V2 Lab 2 - Steps 2A

Like | Updated 3 July 2017 by Katamneni, Krishna Sravya | Tags: *None*

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`

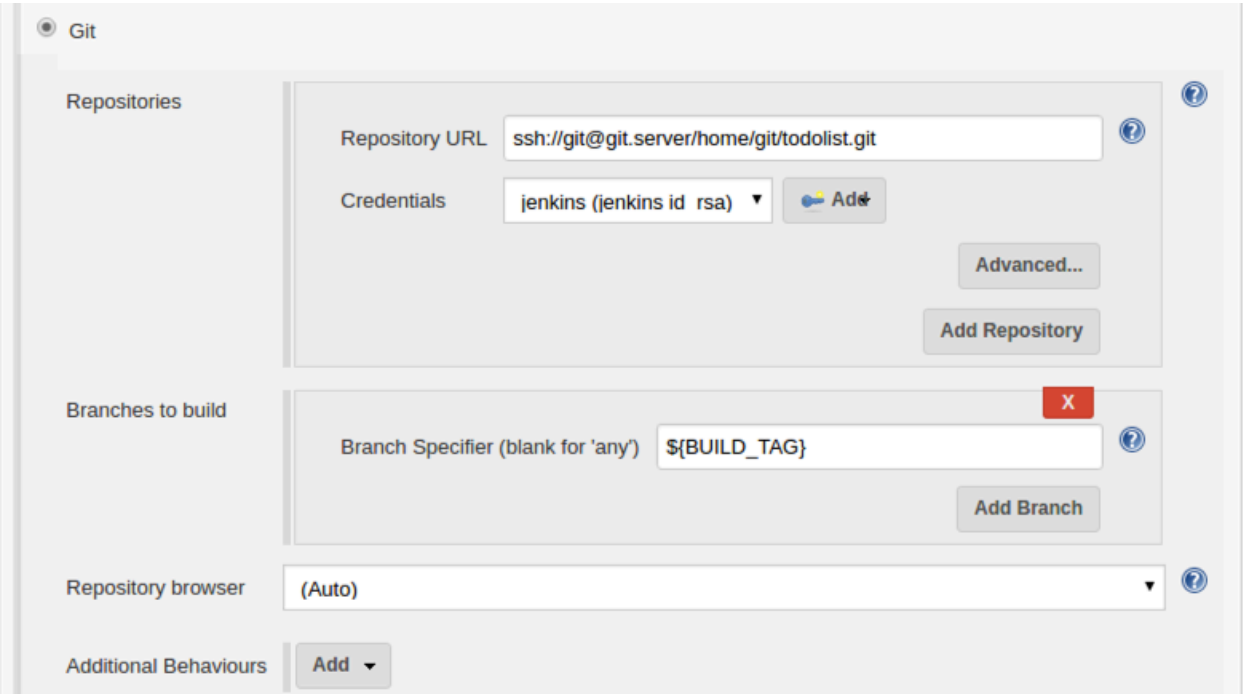
(A) Extend pipeline to production

The objective of this lab is to extend your pipeline to production by promoting your built package to downstream environments

As you promote your code further down your pipeline, the environments that you deploy to should become more and more production-like, and the pipeline should provide increasing levels of assurance. We will firstly configure the pipeline to deploy to the SI environment, which is configured to use a real backend, which provides data persistence functionality.

- Create the *todolist-deploy-si* job. Navigate to Jenkins, http://jenkins.server:8080/, and create a new job by clicking on *New Item.* Name the job by entering in *todolist-deploy-si* in the *Enter an item name* enter field. Create the job by clicking *Freestyle project* and then *OK.*

