



Fullstack Cluj Meetup

Continuous Deployment with Jenkins and Docker

Date: **01.11.2016**

Type: **Workshop**

Coordinator: **Cosmin Harangus**

Why continuous deployment

The days of manually deploying your code are long gone and even if your daily job does not require you to know what happens to your code past pushing it to a source control repository it is important to understand the entire flow.

This is especially useful if you want to build a startup where automating your deployments helps you release features faster and keeps you concentrated on your code removes unnecessary distractions out of your way.

In order to have a good continuous deployment pipeline you first need to understand the entire structure of your application so that you can script each step needed to build and deploy your code to another server.

Things you should watch for:

- Does your application require any server dependencies? JVM, PHP Server, Apache, etc. How do you install these dependencies on a new server?
- Are there any configuration files that should change based on the environment you want to deploy on?
- Is your application composed of multiple moving parts? Can these parts be deployed individually?
- What are the steps you want to include in your build/deploy pipeline?
- How do you test each step of the pipeline?

Workshop goals

For this workshop we will try to automate a simple web application composed of the following components:

- RESTFul API written in NodeJs talking with a RethinkDb database
- HTML frontend
- Nginx proxy to have the frontend and the API on the same domain
- Optionally deploy an nginx-proxy container to map virtual hosts to running containers or handle SSL support.

In order to automate the deployment of this application we will install and configure the following:

- Docker, Docker Machine and Docker Compose
- Create two Docker machines
- A Jenkins master container where we will setup user access, install necessary plugins

- A Jenkins slave container that will handle our running jobs
- Setup two Jenkins jobs for each of our repositories
- Add a Jenkins file, build images and push them on a Docker Registry
- Connect to the staging or production server and deploy the images.

Prerequisites

In order to get started we first need to install the following:

- Docker: <https://www.docker.com/products/overview>
- Docker Machine: Installed with Docker or Docker Toolbox (<https://docs.docker.com/toolbox/overview/>)
- Docker Compose: <https://docs.docker.com/compose/install/>
- VirtualBox from Oracle: Installed with Docker Toolbox or individually.

These are required because everything we will be doing depends on Docker. Jenkins will run as a Docker container, our application will be wrapped in Docker images and deployed using Docker Compose on a separate Docker VM or external server.

Once we build our API, Frontend and Proxy we will push them to a Docker Registry and pull them on the target server and then use Docker Compose again to update the running containers.

The code for this workshop is available on github at the following url <https://github.com/FullStackCluj/Meetups> in the 2016-11-01 Continuous Deployment with Jenkins and Docker folder.

Download the repo locally so that you have the code we will use to follow along with the next steps.

Jenkins Master-Slaves Cluster

From the 01-install-jenkins folder follow the steps from the Readme file.

First start two VMs: one for Jenkins Master and Slave and one we will use as a deployment server.

After the VMs are up and running check the IPs of each VM and add the following lines to your hosts file:

```
> 192.168.99.100 jenkins.fullstackcluj.ro  
> 192.168.99.101 staging.fullstackcluj.ro
```

Once that is done we connect to the first server and install Jenkins Master and Slave in two containers.

The slave can be deployed to other servers or other VMs in order to have a distributed environment and have multiple simultaneous builds running. For now we will only install it on one server.

First run the `./start-proxy.sh` script to install create a container that allows us to have multiple domains on the same server without binding ports to the server.

Then start the master by running `./start-master.sh`. Once the container is running navigate to `jenkins.fullstackcluj.ro` and follow the steps on the screen.

To get the initial admin password execute the following command:

```
> docker exec -it jenkins_master_1 cat /var/lib/jenkins/secrets/initialAdminPassword
```

Also create an admin user and choose the plugins you want to install. For now you can use the recommended ones.

From the Manage Plugins section:

- Install the Swarm Plugin and restart the jenkins master

From the Configure Global Security section:

- Set the "TCP port for JNLP agents" value to Fixed with a specific value for the port. Ex: 50000

From the Manage Users section:

- Add a new user called slave with the password set in the `./start-slave.sh` file

Run the `./start-slave.sh` script and check the nodes section in Jenkins to make sure it connected successfully.

Our demo application

In the `02-demo-app/` folder you can find the code for the demo application.

