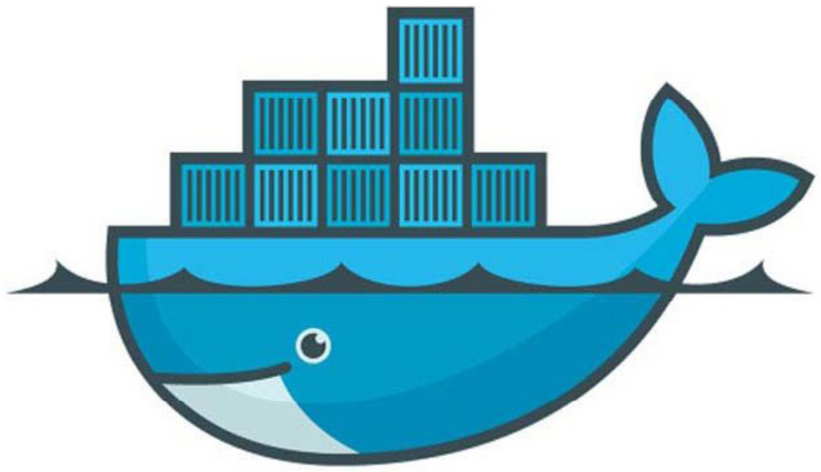
14
How Testers Can Use Docker to Shift Left and Automate Deployments

15
Additional Resources

Deploying, Running Applications in Docker Containers

By Jeff Pierce

Deploying and running applications in Containers is one of the hottest trends in DevOps and IT today. Docker, a containerization platform that lets users easily package, deploy, and manage their applications within containers, is principally responsible for bringing containers to the mainstream. This article provides information about what containers are, their relationship with DevOps, and the benefits derived from container implementation.



What are Containers?

Containerization is a virtualization method for encapsulating software and the minimum set of resources and dependencies necessary for that software to run. Containers encapsulate functionality at a higher level than Virtual Machines (VMs) as they do not typically contain a full operating system or services, rather they have a minimalist OS which makes them orders of magnitude smaller than a typical VM (a number of containers can fit into a typical VM disc footprint) and makes them extremely efficient and portable. Containers, with their stripped down OS and minimal services, are typically much faster to deploy and initialize than a full VM. Containers are extremely flexible and facilitate higher workload densities for deployments on bare metal, virtualized, and cloud based infrastructures.

What Do They Have to Do with DevOps?

Modern data center infrastructure management solutions are helping to change software development. The increasing popularity of virtualization technologies, including containerization, has made infrastructure more available and flexible, supporting agile software development processes and practices while increasing integration and collaboration between development and operations. Containers facilitate DevOps and Continuous Integration (CI), Continuous Deployment (CDep), and Continuous Delivery (CDel) by creating smaller, isolated, and more efficient workload deployment packages that are more easily managed and integrated with CI/CD build pipelines.

What Benefits Can One Gain from Containers?

- Resource utilization/efficiency (ROI)
- Standardization and consistency
- Workload Portability
- CI/CD efficiency and Rapid Deployment
- Cloud multi-platform support
- Security through isolation and segregation of applications and resources

3
Why Choose Containerization?

5
Virtualization or Containerization? Choosing the Right Strategy

6
Deploying, Running Applications in Docker Containers

7
How Docker Enables Agile Software Development

12
Demystifying DevOps: A Day in the Life of a DevOps Tester

13
Exploring Containers: Creating a Dockerfile

14
How Testers Can Use Docker to Shift Left and Automate Deployments

15
Additional Resources

How Docker Enables Agile Software Development

By Deepak Vohra

Docker is the de facto containerization platform, and it has revolutionized how software is packaged, distributed, and deployed. It runs software that is packaged and distributed as Docker images in Docker containers that run on Docker Engine.

Docker has facilitated the adoption of the microservices architecture, which decouples services components and facilitates making iterative changes to software services. In fact, Docker makes software development and deployment more agile. Here's how.

Simple, Modular, Sustainable Design

Docker design is sustainable, as it makes a more efficient use of the operating system compared to a virtual machine.

Virtual machines run on top of a hypervisor, which runs on top of an underlying OS, as illustrated in figure 1. Each VM uses up a whole guest operating system, which is not very efficient or sustainable in terms of resource consumption.

