

开源知识图谱框架 SmartKG 安装部署说明书

1. SmartKG 简介

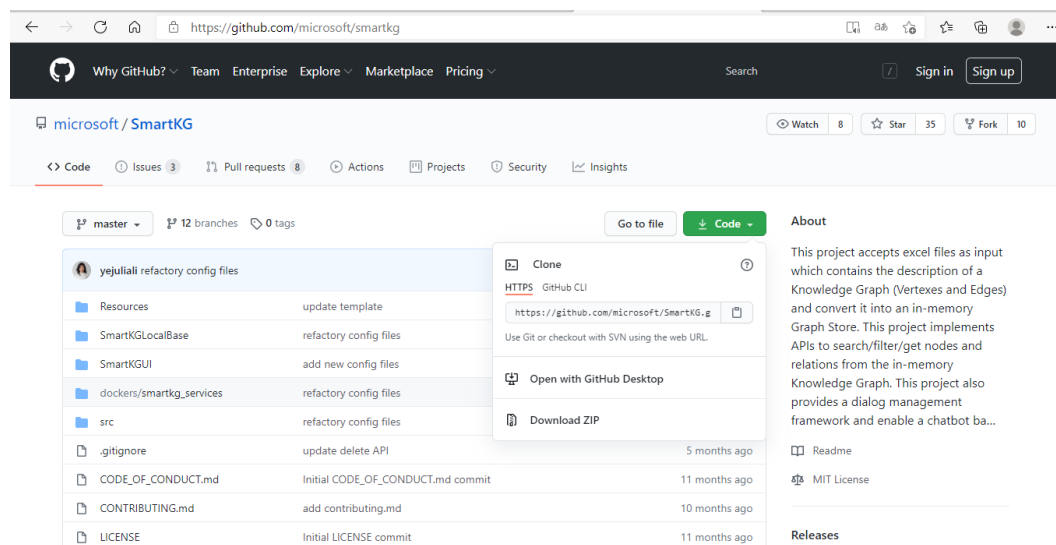
SmartKG 是一款轻量级知识图谱可视化+智能对话框架。它能够根据用户输入的实体和关系数据自动生成知识图谱，并提供图谱可视化及基于图谱的智能对话机器人。

2. 下载 SmartKG 代码库

2.1 Git Clone SmartKG Repo (Repository)

SmartKG 在 GitHub 上的地址为 <https://github.com/microsoft/smartkg>

首先，从 GitHub 上将 SmartKG 的代码库 pull 下来，点击 Code 按钮，选择复制 <https://github.com/microsoft/SmartKG.git>



随后，在 git 上，输入命令，git clone <https://github.com/microsoft/SmartKG.git>。如果没有 git，还需要提前进行安装，网站地址为 <https://git-scm.com>

```
$ git clone https://github.com/microsoft/SmartKG.git
Cloning into 'SmartKG'...
remote: Enumerating objects: 230, done.
remote: Counting objects: 100% (230/230), done.
remote: Compressing objects: 100% (144/144), done.
remote: Total 1935 (delta 122), reused 161 (delta 76), pack-reused 1705
Receiving objects: 100% (1935/1935), 486.61 MiB | 10.29 MiB/s, done.
Resolving deltas: 100% (1074/1074), done.
Updating files: 100% (198/198), done.
```



2.2 文档目录结构

在目录 src 下面, 是 SmartKG 后端服务的源代码。这部分源代码是基于 Asp.NET 框架, 用 C# 开发的。

在目录 SmartKGUI 下面, 是 SmartKG UI 的源代码。是基于 Node.js, 用 JavaScript 开发的。

在目录 SmartKGLocalBase 里, 是 SmartKG 后端服务会调用的一些 Python 文件和用于存储运行时数据的本地文件的目录。

在目录 Resources 里, 有用户上传数据的模板和用来做测试的图谱数据。其中, template 子目录中是模板, 用户如果要创建自己的知识图谱, 就需要按照模板的格式要求, 填入相应实体和实体关系。Input 子目录内有包括西游记、红楼梦、中学物理课以及 COVID19 等数据。

 input	2021/2/2 11:32	文件夹
 template	2021/2/2 11:32	文件夹

在目录 dockers 里, 是已经编译好前后端服务的二进制码、配置文件, 以及对应的 docker image。

3. 在本地直接启动 SmartKG 前后端 (Windows OS)

3.1 运行环境

如果想在本地运行 SmartKG 后端服务, 需要安装:

