Kotlin all the things

So, after all, it seems JetBrains is very serious with Kotlin. And I have to admit it comes with some handy features and good IDE support. But this is not about the Kotlin language, this is about where it can be used.

As we have seen in Migrating from Gradle to Gradle writing Gradle build scripts using the new Kotlin DSL is supported. So far we have

- our sources in Kotlin
- our build configuration in Kotlin
- but not our Continuous Integration configuration
  (depending on how far you want to push it your build chain or pipeline as well)

Since TeamCity (the CI server) is from JetBrains as well, it supports storing your build configuration not only over the UI. It supports storing the configuration in your VCS in XML format and (since around version 10 and 2017 in a Kotlin format. The current version 2019.1 comes with even more improvements and simplifications in this section. So throw away your build config yaml file! Kotlin you build config too! Although this seems a bit weird at the beginning, there are some big advantages:

- it real source code
- it compiles
- you can share common pipeline definitions via libraries
- since Kotlin is a typed language, there is a nice support for in your IDE with auto-completion
- you can compile the code prior to pushing it - compared to the try and error cycle that YAML config files come with, I'll argue it is the better method

There is a very nice Blog from JetBrains on this topic that I can highly recommend to read:

- Configuration as Code, Part 1: Getting Started with Kotlin DSL
- Configuration as Code, Part 2: Working with Kotlin Scripts
- Configuration as Code, Part 3: Creating Build Configurations Dynamically
- Configuration as Code, Part 4: Extending the TeamCity DSL
- Configuration as Code, Part 5: Using DSL extensions as a library
- Configuration as Code, Part 6: Testing Configuration Scripts

And while you're at it, maybe watch the webinar on "Turbocharging TeamCity with Octopus Deploy" as well. Octopus is an additional commercial service. But distinguishing between continuous integration and deployment seems a good split in responsibilities.

Since I consider you know to be convinced let us test-drive it! We first need a TeamCity Server, a TeamCity Agent, and an example project.

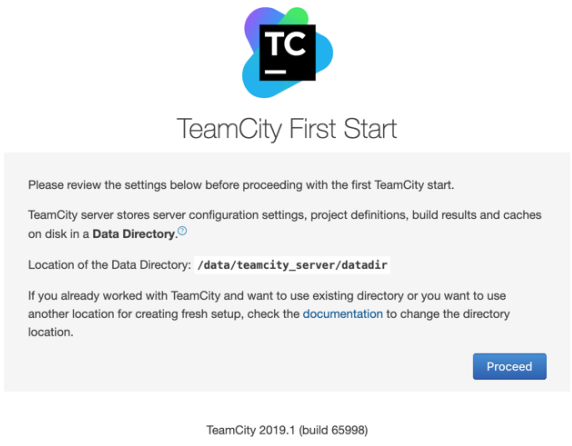
To have it quickly set up to test-drive, I created a docker-compose.yaml file (yes a yaml file, isn't it ironic): https://github.com/brontofundus/kotlin-all-the-things

Clone the repository and fire it up:

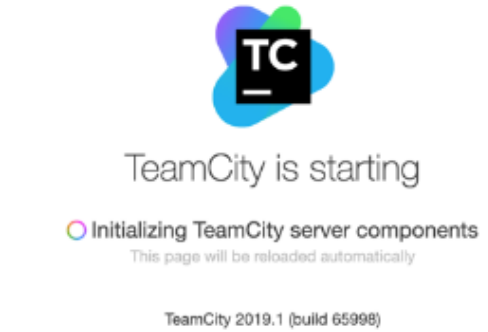
```
$ docker-compose pull
$ docker-compose up -d
```

