**Lab2**

Deploying Your First Web App to Tomcat on Docker

Docker is a platform for developers and sysadmins to develop, deploy, and run applications with containers. As we explained in the previous post, Docker containers wrap up software and its dependencies into a standardized unit for software development that includes everything it needs to run code, runtime, system tools, and libraries.

In this post, we are going to step through a basic tutorial on getting a web application running on Tomcat Docker Container.

## **Tomcat**

The Tomcat server is the most widely used open-source implementation of the Java Servlet, Java Server Pages, Java Expression Language, and Java WebSocket technologies. Apache Tomcat software powers numerous large-scale, mission-critical web applications across a diverse range of industries and organizations.

The sample Tomcat application we will be using for this exercise is included in a git repository so that you can run through this tutorial easily.

In Tomcat, we must move the war file to the CATALINA\_BASE/webapps directory. Tomcat will then install it automatically and deploy the application for you.

We are going to use Tomcat 9.0 for this exercise.

## **Objective**

By the end of this Lab, you should be able to:

1. Understand the basic concepts of Docker containers
2. Run containers using Docker images
3. Get Tomcat server running on a container
4. Deploy web application on the Tomcat server
5. Build your own Docker images using Dockerfile
6. Mapping ports from container on to the host machine

## **Install Docker**

We are going to use Docker Community Edition (CE) as it is ideal to get started with Docker and experimenting.

Download and install Docker from [https://docs.docker.com/install/](https://www.google.com/url?q=https://docs.docker.com/install/&sa=D&ust=1540975110567000)

Choose the appropriate installation method depending on the OS you are using.

Once the installation is complete, open a command prompt terminal and type the command as below. Output like below verifies that your installation went ok.

|  |
| --- |
| $ docker run hello-world  Unable to find image ‘hello-world:latest’ locally latest: Pulling from library/hello-world d1725b59e92d: Already exists Digest: sha256:0add3ace90ecb4adbf7777e9aacf18357296e799f81cabc9fde470971e499788 Status: Downloaded newer image for hello-world:latest  Hello from Docker! This message shows that your installation appears to be working correctly. |

As explained further down in the output generated, many steps were carried out in that one command.

To generate this message, Docker took the following steps:

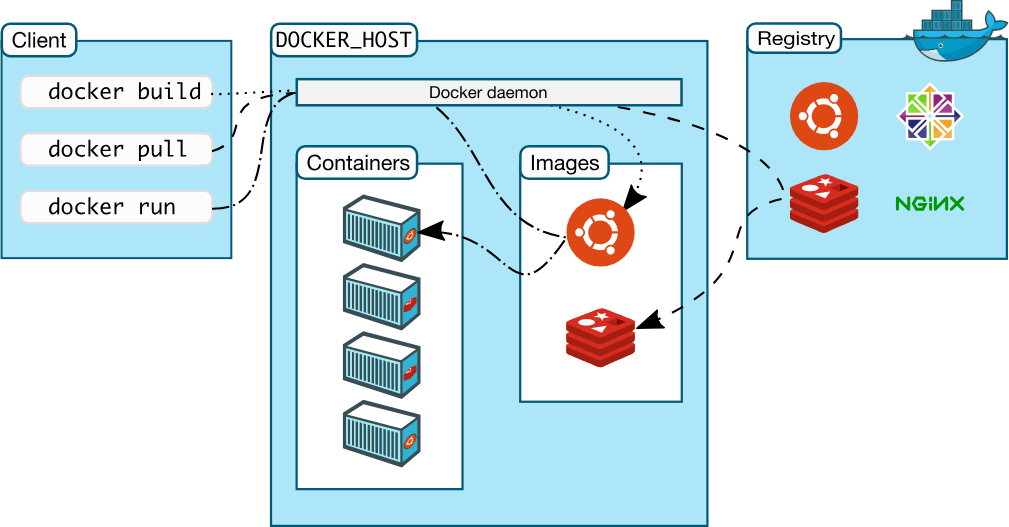
1. The Docker client contacted the Docker daemon.
2. The Docker daemon pulled the “hello-world” image from the Docker Hub.
3. The Docker daemon created a new container from that image which runs the executable that produces the output you are currently reading.
4. The Docker daemon streamed that output to the Docker client, which sent it to your terminal.

