



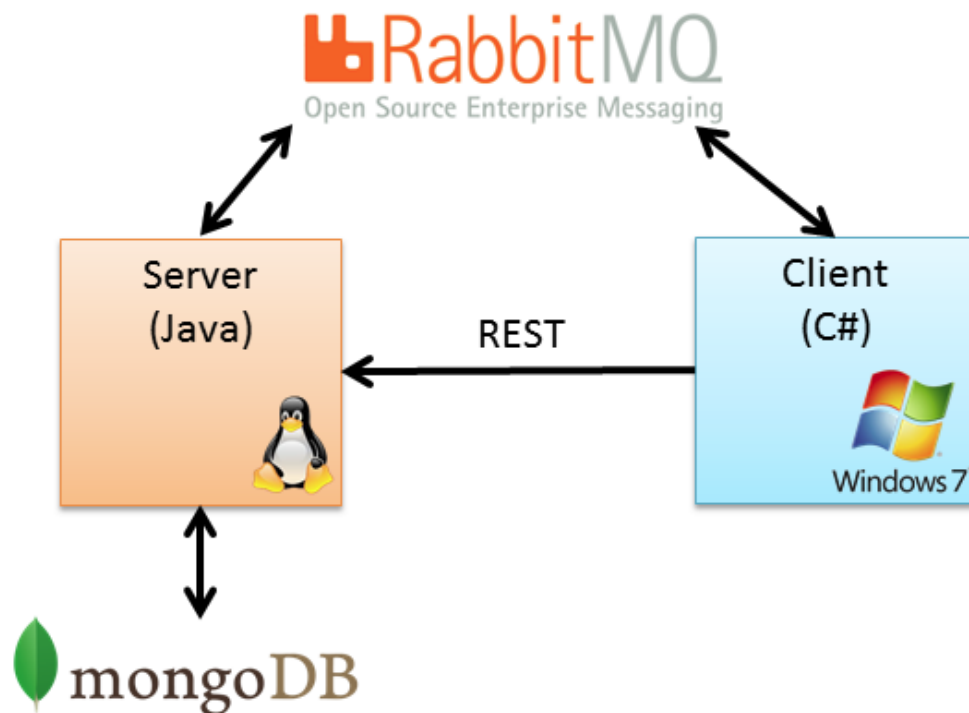
19 Jan 2014

# Continuous Integration and Deployment with Jenkins and Docker (Part I)

I had been wanting to update our software development process to include "Continuous Integration" (and deployment) but it always took a back seat to other, more pressing issues. Finally things calmed down enough in December, after a major release at the end of November, that I could finally work on getting this in place.

## Project Architecture

The project that I chose to use as the testbed for the CI server consists of a couple different software components and a few external dependencies. The first component is a web server written in Java that runs on Linux and connects to a Mongo database and a RabbitMQ server. The second component is a C# GUI that runs on Windows 7 and connects to the server component as well as the same RabbitMQ server the server component connects to.



## Goals

I started by defining the capabilities I wanted to have once it was all over:

- Automatic building and unit testing whenever new code is committed to specific branches
- If the server component changes, the client should be rebuilt/tested even if it hasn't changed. If the client changes the server does not need to be rebuilt
- Automatic distribution and running of successful builds to our test server. Since multiple branches may be monitored, it should be possible to have multiple deployed builds running at the same time on the same test machine
- Maintain database between builds of the same branch. For example, if the *develop* branch gets rebuilt, the data contained in the database for the currently running *develop* branch should be copied to the database for the newly build *develop* branch.
- Post successful builds to a location for download

Satisfying most of these goals seemed pretty straightforward except for the requirement of running multiple builds representing multiple different branches of the software on the same machine.

The server component provides a web and REST interface which is used by the client component and both communicate using RabbitMQ. Each build of the server will need its own running instance of MongoDB and RabbitMQ and will need to expose different ports

which the client can use to connect to the right server.

## Software Used

### Jenkins

Although there are alternatives to Jenkins that look pretty promising (Strider for one), I went with Jenkins due to its widespread use and number of available plugins.

