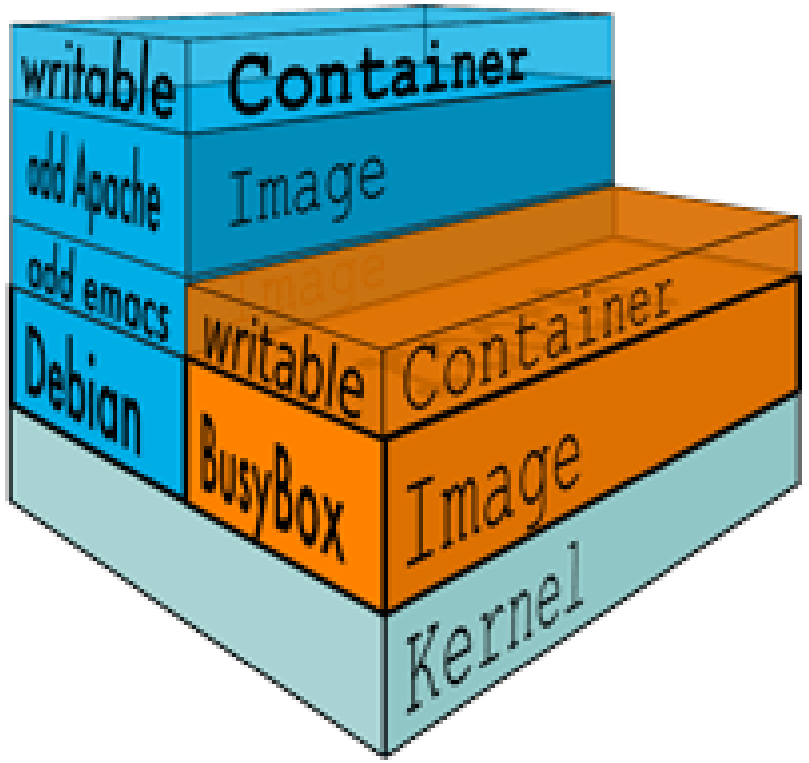
Docker Fundamentals

Mohanraj Shanmugam

What is Docker?



- Docker allows you to package an application with all of its dependencies into a standardized unit for software development.
- Docker containers wrap up a piece of software in a complete filesystem that contains everything it needs to run:
  - code,
  - runtime,
  - system tools,
  - system libraries –

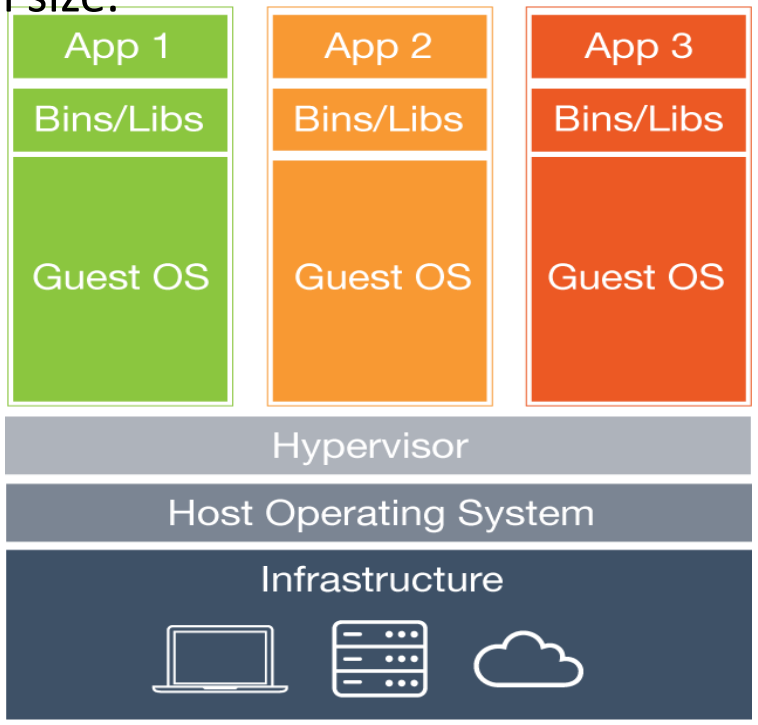
  anything you can install on a server.
- This guarantees that it will always run the same application, regardless of the environment it is running in.

