

Docker Registry

Docker Registry

- If containers were the rain, then they will rain from the docker registry which are the clouds.
- All the docker images are stored in a central repository named Docker Registry.
- Let's look at a simple nginx container
`docker run nginx`
- The image name used here is nginx
- But what is this image?
- Where is this image pulled from?

Docker Registry

- **image:** nginx

- This image name follows docker's image naming convention.
- Nginx here is the image or the repository name.
- When you say nginx it's actually

- **image:** nginx/nginx

image: nginx/ nginx


User/Account

Image/Repository

- If you don't provide a user/account name it assumes it to be same as the image name.
- The username is usually your Docker Hub account name or the name of the organization

Docker Registry

- Where are these images stored and pulled from?
- When you do not specify the image location, it assumes to pull from Docker's default registry "Docker Hub"



docker.io
Docker Hub

image: docker.io/nginx/ nginx

Registry

User/Account

Image/Repository

- The DNS name for Docker Hub is docker.io
- Whenever you create a new image or update an existing image you push it to the registry
- When you want to run a container images are pulled from that registry.

Docker Registry

- Other popular public registry is **gcr.io - Google's Registry**
- In public registry images are publicly accessible to everyone
- You shouldn't make your custom-built application images to public registry
- Hosting an internal private registry is an ideal solution
- Many cloud service providers such as AWS, Azure or GCP provide a private registry service.
- On any of these solutions be it Docker Hub or AWS, etc., you may choose to make a repository private so that it is only accessed using a set of credentials

Docker Registry

- How do we run container from images hosted in Private Registry
- First log in to your private registry
`docker login prabhav-registry.io`
- Once the login is successful, run the application using private registry as part of the image name
`docker run prabhav-registry.io/apps/internal-app`
- Remember to always log in before pulling or pushing to a private registry

Deploy Docker Private Registry

- How to deploy private registry within the organization (on-premise)?
- There is an application provided by Docker, i.e. "Docker Registry"
- It is available as a docker image, the name of the image is "registry"
- It exposes the API on port 5000

```
docker run -d -p 5000:5000 --name=registry registry
```

- Now we have our custom registry running at port 5000 on this docker a host.

Deploy Docker Private Registry

- How do we push our own image to this private registry?
- Tag the image

```
docker image tag my-image localhost:5000/my-image
```

- Push the image

```
docker push localhost:5000/my-image
```

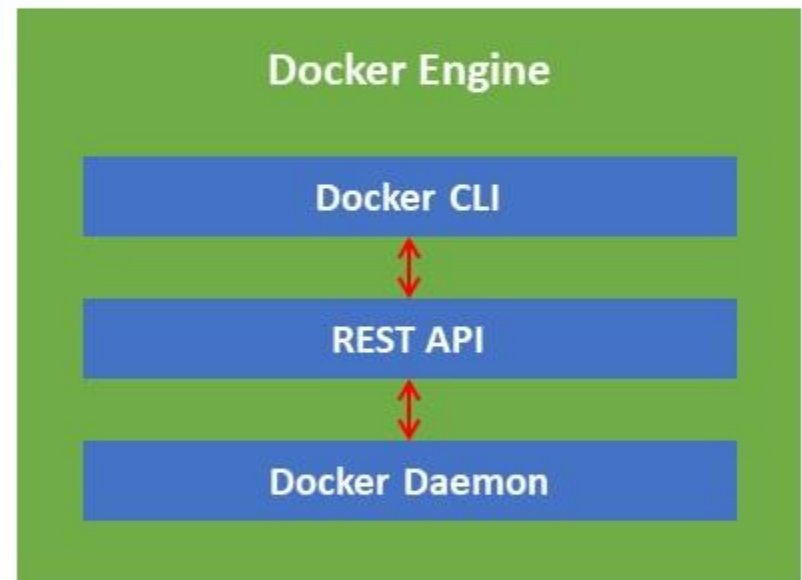
- How do we pull images from this private registry?

```
docker pull localhost:5000/my-image
```


Docker Engine

Docker Engine

- Docker demon is a background process that manages Docker objects such as the images, containers, volumes and networks.
- Docker REST API server is the API interface that programs can use to talk to the demon and provide instructions. We can create our own tools using this REST API
- Docker CLI is the command line interface to perform actions such as running a container, stopping containers, destroying images etc.
- Docker CLI uses the REST API to interact with the docker demon.



