Version 1.2, December 6, 2019

Deploying Liferay DXP Using Docker

Liferay

Table of Contents

Introduction

There has been growing interest in using Liferay Digital Experience Platform (DXP) with Docker, a popular software container platform that makes it easier to use containers to create, run, test and deploy applications faster. This guide aims to provide a starting point for companies that would like to use Docker as a runtime environment for Liferay DXP. It will also provide guidance for how a basic Liferay DXP deployment can be modeled using Docker's capabilities.

We will cover several key topics:

- How to create a basic Docker image for Liferay DXP and Elasticsearch
- Options for document storage, including Docker volume plugins and Document implementations not based on a file system
- How to mount configurations into Docker
- How to configure networking

What Is Docker?

Docker is described by its creators as follows:

> *Docker allows you to package an application with all of its dependencies into a standardized unit for software development.*

These units are called Docker images and they can be run as containers in supported operating systems. The image can define one or more network ports that will be exposed from the container (accessible from the network) and other containers or standalone applications. It can then use these ports to interact with the application running inside the container.

Docker is a virtualization concept similar to virtual machines, but instead of trying to mimic physical hardware as closely as possible (which is the aim of traditional virtualization providers like VirtualBox or VMWare), Docker's main goals are:

- Being descriptive in how the Docker image was built. Every image is described as a set of individual configuration steps executed on top of the base image. These steps can be written in the form of a Dockerfile.
- Keeping images smaller than a full-featured virtual machine. This is possible thanks to the fact that Docker containers are sharing the kernel with the host OS. Necessary tools, libraries and/or applications can be installed as needed.

- Quicker startup/shutdown of containers compared to virtual machines. Also, the guest OS used with Docker is often a stripped-down version of the OS, leaving only what's crucial, which helps the speed even more.

Docker containers run natively in all modern Linux systems and supplemental software is available to run containers in OS X- or Windows-based systems.

Docker was released in early 2013 and has gained huge popularity, especially in the microservices-oriented world and among DevOps. It has both supporters and critics, and the use of Docker, especially in production, should be carefully considered.

Shipping Liferay DXP Using Docker

Liferay DXP can operate in a variety of environments and can be configured to run inside a Docker container as well. Liferay DXP is a standard Java-based web application that can run on many application servers and supported operating systems. Java application servers are Java Virtual Machine (JVM) based; success is defined by hosting a JVM with application server binaries inside and Liferay DXP deployed.

Liferay DXP can therefore run in any Docker container that has the JVM installed. Running DXP inside a Docker container is functionally identical to running a standard Linux distribution.

Please note that in order to be able to receive support from Liferay, you will need to select a supported Application server, database, JDK and operating system. Please check Liferay's Compatibility Matrix for more details.

Building Docker Images for Liferay Components

Liferay DXP has two major components which need computational power and therefore are good candidates for becoming separate Docker containers:

- Liferay DXP itself, including the front end and back end, processing all requests coming to Liferay DXP
- The search engine, as a separate system where indexing and searching of Liferay assets is performed

Liferay also needs to store its data persistently, which is handled by:

- Access to an existing relational database, either running in another Docker container or as a traditional database server listening on a known hostname and port. This topic won't be covered in this guide. Please see other Liferay documentation on setting up a database for Liferay DXP.
- Access to location for storing Documents & Media files. This will be covered in more detail later in this guide.

Liferay DXP

Liferay provides official Docker images for Liferay DXP through the Liferay DXP repository on Docker Hub. Issues with these images can be reported through Subscription Services channels. Reported issues will not be assigned a severity and are not subject to a service level agreement since these images are not supported yet through Enterprise Subscriptions. Documentation is available here.

Elasticsearch

In Liferay DXP, the default search engine is an *embedded* Elasticsearch (ES) server. Although this approach is an easy out-of-the-box solution, Liferay will only support a search engine running *outside* of the JVM running Liferay DXP. These two applications need very different tuning parameters and many of the issues stem from sharing the JVM for processing requests and indexing/searching the data in the same JVM. External Solr server engine is supported the same way as in previous versions of the Liferay platform. This document focuses on external Elasticsearch engine setup, but the principle of setting up Solr would be very similar.