- 1) .NET Core 2.1 运行时环境
- 2) Python 3 的运行时环境, 推荐版本为 3.7







Python 的 xlrd 支持库 1.2.0 版本

注意: xlrd 新版不再支持处理xlsx 文档, 因此必须要 1.2.0 版本, 所用命令为: `pip install xlrd==1.2.0`





3) 如果要启动 SmartKG UI, 需要安装 Node.js , 推荐版本为 14.15.4。

3.2 启动前准备

在 Windows 环境里启动 SmartKG, 首先应该新建一个目录, 例如, 我们创建一个名为 temp 的目录。然后, 从本地 Repo 中 dockers 目录内的 smartkg 子目录中, 将两个 zip 文件和 local_config 文件夹复制到 temp 文件夹中。

 local_config	2021/2/2 11:32	文件夹	
 appsettings.json	2021/2/2 11:32	JSON 文件	2 KB
 Dockerfile	2021/2/2 11:32	文件	1 KB
 requirements.txt	2021/2/2 11:32	文本文档	1 KB
 smartkg.zip	2021/2/2 11:32	压缩(zipped)文件...	10,778 KB
 SmartKGLocalBase.zip	2021/2/2 11:32	压缩(zipped)文件...	18 KB

此外, 还需要从 dockers 目录的 ui 子目录中, 将唯一一个 zip 文件和 local_config 文件夹复制到 temp 文件夹中。

 local_config	2021/2/2 11:32	文件夹	
 config.js	2021/2/2 11:32	JavaScript 文件	1 KB
 Dockerfile	2021/2/2 11:32	文件	1 KB
 smartkgui.zip	2021/2/2 11:32	压缩(zipped)文件...	180 KB



3.3 SmartKG 的配置

在 local_config 中有三个文件, 其中两个以 appsettings 开头, 即 appsettings.File.json 和 appsettings.MongoDB.json。

3.3.1 appsettings

这两个都是 SmartKG 后端服务的配置文件, 它们代表了两种不同的配置, 具体到运行时, 选取其中之一即可。

用户使用哪种配置文件, 将决定最终生成的知识图谱以何种形式进行存储, 选择 appsettings.File.json 是存放在本地, 而选择 appsettings.MongoDB.json 则是存储在 MongoDB 中。

 appsettings.File.json	2021/2/2 11:32	JSON 文件	2 KB
 appsettings.MongoDB.json	2021/2/2 11:32	JSON 文件	2 KB

我们先来对比一下这两个目录的不同:

首先, 两者的 PersistenceType 不同, 一个是 File:

```
"PersistenceType": "File"
```

一个是 MongoDB:

```
"PersistenceType": "MongoDB"
```

File 文件中还多了 FileDataPath, 这个参数指明了最终生成的知识图谱是以本文文件的形式存储在对应的 RootPath 目录下的。

在下面的例子里, 我们使用的是 SmartKGLocalBase 下的 DataStores, 如果不使用这个目录也没关系, 只要是在用户中真实存在的目录就可以。在 ContextFilePath 中, 如果没有 kgbot_context.json 文件, 也可以生成一个空白文件, 放在该目录下。

```
"FileDataPath": {
  "RootPath":
"C:\\Users\\jull\\source\\github\\SmartKG\\SmartKGLocalBase\\DataStores",
  "ContextFilePath":
"C:\\Users\\jull\\source\\github\\SmartKG\\SmartKGLocalBase\\kgbot_context.json"
}
```

在 MongoDB 文件中, 没有 FileDataPath 而是有 ConnectionStrings。在 MongoClientConnection 中, 需要链接一个真实的 MongoDB 地址, 这个 MongoDB 在本地或者远程都可以, 只要能联通即可。

```
"ConnectionStrings": {
  "MongoDbConnection": "mongodb://{username}:{password}@{ip}:{port}",
  "DataStoreMgmtDatabaseName": "DataStoreMgmt",
  "ContextDatabaseName": "KGBot_Context"
}
```

除了以上的不同外, 两份文档的其余内容一样。

在 File 和 MongoDB 文件中都有 FileUploadConfig, 它指向的是 SmartKGLocalBase, 这里面存放了 Python 脚本, 用于进行部分生成知识图谱过程中的数据处理工作:

```
"FileUploadConfig": {
  "PythonEnvPath": "python",
```

```
    "ConvertScriptPath":  
    "..\\SmartKGLocalBase\\scripts\\ProcessUploadedFiles.py",  
    "ColorConfigPath": "..\\SmartKGLocalBase\\scripts\\config",  
    "ExcelDir": "..\\SmartKGLocalBase\\temp",  
    "LocalRootPath": "..\\SmartKGLocalBase\\DataStores"  
}
```

这部分设置可以采用相对路径，也可以采用绝对路径。

3.3.2 config.js

local_config 中的 config.js 是用于 UI 的配置文件。

3.4 本地启动 SmartKG

【1】把 temp 目录中的三个压缩文件直接就地解压缩（Extraction Here）。

【2】并将 temp/local_config 中的 config.js 移动到解压缩后的 temp/smartkgui/public 中。

【3】将 local_config 的 appsettings.File.json 文件复制一份，并改名为 appsettings.json，移动到 temp/smartkg 中。

【4】命令行进入 temp/smartkg，运行命令：

```
dotnet SmartKG.KGBot.dll
```

此命令用于启动 SmartKG 后端。启动后，会生成一个 Now listening on 地址，我们直接访问地址就可以。在浏览器中输入地址（例如：<http://localhost:5000>），可以访问后，说明后台启动成功了。