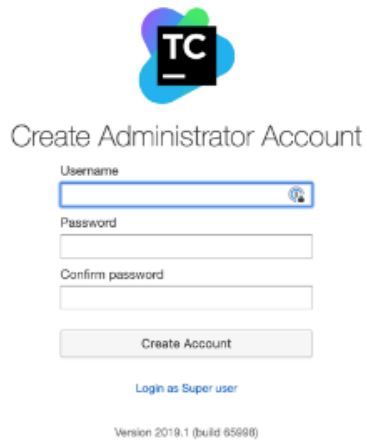
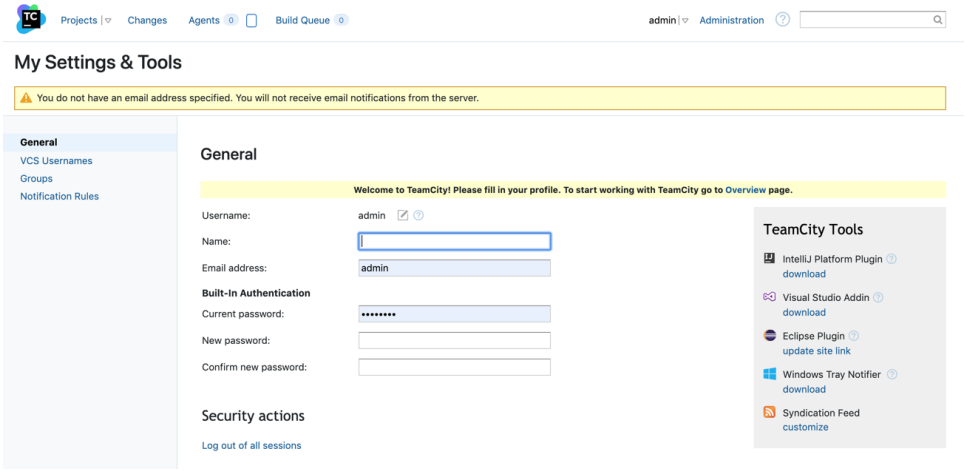
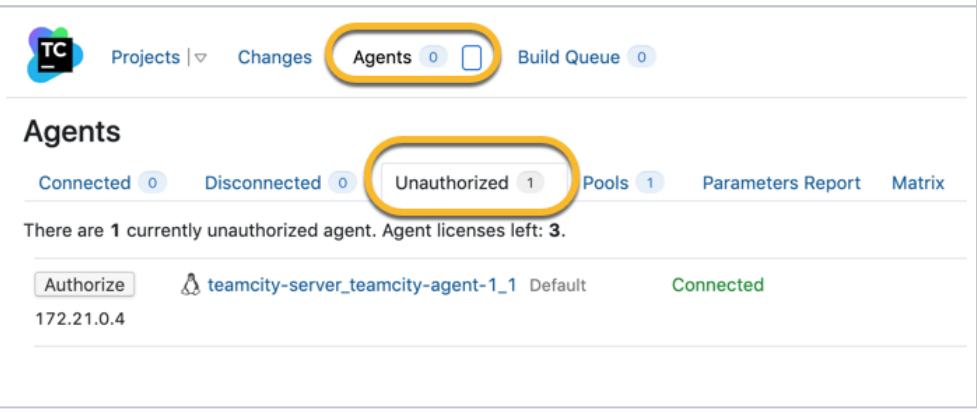
Then point a browser to http://localhost:8111/

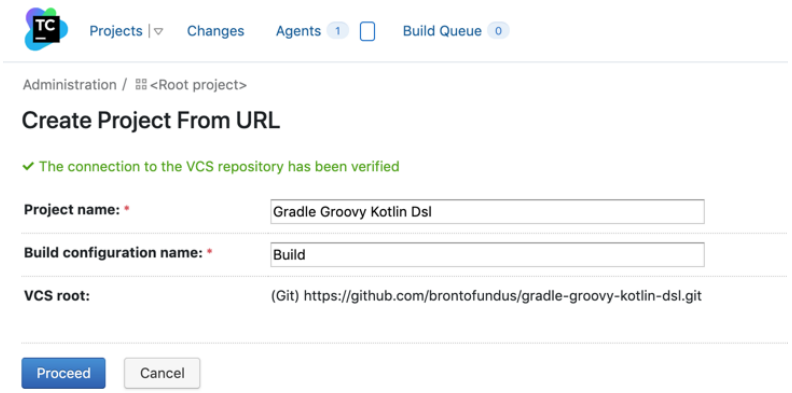
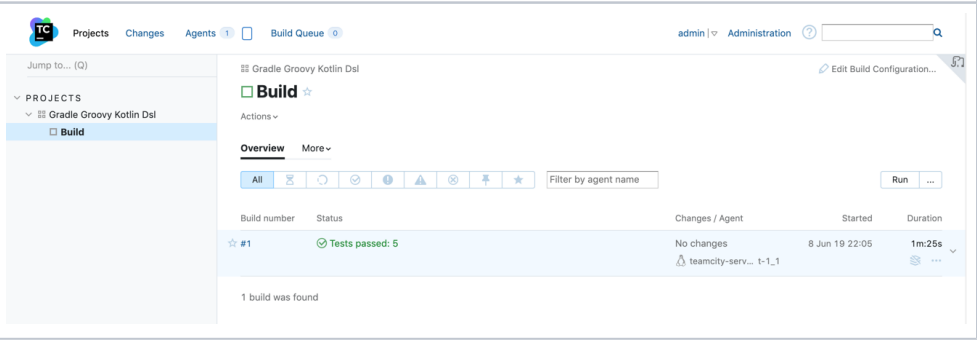
This will bring you to the TeamCity installer. But don't panic, it will only take a few minutes!

