PyCharm+Docker：打造深度学习的炼丹炉

首先你需要满足以下两个必备条件:

- **使用 PyCharm 专业版，记住一定是专业版（社区版不提供远程服务）**
- **在服务器上安装 docker 和 nvidia-docker1.** 新建 **docker container**

这里我使用 uforym/deepo 的 docker 配置，deepo 的github地址：https://github.com/ufoym/deepo

2. 配置 **SSH** 服务

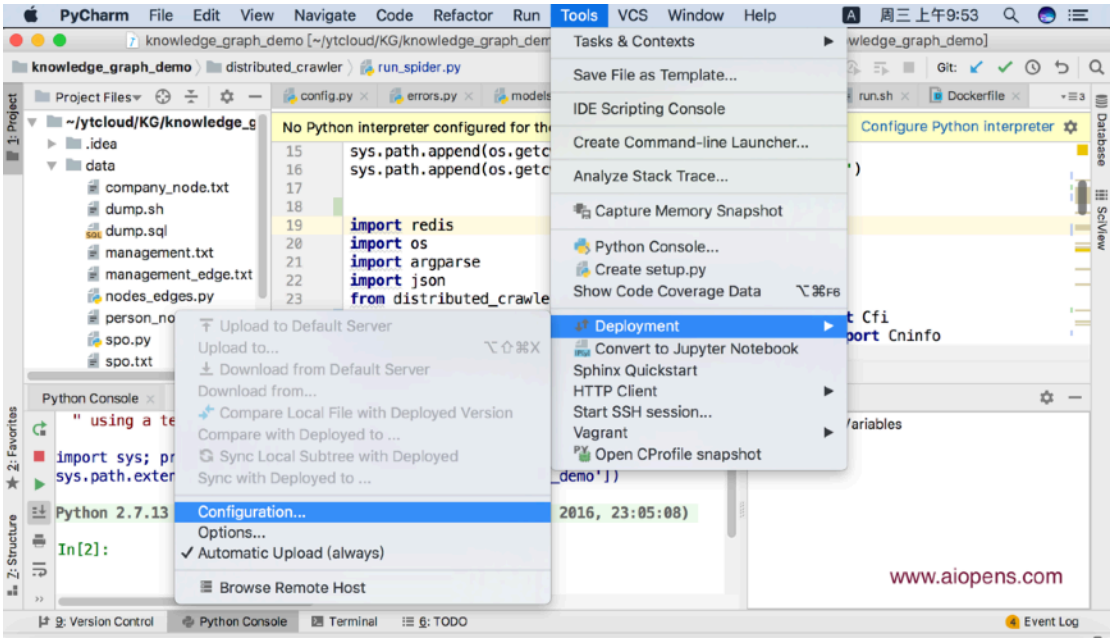
接着我们在刚刚新建的容器里配置 SSH 服务，在服务器（宿主机）上（不是服务器的 docker 里）8022端口转发新建 docker 容器中的 22 端口。

我使用了docker-compose.yml来配置 CPU 类型的deepo docker，请参看 https://github.com/jamess010/AIOpen/tree/master/algorithm/frameworks/deepo 中的配置文件。如果想使用 GPU 配置，请参看第一步中的deepo github。

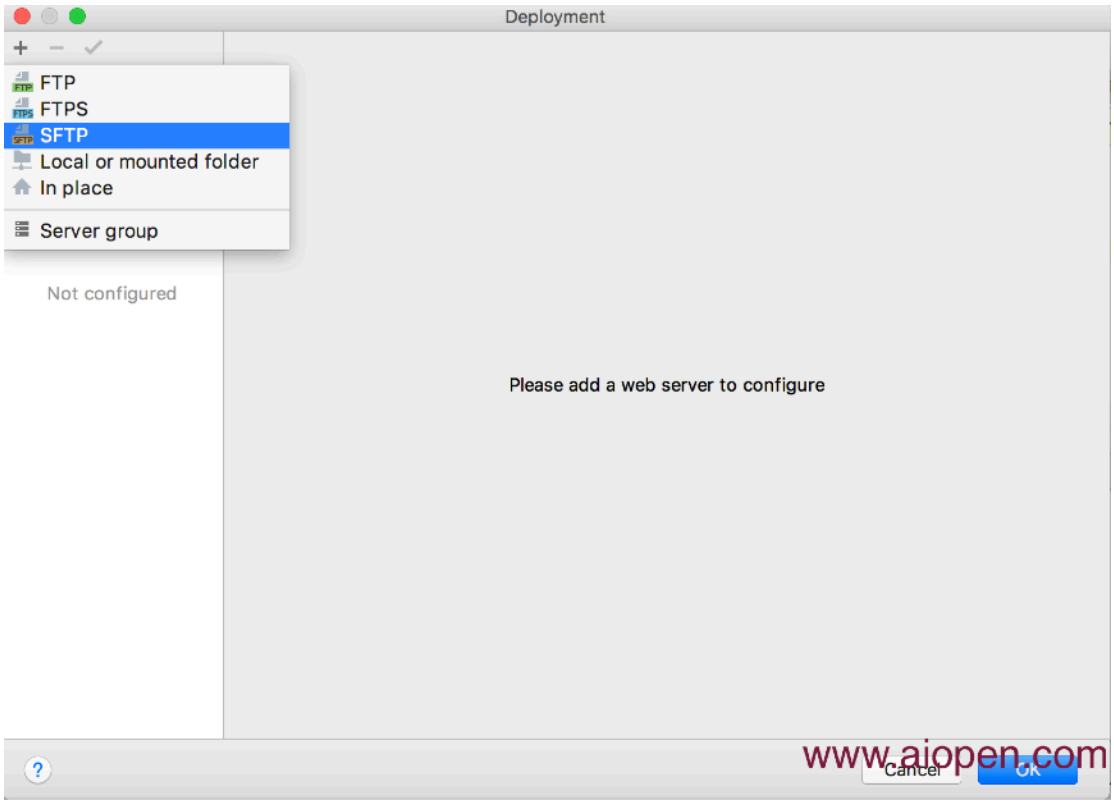
注意：GPU 需要使用 nvidia-docker 或 nvidia-docker-compose来启动，并且在 docker 中的 .bashrc 中需要添加CUDA的PATH 和 LD_LIBRARY_PATH .

