Installation steps ConcourseCI

Prerequisites:

- Docker installed
- Docker Compose installed
- Demo project cloned / downloaded: https://github.com/codecentric/TDDTrainingApplication
- Checkout the “concourse” branch:

```
git checkout concourse
```

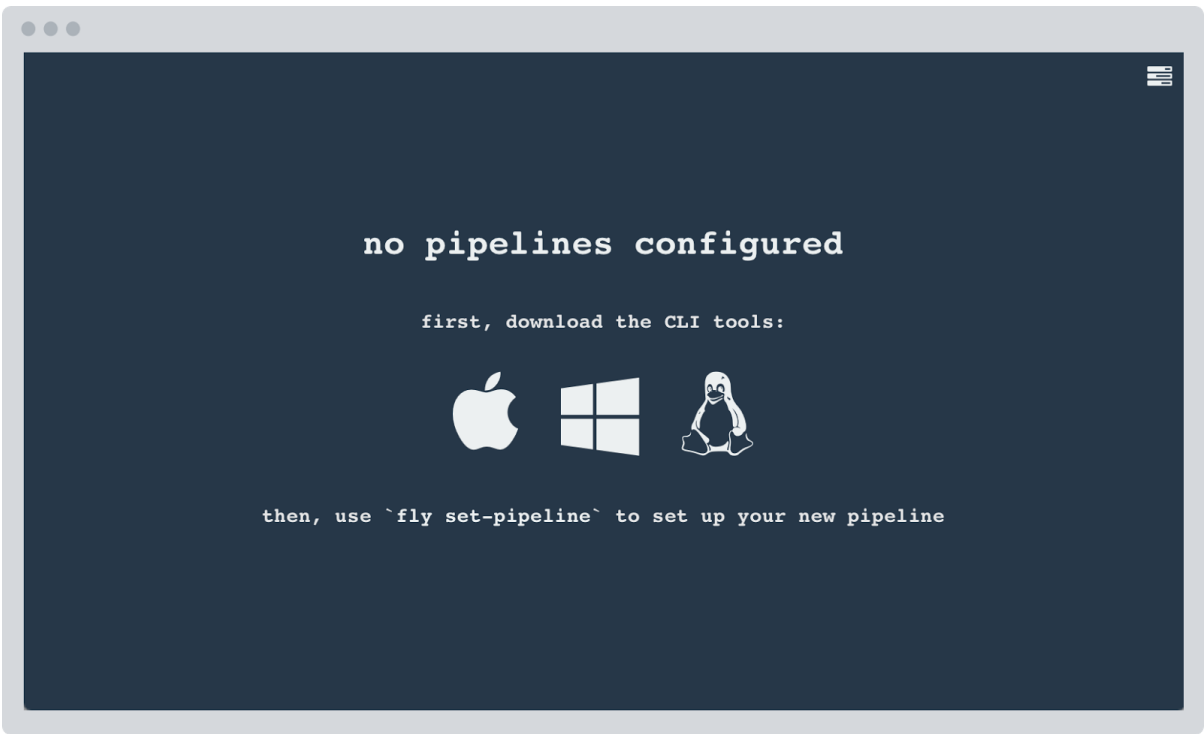
Concourse Install Steps

- Follow first steps
  https://concourse.ci/docker-repository.html
  For `CONCOURSE_EXTERNAL_URL` choose your external IP if running on your local machine, otherwise the docker-machine IP, e.g.

```
$ export CONCOURSE_EXTERNAL_URL=http://10.76.213.59:8080
```

- Wait until fully started after the “`docker-compose up`” command
- Make sure you can login at http://localhost:8080 with concourse / changeme
- Download the CLI tools for your own platform (MacOS, Linux or Windows), add to the PATH and make executable with:

```
$ chmod +x fly
```



- Set default target for the fly CLI tool (login with same username / password as before):
  ```
  $ fly -t lite login -c http://localhost:8080
  ```

  (if you get a message "resource not found", try to use the same IP as you have used for the external address instead of localhost)

  You can now use "-t lite" with commands to always connect to this instance.

Now we're all setup to create our first build pipeline!

First Build Pipeline

Test Build

First we'll execute a single command with Concourse, just to test everything works.
Create a yaml file named "task_test_maven.yml" with the following contents:

```
---
platform: linux

image_resource:
  type: docker-image
  source: {repository: maven, tag: "3-jdk-8"}

run:
  path: mvn
  args: [-version]
```

Then execute the command:

```
fly -t lite execute -c task_test_maven.yml
```

This should download the Maven Docker container and run the "maven -version" command, outputting the version information.
We'll use this as basis to create the first build-step.

Maven Build

Next we'll create our first real build-step based on the above task file.

