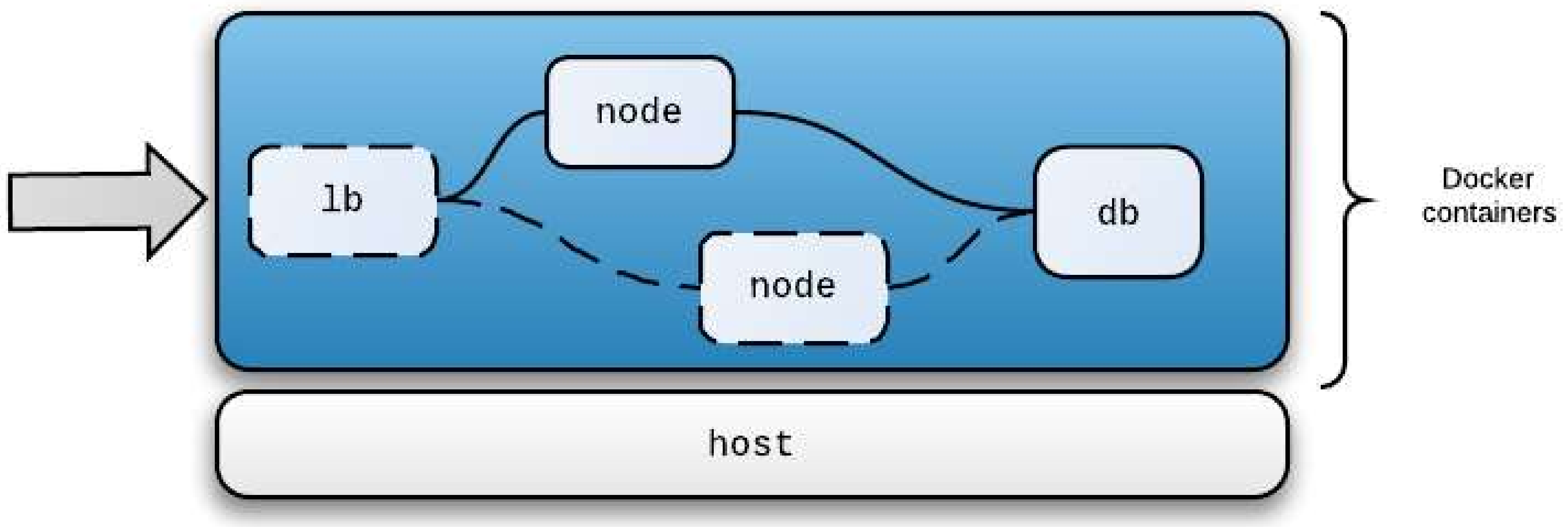
Runninng Docker containers as systemd services

Starting and stopping Docker containers is easy with the `docker run` and `docker {stop|kill}` commands. By default, when the Docker daemon is restarted, running containers are also restarted. This is a great feature but sometimes you need more control over the container's lifecycle.

Imagine having a container which is linked to another one. Docker does not provide any way of specifying which container should be started when or what to do if a container dies.

Deployment scenario

Consider this pretty simple deployment:



When we start such a deployment we need to ensure that the database is started first, so we can later link it to the WildFly node.

Note The load balancer set up is not part of this guide, but is shown (along with another WildFly node) using a dotted outline. The mod_cluster (http://mod-cluster.jboss.org/) project may be a good choice for a load balancer.

We'll use following images to power this setup:

1. fedora/mariadb for the the database
2. a custom image based on jboss/wildfly to run the node

systemd to the rescue

Systemd (http://www.freedesktop.org/wiki/Software/systemd/) is a system management daemon which replaced SysV init scripts in Fedora some time ago. The systemd project provides a very flexible and powerful way to manage services. This is a really big project and all the various use cases for it make it a bit hard to understand. Luckily our deployment is very simple to implement in systemd.

To be able to manage a service with systemd we need to create a service file for it. In our case a service is equal to a running container.

The `docker-mariadb` service

Let's create a `docker-mariadb.service` file for the database container:

```
[Unit]
Description=MariaDB database
Requires=docker.service
After=docker.service

[Service]
User=goldmann
Restart=on-failure
RestartSec=10
ExecStartPre=-/usr/bin/docker kill mariadb
ExecStartPre=-/usr/bin/docker rm mariadb
ExecStart=/usr/bin/docker run --name=mariadb fedora/mariadb
ExecStop=-/usr/bin/docker stop mariadb

[Install]
WantedBy=multi-user.target
```

This file should be stored in the /etc/systemd/system/multi-user.target.wants/ directory.

Note In this example all data will be lost when you remove the `mariadb` container. You may want to look at various options for managing data in containers (https://docs.docker.com/userguide/dockervolumes/).

Service file explained

Let's now go through the docker-mariadb.service file:

The [Unit] section contains information about the service itself. Here you can find the `Description` which will be visible almost everywhere (try `systemctl list-units`).