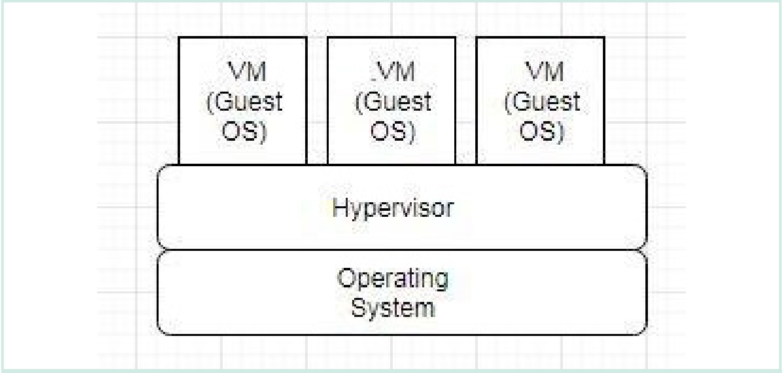


Figure 1

A Docker container does not make use of a whole operating system, instead only employing a snapshot of the underlying OS kernel, thus making it more lightweight and sustainable in terms of resource consumption. Multiple Docker containers run in isolation, with each having its own file system and networking, on top of a single Docker Engine using the same OS kernel, as illustrated in figure 2.

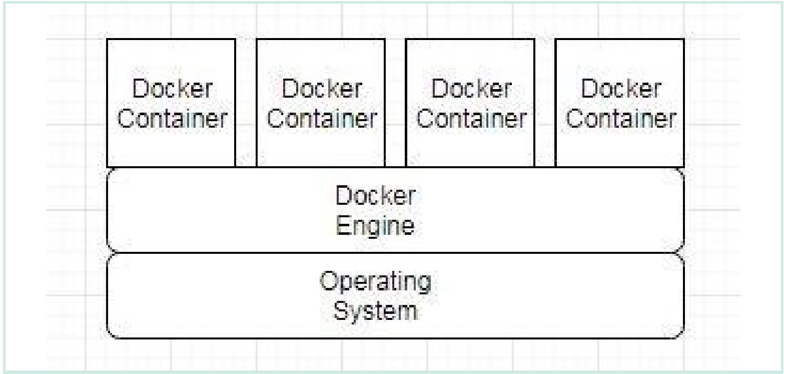


Figure 2

Docker design is simpler, modular, and less resource-intensive, which encourages leaner, more agile development practices.

Efficient Software Delivery

Docker delivers pre-packaged software in the form of reusable, modular Docker images. More specifically, a Docker image is built from a Dockerfile, which consists of instructions and commands to run in order to download, install, and run the software.

3
Why Choose Containerization?

5
Virtualization or Containerization? Choosing the Right Strategy

6
Deploying, Running Applications in Docker Containers

7
How Docker Enables Agile Software Development

12
Demystifying DevOps: A Day in the Life of a DevOps Tester

13
Exploring Containers: Creating a Dockerfile

14
How Testers Can Use Docker to Shift Left and Automate Deployments

15
Additional Resources

A Docker image is a set of layers, with each layer representing an instruction or command in a Dockerfile. This takes away the hassle of downloading and installing individual components of software.

Docker images may be pulled or downloaded from a repository, such as Docker Hub, and are provided for Linux and Windows OS and support several different types of architectures, including amd64, arm32v5, arm32v7, arm64v8, i386, ppc64le, s390x, and windows-amd64.

Working Software

Docker provides working software in that the software is ready to be run without further configuration. The simple command docker run <image> runs the software packaged in a Docker image and all the dependencies packaged with it.

For example, if a software depends on a specific version of Java, the Java version is also downloaded and installed with the other software that is downloaded and installed. Running software from a Docker image is illustrated in figure 3.

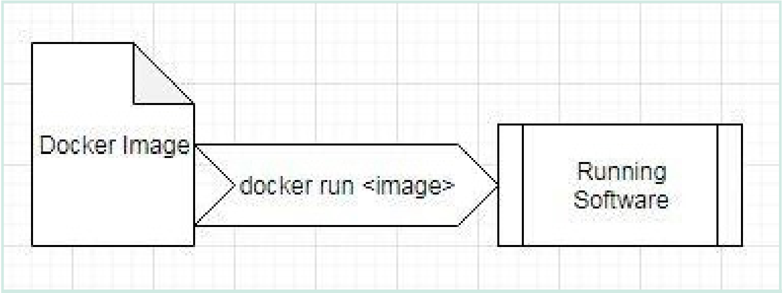


Figure 3

Accommodating Changing Requirements

A Docker image is built from a Dockerfile, which consists of Docker syntax instructions. A Dockerfile gets built into a Docker image with the docker build command, and the image is tagged to distinguish the different builds generated from the same Dockerfile.