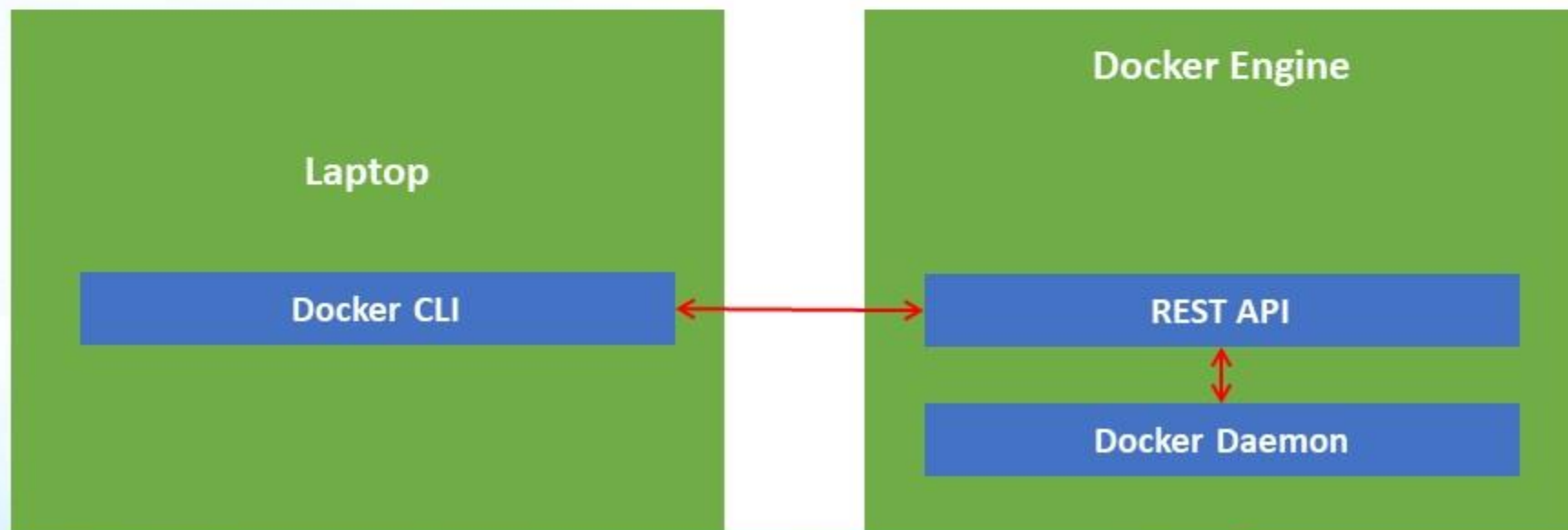
Docker Engine

- Docker's CLI can be on a different host and can still work with a remote Docker engine.

```
docker -H=remote-docker-engine:2375
```

- Run a nginx container on remote docker engine 10.0.0.5

```
docker -H=10.0.0.5:2375 run nginx
```



