Flask op de raspberry pi

Flask op de raspberry pi 1

Installatie op de raspberry pi

Python en Flaskprojecten kunnen zonder veel installatie vanop je rasp uitgevoerd worden.

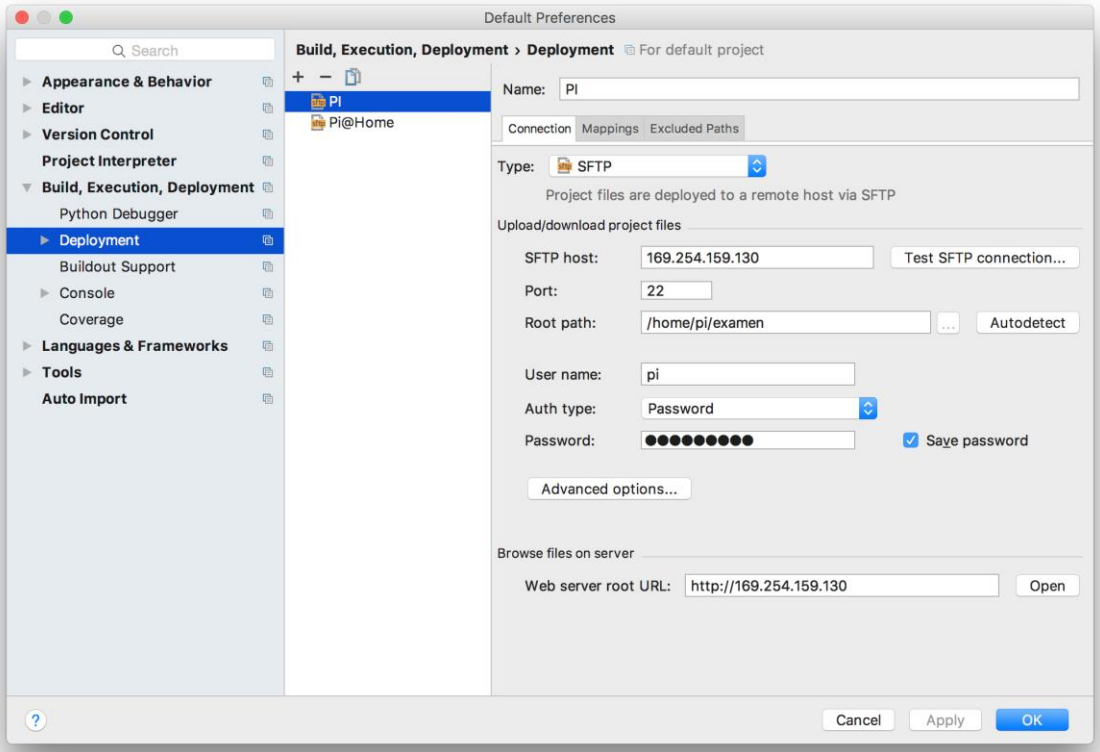
Instellen van Pycharm voor remote connection

Tijdens de labo's van *Device Programming 1* hebben we ervoor gekozen om de Flask Webserver te laten werken op onze lokale machine (127.0.0.1: 5000). Dit zorgde ervoor dat we geen extern device nodig hadden en dat we de oefeningen ook snel en makkelijk konden testen.

Voor de module Project 1 zal het echter nodig zijn om de bestanden up te loaden naar de Raspi en de server daar aan de praat te krijgen.

Deployment environment instellen

Ga in Pycharm naar de default setting en kies daar bij *Build, Execution, Deployment* voor *Deployment*



Let natuurlijk op dat je de juiste gegevens invult. De SFTP host is het netwerkadres van je Raspi. Bij mij is deze aan de computer gekoppeld met een rechtreekse kabel (Apipa). Indien je een extra user gemaakt hebt (niet pi – raspberry), dan kan je deze invullen bij User Name.

Druk nadien op Test SFTP connection. Indien de connectie geslaagd is kan je verdergaan. Klik ook nog even op de Autodetectknop onder de Test SFTP knop. Klik daarna onderaan op OK.

Sluit ook het volgende venster door op OK te drukken.