3. 在 **PyCharm** 里配置部署环境

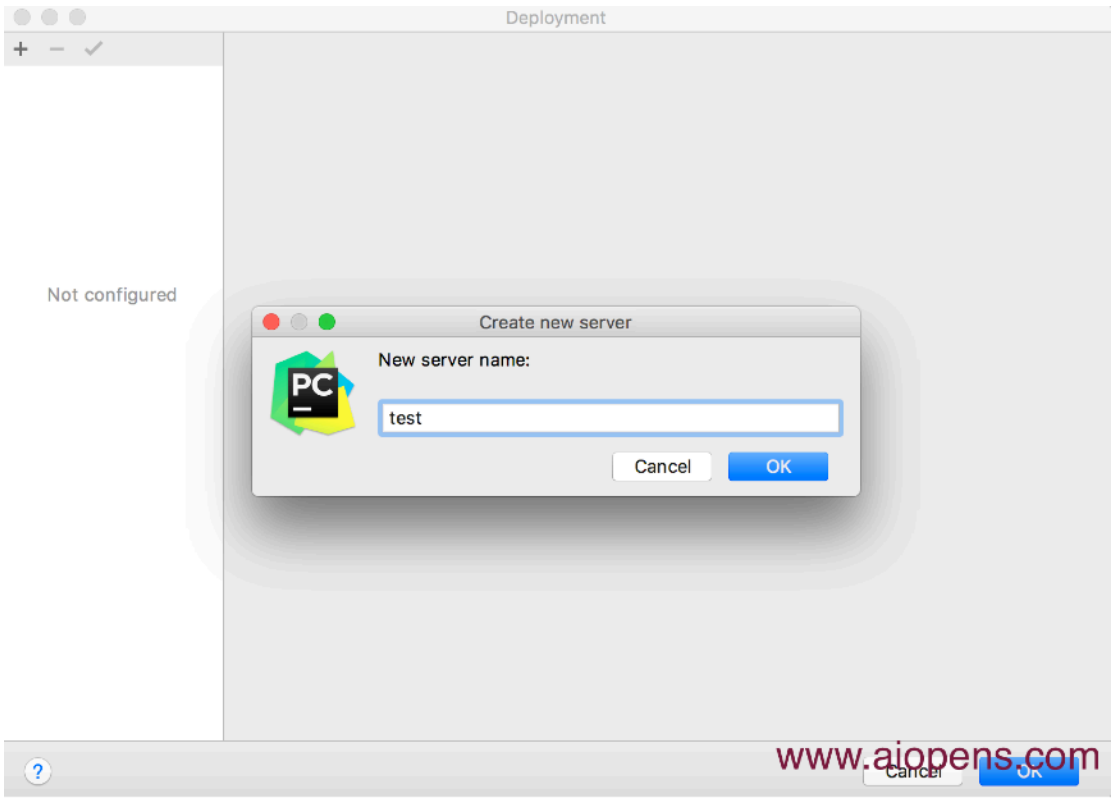
点击 Tools > Deployment > Configuration,



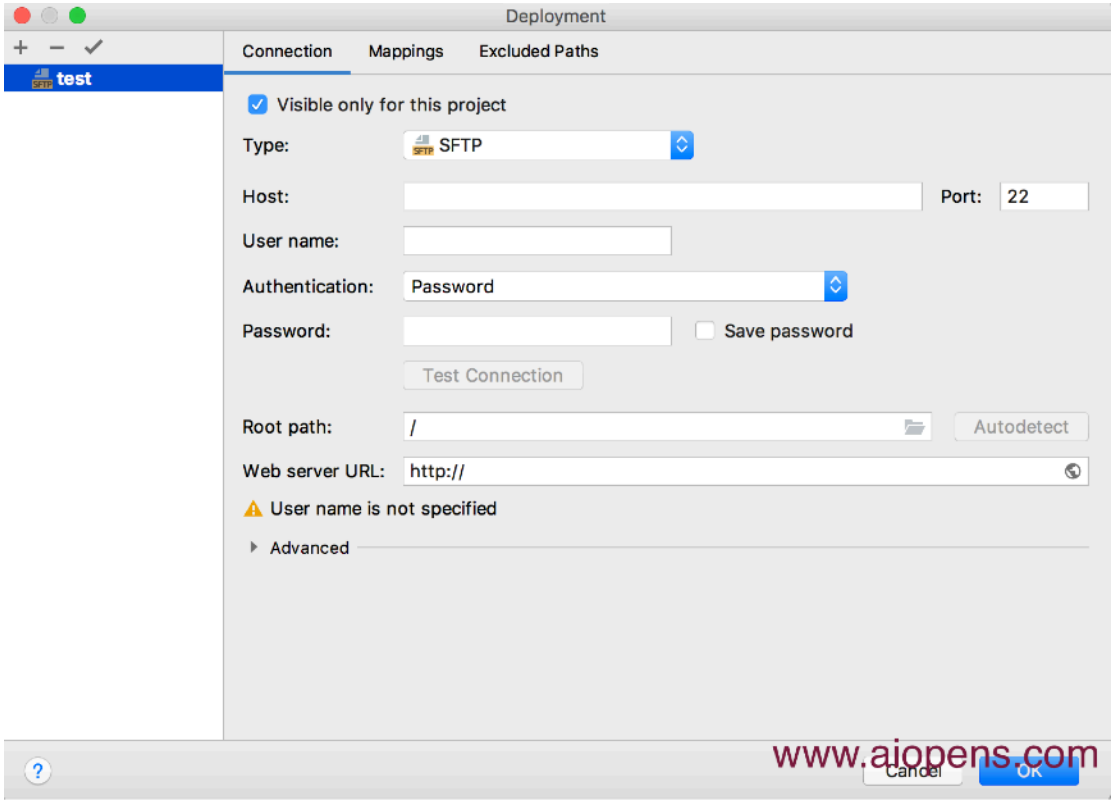
新建一个 SFTP 服务器:



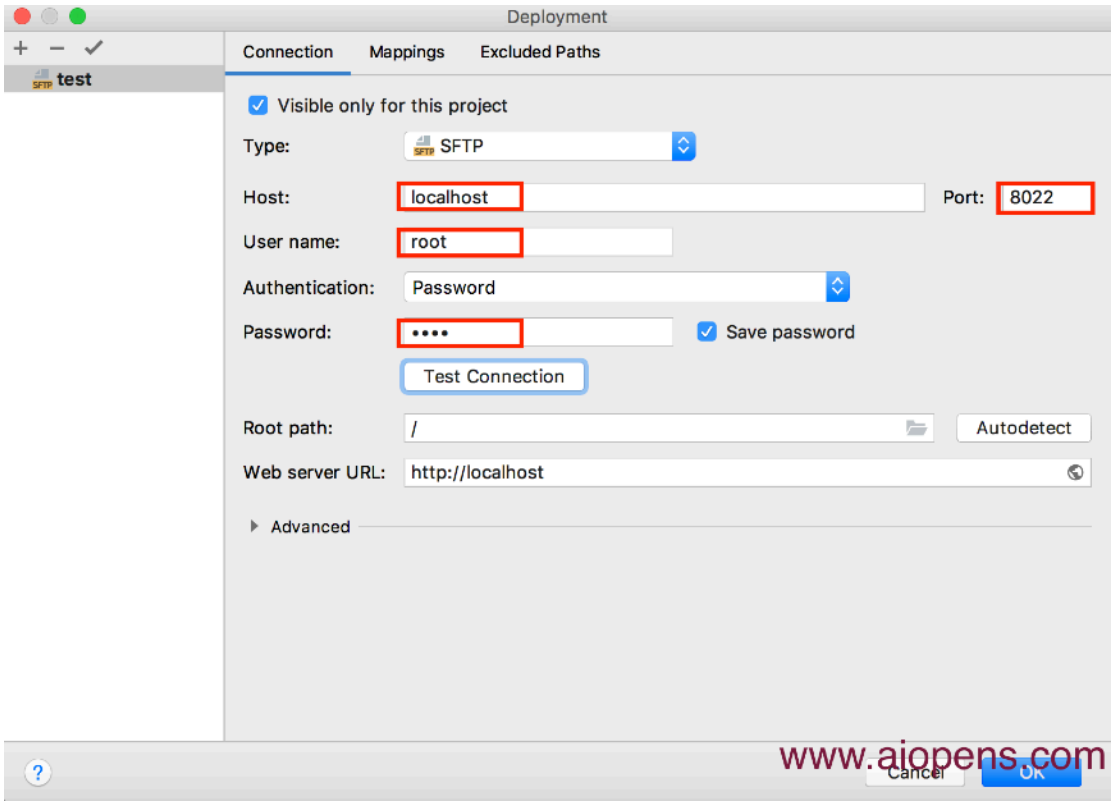
服务器名字自己取（这里是test）：



出现如下配置界面：

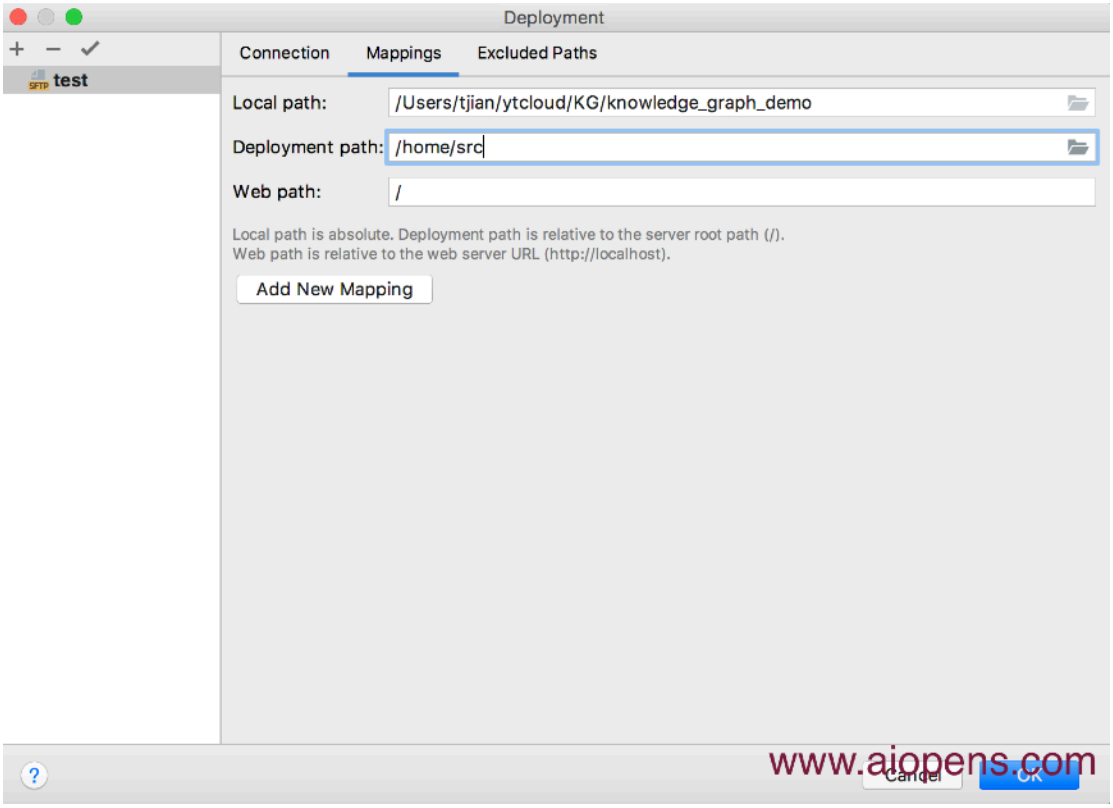


在上面输入如下图配置，注意这里的端口是你刚刚设置的映射到服务器主机 22 端口的 docker 容器中的端口 8022；账号：root，密码：test ，这是在Dockerfile里配置的，请参看 https://github.com/jamess010/AIOpen/blob/master/algorithm/frameworks/deepo/Dockerfile ，Host 设置为远程 docker 容器里的路径（这里 localhost）：