Create a new yaml file based on the previous, in the following location:

```
TDDTrainingApplication/TDDTrainingApplicationCC/ci/task_build.yml
```

By convention the Concourse configuration files are placed in a sub-directory of the code that is being tested.

To have Maven really build our project, we need to load the files into the container. Concourse will do this for us by using “inputs”. Add this to the “task_build.yml” file:

```
inputs:
  - name: TDDTrainingApplicationCC
```

The name needs to be chosen carefully. If it matches the directory from where running the “fly” command, it will pick up this automatically. Otherwise you need to supply inputs using “-i name=directory” where “name” matches the name in the yaml file.

Next also change the “run” section to provide the “clean install” parameters and make sure the working directory for the Maven command is set correctly (tip: see the available parameters here: https://concourse.ci/running-tasks.html#task-run)

Next, try to see if the build works:

```
fly -t lite execute -c ci/task_build.yml
```

TIP: To see it really works, comment out a section of the code so that it won’t compile or tests will fail, and then run the execute command again. The build should end with “failed”.

Integration Build

For the next step we’ll execute the integration / BDD tests using a separate Maven profile and with Firefox. We’ll need a different Docker image containing Firefox. You can use this one: “khozzy/selenium-java-firefox”.

To execute the tests, use the following Maven command:

```
mvn verify -Pintegration
```

You’ll need to wrap the command using xvfb (a virtual framebuffer) because we don’t have an actual X environment. The command for this is “`xvfb-daemon-run`”

This task should also finish with “succeeded”.

Creating the pipeline

Now we’ll create the pipeline out of these two build steps.
There is one main difference! Up until now, tasks could be executed from local sources. But for a pipeline to work, we need to use Git, S3, or some other source.

Create a “pipeline.yml” file and define a git resource with:
- uri = https://github.com/codecentric/TDDTrainingApplication.git
- branch = concourse-solution

See the resources here how to define all parameters: https://github.com/concourse/git-resource

Next, also define the first “job”. We’ll refer to the build task created earlier, however now it’s checked-in together with our sources. Therefore the “paths” must match with the name of the resource defined. The job-part should look something like this (for you to fill in the blanks):

```
jobs:
- name:
  plan:
  - get:
  - task:
    file:
```

(Name, get, task and file fields must be supplied, information here: https://concourse.ci/configuring-jobs.html)

When you’re finished, create the pipeline in your Concourse installation (you can use the same command for updating an existing pipeline):

```
fly -t lite set-pipeline -c ci/pipeline.yml -p tdd-app-pipeline
```

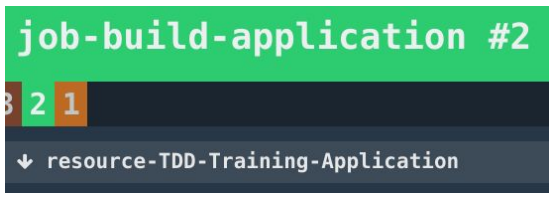
Or:

```
fly -t lite sp -c ci/pipeline.yml -p tdd-app-pipeline
```

(In the second command, “sp” is shorthand for “set-pipeline”)

Enable the pipeline from the GUI by opening the menu on the left, and clicking the “play” icon. Then start the pipeline by opening the job and clicking on the “plus” icon on the top-right. You should now see some output.

To explain a bit more; we have now one **job** with a build-**plan** defined, consisting of two steps. The first **get** step fetches ‘down’ one of the available resources. You can see this in the GUI with the down-arrow next to the resource name:



The second step is the **task**, referring to an external file for the task definition. Tasks could also be defined in-line, but for readability and reuse, it's advised to keep them separate.
Do keep in mind though that by extracting the tasks into separate Yaml files stored in the repository, you need to "`git commit`" and "`git push`" for the pipeline to fetch any updated task information.

Pipeline with Two Steps

Now let's add a second step, which is triggered when the first build step is finished successfully.

To correctly add this second step, the following changes are needed:

- Create a second **job** with **name** "job-integrationtest-application", comparable to the first **job**.
- Add the "`serial: true`" value to the first build-job, so that it won't run in parallel with the second job.
- Add a trigger to the second job, so that it will be started when the first is finished (and _only_ when it is finished successfully). See here for reference: https://concourse.ci/get-step.html#passed

Update the pipeline with:

```
fly -t lite sp -c ci/pipeline.yml -p tdd-app-pipeline
```

Next, navigate to the web interface again and verify that two jobs are shown. Start the first job and see that the pipeline finishes correctly.

Cleanup

If you want to cleanup after the Concourse workshop, remove the following files and directories:

- The test project
- The Concourse "fly" command
- The ".flyrc" file in your home-directory
- The Concourse docker containers and images