FullStack.Cafe - Kill Your Tech Interview

Q1: What is Docker? ☆

Topics: Docker

Answer:

- Docker is a containerization platform which packages your application and all its dependencies together in the form of containers so as to ensure that your application works seamlessly in any environment be it development or test or production.
- Docker containers, wrap a piece of software in a complete filesystem that contains everything needed to run: code, runtime, system tools, system libraries etc. anything that can be installed on a server.
- This guarantees that the software will always run the same, regardless of its environment.

Q2: What does *Containerization* mean? ☆☆

Topics: DevOps Docker

Answer:

Containerisation is a type of *virtualization* strategy that emerged as an alternative to traditional hypervisor-based virtualization.

In containerization, the operating system is shared by the different containers rather than cloned for each virtual machine. For example Docker provides a container virtualization platform that serves as a good alternative to hypervisor-based arrangements.

Q3: What is the difference between a Docker image and a container? ☆☆

Topics: Docker

Answer:

An instance of an image is called a container. You have an image, which is a set of layers. If you start this image, you have a running container of this image. You can have many running containers of the same image.

You can see all your images with `docker images` whereas you can see your running containers with `docker ps` (and you can see all containers with `docker ps -a`).

So a running instance of an image is a container.

Q4: What is the difference between the `COPY` and `ADD` commands in a Dockerfile? ☆☆

Topics: Docker

Answer:

Although `ADD` and `COPY` are functionally similar, generally speaking, `COPY` is preferred.

That’s because it’s more transparent than ADD. COPY only supports the basic copying of local files into the container, while ADD has some features (like local-only tar extraction and remote URL support) that are not immediately obvious. Consequently, the best use for ADD is local tar file auto-extraction into the image, as in ADD rootfs.tar.xz /.

Q5: What is the difference between CMD and ENTRYPOINT in a Dockerfile? ☆☆

Topics: Docker

Answer:

Both `CMD` and `ENTRYPOINT` instructions define what command gets executed when running a container. There are few rules that describe their co-operation.

1. Dockerfile should specify at least one of `CMD` or `ENTRYPOINT` commands.
2. `ENTRYPOINT` should be defined when using the container as an executable.
3. `CMD` should be used as a way of defining default arguments for an `ENTRYPOINT` command or for executing an ad-hoc command in a container.
4. `CMD` will be overridden when running the container with alternative argumen

Q6: What is Docker image? ☆☆

Topics: Docker

Answer:

Docker image is the source of Docker container. In other words, Docker images are used to create containers. Images are created with the build command, and they’ll produce a container when started with run. Images are stored in a Docker registry such as `registry.hub.docker.com` because they can become quite large, images are designed to be composed of layers of other images, allowing a minimal amount of data to be sent when transferring images over the network.

Q7: What is Docker container? ☆☆

Topics: Docker

Answer:

Docker containers include the application and all of its dependencies, but share the kernel with other containers, running as isolated processes in user space on the host operating system. Docker containers are not tied to any specific infrastructure: they run on any computer, on any infrastructure, and in any cloud.

Q8: What is Docker hub? ☆☆

Topics: Docker

Answer:

Docker hub is a cloud-based registry service which allows you to link to code repositories, build your images and test them, stores manually pushed images, and links to Docker cloud so you can deploy images to your hosts. It provides a centralized resource for container image discovery, distribution and change management, user and team collaboration, and workflow automation throughout the development pipeline.

Q9: Do I lose my data when the Docker container exits? ☆☆

Topics: Docker

Answer:

There is no loss of data when any of your Docker containers exits as any of the data that your application writes to the disk in order to preserve it. This will be done until the container is explicitly deleted. The file system for the Docker container persists even after the Docker container is halted.

Q10: What are the various states that a Docker container can be in at any given point in time? ☆☆

Topics: Docker

Answer:

There are four states that a Docker container can be in, at any given point in time. Those states are as given as follows:

- Running
- Paused
- Restarting
- Exited

Q11: Is there a way to identify the status of a Docker container? ☆☆

Topics: Docker

Answer:

We can identify the status of a Docker container by running the command

```
docker ps –a
```

which will in turn list down all the available docker containers with its corresponding statuses on the host. From there we can easily identify the container of interest to check its status correspondingly.

Q12: What is Build Cache in Docker? ☆☆

Topics: Docker

Answer:

When we build an Image, Docker will process each line in Dockerfile. It will execute the commands on each line in the order that is mentioned in the file. But at each line, before running any command, Docker will check if there is already an existing image in its cache that can be reused rather than creating a new image.

Q13: What are the most common instructions in Dockerfile? ☆☆

Topics: Docker

Answer:

Some of the common instructions in Dockerfile are as follows:

- **FROM**: We use FROM to set the base image for subsequent instructions. In every valid Dockerfile, FROM is the first instruction.
- **LABEL**: We use LABEL to organize our images as per project, module, licensing etc. We can also use LABEL to help in automation.
  In LABEL we specify a key value pair that can be later used for programmatically handling the Dockerfile.
- **RUN**: We use RUN command to execute any instructions in a new layer on top of the current image. With each RUN command we add something on top of the image and use it in subsequent steps in Dockerfile.
- **CMD**: We use CMD command to provide default values of an executing container. In a Dockerfile, if we include multiple CMD commands, then only the last instruction is used.

