You are in: DevOps Technical Workshop Wiki > V2 Lab I The Environment Provisioning Menace > V2 Lab 1 - Steps 2C & 2D

V2 Lab 1 - Steps 2C & 2D

Like | Updated 13/06/2018 by Roy Mitchley | Tags: *None*

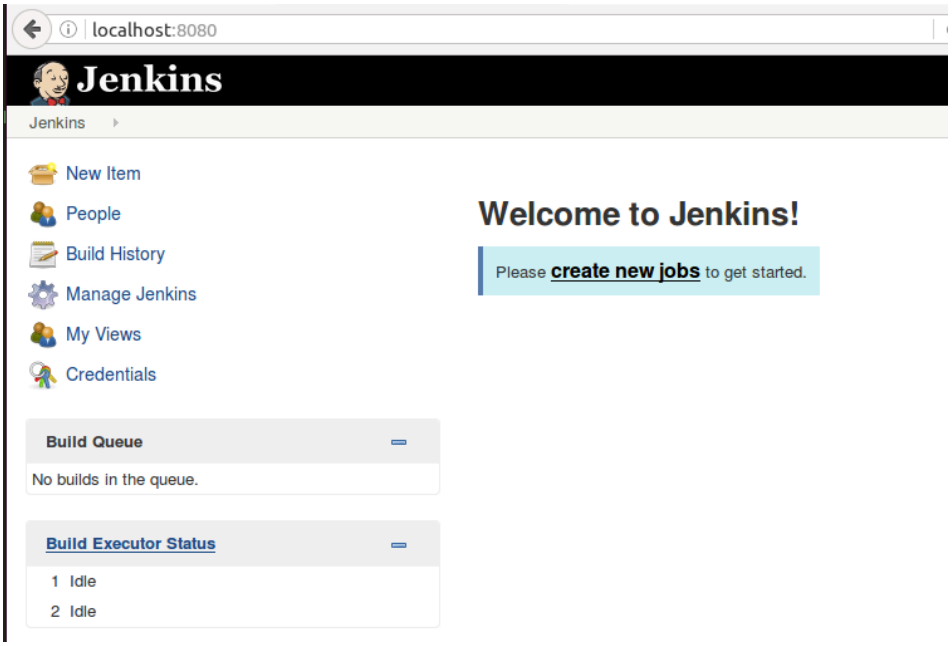
C) Installing Jenkins Plugins and creating a Hello-World job.

The objective of this lab is to install some plugins, see the different ways they can be installed and create your first Jenkins Job

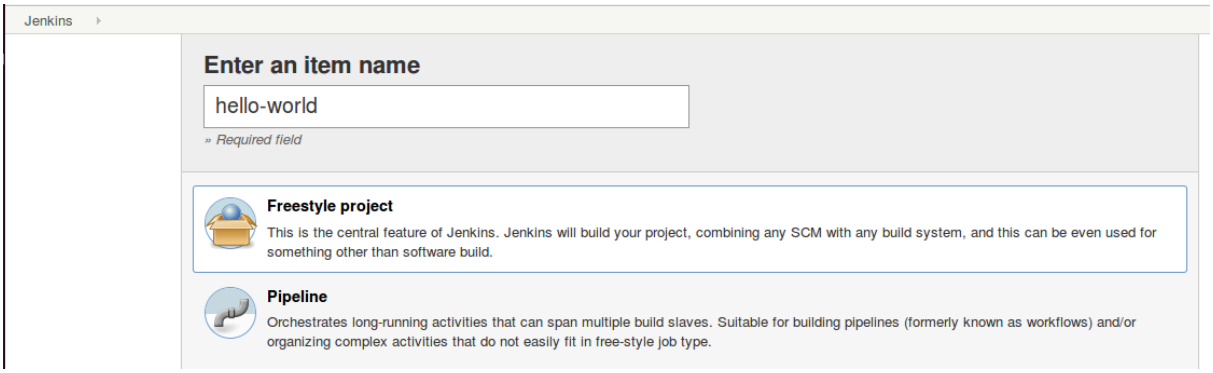
Atom	Anything in this box needs to be edited in Atom Text

Terminal	This box contains lists of commands that should executed in order on the Terminal

- You should see the following screen before continuing



- Create a *New Item* and enter a name eg `hello-world`. Then *Freestyle Project* and select `OK`