The code was also deployed on github in the following repos:

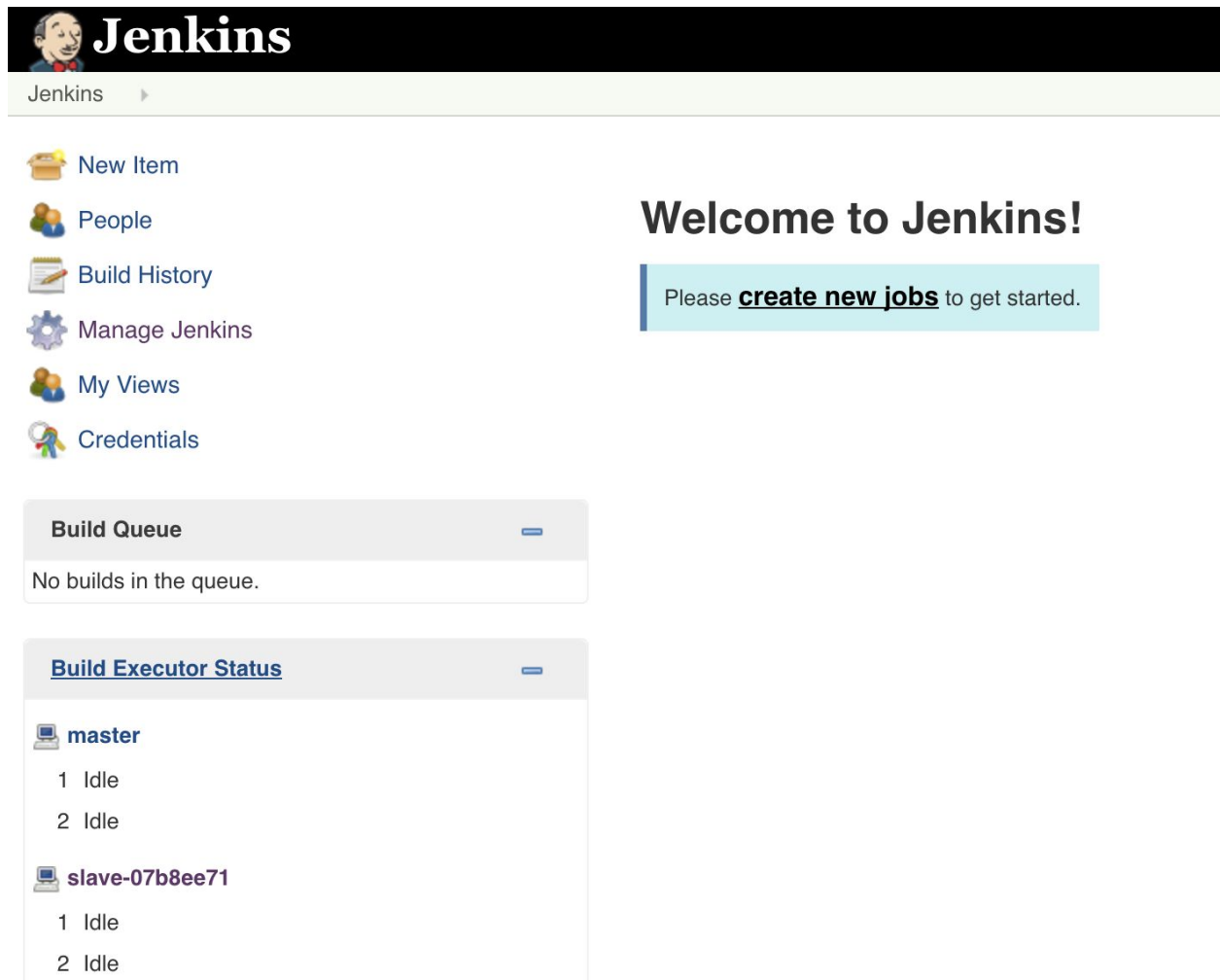
- <https://github.com/FullStackCluj/nodejs-demo-api>
- <https://github.com/FullStackCluj/nodejs-demo-web>

Each repo contains a develop branch and a master branch.

It also contains a Dockerfile that is used to wrap the application in a Docker image.

Creating our first build job

Log in to your Jenkins admin area and you should see the following screen:



The screenshot shows the Jenkins dashboard. At the top is a black header with the Jenkins logo and name. Below it is a light green navigation bar with the text 'Jenkins' and a right arrow. On the left side, there is a vertical list of links with icons: 'New Item' (box icon), 'People' (people icon), 'Build History' (document icon), 'Manage Jenkins' (gear icon), 'My Views' (people icon), and 'Credentials' (key icon). On the right side, there is a large 'Welcome to Jenkins!' message in bold. Below this message is a light blue box containing the text 'Please **create new jobs** to get started.' Below the welcome message, there are two expandable panels. The first panel is titled 'Build Queue' and shows 'No builds in the queue.' The second panel is titled 'Build Executor Status' and shows the status of the master and slave nodes. The master node has two idle executors, and the slave node (slave-07b8ee71) also has two idle executors.