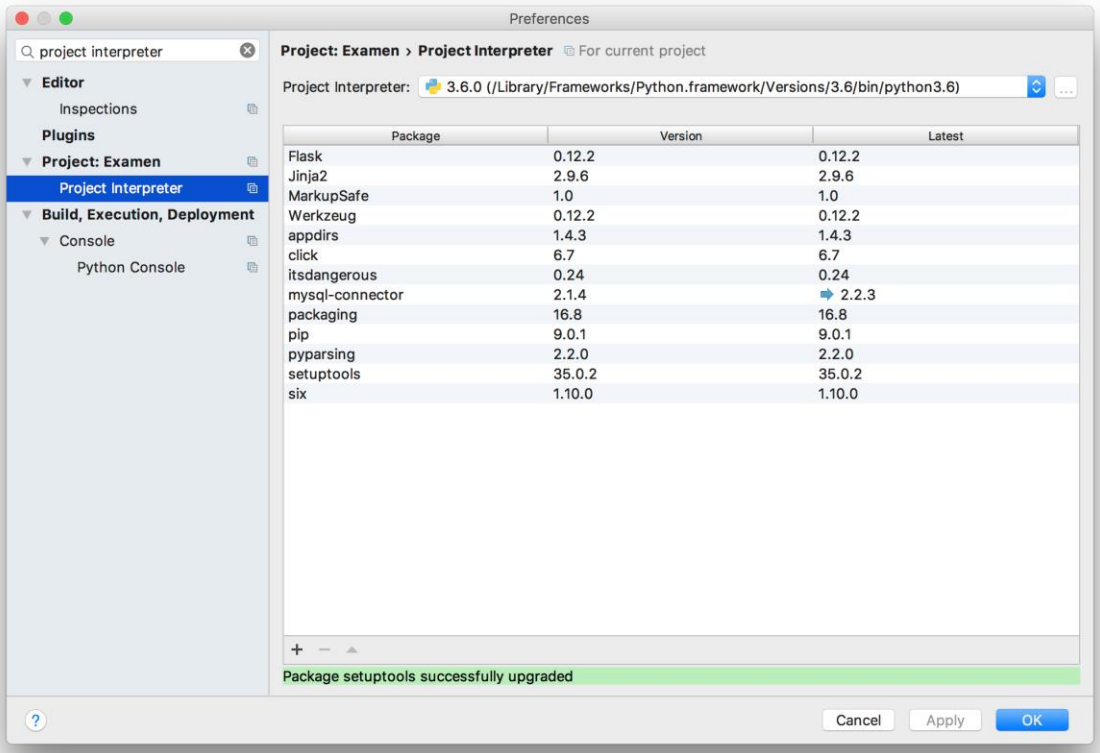
Daarna zal PyCharm connectie maken met het externe device en een aantal "Pycharm helpers uploaden" naar de raspi. Heb geduld en sla dit zeker niet over.

Door een deployment environment aan te maken, zal PyCharm onze externe device kennen en ermee kunnen communiceren.

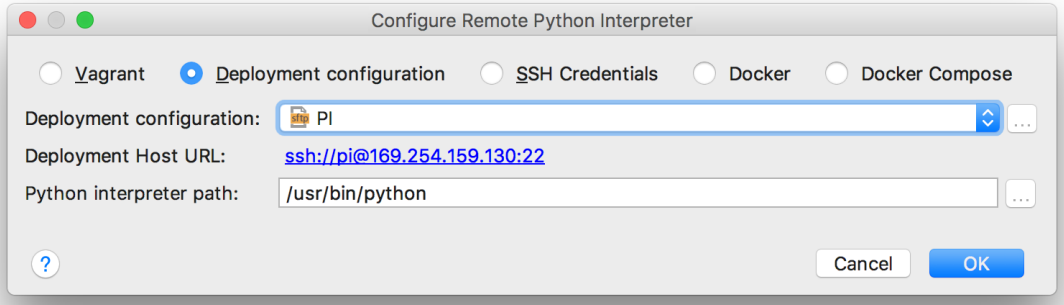
Project interpreter instellen

Druk in PyCharm op Ctrl+Shift+A en tik "project interpreter" en druk op enter.

In het venster dat opent klik je rechtbovenop de … en kies voor *add remote.*



In het volgende venster kies je voor Deployment configuration en kies je voor de deployment configuration die je in de vorige stappen aangemaakt hebt.



