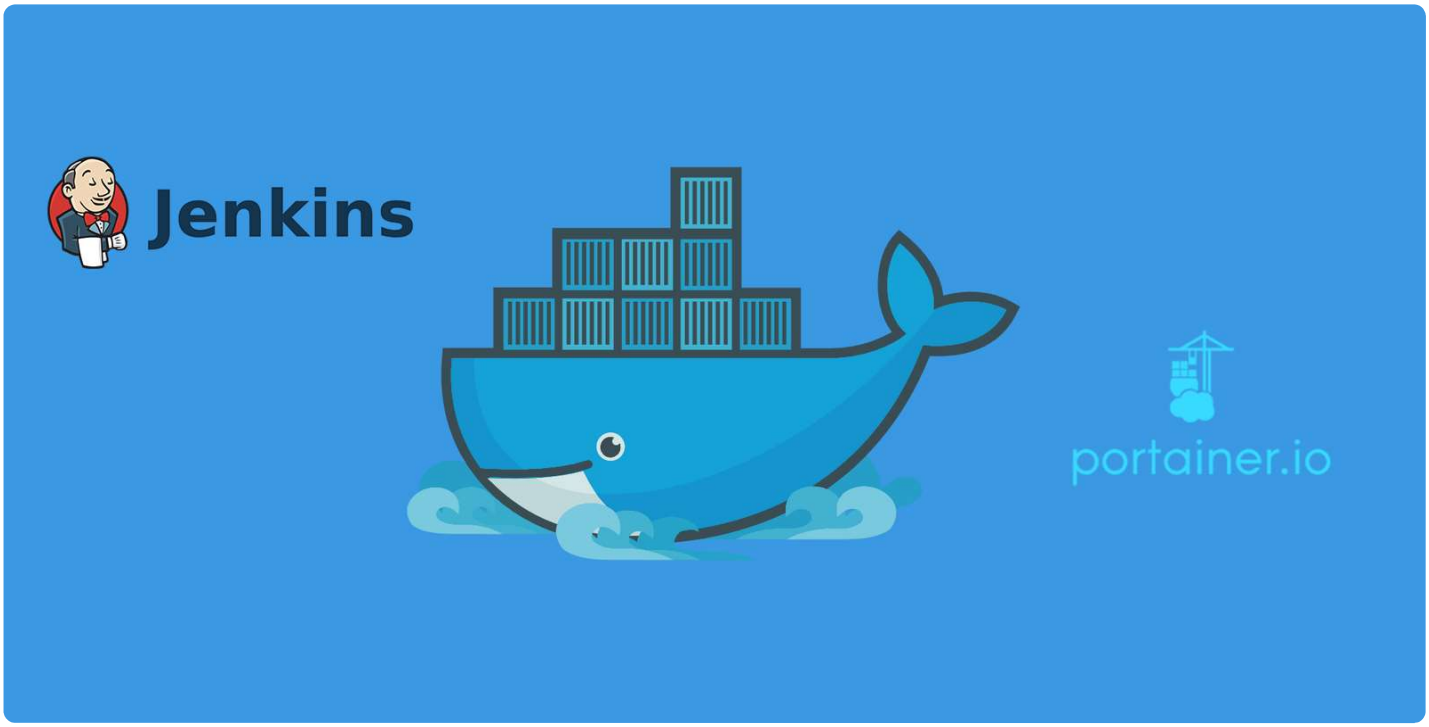
Scalified (https://scalified.com/)

Building Jenkins Pipelines – Part 1. Setting Up Docker Swarm

October 8, 2018 in DevOps (https://scalified.com/category/devops/), Docker (https://scalified.com/category/docker/), Software Development (https://scalified.com/category/software-development/)