Features of Docker

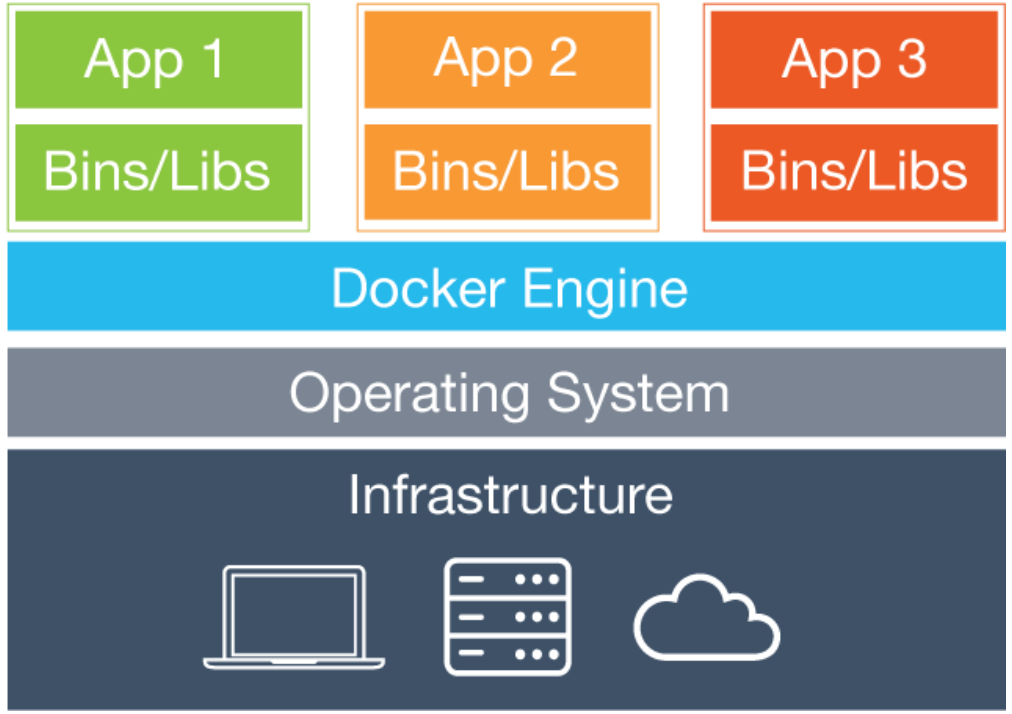
- Lightweight
  - Containers running on a single machine all share the same operating system kernel so they start instantly and make more efficient use of RAM.
  - Images are constructed from layered filesystems so they can share common files, making disk usage and image downloads much more efficient.
- Open
  - Docker containers are based on open standards allowing containers to run on all major Linux distributions and Microsoft operating systems with support for every infrastructure.
- Secure
  - Containers isolate applications from each other and the underlying infrastructure while providing an added layer of protection for the application.

VM Vs Docker

- Virtual Machines
  - Each virtual machine includes the application, the necessary binaries and libraries and an entire guest operating system - all of which may be tens of GBs in size.



- Containers
  - Containers include the application and all of its dependencies, but share the kernel with other containers.
  - They run as an isolated process in userspace on the host operating system.
  - They're also not tied to any specific infrastructure – Docker containers run on any computer, on any infrastructure and in any cloud.



Docker helps to build better Software

- Accelerate Developer Onboarding
  - Stop wasting hours trying to setup developer environments, spin up new instances and make copies of production code to run locally.
  - With Docker, you can easily take copies of your live environment and run on any new endpoint running Docker.
- Empower Developer Creativity
  - The isolation capabilities of Docker containers free developers from the worries of using “approved” language stacks and tooling.
  - Developers can use the best language and tools for their application service without worrying about causing conflict issues.

