Isolation

Singularity containers can be isolated from the host environment in various steps.
From the operating system's (aka host's) view containers are just processes es all other processes running under the kernel. However, the processes of a container can be constraint and get their own namespaces, so that from their viewpoint they might not see the same paths as the other processes anymore (for example).

Fast Containment

Use the '--contain' or '–containall' switches to restrict a container instance as far as possible.

With --contain no paths/file systems from your host will be visible in the container (you can explicitly add some back with --bind ...)

and with --containall the container instance will be fully locked down, not seeing any other processes etc.

Mount Namespaces

To limit/enable what a process in a container can see in the filesystem, mount namespaces are used. A container started without bind-mounted file systems might only be able to get a limited view on what is available on the host (which is intended in many cases as to prevent harm etc.)

So, if youi have /cvmfs available on your host, you might want to have it also available in the container

```
> singularity shell --bind /cvmfs:/cvmfs:ro ContainerName
```

Let's say, you want also to read from your AFS directory, but as you are writing output only to DUST and want to protect your stuff on AFS from harm, you can mount it as read-only (:ro) and DUST as writable (:rw)

```
> singularity shell --bind /afs:/afs:ro,/nfs:/nfs:rw ContainerName
```

Process Namespaces

With the default, you will be able to see in your container all other processes on your host

> singularity shell ContainerName

```
Singularity ContainerName:~> ps axf | head -n2
  PID TTY      STAT   TIME COMMAND
    2 ?        S      0:00 [kthreadd]
```

For more containment you can jail the container into its own process namespace, so that it sees only the processes started within it

```
> singularity shell --pid test

Singularity ContainerName:~> sleep 300 &
Singularity ContainerName:~> ps axf
  PID TTY      STAT   TIME COMMAND
    1 pts/5    S      0:00 shim-init
    2 pts/5    S      0:00 /bin/bash --norc
   15 pts/5    S      0:00  \_ sleep 300
   28 pts/5    R+     0:00  \_ ps axf
```

On another shell on your host you will see, that the container's processes are living in the process tree as usual (from your kerrnel's point of view)

```
> ps axf
...
3089 pts/2    S+     0:00            |  |  \_ less
19096 pts/5   Ss     0:00            |  \_ /bin/bash
2911 pts/5    S      0:00            |     \_ /usr/libexec/singularity/bin/action-suid
2915 pts/5    S      0:00            |        \_ shim-init
2916 pts/5    S+     0:00            |           \_ /bin/bash --norc
2941 pts/5    S      0:00            |              \_ sleep 300
2274 ?        S      0:00            \_ /usr/bin/python2 /usr/bin/blueberry-tray
2532 ?        Sl     0:01            |  \_ /usr/bin/python2 /usr/lib/blueberry/blueberry-tray.py
...
```

you just cannot see the other processes anymore from within your container.

Interprocess Communication Namespaces

With the --ipc flag a container would gets its own inter process communication namespace. This has the advantage/disadvantage, that a process cannot communicate or share memory with other processes - so, if you have a number of similar containers/processes that would be able to share libraries or so in a shared memory, each one would get its own segment (with the drawback of course, that your memory usage get's blown up as nothing is shared anymore - use it with care, if you need it...)