Isolate containers with a user namespace

- About remapping and subordinate user and group IDs
- Prerequisites
- Enable userns-remap on the daemon
- Disable namespace remapping for a container
- User namespace known limitations

Linux namespaces provide isolation for running processes, limiting their access to system resources without the running process being aware of the limitations. For more information on Linux namespaces, see Linux namespaces.

The best way to prevent privilege-escalation attacks from within a container is to configure your container's applications to run as unprivileged users. For containers whose processes must run as the `root` user within the container, you can re-map this user to a less-privileged user on the Docker host. The mapped user is assigned a range of UIDs which function within the namespace as normal UIDs from 0 to 65536, but have no privileges on the host machine itself.

> With `userns-remap`, the Docker daemon still runs as root. To run both the daemon and containers without root privileges, see Rootless mode instead.

About remapping and subordinate user and group IDs

The remapping itself is handled by two files: `/etc/subuid` and `/etc/subgid`. Each file works the same, but one is concerned with the user ID range, and the other with the group ID range. Consider the following entry in `/etc/subuid`:

```
testuser:231072:65536
```

This means that `testuser` is assigned a subordinate user ID range of `231072` and the next 65536 integers in sequence. UID `231072` is mapped within the namespace (within the container, in this case) as UID `0` (`root`). UID `231073` is mapped as UID `1`, and so forth. If a process attempts to escalate privilege outside of the namespace, the process is running as an unprivileged high-number UID on the host, which does not even map to a real user. This means the process has no privileges on the host system at all.

> It is possible to assign multiple subordinate ranges for a given user or group by adding multiple non-overlapping mappings for the same user or group in the `/etc/subuid` or `/etc/subgid` file. In this case, Docker uses only the first five mappings, in accordance with the kernel's limitation of only five entries in `/proc/self/uid_map` and `/proc/self/gid_map`.

When you configure Docker to use the `userns-remap` feature, you can optionally specify an existing user and/or group, or you can specify `default`. If you specify `default`, a user and group `dockremap` is created and used for this purpose.

> Some distributions do not automatically add the new group to the `/etc/subuid` and `/etc/subgid` files. If that's the case, you may have to manually edit these files and assign non-overlapping ranges. This step is covered in Prerequisites.

It is very important that the ranges do not overlap, so that a process cannot gain access in a different namespace. On most Linux distributions, system utilities manage the ranges for you when you add or remove users.

This re-mapping is transparent to the container, but introduces some configuration complexity in situations where the container needs access to resources on the Docker host, such as bind mounts into areas of the filesystem that the system user cannot write to. From a security standpoint, it is best to avoid these situations.

Prerequisites

1. The subordinate UID and GID ranges must be associated with an existing user, even though the association is an implementation detail. The user owns the namespaced storage directories under `/var/lib/docker/`. If you don't want to use an existing user, Docker can create one for you and use that. If you want to use an existing username or user ID, it must already exist. Typically, this means that the relevant entries need to be in `/etc/passwd` and `/etc/group`, but if you are using a different authentication back-end, this

requirement may translate differently.
To verify this, use the `id` command:

```
$ id testuser


uid=1001(testuser) gid=1001(testuser) groups=1001(testuser)
```

2. The way the namespace remapping is handled on the host is using two files, `/etc/subuid` and `/etc/subgid`. These files are typically managed automatically when you add or remove users or groups, but on some distributions, you may need to manage these files manually.
   Each file contains three fields: the username or ID of the user, followed by a beginning UID or GID (which is treated as UID or GID 0 within the namespace) and a maximum number of UIDs or GIDs available to the user. For instance, given the following entry:

   ```
   testuser:231072:65536
   ```

   This means that user-namespaced processes started by `testuser` are owned by host UID `231072` (which looks like UID `0` inside the namespace) through 296607 (231072 + 65536 - 1). These ranges should not overlap, to ensure that namespaced processes cannot access each other's namespaces.
   After adding your user, check `/etc/subuid` and `/etc/subgid` to see if your user has an entry in each. If not, you need to add it, being careful to avoid overlap.
   If you want to use the `dockremap` user automatically created by Docker, check for the `dockremap` entry in these files after configuring and restarting Docker.
3. If there are any locations on the Docker host where the unprivileged user needs to write, adjust the permissions of those locations accordingly. This is also true if you want to use the `dockremap` user automatically created by Docker, but you can't modify the permissions until after configuring and restarting Docker.
4. Enabling `userns-remap` effectively masks existing image and container layers, as well as other Docker objects within `/var/lib/docker/`. This is because Docker needs to adjust the ownership of these resources and actually stores them in a subdirectory within `/var/lib/docker/`. It is best to enable this feature on a new Docker installation rather than an existing one.
   Along the same lines, if you disable `userns-remap` you can't access any of the resources created while it was enabled.
5. Check the limitations on user namespaces to be sure your use case is possible.