Don’t worry if the above is a mouthful, here is all you need to know for now.

Docker Engine is a client-server application with these major components:

* The Docker daemon is a service that runs on your host operating system. When you type any command, it is interpreted by the demon, and it takes necessary actions.
* A REST API to talk to the daemon and instruct it what to do.
* A command line interface (CLI) client (the docker command).

Additionally, Docker Hub is the place where open Docker images are stored. You can pull images, make changes and push them back into this repository – sounds much like GitHub/Gitlab isn’t it?



## **Source Code**

All the code and the sample application needed for this tutorial is available in:

<https://gitlab.com/123Balu42/docker.git>

Clone it to your local computer

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| $ git clone https://gitlab.com/123Balu42/docker.git  $ ls  Dockerfile README.md spring-mvc-maven.war **Image** First, we search docker hub for an official image of Tomcat. So, what is an image?  To use a programming metaphor, if an image is a class, then a container is an instance of a class. Images are created with the build command, and they’ll produce a container when started with run     |  | | --- | | $ docker search tomcat |   One of the results you should see is that of an official image.   |  | | --- | | Tomcat Apache Tomcat is an open source implementation… [OK] |   Alternately, you could search directly on Docker Hub website. Tomcat image information is available in [https://hub.docker.com/r/\_/tomcat/](https://www.google.com/url?q=https://hub.docker.com/r/_/tomcat/&sa=D&ust=1540975110573000)  Checking the documentation for the Tomcat page, we see that Tomcat built on the alpine version of Linux is smaller in size and ideal for experimentation. **Dockerfile** Once we have identified the image to use, we can write a Dockerfile specifying the base image to be used, the webapp to be used, etc.  It can build images by reading the instructions from a Dockerfile. Dockerfile contains all the commands a user could call on the command line to assemble an image. Using docker build users can create an automated build that executes several command-line instructions in succession.  Our simple Dockerfile is as follows:  $ cat Dockerfile   1. The FROM instruction initializes a new build stage and sets the Base Image for subsequent instructions. 2. The LABEL instruction sets the Author field of the generated images. You could use any key-value pair in labels. 3. The ADD instruction copies new files, directories, or remote file URLs from <src> and adds them to the filesystem of the image at the path <dest>.   In our case, we are adding the sample webapp and placing it in the folder /usr/local/tomcat/webapps/ on the container. That is because according to the Tomcat documentation, the War should be placed under CATALINA\_BASE/webapps. It will be automatically expanded and deployed. From the Tomcat image documentation, we know that the default path CATALINA\_BASE corresponds to /usr/local/tomcat on the container.   1. The EXPOSE instruction informs Docker that the container listens on the specified network ports at runtime. 2. The CMD instruction specifies what to run when the container (not the image) is run. In our case, Tomcat server is started by running the shell script that starts the web container. There can only be one CMD instruction in a Dockerfile.   Don’t confuse RUN with CMD. RUN actually runs a command at build time. **Build** The build is run by the Docker daemon, not by the CLI. It downloads any images that are necessary and also executes the commands specified in the Dockerfile.  In the below command, the Dockerfile we created earlier is used (Docker daemon looks for Dockerfile specified by the current directory using a dot) and the newly built image is tagged mywebapp.  $ docker build -t mywebapp  If this is the first time you are running this, it might take a few minutes to complete as the Tomcat image (and any of its dependencies) has to be downloaded.  You can now verify that the mywebapp image is built and ready to be used. Note that at this point, you have only built the image, there is no running container.   |  | | --- | | $ docker image ls |  **Run the Container** The CLI has a command called run which will start a container based on a Docker Image. The structure is docker run <options> <image-name>.  As we mentioned before, the EXPOSE instruction in the Dockerfile doesn’t actually publish the port. To so that when running the container, use the -p flag on docker run to publish and map one or more ports.  So for mapping the container port 8085 for the mywebapp image to port 80 on the host machine, we execute:     |  | | --- | | $ docker run -d -p 80:8085 mywebapp | | Open http://<public ip>:80/ in a browser to see the sample web application running.  Graphical user interface, text, application, website  Description automatically generated | | **Note: Use public ip of ec2 instead of localhost in url** | |