Bij het Python interpreter path kies je best voor de map waarin python3 geïnstalleerd staat.

Meestal is dit /usr/bin/python3.

Indien je dit niet aanpast zal je blijven hangen aan een python2 interpreter.

Daarna klik je een aantal keer op OK tot je opnieuw uit de instellingen bent.

Nu zal je je project kunnen uploaden naar de raspi.

Project extern bereikbaar maken

Indien je nu je project runt zal dit uitgevoerd worden op de raspi, waarbij je nu op je raspi naar 127.0.0.1:5000 kan surfen om de pagina te tonen. Dat zal echter alleen lukken op de pi zelf.

Indien je via je laptop naar je project wil surfen, zal je je project moeten aanpassen in de *.py file.

Tik daar onderaan volgende code en start je project opnieuw:

```
if __name__ == "__main__":
    # check omgevingsvariabele voor poort of neem 8080 als standaard
    port = int(os.environ.get("PORT", 8080))
    host = "0.0.0.0"    # luister op alle IP's i.p.v. enkel 127.0.0.1
    app.run(host=host, port=port, debug=True)
```

Je zal nu (vanaf je laptop) kunnen surfen naar http://<raspi_ip>:<poort> (bv. http://169.254.1.10:8080).

Opgelet: **je kan enkel poorten > 1024 gebruiken**, die daaronder zijn “privileged” en kan enkel root openen (en je webserver als root draaien is een heel slecht idee). Met bovenstaande code kan je optioneel een andere poort meegeven op de command line:

```
pi@raspberrypi:~/PyCharm/rfid $ PORT=1080 ./webapp.py
 * Running on http://0.0.0.0:1080/ (Press CTRL+C to quit)
 * Restarting with stat
```

Als je de website toch via de standaardpoort (80) toegankelijk wil maken kan je een “reverse proxy” opzetten die poort 80 doorgeeft aan de poort waarop Flask draait, maar dat is niet zo eenvoudig. Zie bv. https://iotbytes.wordpress.com/python-flask-web-application-on-raspberry-pi-with-nginx-and-uwsgi/

Startupscripts

Loginscripts

Telkens als je inlogt, worden een aantal scripts uitgevoerd. In je homedirectory vind je het bestand .bashrc, hier kan je commando's toevoegen om de shell naar je hand te zetten. Zo kan je met het commando alias een andere (kortere) naam geven. Gebruik je bijvoorbeeld veel ls -l, dan kan je volgende regel toevoegen aan ~/ .bashrc om daarvoor de alias ll aan te maken:

```
alias ll='ls -l'
```

Het script in je homedirectory geldt natuurlijk alleen voor jouw gebruiker. Wil je een instelling globaal voor alle gebruikers maken, dan kan dat met het bestand /etc/bashrc (sudo gebruiken om te wijzigen).

Deze scripts zullen enkel geladen worden voor interactieve shells, dus niet wanneer je bv. een script uitvoert. Daarvoor kan je ~/.profile of /etc/profile gebruiken.

Initscripts

Wat in Windows "services" zijn heet onder Linux een "daemon", en wordt beheerd door het init-proces. Dat is het eerste proces dat gestart wordt bij het booten en beheert het starten en stoppen van daemons.

Als we een script automatisch willen laten starten, hebben we twee mogelijkheden. Het eenvoudigst is van een regel toe te voegen aan /etc/rc.local. Dat script wordt als allerlaatste door init uitgevoerd, nadat alle daemons gestart zijn. Let wel op dat de regel "exit 0" moet blijven staan op het einde, voeg de verwijzing naar je script dus *daarvoor* toe! Om te zorgen dat init niet blijft 'hangen' tot je script ten einde is, start je het best op de achtergrond door een & toe te voegen.

Een tweede, correctere maar ook veel ingewikkeldere methode is om zelf een daemon te maken. Daarvoor moet je een script in /etc/init.d voorzien dat minimum de parameters 'start' en 'stop' aanneemt. Dan kun je achteraf het script met het commando `service` beheren zoals de andere daemons, en ook automatisch laten starten. Instructies en voorbeeldscript hiervoor vind je op onderstaande link:

http://blog.scphillips.com/posts/2013/07/getting-a-python-script-to-run-in-the-background-as-a-service-on-boot/