Docker helps to build better Software

- Eliminate Environment Inconsistencies
  - By packaging up the application with its configs and dependencies together and shipping as a container, the application will always work as designed locally, on another machine, in test or production.
  - No more worries about having to install the same configs into a different environment.

Docker helps to Share Software

- Distribute and share content
  - Store, distribute and manage your Docker images in your Docker Hub with your team.
  - Image updates, changes and history are automatically shared across your organization.
- Simply share your application with others
  - Ship one or many containers to others or downstream service teams without worrying about different environment dependencies creating issues with your application.
  - Other teams can easily link to or test against your app without having to learn or worry about how it works.

Docker helps to Ship Software

- Ship 7X More
  - Docker users on average ship software 7X more after deploying Docker in their environment.
  - More frequent updates provide more value to your customers faster.
- Quickly Scale
  - Docker containers spin up and down in seconds making it easy to scale an application service at any time to satisfy peak customer demand, then just as easily spin down those containers to only use the resources you need when you need it

Docker helps to Ship Software

- Easily Remediate Issues
  - Docker make it easy to identify issues and isolate the problem container, quickly roll back to make the necessary changes then push the updated container into production.
  - The isolation between containers make these changes less disruptive than traditional software models.

Docker is a shipping container system for code

Multiplicity of Stacks

Static website

User DB

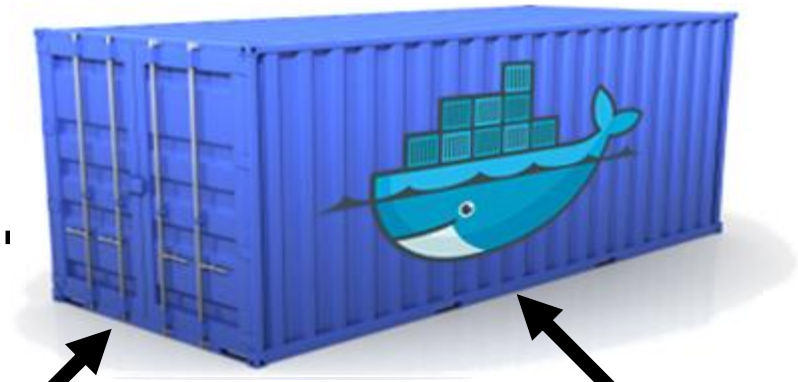
Web frontend

Queue

Analytics DB

Do services and apps interact appropriately?

An engine that enables any payload to be encapsulated as a lightweight, portable, self-sufficient container…



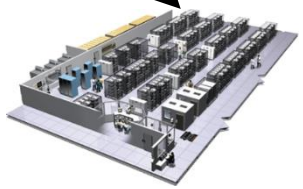
…that can be manipulated using standard operations and run consistently on virtually any hardware platform

Multiplicity of hardware environments

Development VM

QA server

Customer Data Center



Public Cloud

Production Cluster

Contributor's laptop

Can I migrate smoothly and quickly

Who are using Docker

Docker and the Industry

Transforming Business Through Software

- Gone are the days of private datacenters running off-the-shelf software and giant monolithic code bases that you updated once a year.
- Everything has changed. Whether it is moving to the cloud, migrating between clouds, modernizing legacy or building new apps and data structure, the desired results are always the same – speed.
- The faster you can move defines your success as a company
- Software is the critical IP that defines your company even if the actual product you are selling may be a t-shirt, a car, or compounding interest.
- Software is how you engage your customers, reach new users, understand their data, promote your product or service and process their order.

MicroServices

- Small pieces of software that are designed for a very specific job are called microservices.
- The design goal of microservices is to have each service built with all of the necessary components to “run” a specific job a with just the right type of underlying infrastructure resources.
- Then, these services are loosely coupled together so they can be changed at anytime, without having to worry about the service that comes before or after it.