【5】进入 temp/smartkgui 目录下，输入命令：

```
npm i
```

这个命令只需要第一次使用时运行，如果已经运行过，就可以跳过了。此命令运行成功后，再输入命令：

```
npm run serve
```

运行成功后，会给一个访问地址，例如：<http://localhost:8080/>

用这个地址就可以直接在浏览器中访问 SmartKG 了。

4. 启动 SmartKG 的 Docker-compose OneBox (On Linux)

4.1 运行环境

在 Linux 环境部署前，需要提前安装好 Docker 和 Docker Compose。

4.2 启动 OneBox

【1】打开目录 dockers，将里面的 smartkg_services 目录整体压缩，并拷贝到 Linux 机器上，一般放在用户目录下。

【2】在 Linux 系统进入用户目录，并解压缩 smartkg_services.zip。

【3】进入 解压缩后的 smartkg_services 目录，运行命令：

```
sudo docker-compose build --build-arg DOCKER_HOST=${docker_host_ip}
```

其中，`${docker_host_ip}` 是当前 linux 环境对外的 IP 地址。

【4】Build 结束后，同一位置，运行命令：

```
sudo docker-compose up
```

启动过程可能会有些慢，因为它会在本地启动 MongoDB 的 instance，以及启动 Node.js

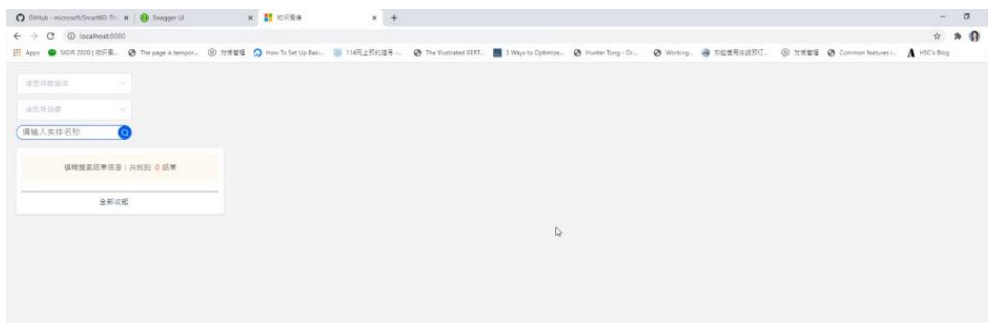
【5】等到所有服务全部正常启动后，在 Browser 中输入 [http://\\${docker_host_ip}:8083](http://${docker_host_ip}:8083)，访问 SmartKG。

5. 使用 SmartKG

5.1 访问主界面

通过浏览器访问 SmartKG （地址为 <http://localhost:8080/> 或者 [http://\\${docker_host_ip}:8083](http://${docker_host_ip}:8083)）。

正常访问会出现图谱的展示页面：



5.2 导入数据

在浏览器的访问地址后加上后缀 “/upload”，地址称为 <http://localhost:8080/upload> 或者 [http://\\${docker_host_ip}:8083/upload](http://${docker_host_ip}:8083/upload) ，进入图谱数据导入页面。



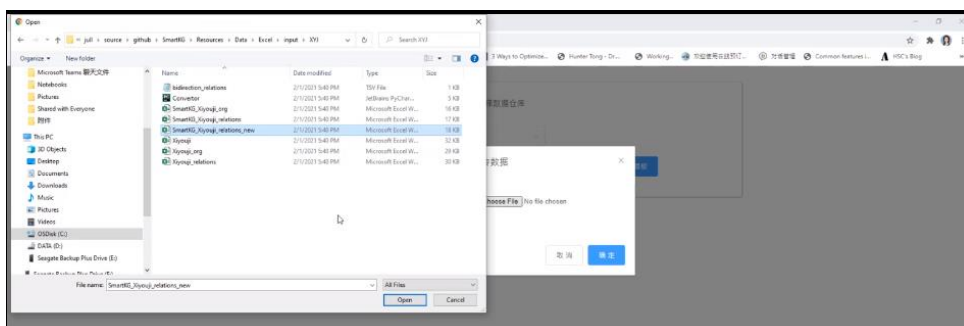
进入页面后，选择新建数据仓库，例如 Test1。



然后，选择新建的数据仓库，并点击上传数据。



上传数据，可以使用从 Repo 里的 Resources/Data/Excel/input/XYJ/ 路径下的 SmartKG_Xiyouji_relations_new.xlsx 文件，这份文件中的数据比较全面。



