Assignment 1: RESTful services, Docker and Kubernetes

Link of the document: https://shorturl.at/jvS79

1 Introduction

In this tutorial we will use OpenAPI to define a RESTful web service and Python to implement it. The RESTful web service will be using a database to store data.
More specifically the steps of this tutorial are the following:

1. Write OpenAPI definition using swaggerhub
2. Generate the server stubs in Python and implement the logic
3. Build Test and Publish Docker Image
4. Write Tests
5. Deploy Web Service on Kubernetes (microk8s)

2 Reporting and assessment

2.1 Reporting

At the end of this assignment you are expected to submit the following:

- Each student should write a short report (max 3 pages) with the following structure:
    - **Introduction**: List which of the DevOps stages you practiced with this assignment and what are their primary objectives.
    - **OpenAPI Exercises**: Report on the following exercises (for details see on the Section OpenAPI Exercises):
    - Define Objects
    - Add Delete method
    - **Docker Exercises (Optional)**: If you choose to integrate an external database report on how you modified your code and Dockerfile to connect to DB.
    - **Question 1**: In the Docker file we use the 'python:3.6-alpine' image as a base image. What are the advantage and advantages of using the Alpine Linux distribution in Docker.
    - **Question 2**: Based on the use case discussion during the lecture, please discuss the advantages and risks of applying agile approach.

2.2 Assessment

IMPORTANT
In order to be given a grade you must submit the following:

- Written report (see above for details)
- Name of the published docker(s) in https://hub.docker.com/. Must be able to perform (docker pull)
- Git repository link
- If you choose the extra task i.e. the external database integration submit **only** the code using the external database.
- If you choose the extra task i.e. the external database integration the testing will be done using the 'docker-compose.yaml' provided below

Do not add your code when submitting in Canvas.
All links such as DockerHub and Git must be accessible from the day of submission and onwards.

- If the source code and dockerized service pass all the tests defined in Newman you get 70%.
- A passing mark will be given if the source code and dockerized service pass tests 1-5 in Newman (see Section Write Tests).
- The tests will run on the docker created from source i.e. from building the docker file: `docker build .` and the docker image from your DockerHub account
- An extra point will be given for the Docker Exercises i.e. the external database integration
- The rest will be determined by your report.

3 Background

3.1 OpenAPI and Swagger

Swagger is an implementation of OpenAPI. Swagger contains a tool that helps developers design, build, document, and consume RESTful Web services. Applications implemented based on OpenAPI interface files can automatically generate documentation of methods, parameters, and models. This helps keep the documentation, client libraries, and source code in sync.

3.2 Git

Git is an open source distributed version control system. Version control helps keep track of changes in a project and allows for collaboration between many developers.

3.3 GitHub Actions

GitHub Actions automate your software development workflows from within GitHub. In GitHub Actions, a workflow is an automated process that you set up in your GitHub repository. You can build, test, package, release, or deploy any project on GitHub with a workflow.

3.4 Docker

Docker performs operating-system-level virtualization, also known as "containerization". Docker uses the resource isolation features of the Linux kernel to allow independent "containers" to run within a Linux instance.

4 Prepare your Development Environment

If you want you may use VirtualBox and Ubuntu 20.04 as your development environment.
You can find VirtualBox here: https://www.virtualbox.org/wiki/Downloads , and Ubuntu here: https://ubuntu.com/download/desktop

4.1 Create GitHub Account

If you don't have an account already follow these instructions: https://github.com/join

4.2 Setup Docker Hub

If you don't have an account already follow these instructions: https://hub.docker.com/signup

4.3 Swaggerhub Account

If you have a GitHub Account you may go to https://app.swaggerhub.com/login and select 'Log In with GitHub'. Alternatively, you can select to sign up.

4.4 Install Docker and Docker Compose on your Local machine

You can find instructions on how to install Docker here: https://docs.docker.com/get-docker/. You may also find a detailed tutorial on Docker here: https://docker-curriculum.com/.
To test if your installation is running you may test docker by typing:

```
docker run hello-world
```

You can find instructions on how to install Docker Compose here: https://docs.docker.com/compose/install/.

4.5 Install Pycharm

In this tutorial we will use the Pycharm Integrated Development Environment (IDE). If you have a preferred IDE you are free to use it.
You can find instructions on how to install Pycharm here: https://www.jetbrains.com/pycharm/download/