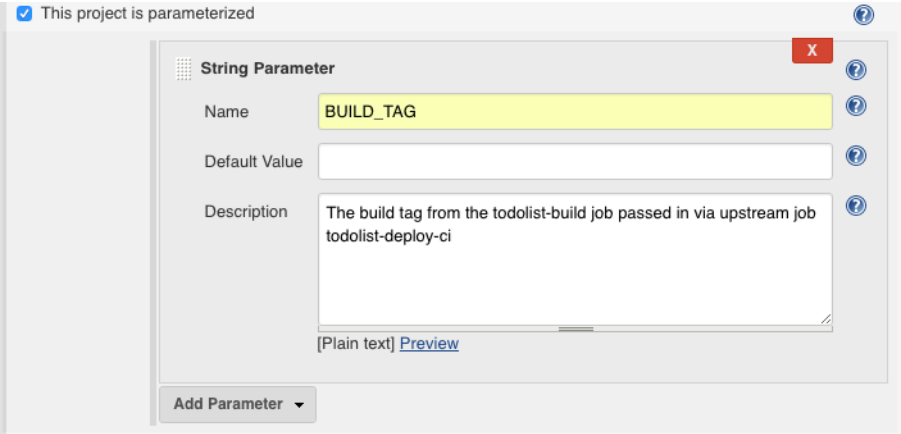
- Set the *Repository URL* to be `ssh://git@git.server/home/git/todolist.git` and the *Branch Specifier* to be `${BUILD_TAG}`. Set the *Credentials* to the newly created credential, which will appear as `jenkins (jenkins id rsa)` if you used the description above. Your configuration should now look like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



- Now we need to add the configuration for Jenkins to deploy the docker image created in the upstream *todolist-deploy-ci* job. Navigate to the *Build* section then *Add build step* and select *Execute shell.* Enter:

```
npm install
grunt deploy:si:${BUILD_TAG}
```

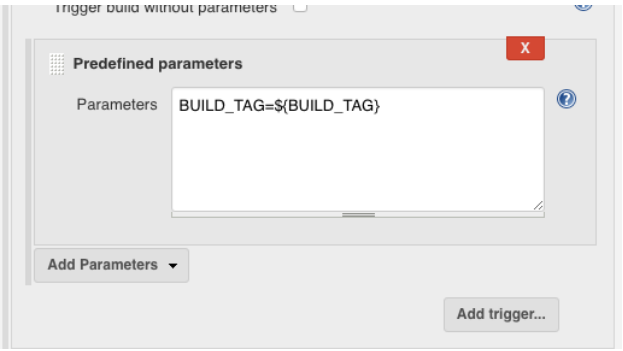
- Seeing as we want to deploy the same docker image that was created with the tag in the upstream todolist-*deploy-ci* job, the todolist-*deploy-si* job will require the same *${BUILD_TAG}* passed in from that job. Navigate to the *General* tab and click the checkbox *The project is parameterized* > *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 passed in via upstream job todolist-deploy-ci`*:*



- Now save the job by clicking *Save*.

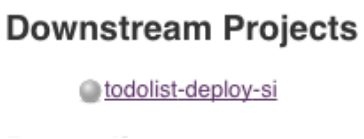
The next thing to do is to extend the pipeline by wiring the *todolist-deploy-ci* into the todolist-deploy-si just as we did before with the *todolist-build* and *todolist-deploy-ci* jobs.

- Go to Jenkins homepage and then click the *todolist-deploy-ci* job to get to the *todolist-deploy-ci* page. Now configure the job to trigger a downstream job, *todolist-deploy-si*, 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-si* as the *Projects to build*. Click Add Parameters and select *Predefined parameters* from the drop down. Now enter `BUILD_TAG=${BUILD_TAG}`:



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

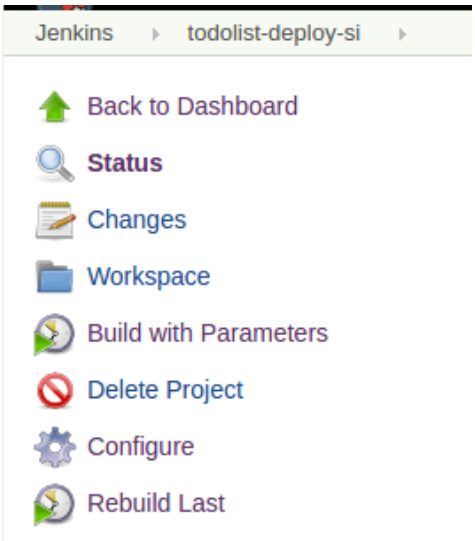
Now we have extended the pipeline using the *Parameterized trigger plugin*, you should see that a new heading has appeared in the job overview, named *Downstream Projects:*