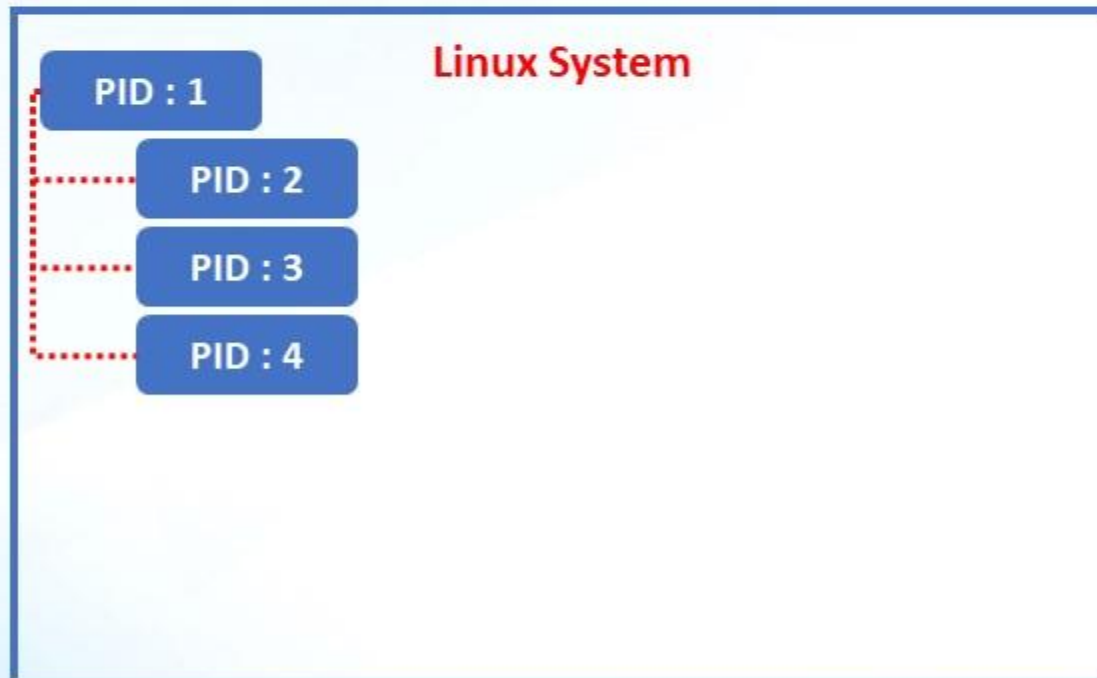
Namespace

- Docker uses namespace to isolate workspace



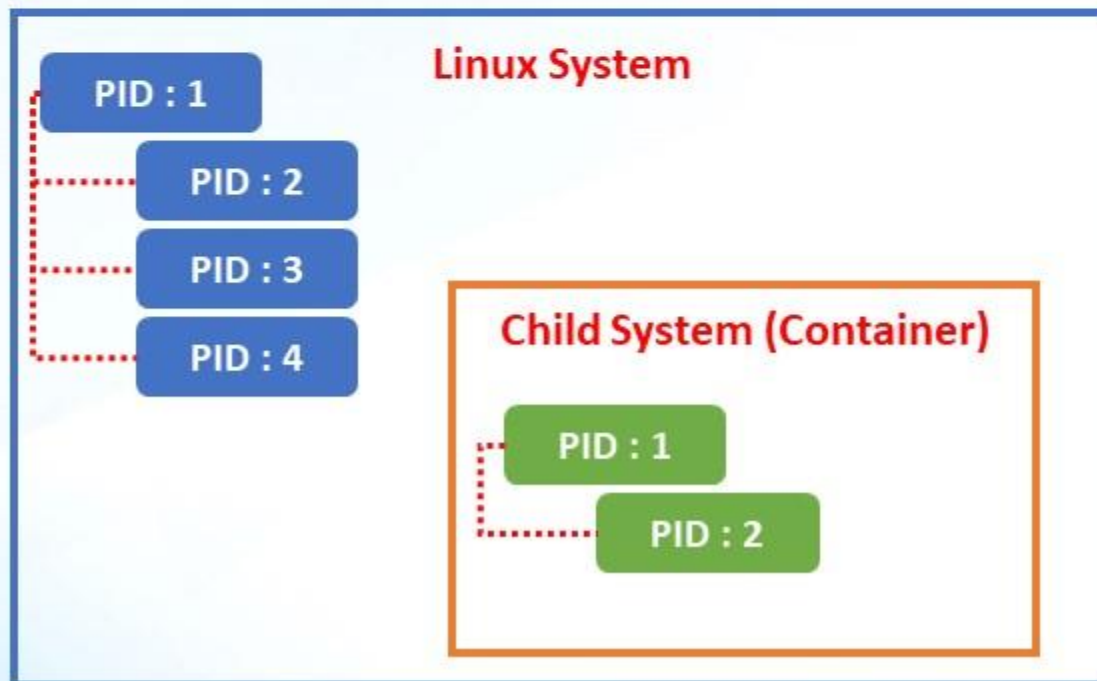
Namespace - PID

- Whenever a Linux system boots up, it starts with just one process with a **Process ID 1**
- This is the root process which starts all the other processes in the system
- When system starts completely, we have multiple processes running
- The process ids are unique, and two processes cannot have the same process ID.