If some requirement changes, the Dockerfile could be modified accordingly to generate a new image with a new tag. Consequently, multiple versions of software could be made available using different tags.

The default tag is "latest," and a subsequent Docker image built using a tag that already exists overwrites an earlier image with the same tag. Tagged Docker images for three different versions (v1, v2, and v3) of a Dockerfile are illustrated in figure 4.

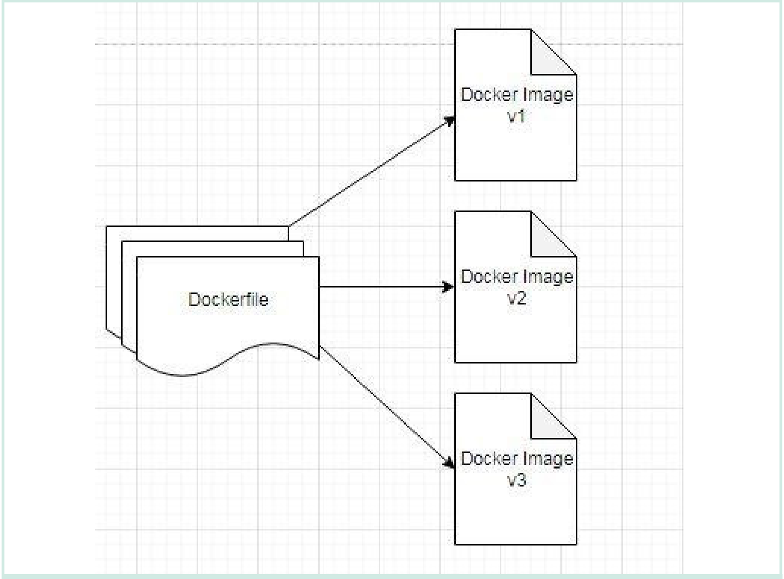


Figure 4

Iterative, Test-Driven Development

Because Docker distributes ready-to-install software as pre-packaged Docker images, it supports iterative, test-driven development.

The source code could be hosted on an online repository such as GitHub. A single command docker build creates a Docker image from the source code Dockerfile. The Docker image could be tested in a test environment before deploying in production. A single com-

3 Why Choose Containerization?
5 Virtualization or Containerization? Choosing the Right Strategy
6 Deploying, Running Applications in Docker Containers
7 How Docker Enables Agile Software Development
12 Demystifying DevOps: A Day in the Life of a DevOps Tester
13 Exploring Containers: Creating a Dockerfile
14 How Testers Can Use Docker to Shift Left and Automate Deployments
15 Additional Resources

mand docker run deploys the Docker image as running software. The process is illustrated below in figure 5.

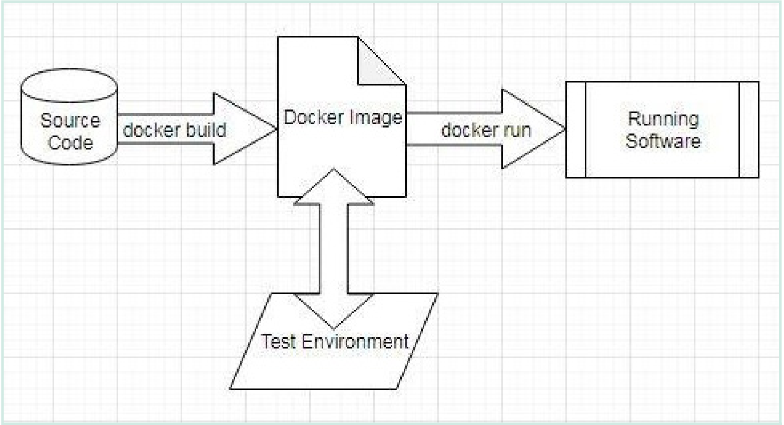


Figure 5

Most Docker container orchestration platforms support rolling upgrades so that software can be updated and deployed iteratively.