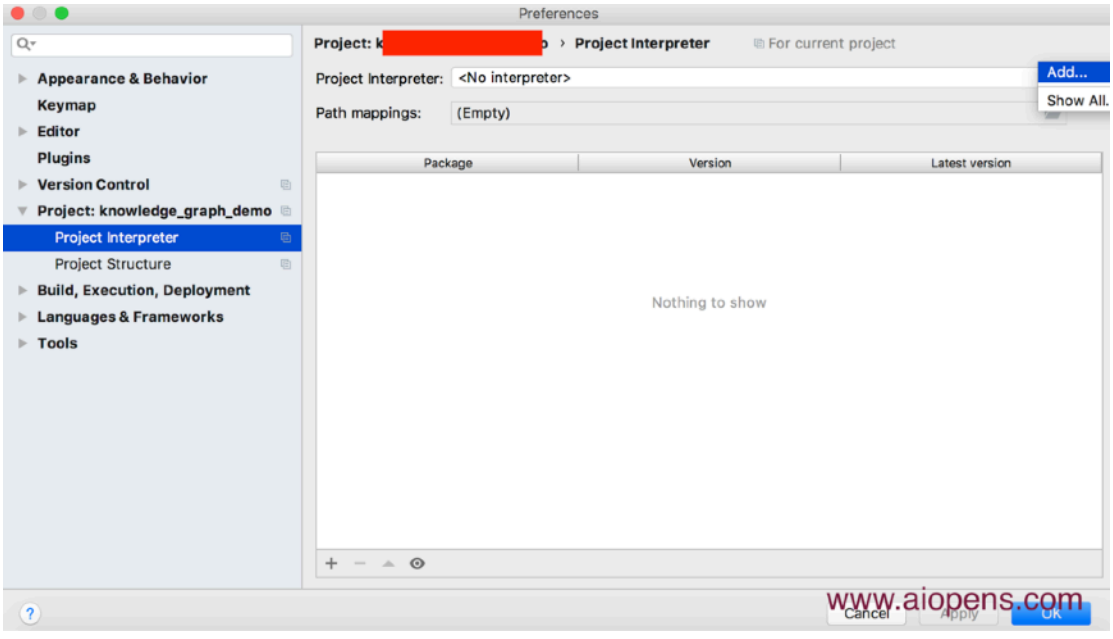
配置完点击 Test onnection, 如果成功就恭喜你，可以进行下一步了。

最后在 Mappings 中配置路径，这里的路径是你本地存放代码的路径，与刚刚配置的 Root Path 相互映射（意思是 Mapping 里本机的路径映射到远程的 Root Path），方便以后在本地和远程 docker 中进行代码和其他文件同步。