Introduction

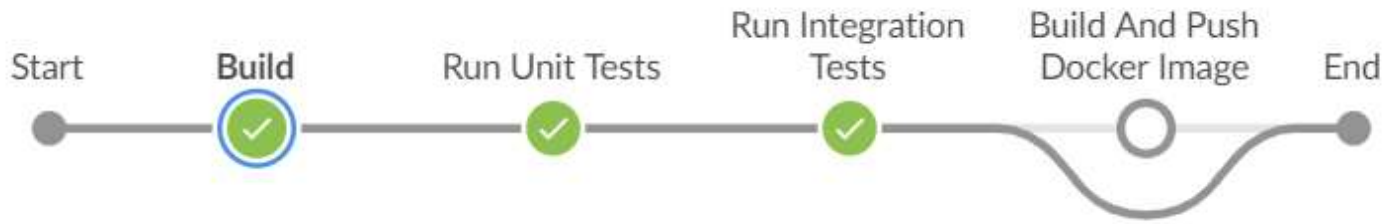


It’s difficult to overestimate the importance of continuous integration and continuous delivery in modern development practices. However, proper implementation of these practices might be not so easy for many software development teams. One of the possible ways to achieve this is to use open source solutions like Jenkins Pipelines which is free and extensible, though still requires some time and knowledge.

Problematics with CI/CD configuration

Only a few ears ago, setting up CI/CD for the project was not more than just installing and configuring standalone build server such as TeamCity or regular Jenkins. Finally, the final configuration included building artifacts, running unit tests, execution of integration tests, e2e tests, deployment to multiple environments and even production releases. The typical case was to have different build agents or agent pools per technology stack, which were shared among departments. Such a setup is still quite common nowadays.

Many teams still use build agents for building the project, having all the necessary software installed on them. One of the main problem here is that building environment is shared between different builds and thus can cause unpredictable results. Consider the case when integration or system tests that use the same database are running simultaneously on the same agent. In such situation an extra effort required to make them run successfully. One can implement an every time database instance creation with unique name to have it isolated, another can go with tests data clearing after execution. But all these ways doesn't solve the problem completely and, moreover, increase the general complexity of the project infrastructure.



In order to address all these problems, many organizations use Docker to unify their build and test environments across machines, and to provide an efficient mechanism for deploying applications. Starting with Jenkins Pipelines versions 2.5 and higher, Jenkins Pipelines has built-in support for interacting with Docker which perfectly suits to fit the purpose.

In Part 1. of this article we are going to shed some light on setting up and configuring Docker Swarm – a clustering feature of Docker and Portainer (https://portainer.io/) – management UI for it.

Docker Swarm Set Up

Prerequisites

You may choose any environment you like to run Docker Swarm. The only thing you should avoid is using Docker in Docker (DID) set up. While it might seem to be a convenient solution in terms of deployment and configuration, there are many things, which could happen running DID setup such as conflicts, data corruption etc.

We shall go with what we consider to be the best prerequisites:

- Centos OS on physical machine
- Docker CE installed (you may refer Get Docker CE for CentOS manual)
- Domain name defined (we will use my.company.com in this tutorial)

Certificates Generation

First of all, we want our Docker Swarm environment to be secure. Therefore, the first step to take would be certificates generation. Certificates secure SSL connections to Docker Swarm Server, so that client needs to provide the valid client certificates which server will then validate applying server certificates.

You may refer original Docker Documentation for generating certificates – Create a CA, server and client keys with OpenSSL or follow the steps below:

Server Certificates

1. Create /etc/docker/certificates/server directory:

```
mkdir -p /etc/docker/certificates/server
```

2. Navigate to the created directory:

```
cd /etc/docker/certificates/server
```

3. Generate CA private and public keys:

```
openssl genrsa -aes256 -out ca-key.pem 4096
```

```
openssl req -new -x509 -days 1000 -key ca-key.pem -sha256 -out ca.pem
```

4. Create a server key and certificate signing request (CSR):

```
openssl genrsa -out server-key.pem 4096
openssl req -subj "/CN=my.company.com" -sha256 -new -key server-key.pem -out server.csr
```

5. Sign the public key with CA:

```
echo subjectAltName = DNS:my.company.com,IP:127.0.0.1 >> extfile.cnf
echo extendedKeyUsage = serverAuth >> extfile.cnf
```

6. Generate the key:

```
openssl x509 -req -days 1000 -sha256 -in server.csr -CA ca.pem -CAkey ca-key.pem -CAcreateserial -out
server-cert.pem -extfile extfile.cnf
```

