Proxy server settings

Proxy servers can block connections to your web app once it’s up and running. If you are behind a proxy server, add the following lines to your Dockerfile, using the ENV command to specify the host and port for your proxy servers:

```
# Set proxy server, replace host:port with values for your servers
ENV http_proxy host:port
ENV https_proxy host:port
```

Docker registries

A Docker *registry* stores Docker images. Docker Hub is a public registry that anyone can use, and Docker is configured to look for images on Docker Hub by default. You can even run your own private registry. If you use Docker Datacenter (DDC), it includes Docker Trusted Registry (DTR).

CONTAINERS

A container is a runnable instance of an image. You can create, start, stop, move, or delete a container using the Docker API or CLI. You can connect a container to one or more networks, attach storage to it, or even create a new image based on its current state.

By default, a container is relatively well isolated from other containers and its host machine. You can control how isolated a container’s network, storage, or other underlying subsystems are from other containers or from the host machine.

A container is defined by its image as well as any configuration options you provide to it when you create or start it. When a container is removed, any changes to its state that are not stored in persistent storage disappear.

SERVICES

Services allow you to scale containers across multiple Docker daemons, which all work together as a *swarm* with multiple *managers* and *workers*. Each member of a swarm is a Docker daemon, and the daemons all communicate using the Docker API. A service allows you to define the desired state, such as the number of replicas of the service that must be available at any given time. By default, the service is load-balanced across all worker nodes. To the consumer, the Docker service appears to be a single application. Docker Engine supports swarm mode in Docker 1.12 and higher.

Namespaces

Docker uses a technology called namespaces to provide the isolated workspace called the *container*. When you run a container, Docker creates a set of *namespaces* for that container.

These namespaces provide a layer of isolation. Each aspect of a container runs in a separate namespace and its access is limited to that namespace.

Docker Engine uses namespaces such as the following on Linux:

- **The pid namespace:** Process isolation (PID: Process ID).
- **The net namespace:** Managing network interfaces (NET: Networking).
- **The ipc namespace:** Managing access to IPC resources (IPC: InterProcess Communication).
- **The mnt namespace:** Managing filesystem mount points (MNT: Mount).
- **The uts namespace:** Isolating kernel and version identifiers. (UTS: Unix Timesharing System).

Control groups

Docker Engine on Linux also relies on another technology called *control groups* (cgroups). A cgroup limits an application to a specific set of resources. Control groups allow Docker Engine to share available hardware resources to containers and optionally enforce limits and constraints. For example, you can limit the memory available to a specific container.

Union file systems

Union file systems, or UnionFS, are file systems that operate by creating layers, making them very lightweight and fast. Docker Engine uses UnionFS to provide the building blocks for containers. Docker Engine can use multiple UnionFS variants, including AUFS, btrfs, vfs, and DeviceMapper.

Container format

Docker Engine combines the namespaces, control groups, and UnionFS into a wrapper called a container format. The default container format is libcontainer. In the future, Docker may support other container formats by integrating with technologies such as BSD Jails or Solaris Zones.

Best Practices: https://docs.docker.com/develop/dev-best-practices/

Docker’s networking subsystem is pluggable, using drivers.

Network driver summary

- **User-defined bridge networks** are best when you need multiple containers to communicate on the same Docker host.
- **Host networks** are best when the network stack should not be isolated from the Docker host, but you want other aspects of the container to be isolated.
- **Overlay networks** are best when you need containers running on different Docker hosts to communicate, or when multiple applications work together using swarm services.
- **Macvlan networks** are best when you are migrating from a VM setup or need your containers to look like physical hosts on your network, each with a unique MAC address.
- **Third-party network plugins** allow you to integrate Docker with specialized network stacks.

Disconnect a container from a user-defined bridge

To disconnect a running container from a user-defined bridge, use the docker network disconnectcommand. The following command disconnects the my-nginx container from the my-net network.

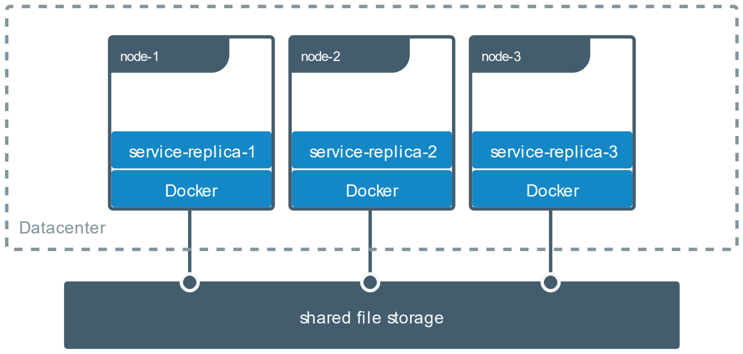
```
$ docker network disconnect my-net my-nginx
```

The default bridge network is considered a legacy detail of Docker and is not recommended for production use.

By default, when you create a container, it does not publish any of its ports to the outside world. To make a port available to services outside of Docker, or to Docker containers which are not connected to the container’s network, use the --publish or -p flag. This creates a firewall rule which maps a container port to a port on the Docker host

Share data among machines

When building fault-tolerant applications, you might need to configure multiple replicas of the same service to have access to the same files.



There are several ways to achieve this when developing your applications. One is to add logic to your application to store files on a cloud object storage system like Amazon S3. Another is to create volumes with a driver that supports writing files to an external storage system like NFS or Amazon S3.

Volume drivers allow you to abstract the underlying storage system from the application logic. For example, if your services use a volume with an NFS driver, you can update the services to use a different driver, as an example to store data in the cloud, without changing the application logic.

All of Docker's iptables rules are added to the DOCKER chain. Do not manipulate this table manually. If you need to add rules which load before Docker's rules, add them to the DOCKER-USER chain. These rules are loaded before any rules Docker creates automatically.

By default, all external source IPs are allowed to connect to the Docker daemon. To allow only a specific IP or network to access the containers, insert a negated rule at the top of the DOCKER filter chain. For example, the following rule restricts external access to all IP addresses except 192.168.1.1:

```
$ iptables -I DOCKER-USER -i ext_if ! -s 192.168.1.1 -j DROP
```

Prevent Docker from manipulating iptables

To prevent Docker from manipulating the iptables policies at all, set the iptables key
to false in /etc/docker/daemon.json

➔By default, when the Docker daemon terminates, it shuts down running containers. Starting with Docker Engine 1.12, you can configure the daemon so that containers remain running if the daemon becomes unavailable. This functionality is called *live restore*. The live restore option helps reducea container downtime due to daemon crashes, planned outages, or upgrades.

Linux Containers rely on control groups which not only track groups of processes, but also expose metrics about CPU, memory, and block I/O usage. You can access those metrics and obtain network usage metrics as well.

Each image consists of a series of layers. Docker makes use of union file systems to combine these layers into a single image. Union file systems allow files and directories of separate file systems, known as branches, to be transparently overlaid, forming a single coherent file system.