**An Overview of Docker Container Linking**

Docker consists of a linking system that allows multiple containers to be linked together. This linking system allows connection information to be sent from a source container to a recipient container.

This article will take you through Docker container linking. It will provide an overview of how Docker container linking works and explain how communication works across links.

**Introduction to Docker container linking**

In container orchestration, communication between containers is essential. This is where Docker container linking comes in.

Docker consists of a --link legacy feature that enables two containers to be linked to each other. Once a connection has been established, the connection information can be shared between the two containers.

Docker container linking allows the recipient container to get connection information relating to the source container.

Although Docker introduced a Docker networking feature that enhances communication between containers, container linking is still in use.

It is important to understand container linking since it is a resourceful alternative to networking.

Container linking is not limited to two containers. It can be applied to as many containers as possible.

The linking system can establish a link of multiple containers to enhance communication between them.

**When to use Docker container linking**

As mentioned earlier, there are two ways to establish communication between containers: Docker networking and Docker container linking.

It is important to understand when to use Docker container linking in order to avoid challenges.

You should use Docker container linking in the following instances:

**When you are using the default bridge network in Docker**

A bridge network is a layer device that links two or more network segments. It forwards traffic from one network segment to another.

It can be a hardware or software that can run within the kernel of a host machine.

Docker bridge networks employ a software bridge that enhances communication between containers that are connected to it.

The software bridge prevents containers that are not connected to it from getting or establishing communication.

This means that containers using different bridges cannot communicate with each other.

Docker container linking should not be used when you are using user-defined bridges. These bridges are utilized in Docker networking.

**When you want to share environmental variables**

Docker default bridge networks are the ideal networks for sharing environmental variables between containers.

This is because the user-defined networks do not support the sharing of these variables.

**How Docker container linking works**

Docker container linking works through two main processes: the naming of containers and communication across links.

**Naming of containers**

Docker container linking relies on container names to establish links between the containers.

This means that the naming of containers is necessary.

When you create a container, a name will be created automatically.

The naming of containers provides a unique identity that enables the user to remember. It also enables Docker to refer to two or more containers.

Let’s assume we have two containers named db and web. You can use the names to specify that you want to link db to web.

You can use the --name flag to name a container.

**Communication across links**

Docker uses two methods to enhance communication between the containers: environmental variables and updating the host file.

**Environmental variables**

Docker can enhance communication between containers through the sharing of environmental variables from the source container to the target container.

When a link is created, Docker generates many environmental variables (automatically) in the recipient container depending on the --link parameters that were passed.

All the environmental variables that originate from the source container are exposed to the target container. This enables the target container to discover information relating to the source container.

**Updating the /etc/hosts file**

Apart from sharing the environmental variables, Docker provides a host entry to the /etc/hosts file in regards to the source container. Two entries are added in the /etc/hosts file.

The first one is for the recipient container, which consists of a hostname in the form of a container ID. The second entry employs the link identity to provide information of the source container (IP address).

This information is updated automatically if the source container is restarted. If this is the case, the recipient container will be provided with a reference to the new information of the source container.

This enhances the continuity of linked communication.

**Advantages and disadvantages of Docker container linking**

**Advantages**

It provides an effective way of enhancing communication between containers.

It supports the sharing of environmental variables, unlike in Docker networks.

After installing Docker, the default bridge network is created automatically.

It enhances the isolation of containers that are not linked to a given bridge. Containers that are not connected to the same bridge cannot establish direct communication between them.

**Disadvantages**

With the introduction of the Docker networks feature, container linking may be removed in the future.

The user-defined bridges in the Docker networks feature create better isolation of containers than the default bridge networks in container linking.

**Conclusion**

This article has provided the basics of Docker container linking. The following are the main points we have learned:

Docker container linking allows multiple containers to be linked to each other.

It allows the recipient container to get connection information relating to the source container.

You should use Docker container linking when you are using default bridge networks and when you want to share environmental variables.

Docker container linking works through two main processes: the naming of containers, and communication across links.

Communication across links is achieved through two main ways: sharing environmental variables, and updating the /etc/hosts file.

----------------------------------------------------------------------------------------------------

**Docker Compose**

Docker Compose is used to run multiple containers as a single service. For example, suppose you had an application which required NGNIX and MySQL, you could create one file which would start both the containers as a service without the need to start each one separately.

In this chapter, we will see how to get started with Docker Compose. Then, we will look at how to get a simple service with MySQL and NGNIX up and running using Docker Compose.

**Docker Compose ─ Installation**

The following steps need to be followed to get Docker Compose up and running.

**Step 1** − Download the necessary files from github using the following command –

curl -L "https://github.com/docker/compose/releases/download/1.10.0-rc2/dockercompose

-$(uname -s) -$(uname -m)" -o /home/demo/docker-compose

The above command will download the latest version of Docker Compose which at the time of writing this article is 1.10.0-rc2. It will then store it in the directory /home/demo/.



**Step 2** − Next, we need to provide execute privileges to the downloaded Docker Compose file, using the following command −

chmod +x /home/demo/docker-compose



**Syntax**

docker-compose version

**Parameters**

version − This is used to specify that we want the details of the version of Docker Compose.

**Output**

The version details of Docker Compose will be displayed.

**Example**

The following example shows how to get the docker-compose version.

sudo ./docker-compose -version

**Output**

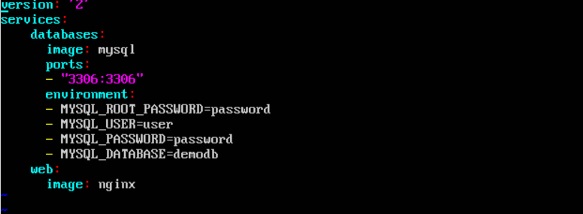
You will then get the following output –



**Creating Your First Docker-Compose File**

Now let’s go ahead and create our first Docker Compose file. All Docker Compose files are YAML files. You can create one using the vim editor. So execute the following command to create the compose file −

sudo vim docker-compose.yml



Let’s take a close look at the various details of this file −

The database and web keyword are used to define two separate services. One will be running our mysql database and the other will be our nginx web server.

The image keyword is used to specify the image from dockerhub for our mysql and nginx containers

For the database, we are using the ports keyword to mention the ports that need to be exposed for mysql.

And then, we also specify the environment variables for mysql which are required to run mysql.

Now let’s run our Docker Compose file using the following command −

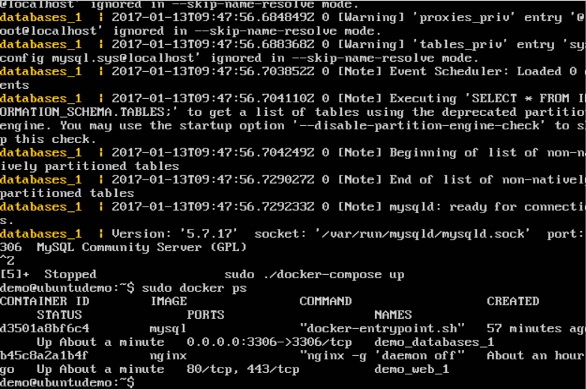
sudo ./docker-compose up

This command will take the docker-compose.yml file in your local directory and start building the containers.



Once executed, all the images will start downloading and the containers will start automatically.

And when you do a docker ps, you can see that the containers are indeed up and running.



-----------------------------------------------------------------------------------------------