Differences

Kill vs Stop

In Docker, the `docker stop` and `docker kill` commands are used to stop running containers, but they have different behaviors and implications.

The `docker stop` command sends a `SIGTERM` signal to the container's main process, requesting it to stop gracefully. The container is given a grace period of 10 seconds by default (can be customized with `--time` option), during which it can perform cleanup operations and exit its processes cleanly. If the container does not stop within the grace period, Docker will send a `SIGKILL` signal to force it to stop.

On the other hand, the `docker kill` command sends a `SIGKILL` signal to the container's main process, immediately terminating it and all of its child processes. This command does not give the container a chance to perform any cleanup operations or shutdown its processes gracefully, which can result in data loss or corruption.

In summary, `docker stop` is a more gentle way of stopping a container, giving it a chance to shut down gracefully and perform cleanup operations, while `docker kill` is a more forceful way of stopping a container, immediately terminating its processes without giving them a chance to clean up.

It's important to note that the choice between `docker stop` and `docker kill` depends on the situation and the needs of your application. For example, if you need to gracefully shut down your application to save data and release resources, you should use `docker stop`. However, if your application is unresponsive or stuck in a loop, you may need to use `docker kill` to terminate it forcibly.

Pause vs Stop

In Docker, the `docker pause` and `docker stop` commands are used to manage the lifecycle of containers, but they have different behaviors and implications.

The `docker pause` command suspends all processes inside a running container, effectively putting it into a frozen state. While the container is paused, its memory and other resources are still allocated and reserved for it, but its processes are not running. Pausing a container can be useful when you need to free up system resources temporarily, debug or troubleshoot an issue, or take a snapshot of a container's state.

On the other hand, the `docker stop` command sends a `SIGTERM` signal to the container's main process, requesting it to stop gracefully. The container is given a grace period of 10 seconds by default (can be customized with `--time` option), during which it can perform cleanup operations and exit its processes cleanly. If the container does not stop within the grace period, Docker will send a `SIGKILL` signal to force it to stop. Stopping a container can be useful when you need to cleanly shut down the container's processes and release resources, or when you need to update or modify the container's configuration or environment.

In summary, pausing a container suspends its processes without stopping or terminating them, while stopping a container requests its processes to stop gracefully and exit cleanly. Pausing a container can be useful in certain situations, such as when you need to temporarily free up system resources or debug an issue, while stopping a container is typically used to cleanly shut down its processes and release resources

Remove vs Kill

In Docker, the `docker rm` and `docker kill` commands are used to remove or stop running containers, respectively, but they have different behaviors and implications.

The `docker rm` command removes a stopped or running container from the host system. Before removing the container, Docker will attempt to stop it gracefully using the `docker stop` command if it is running. If the container cannot be stopped gracefully, Docker will forcibly terminate its processes using the `SIGKILL` signal and then remove the container. Removing a container using the `docker rm` command frees up any resources that were allocated to it, including its file system, network settings, and configuration.

On the other hand, the `docker kill` command sends a `SIGKILL` signal to a running container's main process, immediately terminating it and all of its child processes. This command does not give the container a chance to perform any cleanup operations or shutdown its processes gracefully, which can result in data loss or corruption. Killing a container using the `docker kill` command should be used as a last resort when the container is unresponsive or stuck in a loop.

In summary, the `docker rm` command is used to remove a stopped or running container from the host system, while the `docker kill` command is used to forcibly terminate a running container's processes immediately. `docker rm` attempts to stop the container gracefully before removing it, while `docker kill` immediately terminates the container without giving it a chance to clean up.

It's important to note that removing or killing a container will result in the loss of any data or changes made to the container's file system that were not persisted using volumes or other persistence mechanisms. Therefore, it's important to ensure that any important data or changes are persisted before removing or killing a container.