Q14: What type of applications - Stateless or Stateful are more suitable for Docker Container? ☆☆

Topics: Docker

Answer:

It is preferable to create Stateless application for Docker Container. We can create a container out of our application and take out the configurable state parameters from application. Now we can run same container in Production as well as QA environments with different parameters. This helps in reusing the same Image in different scenarios. Also a stateless application is much easier to scale with Docker Containers than a stateful application.

Q15: What is the difference between ‘docker run’ and ‘docker create’? ☆☆

Topics: Docker

Answer:

The primary difference is that using **‘docker create’** creates a container in a stopped state.

Bonus point: You can use **‘docker create’** and store an outputed container ID for later use. The best way to do it is to use **‘docker run’** with **--cidfile FILE_NAME** as running it again won’t allow to overwrite the file. A good practice is to keep well ogranised directory structure: /containers/web/server1/ws.cid containers/web/server3/ws.cid

Q16: How to build envrionment-agnostic systems with Docker? ☆☆

Topics: Docker

Answer:

There are three main features helping to achieve that:

- Volumes
- Environment variable injection
- Read-only file systems

Q17: Can you remove (‘docker rm’) a container that is paused? ☆☆

Topics: Docker

Answer:

No, to remove a container it must be stopped first.

Q18: When would you use ‘docker kill’ or ‘docker rm -f’? ☆☆

Topics: Docker

Answer:

If you must stop the container really quickly… (someone pushed something to production on Friday evening?… ;))

Q19: What’s the difference between a repository and a registry? ☆☆

Topics: Docker

Answer:

- **Docker registry** is a service for hosting and distributing images (the default one is the Docker Hub).
- **Docker repository** is a collection of related Docker images (the same name but with different tags).

Q20: How to link containers? ☆☆

Topics: Docker

Answer:

The simplest way is to use network port mapping. There’s also the **- -link** flag which is deprecated.

FullStack.Cafe - Kill Your Tech Interview

Q1: Explain a use case for *Docker* ☆☆☆

Topics: DevOps Docker

Answer:

- Docker a low overhead way to run virtual machines on your local box or in the cloud. Although they're not strictly distinct machines, nor do they need to boot an OS, they give you many of those benefits.
- Docker can encapsulate legacy applications, allowing you to deploy them to servers that might not otherwise be easy to setup with older packages & software versions.
- Docker can be used to build test boxes, during your deploy process to facilitate continuous integration testing.
- Docker can be used to provision boxes in the cloud, and with swarm you can orchestrate clusters too.

Q2: Explain basic Docker usage workflow ☆☆☆

Topics: Docker

Answer:

1. Everything starts with the **Dockerfile**. The Dockerfile is the source code of the Image.
2. Once the Dockerfile is created, you build it to create the **image** of the container. The image is just the "compiled version" of the "source code" which is the Dockerfile.
3. Once you have the image of the container, you should redistribute it using the **registry**. The registry is like a git repository -- you can push and pull images.
4. Next, you can use the image to run **containers**. A running container is very similar, in many aspects, to a virtual machine (but without the hypervisor).

```
    +------------+  docker build   +--------------+  docker run -dt   +-----------+  docker exec -it   +-
-----+
    | Dockerfile | --------------> |    Image     | ---------------> | Container | -----------------> |
Bash |
    +------------+                 +--------------+                  +-----------+                    +-
-----+
                                     ^
                                     | docker pull
                                     |
                                   +--------------+
                                   |   Registry   |
                                   +--------------+
```

Q3: What is the difference between Docker Image and Layer? ☆☆☆

Topics: Docker

Answer:

- **Image**: A Docker image is built up from a series of **read-only** layers
- **Layer**: Each layer represents an instruction in the image’s Dockerfile.

The below Dockerfile contains four commands, each of which creates a layer.

```
FROM ubuntu:15.04
COPY . /app
RUN make /app
CMD python /app/app.py
```

Importantly, each layer is only a set of differences from the layer before it.

Q4: What is virtualisation? ☆☆☆

Topics: Docker

Answer:

In its conceived form, **virtualisation** was considered a method of logically dividing mainframes to allow multiple applications to run simultaneously. However, the scenario drastically changed when companies and open source communities were able to provide a method of handling the privileged instructions in one way or another and allow for multiple operating systems to be run simultaneously on a single x86 based system.

The net effect is that virtualization allows you to run two completely different OS on same hardware. Each guest OS goes through all the process of bootstrapping, loading kernel etc. You can have very tight security, for example, guest OS can't get full access to host OS or other guests and mess things up.

The virtualization method can be categorized based on how it mimics hardware to a guest operating system and emulates guest operating environment. Primarily, there are three types of virtualization:

- Emulation
- Paravirtualization
- Container-based virtualization

Q5: What is Hypervisor? ☆☆☆

Topics: Docker

Answer:

The **hypervisor** handles creating the virtual environment on which the guest virtual machines operate. It supervises the guest systems and makes sure that resources are allocated to the guests as necessary. The hypervisor sits in between the physical machine and virtual machines and provides virtualization services to the virtual machines. To realize it, it intercepts the guest operating system operations on the virtual machines and emulates the operation on the host machine's operating system.