数据选定后，还需要起一个场景名，名字可以为任何字符串，例如，Background。

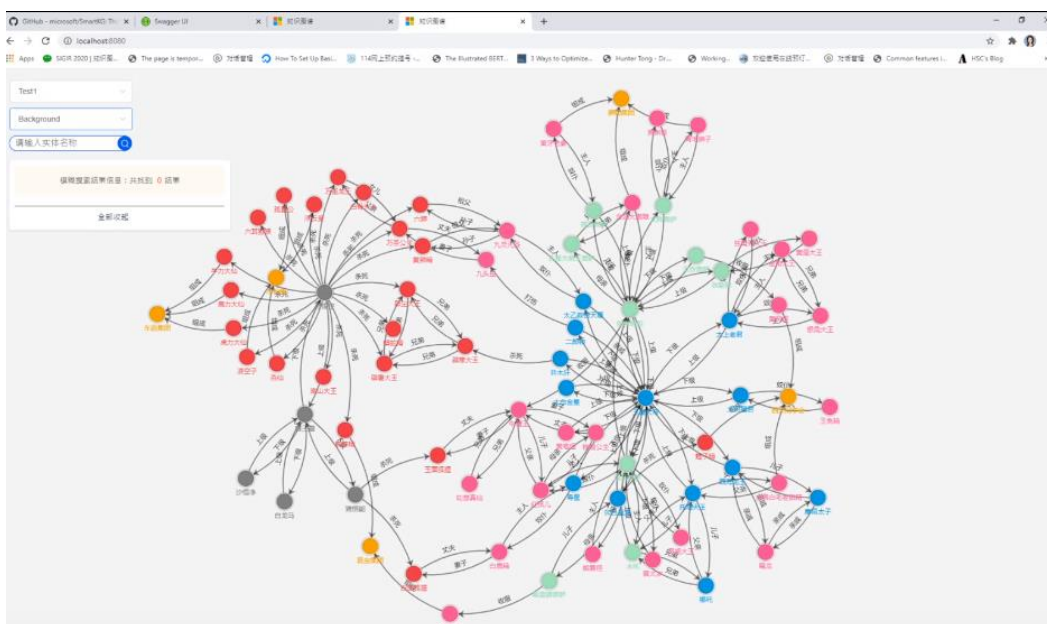


然后点击确定上传图谱文件。

5.3 知识图谱可视化

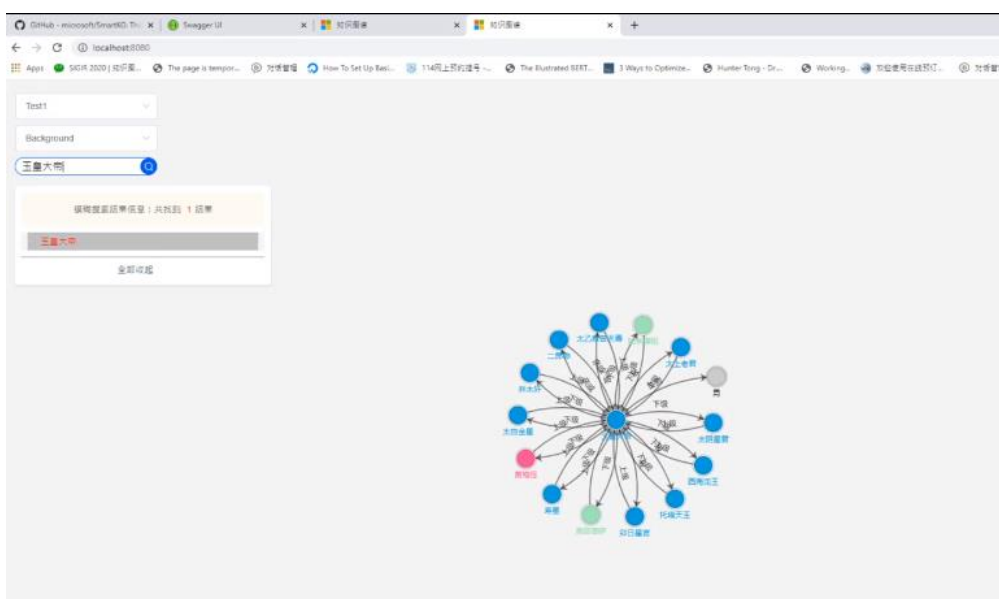
上传成功后，就可以回到 <http://localhost:8080/> 或者 [http://\\${docker_host_ip}:8083](http://${docker_host_ip}:8083) 页面了。

在左上角的两个下拉菜单中分别 选择 Test1 和 Background，就可以展示出完整的知识图谱了。



5.4 知识图谱查询

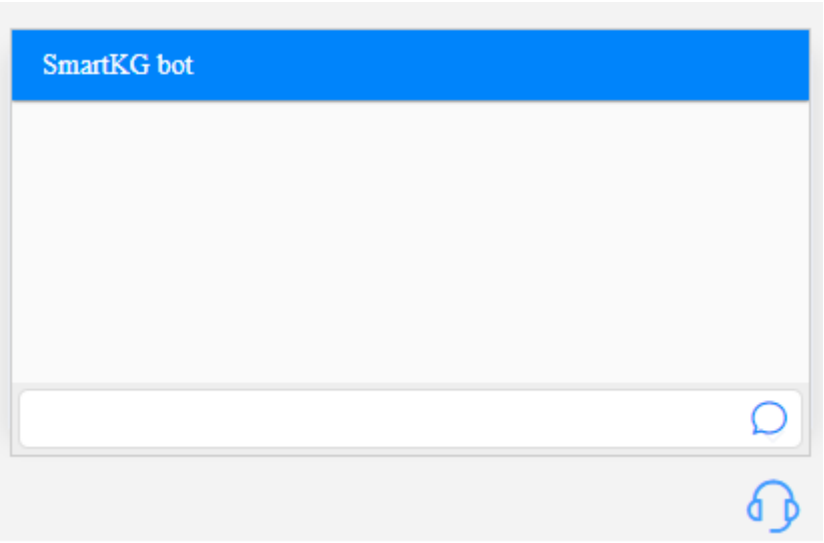
知识图谱生成后，用户可以进行查询。例如在左侧文本框中输入“玉皇大帝”，就可以看到“玉皇大帝”这一实体和与其具有直接关联的其他若干实体的可视化显示。