> *Note, `my.company.com` must be replaced with your valid domain name.*

Client Certificates

1. Create /etc/docker/certificates/client directory:

```
mkdir -p /etc/docker/certificates/client
```

2. Navigate to the created directory:

```
cd /etc/docker/certificates/client
```

3. Create a client key and certificate signing request:

```
openssl genrsa -out key.pem 4096
openssl req -subj '/CN=client' -new -key key.pem -out client.csr
```

4. Create an extensions config file:

```
echo extendedKeyUsage = clientAuth >> extfile.cnf
```

5. Sign the private key:

```
openssl x509 -req -days 1000 -sha256 -in client.csr -CA ../server/ca.pem -CAkey ../server/ca-key.pem -
CAcreateserial -out cert.pem -extfile extfile.cnf
```

Once certificates are generated, you may remove certificate signing requiests:

```
rm -f /etc/docker/certificates/client/client.csr /etc/docker/certificates/server/server.csr
```

In order to work with Docker REST API in browser, client's cert.pem must be exported into PFX format and then added into Trusted Root Certification Authorities:

```
openssl pkcs12 -export -in cert.pem -inkey key.pem -out cert.pfx
```

Docker Service Set Up

In order to configure Docker service perform the following steps:

1. Create the file Docker daemon configuration file in /etc/docker/daemon.json with the following content:

```
{
    "debug": false,
    "tls": true,
    "tlsverify": true,
    "tlscacert": "/etc/docker/certificates/server/ca.pem",
    "tlscert": "/etc/docker/certificates/server/server-cert.pem",
    "tlskey": "/etc/docker/certificates/server/server-key.pem",
    "hosts": ["tcp://0.0.0.0:2376", "unix:///var/run/docker.sock"]
}
```

2. Enable docker daemon at startup:

```
systemctl enable docker
```

3. Start docker service:

```
systemctl start docker
```

Enabling Docker Swarm Mode

Ability to scale easily is the most critical point to consider when working with virtual or containerized systems. Docker Swarm Mode provides possibility to establish and manage a cluster of Docker nodes as a single virtual system. In order to enable it, we should execute the following command once:

```
docker swarm init
```

Portainer Integration

Once we have Docker Server running, it would be great to have some sort of UI for management. Portainer is currently the best option. It is an open-source lightweight management UI which allows to easily manage Docker hosts or Swarm clusters.

Most noteworthy here is that Portainer doesn't require any special configuration and can be started as a container inside the existing Docker cluster.
In order to retain Portainer data it is better to create some folder on Docker Server:

```
mkdir -p /opt/portainer/data
```

Then, simply install Portainer as a Docker service, so that it would start automatically with Docker Server startup:

```
docker service create --name portainer --publish mode=host,target=9000,published=80 --constraint 'node.role == manager' --mount type=bind,src=/var/run/docker.sock,dst=/var/run/docker.sock --mount type=bind,src=/opt/portainer/data,dst=/data portainer/portainer -H unix:///var/run/docker.sock
```

Finally, if you don't like keeping history records and showing them in Portainer interface, you can simply limit the task history to preserve the specified number of records:

```
docker swarm update --task-history-limit 1
```

Portainer Integration

Congratulations, our Docker Swarm cluster is now prepared, so we're ready to setup Jenkins and configure some Jenkins Pipelines. Please share this article in social media if it was helpful, and also feel free to ask your questions regarding the topic.