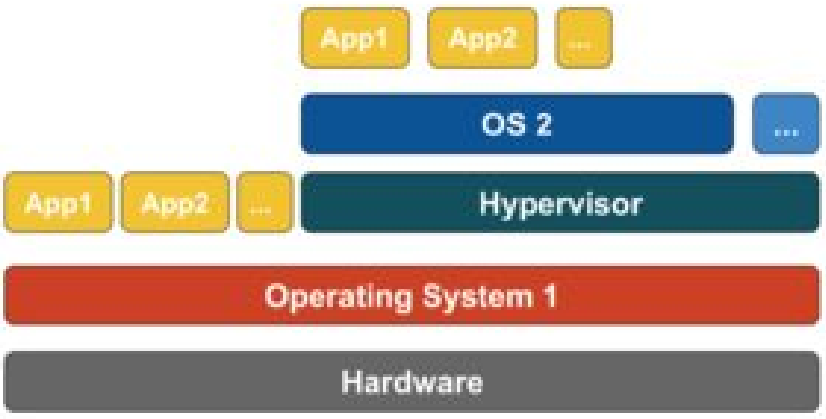
The rapid development of virtualization technologies, primarily in cloud, has driven the use of virtualization further by allowing multiple virtual servers to be created on a single physical server with the help of hypervisors, such as Xen, VMware Player, KVM, etc., and incorporation of hardware support in commodity processors, such as Intel VT and AMD-V.

Q6: Could you explain what is Emulation? ☆☆☆

Topics: Docker

Answer:

Emulation is a type of **virtualization**. **Emulation**, also known as **full virtualization** runs the virtual machine OS kernel entirely in software. The hypervisor used in this type is known as Type 2 hypervisor. It is installed on the top of host operating system which is responsible for translating guest OS kernel code to software instructions. The translation is done entirely in software and requires no hardware involvement. Emulation makes it possible to run any non-modified operating system that supports the environment being emulated. The downside of this type of virtualization is an additional system resource overhead that leads to decrease in performance compared to other types of virtualizations.



Examples in this category include VMware Player, VirtualBox, QEMU, Bochs, Parallels, etc.

Q7: Should I use Vagrant or Docker for creating an isolated environment? ☆☆☆

Topics: Docker

Answer:

The short answer is that if you want to manage machines, you should use Vagrant. And if you want to build and run applications environments, you should use Docker.

Vagrant is a tool for managing virtual machines. Docker is a tool for building and deploying applications by packaging them into lightweight containers. A container can hold pretty much any software component along with its dependencies (executables, libraries, configuration files, etc.), and execute it in a guaranteed and repeatable runtime environment. This makes it very easy to build your app once and deploy it anywhere - on your laptop for testing, then on different servers for live deployment, etc.

Q8: What is the difference between CMD and ENTRYPOINT in a Dockerfile? ☆☆☆

Topics: Docker

Answer:

Both `CMD` and `ENTRYPOINT` instructions define what command gets executed when running a container. There are few rules that describe their co-operation.

1. Dockerfile should specify at least one of `CMD` or `ENTRYPOINT` commands.
2. `ENTRYPOINT` should be defined when using the container as an executable.
3. `CMD` should be used as a way of defining default arguments for an `ENTRYPOINT` command or for executing an ad-hoc command in a container.

4. `CMD` will be overridden when running the container with alternative argumen

Docker has a default entrypoint which is `/bin/sh -c` but does not have a default command. When you run docker like this:

```
docker run -i -t ubuntu bash
```

The entrypoint is the default `/bin/sh -c`, the image is `ubuntu` and the command is `bash`. The command is run via the entrypoint. i.e., the actual thing that gets executed is `/bin/sh -c bash`. This allowed Docker to implement `RUN` quickly by relying on the shell's parser. Later on, people asked to be able to customize this, so `ENTRYPOINT` and `--entrypoint` were introduced.

Q9: What is the difference between “expose” and “publish” in Docker? ☆☆☆

Topics: Docker

Answer:

In Docker networking, there are two different mechanisms that directly involve network ports: exposing and publishing ports. This applies to the default bridge network and user-defined bridge networks.

- You expose ports using the `EXPOSE` keyword in the Dockerfile or the `--expose` flag to docker run. **Exposing ports is a way of documenting which ports are used, but does not actually map or open any ports. Exposing ports is optional**.

- You publish ports using the `--publish` or `--publish-all` flag to `docker run`. This tells Docker which ports to open on the container’s network interface. When a port is published, it is mapped to an available high-order port (higher than `30000`) on the host machine, unless you specify the port to map to on the host machine at runtime. **You cannot specify the port to map to on the host machine when you build the image (in the Dockerfile), because there is no way to guarantee that the port will be available on the host machine where you run the image**.

Q10: Docker Compose vs. Dockerfile - which is better? ☆☆☆

Topics: Docker

Answer:

The answer is neither.

Docker Compose (herein referred to as compose) is a tool for defining and running multi-container Docker applications and will use the Dockerfile if you add the build command to your project's `docker-compose.yml`. Your Docker workflow should be to build a suitable `Dockerfile` for each image you wish to create, then use compose to assemble the images using the `build` command.

You can specify the path to your individual Dockerfiles using `build /path/to/dockerfiles/blah` where `/path/to/dockerfiles/blah` is where blah's `Dockerfile` lives.

In your workflow, you could add a Dockerfile for each part of the system and configure it that each part could run individually. Then you add a docker-compose.yml to bring them together and link them.

