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:

```
NAMESPACES(7)            Linux Programmer's Manual           NAMESPACES(7)

NAME
       namespaces - overview of Linux namespaces

DESCRIPTION
       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...
```

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 fullly-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.

```
pvn@gyarados /home/pvn> grep bozo /etc/passwd
bozo:x:5000:5000::/home/bozo:/bin/false
pvn@gyarados /home/pvn> grep bozo /etc/group
bozo:x:5000:
pvn@gyarados /home/pvn> cat /etc/subuid
bozo:100000:65536
pvn@gyarados /home/pvn> cat /etc/subgid
bozo:100000:65536
```

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:

```
pvn@gyarados /home/pvn> sudo docker daemon --userns-remap=bozo &
[1] 659
pvn@gyarados /home/pvn> WARN[0000] Running experimental build
INFO[0000] User namespaces: ID ranges will be mapped to subuid/subgid ranges of: bozo:bozo
WARN[0000] devmapper: Usage of loopback devices is strongly discouraged for production use. Please use
`--storage-opt dm.thinpooldev` or use `man docker` to refer to dm.thinpooldev section.
INFO[0000] devmapper: Creating filesystem xfs on device docker-8:19-30671130-base
INFO[0000] devmapper: Successfully created filesystem xfs on device docker-8:19-30671130-base
INFO[0001] Graph migration to content-addressability took 0.00 seconds
INFO[0001] Firewalld running: true
INFO[0001] Default bridge (https://success.docker.com/api/asset/.%2Fintroduction-to-user-namespaces-in-
docker-engine%2Fdocker0) is assigned with an IP address 172.17.0.0/16. Daemon option --bip can be used
to set a preferred IP address
INFO[0002] Loading containers: start.

INFO[0002] Loading containers: done.
INFO[0002] Daemon has completed initialization
INFO[0002] Docker daemon                                 commit=79edcc5 execdriver=native-0.2
graphdriver=devicemapper version=1.11.0-dev
INFO[0002] API listen on /var/run/docker.sock

pvn@gyarados /home/pvn>
```

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

```
pvn@gyarados /home/pvn> docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
pvn@gyarados /home/pvn>
```

A quick investigation in /var/lib/docker will reveal what's going on:

```
pvn@gyarados /home/pvn> sudo ls -lF /var/lib/docker/
total 80
drwx------.  9 bozo   bozo   4096 Mar 11 20:03 100000.100000/
drwx------. 15 root   root   4096 Mar 11 09:35 containers/
drwx------.  5 root   root   4096 Jul  4  2015 devicemapper/
drwxr-xr-x.  2 root   root   4096 Feb  4 08:25 discovery_certs/
drwx------. 66 root   root  16384 Dec 23 11:31 graph/
drwx------.  3 root   root   4096 Dec 23 21:00 image/
drwx------.  2 root   root   4096 Jul  4  2015 init/
-rw-r--r--.  1 root   root  13312 Mar 11 09:26 linkgraph.db
drwxr-x---.  3 root   root   4096 Oct 15 16:52 network/
-rw-------.  1 root   root   1257 Dec 23 11:31 repositories-devicemapper
drwx------.  6 root   root   4096 Mar 11 20:02 tmp/
drwx------.  2 root   root   4096 Jul  4  2015 trust/
drwx------. 17 root   root   4096 Mar 11 09:25 volumes/
pvn@gyarados /home/pvn>
```

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.

```
pvn@gyarados /home/pvn> sudo ls -F /var/lib/docker/100000.100000/
containers/  devicemapper/  image/  network/  tmp/  trust/  volumes/
pvn@gyarados /home/pvn>
```

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
7f47d752ba9d110c162acfcac7d0ed696495b60b8677b1556de771b382429c2c
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:

```
pvn@gyarados /home/pvn> docker run -it alpine /bin/sh
ERRO[0102] Handler for POST /v1.23/containers/create returned error: No such image: alpine:latest
Unable to find image 'alpine:latest' locally
latest: Pulling from library/alpine
4d06f2521e4f: Pull complete
Digest: sha256:7739b19a213f3a0aa8dacbd5898c8bd467e6eaf71074296a3d75824e76257396
Status: Downloaded newer image for alpine:latest
/ # ps
PID   USER     TIME   COMMAND
    1 root       0:00 /bin/sh
    9 root       0:00 ps
/ #

[^P^Q]

pvn@gyarados /home/pvn> ps au | grep [b]in/sh
100000      6082  0.1  0.0   1516     4 pts/1    Ss+  12:41   0:00 /bin/sh
pvn@gyarados /home/pvn>
```

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?

```
pvn@gyarados /home/pvn> docker run -d --name=mprime1 pvnovarese/mprime:latest
e087aee0a9ce5a65285db081159f676ae8a5eecb62e019175721368d8d83f653
pvn@gyarados /home/pvn> docker ps
CONTAINER ID        IMAGE                      COMMAND             CREATED              STATUS
PORTS               NAMES
e087aee0a9ce        pvnovarese/mprime:latest   "/mprime -t"        11 seconds ago       Up 8 seconds
mprime1
7f47d752ba9d        pvnovarese/mprime:latest   "/mprime -t"        About a minute ago   Up 59 seconds
mprime0
pvn@gyarados /home/pvn> ps aux | grep [m]prime
100000      1518 99.5  0.0  15224 11652 ?        RNs  20:12   1:09 /mprime -t
100000      1657 97.9  0.0  15224 11648 ?        RNs  20:13   0:19 /mprime -t
pvn@gyarados /home/pvn>
```

> **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).