Proceed	

• Select PostgreSQL
• download the driver
• and use the same settings as used in the docker-compose file:
  • "postgres" as host
  • "teamcity" as username, password and database name
 	
Proceed
	

Wait	
Scroll down and agree to the license (well read it of course - but don't tell me you are not used to selling your soul)	

Create an admin account (for simplicity use "admin" and "password" here)	
There you are	
As you may notice on top there are 0 agents. Which is not entirely true. But to have the one we have enabled it needs to be authorized first	
Go to "Agents" and "Unauthorized" and enable the agent	

If you now go to the start page (click on the logo on the top left) you will be able to add a project	
Builds run already	

If you have a close look at the project you will notice it contains a "`.teamcity`" directory that contains the build configuration: https://github.com/brontofundus/gradle-groovy-kotlin-dsl/tree/kts/.teamcity

The new 2019.1 format comes in a "portable" variant. So the number of files to earlier TeamCity version is reduced to only the settings.kts file and the pom.xml

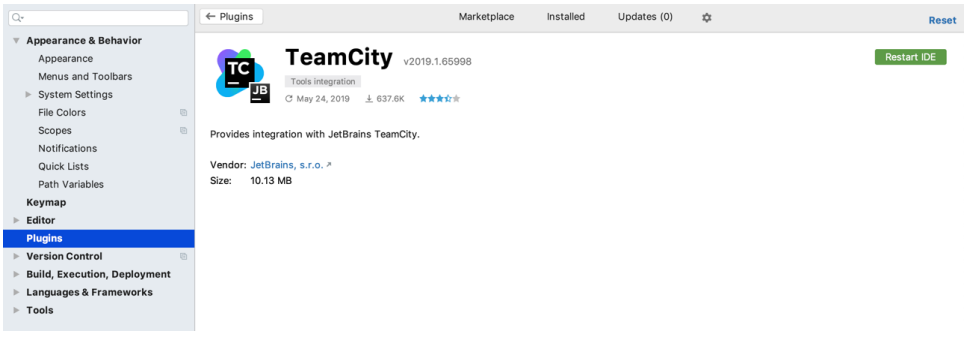
From here on no more clicking in the UI is necessary.

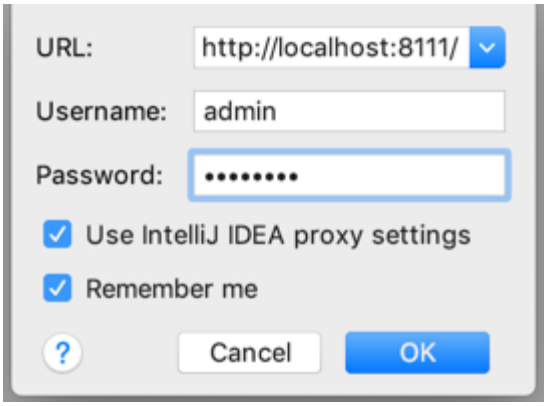
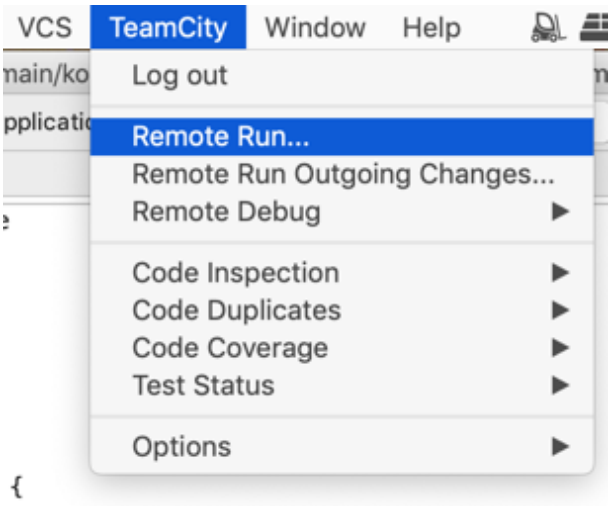
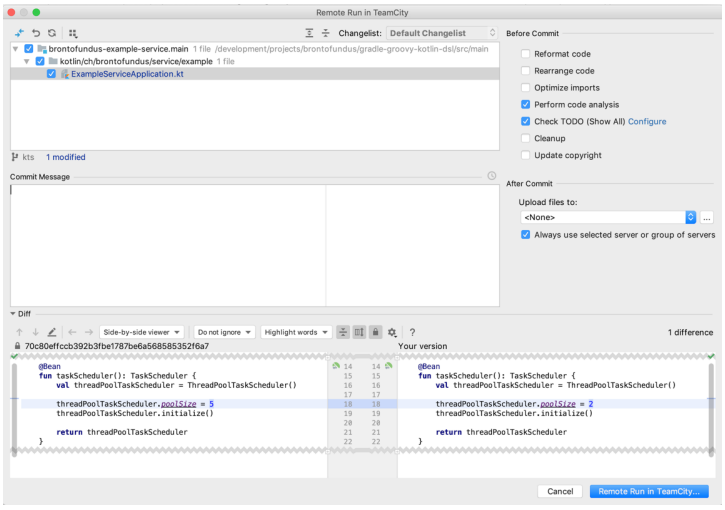
And even more comfort