Jenkins

Jenkins ▶

- New Item
- People
- Build History
- Manage Jenkins
- My Views
- Credentials

Welcome to Jenkins!

Please **create new jobs** to get started.

Build Queue

No builds in the queue.

Build Executor Status

master

- 1 Idle
- 2 Idle

slave-07b8ee71

- 1 Idle
- 2 Idle

Setting up a new job on Jenkins is easy.


Click the **New Item** link from the left menu, add a name for the job (all lower cased without spaces, like so: fullstack-website) and select Multibranch pipeline.

This will look at a repository and build any branch that has a Jenkinsfile in it.


Optionally you can create a pipeline job if you don't want to only build on changes to specific branches.

Enter an item name


» This field cannot be empty, please enter a valid name




Freestyle project
This is the central feature of Jenkins. Jenkins will build your project, combining any SCM with any build system, and this can be even used for something other than software build.




Pipeline
Orchestrates long-running activities that can span multiple build slaves. Suitable for building pipelines (formerly known as workflows) and/or organizing complex activities that do not easily fit in free-style job type.




External Job
This type of job allows you to record the execution of a process run outside Jenkins, even on a remote machine. This is designed so that you can use Jenkins as a dashboard of your existing automation system.




Multi-configuration project
Suitable for projects that need a large number of different configurations, such as testing on multiple environments, platform-specific builds, etc.



Folder
Creates a container that stores nested items in it. Useful for grouping things together. Unlike view, which is just a filter, a folder creates a separate namespace, so you can have multiple things of the same name as long as they are in different folders.




GitHub Organization
Scans a GitHub organization (or user account) for all repositories matching some defined markers.



Multibranch Pipeline
Creates a set of Pipeline projects according to detected branches in one SCM repository.

if you want to create a new item from other existing, you can use this option:



Copy from

OK

This step has to be done for any new project. There is however a solution to create Multibranch pipelines for an entire GitHub organisation, but as the name implies that only works with GitHub repositories.

Once you have the job set up along with the right triggers you can start committing your code as usual.

One of the first things you may want to do is create a Jenkinsfile so that you have a test and build process running.

In the next section we will look at what this file might contain and some of the basic building blocks we can use to automate our build and deployment process.

The Jenkinsfile

A Jenkinsfile is a [Groovy](#) script that describes what are the steps required in order to build, test and deploy the application.

A Jenkinsfile exists for each of our two repos in the /03-jenkinsfile folders. It also contains another directory called devops that will contain any extra files we will need to build and deploy the code to our server.

Our Jenkins files contain multiple pipeline stages:

- Checkout
- Build
- Test
- Release
- Deploy (found in the 04-deployment folder)

Apart from the Jenkinsfile we also need to set a series of credentials on the Jenkins job for your project in order to deploy the code on staging and production servers.

Handling deployment with Docker

After you create your Multibranch job and add your code to the repository you need to let the pipeline know:

- In which docker repository the built images should be pushed and how to connect to it?
- Where should the develop branch be deployed to?
- Where should the master branch be deployed to?

In your project create the following credentials:

- **BitBucket:**
 - Required to connect to the Bitbucket server
 - Type: Username/Password
 - Id: Anything (not used in the pipeline script)
- **Docker Registry Credentials**
 - Required in order to push the images to a Docker Registry from which they will later be deployed
 - Type: Username/Password
 - Id: around25_docker_registry
- **Staging Server for "develop" branch**
 - Staging Docker Server Host
 - Id: staging-docker-server-host
 - Type: Secret text
 - Staging Docker Server CA

- Id: staging-docker-server-ca
 - Type: Secret file
- Staging Docker Server Cert
 - Id: staging-docker-server-cert
 - Type: Secret file
- Staging Docker Server Key
 - Id: staging-docker-server-key
 - Type: Secret file
- **Production Server for "master" branch**
 - Production Docker Server Host
 - Id: production-docker-server-host
 - Type: Secret text
 - Production Docker Server CA
 - Id: production-docker-server-ca
 - Type: Secret file
 - Production Docker Server Cert
 - Id: production-docker-server-cert
 - Type: Secret file
 - Production Docker Server Key
 - Id: production-docker-server-key
 - Type: Secret file

You can find the required files in your Docker Machine configuration folder in
`~/.docker/machine/machines/fullstack-02/`.

After you added the credentials update your repositories with the Jenkinsfiles from the
 04-deployment folder and push the code in your develop branch.

Wrapping things up

Running each job should tell you if your build is successful or not. Once you have a successful build add some build triggers for your jobs and start pushing the code to the develop or master branches.

The job will pick up the changes to the repository and start running the pipeline to build, test and deploy your code to the desired environment.