Q11: What is Docker Swarm? ☆☆☆

Topics: Docker

Answer:

Docker Swarm is native clustering for Docker. It turns a pool of Docker hosts into a single, virtual Docker host. Docker Swarm serves the standard Docker API, any tool that already communicates with a Docker daemon can use Swarm to transparently scale to multiple hosts.

Q12: What is the preferred way of removing containers - ‘docker rm -f’ or ‘docker stop’ then followed by a ‘docker rm’? ☆☆☆

Topics: Docker

Answer:

The best and the preferred way of removing containers from Docker is to use the `docker stop`, as it will allow sending a `SIG_HUP` signal to its recipients giving them the time that is required to perform all the finalization and cleanup tasks. Once this activity is completed, we can then comfortably remove the container using the `docker rm` command from Docker and thereby updating the docker registry as well.

Q13: What exactly do you mean by “Dockerized node”? Can this node be on-premises or in the cloud? ☆☆☆

Topics: Docker

Answer:

A Dockerized node is anything i.e a bare metal server, VM or public cloud instance that has the Docker Engine installed and running on it.

Docker can manage nodes that exist on-premises as well as in the cloud. Docker Datacenter is an on-premises solution that enterprises use to create, manage, deploy and scale their applications and comes with support from the Docker team. It can manage hosts that exist in your datacenter as well as in your virtual private cloud or public cloud provider (AWS, Azure, Digital Ocean, SoftLayer etc.).

Q14: How can we control the startup order of services in Docker compose? ☆☆☆

Topics: Docker

Problem:

I have a container which depends on a redis databse container. However, it takes longer for redis to load in memory which causes the first container to exit. I was wondering if there is a better alternative as I would try to avoid the wait for it script?

Solution:

You can control the order of service startup and shutdown with the `depends_on` option that expresses dependency between services. Service dependencies cause the following behaviors:

- `docker-compose up` starts services in dependency order. In the following example, `db` and `redis` are started before `web`.

- `docker-compose up SERVICE` automatically includes `SERVICE`'s dependencies. In the following example, `docker-compose up web` also creates and starts `db` and `redis`.
- `docker-compose stop` stops services in dependency order. In the following example, `web` is stopped before `db` and `redis`.

```
version: '3'
services:
  web:
    build: .
    depends_on:
      - db
      - redis
  redis:
    image: redis
  db:
    image: postgres
```

You can also specify a `healthcheck` in your redis container and add `condition: service_healthy` to your `depends_on` field. This works since compose 2.1:

```
    version: "2.1"
    services:
        web:
        build: .
    ports:
        -"80:8000"
    depends_on:
        "db":
        condition: service_healthy
    command: ["python", "app.py"]
    db:
        image: postgres
```

Q15: How will you monitor Docker in production? ☆☆☆

Topics: Docker

Answer:

Docker provides tools like docker stats and docker events to monitor Docker in production. We can get reports on important statistics with these commands.

- **Docker stats**: When we call docker stats with a container id, we get the CPU, memory usage etc of a container. It is similar to top command in Linux.

- **Docker events**: Docker events are a command to see the stream of activities that are going on in Docker daemon.

  Some of the common Docker events are: attach, commit, die, detach, rename, destroy etc. We can also use various options to limit or filter the events that we are interested in.

Q16: What is the purpose of EXPOSE command in Dockerfile? ☆☆☆

Topics: Docker

Answer:

When writing your Dockerfiles, the instruction EXPOSE tells Docker the running container listens on specific network ports. This acts as a kind of port mapping documentation that can then be used when publishing the ports.

```
EXPOSE <port> [<port>/<protocol>...]
```

or

```
docker run --expose=1234 my_app
```

But EXPOSE will not allow communication via the defined ports to containers outside of the same network or to the host machine. To allow this to happen you need to *publish* the ports.

Q17: Can you create containers wihout their own PID namespace? ☆☆☆

Topics: Docker

Answer:

Yes.

Q18: What is the default CPU limit set for a container? ☆☆☆

Topics: Docker

Answer:

No limit by default, it can consume up to 100% of the CPU.

Q19: What happens if you add more than one CMD instruction to a Dockerfile? ☆☆☆

Topics: Docker

Answer:

Only the last CMD will take effect.

Q20: What is the difference between Kubernetes and Docker? ☆☆☆

Topics: Docker Kubernetes DevOps

Problem:

And what are they used for?

Solution:

Docker and Kubernetes are complementary.

- **Docker** provides an open standard for packaging and distributing containerized applications, while
- **Kubernetes** provides for the orchestration and management of distributed, containerized applications created with Docker.

In other words, Kubernetes provides the infrastructure needed to deploy and run applications built with Docker.

FullStack.Cafe - Kill Your Tech Interview

Q1: Explain when to use Docker vs Docker Compose vs Docker Swarm vs Kubernetes ☆☆☆

Topics: Kubernetes Docker

Answer:

- **Docker** is a container engine, it makes you build and run usually no more than one container at most, locally on your PC for development purposes.
- **Docker Compose** is a Docker utility to run multiple containers and let them share volumes and networking via the docker engine features, runs locally to emulate service composition and remotely on clusters. Docker Compose is mostly used as a helper when you want to start multiple Docker containers and don't want to start each one separately using `docker run ...`.
- **Docker Swarm** is for running and connecting containers on *multiple* hosts. It does things like scaling, starting a new container when one crashes, networking containers.
- **Kubernetes** is a container orchestration platform, it takes care of running containers and enhancing the engine features so that containers can be composed and scaled to serve complex applications (sort of PaaS, managed by you or cloud provider). Kubernetes' goal is very similar to that for Docker Swarm but it's developer by Google.