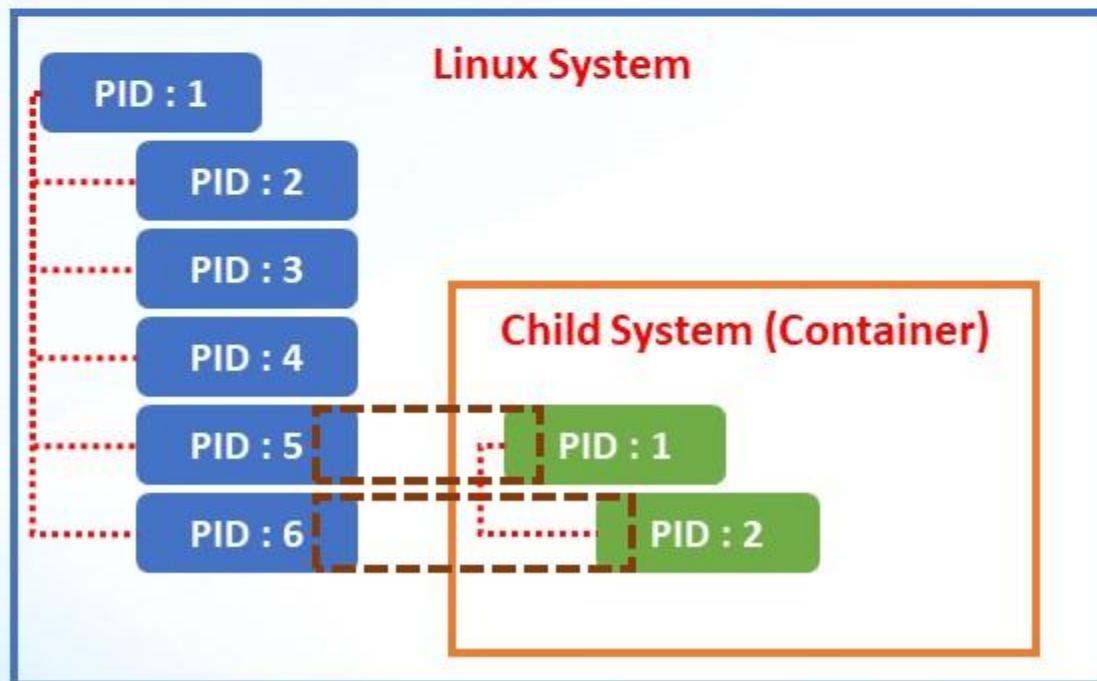
Namespace - PID

- If we create a container(Child System) in the current system, it acts like an independent system
- It has its own set of processes originating from a root process with a **Process ID 1**
- But there is no hard isolation between the containers and the underlying host.
- Processes running inside the container are in fact processes running on the underlying host.