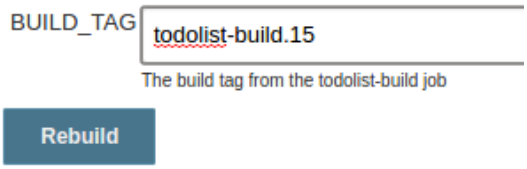
- Finally, trigger the pipeline. Navigate back to Jenkins homepage and then click on the *todolist-build* job page. Click *Build Now***.** Once the *todolist-build* job has run successfully and the *todolist-deploy-ci* job has run successfully, ensure that the *todolist-deploy-si* has also triggered automatically.

The *todolist-deploy-si* job will run the docker container that was built and tagged by the upstream *todolist-deploy-ci* job.

NOTE: If the todolist-deploy-si build ***fails****, you can rerun the build by selecting the Rebuild Last icon: This will then pre-fill the build tag as follows:*



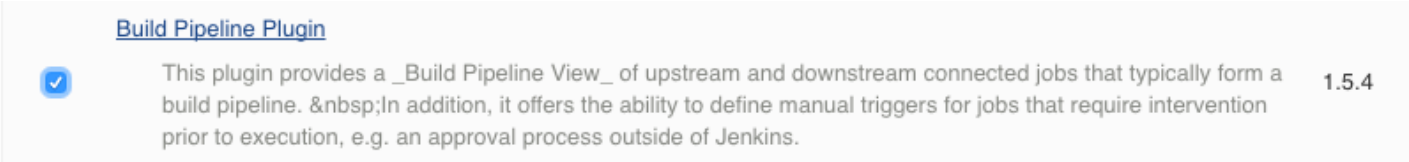
The next screen will show you the parameters that were past into the last build and provide you an option to run a Rebuild as follows:



- By passing in *si* to the deploy task, we have configured this environment to use a real backend. This means that your application will now have real server endpoints that talk to a real mongodb database. Now that you have your persistence layer, navigate to the si environment at http://localhost:9002/, add some todos and refresh your page. Voila! we have persistence!

- As this is a test environment, we seed it with test data each time we deploy. Rerun the *todolist-deploy-si* job with the same BUILD_TAG that was used to trigger the last successful deploy and, once successful, navigate to http://localhost:9002/ and you will notice that all those useful todos you created are gone! The next step is to extend the pipeline to production, and configure that environment to not seed the database.

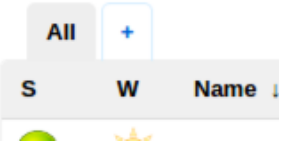
- Now that out jobs are feeling more like a pipeline, lets visualise it as one. To do this we will install the *Build Pipeline* plugin in Jenkins. Just like when we installed the *Green Balls* plugin, Go to the Jenkins homepage (http://localhost:8080/) then *Manage Jenkins* > *Manage Plugins* > *Available.* Now filter by typing `Build Pipeline` and select the plugin:



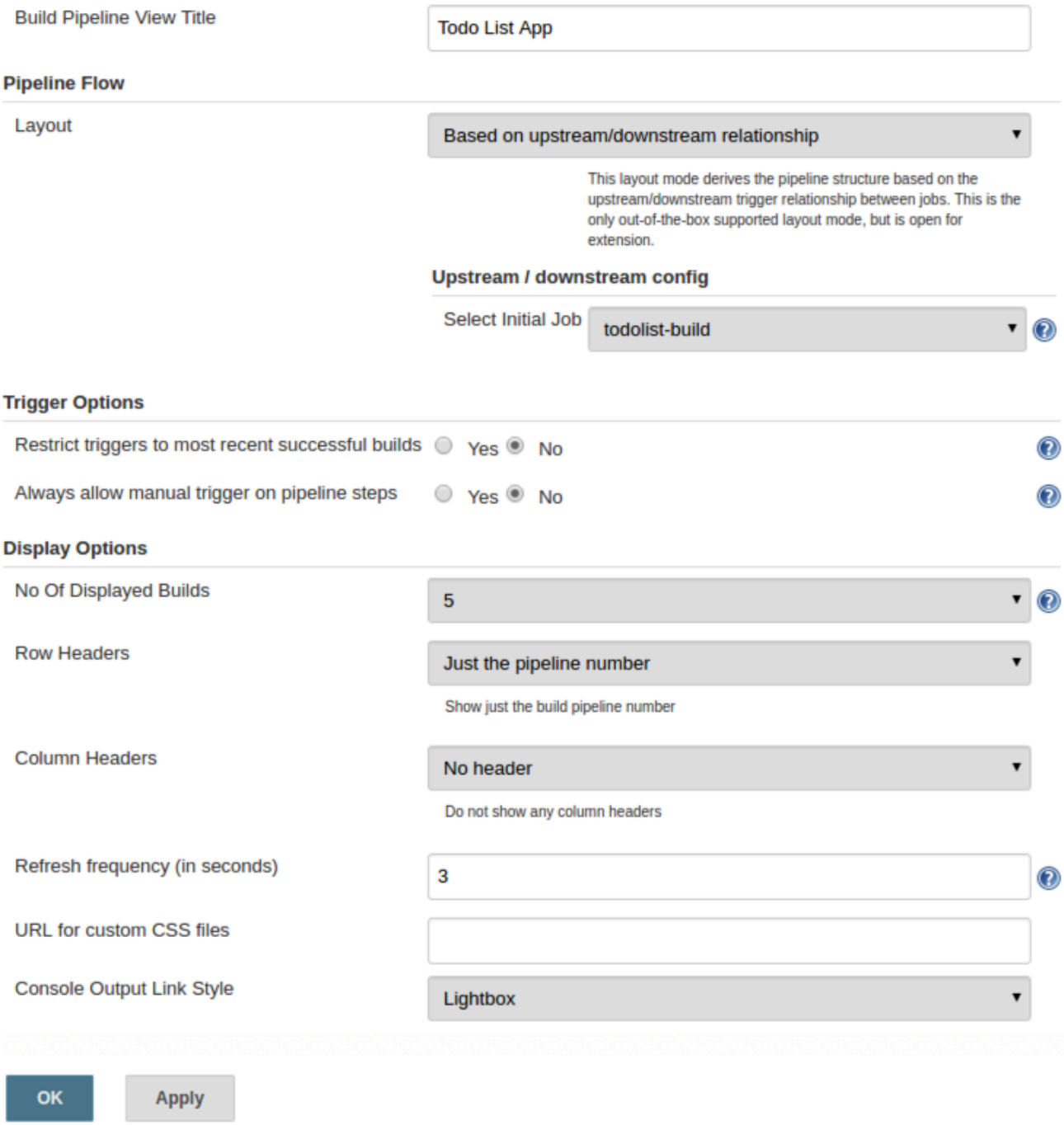
NOTE - It may take some time when loading this page, as it can appear blank. Try refreshing the page if it fails to load. Sometimes, Jenkins will fail to load all the plug-ins and the page will remain blank. In this case, try checking your internet connection with the VM. If there is no connection, you 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://jenkins.server:8080/restart and click **OK** on restart.

- Create a build pipeline view by Navigating to the Jenkins homepage, and then click on the + icon in the view tabs bar:



- Enter `todolist-pipeline` in the *Enter an item name* field. Select the *Build Pipeline View* radio button and then *OK.*
- Enter `Todo List App` as the *Build Pipeline View Title* and select *todolist-build* as the *Select Initial Job.* Select *5* as the *No Of Displayed Builds*:



- Click *OK* to save the configuration. If correctly configured, your pipeline should now be shown:

The pipeline id on the left is the build number associated with the pipeline starting job, *todolist-build*. The plugin uses the downstream/upstream jobs to determine the order and jobs in the pipeline. You can see that build 17 above, has resulted in a successful promotion of the build to si as denoted by a pipeline of green steps. The build before was not configured to trigger the si job, as denoted by light blue for not run. Finally you can see that build 15 failed to deploy to ci the first time, as denoted by a red step. You may later see amber steps, which means an unstable build has occurred.