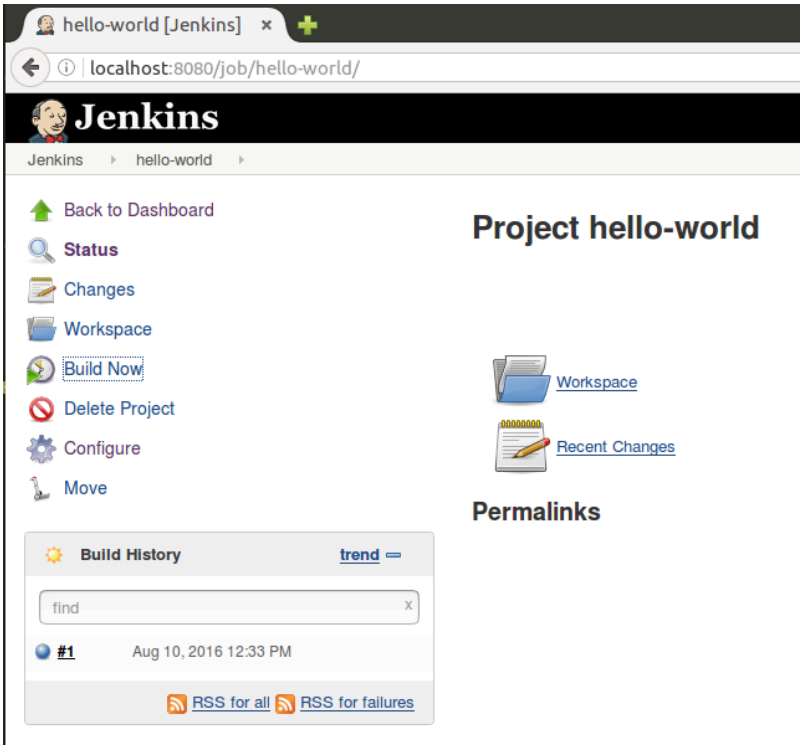
- Hit the *Build* tab then *Add Build Step* > *Exectute shell*
- Type the following into the text box.

```
echo "Hello World, I am jenkins from ${JOB_NAME}.${BUILD_NUMBER}" >>  hello.txt
```

```
mv hello.txt /home/devops/Desktop
```

Jenkins exposes various Environment Variables (the items denoted with ${}), which are made available to each job as it is executing. They are available and can be set at http://jenkins.server:8080/env-vars.html/ Environment variables allow common things to be set centrally.

- Hit *Apply* then *Save*
- Back on the Jenkins project page, hit *Build Now*. You should see something like this.

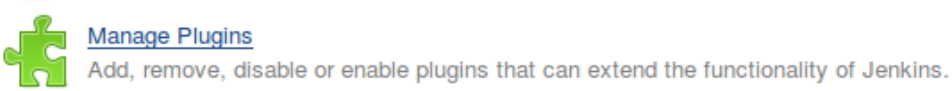


- Jenkins by default uses Blue colour to mean success for a build; Our next step is to change this to green using a plugin

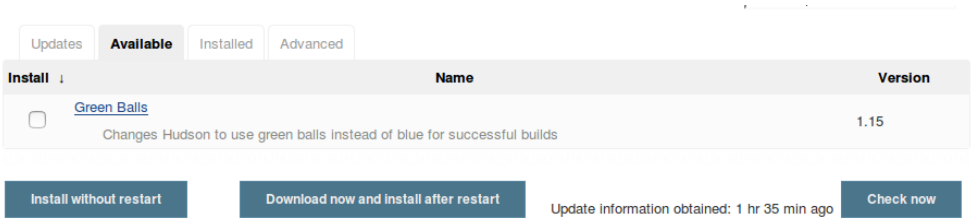
There are three ways to install a plug-in in Jenkins, some of which will be explored:

- Download and install the plug-in using the Internet
- Upload a compiled plugin (.jpi file) manually
- Copy existing .jpi files into the $JENKINS_HOME/plugins directory

- Go to Jenkins homepage (http://jenkins.server:8080/) then *Manage Jenkins* > *Manage Plugins*



- Hit the *Available* tab and navigate to the `Green Balls` plugin or use the Filter. Hit the the checkbox and *Install without Restart*



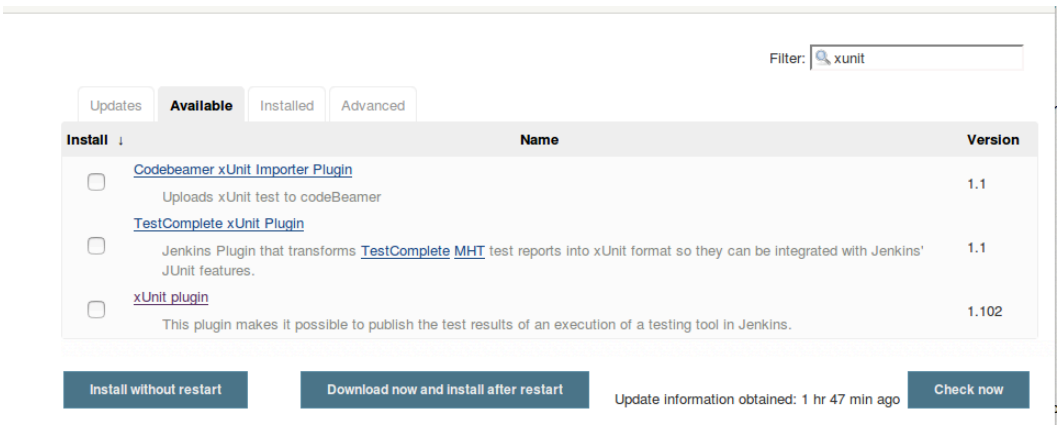
NOTE - It may take some time when loading this page, as it can appear blank. Try refreshing the page if it fails to load. Sometimes, Jenkins will fail to load all the plug-ins and the page will remain blank. In this case, try checking your internet connection with the VM. If there is no connection, you will not see the plug-in list. If there is Internet connectivity but still you are unable to view the list, restart Jenkins by going to the following URL:

http://jenkins.server:8080/restart and click **OK** on restart.