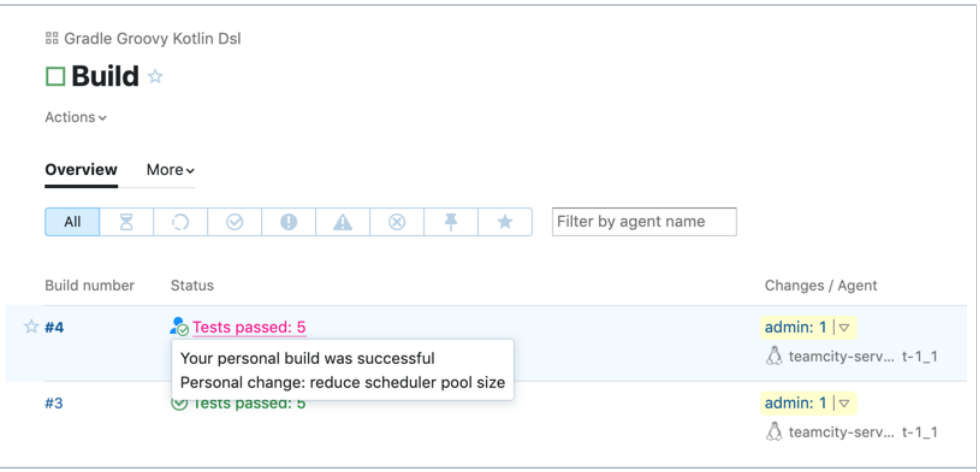
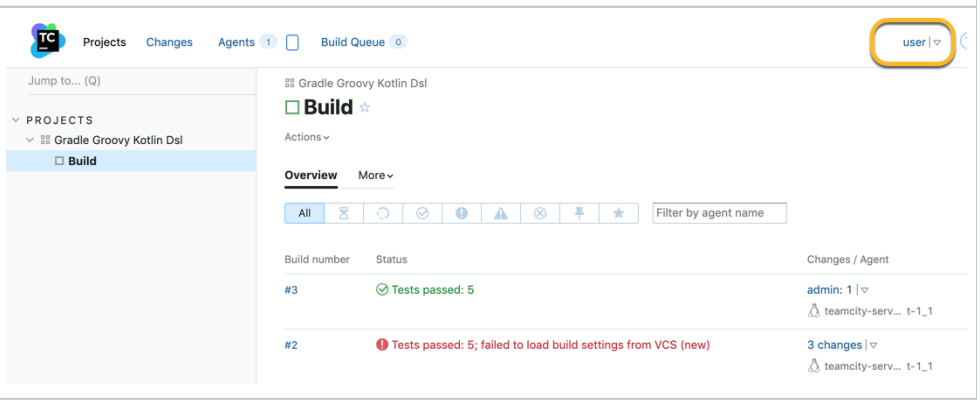
Since you probably use IntelliJ to develop in Kotlin anyway and you now have a running TeamCity server from the same company, even more, comfort is possible!

Just install the TeamCity plugin in IntelliJ and point it via the new menu entry to your local server.

This will show the build status of your projects, but in addition, also allows you to run your local changes remotely as personal build! This is not a new feature but people tend to forget about it:

Install the plugin in IntelliJ	

Restart and point it to the local TeamCity server (we used "password" as password above)	
And you can now remote run builds with local changes!	
Even with not yet committed changes	

Personal builds have this nice additional icon to mark personal builds. These are only visible to the user that created them	
If you create an additional user and re-login you will not see other peoples personal builds	

So we have:

- our software in Kotlin
- our Gradle build script in Kotlin
- our CI configuration in Kotlin - able to share and re-use our build chains (read the JetBrains blog above for more details on this)
- and free of charge with this setup: remote runs

What a Kotlin world to live in!