Q2: Which problems does a *Container Orchestration* solve? ☆☆☆

Topics: Docker Kubernetes DevOps

Answer:

Containers run in an isolated process (usually in it's own namespace). This means that by default the container will not be aware of other containers. Additionally, it will not be aware of the systems files, network interfaces, and processes. While this can greatly help with portability of the software it does not solve several production issues such as microservices, container discovery, scalability, disaster recovery, or upgrades.

Adding a container orchestrator can greatly reduce the complexity in production as these tools are designed to resolve the issues outlined above. For example, Kubernetes is built to allow containers to be linked together, deploy containers across an entire network, scale and load balance the network based on container resource consumption, and allow upgrades of individual containers with no downtime.

If you are only running a single container or two containers together you are correct in that an orchestrator may be unnecessary and add unneeded complexity.

Q3: Explain what are some Pods usage patterns? ☆☆☆

Topics: Kubernetes Docker

Answer:

Pods can be used in two main ways:

- **Pods that run a single container.** The simplest and most common Pod pattern is a single container per pod, where the single container represents an entire application. In this case, you can think of a Pod as a

wrapper.
- **Pods that run multiple containers that need to work together.** Pods with multiple containers are primarily used to support colocated, co-managed programs that need to share resources. These colocated containers might form a single cohesive unit of service—one container serving files from a shared volume while another container refreshes or updates those files. The Pod wraps these containers and storage resources together as a single manageable entity.

Each Pod is meant to run a single instance of a given application. If you want to run multiple instances, you should use one Pod for each instance of the application. This is generally referred to as *replication*. Replicated Pods are created and managed as a group by a controller, such as a Deployment.

Q4: How is *Container* different from a *Virtual Machine*? ☆☆☆☆

Topics: DevOps Docker

Answer:

- Unlike a virtual machine, a container does not need to boot the operating system kernel, so containers can be created in less than a second. This feature makes container-based virtualization unique and desirable than other virtualization approaches.
- Since container-based virtualization adds little or no overhead to the host machine, container-based virtualization has near-native performance
- For container-based virtualization, no additional software is required, unlike other virtualizations.
- All containers on a host machine share the scheduler of the host machine saving need of extra resources.
- Container states (Docker or LXC images) are small in size compared to virtual machine images, so container images are easy to distribute.
- Resource management in containers is achieved through cgroups. Cgroups does not allow containers to consume more resources than allocated to them.

Q5: How virtualization works at low level? ☆☆☆☆

Topics: Docker

Answer:

VM manager takes over the CPU ring 0 (or the "root mode" in newer CPUs) and intercepts all privileged calls made by guest OS to create illusion that guest OS has its own hardware. Fun fact: Before 1998 it was thought to be impossible to achieve this in x86 architecture because there was no way to do this kind of interception. The folks at VMWare were the first who had an idea to rewrite the executable bytes in memory for privileged calls of guest OS to achieve this.

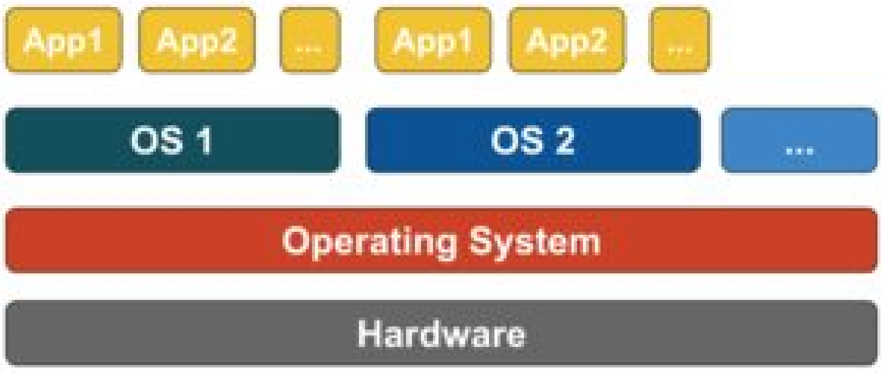
The net effect is that virtualization allows you to run two completely different OS on same hardware. Each guest OS goes through all the process of bootstrapping, loading kernel etc. You can have very tight security, for example, guest OS can't get full access to host OS or other guests and mess things up.

Q6: What is Paravirtualization? ☆☆☆☆

Topics: Docker

Answer:

Paravirtualization, also known as Type 1 hypervisor, runs directly on the hardware, or “bare-metal”, and provides virtualization services directly to the virtual machines running on it. It helps the operating system, the virtualized hardware, and the real hardware to collaborate to achieve optimal performance. These hypervisors typically have a rather small footprint and do not, themselves, require extensive resources.