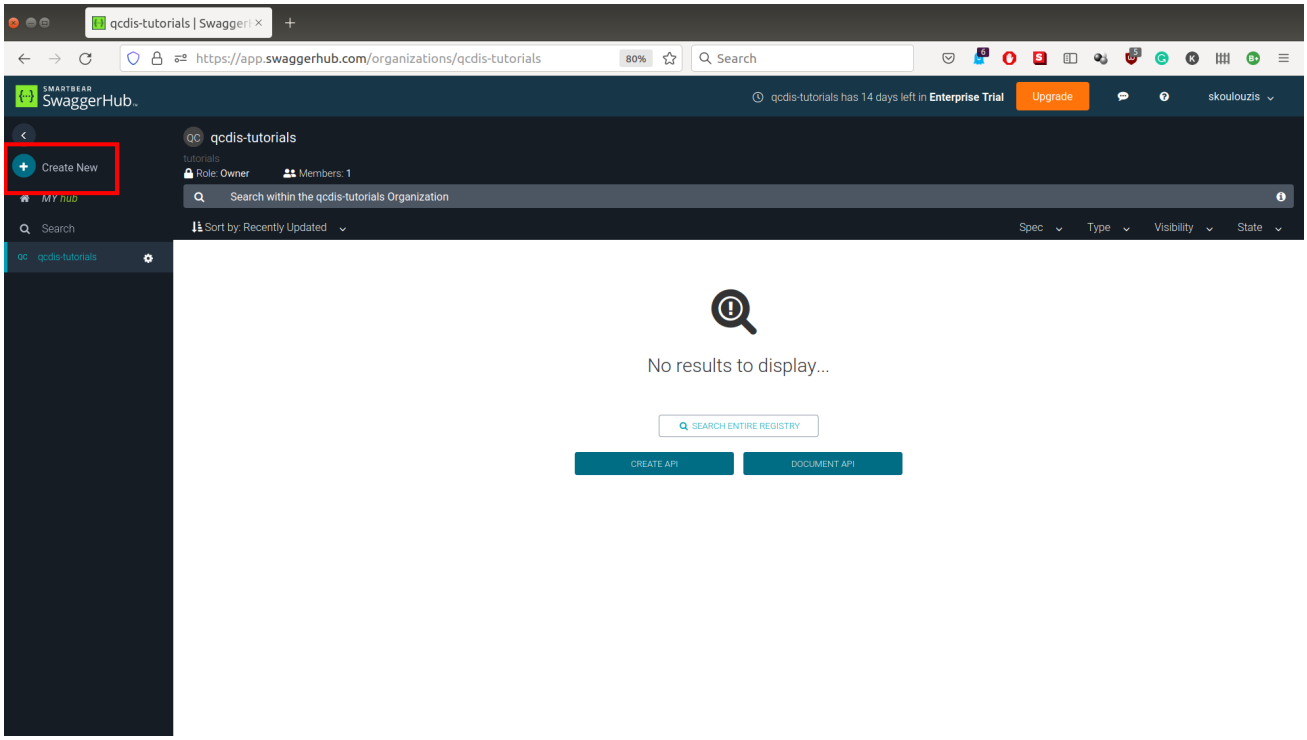
If you are using snap you can type:

```
sudo snap install pycharm-community --classic
```

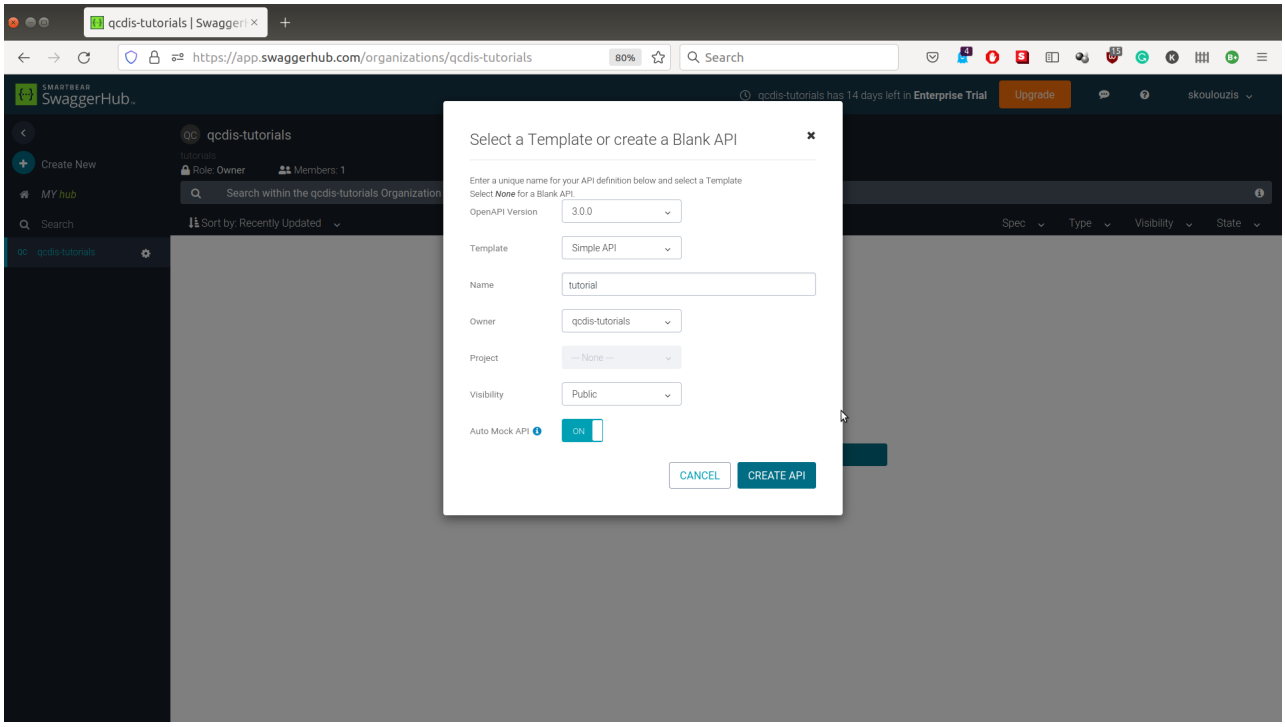
You may also find a detailed tutorial on Pycharm here:
https://www.jetbrains.com/help/pycharm/creating-and-running-your-first-python-project.html