Automation

Docker lends itself to automation very well. Each of the build, test, and deploy processes could be automated with pipeline-based automation tools such as Jenkins.

Continuous Integration and Continuous Testing

In the context of Docker, continuous integration refers to integrating source code that is checked into a source code control system (like GitHub) into a Docker image continuously with each successive check-in.

Build automation tools like Jenkins could be used to develop a build pipeline that builds source code on GitHub into a new Docker image each time code is committed to GitHub. The Docker image also could be tested continuously using automated tests in the build pipeline.

After testing a Docker image, it could be uploaded to a Docker image repository, such as Docker Hub, using the docker push command, and this process can also be automated in the build pipeline.

As a result, the source code for software could be integrated continuously into a usable form of a Docker image. The Jenkins pipeline for a continuous integration process is illustrated in figure 6.

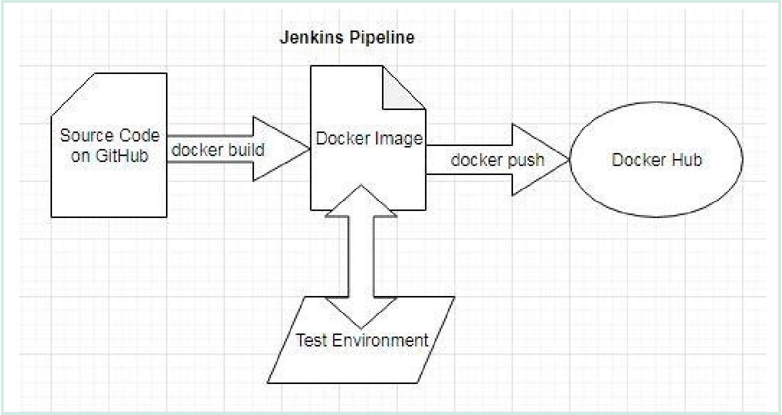


Figure 6

Continuous Delivery

Continuous delivery is the next phase in the software development process. Continuous delivery is defined as making usable software available for deployment without actually deploying the software into production.

Docker lends itself to automation very well. Each of the build, test, and deploy processes could be automated with pipeline-based automation tools such as Jenkins.