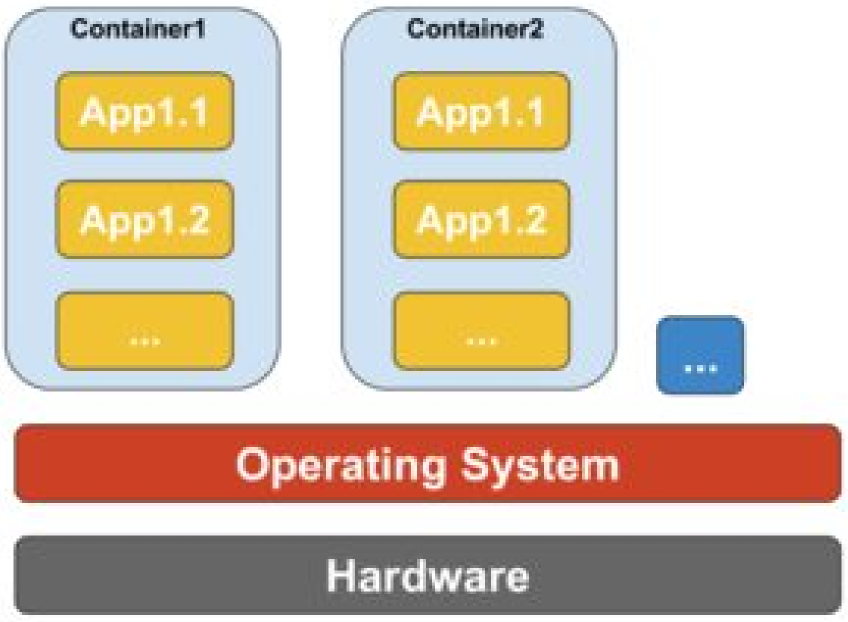
Examples in this category include Xen, KVM, etc.

Q7: How is Docker different from a virtual machine? ☆☆☆☆

Topics: Docker

Answer:

Docker isn't a virtualization methodology. It relies on other tools that actually implement **container-based virtualization** or operating system level virtualization. For that, Docker was initially using LXC driver, then moved to `libcontainer` which is now renamed as `runc`. Docker primarily focuses on automating the deployment of applications inside application containers. Application containers are designed to package and run a single service, whereas system containers are designed to run multiple processes, like virtual machines. So, Docker is considered as a container management or application deployment tool on containerized systems.



- Unlike a virtual machine, a container does not need to boot the operating system kernel, so containers can be created in less than a second. This feature makes container-based virtualization unique and desirable than other virtualization approaches.
- Since container-based virtualization adds little or no overhead to the host machine, container-based virtualization has near-native performance

- For container-based virtualization, no additional software is required, unlike other virtualizations.
- All containers on a host machine share the scheduler of the host machine saving need of extra resources.
- Container states (Docker or LXC images) are small in size compared to virtual machine images, so container images are easy to distribute.
- Resource management in containers is achieved through cgroups. Cgroups does not allow containers to consume more resources than allocated to them. However, as of now, all resources of host machine are visible in virtual machines, but can't be used. This can be realized by running top or htop on containers and host machine at the same time. The output across all environments will look similar.

Q8: Is it possible to generate a Dockerfile from an image? ☆☆☆☆

Topics: Docker

Answer:

To understand how a docker image was built, use the command:

```
$ docker history --no-trunc <IMAGE_ID>
```

You can build a docker file from an image, but it will not contain everything you would want to fully understand how the image was generated. Reasonably what you can extract is the:

- `MAINTAINER`,
- `ENV`,
- `EXPOSE`,
- `VOLUME`,
- `WORKDIR`,
- `ENTRYPOINT`,
- `CMD`,
- `ONBUILD`

parts of the dockerfile.

Q9: Can you explain dockerfile ONBUILD instruction? ☆☆☆☆

Topics: Docker

Answer:

The `ONBUILD` instruction adds to the image a trigger instruction to be executed at a later time, when the image is used as the base for another build. This is useful if you are building an image which will be used as a base to build other images, for example an application build environment or a daemon which may be customized with user-specific configuration.

Q10: What are the different kinds of namespaces available in a Container? ☆☆☆☆

Topics: Docker

Answer:

In a Container we have an isolated environment with namespace for each resource that a kernel provides. There are mainly six types of namespaces in a Container.

- **UTS Namespace**: UTS stands for Unix Timesharing System. In UTS namespace every container gets its own hostname and domain name.
- **Mount Namespace**: This namespace provides its own file system within a container. With this namespace we get root like / in the file system on which rest of the file structure is based.
- **PID Namespace**: This namespace contains all the processes that run within a Container. We can run ps command to see the processes that are running within a Docker container.
- **IPC Namespace**: IPC stands for Inter Process Communication. This namespace covers shared memory, semaphores, named pipes etc resources that are shared by processes. The items in this namespace do not cross the container boundary.
- **User Namespace**: This namespace contains the users and groups that are defined within a container.
- **Network Namespace**: With this namespace, container provides its own network resources like- ports, devices etc. With this namespace, Docker creates an independent network stack within each container.