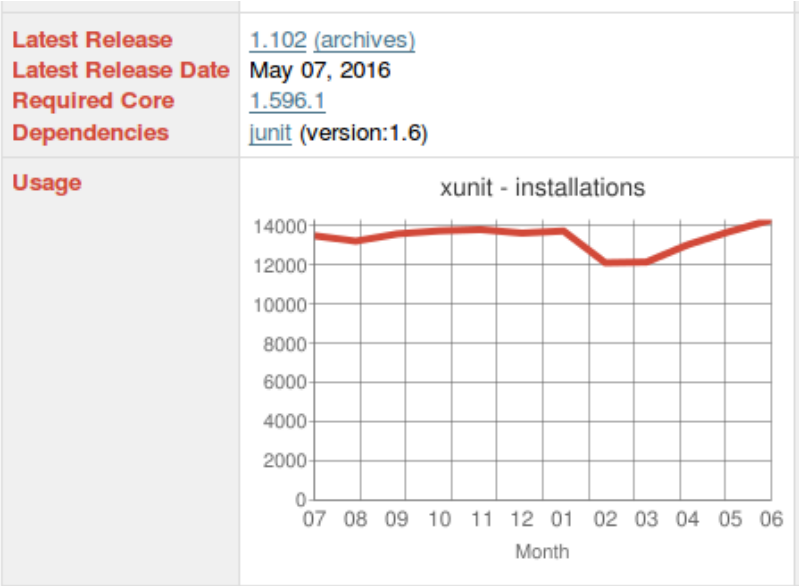
- Run another build of the *hello-world* job and you'll discover the Green Balls plugin has not taken effect. Restart Jenkins for the changes to take affect

http://jenkins.server:8080/restart and click **OK** on restart.

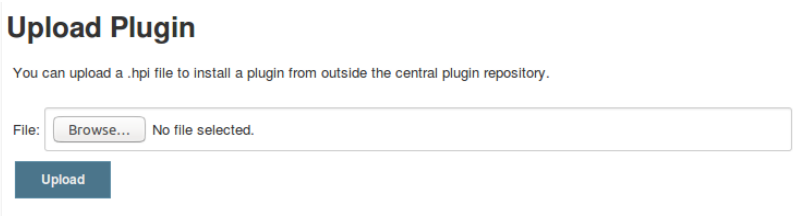
- To manually install a plugin; go to back to the *Available* on the *Manage Plugins* page and use the filter to search for `Xunit` this will be used later in labs



- Click the link and go to the wiki page for the plugin. When deciding on what plug-ins to use, it's best to check how frequent the releases are or the plug-ins popularity, as it will indicate the level of support and commitment to the plug-in. Select the latest release link (your version may be different) to download the `xunit.hpi` file.



- Back on Jenkins Manage Plugins page, hit advanced then Upload Plugin and navigate to the plugin you just downloaded.



- Hit upload then you'll see the updateCenter page. NOTE - this method of installing plugins will not bring in all the dependencies for the plugin but can be useful if Jenkins has no Internet access and plugins are needed, they can be stored in a repository and installed manually.

–

(D) Exploring the Sample Application

The objective of this lab is to clone and explore the todo list application

- Open the terminal and run the following in any directory to bring in the sample application:

```
cd $HOME
git clone https://bitbucket.org/ibmdevopscourse/todolist.git
```

- Now point the remote *origin* a local version so that you can make changes to it. Run the following:

```
cd todolist
git remote remove origin
git remote add origin ssh://git@git.server/home/git/todolist.git
git push -u origin --all
```

- We will be making all of our changes to the develop branch. So ensure that you are on the develop branch by executing the following from the todolist project directory:

```
git status
```

- Ensure you are on the develop branch by executing the following form the todolist project directory:

```
git checkout develop
```

- The application uses Nodejs as the runtime. To install this in our VM, puppet will be used. Run the following from any directory to open the lab-material project in the Atom editor:

```
atom /share/lab-material
```

```
From the lab-material project, open the puppet/build-vm.pp manifest
```

- Import the `node-js.pp` manifest file into the `build-vm.pp` by adding the following include statement under the `import 'install-jenkins.pp'` statement:

```
# Node and node dependecies install...
import 'node-js.pp'
```

The top of the file manifest should now look like this:

Take a closer look at the `node-js.pp` manifest file:

```
From the lab-material project, open the puppet/node-js.pp manifest
```

This manifest installs nodejs v4.4.7 for use in the VM. It uses the nodejs puppet module which allows us use an additional provider (npm) to install other packages. The npm provider is used in the puppet manifest with the puppet package module to install: `grunt-cli, http-server and phantomjs packages` node modules.