5.5 基于知识图谱的聊天机器人

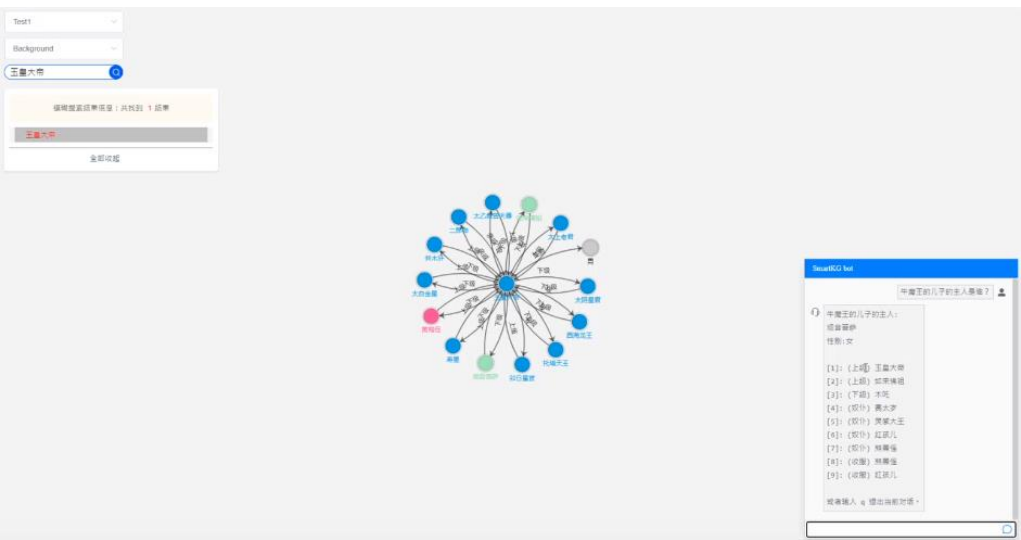
一个知识图谱生成后，SmartKG 会自动创建一个基于该知识图谱的对话机器人，用户可与对话机器人进行有关图谱知识的问答。