Since Elasticsearch should run inside Docker, a separate image with its installation is needed. Although the Docker image can be created manually, for example by writing a Dockerfile, it will be more straightforward to use the official image for Elasticsearch which is available on DockerHub. Moreover, due to the fact that Liferay DXP uses TCP to communicate with Elasticsearch, the VMs of these two applications should be as similar as possible, but at minimum:

- the same vendor (e.g., either Oracle JDK or OpenJDK);
- the same major version (JDK 8 in our case — JDK 7 is not supported in Liferay DXP).

Since OpenJDK is now supported for Liferay DXP, the best approach is to base both images on *openjdk:8-jre*.

Once a Docker container with an Elasticsearch node is started, Liferay DXP needs to be told to use the external Elasticsearch engine and which endpoint of Elasticsearch should be used for interacting with this engine. For the necessary Liferay DXP configuration, check Liferay documentation. Please note that only Elasticsearch 2.2.x instances are currently supported by Liferay DXP.

If deploying multiple Elasticsearch containers to form a cluster of Elasticsearch nodes, the situation becomes more complicated since Docker networking needs to be taken into consideration. Please see section Options for Configuring Networking for more details, which apply both to clustering Elasticsearch and Liferay DXP.

Options for Liferay Document Storage

By default, Liferay DXP stores *Documents & Media* files (their binary content) on the local file system. With default configuration (`FileSystemStore` or `AdvancedFileSystemStore` being used), this will be inside `data/document_library` directory in Liferay DXP's home directory, unless changed using OSGi configurations (i.e., `data/my_own_file_system_structure`). Since all file system data inside the Docker container is ephemeral and available only for the time of the container's life, the data needs to be stored outside of the container to be persistent. There are several options to achieve this task.

Docker Volumes and Volume Plugins

One option is to leverage *Docker volumes*. On the container's start, the volume will be mounted from the container to the host, possibly to a specific mount point on the host. This will make sure that any change made in the mounted folder from inside the container will be visible in the directory on the host and vice versa. A Docker volume can be defined by adding a `VOLUME` directive into the Dockerfile. This solution is, however, host-dependent and requires a setup outside of the Docker image to share the volumes between multiple Containers or even Docker hosts.

If running a cluster of Liferay DXP nodes, the *Documents & Media* directory needs to be shared between all the nodes. Docker offers support to enable Docker volumes to persist across multiple Docker hosts, using *Volume plugins*. Each of these plugins offers different implementation on how the defined volumes can be shared between multiple containers and hosts. Please check Docker documentation for more details.

This principle is similar to the use of Network-Attached Storage (NAS) devices, SAN drives or NFS-mounted volumes in a non-Docker deployment of Liferay DXP.

Alternate Store Implementations

Another option is to use an implementation of Liferay store which does not use the local file system. That way, the data is not stored locally in the Docker container and even when it ceases to exist, the data still exists in the outside system.

Some examples could be `S3Store`, storing data in a configured S3 bucket inside Amazon Web Services, or `DBStore`, storing binary files in Liferay's relational database. Please note that each of the stores has its usage specifics and `DBStore` is not recommended for use in production environments with moderate or heavy use of *Documents & Media*. Please see Liferay documentation for further details.

Options for Mounting Configurations Into Docker

In Liferay DXP, the configuration is read from three basic places:

1. The legacy properties in *portal.properties* and its overrides using various *portal-*.properties*.
2. Since Liferay began to support reading configuration properties from an OS environment, it is possible to set up environment variables into the machine hosting Liferay, following the name convention described in the linked JIRA ticket. The Setup Wizard could be set in this way: `LIFERAY_SETUP_PERIOD_WIZARD_PERIOD_ENABLED=false`.
3. Files loaded by *OSGi Config Admin*, placed inside *[liferay_home]/osgi/configs*.

One option for configuring an existing Liferay bundle inside a Docker image is by identifying the Docker volumes that can be used to propagate files and folders, and mounting them from the host into the desired location in the Liferay bundle inside of the container. Check Docker volumes documentation for more details.

Another option is to set up a configuration system (such as *etcd, Consul* or *ZooKeeper*) to hold the important configuration keys and values, and then configure local daemon (such as *confd*) to read the configuration and write the necessary files into the Liferay DXP bundle inside the container.