5 Write OpenAPI Definition

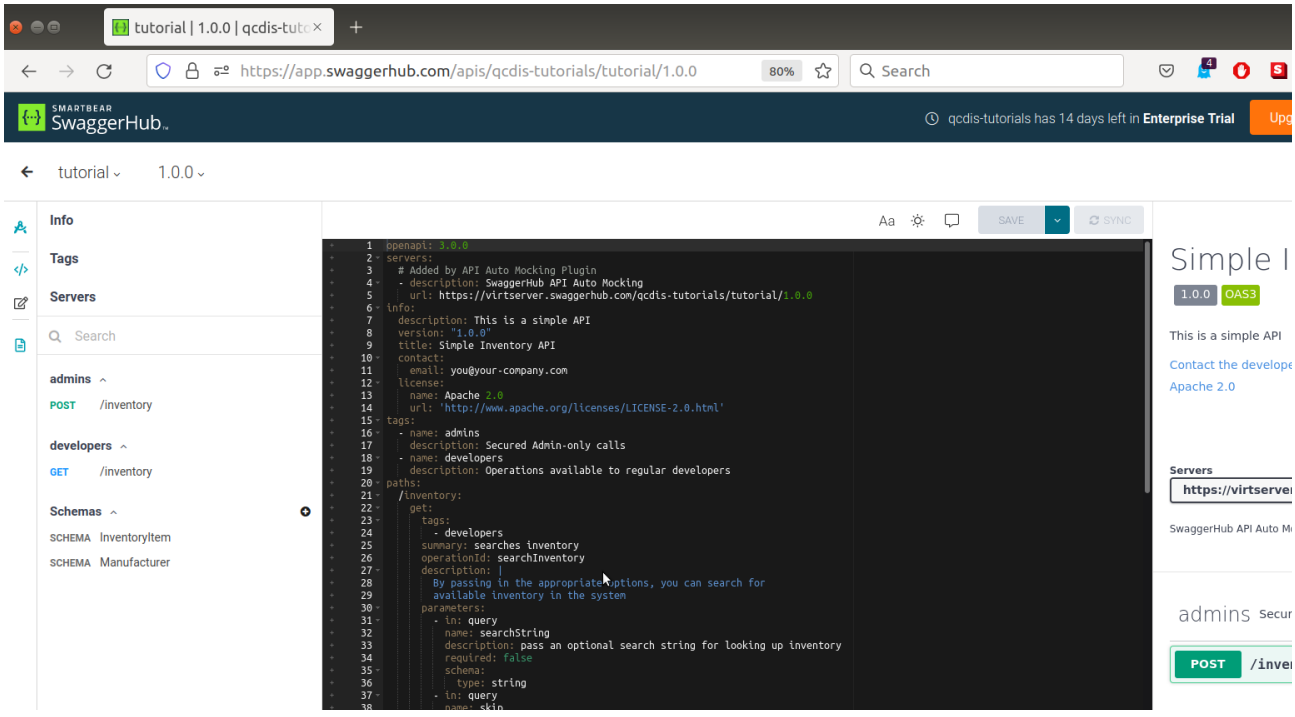
In this section we will define a web service interface that will support the Create, Read, Update, Delete (CRUD) pattern for manipulating resources using OpenAPI. To get a more in-depth understanding of Swagger and OpenAPI you may follow this tutorial
https://idratherbewriting.com/learnapidoc/openapi_tutorial.html
Log in to your Swaggerhub account at https://app.swaggerhub.com/login and select 'Create New' -> 'Create New API'



Then select version 3.0.0 and 'Template' 'Simple API' and press 'CREATE API'.



You will get a OpenAPI template



Name your API 'tutorial'.
Replace the definition with the following: openAPI_1.yaml

You will notice that the editor at the bottom throws some errors:

```
Errors (3)
Could not resolve reference: #/components/schemas/Student
$refs must reference a valid location in the document
$refs must reference a valid location in the document
```

Effectively what is said here is that the "#/components/schemas/Student" is not defined. You can find more about '$refs' here: https://swagger.io/docs/specification/using-ref/

5.1 OpenAPI Exercises

5.1.1 Define Objects

Scroll down to the bottom of the page and create a new node called 'components', a node 'schemas' under that and a node called 'Student' under that. The code should look like this:

```
components:
  schemas:
    Student:
      type: object
      required:
        - first_name
        - last_name
      properties:
        ...
    GradeRecord:
      type: object
      required:
        - subject_name
        - grade
      properties:
        …
```

Define the Student's and GradeRecord object properties.
The Student properties to define are:

Property Name	Type
student_id	number (integer format)
first_name	string
last_name	string
gradeRecords	array

The GradeRecord properties to define are:

Property Name	Type
subject_name	string
grade	number (float format, minimum: 0, maximum:

	10)

NOTE
Take not on which properties are required and which are not. This will affect the services' validation process.
To get more information on the 'required' see:https://swagger.io/docs/specification/data-models/data-types/ in the 'Required Properties' Section.

It is useful to add 'example' fields in the properties. That way you API is easier to consume. You can find details about the 'example' field here: https://swagger.io/docs/specification/adding-examples/ You can find details about data models here: https://swagger.io/docs/specification/data-models/

