What is a Docker container?

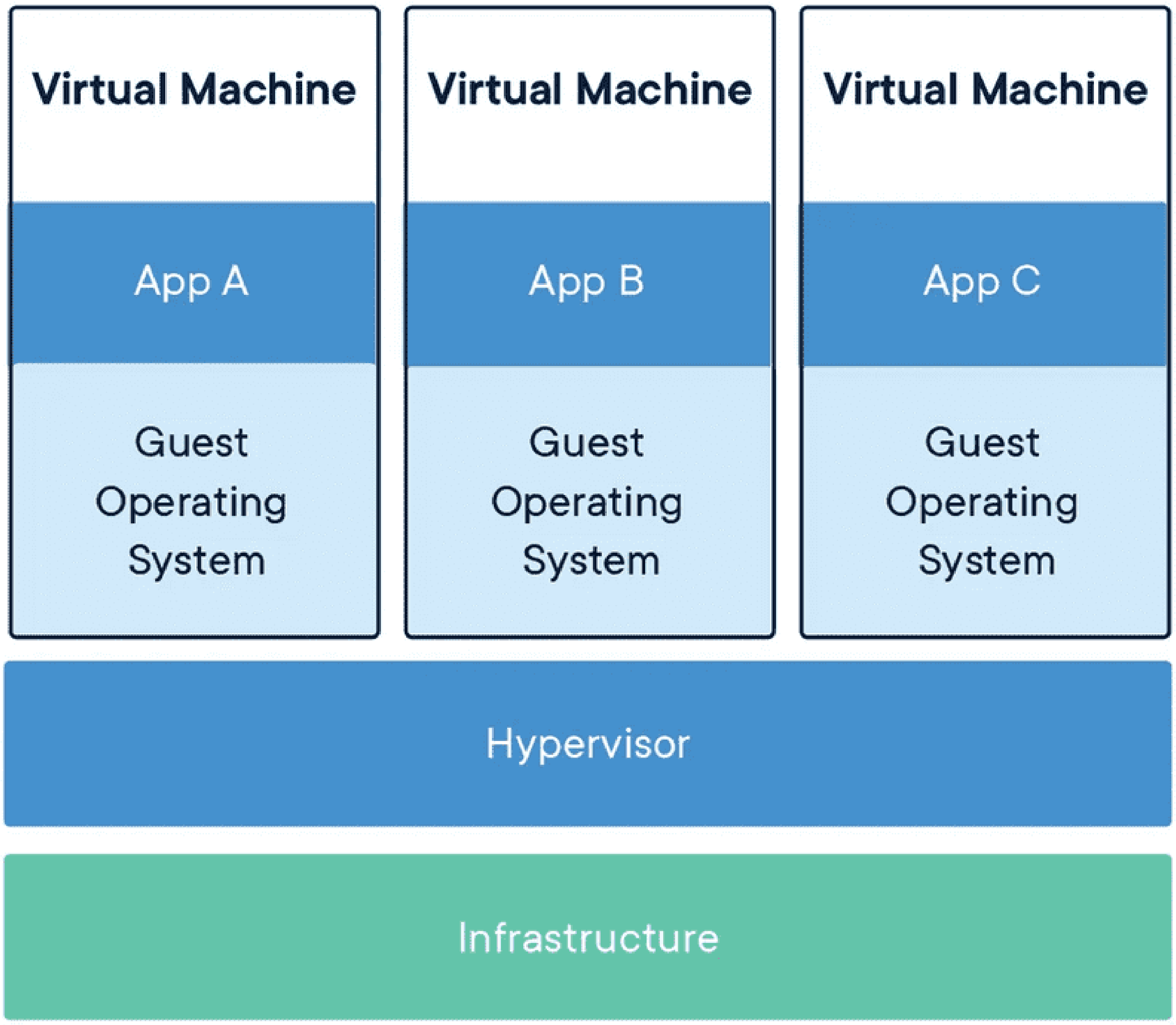
Pratap Sharma

August 17th, 2023 - 75 views / 4 mins read

docker vm

You're probably familiar with the concept of a "Virtual Machine." Virtual machines emulate entire operating systems, allowing you to run software designed for different platforms. For example, you can run a Windows virtual machine on your MacOS computer to use Windows-specific applications.