The `Requires` parameter specifies which service should be **activated and marked to be started before our service**. In our case (if we want to start/enable the `docker-mariadb.service`) the `docker.service` service will be started/enabled too.

The `After` parameter configures the ordering of services. In our case the `docker-mariadb.service` will wait for the `docker.service` to start and only then will it be started. If we don't provide the `After` parameter, our service would be started in parallel with the `docker.service`. This could trigger some issues because there is some chance that the Docker service would be not fully started, when we try to run the container.

The `[Service]` section defines the actual service.

The `User` parameter specifies the user that will be used to execute the command. This is optional but it's a good idea to **keep user privileges as minimal as possible**.

The `Restart` parameter specifies when the service should be restarted. The `on-failure` value means that the service will be restarted when the service is terminated in an unclean way. The `RestartSec` parameter specifies how long we should wait before we try to restart the service.

The `Environment` parameter is useful to set environment variables for the processes that will be started. I use it to lower the memory requirements of the JVM for WildFly (see below).

The `ExecStartPre` parameters specify which command should be run before we start the main process. I use it to clean up (stop and remove) containers. Please note the minus sign just before the command. This tells systemd to not fail the boot even if the command fails.

The `ExecStart` parameter is the heart of the service. Here we define **what should be started to run our service**.

The `ExecStop` parameter is similar to `ExecStart`, but will run if we want to stop the service.

The `[Install]` section defines the configuration used at service install time.

The `WantedBy` parameter specifies that our service should be started when the `multi-user` target is reached. Since this particular target is the default, we enable our service by default.

Note If you want to learn more about systemd a good place to start is the systemd.service (http://www.freedesktop.org/software/systemd/man/systemd.service.html) and systemd.unit (http://www.freedesktop.org/software/systemd/man/systemd.unit.html) man pages. The systemd homepage (http://www.freedesktop.org/wiki/Software/systemd/) has a lot of resources too, including nice blog posts for administrators.

The `docker-wildfly` service

Now we create a similar `docker-wildfly.service` file for the WildFly application server:

```
[Unit]
Description=WildFly node
Requires=docker-mariadb.service
After=docker-mariadb.service

[Service]
User=goldmann
Restart=on-failure
RestartSec=10
Environment="JAVA_OPTS=-Xms64m -Xmx512m"
ExecStartPre=-/usr/bin/docker kill wildfly-mariadb
ExecStartPre=-/usr/bin/docker rm wildfly-mariadb
ExecStart=/usr/bin/docker run --name wildfly --link mariadb:mariadb -p 8080:8080 wildfly-mariadb
ExecStop=-/usr/bin/docker stop wildfly-mariadb

[Install]
WantedBy=multi-user.target
```

Again, place it in the /etc/systemd/system/multi-user.target.wants/ directory.

This file is similar to the first, but with one **very important difference**: the `After` and `Requires` parameters specify the `docker-mariadb` service. This is because we want to boot WildFly after the MariaDB database.

If you want to run WildFly on multiple nodes, just copy the file and edit appropriate values (node name).

Note Although it is possible to run multiple containers from one systemd service file (hint: `Type=oneshot` and `RemainAfterExit=true`) I recommend you have one service per container. This way you'll have better control over them.

The `wildfly-mariadb` image

In the service above we used the `wildfly-mariadb` Docker image. The Dockerfile and instructions on how to build it are available on GitHub (https://github.com/goldmann/wildfly-mariadb).

Enable and run the service

To be able to enable or start the service, systemd needs to be reloaded to pick up our new service files. Run:

```
$ systemctl daemon-reload
```

This command will also mark our service to be started on boot (remember the `WantedBy` parameters?). Now we can start our application:

```
$ systemctl start docker-wildfly
```

To confirm that everything worked, run:

```
$ systemctl status docker-wildfly
$ docker ps
```

Now you can point your browser to http://localhost:8080/wildfly-kitchensink/ (http://localhost:8080/wildfly-kitchensink/) and everything should work.

Note if you use systemd to start the containers on boot, it's a good idea to disable the buit-in Docker functionaly by copying the /usr/lib/systemd/system/docker.service to /etc/systemd/system/multi-user.target.wants/docker.service and adding the following switch: --restart=false to the Docker daemon in the `ExecStart` line. Do not forget to reload the systemd daemon afterwards.

Disable and remove the service

When using custom scripts (like we do) we cannot use the `systemctl disable` commands (this is by systemd design — if you want know more please read this comment (https://bugzilla.redhat.com/show_bug.cgi?id=955379#c14)). In our case we need to remove (or rename) the `docker-*.service` files and then run:

```
$ systemctl reset-failed
```

Summary

This was just a small introduction to the Docker and systemd world. I hope you can leverage it to run your services since it's a pretty good choice. I should mention here the geard project (http://openshift.github.io/geard/) which aims to do what I described above (and much, much more).