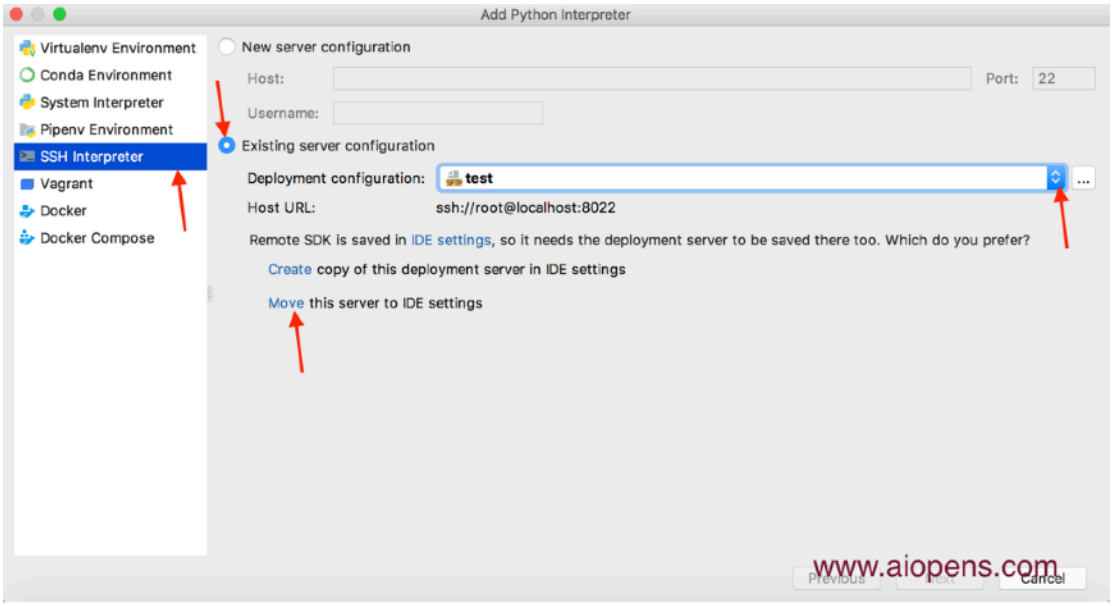


4. 在 PyCharm 里配置远程解释器

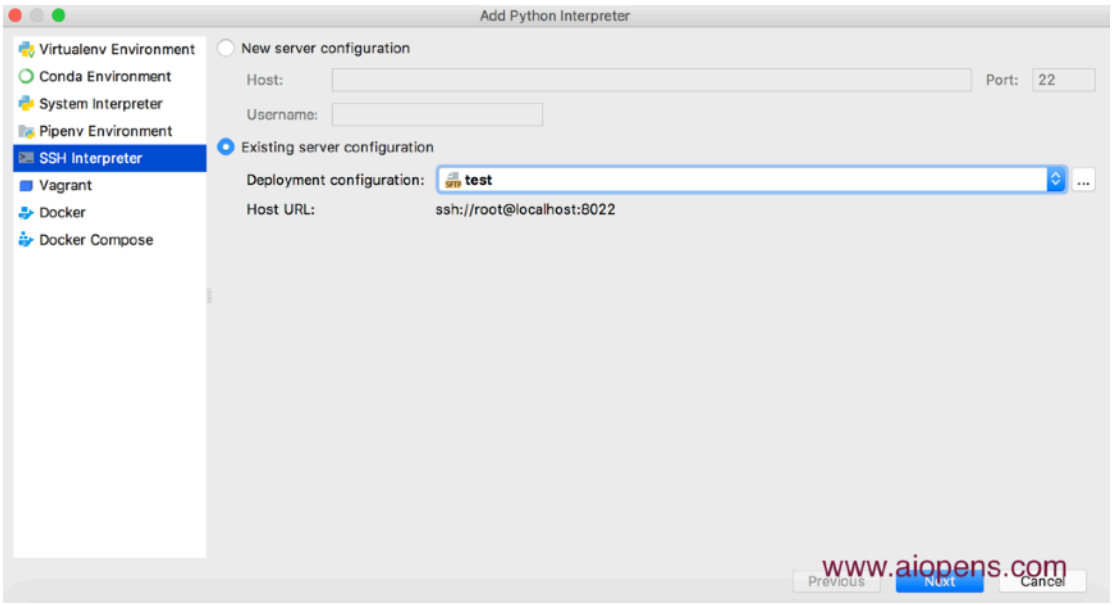
点击 Pycharm > Preferences > Project Interpreter ，右边的设置按钮add，新建一个项目的远程解释器:



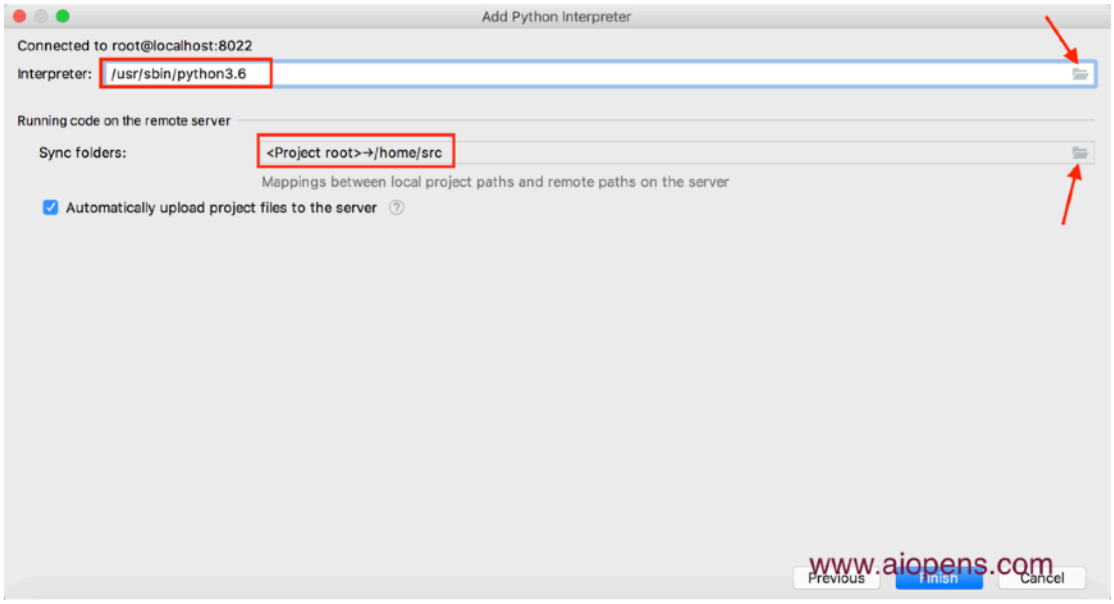
点击SSH Interpreter > Existing server configuration, 在选择框中选test，再选择 Move this server to IDE settings