Docker Advantage

- Agility:
  - The speed and simplicity of Docker was an instant hit with developers and is what led to the meteoric rise in the open source project.
  - Developers are now able to very simply package up any software and its dependencies into a container.
  - Developers are able use any language, version and tooling because they are all packaged in a container, which “standardizes” all that heterogeneity without sacrifice

Docker Advantage

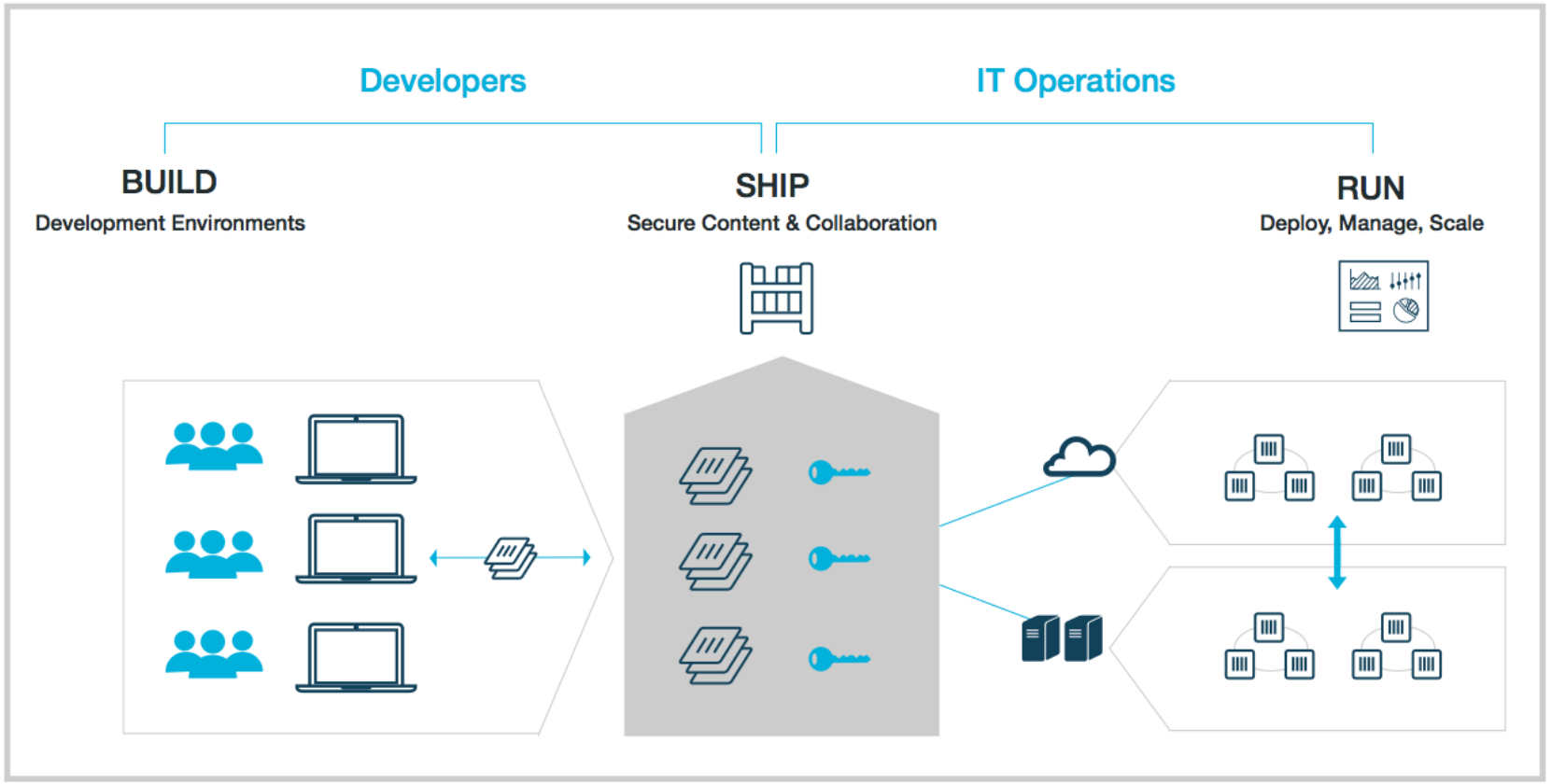
- Portability:
  - They can ship their applications from development, to test and production and the code will work as designed every time.
  - Any differences in the environment did not affect what was inside the container.
  - Nor did they need to change their application to work in production.
  - This was also a boon for IT operations teams as they can now move applications across datacenters for clouds to avoid vendor lock in