Consider this diagram of a virtual machine setup:

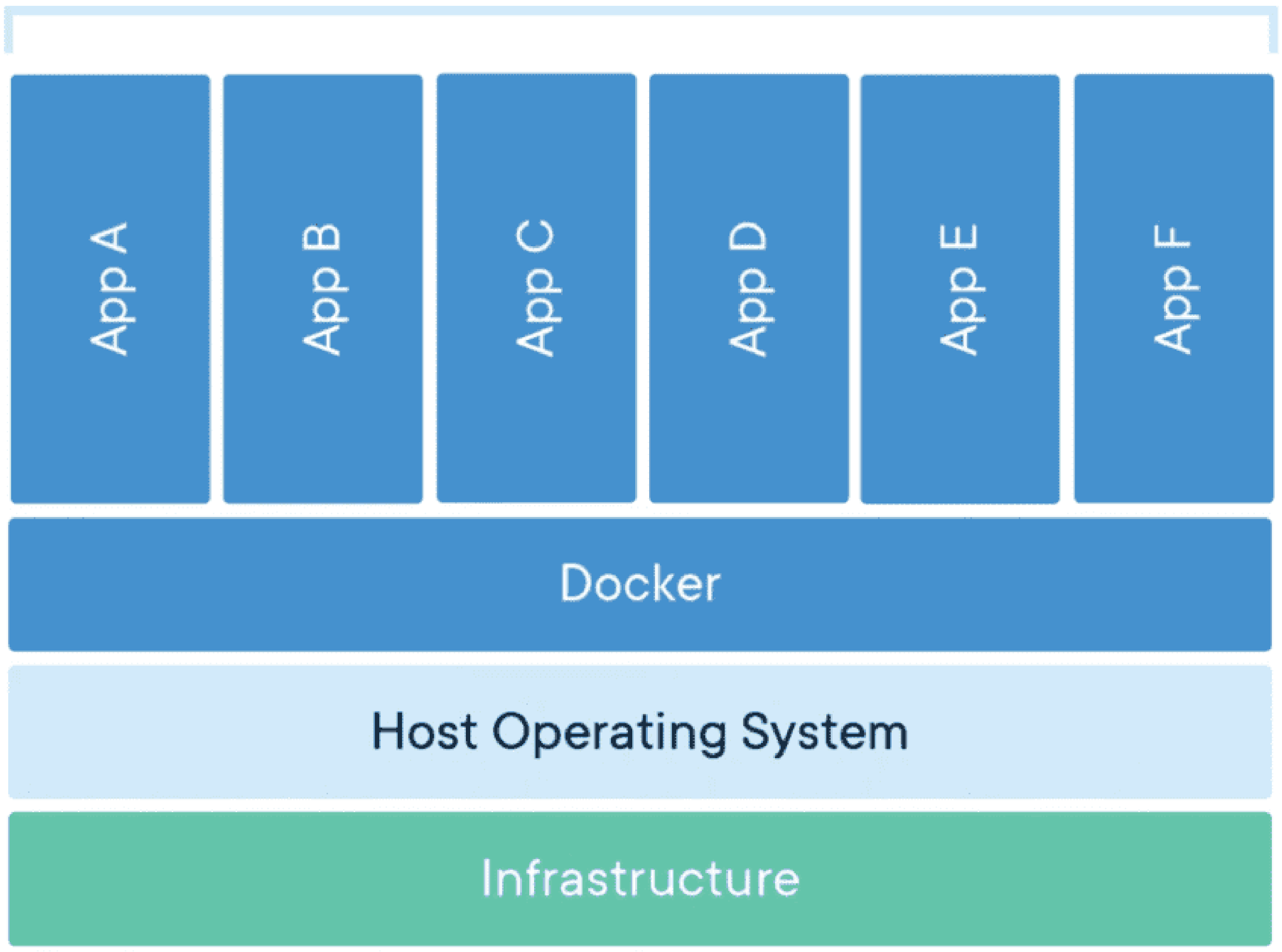


When operating a virtual machine, you can specify the hardware resources it can access, such as a portion of the host's RAM and CPU cores.

Docker containers take a different approach. They don't emulate an entire operating system but utilize your computer's OS kernel to run as processes within the host environment.

Containers have their storage and networking, yet they are notably more lightweight due to their lack of OS emulation.