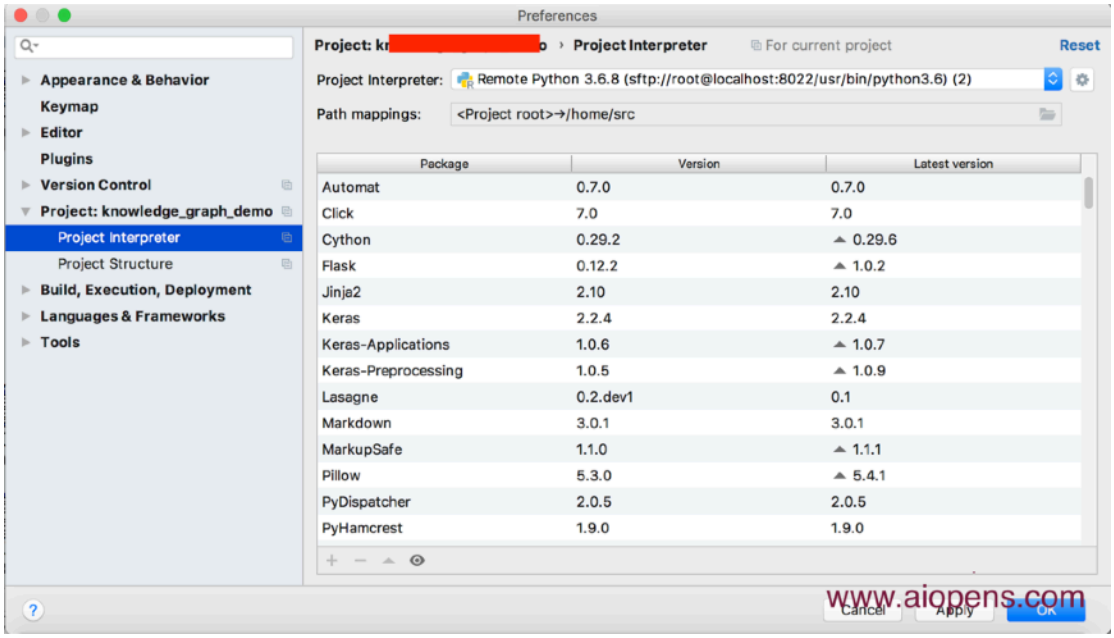
出现下面画面，然后点击Next:



通过上面箭头位置，选择需要使用的 python，配置 Interpreter；通过下部箭头位置，选择同步的文件夹：

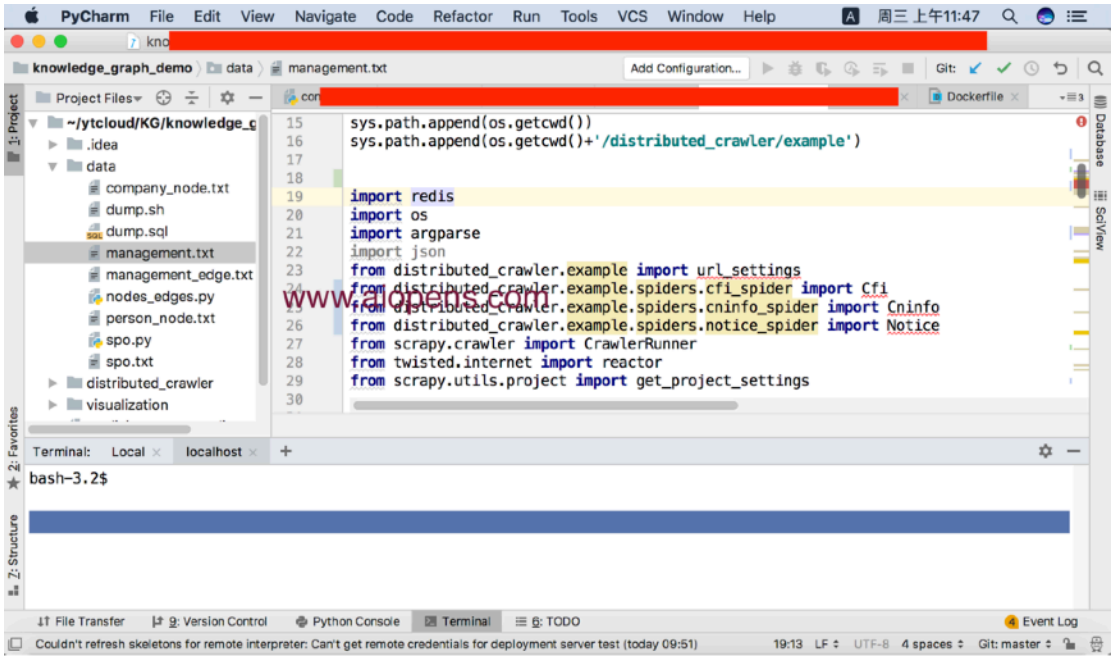


配置完成以后在项目解释器界面就会出现如下图所示，可以看到此时已经完成远程解释器的本地化，然后Apply > OK：



配置完成以后需要等本地和远程的环境同步一下，到这里，恭喜你，可以用最舒服的姿势。。。写代码了。

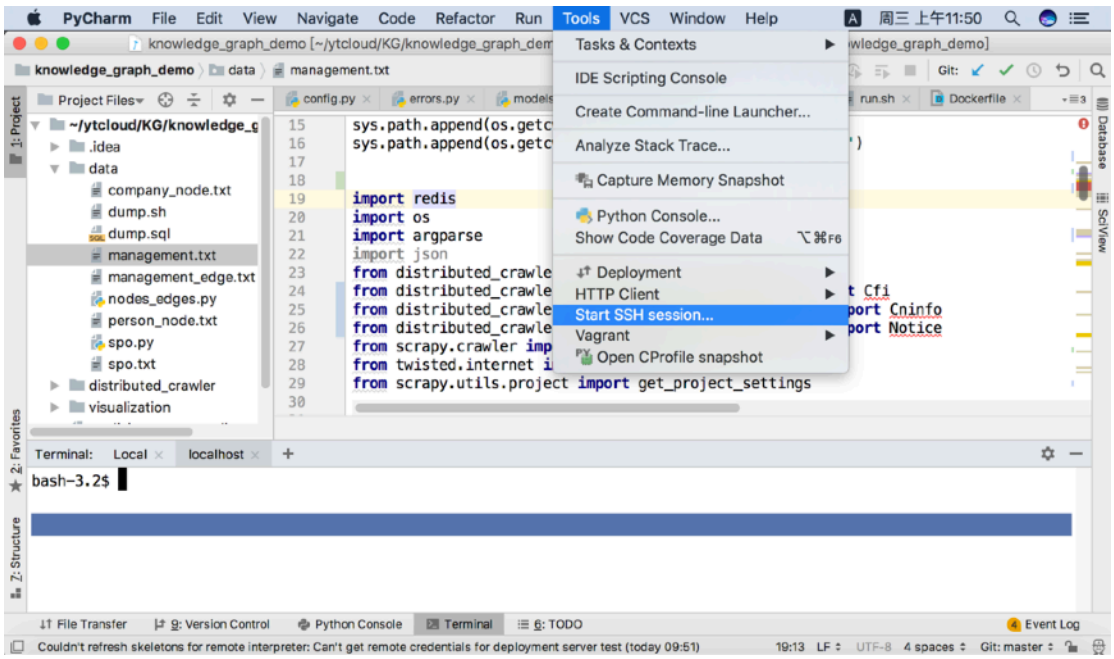
配置完成以后的日常是这样的：



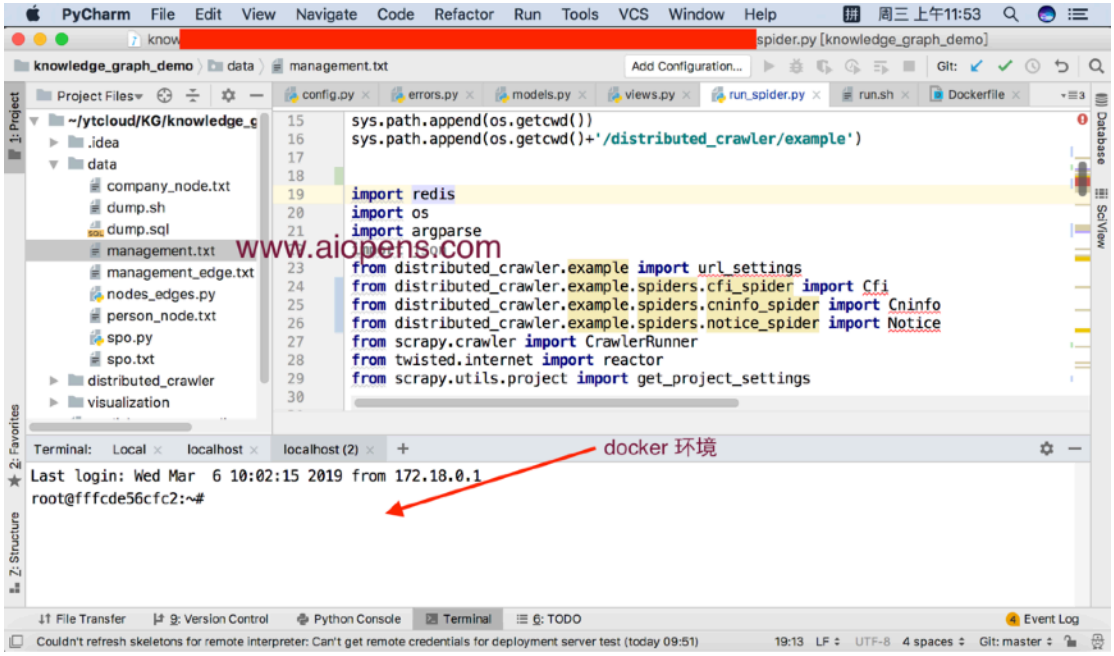
左边是本地的文件，修改之后可以随时右键 deployment->upload 到远程主机，或者直接在本地调试运行；最右边是远程主机的文件，假如直接在远程修改了文件刷新一下同样可以右键下载到本地，但是我不建议这样做，因为这样很容易带来冲突（毕竟没有很好的版本控制）。目前最好的实践是在局域网的服务器上，时延低，同步速度快。

5. 在 PyCharm 里访问 Docker 环境

点击 Tools > Start SSH session，选名为 test 的服务器：



可以在 Docker 环境中执行命令：



6. 在 **PyCharm** 里使用远程 **Docker** 环境中的 **Python**

点击底端的 Python Console，可以连接到名为test服务器中 docker 的 python