Please note that for production deployments, Liferay recommends placing all relevant configurations into the Liferay bundle directly and including it in the Docker image as a whole, instead of mounting additional configurations into a container and possibly changing the files during Liferay's runtime. It might,

however, be useful during the development phase, when testing various configuration changes (assuming Liferay DXP supports the reloading of a given type of configuration from the file).

Options for Configuring Networking

One of the basic networking needs of Liferay DXP is the exposure of the app server ports of the HTTP or AJP connectors, which is straightforward in Docker as shown in the Exposing Liferay ports section. However, there are some additional steps if you plan to run Liferay DXP in a clustered setup, since the operation of Cluster Link networking will also need to be configured.

With default configuration, Cluster Link relies on multicast for nodes discovery and UDP for communication between cluster nodes. This setup might not be reliable in virtualized environments, like Docker, so it's safer to opt for using alternate discovery protocols (JDBC_PING or FILE_PING) and TCP unicast for communication. Since FILE_PING would require a shared file system between all Liferay node containers, it's usually easier to set up JDBC_PING, requiring similar configuration as configuring relational database access for Liferay DXP itself.

Once the discovery of Liferay nodes is handled, the jGroups' ports must be opened for incoming network traffic and the bind address of jGroups must be resolvable to an IP that is reachable by all other Docker containers (other Liferay nodes running in Docker containers) within the defined Docker network. Cluster Link uses the jGroups library for communication between the nodes. It typically needs two ports to bind to — one for control channel, one for transport channel.

Docker Networking

Docker added container networking capabilities into the base product, and it is now supported natively by the Docker engine itself. Several network drivers are available and are very well described in official Docker documentation.

Networking on one Docker host (all containers running on the same machine) is typically not problematic and can be handled using networks with the *bridge* driver. Use either the default *bridge* network (named the same as the underlying driver it uses) or create a custom one with the *bridge* driver. All Docker containers started into this network can immediately communicate with each other. A private network is created and each new container is given a new address within this network, just as a DHCP client would do in a standard networking infrastructure.

Multi-host networking is more involved. The *bridge* network driver does not support this type of networking. Instead, an *overlay* driver needs to be used. This driver requires an external service being available (a key-value store) and uses it to organize the routing of the traffic between multiple Docker hosts. After being set up, the container will have the same logical view of the network, even though it is physically spanning over multiple Docker hosts. Please check Docker documentation for detailed information.

Docker also supports the installation and use of *Networking plugins* which offer additional features and possibilities for how to make networking between Docker containers work. See Docker documentation for more details.

Legacy Container Links

Before explicit networking support in Docker was implemented, users could use Container links. Docker containers could "import" ports exposed by another running Docker container by aliasing them into the container, including the aliases for the hostnames.

Please note that using container links is discouraged in Docker's documentation and only works on the default bridge network, for backward compatibility.

Summary

This document outlines how Liferay DXP can be deployed and run as a Docker container, together with an external Elasticsearch in a separate Docker container. The necessary steps were documented to build a Docker image with a Liferay DXP bundle and Oracle JDK installed inside and how to run a Docker container based on this image.

Some advanced topics were also discussed, like options for Document & Media storage within Docker infrastructure and possible ways to plug configuration into a Docker container with Liferay DXP. Important considerations to take into account when defining and implementing networking between Liferay Docker nodes were listed.

Moving Forward

Liferay DXP Cloud

If you don't want to worry about building and orchestrating your Docker containers, let Liferay DXP take care of your infrastructure needs. With our official Docker images, you don't need to build these yourselves. Rely on a secure PaaS solution that provides autoscaling, development tools, environments, monitoring and more for Liferay DXP. To learn more, contact sales@liferay.com.

Liferay Global Services

Learn how Liferay's Global Services team can support your Liferay DXP project. Our consultants work alongside your team to create a personalized solution for your business. Contact us at liferay.com/consulting.

Liferay makes software that helps companies create digital experiences on web, mobile and connected devices. Our platform is open source, which makes it more reliable, innovative and secure. We try to leave a positive mark on the world through business and technology. Hundreds of organizations in financial services, healthcare, government, insurance, retail, manufacturing and multiple other industries use Liferay. Visit us at liferay.com.

191206