点击图谱展示界面右下角的耳机形图标，即可打开对话机器人测试窗口。



在测试窗口中输入问题就可以提问题了。

例如，我们输入：“牛魔王的儿子的主人是谁？”，机器人给出结果：“观音菩萨”。



除了显示对观音菩萨的解释，还会显示出与其直接相关的其他人物的提示信息。如果想查看其中某个人物，可以输入对应的数字。

如果在对话中想退出，可以输入“q”。

5.6 设置节点颜色

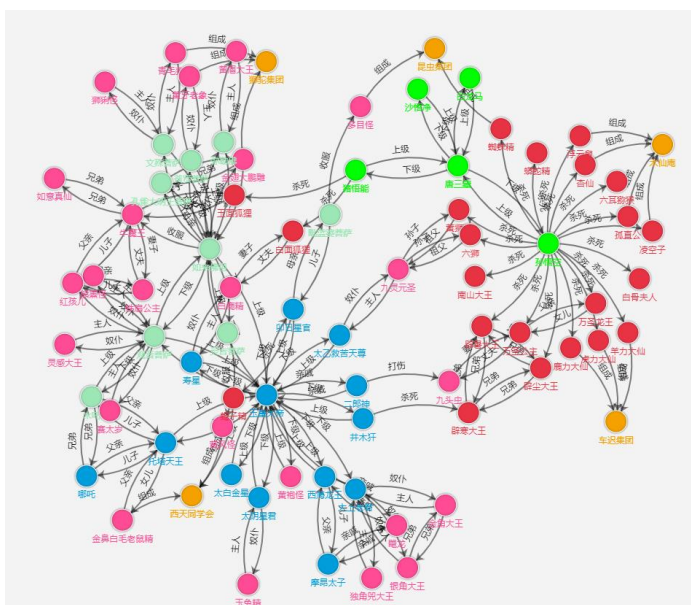
在 SmartKG 的上传数据界面，除了导入数据，还可以设置节点的颜色。

我们首先点击设置颜色按钮，进入到设置颜色窗口后，选择数据库和场景，然后该数据库内该场景中的所有节点类型都会显示出来：



设置颜色时，需要将选定颜色的代码，复制到指定位置。例如，选择高亮的绿色，然后将其复制到“和尚”对应的颜色框中。修改后保存。

回到图谱展示界面后，需要重新选择，就可以看到颜色变化了。



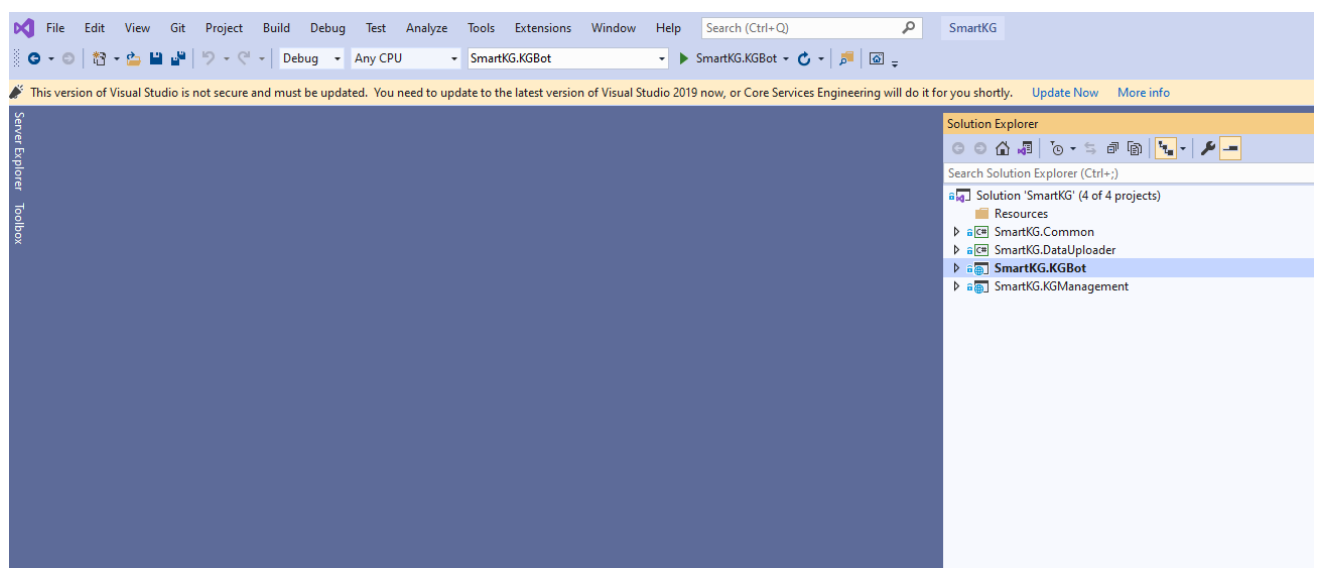
6. 编辑、编译源代码

6.1 运行环境