Docker Advantage

- Control:
  - Docker “standardizes” your environment while maintaining the heterogeneity your business requires.
  - Docker provides the ability to set the appropriate level of control and flexibility to maintain your service levels, performance and regulatory compliance.
  - IT operations teams can provision, secure, monitor and scale the infrastructure and applications to maintain peak service levels.
  - No two applications or businesses are alike and the Docker allows you to decide how to control your application environment.

Containers as a Service (Caas)

- What is Containers as a Service (CaaS)?
  - It is an IT managed and secured application environment of infrastructure and content where developers can in a self service manner, build and deploy applications.



Containers as a Service (Caas)

- Development and IT operations team collaborate through the registry.
- This is a service in which a library of secure and signed images can be maintained.
- From the registry, developers on the left are able to pull and build software at their pace and then push the content back to the registry once it passes integration testing to save the latest version.
- Depending on the internal processes, the deployment step is either automated with tools or can be manually deployed

Containers as a Service (Caas)

- The IT operations team manage the different vendor contracts for the production infrastructure such as compute, networking and storage.
- These teams are responsible provision the compute resources needed for the application and use the Docker Universal Control Plane to monitor the clusters and applications over time.
- Then, they can move the apps from one cloud to another or scale up or down a service to maintain peak performance

CaaS Characteristics and Considerations

- The needs of developers and operations.
  - Many tools specifically address the functional needs of only one team; however, CaaS breaks the cycle for continuous improvement.
  - To truly gain acceleration in the development to production timeline, you need to address both users along a continuum.
  - Docker provides unique capabilities for each team as well as a consistent API across the entire platform for a seamless transition from one team to the next.

CaaS Characteristics and Considerations

- All stages in the application lifecycle.
  - From continuous integration to delivery and devops, these practices are about eliminating the waterfall development methodology and the lagging innovation cycles with it.
  - By providing tools for both the developer and IT operations, Docker is able to seamlessly support an application from build, test, stage to production.
- Any language
  - Developer agility means the freedom to build with whatever language, version and tooling required for the features they are building at that time.
  - Also, the ability to run multiple versions of a language at the same time provides a greater level of flexibility.
  - Docker allows your team to focus on building the app instead of thinking of how to build an app that works in Docker

CaaS Characteristics and Considerations

- Any operating system.
  - The vast majority of organizations have more than one operating system. Some tools just work better in Linux while others in Windows.
  - Application platforms need to account and support this diversity, otherwise they are solving only part of the problem.
  - Originally created for the Linux community, Docker and Microsoft are bringing forward Windows Server support to address the millions of enterprise applications in existence today and future applications.
- Any infrastructure
  - When it comes to infrastructure, organizations want choice, backup and leverage.
  - Whether that means you have multiple private data centers, a hybrid cloud or multiple cloud providers, the critical component is the ability to move workloads from one environment to another, without causing application issues.
  - The Docker technology architecture abstracts the infrastructure away from the application allowing the application containers to be run anywhere and portable across? any other infrastructure.

CaaS Characteristics and Considerations

- Open APIs, pluggable architecture and ecosystem.
  - A platform isn't really a platform if it is an island to itself.
  - Implementing new technologies is often not possible if you need to re-tool your existing environment first.
  - A fundamental guiding principle of Docker is a platform that is open.
  - Being open means APIs and plugins to make it easy for you to leverage your existing investments and to fit Docker into your environment and processes.
  - This openness invites a rich ecosystem to flourish and provide you with more flexibility and choice in adding specialized capabilities to your CaaaS.