3
Why Choose Containerization?

5
Virtualization or Containerization? Choosing the Right Strategy

6
Deploying, Running Applications in Docker Containers

7
How Docker Enables Agile Software Development

12
Demystifying DevOps: A Day in the Life of a DevOps Tester

13
Exploring Containers: Creating a Dockerfile

14
How Testers Can Use Docker to Shift Left and Automate Deployments

15
Additional Resources

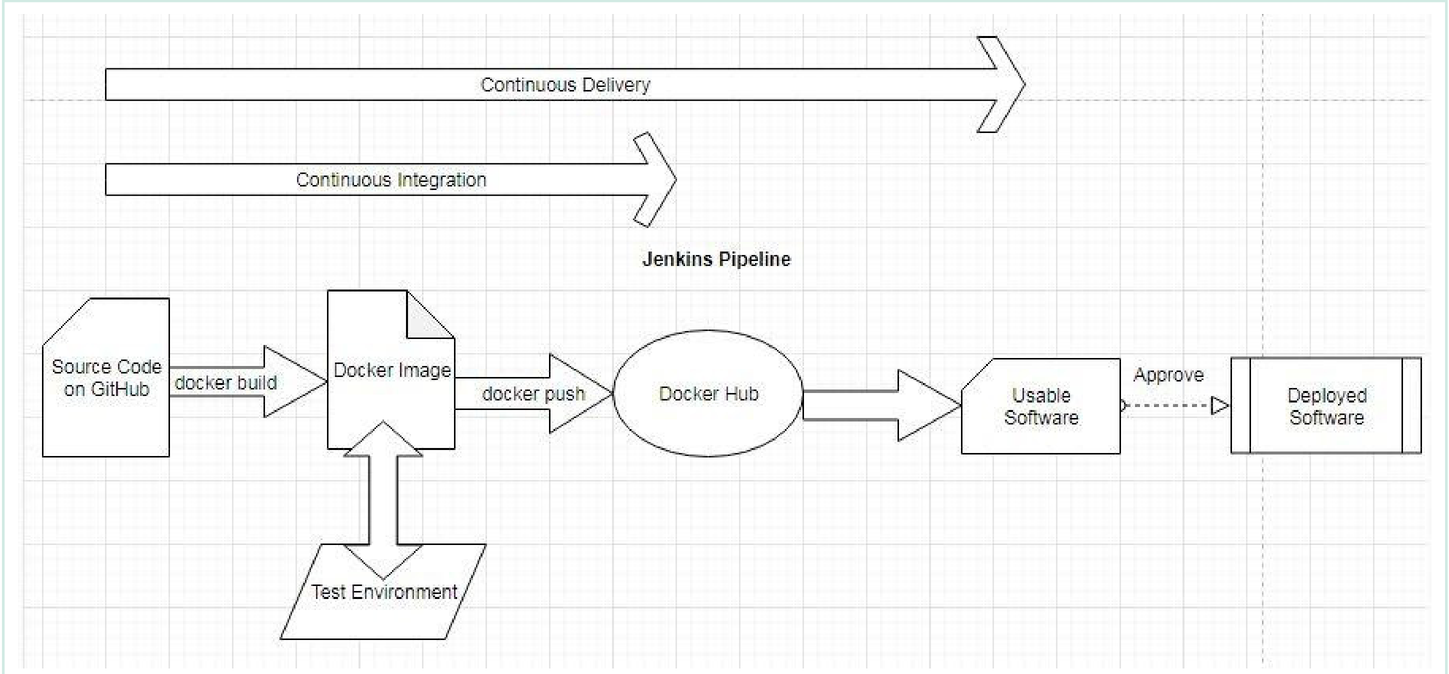


Figure 7

Continuous delivery may include deploying software into some staging environment after passing CI and running a suite of tests against the software in that environment. A user or administrator has to approve the software for deployment into production. A build pipeline again could be used for continuous delivery, as illustrated in figure 7.

Converting a Docker image into production-quality software could involve further testing to sure an image is usable. Some services also require the microservices they depend on to be available in some way to make the service useful.

Continuous Deployment

Continuous deployment fully automates software development, testing, and running an application. The usable software is deployed continuously to production without user intervention by using rolling upgrades, as illustrated in figure 8. A build pipeline could be used for continuous deployment as well.

Collaboration with Software Users

By automating the Docker build, test, deliver, and deployment processes, it becomes easier to collaborate with your software's end-users.

