This guide provides an overview of user namespaces, which was introduced as a new containment namespace in Docker Engine 1.10.

**What are Namespaces?**

From the manpage:

<table>
<thead>
<tr>
<th>NAME</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>namespaces - overview of Linux namespaces</td>
<td>A namespace wraps a global system resource in an abstraction that makes it appear to the processes within the namespace that they have their own isolated instance of the global resource...</td>
</tr>
</tbody>
</table>

Namespaces of various flavors are essential to the functioning of containers as we know them. For example, the PID namespace is what keeps processes in one container from seeing or interacting with processes in another container (or, for that matter, on the host system). A process might have the apparent PID 1 inside a container, but if we examine it from the host system, it would have an ordinary PID, eg:

```
pvn@gyarados /home/pvn> docker run -it alpine /bin/sh
/ # ps
PID USER TIME COMMAND
 1 root 0:00 /bin/sh
 9 root 0:00 ps
/ #
[^P^Q]
pvn@gyarados /home/pvn> ps a | grep "[b]in/sh"
26385 pts/3 Ss+ 0:00 /bin/sh
```

The PID namespace is the mechanism for remapping PIDs inside the container. Likewise, there are other namespaces (e.g. net, mnt, ipc, uts) that (along with cgroups) provide the isolated environments we know as containers. The user namespace, then, is the mechanism for remapping UIDs inside a container, and this is the newest namespace to be implemented in the Docker Engine, starting in 1.10.

**How are User Namespaces Activated?**

You can start remapping UIDs in Docker Engine with the --userns-remap flag. However, there is a bit of configuration you have to set up before this will actually do anything. The flag takes a single argument, a username. This username must exist in the /etc/passwd file, though it doesn't necessarily need to be a fully-fledged user (i.e. you can use something like /sbin/nologin for the shell, etc). You also need to have subordinate UID and GID ranges specified in the /etc/subuid and /etc/subgid files, respectively.
Note here, the UID/GID we are actually remapping to does not have to match the UID/GID of the username in /etc/passwd. Whatever is in the subuid file (the subordinate UID) is what will actually own the processes we start. Despite this, you do actually have to match the user name itself in the passwd and subuid files with the name you pass on the command line to the engine in the --userns-remap flag. Also, note in this example I reserved a range of 65536 UIDs (the numbers in the subuid file are the starting UID and the number of UIDs available to that user) but Docker Engine will only use the first one in the range (for now, Engine is only capable of remapping to a single UID).

Using Namespaces

In any case, let's start up the engine with the --userns-remap flag:

The first thing you will notice at this point is that any images you had originally pulled will be gone.

A quick investigation in /var/lib/docker will reveal what's going on:
OK, so this remapped engine will basically operate in a new environment (in the 100000.100000 directory). Every remapping will get its own directory (format XXX.YYY where XXX is the subordinate UID and YYY is the subordinate GID) - we can look in there and see it's essentially a new, isolated /var/lib/docker.

OK, let's pull something and fire it up.

```
pvn@gyarados /home/pvn> docker pull pvnovarese/mprime
Using default tag: latest
latest: Pulling from pvnovarese/mprime
a3ed95caeb02: Pull complete
546e579918ed: Pull complete
Digest: sha256:21561b776f6e3f30044d09e40f31d696425354e4a1885da10c153eb5bb707237
Status: Downloaded newer image for pvnovarese/mprime:latest
```

```
pvn@gyarados /home/pvn> docker run -d --name=mprime0 pvnovarese/mprime:latest
```

```
pvn@gyarados /home/pvn> ps aux | grep [m]prime
100000 1518 91.0 0.0 15224 11652 ? Rns 20:12 0:07 /mprime -t
```

As you can see, no new commands are needed from the operator perspective. Once the daemon is running, the operator uses the same pull/run commands but the processes run as the remapped subordinate UID (in this case, 100000) instead of root.

But what do those processes look like inside the container? We can look inside a running container and compare the UID for the same process as seen from the host:
So, in that example, the /bin/sh process is owned by root inside the container, but it's owned by the subordinate UID outside of the container. This same example also shows the pid namespace remapping, as the process has PID 1 inside the container but 6082 outside the container.

What if we run multiple containers?

Note: Processes in both containers are using the same UID; as noted before, even though there is a range of subordinate UIDs specified in the /etc/subuid file, Docker Engine will only use the first one (for now).