Docker Implementation

- On-Premises:
  - For organizations who need to keep their IP within their network, Docker Trusted Registry and Docker Universal Control Plane can be deployed on-premises or in a VPC and connected to your existing infrastructure and systems like storage, Active Directory/LDAP, monitoring and logging solutions.
  - Trusted Registry provides the ability to store and manage images on your storage infrastructure while also managing role based access control to the images.
  - Universal Control Plane provides visibility across your Docker environment including Swarm clusters, Trusted Registry repositories, containers and multi container applications.

Docker Implementation

- In the Cloud:
  - For organizations who readily use SaaS solutions, Docker Hub and Tutum by Docker provide a registry service and control plane that is hosted and managed by Docker.
  - Hub is a cloud registry service to store and manage your images and users permissions.
  - Tutum by Docker provisions and manages the deployment clusters as well as monitors and manages the deployed applications.
  - Connect to the cloud infrastructure of your choice or bring your own physical node to deploy your application.

Docker Community

- Docker Community is a vibrant community
- **docker** project is our most active with over 16K commits in a month
- **swarm** project which has well over 1,800 commits in a month
- At 963 contributors, **docker** project is currently far ahead of its younger "sibling“ **compose**.
- **compose** has the second highest number of contributors at 96

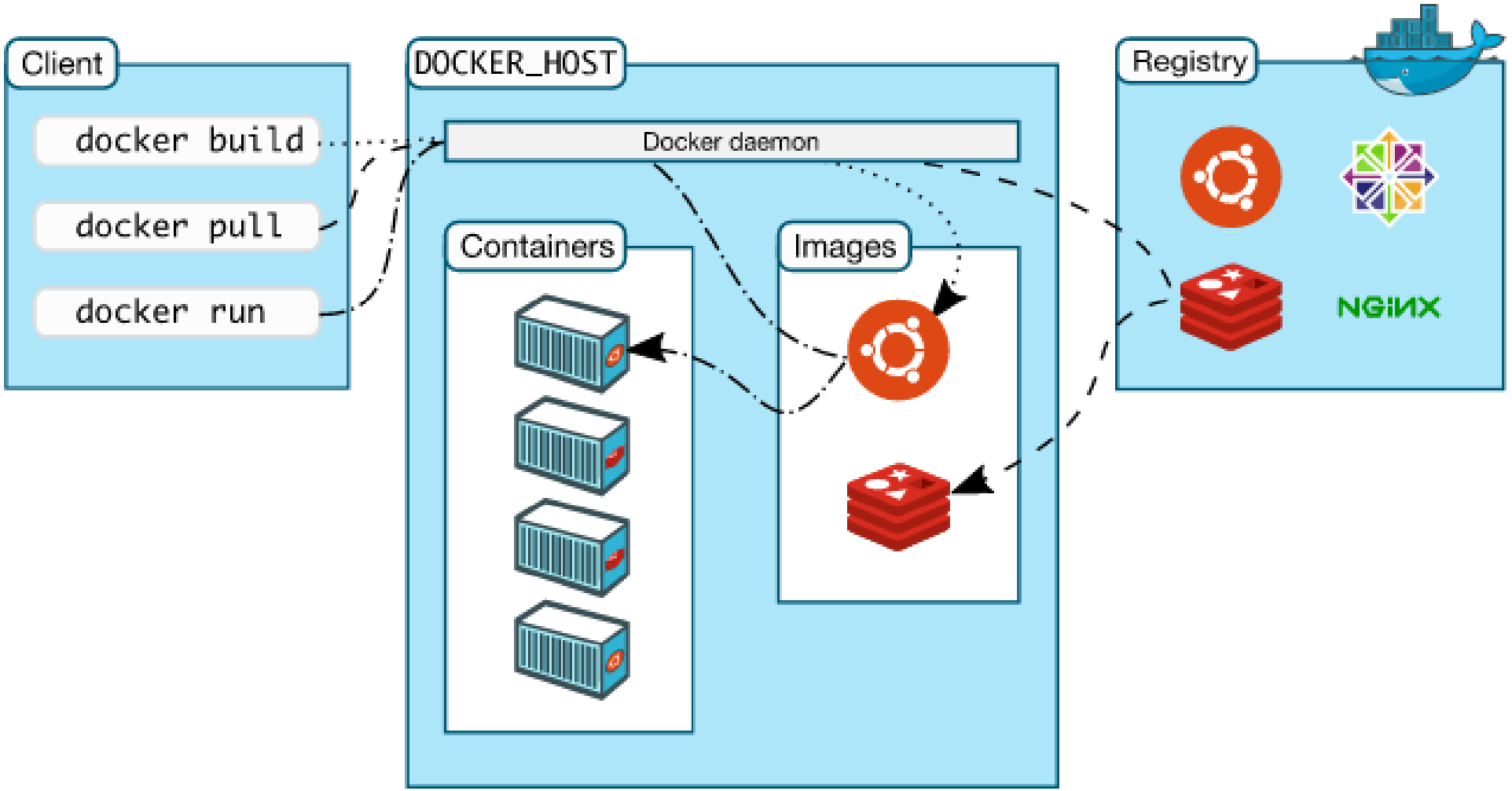
Want to Contribute

- Here is the community guidelines to contribute
  - https://github.com/docker/docker/blob/master/CONTRIBUTING.md#docker-community-guidelines
- There are multiple project under docker you can contribute
  - `docker/docker`
  - `docker/compose`
  - `docker/machine`
  - Docker/swarm
  - And much more
- There is a GIT repo for all of these projects separately
  - https://github.com/docker/dockerhttps://github.com/docker/docker

Want to Contribute

- Here is the community guidelines to contribute
  - https://github.com/docker/docker/blob/master/CONTRIBUTING.md#docker-community-guidelines
- There are multiple project under docker you can contribute
  - `docker/docker`
  - `docker/compose`
  - `docker/machine`
  - Docker/swarm
  - And much more
- There is a GIT repo for all of these projects separately
  - https://github.com/docker/dockerhttps://github.com/docker/docker

Docker Architecture



Docker Daemon

- Docker uses a client-server architecture.