3
Why Choose Containerization?

5
Virtualization or Containerization? Choosing the Right Strategy

6
Deploying, Running Applications in Docker Containers

7
How Docker Enables Agile Software Development

12
Demystifying DevOps: A Day in the Life of a DevOps Tester

13
Exploring Containers: Creating a Dockerfile

14
How Testers Can Use Docker to Shift Left and Automate Deployments

15
Additional Resources

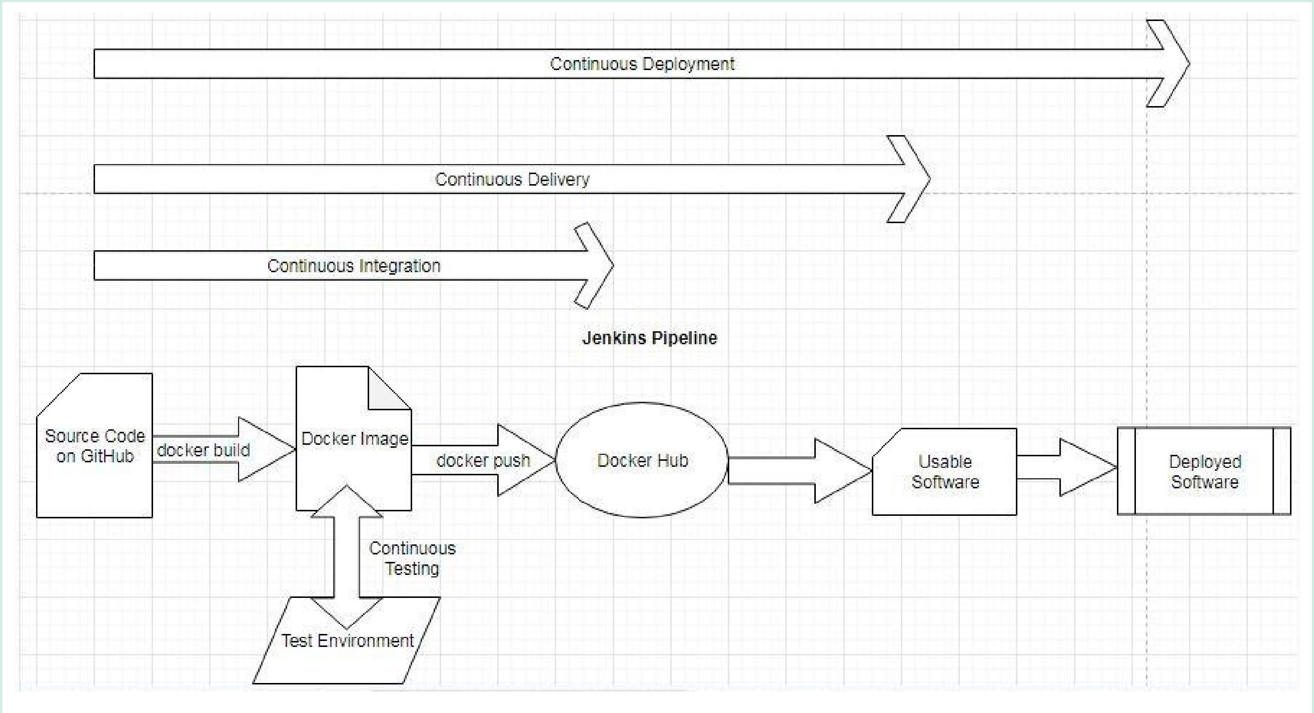


Figure 8

End-user production deployments of artifacts that have passed through a continuous delivery cycle are valuable. This gives the development team immediate feedback on the software while waiting for the users to be ready to accept a new release on their terms. Because Docker images are tagged, different end-users could use different versions of the same software customized to their needs.

A multi-branch Jenkins pipeline provides for further collaboration with software end users. For example, some of the branches of the pipeline could be allocated to the software end-user team while the other branches are managed by the software development team. The end-users may suggest changes more frequently than when using a non-Docker application, as it is easier to update software packaged, distributed, and deployed with Docker.

A Tool for Agile Work

Docker facilitates modular design for working software, sustainable resource consumption, efficient software delivery, continuous integration, continuous delivery, continuous deployment, and collaboration with end-users, all of which are founding principles of agile software development. In this way, using Docker as your containerization platform can actually help make your software development, testing, delivery, and deployment more agile.

3
Why Choose Containerization?

5
Virtualization or Containerization? Choosing the Right Strategy

6
Deploying, Running Applications in Docker Containers

7
How Docker Enables Agile Software Development

12
Demystifying DevOps: A Day in the Life of a DevOps Tester

13
Exploring Containers: Creating a Dockerfile

14
How Testers Can Use Docker to Shift Left and Automate Deployments

15
Additional Resources

Demystifying DevOps: A Day in the Life of a DevOps Tester

By Justin Rohrman

The idea of working as a test specialist on a team using DevOps can feel intimidating. There are at least two technology stacks, containerization and continuous integration, that you need to be familiar with. Add a source code repository like Git, a few test frameworks, and a scripting language to bundle everything together, and you start to approach a useful skill set.

My experience has been that very few people need to be able to start from scratch. Here is what a normal day of testing in DevOps looks like for me.

My current team follows pairing and Extreme Programming practices. For every code change, there are two developers and one test specialist. Most of the time, we start a change request by building a new test. This might be in RSpec or through Cucumber. Either a developer or a test specialist will write the test, and then a developer will write code to make that test pass. This helps us to understand the code we are writing, to refactor with less worry, and to know when we are done.

Throughout this red, green, refactor process, we are building new environments locally. One of us will push a change, and I might pull those changes and build a local environment.

The containers come in when we are closer to a usable version of the change. Many of the DevOps stories I read involve using commands I can never remember in workflows that feel archaic. I am able to do everything through our continuous integration system.

First I use a drop list to select the branch I want to use to build a new container. Building the container takes about five minutes. After my container is built, I select which environment to deploy to. After a few more minutes, I have a production-like environment available to explore and perform test ideas that are more complex than the programmatic testing we do during the development flow.

Pairing and having a test specialist removes most of the easy-to-find bugs that normal development leaves for the testing role. Generally I will still find problems related to the unpredictable way customers can use the product. I demo these bugs, then we make a decision as a group about whether they matter. Deciding to make a fix puts us back in the red, green, refactor cycle.