编译源代码，需要提前安装 Visual Studio 2019

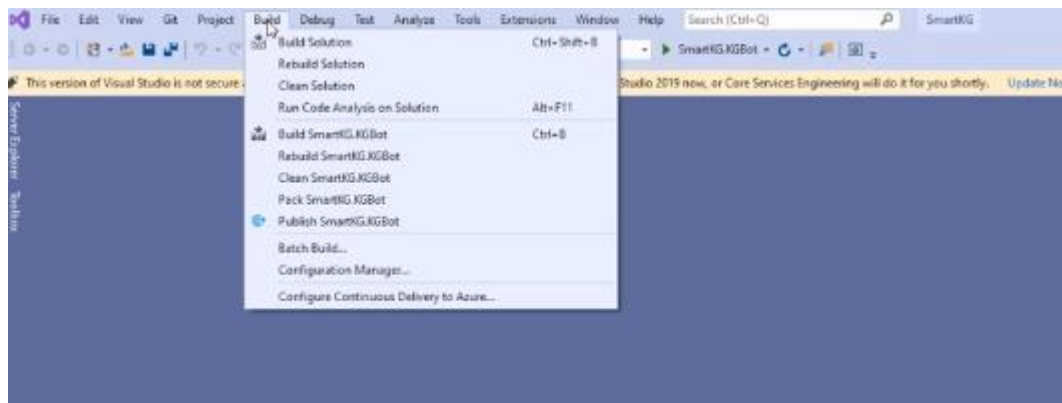
6.2 启动 VS，展示源代码

回到 Repo 目录，进入 src 文件夹，启动 SmartKG.sln。



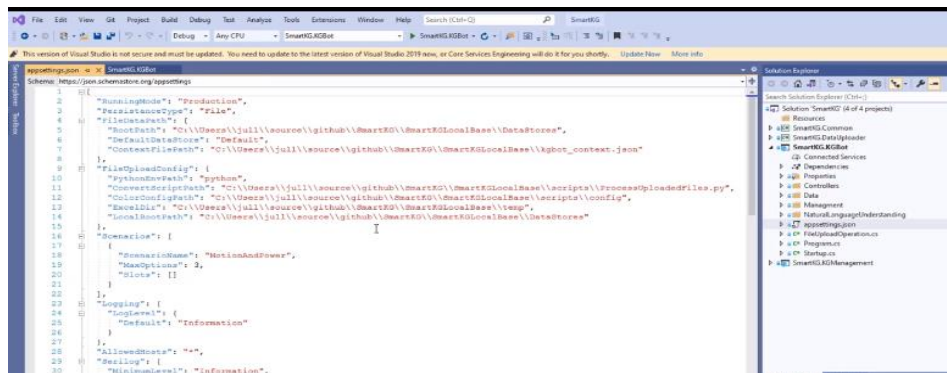
6.3 Build Solution

进入到 VS 后，点击 Build 的 Build Solution，编译源代码

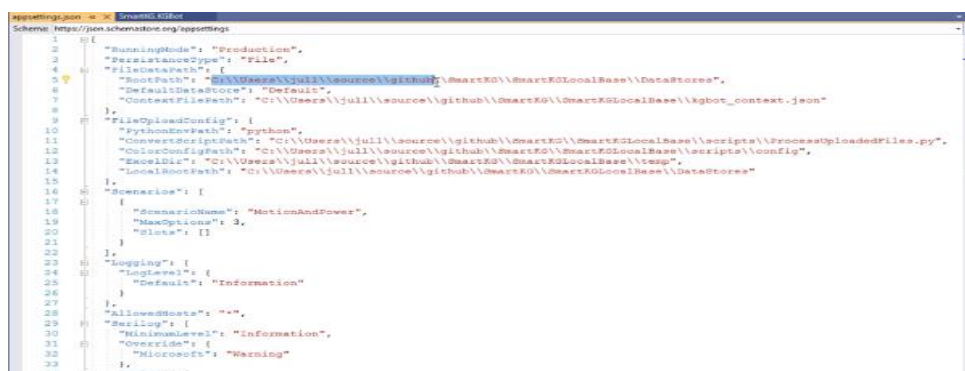


6.4 本机调试

如果在本地做测试，会用到当前 source 目录下的 SmartKG.KGBot 下的 appsettings.json。

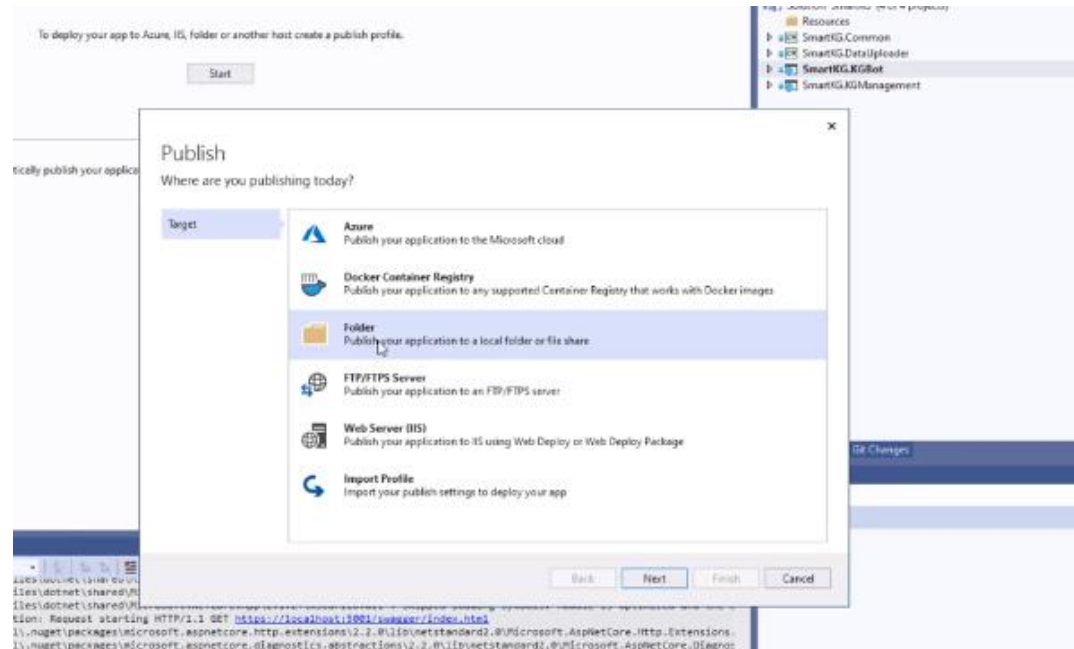


在调试时，需注意“RootPath”使用的是绝对路径，需要根据用户自己的实际情况，进行修改。此外，其他涉及到文件路径的代码，都要根据用户的实际情况进行修改。修改完地址后，就可以点击运行。