### Docker

Docker was the perfect solution for maintaining separation between multiple builds running on the same machine. Each build runs within a Docker container which is configured to expose the ports required by the client. The challenge with this was getting the values of the exposed ports to the Jenkins process performing the builds of the client so that the client could be configured to connect to the right running instance of the server.

## Installation and Configuration

Due to the mixture of Java and C# we need to have two

CI servers, one runs Linux and the other runs Windows  
7. The Linux CI server is configured as the master and the Windows server is the slave.

## Docker

Docker installation is pretty straightforward, although you may have to upgrade your kernel if you're using Ubuntu 12.04. We followed the instructions at <http://docs.docker.io/en/latest/installation/ubuntulinux/> and didn't have any issues.

## Jenkins on Linux

Installing Jenkins on the Linux server was as easy as following the instructions at <https://wiki.jenkins-ci.org/display/JENKINS/Installing+Jenkins>. We're running Ubuntu so after a couple *apt-get install* commands we had Jenkins up and running.

## Jenkins on Windows

Since the Jenkins instance on Windows was going to be configured as a slave to the Linux instance, we followed the instructions at: <https://wiki.jenkins-ci.org/display/JENKINS/Step+by+step+guide+to+set+up+master+and+slave+machines>

for configuring Jenkin to run as a service on Windows.

I recommend running the "Jenkins Slave" service under a unique user account (i.e. an account called "jenkins"). We did this so that we could generate SSH keys for this account which Jenkins would use to pull code from our BitBucket repository. We had to assign a password to the Jenkins user account in order to get the Jenkins service to actually run under this account.

To get the service to run under this account, open "Services" and right click on the "Jenkins Slave" service and choose "Properties". Select the "Log On" tab and choose the "This account" option then choose the Jenkins user account.

## **GIT Access via SSH**

In order for Jenkins to pull down code from your repository, you need to provide it with an SSH key and configure your repository with this key. We use BitBucket to host our repository but the steps should be similar regardless of what you use.

We used PuttyGen to generate an SSH key for the Jenkins user and Pageant as the SSH authentication agent. Both can be downloaded from <http://www.chiark.greenend.org.uk/~sgtatham/putty>

[/download.html](#).

Run PuttyGen to generate a public/private key pair for the Jenkins user. Save both the generated public and private keys in a directory called ".ssh" in your users home directory (i.e. C:\Users\jenkins.ssh). Keep the default extension of "ppk" for your private key; that will be needed later.

Now you need to copy the contents of your public key to your repository. The public keys generated by PuttyGen do not match the standard OpenSSH format, so you will probably need to modify them. You can do this a few different ways but I just did it by hand. Open the public key in a text editor and remove the first two lines (BEGIN SSH and Comment) and the last line (END SSH) then add the string "ssh-rsa" to the front of your public key.

Next you need to install this public key into your repository. If you're using BitBucket, follow the instructions in step 6 on this page:

<https://confluence.atlassian.com/pages/viewpage.action?pageId=270827678>. Other hosting services should be pretty similar.

Finally, you need to use Pageant to load your private key

and associate the "ppk" extension with the Pageant application. To do this, right click on your private key and select "Open With". Choose "Pageant" (you may need to click the "Browse" button and navigate to where it's installed) and make sure to select "Always use the selected program to open this kind of file" before clicking the "Ok" button. Now Pageant should be running with your private key. To ensure that Pageant starts automatically for the Jenkins user if the OS is restarted, add a shortcut to the Jenkins users private key to the "Startup" folder in the start menu.

## Part 2 coming soon...

This is getting a little long so I'll break it into parts. Part 2 will cover configuring Jenkins to do what you want it to do!

---

### Adam Stull

CTO and Software Engineer at a small software startup creating software for RPAs (drones).  
@inpursuit on twitter

### Share this post






Ghostery hat Kommentare von Disqus blockiert.



comments powered by Disqus



All content copyright InPursuit © 2014 • All rights reserved.

Proudly published with  **Ghost**