Visualize how Linux containers function within a Linux host:



This diagram illustrates that the docker -> container segment is significantly more efficient compared to running a virtual machine because it utilizes the host's kernel instead of deploying its own.

The Role of the Kernel

An Operating System comprises two core components:

- The kernel
- The OS's bundled files and programs

For instance, the Linux kernel underpins all Linux-based Operating Systems like Ubuntu, Fedora, and Debian.

Containers depend on the host's kernel, making it impossible to natively run a Windows Docker container on a MacOS host. Similarly, running a Linux

container directly on Windows or MacOS hosts is not feasible.

Running Linux Containers on Windows or MacOS

When utilizing Docker Desktop (which will be explored in an upcoming lecture), it involves the initiation of a Linux Virtual Machine. This virtual machine then serves as the underlying platform for running Linux containers. However, this sequence of steps might appear as follows:

Physical Hardware -> **macOS/Windows** -> **Hypervisor** -> **Linux Virtual Machine** -> **Docker Engine** -> **Container** -> **Program within the Container**

At first glance, this process might seem less efficient compared to directly executing the program within a Linux virtual machine.

It's important to note that while running Linux containers on macOS or Windows is indeed relatively less optimal than executing them within a dedicated Linux VM, this arrangement serves its purpose effectively in the majority of scenarios. Approximately 99% of the time, Linux containers are run on a Linux host, which significantly improves overall efficiency.

The preference for using Linux containers on a Linux host can be attributed to various factors. When deploying applications to share with users, these applications are usually hosted on Linux servers provided by deployment companies. There are multiple reasons driving this choice, including the cost-effectiveness of Linux as an operating system.

Running Applications in Docker Containers

To operate your Flask app within a Docker container, you must create a Docker image containing all the essential dependencies for your app, excluding the OS kernel:

- Python
- Dependencies from requirements.txt
- Potentially nginx or gunicorn (details in the deployment discussion)

What is a Docker Image?

Certainly, here's a shorter version divided into sections:

Docker images capture all essentials for running applications in containers:

- Source code
- Libraries
- Dependencies
- Tools

(Note: Excludes OS kernel)

Ready-Made Images: Convenience and Variety

Pre-built Docker images offer:

- Instant usability
- Diverse configurations
- Complete Linux distributions (e.g., Ubuntu)

Why Ubuntu for Containers

Ubuntu is favored due to:

- Host OS kernel utilization
- Necessity for OS-specific programs
- Reliance on Ubuntu's tools (e.g., C compiler)

Dockerfile

Here's how you define a Docker image using a docker file:

```
FROM python:3.10
EXPOSE 5000
WORKDIR /app
```

```
RUN pip install flask
COPY . .
CMD ["flask", "run", "--host", "0.0.0.0"]
```

This is a Dockerfile, outlining how to create a Docker image. Once you have this file, you instruct Docker to generate the Docker image. Subsequently, you can run the image as a container.

```
Dockerfile ---build--> Docker image ---run--> Docker container
```

In the provided Dockerfile, the initial line states: `FROM python:3.10`. This prompts Docker to first fetch the `python:3.10` image, which someone else has already created. Afterward, Docker executes the subsequent commands to build upon this image.

The separation between Docker images and containers offers a notable advantage: once an image is created, it can be easily shared and deployed:

1. **Sharing**: Images can be shared with developers, ensuring consistent environments.
2. **Deployment**: Images can be deployed to servers, simplifying setup and reducing inconsistencies.

Conclusion

In essence, Docker's clear distinction between images and containers revolutionizes software deployment. Portable images facilitate seamless sharing among teams, while deployment becomes effortless, promoting consistency across environments. This approach, at the core of modern development, empowers efficient application building, sharing, and deployment, marking Docker as a pivotal tool in contemporary software practices.

Learn More

1. Exploring the Power of macro_rules! in Rust
2. How to Setup Path Aliases in a React Native app with Typescript
3. SSH Not Working In MacOS Ventura: How To Fix

Please let me know if there's anything else I can add or if there's any way to improve the post. Also, leave a comment if you have any feedback or suggestions.

Up next

One-to-Many Relationships using Flask SQLAlchemy

How to Setup Path Aliases in a React Native app with Typescript

Author

Hey, I'm Pratap, a full stack software engineer and an instructor. I write about what I know to help viewers like you. If you enjoy my content, please consider supporting what I do!

Buy me a coffee

Become a Patron