- Install `bower` (a JavaScript client package manager) by adding a new package entry into the node-js manifest file:

```
package { 'bower':
  ensure   => '1.7.9',
  provider => 'npm',
  require => Package['nodejs']
}
```

so that it looks as follows:

```
class { '::nodejs':
  repo_url_suffix => 'node_4.x',
} ->

package { 'bower':
  ensure => '1.7.9',
  provider => 'npm',
  require  => Package['nodejs']
}

package { 'grunt-cli':
  ensure   => '1.2.0',
  provider => 'npm',
  require  => Package['nodejs']
}

package { 'http-server':
  ensure   => '0.9.0',
  provider => 'npm',
  require  => Package['nodejs']
}

package { 'phantomjs':
  ensure   =>  '2.1.1',
  provider => 'npm',
  require  => Package['nodejs']
} ->

file { ['/etc/profile.d/phantomjs.sh','/var/lib/jenkins/.profile']:
  ensure  =>  present,
  content =>  inline_template('export PHANTOMJS_BIN="/usr/bin/phantomjs"'),
}
```

- Then save all edited files and run the puppet manifest:

```
sudo puppet apply build-vm.pp
```

After this has run successfully, commit your changes by running the following from the `/share/lab-material` directory:

```
git status          #look through your changeset to ensure it contains
everything you                      expect
git add .           #stage your changes by adding it to the index
git commit -m 'install node and dependencies'
                    #commit the staged changes with a meaningful message
git pull            #get the latest changes from the remote
git push            #push your changes to the remote (origin) set up in step 1
```

node should now be installed, along with the specified node_modules above.

- Now install the dependencies required by the application by running the following command from the `~/todolist` directory:

```
npm install
bower install
```

Comment: When running npm install the following kerberos errors might be displayed (note that the errors have been highlighted in this composite screenshot).

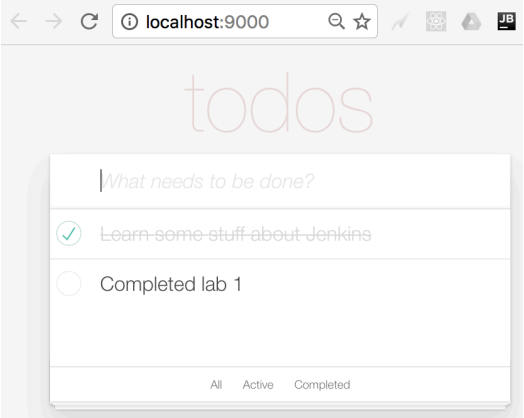
These can be ignored as long as the npm command completes successfully otherwise. Mongodb will be installed during a later step.

```
devops@devops-v2: ~/todolist
File  Edit  View  Search  Terminal  Help
In file included from ../lib/kerberos.cc:1:0:
../lib/kerberos.h:5:27: fatal error: gssapi/gssapi.h: No such file or directory
compilation terminated.
kerberos.target.mk:100: recipe for target 'Release/obj.target/kerberos/lib/kerberos.o'
failed
make: *** [Release/obj.target/kerberos/lib/kerberos.o] Error 1
make: Leaving directory '/home/devops/todolist/node_modules/connect-mongo/node_modules/
mongodb/node_modules/kerberos/build'
gyp ERR! build error
gyp ERR! stack Error: `make` failed with exit code: 2
gyp ERR! stack     at ChildProcess.onExit (/usr/lib/node_modules/npm/node_modules/node-
gyp/lib/build.js:276:23)
gyp ERR! stack     at emitTwo (events.js:87:13)
gyp ERR! stack     at ChildProcess.emit (events.js:172:7)
gyp ERR! stack     at Process.ChildProcess._handle.onexit (internal/child_process.js:21
1:12)
gyp ERR! System Linux 4.4.0-57-generic
gyp ERR! command "/usr/bin/node" "/usr/lib/node_modules/npm/node_modules/node-gyp/bin/n
ode-gyp.js" "rebuild"
gyp ERR! cwd /home/devops/todolist/node_modules/connect-mongo/node_modules/mongodb/node
_modules/kerberos
gyp ERR! node -v v4.8.6
gyp ERR! node-gyp -v v3.4.0
gyp ERR! not ok

> kerberos@0.0.23 install /home/devops/todolist/node_modules/mongoose/node_modules/mong
odb/node_modules/mongodb-core/node_modules/kerberos
> (node-gyp rebuild) || (exit 0)

make: Entering directory '/home/devops/todolist/node_modules/mongoose/node_modules/mong
odb/node_modules/mongodb-core/node_modules/kerberos/build'
  CXX(target) Release/obj.target/kerberos/lib/kerberos.o
In file included from ../lib/kerberos.cc:1:0:
../lib/kerberos.h:5:27: fatal error: gssapi/gssapi.h: No such file or directory
compilation terminated.
kerberos.target.mk:100: recipe for target 'Release/obj.target/kerberos/lib/kerberos.o'
failed
```

- For local development, you can serve the application by running the following command from the `~/todolist` directory:

```
grunt serve
```

Welcome to the todo list application! Grunt serve should have automatically launched a browser for you containing the todo list application:

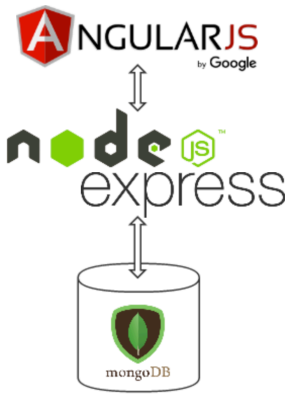


If you ever feel lost, provided your application is still being served, you can navigate to http://localhost:9000/ and return to the application.