5.1.2 Add Delete method

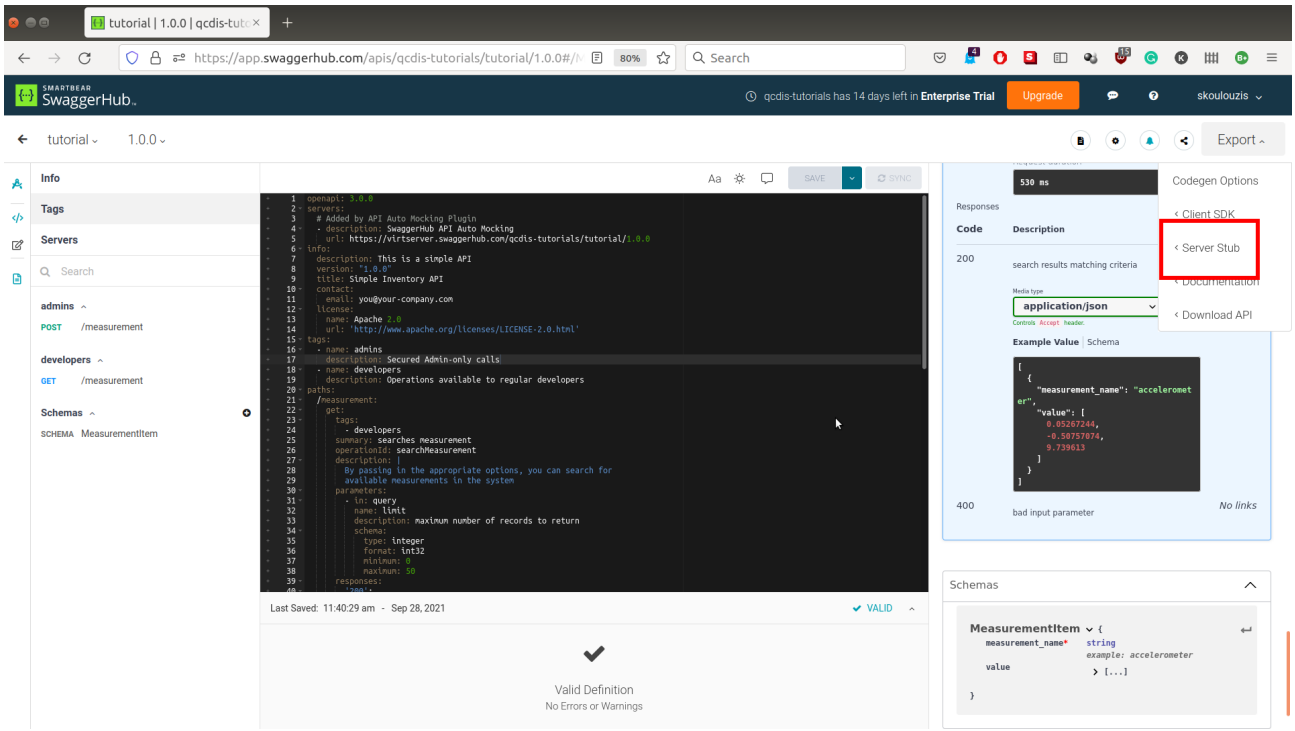
The API definition at the moment only has 'GET' and 'POST' methods. We will add a 'DELETE' method. Under the '/student/{student_id}' path add the following:

```
    delete:
      summary: gets student
      operationId: deleteStudent
      description: |
        delete a single student
      parameters:
        - in: path
          name: student_id
          description: the uid
          required: true
          schema:
            type: number
            format: integer
      responses:
        …
```

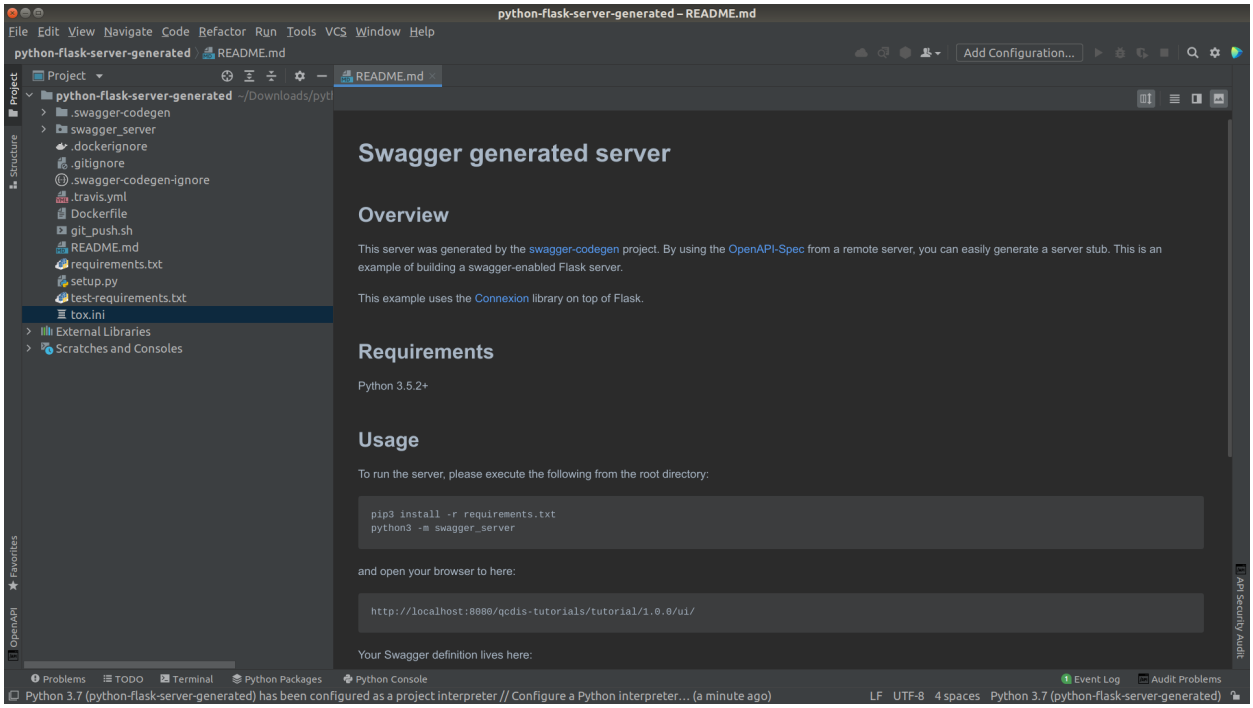
You will need to fill in the proper responses for 200, 400, and 404. More information about responses can be found here: https://swagger.io/docs/specification/describing-responses/
Now that we have the definition completed we can try some mockup calls. Press 'SAVE' and select the POST method and press execute. You should get an example of all the responses.