Over the course of a feature change, I might make one container, or I might make several. But it is very easy, taking somewhere around seven minutes each time I need to create a new environment from scratch. Once we feel like we are in a deliverable state, we create one last container environment in a branch merged with the head branch and demo to our product owner. That demo includes a discussion of any lingering things we decided not to fix so we can get some fresh perspective on them.

Testing in DevOps can feel scary, but like most things, once you get in a day-to-day groove, it becomes an indispensable part of your workflow.

3
Why Choose Containerization?
5
Virtualization or Containerization? Choosing the Right Strategy
6
Deploying, Running Applications in Docker Containers
7
How Docker Enables Agile Software Development
12
Demystifying DevOps: A Day in the Life of a DevOps Tester
13
Exploring Containers: Creating a Dockerfile
14
How Testers Can Use Docker to Shift Left and Automate Deployments
15
Additional Resources

Exploring Containers: Creating a Dockerfile

By Deepak Vohra

Docker is the most popular containerization platform. A Docker container runs software isolated from all other containers on a Docker engine, and each has its own filesystem and networking. Docker images encapsulate all the software, including dependencies that are needed to launch a Docker container.

A Docker image is a set of layers that interact with each other. Each layer encapsulates a single command or instruction in a Dockerfile, which needs to be implemented to build a Docker image. The layers are stacked and share the files and directories from underlying layers.

Each of an image's layers is read-only. When a container is created from an image, a layer is added on top that is writable. Multiple containers may be run using the same image, and each new container shares the image's layers.

Let's look at what goes into creating a Dockerfile, which could be used to build a runnable Docker image.

Creating an image

The first command or instruction in a Dockerfile specifies which image is used as the base image, using the FROM command. When an image is built from another image, it doesn't create all the layers from the base image again; it shares the layers with the base image.

An image doesn't have to be built from another. It may be built by specifying the FROM scratch command. This gives a mostly empty container as a base state for creating new images.

Copying files and directories

The COPY command is used to copy files and directories to a destination directory in the container created from an image. The ADD command is similar, with the additional feature to copy from a remote file URL.

Running an image

When an image is run, a container is created. The CMD instruction specifies what command to run within a new container.

The CMD instruction has a shell form and an exec form. The exec form includes the executable or application to run, along with the parameters for the application. For service-based images, such as an image to run an Apache2 httpd service, the exec form should be used. The shell form is for all other uses.

The ENTRYPOINT instruction also configures the command to run in a new container and also has exec and shell forms. The difference is that ENTRYPOINT must specify an executable, while CMD could specify some parameters supplied to the ENTRYPOINT.

The RUN instruction is also used to run a command or an executable and also has exec and shell forms. It is often confused as an alternative to CMD and ENTRYPOINT, but RUN is not run in a new container. It's used to build the Docker image and commit the result, implying that the result is saved in the image as a layer.

Setting the working directory

The working directory may be set with the WORKDIR instruction. It may be a relative path or an absolute path. Multiple WORKDIR instructions may be specified in a single Dockerfile, and if a relative path is provided, it is relative to the directory path of the preceding WORKDIR instruction.

3
Why Choose Containerization?

5
Virtualization or Containerization? Choosing the Right Strategy

6
Deploying, Running Applications in Docker Containers

7
How Docker Enables Agile Software Development

12
Demystifying DevOps: A Day in the Life of a DevOps Tester

13
Exploring Containers: Creating a Dockerfile

14
How Testers Can Use Docker to Shift Left and Automate Deployments

15
Additional Resources

How Testers Can Use Docker to Shift Left and Automate Deployments

By Artem Golubev

Docker is an increasingly popular lightweight virtualization system. It has several advantages over virtual machines, such as having a declarative description of the file system, and it's easier to deal with, starts up faster, and requires fewer resources in general.

In particular, the layered file system lets you deliver new updates to data centers and get updates from developers faster, and being lightweight means it is much easier and more efficient to run locally. There are images as small as 2 Mb—you'll never find a VM image that small.

It's also easier for developers and ops to create lightweight OS images with software already set up, and they can be run exactly the same way on both the developer's machine and the test and production environments.

There are multiple ways Docker can be used. I'll dig deeper into how it can help with testing in continuous delivery and integration (CD and CI) in certain cases.