(https://scalified.com/team/vladyslav-baidak/)
Vladyslav Baidak
Backend Engineer at Scalified

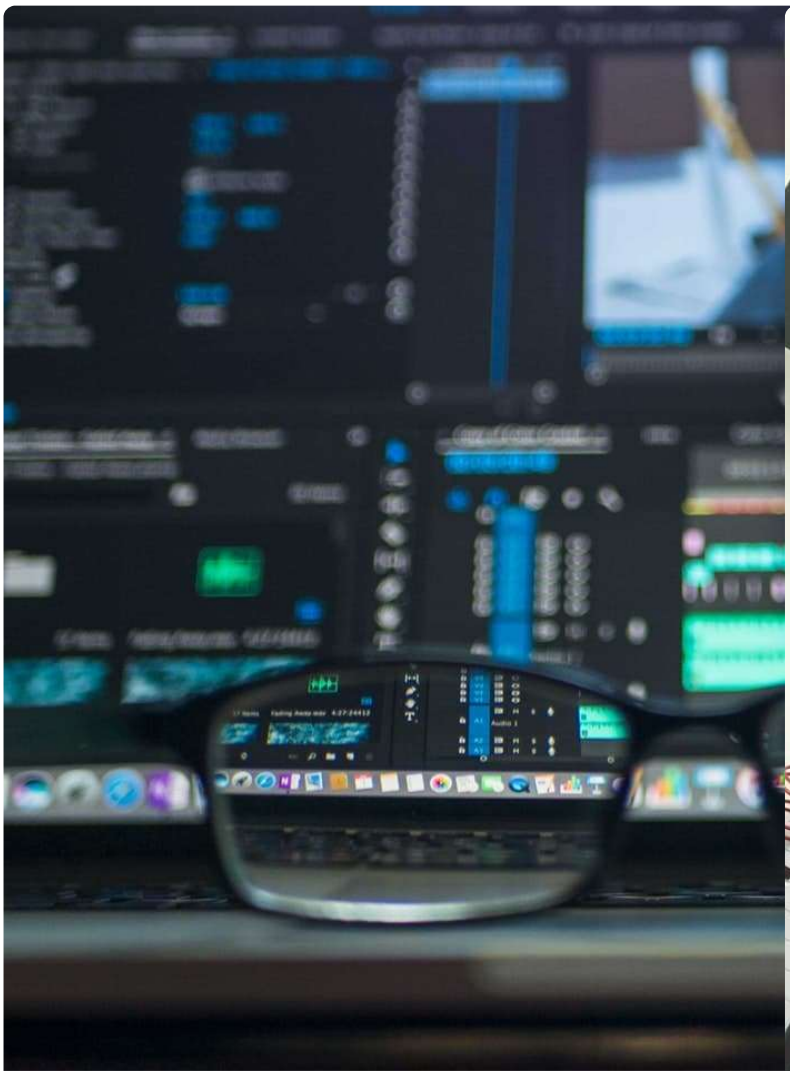
Tags: **continuous delivery (https://scalified.com/tag/continuous-delivery/)**, **continuous deployment (https://scalified.com/tag/continuous-deployment/)**, **continuous integration (https://scalified.com/tag/continuous-**

integration/), **development (https://scalified.com/tag/development/)**, **docker (https://scalified.com/tag/docker/)**, **jenkins (https://scalified.com/tag/jenkins/)**

Share on Facebook (https://www.facebook.com/sharer/sharer.php?u=https://scalified.com/2018/10/08/building-jenkins

Share on Twitter (https://twitter.com/share?url=https://scalified.com/2018/10/08/building-jenkins-pipelines-docker-swa

More recent stories



(https://scalified.com/2017/11/01/nodejs-typescript-intellij-debug/) (https://scalified.com/2017/11/14/centering-images-html-css

November 1, 2017

Debugging Node.js backend with Typescript and IntelliJ (https://scalified.com/2017/11/01/nodejs-typescript-intellij-debug/)

Read More (https://scalified.com/2017/11/01/nodejs-typescript-intellij-debug/)

November 14, 2017

Centering images with HTML and CSS (https://scalified.com/2017/11/14/centering-images-html-css/)

Read More (https://scalified.com/2017/11/14/centering-images-html-css/)

Scalified (https://scalified.com/)

Home (/)

About Us (https://scalified.com/about/)

Portfolio (https://scalified.com/portfolios/)

Blog (https://scalified.com/blog/)

(https://www.facebook.com/scalified/) (https://www.linkedin.com/company/scalified/)

(https://github.com/Scalified) (https://plus.google.com/u/2/116759277817676986149)

Outsource software development based in Ukraine