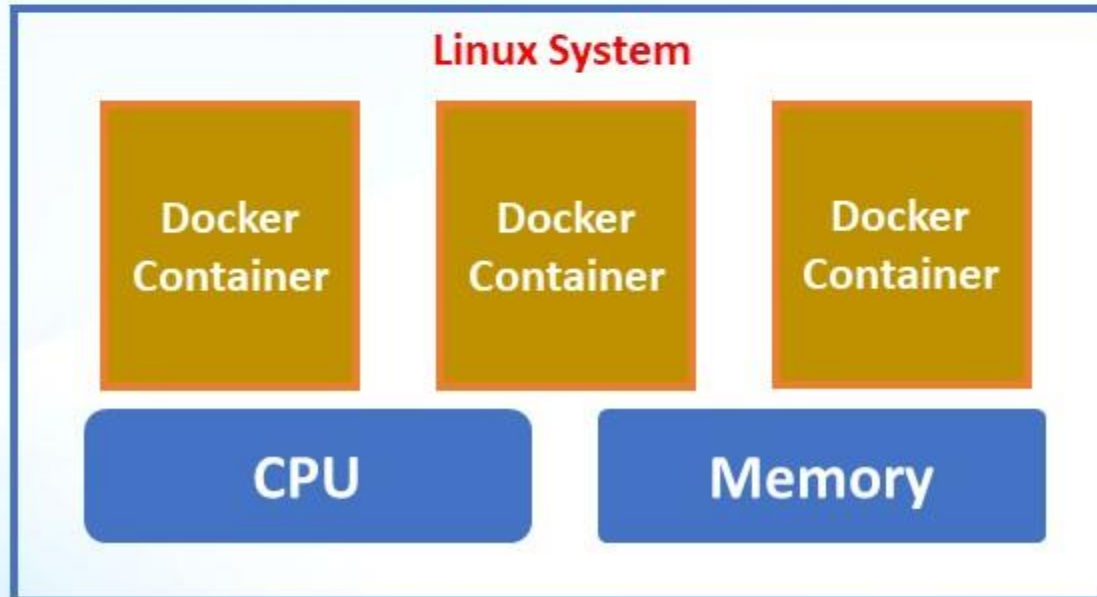
Namespace - PID

- Each process can have multiple process IDs associated with it.
- When the processes start in the container, it's just another set of processes on the underlying host and it gets the next available process ID



cgroups

- Docker host and the containers share the same system resources such as CPU and memory.
- By default, there is no restriction on resource utilization by a container
- Container may even end up utilizing all resources on the underlying host.



- How much of the resources are dedicated to the host and the containers?
- How does Docker manage and share the resources between the containers?

cgroups

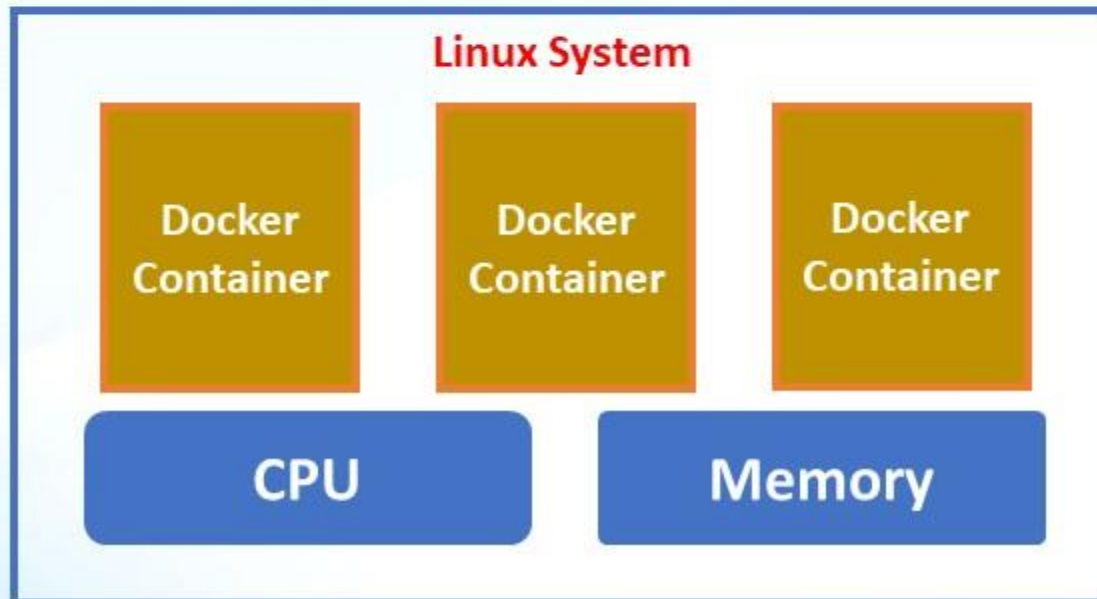
- Docker uses **cgroups** or **control groups** to restrict the amount of hardware resources allocated to each container.

- Restrict CPU utilization of a container to 50%

```
docker run --cpu=.5 nginx
```

- Restrict Memory utilization of a container to 100Mb

```
docker run --memory=100m nginx
```



- **How to restrict the amount of CPU or memory a container can use?**

Docker PID - Demo