The increasingly popular way of setting up a development process is a pull request-based workflow popularized by GitHub:

1. A developer writes code and creates a pull request, adding their code to the main code
2. Other developers can now review the code, and it can also be deployed
3. Once review and testing is complete, code is merged to the main code

The new code can be directly deployed to the production environment (full CD) or to a test environment (CI).

When Docker is added to this workflow, a deployable Docker image is created during step one, which is the same image that eventually will go to production. It also usually can be pulled to a developer's machine for debugging, to avoid the "But it works on my machine" issue. In this workflow, not only can all end-to-end tests be run on the pull request code under test, but they also can be deployed and tested by QA.

Docker Compose is a way to join several images to work in tandem, such as a web server and database in one compose. Using Docker and Docker Compose, it is possible to deploy the pull request's code on a completely fresh environment with a fresh database, eliminating issues with contaminated data.

The alternative, more complex way to do the same thing would be by using Kubernetes, which is designed to deploy sets of images (similar to Compose) to production or test environments.

The ability for developers to use exactly the same environment that will be used in production helps to not only eliminate and debug environment issues, but also simplify and streamline delivery of the code to both test environment and production. The same image can be used on a developer machine as on a test environment and, once tested, can be delivered to production.

This also enables deploying those environments on demand and working with each image (potentially related to a code branch) separately, isolating any issues. This constitutes an effective shift left, which allows QA and developers to work much more effectively.

3 Why Choose Containerization?
5 Virtualization or Containerization? Choosing the Right Strategy
6 Deploying, Running Applications in Docker Containers
7 How Docker Enables Agile Software Development
12 Demystifying DevOps: A Day in the Life of a DevOps Tester
13 Exploring Containers: Creating a Dockerfile
14 How Testers Can Use Docker to Shift Left and Automate Deployments
15 Additional Resources

Additional Resources

MORE INFORMATION FOR SOFTWARE PROFESSIONALS

TECHWELLHUB A SLACK COMMUNITY

The TechWell Hub is a great resource to get your questions answered, help others with problems they're stuck on, and engage with experts in software. Follow channels like #DevOps, #DevSecOps, and more.

JOIN HERE

NARROW YOUR SEARCH TO A SPECIFIC TYPE OF RESOURCE:

AgileConnection Community

AgileConnection brings you the latest in agile and DevOps principles, practices, and technologies. Check out articles and interviews from experienced software professionals and thought leaders, and join the community to gain access to member-exclusive content such as conference presentations, weekly newsletter updates, Q&A discussions, and more.

DevOps Articles

AgileConnection is home to thousands of DevOps software resources, including articles, archived *Better Software* magazine articles, conference presentations, and interviews with industry notables. Check out the latest agile and DevOps articles and read about how to speed delivery, reduce risk, and build security in from the start.

TechWell Conference Presentations

Couldn't make it to a TechWell conference to sharpen your agile and DevOps skills and knowledge? TechWell conference presentations are available to AgileConnection members soon after conferences end. **Click here** to join AgileConnection and access conference presentations related to security and DevSecOps.

Interviews

Each year, TechWell interviews dozens of software professionals, including well-known thought leaders, seasoned practitioners, and respected conference speakers. **Click here** to read, listen to, and watch interviews with DevOps, agile, and other experts.

Agile + DevOps Virtual

In light of recent events, TechWell has morphed the popular Agile + DevOps East conference into a fully virtual experience this year! From the comfort of your own digital device, you will have access to all of the same great content and experts you have come to expect from an Agile + DevOps East conference.

DevSecOps Summit at Agile + DevOps Virtual

The DevSecOps Summit is a fully-virtual, multi-day series of first-person talks, giving an ideal perspective on how you and your team can enable faster application development with more rapid deployment to production while integrating security into your DevOps initiatives. Explore the program **here**.

coveros **Accelerate Delivery**

Our partner, Coveros, has significant DevOps experience (including containerization) and can help organizations implement DevOps with security in mind or integrate security capabilities into existing DevOps processes. Coveros offers more than a dozen courses on DevSecOps, DevOps, and security—all of which include best practices taught by industry leaders. Whether you're looking to get hands-on experience for yourself, your team, or your organization, Coveros has a learning solution for you.

DevOps & DevSecOps Courses | **Software Security Courses** | **Agile & DevOps Transformations** | **DevOps Engineering** | **DevSecOps**

TECHWELL™