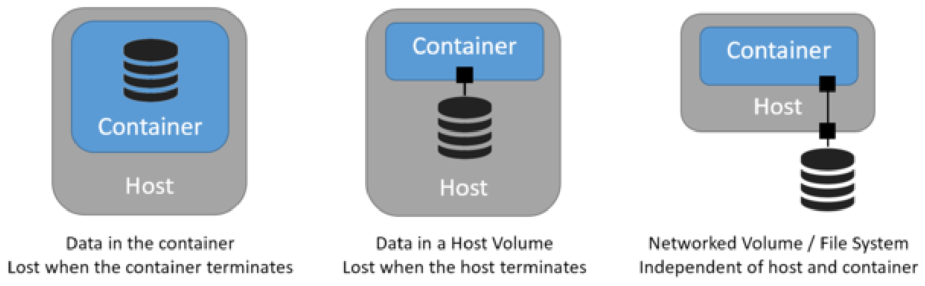
Q11: Is it good practice to run stateful applications on Docker? What are the scenarios where Docker best fits in? ☆☆☆☆

Topics: Docker

Answer:

he problem with statefull docker aplications is that they by default store their state (data) in the containers filesystem. Once you update your software version or want to move to another machine its hard to retrieve the data from there.

What you need to do is bind a volume to the container and store any data in the volume.


Container
Host
Data in the container
Lost when the container terminates
Container
Host
Data in a Host Volume
Lost when the host terminates
Container
Host
Networked Volume / File System
Independent of host and container

if you run your container with: `docker run -v hostFolder:/containerfolder` any changes to `/containerfolder` will be persisted on the `hostfolder`. Something similar can be done with a nfs drive. Then you can run you application on any host machine and the state will be saved in the nfs drive.

Q12: What is an orphant volume and how to remove it? ☆☆☆☆

Topics: Docker

Answer:

An orphant volume is a volume without any containers attached to it. Prior Docker v. 1.9 it was very problematic to remove it.

Q13: When you limit the memory for a container, does it reserve (guarantee) the memory? ☆☆☆☆

Topics: Docker

Answer:

No, it only protects from overconsumption.

Q14: What is the difference between Docker RUN, CMD and ENTRYPOINT? ☆☆☆☆

Topics: Docker

Answer:

CMD does not execute anything at build time, but specifies the intended command for the image.
RUN actually runs a command and commits the result.
If you would like your container to run the same executable every time, then you should consider using **ENTRYPOINT** in combination with CMD.

Q15: Can you run Docker containers natively on Windows? ☆☆☆☆

Topics: Docker

Answer:

With Windows Server 2016 you can run Docker containers natively on Windows, and with Windows Nano Server you’ll have a lightweight OS to run inside containers, so you can run .NET apps on their native platform.

Q16: Why do we need Kubernetes (and other orchestrators) above containers? ☆☆☆☆

Topics: Kubernetes Docker

Answer:

In the quality assurance (QA) environments, we can get away with running containers on a single host to develop and test applications. However, ***when we go to production, we do not have the same liberty***, as we need to ensure that our applications:

- Are fault-tolerant
- Can scale, and do this on-demand
- Use resources optimally
- Can discover other applications automatically, and communicate with each other
- Are accessible from the external world
- Can update/rollback without any downtime.

Container orchestrators are the tools which group hosts together to form a cluster, and help us fulfill the requirements mentioned above.

Nowadays, there are many container orchestrators available, such as:

- **Docker Swarm:** Docker Swarm is a container orchestrator provided by Docker, Inc. It is part of Docker Engine.

- **Kubernetes:** Kubernetes was started by Google, but now, it is a part of the Cloud Native Computing Foundation project.
- **Mesos Marathon:** Marathon is one of the frameworks to run containers at scale on Apache Mesos.
- **Amazon ECS:** Amazon EC2 Container Service (ECS) is a hosted service provided by AWS to run Docker containers at scale on its infrastructrue.
- **Hashicorp Nomad:** Nomad is the container orchestrator provided by HashiCorp.

Q17: How does Docker run containers in non-Linux systems? ☆☆☆☆☆

Topics: Docker

Answer:

The concept of a container is made possible by the namespaces feature added to Linux kernel version 2.6.24. The container adds its ID to every process and adding new access control checks to every system call. It is accessed by the clone() system call that allows creating separate instances of previously-global namespaces.

If containers are possible because of the features available in the Linux kernel, then the obvious question is that how do non-Linux systems run containers. Both Docker for Mac and Windows use **Linux VMs** to run the containers. Docker Toolbox used to run containers in Virtual Box VMs. But, the latest Docker uses Hyper-V in Windows and Hypervisor.framework in Mac.

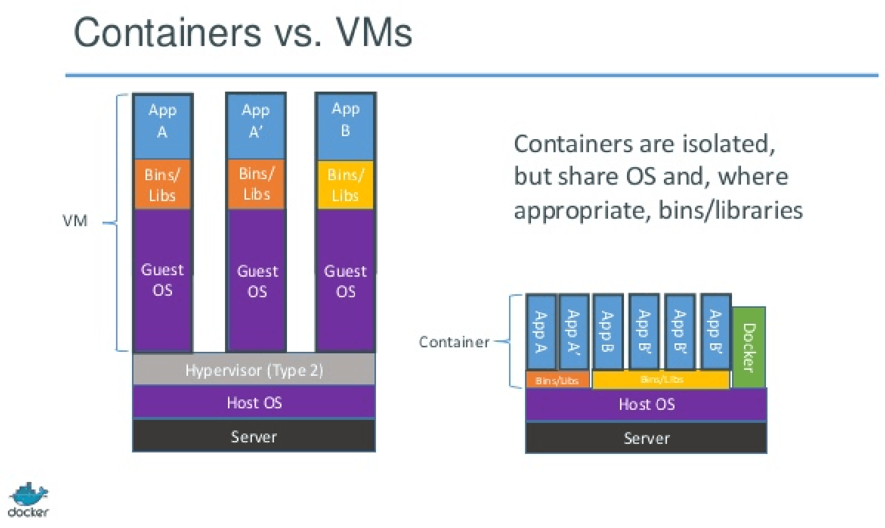
Q18: How containers works at low level? ☆☆☆☆☆