- The next step is to extend the pipeline through to production. Navigate to Jenkins, http://localhost:8080/, and create a new job by clicking on *New Item.* Name the job by entering in `todolist-deploy-production` in the *Enter an item name* enter field. This time we will make a shortcut. Enter in `todolist-deploy-si` in the *Copy from* field then click *OK:*

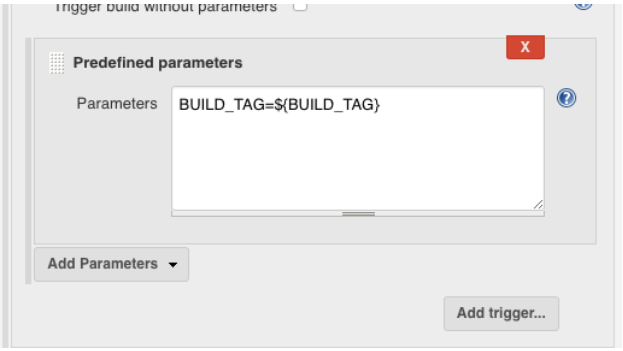
- The *todolist-deploy-production* job is almost identical to the *todolist-deploy-si*, except for the environment configuration.
- Navigate to the *Execute Shell* step in the *Build* section and replace `si` with `production` as follows:

```
npm install

grunt deploy:production:${BUILD_TAG}
```

- Now we need to reconfigure the upstream job, *todolist-deploy-si*, to trigger the *todlist-deploy-production* and provide it with the *BUILD_TAG.* Go to Jenkins homepage and navigate to the *todolist-deploy-si* build job and click *Configure*.
- Now configure the job to trigger the downstream job, *todolist-deploy-production*, with the *BUILD_TAG* parameter. Navigate to the *Post-build Actions* tab. Click *Add post-build action* and select *Trigger parameterized build on other projects*

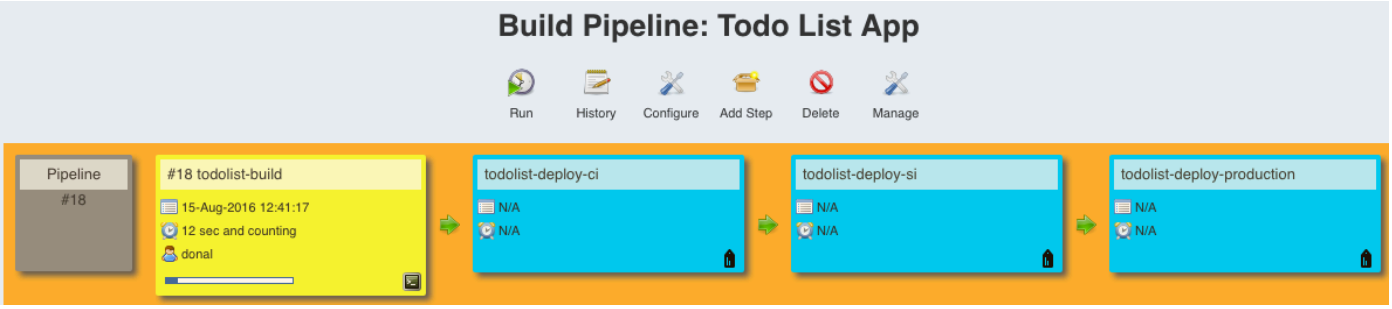
and enter `todolist-deploy-production` as the *Projects to build*. Click Add Parameters and select *Predefined parameters* from the drop down. Now enter `BUILD_TAG=${BUILD_TAG}` as follows



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

- Now we can kick off our pipeline build. Navigate to Jenkins home and click on the *todolist-pipeline* view created previously:

- You will now see that there is a new stage in our pipeline, the *todolist-deploy-production* step. You can run the pipeline from here by clicking *Run:*



Whilst the application is deploying, I shall let you into a secret. The production version of the application is currently configured to seed the DB. This is not what we want in our production instance. In production, we want to persist my todo list forever so that I don't get in trouble for not buying milk.