- The Docker *client* talks to the Docker *daemon*, which does the heavy lifting of building, running, and distributing your Docker containers.
- Both the Docker client and the daemon *can* run on the same system, or you can connect a Docker client to a remote Docker daemon.
- The Docker client and daemon communicate via sockets or through a RESTful API.
- the Docker daemon runs on a host machine. The user does not directly interact with the daemon, but instead through the Docker client.

The Docker client

- The Docker client, in the form of the `docker` binary, is the primary user interface to Docker.
- It accepts commands from the user and communicates back and forth with a Docker daemon.

Docker images

- A Docker image is a read-only template.
- For example, an image could contain an Ubuntu operating system with Apache and your web application installed.
- Images are used to create Docker containers.
- Docker provides a simple way to build new images or update existing images, or you can download Docker images that other people have already created.
- Docker images are the **build** component of Docker.

Docker registries

- Docker registries hold images.
- These are public or private stores from which you upload or download images.
- The public Docker registry is provided with the Docker Hub.
- It serves a huge collection of existing images for your use.
- You can create your own private repositories as well
- These can be images you create yourself or you can use images that others have previously created.
- Docker registries are the **distribution** component of Docker.

Docker containers

- Docker containers are similar to a directory.
- A Docker container holds everything that is needed for an application to run.
- Each container is created from a Docker image.
- Docker containers can be run, started, stopped, moved, and deleted.
- Each container is an isolated and secure application platform.
- Docker containers are the **run** component of Docker.

How does a Docker image work?

- Docker images are read-only templates from which Docker containers are launched.
- Each image consists of a series of layers.
- Docker makes use of union file systems to combine these layers into a single image.
- **Unionfs**
  - It is a filesystem service for Linux, FreeBSD and NetBSD which implements a union mount for other file systems.
  - It allows files and directories of separate file systems, known as branches, to be transparently overlaid, forming a single coherent file system.
  - Contents of directories which have the same path within the merged branches will be seen together in a single merged directory, within the new, virtual filesystem.

