Docker: The software development revolution continued

First, Docker transformed containers into a universally useful technology. Today, its development tools make it easy to create, package, and ship programs across all platforms.

By Steven J. Vaughan-Nichols

The software development revolution continued

A decade ago, Solomon Hykes founded Docker. The container revolution this sparked is still burning bright to this day. Its trilogy of a long-running daemon process server, dockerd; application programming interfaces (API); and command line interface (CLI) client, docker, would prove the foundation for the evolution of a comprehensive set of developer tools. Today, Docker's container-oriented development tools form a comprehensive foundation for building, packaging, shipping, and securing code.

While Docker's foundations come from container development, it offers much more than that. Docker provides a comprehensive container development stack designed to help teams better collaborate and innovate. Indeed, the entire cloud-native revolution couldn't have happened without Docker. It's the well-spring of modern cloud computing. Read on to learn how to make the most out of Docker's developer tools by leveraging its full ecosystem of software development tools.

Understanding the Docker Approach

Before Docker jump started the container revolution, we used virtual machines (VMs) to maximize our use of hardware and deploy applications. They were great in their day, but the newer container abstraction provides superior application portability, easy customization, consistency, and improved testing and confidence from dev to production. Containers also improved IT staff headaches from managing VM patching and configuration. Because VMs are system resources fat, while containers are resources slender. For practical purposes, that means you can run far more containers than VMs on the exact same hardware.

Docker's core idea is to bridge the gap between development and production environments, so there are fewer problems when software is deployed. This gives developers the flexibility to use the tools they want, making it easier to code and ultimately deliver high-quality solutions. Docker does this by separating applications from the underlying infrastructure. This separation enables rapid software delivery and ensures security. It also lets users manage their infrastructure in the same way they manage their applications. By leveraging Docker's methodologies for shipping, testing, and deploying code swiftly, teams can significantly shorten the time lag between code writing and its deployment in production.

All of this helps to increase the time developers spend on innovation and decrease the time they spend on everything else. The company accomplishes this by making containers manageable - and the inner loop of development devoid of complexity - by providing a toolkit of products and services that out-of-the-box are ready to use and get developers up and running quickly with creating containers and executing CI/CD pipelines fluidly.

This is all done within a comprehensive set of software development tools, the heart of which is the hub and spoke model.

In this approach, the hub and spoke provide a design blueprint that's used to organize intricate and expansive systems. Within this framework, the "hub" serves as the core connection point for various "spokes." Each spoke represents unique tools or components such as IDEs, Git, CI, CD, FOSS, etc. This model fosters loose interconnection and strong unity among system parts.

This ensures that individual components aren't strongly interdependent. This, in turn, simplifies the process of modifying or substituting one without disturbing the rest. At the same time, its strong unity ensures that every component has a distinct and clear role. This model reflects how developers work, and abstracts complexity away from developers enabling them to focus on solving problems with code.

Container-first with Docker

Docker enables the packaging and execution of an application in a loosely isolated environment termed a 'container.' This isolated container guarantees security and allows numerous containers to run concurrently on a given host.

Containers do this by sharing operating system functionality. This means they are much more efficient than VM hypervisors in system resource terms. Instead of virtualizing hardware, containers rest on top of a single Linux instance. By drastically reducing the memory footprint, you're left with a small, neat capsule containing your application.

These lightweight containers house all the necessities for running an application. Therefore, users need not rely on the existing software on the host. Docker's ease of sharing these containers ensures that everyone involved receives an identical, consistently functioning container.

Another reason containers are popular is they lend themselves to Continuous Integration/Continuous Deployment (CI/CD). This is a DevOps methodology designed to encourage developers to integrate their code into a shared repository early and often, and then deploy the code quickly and efficiently.

In short, Docker enables developers to easily pack, ship, and run any application as a lightweight, portable, self-sufficient container, which can run virtually anywhere. Thus, containers give you instant application portability.

For instance, developers can write code locally and share their work with their colleagues using containers. Since containers enable programmers to develop in the same environment used in production, this ensures the software will work correctly in dev and ops. Docker facilitates pushing applications into a test environment and executing automated and manual tests. Bugs found during this process can be fixed in the development environment and redeployed to the test environment for testing and validation. Once the testing phase concludes, delivering the fix to the customer is as simple as pushing the updated image to the production environment.