- Whilst the pipeline is running, open up the todolist application code in Atom by running the following:

Terminal	`atom ~/todolist`

- Edit the file in *server/config/environment/production.js* and change the value of the `seedDB` property to `false` as follows:

Atom	*`server/config/environment/production.js`*

```
todos.html | todo.spec.js | todo.model.js | todos.spec.js
'use strict';

// Production specific configuration
// =================================
module.exports = {
  // Server IP
  ip:       process.env.OPENSHIFT_NODEJS_IP ||
            process.env.IP ||
            '0.0.0.0',

  // Server port
  port:     process.env.OPENSHIFT_NODEJS_PORT ||
            process.env.PORT ||
            80,

  // MongoDB connection options
  mongo: {
    uri:    process.env.MONGOLAB_URI ||
            process.env.MONGOHQ_URL ||
            process.env.OPENSHIFT_MONGODB_DB_URL+process.env.OPENSHIFT_APP_NAME ||
            'mongodb://mongo.server/todolist-prod'
  },
  seedDB: false
};
```

This file contains the environment specific configuration for each of the environment. It is important to note that this is configuration as code. All of the build, test and deploy steps use this file as a reference, which is under version control and versioned with each build of the app. This ensures tight control over configuration management.

To save space in the VM however, we use the same mongo instance for each environment, and separate the data by providing each environment with their own table name, which is `todlist-prod` in the production environment. This can be overridden by environment variables if necessary. If we were not constrained, then we would obviously separate the data into different mongo intances, and potentially make the backup strategy more production like the further downstream the pipeline the code is, for example we may introduce sharding and/or load balancing.

- Now save the file with `CTRL+ S` (or File > Save) and commit and push the code to the remote. First check that your changes have been recognised by git by running the following from within the `~/todolist` directory:

```
git status
```

The expected output is:

```
➜ todolist git:(develop) git status
On branch develop
Your branch is up-to-date with 'origin/develop'.
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

        modified:   server/config/environment/production.js

no changes added to commit (use "git add" and/or "git commit -a")
```

- Now to push these changes to the remote, you need to add the files to the index, also knows as stage the files, execute:

```
git add server/config/environment/production.js
```

There will be no output from this command.

- To commit these changes to your local git repo, execute:

```
git commit -m 'do not seed production environment so that production data
is persisted on environment restart'
```

The expected output is:

```
➜ todolist git:(develop) ✗ git commit -m 'do not seed production environment so that production data is persisted on en
vironment restart'
[develop 0743f41] do not seed production environment so that production data is persisted on environment restart
 1 file changed, 2 insertions(+), 1 deletion(-)
```

- To push these changes your the remote, execute:

```
git push origin develop
```

The expected output is:

```
Counting objects: 6, done.
Delta compression using up to 8 threads.
Compressing objects: 100% (6/6), done.
Writing objects: 100% (6/6), 628 bytes | 0 bytes/s, done.
Total 6 (delta 4), reused 0 (delta 0)
To ssh://git@springdo.sbank.uk.ibm.com:2224/home/git/todolist.git
   0e9f0af..0743f41  develop -> develop
```

The update to the remote will trigger a build of the *todolist-build* job on Jenkins. Remember that before we asked Jenkins to poll for SCM changes every minute. This has helpfully caused the *todolist-pipeline* to trigger, and will in turn automatically deploy to production.

- Go back to the *todolist-pipeline* view and wait for the deploy to production to occur. Now try out your changes by going to http://todolist.ibm.com/, add some todos, and redeploy the application by triggering a *parameterized build* on *todolist-deploy-production*. Make sure that you enter the correct build number in the build tag, otherwise you will get an old version of the application without this new functionality.

- Once the *todolist-deploy-production* build has completed go to http://todolist.ibm.com/ and see that your todos are still persisted even after a redeploy.

We have just completed our skeleton pipeline that builds and promotes all the way to production! In later labs we will be filling out this pipeline to increase our confidence in the built application.

Comments

There are no comments.