The application currently uses a stubbed backend. Try adding a new todo and refresh the page. This todo list app has some memory loss issues at the moment, but do not worry we will fix this later in the labs.

`grunt serve` does some pretty magical stuff. If you make a change to the application code, it will recompile and livereload the change into the browser, which gives you, the developer, some pretty hasty feedback! The application was generated with *yeoman* using the *angular-fullstack* generator. You can learn more about yeoman at http://yeoman.io/.

This is a simple three tiered application, using AngularJS for the client, NodeJS and Express for the application server and MongoDB, a nosql db, persistence layer, or MEAN for short:

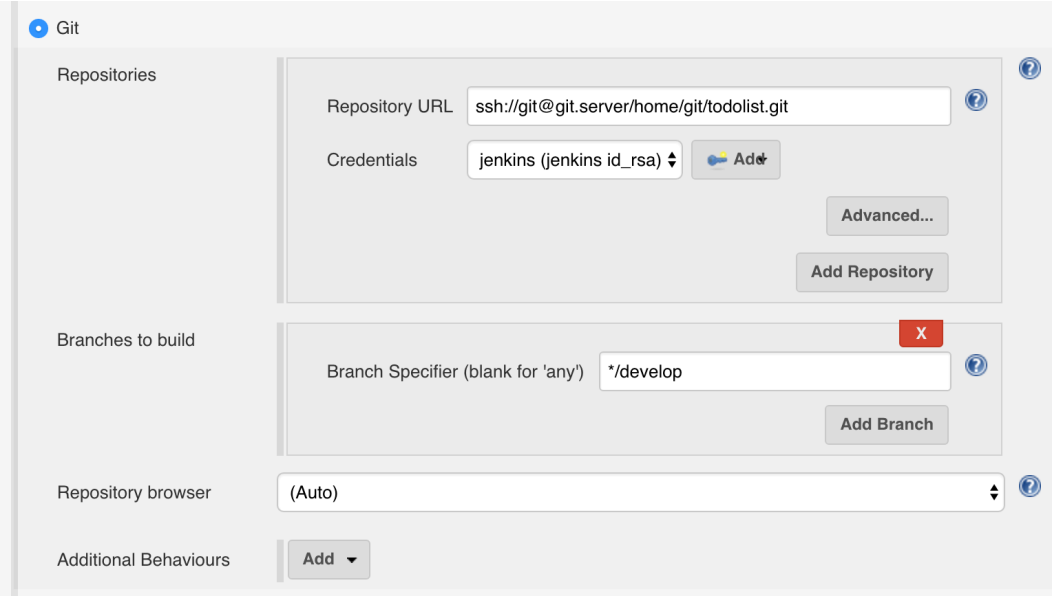


(E) Build and Deploy the todo list app with Jenkins Jobs

The objective of this lab is to create your first build and deploy pipeline for the todolist application using Jenkins and Docker

- Create the Jenkins job to build the todolist application. Navigate to Jenkins, http://jenkins.server:8080/, OR use the Desktop shortcut. Create a new job by clicking on *New Item.* Name the job by entering in `todolist-build` in the *Enter an item name* enter field. Create the job by clicking *Freesyle project* and then *OK.*
- Configure Jenkins to clone the todolist app. In the *Source Code Management* tab, select the *Git* radio button.
- Jenkins needs to have passwordless authentication to clone the git repository. Hit the *Add* button and Select *Jenkins* to add new credentials. Select *SSH Username with private key* as the *Kind*. Under *Private Key* hit the *From the Jenkins master ~/.ssh* radio button. This will use the `/var/lib/jenkins/.ssh/id_rsa`, the key we just created with puppet. Now give it a meaningful *Description*:

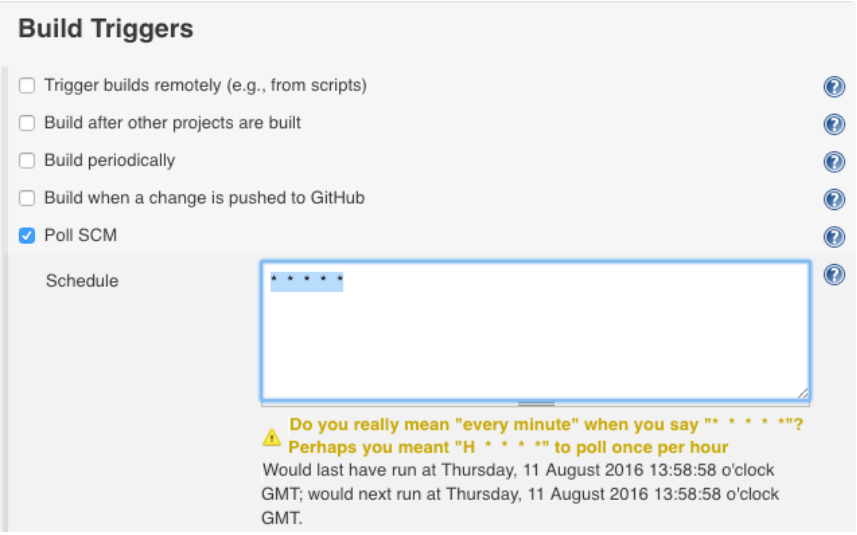
- Set the *Repository URL* to be `ssh://git@git.server/home/git/todolist.git` and the *Branch Specifier* to be `*/develop`. Set the *Credentials* to the newly created credential, which will appear as `jenkins (jenkins id_rsa)` if you used the description above. Your configuration should now look like this:



- To have Jenkins build on a change to our source code for the app, we will set up a *Build Trigger.* In the *Build Triggers* section, click *Poll SCM* and enter in the following cron-style schedule:

```
*   *   *   *   *
```

This is what it should look like in Jenkins:



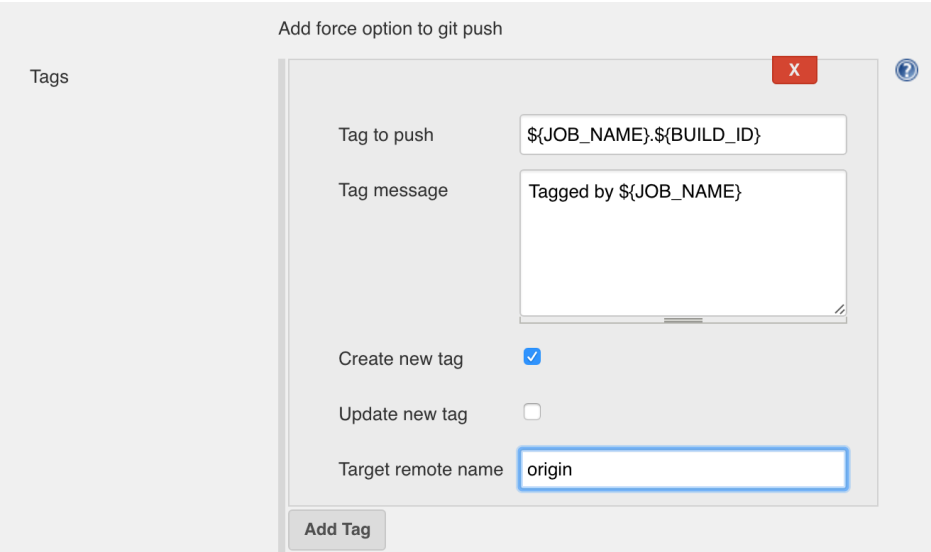
NOTE: the warning is as we would expect. We would usually have the remote git server prod Jenkins when it receives a commit, and use a plugin in Jenkins to interpret that prod and trigger the correct job, however, it is not feasible to fit all this in the VM, so we use a simplistic polling setup.

- Now add the meat of the Jenkins job, which is to build (compile) our client code. Navigate to the *Build* tab and *Add Build Step* > *Execute shell.* In the box add the following line:

```
npm install
bower install
grunt build
```

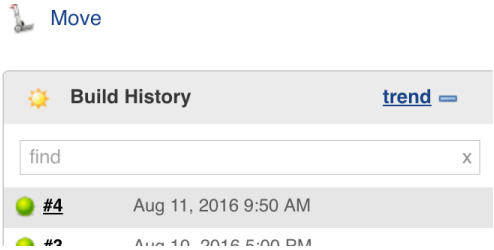
NOTE: Strictly speaking JavaScript does not compile, however there are various compile-like steps that need to happen for client code such as minification, uglification and less compilation to css.

- Now add a *post build action* to add a tag to the git repository for the revision that has just been built. This is for traceability and allows any successful build to be rebuilt later. This is very useful if you need to debug an issue with the code for a particular version of the application. Navigate to the *Post-build actions* click *add post-build action* and then select *Git publisher* from the drop down. Click the *Tags* button to add a tag. In the Tag to push field enter `${JOB_NAME}.${BUILD_ID}` *(*this is a naming convention for the *build tag),* click the checkbox create new tag and enter `origin` as *Target remote name*:



- Finally, save your configuration changes to todolist-build and click *Build Now*, verifying that the build completes

successfully with a new green *Build Result* in the *Build History*:



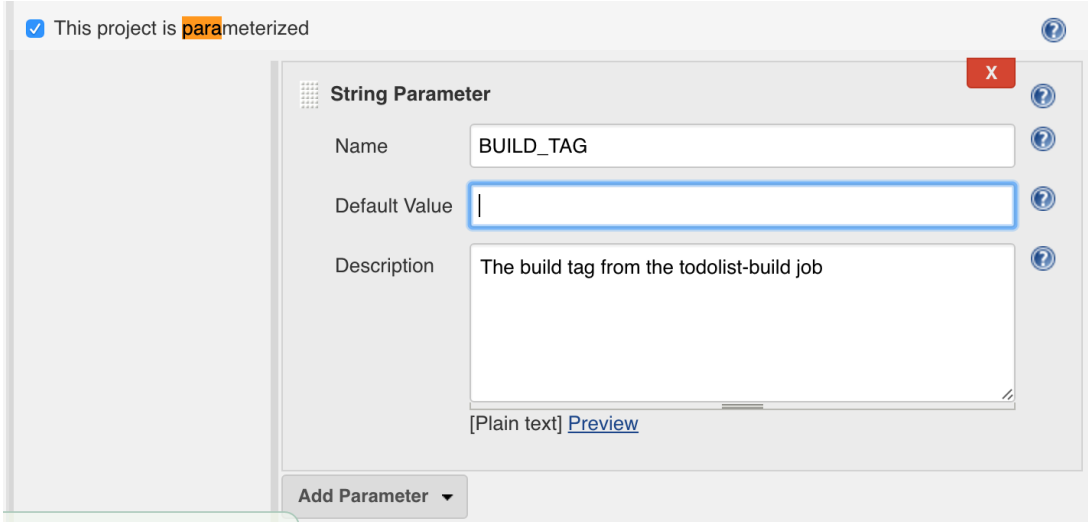
NOTE: The first run of this build will take a while, however future runs should be a little quicker.

- Next create the *todolist-deploy-ci* job. Navigate to Jenkins, http://localhost:8080/, and create a new job by clicking on *New Item.* Name the job by entering in `todolist-deploy-ci` in the *Enter an item name* enter field. Create the job by clicking *Freesyle project* and then *OK.*
- Set the *Repository URL* to be `ssh://git@git.server/home/git/todolist.git` and the *Branch Specifier* to be `${BUILD_TAG}`. Set the *Credentials* to the newly created credential, which will appear as `jenkins (jenkins id_rsa)` if you used the description above. Your configuration should now look like this:

- Now we need to add the configuration for Jenkins to execute the Grunt task to build the docker image that we will use to in this deploy job and all future deploy/promote jobs. The job will also then deploy the docker image to ci. Navigate to the *Build* section then *Add build step* and select *Execute shell.* Enter*:*

```
npm install
bower install
grunt build
grunt build-image:${BUILD_TAG}
grunt deploy:ci:${BUILD_TAG}
```

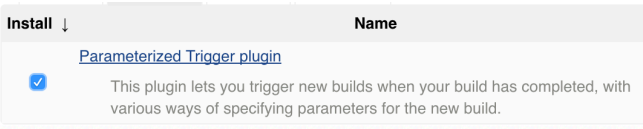
- As you can see, the deploy requires the *${BUILD_TAG}* variable. Luckily it is easy to add a parameter to a job. Navigate to the *General* tab and click the checkbox *The project is parameterized* > *Add Parameter* > *String Parameter* and enter `BUILD_TAG` into the *Name* field and give it a description of *The build tag from the todolist-build job*:



- Now save the job by clicking *Save*.

The next thing to do is to create a simple pipeline by wiring the todolist-build and todolist-deploy-ci jobs together.

- The upstream job masters the BUILD_TAG, it is an identifier unique to each build and is made up of the jobs name and the build number. We need to pass this parameter into our *todolist-deploy-ci* job. We are going to need the help of the *parameterized trigger plugin*. Just like when we installed the *Green Balls* plugin, Go to the Jenkins homepage (http://localhost:8080/) then *Manage Jenkins* > *Manage Plugins* > *Available.* Now filter by typing `parameterized trigger` and select the plugin:



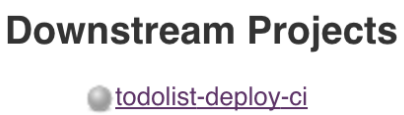
NOTE - It may take some time when loading this page, as it can appear blank. Try refreshing the page if it fails to load. Sometimes, Jenkins will fail to load all the plug-ins and the page will remain blank. In this case, try checking your internet connection with the VM. If there is no connection, you will not see the plug-in list. If there is Internet connectivity but still you are unable to view the list, restart Jenkins by going to the following URL:

http://localhost:8080/jenkins/restart and click **OK** on restart.

- After Jenkins has restarted, go to Jenkins homepage and then click the *todolist-build* job to get to the *todolist-build* page. Now configure the job to trigger a downstream job, *todolist-deploy-ci*, with the *BUILD_TAG* parameter. Click on *Configure*, navigate to the *Post-build Actions* tab. Click *Add post-build action* and select *Trigger parameterized build on other projects* and enter *todolist-deploy-ci* as the *Projects to build*. For *Trigger when build is* option select *Stable or unstable but not failed*. Click Add Parameters and select *Predefined parameters* from the drop-down. Now enter `BUILD_TAG=${JOB_NAME}.${BUILD_ID}`as shown below:

- Now save and build the job by clicking *Save*

Now we have created a pipeline using the *Parameterized trigger plugin* you should see that a new heading has appeared in the job overview, named *Downstream Projects:*



- Finally, trigger the pipeline by clicking *Build Now* on the *todolist-build* job page. Once the *todolist-build* job has run successfully, confirm that the *todolist-deploy-ci* job has been triggered automatically. Click on the *todolist-deploy-ci* job and you should see the build running.