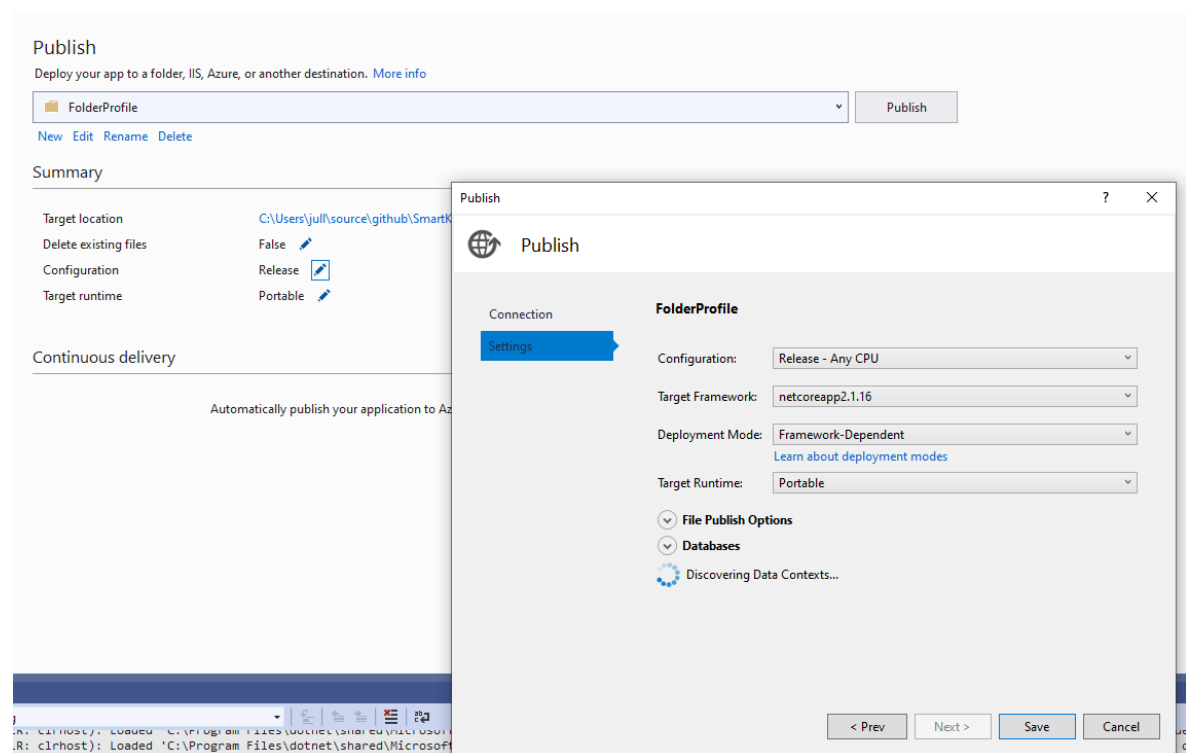


6.5 Publish 源代码

右键点击 SmartKG.KGBot project，选择点击 Publish，接着点击 Folder，以及 Next：



确认编译选项：



“Save”后 点击“Publish”，进行发布。编译后的文件地址，会在 Output 中显示出来。

Output

Show output from: Build

```
2>C:\Users\jull\source\github\SmartKG\src\SmartKG\KGManagement\bin\Release\netcoreapp2.1\SmartKG.KGManagement.dll
3>C:\Users\jull\source\github\SmartKG\src\SmartKG\KGManagement\Controllers\GraphController.cs(237,50,237,53): warning CS1998: This async method lacks 'await' operators
3>C:\Users\jull\source\github\SmartKG\src\SmartKG\KGManagement\Controllers\DataStoreGetController.cs(224,57,224,67): warning CS1998: This async method lacks 'await'
5>SmartKG.KGManagement -> C:\Users\jull\source\github\SmartKG\src\SmartKG\KGManagement\bin\Release\netcoreapp2.1.16\SmartKG.KGManagement.dll
3>Done building project "SmartKG.KGManagement.csproj".
4>----- Build started: Project: SmartKG.KGBot, Configuration: Release Any CPU -----
4>C:\Users\jull\source\github\SmartKG\src\SmartKG.KGBot\Management\DialogManager.cs(54,46,54,47): warning CS1998: This async method lacks 'await' operators and will run
4>SmartKG.KGBot -> C:\Users\jull\source\github\SmartKG\src\SmartKG.KGBot\bin\Release\netcoreapp2.1.16\SmartKG.KGBot.dll
4>Done building project "SmartKG.KGBot.csproj".
5>----- Publish started: Project: SmartKG.KGBot, Configuration: Release Any CPU -----
Connecting to C:\Users\jull\source\github\SmartKG\src\SmartKG.KGBot\bin\Release\netcoreapp2.1.16\publish\...
SmartKG.KGBot -> C:\Users\jull\source\github\SmartKG\src\SmartKG.KGBot\bin\Release\netcoreapp2.1.16\SmartKG.KGBot.dll
SmartKG.KGBot -> C:\Users\jull\source\github\SmartKG\src\SmartKG.KGBot\obj\Release\netcoreapp2.1.16\PubTmp\Out\
Web App was published successfully http://c:\Users\jull\source\github\SmartKG\src\SmartKG.KGBot\bin\Release\netcoreapp2.1.16\publish/

***** Build: 4 succeeded, 0 failed, 0 up-to-date, 0 skipped *****
***** Publish: 1 succeeded, 0 failed, 0 skipped *****
```