Enable userns-remap on the daemon

You can start `dockerd` with the `--userns-remap` flag or follow this procedure to configure the daemon using the `daemon.json` configuration file. The `daemon.json` method is recommended. If you use the flag, use the following command as a model:

```
$ dockerd --userns-remap="testuser:testuser"
```

1. Edit `/etc/docker/daemon.json`. Assuming the file was previously empty, the following entry enables `userns-remap` using user and group called `testuser`. You can address the user and group by ID or name. You only need to specify the group name or ID if it is different from the user name or ID. If you provide both the user and group name or ID, separate them by a colon (`:`) character. The following formats all work for the value, assuming the UID and GID of `testuser` are `1001`:
   - `testuser`
   - `testuser:testuser`
   - `1001`
   - `1001:1001`
   - `testuser:1001`
   - `1001:testuser`

   ```
   {
     "userns-remap": "testuser"
   }
   ```

   > To use the `dockremap` user and have Docker create it for you, set the value to `default` rather than `testuser`.

   Save the file and restart Docker.
2. If you are using the `dockremap` user, verify that Docker created it using the `id` command.

   ```
   $ id dockremap


   uid=112(dockremap) gid=116(dockremap) groups=116(dockremap)
   ```

   Verify that the entry has been added to `/etc/subuid` and `/etc/subgid`:

   ```
   $ grep dockremap /etc/subuid


   dockremap:231072:65536


   $ grep dockremap /etc/subgid


   dockremap:231072:65536
   ```

   If these entries are not present, edit the files as the `root` user and assign a starting UID and GID that is the highest-assigned one plus the offset (in this case, `65536`). Be careful

not to allow any overlap in the ranges.
3. Verify that previous images are not available using the `docker image ls` command. The output should be empty.
4. Start a container from the `hello-world` image.

   ```
   $ docker run hello-world
   ```

5. Verify that a namespaced directory exists within `/var/lib/docker/` named with the UID and GID of the namespaced user, owned by that UID and GID, and not group-or-world-readable. Some of the subdirectories are still owned by `root` and have different permissions.

   ```
   $ sudo ls -ld /var/lib/docker/231072.231072/

   drwx------ 11 231072 231072 11 Jun 21 21:19 /var/lib/docker/231072.231072/

   $ sudo ls -l /var/lib/docker/231072.231072/

   total 14
   drwx------ 5 231072 231072 5 Jun 21 21:19 aufs
   drwx------ 3 231072 231072 3 Jun 21 21:21 containers
   drwx------ 3 root   root   3 Jun 21 21:19 image
   drwxr-x--- 3 root   root   3 Jun 21 21:19 network
   drwx------ 4 root   root   4 Jun 21 21:19 plugins
   drwx------ 2 root   root   2 Jun 21 21:19 swarm
   drwx------ 2 231072 231072 2 Jun 21 21:21 tmp
   drwx------ 2 root   root   2 Jun 21 21:19 trust
   drwx------ 2 231072 231072 3 Jun 21 21:19 volumes
   ```

   Your directory listing may have some differences, especially if you use a different container storage driver than `aufs`.
   The directories which are owned by the remapped user are used instead of the same directories directly beneath `/var/lib/docker/` and the unused versions (such as `/var/lib/docker/tmp/` in the example here) can be removed. Docker does not use them while `userns-remap` is enabled.

Disable namespace remapping for a container

If you enable user namespaces on the daemon, all containers are started with user namespaces enabled by default. In some situations, such as privileged containers, you may need to disable user namespaces for a specific container. See user namespace known limitations for some of these limitations.

To disable user namespaces for a specific container, add the `--userns=host` flag to the `docker container create`, `docker container run`, or `docker container exec` command.

There is a side effect when using this flag: user remapping will not be enabled for that container but, because the read-only (image) layers are shared between containers, ownership of the containers filesystem will still be remapped.

What this means is that the whole container filesystem will belong to the user specified in the `--userns-remap` daemon config (`231072` in the example above). This can lead to unexpected behavior of programs inside the container. For instance `sudo` (which checks that its binaries belong to user `0`) or binaries with a `setuid` flag.

User namespace known limitations

The following standard Docker features are incompatible with running a Docker daemon with user namespaces enabled:

- Sharing PID or NET namespaces with the host (`--pid=host` or `--network=host`).
- External (volume or storage) drivers which are unaware or incapable of using daemon user mappings.
- Using the `--privileged` mode flag on `docker run` without also specifying `--userns=host`.

User namespaces are an advanced feature and require coordination with other capabilities. For example, if volumes are mounted from the host, file ownership must be pre-arranged if you need read or write access to the volume contents.

While the root user inside a user-namespaced container process has many of the expected privileges of the superuser within the container, the Linux kernel imposes restrictions based on internal knowledge that this is a user-namespaced process. One notable restriction is the inability to use the `mknod` command. Permission is denied for device creation within the container when run by the `root` user.

□□□ #2
□ □□□ □□□ 2026-10-31 19:30:23 UTC
□ □□□ □□□ 2026-10-31 19:30:25 UTC