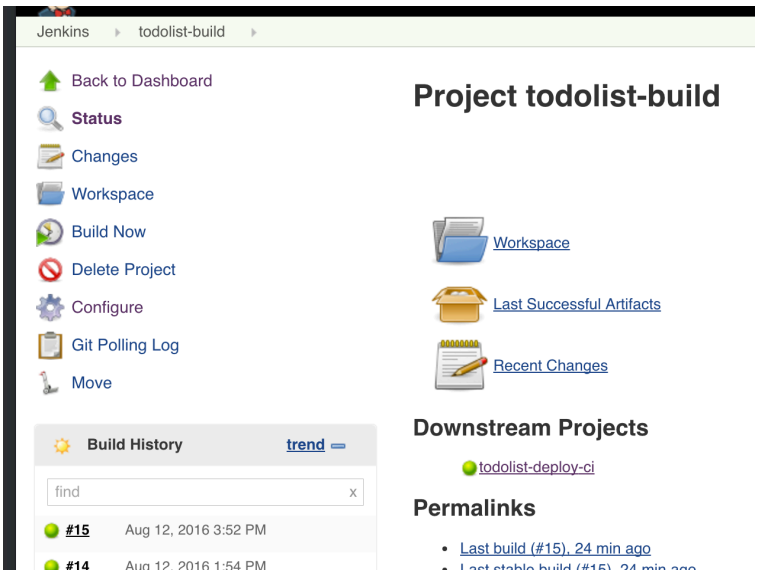
The *todolist-deploy-ci* job will build a docker image and tag it with the build tag created by the upstream *todolist-build* job. The *ci* configuration option on deploy will configure your app to run on http://localhost:9001/. The CI environment also has a stubbed backend, and we will extend the pipeline to use a backend and persistence layer later in the labs.

- This may take some time the first run. You can click on the running build and then *Console Output*

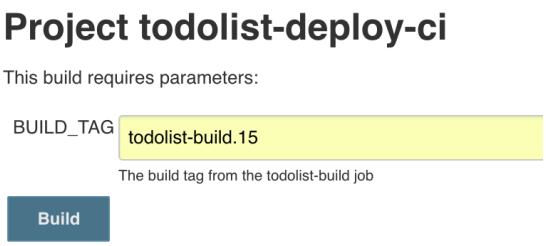
You may see that the logs are stuck on this step here:

However what the slave is not telling you, is that the build is copying all of the upstream artifacts from *todolist-build*, which takes 5 minutes or so.

- If the *todolist-deploy-ci* build **fails**, you can rerun the build and pass in the BUILD_TAG that would have been generated in the *todolist-build* job. This will be a combination of the jobs name and the latest build number, so in the example below:

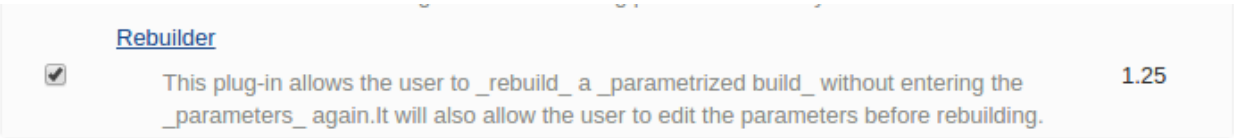


You would pass in `todolist-build.15` as follows:



- Navigate to http://localhost:9001 to see your application running in CI mode. You will notice that if you refresh your browser after making changes, your changes are not persisted in this environment.

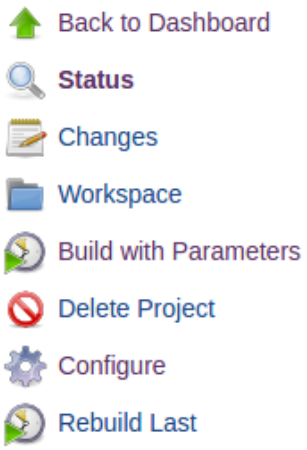
- To make it easier to rebuild the deploy stages when they fail, we can make use of the Jenkins rebuild plugin, which will allow you to trigger a rebuild of a past build. We are going to need the help of the *Rebuilder* plugin. Just like when we installed the *Green Balls* plugin, Go to the Jenkins homepage (http://localhost:8080/) then *Manage Jenkins* > *Manage Plugins* > *Available.* Now filter by typing `rebuild` and select the plugin:



NOTE - It may take some time when loading this page, as it can appear blank. Try refreshing the page if it fails to load. Sometimes, Jenkins will fail to load all the plug-ins and the page will remain blank. In this case, try checking your internet connection with the VM. If there is no connection, you will not see the plug-in list. If there is Internet connectivity but still you are unable to view the list, restart Jenkins by going to the following URL:

http://localhost:8080/jenkins/restart and click **OK** on restart.

Now, if a step fails, you can rebuild the last run by navigating to the failing job, and clicking the following *Rebuild Last* icon:



EXTENSION TASKS

- Use puppet to manage the firewall in the VM and open up Jenkins for access outside of the VM, use the following to get you started: https://forge.puppet.com/puppetlabs/firewall. Use the VirtualBox port forwarding to map Jenkins to your host computer.

Comments

There are no comments.