Topics: Docker

Answer:

Around 2006, people including some of the employees at Google implemented new Linux kernel level feature called namespaces (however the idea long before existed in FreeBSD). One function of the OS is to allow sharing of global resources like network and disk to processes. What if these global resources were wrapped in namespaces so that they are visible only to those processes that run in the same namespace? Say, you can get a chunk of disk and put that in namespace X and then processes running in namespace Y can't see or access it. Similarly, processes in namespace X can't access anything in memory that is allocated to namespace Y. Of course, processes in X can't see or talk to processes in namespace Y. This provides kind of virtualization and isolation for global resources.

This is how **Docker** works: Each container runs in its own namespace but uses exactly the same kernel as all other containers. The isolation happens because kernel knows the namespace that was assigned to the process and during API calls it makes sure that process can only access resources in its own namespace.



Q19: Name some limitations of containers vs VM ☆☆☆☆☆

Topics: Docker

Answer:

Just to name a few:

- You can't run completely different OS in containers like in VMs. However you *can* run different distros of Linux because they do share the same kernel. The isolation level is not as strong as in VM. In fact, there was a way for "guest" container to take over host in early implementations.
- Also you can see that when you load new container, the entire new copy of OS doesn't start like it does in VM.
- All containers share the same kernel. This is why containers are light weight.
- Also unlike VM, you don't have to pre-allocate significant chunk of memory to containers because we are not running new copy of OS. This enables to run thousands of containers on one OS while sandboxing them which might not be possible to do if we were running separate copy of OS in its own VM.

Q20: Why Docker compose does not wait for a container to be ready before moving on to start next service in dependency order? ☆☆☆☆☆

Topics: Docker

Answer:

Compose always starts and stops containers in dependency order, where dependencies are determined by `depends_on`, `links`, `volumes_from`, and `network_mode`: "service:...".

However, for startup Compose does not wait until a container is “ready” (whatever that means for your particular application) - only until it’s running. There’s a good reason for this:

- The problem of waiting for a database (for example) to be ready is really just a subset of a much larger problem of distributed systems. In production, your database could become unavailable or move hosts at any time. Your application needs to be resilient to these types of failures.
- To handle this, design your application to attempt to re-establish a connection to the database after a failure. If the application retries the connection, it can eventually connect to the database.

- The best solution is to perform this check in your application code, both at startup and whenever a connection is lost for any reason.

FullStack.Cafe - Kill Your Tech Interview

Q1: How do I transfer a Docker image from one machine to another one without using a repository, no matter private or public? ☆☆

Topics: Docker

Answer:

You will need to save the Docker image as a tar file:

```
docker save - o <path for generated tar file> <image name>
```

Then copy your image to a new system with regular file transfer tools such as `cp` or `scp`. After that you will have to load the image into Docker:

```
docker load -i <path to image tar file>
```

Q2: How to use Docker with multiple environments? ☆☆☆☆☆

Topics: Docker

Answer:

You'll almost certainly want to make changes to your app configuration that are more appropriate to a live environment. These changes may include:

- Removing any volume bindings for application code, so that code stays inside the container and can't be changed from outside
- Binding to different ports on the host
- Setting environment variables differently (e.g., to decrease the verbosity of logging, or to enable email sending)
- Specifying a restart policy (e.g., restart: always) to avoid downtime
- Adding extra services (e.g., a log aggregator)

For this reason, you'll probably want to define an additional Compose file, say `production.yml`, which specifies production-appropriate configuration. This configuration file only needs to include the changes you'd like to make from the original Compose file.

```
docker-compose -f docker-com
```

Q3: Why did Docker jump from version 1.13 to 17.03? ☆☆☆☆☆

Topics: Docker

Answer:

Starting with version 17.03 Docker is on a monthly release cycle and uses a new **YY.MM** versioning scheme to reflect this.

Q4: Can you explain a relationship between *container runtime* and *container orchestration*? ☆☆☆☆☆

Topics: Docker Kubernetes DevOps

Answer:

- A **container runtime** is the part of your container environment in charge of the creation and basic features of your containers. The obvious one is Docker, but you can also find Containerd, CRI-O and other as runtime for your containers.

- An **orchestrator**, by contrast, will not exactly create your container (ie, an orchestrator is not the technology used to create them). An orchestrator will use your container runtime to manage them. In a way, it's a layer around your container runtime wich allow finer usages, such as scaling, multi host deployments, load balancing...

The most famous orchestrators would be Kurbenetes and Docker Swarm.

Q5: What's the difference between `pm2` and `pm2-runtime` and when to use one? ☆☆☆☆☆

Topics: Node.js Docker

Answer:

The main difference between pm2 and pm2-runtime is

- `pm2-runtime` is designed for Docker container which keeps an application in the foreground and keep the container running,
- `pm2` is designed for normal usage where you send or run the application in the background.

In simple words, the container's life is the life of `CMD` or `entrypoint`.