Containers

Overview

A container is a standard unit of software that packages up code, environment, and all its dependencies so the application runs quickly and reliably from one computing environment to another.

Containers are based on several features of the the linux kernel:

Namespaces

Linux system for limiting what a process can see, allowing for the isolation of containers from other host machine processes.

To namespace a new process in linux, system calls to the kernel are cloned, allowing the process to change values in its own "container" without affecting the host machine.

To set up the namespaced container environment, layers of processes are created. A process is created and has its syscalls namespaced, then calls another similar process to set the environment in that namespace.

Therefore, to call a command in a specific namespace and environment, you'd pass the command to the container, the base container would setup the namespace (clone syscalls) and spawn a child process, then the child process would setup the environment and call the command, making the command run in an isolated environment.

chroot

While some syscalls can be cloned directly, some other commands such as 'ps' analyse items throughout the root proc directory to return the correct value. Therefore, the root for the container must be changed.

The chroot command in linux allows for the root for a process to be moved to another directory. Therefore, to create an isolated container, during the container setup the root can be moved to a newly mounted cloned copy of the linux root. Normally the container clones the filesystem packaged in the container image, which is specified during setup.

So to mock calls to the 'ps' command, the root is changed to a new clone of the linux filesystem, then the proc directory is mounted as a proc sudo filesystem (to allow the kernel to know that information about new processes should place there). Now from within the container the 'ps' command will then search for processes in the new root proc location, only finding processes based in the isolated "container" filesystem.

Changing the root comes with many advantages:

* The container is isolated to its root filesystem, stopping it from interacting with other files on the host machine.
* Since the root filesystem is cloned from the one in the image, upon boot of new containers, the filesystem is always clean allowing for consistent environment management between instances of the container

cgroups

Control groups allow for restriction of the resources available to a process, configured through a psuedo filesystem interface called cgroups. Limitations can be added on specific parameters available to the kernel, such as memory, cpu, I/O, and number of processes.

The cgroups filesystem is typically based in /sys/fs/cgroup and for example, to change the amount of memory available to a container, the ./memory/memory.limit\_in\_bytes file can be changed.

For containers, new control groups can be created by adding a directory named after the new group, to the specific cgroup/<resource-name> with a cgfile inside it specifying the restrictions to be imposed, and a 'cgroup.procs' file. The id number of child process which the container spawns can then be added to the 'cgroup.procs' file and the kernel will restrict the resources as specifid by the files in the control group.

Why

These linux features allow containers to be created and run using the same kernel as the host machine, while remaining isolated. Since the kernel does not have to be started, starting a container process is very quick.

Container images are built in layers, allowing the environment to be setup procedurally and consistently. Developers can quickly take a current container image and build upon it.