Docker's container tools offer high portability for workloads. Docker containers can run on a variety of environments, from a developer’s local laptop to physical or virtual machines in a data center, or even on cloud providers.

Diving Deeper: A Comprehensive Developer Toolkit

From a high level, you start by downloading Docker Desktop on your local machine. With it you get access to an interface (the GUI) that allows your teams to collaborate, connect to container libraries and your preferred cloud ISP, view usage data, and more. In your Docker Desktop install, you'll get all of the typical tools you likely know of when you think of Docker: Docker Engine, the Docker CLI client, Docker Compose, Docker Content Trust, Kubernetes, and Credential Helper. From there, you can get started with containers, including features like autobuilds, Kubernetes services, and Build/BuildKit. You also get access to Docker Hub, the largest repository of container images, to retrieve useful images or publish your own images. And, these can be checked for errors by Docker Scout (we'll go more into these later).

It's worth noting that all of the "magic" of Docker comes from its architecture.

Behind all of this, Docker employs a client-server architecture. The Docker client communicates with the Docker server daemon (dockerd). This is responsible for the heavy lifting of building, running, and distributing Docker containers. Both the Docker client and daemon can run on the same system, or a Docker client can connect to a remote Docker daemon. They communicate using a REST API, over network sockets.

Dockerd listens for Docker API requests and handles Docker objects like images, containers, networks, and volumes. It can also interact with other daemons to manage Docker services. It's primarily used on servers and Linux-based systems. It's a good fit for production environments, CI/CD workflows, and other scenarios where you need to automate Docker in a Linux environment.

Docker Engine is container central, serving as the core runtime that builds and runs containers. As a server-side application, it does the heavy lifting of container management. Running just about your operating system kernel, you use it to build images and then containers that you can run on your local machine.

Included in Docker Desktop, Docker Engine supports the build of creating software images. Docker images are created using a Dockerfile with a simple syntax for defining the steps needed to create and run the image. Every instruction in a Dockerfile adds a layer to the image. Only the changed layers are rebuilt if a Dockerfile is altered and the image is rebuilt. This contributes to Docker images' lightweight, small, and fast features.

Conversely, a container can be controlled using the Docker API or CLI, and its image and configuration options define it. Any changes to a container's state that are not stored in persistent storage vanish when the container is removed.

The Docker Ecosystem

Docker's development programs are designed to work with the inner and outer loops. The inner loop is mainly between the engineer and his machine. This is when the developer and their partners write software on their laptops. Here, they make sense of the project and create code that addresses the project's goals. It's also at this stage that they run tests. When the program starts working, you close the inner loop.

Now you share that code with the rest of your team and go into the outer loop. At that point, you're ready to work with your continuous integration (CI) system.

You could do this with just the Docker CLI client and server, but it's easier to use Docker's higher-level tools, starting with Docker Desktop. Docker Desktop comprises the Docker daemon, Docker client, Docker Compose, Docker Content Trust, Kubernetes, and Credential Helper. Docker Desktop provides the entire environment to develop containers locally, and does all of the heavy lifting when it comes to set-up and maintenance.

Docker Compose is well worth a particular mention. This tool is used for defining and running multi-container Docker applications. This is key for developing applications requiring multiple services to communicate with each other. So, for example, if your application requires other services like databases, caching servers, or microservices where different services interact, Compose is your program of choice.

You can also use Docker Extensions within Docker Desktop. These extend Docker Desktop's functionality and seamlessly connect your favorite development tools to your application development and deployment workflows. Whether you build your own custom add-on using the Docker Extensions Software Development Kit (SDK) or you choose a partner extension from the marketplace, you can add debugging, testing, security, and networking functionalities and tools to your Desktop instance.

Last, but not least, Docker provides foundations for building reliable container images. Docker Hub, Docker's official public image registry (accessible to anyone), is Docker's default search destination for images. Here, you will find Docker Trusted Open Source Content; reliable images that exemplify best practices with Docker Official images, Docker Verified Publisher images, and Docker-sponsored Open Source images. Users can also set up their private registry. The Docker CLI pull or run commands draw required images from the configured registry. Conversely, the Docker push command allows users to push their images to the configured registry.