6 Generate the Server Stubs

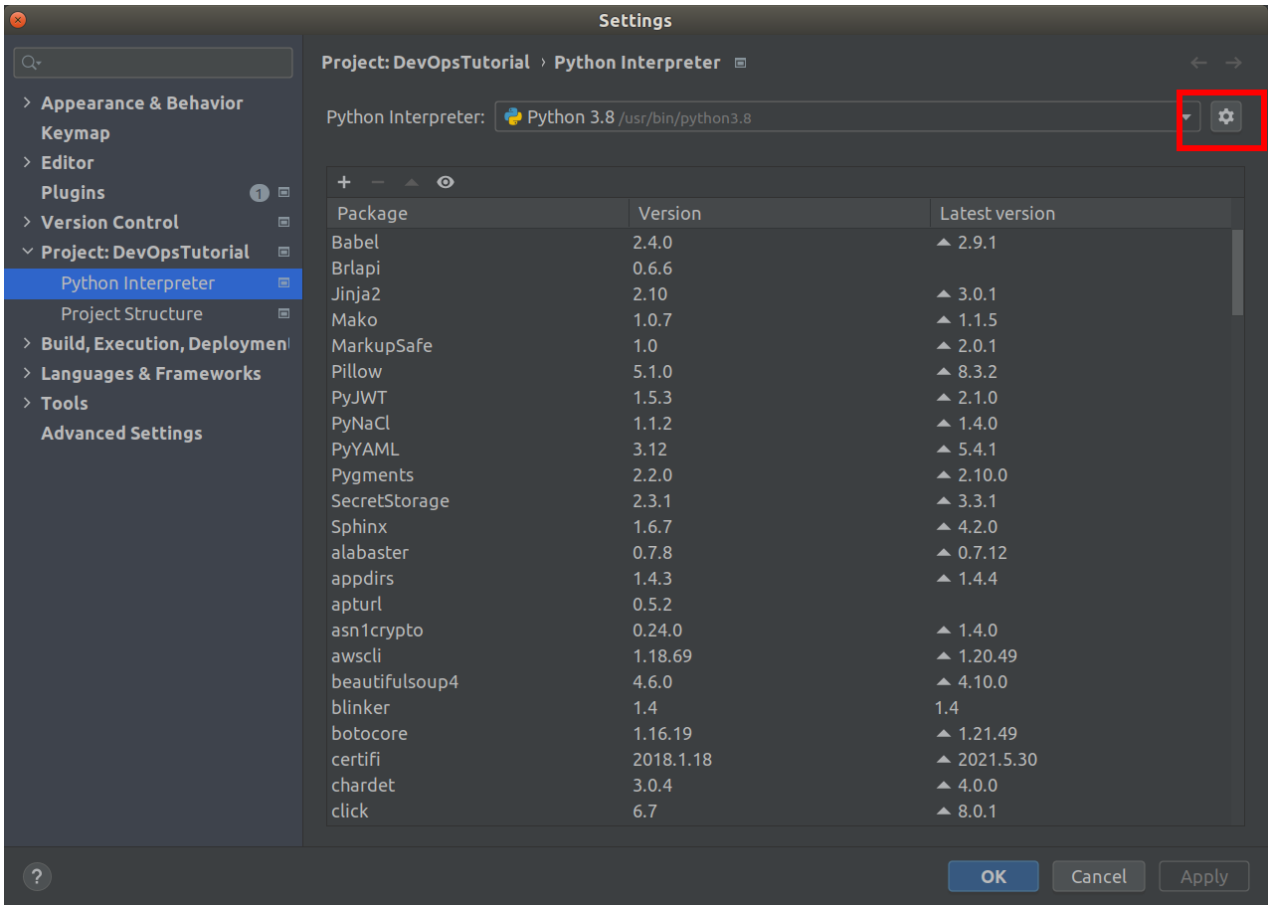
Now that we have the OpenAPI definitions we can create the server stub on python. Select 'Export'->'Server Stub'-> 'python-flask'



Save the 'python-flask-server-generated.zip' and unzip the archive. Open Pycharm and open the project by selecting 'File'->'Open' and selecting the unzipped folder.

