UDC 004.021

USING DOCKER AND JENKINS IN MODERN SOFTWARE DEVELOPMENT

PAVEL TALAIKA, IRYNA BURACHONAK
Polotsk State University, Belarus

In this article, we discuss complete software development lifecycle and the role of such tools as Docker Container Platform and Jenkins for Continuous Integration.

In the process of software development, it is essential to delineate the stages the project is going through before being deployed in a real life production environment. Modern software engineering tools facilitate building and deployment of the projects. We will demonstrate the lifecycle of our project from writing source code to its deployment using Docker and Jenkins.

Docker is a lightweight, autonomous and executable software suite that includes all the necessary tools to run and deploy your projects, it provides source code editing, execution environment, system tools and various libraries. Using this suite, one can easily manage architecture components of the application [1]. Docker solves multiple problems relating to configuring execution environment to interact with a database, setting up application server and GUI. Configuring execution environment locally may be tedious and time-consuming and may also lead to unexpected and hard to tackle problems. If you are developing a project in a team, using Docker is a must.

Jenkins is an open source automation server [2]. Jenkins provides the programmer with hundreds of plugins that facilitate deployment and automation of any project.

The project was developed using Java 1.8, Spring Boot and Spring Data. The database layer is represented by PostgreSQL, the project was build using Maven.

At this point, we have all the tools we need. The first step is project configuration, as a prerequisite one needs to have basic knowledge of Docker and terminal commands of the required operating system. Downloading the image with Jenkins and Docker Store is also a requirement. After having downloaded the image, the container can be started by issuing the *docker run* command and specifying options like ports, container names and the path to the image itself. If it is necessary to use additional programs, one can write a Dockerfile.

During initial stages of the development, a need for additional configuration has arisen, so the Docker file had to be written, moreover having written the Dockerfile allowed for automatic loading of the required plugins for the project. Using plugins is one of the necessary conditions for building and deploying the project. In order to automatically download all the necessary plugins into the project's root directory, one needs to create the *plugins.txt* file that specifies the necessary plugins in the following format *plugin_name:version*, each new plugin needs to be specified on the new line. Commands that need to be added to Dockerfile are described below:

```
COPY ./plugins.txt /usr/share/jenkins/ref/plugins.txt
RUN /usr/local/bin/install-plugins.sh</usr/share/jenkins/ref/plugins.txt
```

After starting the built image, plugins specified in plugins.txt will be automatically downloaded at start-up.

Next we have to create Tomcat and PostgreSQL image for deployment and data storage which is built using .war file generated by Maven. Maven plugin that is installed in Jenkins is responsible for building the project. One should also bear in mind that all containers should create the interconnected system for controlling the building and deployment processes, in order to achieve that you need to forward the ports from Tomcat to PostgreSQL and bind Jenkins by using common volume which can be done by running the Jenkins container with *--volumes-from=tomcat:rw* option and running Tomcat with *–v=tomcat-data:/usr/local/tomcat/webapps:rw* option [3].

Let's create the container with PostgreSQL by issuing the command as described below:

```
docker run -p 5432:5432 --name postgres -e POSTGRES_PASSWORD=postgres -d postgres
```

The next step is to create Tomcat container which is done by running the command as described below:

```
docker run -it -p 8080:8080 --name tomcat -v tomcat-data:/usr/local/tomcat/webapps:rw --link postgres:postgres -d tomcat:latest
```

--link option lets us connect Tomcat and PostgreSQL which also allows volume tomcat-data to be available in Jenkins [4].

Command for running Jenkins are as follows:

docker run -p 8888:8080 -p 50000:50000 --name=jenkins-master --mount source=jenkins-log,target=/var/log/jenkins --mount source=Jenkins-data,target=/var/jenkins_home --volumes-from=tomcat:rw -d jenkins-artteam

As you can see in the code above, data storage and backup volumes are also created. After start-up, Jenkins will be available at http://localhost:8888/.

After having configured the container system, one needs to configure spring.datasource and application.properties and pom.xml to be able to build the project as a .war file.

For more flexibility, VCS is used. When building the project using VCS one needs to specify what changes are affected and what branch needs to be built. Under "Project management" we specify git repository and credentials such as user and password, credentials are needed to log in to Jenkins and configure the build item. Under "Build" we specify the plugin that will build the project. Under "Add build stage" we specify the location of the script which will copy the generated .war file into the volume tomcat-datastartup directory if the build was successful (Figure).

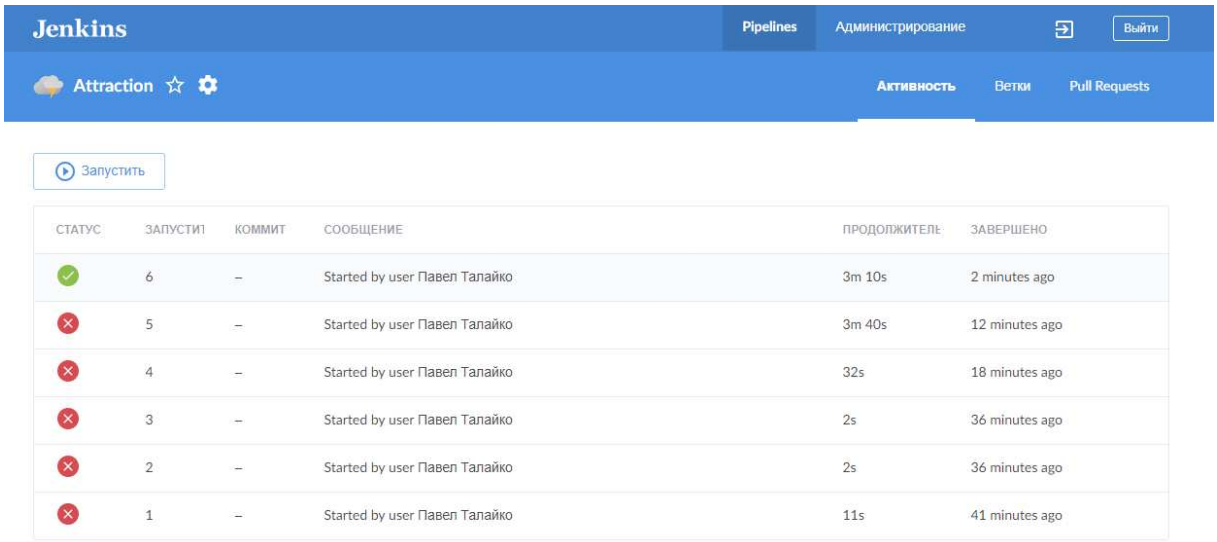

Figure. – List of builds in Jenkins dashboard

Conclusion. The final product of this work is a full-fledged automated docker-container system that facilitates control over project building and deployment, its VCS system allows to track changes that can potentially lead to failed builds. Jenkins also allows for project testing; users of the system can create certain stages that need to be completed in the lifecycle of the project build before being released.

REFERENCES

1. Docker Documentation. Overview of Docker editions [Electronic resource]/ – Access mode: https://docs.docker.com/install/overview/. – Access date: 10.01.2019.
2. Jenkins Documentation. Getting started with the Guided Tour [Electronic resource]. – Access mode: https://jenkins.io/doc/pipeline/tour/getting-started/. – Access date: 11.01.2019.
3. Docker Documentation. Use volumes [Electronic resource]. – Access mode: https://docs.docker.com/storage/volumes/. – Access date: 11.01.2019.
4. Docker Documentation. Use bridge networks [Electronic resource]. – Access mode: https://docs.docker.com/network/bridge/. – Access date: 11.01.2019.