A Day of Docker

Let's take a very high-altitude look at how you'd use the Docker tools. Presuming you already have Docker Desktop, Docker Compose, and Docker Engine installed, your first step to working with a new project is to pull the images you'll need from Docker Hub. There's no need to reinvent the wheel when the applications you need, say Node.JS and Ngnix, are only a pull away.

Better still, starting with November 2022's Docker Desktop v4.14.x release, Docker Desktop includes a "Search" feature in its dashboard. With it, you can search containers, local images, and Docker Hub images. Once found, you can pull and spin up Docker containers right from your search's result. You can also use the “Copy docker run” option to retrieve and run the original run command, plus its parameters/details. This enables you to find and use the container's original environment variables.

Let's say you want to run your application on Red Hat OpenShift. No problem! You can use a Docker Extension to deploy it there without any fuss or muss.

That done, you write the application itself with your IDE of choice. Docker works and plays well with most IDEs. Of course, you're not writing it by yourself.

As believers in developing securely from the start, let's check our images and newly minted code for problems. For that, we'll use security and software supply chain protection programs from Anchore, Snyk, and JFrog via Docker Extensions to look for problems.

If all checks out, you'll put your code changes into your favorite CI/CD pipeline. Almost every CI/CD program, such as Jenkins, GitHub Actions, or CircleCI, supports Docker. While we're at it, we'll automatically recheck it for potential security problems. Additionally, the Harness Drone extension, which defines and executes build pipelines inside containers, is just one example of many where Docker allows teams to expand beyond local development use cases and integrate with the CI/CD solution that aligns with the team's needs.

All's good? Then let's deploy this bad boy to a registry like Docker Hub or a private registry. From there, orchestration tools such as Kubernetes can pull and deploy the image to a production environment.

Available pre-built images and extensions for continuous deployment platforms like Harness Drone, ArgoCD, Gitlab, Travis can also accelerate your time to production and allow you to align workflows with your existing tools.

Finally, you clean up your development environment with Docker Prune. This will clear out any unused, unnecessary Docker artifacts.

Beyond the Docker Basics: Adapting to Developer Needs

Looking ahead, Docker will soon introduce Docker Scout, currently in Early Access. This unified container security solution is designed to help developers secure the complete software supply chain by analyzing container images, identifying and remediating vulnerabilities, and continuously evaluating policies across repositories.

Scout will do this by detecting vulnerabilities in your images across your SDLC workflows, and will also provide up-to-date vulnerability information as you build your images. In addition, Scout analyzes image contents and generates a detailed report of packages and vulnerabilities that it detects.

After that, Scout will provide you with suggestions on how to address the vulnerabilities it detects. Not all vulnerabilities are equally bad.

Docker Scout is integrated throughout Docker user interfaces and the CLI. You'll be able to leverage it through Docker Hub, Docker Desktop, and Docker's integration with GitHub Actions.

The free version of Docker Scout includes unlimited local image analysis, up to 3 Docker Scout-enabled repositories, Software Development Life Cycle (SDLC) integration, including policy evaluation and workload integration, on-prem and cloud container registry integrations, and security posture reporting.

Essentially, Docker Scout will make it easy for developers to secure their applications by assessing images against Docker Scout policies and seeing how the evaluation results change over time. This will help determine focal points for future action for vulnerabilities found in Docker images. As software supply chain infrastructure continues to expand, this will ensure you are always monitoring your entire attack surface.

The Programming World According to Docker

Put it all together, and Docker provides a comprehensive container/Kubernetes programming stack. It doesn't try to be everything to everyone. Instead, it uses its extensions, APIs, and SDKs, making it easy to use with your preferred programming editors, CI/CD pipelines, and other development tools.

With its forthcoming security improvements, any programmer, whether they're working on small, local jobs, or hyper-cloud, cloud-native services for millions, should consider the Docker family. **You'll be glad you did.**

Use Docker's secure development solutions for your infrastructure

Learn more about the #1 most used development tool at www.docker.com.