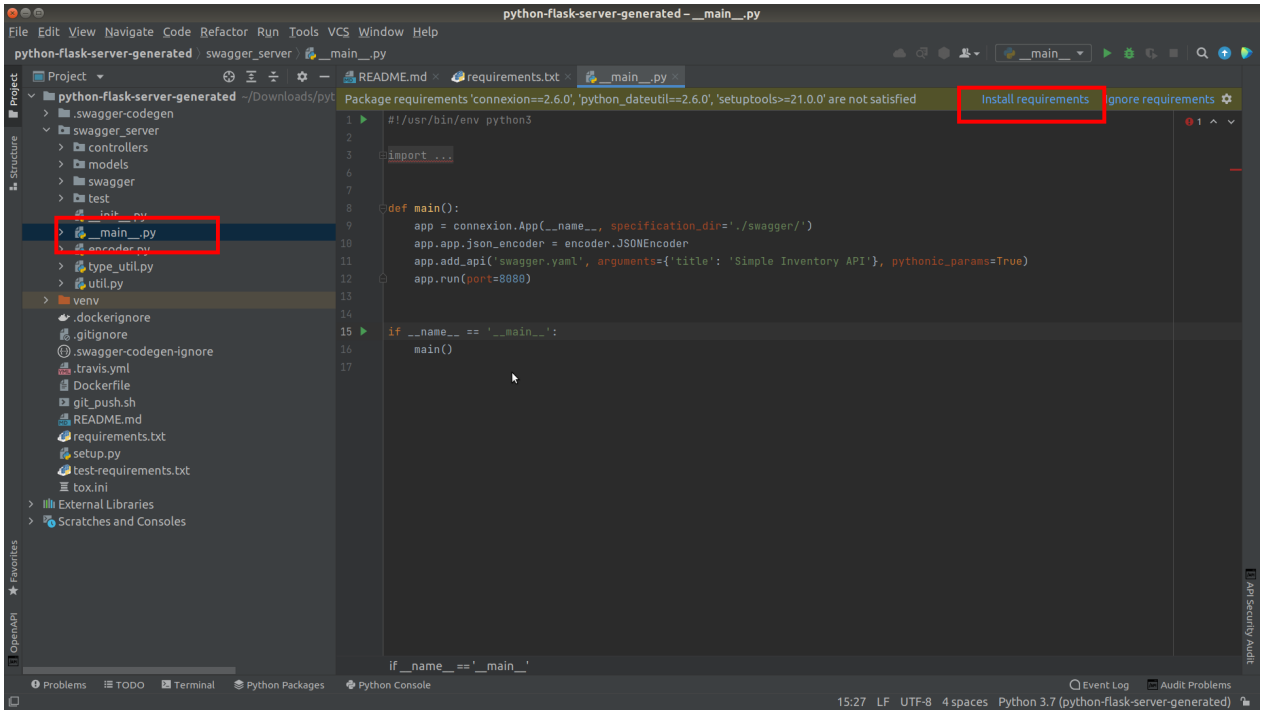


To create the virtual environment for the project go to 'File'->'Settings'->'Project'->'Python Interpreter' or 'Pycharm'->'Preferences'->Project'->'Python Interpreter'. Select Python version 3.8 or later. Then select the gear icon to add a new environment:

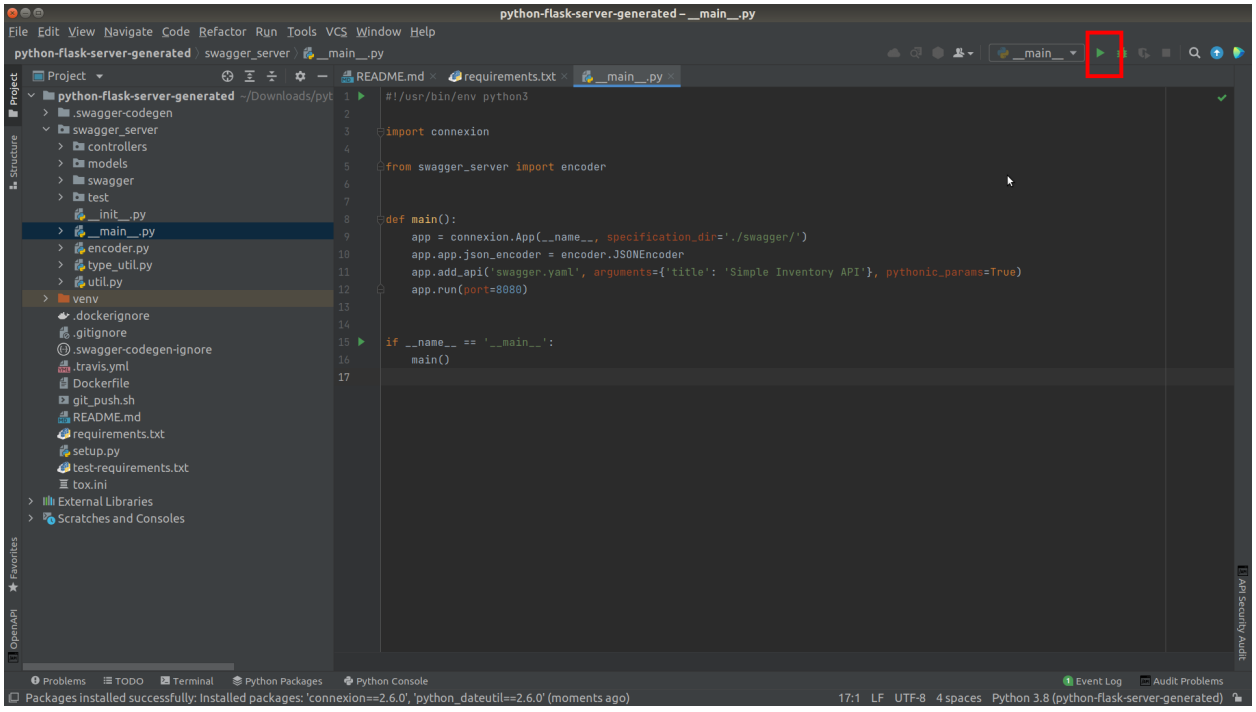


Select 'New environment' and press 'OK'
Replace the requirements.txt file with the following packages: [requirements.txt]

Open the '__main__.py' file and select from the top 'install requirements' to install the packages specified in the requirements.txt file



Press Run to start the flask server:

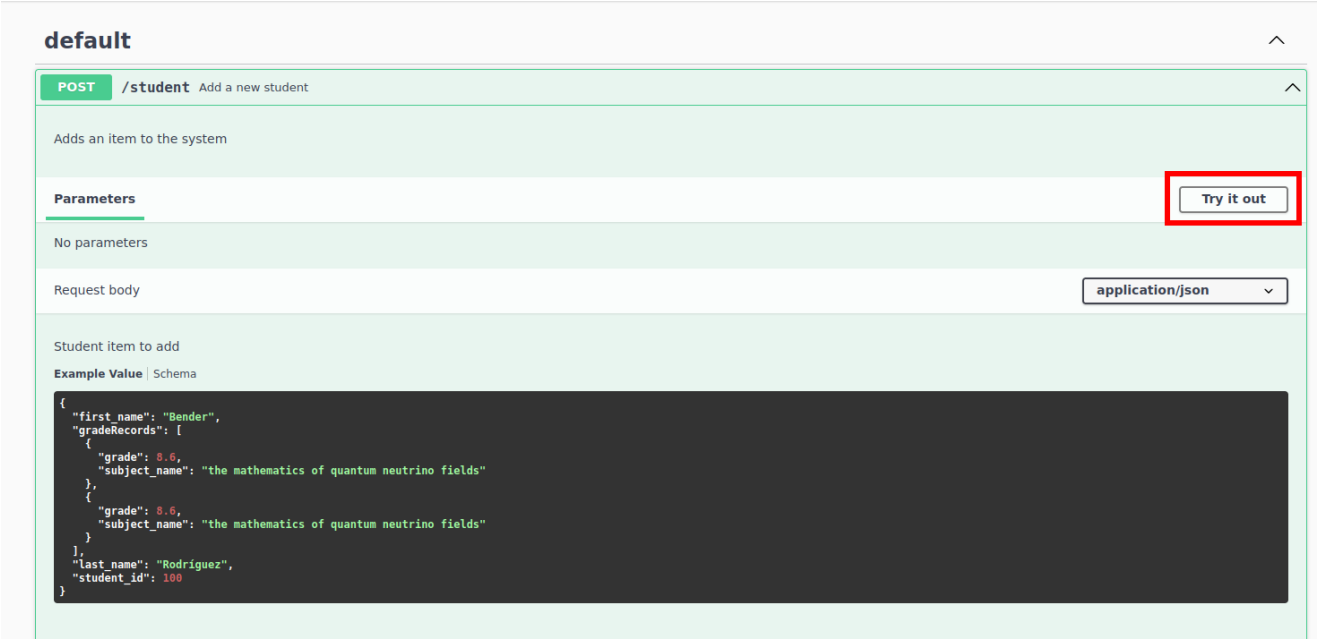


The UI API of your service will be in http://localhost:8080/tutorial/1.0.0/ui/. This is the same as the path defined in the OpenAPI definition in the section 'servers':

```
openapi: 3.0.0
servers:
  # Added by API Auto Mocking Plugin
```

```
  - description: SwaggerHub API Auto Mocking
    url: https://virtserver.swaggerhub.com/tutorial/1.0.0
```

You should see a similar page as the mockup in swaggerhub. Press 'Try it out':



The response body should be: "do some magic!"
In Pycharm if you open the 'default_controller.py' file, you'll see that the method 'add_student' returns the string "do some magic!".

6.1 Create Git Repository and Commit the Code

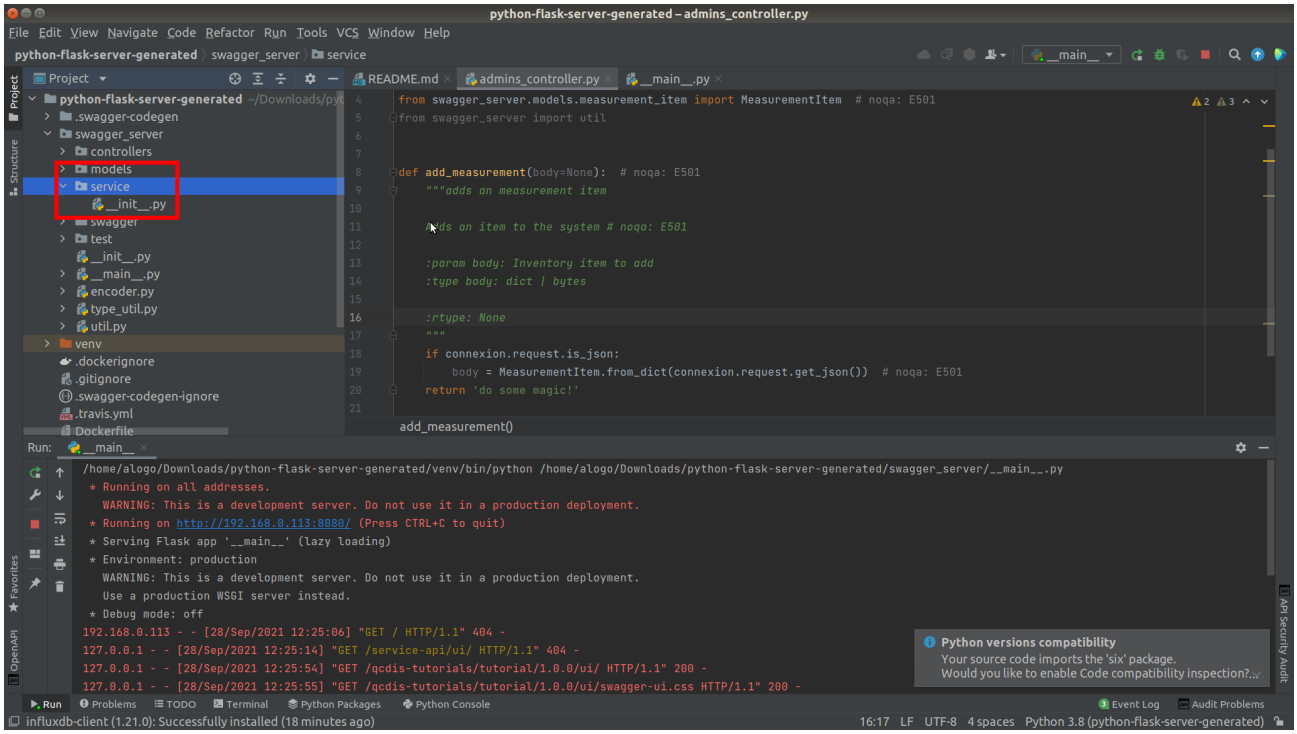
Create a git repository. Go to the directory of the code and initialize the git repository and push the code:

```
git init
git add .
git commit -m "first commit"
git remote add origin <REPOSETORY_URL>
git push -u origin master
```

More information on git can be found here: https://phoenixnap.com/kb/how-to-use-git

6.2 Implement the logic

In Pycharm create a package named 'service'. To do that right click on the 'swagger_server' package select 'New'-> 'Python Package' and enter the name 'service'



Inside the service package create a new python file named 'student_service'
In the student_service add the following code: [student_service.py]

Now you can add the corresponding methods in the 'default_controller.py'. To do that add in the top of the file:

```
from swagger_server.service.student_service import *
```

And in the 'add_student' student method we add the ‘add(body)’ in the rerun statement so now the method becomes :

```
def add_student(body=None):  # noqa: E501
    """Add a new student

    Adds an item to the system # noqa: E501

    :param body: Student item to add
    :type body: dict | bytes

    :rtype: float
    """
    if connexion.request.is_json:
        body = Student.from_dict(connexion.request.get_json())  #
noqa: E501
        return add(body)
    return 500,'error'
```

Do the same for the rest of the methods. For example, in the ‘delete_student’ method in the 'default_controller.py' file, you need to add ‘delete(student_id)’ in the return statement.