How does a Docker image work?

- One of the reasons Docker is so lightweight is because of these layers.
- When you change a Docker image-for example, update an application to a new version— a new layer gets built.
- Thus, rather than replacing the whole image or entirely rebuilding, as you may do with a virtual machine, only that layer is added or updated.
- Now you don’t need to distribute a whole new image, just the update, making distributing Docker images faster and simpler.

How does a Docker image work?

- Every image starts from a base image, for example `ubuntu`, a base Ubuntu image, or `fedora`, a base Fedora image.
- You can also use images of your own as the basis for a new image, for example if you have a base Apache image you could use this as the base of all your web application images.

How does a Docker image work?

- Docker images are then built from these base images using a simple, descriptive set of steps we call *instructions*.
- Each instruction creates a new layer in our image.
- Instructions include actions like:
  - Run a command.
  - Add a file or directory.
  - Create an environment variable.
  - What process to run when launching a container from this image.
- These instructions are stored in a file called a `Dockerfile`.
- Docker reads this `Dockerfile` when you request a build of an image, executes the instructions, and returns a final image.

How does a Docker registry work?

- The Docker registry is the store for your Docker images.
- Once you build a Docker image you can *push* it to a public registry such as the one provided by Docker Hub or to your own registry running behind your firewall.
- Using the Docker client, you can search for already published images and then pull them down to your Docker host to build containers from them.
- Docker Hub provides both public and private storage for images.
- Public storage is searchable and can be downloaded by anyone.
- Private storage is excluded from search results and only you and your users can pull images down and use them to build containers.

How does a container work?

- A container consists of an operating system, user-added files, and meta-data.
- As we've seen, each container is built from an image.
- That image tells Docker what the container holds, what process to run when the container is launched, and a variety of other configuration data.
- The Docker image is read-only.
- When Docker runs a container from an image, it adds a read-write layer on top of the image (using a union file system) in which your application can then run.

What happens when you run a container?

Step 1: **Pulls the BASE image :** Docker checks for the presence of the **BASE** image and, if it doesn't exist locally on the host, then Docker downloads it from Docker Hub. If the image already exists, then Docker uses it for the new container.

Step 2: **Creates a new container:** Once Docker has the image, it uses it to create a container.

Step 3: **Allocates a filesystem and mounts a read-write *layer*:** The container is created in the file system and a read-write layer is added to the image.

What happens when you run a container?

- **Step 4: Allocates a network / bridge interface:** Creates a network interface that allows the Docker container to talk to the local host.
- **Step 5: Sets up an IP address:** Finds and attaches an available IP address from a pool.
- **Step 6: Executes a process that you specify:** Runs your application
- **Step 7: Captures and provides application output:** Connects and logs standard input, outputs and errors for you to see how your application is running.

The underlying technology

- In Linux Docker uses
- Namespaces
  - **The pid namespace:** Used for process isolation (PID: Process ID).
  - **The net namespace:** Used for managing network interfaces (NET: Networking).
  - **The ipc namespace:** Used for managing access to IPC resources
  - **The mnt namespace:** Used for managing mount-points (MNT: Mount).
  - **The uts namespace:** Used for isolating kernel and version identifiers.
- Control groups
  - A key to running applications in isolation is to have them only use the resources you want.
  - This ensures containers are good multi-tenant citizens on a host.
  - Control groups allow Docker to share available hardware resources to containers and, if required, set up limits and constraints.
  - For example, limiting the memory available to a specific container.

The underlying technology

- Union file systems
  - Union file systems, or UnionFS, are file systems that operate by creating layers, making them very lightweight and fast.
  - Docker uses union file systems to provide the building blocks for containers. Docker can make use of several union file system variants including: AUFS, btrfs, vfs, and DeviceMapper.
- Container format
  - Docker combines these components into a wrapper we call a container format.
  - The default container format is called `libcontainer`.
  - In the future, Docker may support other container formats, for example, by integrating with BSD Jails or Solaris Zones.