In general, it is a good idea to write an application using a layered architecture. By segregating an application into tiers, a developer can modify or add a layer, instead of reworking the entire application.
This is why we should create a new package in the code called 'service' and a python file named 'student_service.py'. In this code template, we use a simple file-based database to store and query data called TinyDB.
More information on TinyDB can be found here:
https://tinydb.readthedocs.io/en/latest/getting-started.html Now the 'default_controller.py' just needs to call the service's methods.

NOTE
Don't forget to regularly commit and push your code.

---

7 Build Test and Publish Docker Image

You can now build your web service as a Docker image DockerHub. To do that open the Dockerfile in the Pycharm project.
and update the python version from:

```
FROM python:3.6-alpine
```

To:

```
FROM python:3.8-alpine
```

So the Dockerfile will look like this: Dockerfile

```
FROM python:3.8-alpine

RUN mkdir -p /usr/src/app
WORKDIR /usr/src/app

COPY requirements.txt /usr/src/app/

RUN pip3 install --no-cache-dir -r requirements.txt

COPY . /usr/src/app

EXPOSE 8080

ENTRYPOINT ["python3"]

CMD ["-m", "swagger_server"]
```

Open a new terminal in the location of the Dockerfile and type:

```
docker build --tag <REPO_NAME>/student_service .
```

If the above command is not working you may need to use sudo.
Now test the image:

```
docker run -it -p 8080:8080 <REPO_NAME>/student_service
```

NOTE
Don't forget to stop the server from PyCharm otherwise you'll get an error:
shell docker: Error response from daemon: driver failed programming external connectivity on endpoint trusting_joliot (8cbc8523e15eb68f343d048ab59a9e6d): Error starting userland proxy: listen tcp4 0.0.0.0:8080: bind: address already in use. ERRO[0001] error waiting for container: context canceled

---

Login to docker hub to push the created image by typing:

```
docker login
```

Now push the image:

```
docker push <REPO_NAME>/student_service
```

More information on how to build and publish Docker images can be found here:
https://docker-curriculum.com/

7.1 Docker Exercises

7.1.1 Database Integration (Optional)

The code provided above uses an internal database called TinyDB. Change the code so that your service saves data in an mongoDB. This includes configuration files for the database endpoint database names the Dockerfile itself etc.
For testing your code locally use this docker-compose.yaml file: docker-compose.yaml. Make sure you replace the image with your own.

NOTE
The docker-compose.yaml file above will be also used to run the postman tests during grading.

If you need to install Docker Compose you can follow the instructions here:
https://docs.docker.com/compose/install/.

8 Write Tests

Before writing the tests in Github you need to create a token in Docker hub. To do that follow these instructions: https://docs.docker.com/docker-hub/access-tokens/
Next you need to add your Docker hub and token to your Github project secrets.
To create secrets follow these instructions https://docs.github.com/en/actions/security-guides/encrypted-secrets#creating-encrypted-secrets-for-a-repository.
You need to create two secrets, one named REGISTRY_USERNAME and one REGISTRY_PASSWORD.
Therefore, you need to run the above instructions twice. First time the name of the secret will be REGISTRY_USERNAME and second will be REGISTRY_PASSWORD.

In your code directory create a new folder named 'postman'. In the new 'postman' folder add these files:

- Postman_collection.json
- postman_environment.json

Make sure your code in the Git repository is up to date. Go to the repository and page create a new file with 'Add file'->'Create new file'. On the top define the path of your file.

```
.github/workflows/main.yml
```

Set the contents of your file as: main.yml

9 Deploy Web Service on Kubernetes (microk8s)

9.1 Install microk8s

If you have access to a Kubernetes cluster you may skip this step
You can find microk8s installation instructions: https://microk8s.io/
If you have access to a cloud VM you may install microk8s there. Alternatively, you may also use VirtualBox: https://www.virtualbox.org/wiki/Downloads
After you complete the installation make sure you start microk8s and enable DNS

```
microk8s start
microk8s enable dns
```

9.2 Test K8s Cluster

This is a basic Kubernetes deployment of Nginx. On the master node create a Nginx deployment:

```
microk8s kubectl create deployment nginx --image=nginx
```

You may check your Nginx deployment by typing:

```
microk8s kubectl get all
```

The output should look like this:

```
NAME                         READY   STATUS              RESTARTS
AGE
pod/nginx-6799fc88d8-wttct   0/1     ContainerCreating   0
6s

NAME                 TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)
AGE
service/kubernetes   ClusterIP   10.96.0.1    <none>        443/TCP
134d

NAME                    READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/nginx   0/1     1            0           7s

NAME                               DESIRED   CURRENT   READY   AGE
replicaset.apps/nginx-6799fc88d8   1         1         0       7s
```

You will notice in the first line 'ContainerCreating'. This means that the K8s cluster is

downloading and starting the Nginx container. After some minutes if you run again:

```
microk8s kubectl get all
```

The output should look like this:

```
NAME                         READY   STATUS    RESTARTS   AGE
pod/nginx-6799fc88d8-wttct   1/1     Running   0          39s

NAME                 TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)
AGE
service/kubernetes   ClusterIP   10.96.0.1    <none>        443/TCP
134d

NAME                    READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/nginx   1/1     1            1           40s

NAME                               DESIRED   CURRENT   READY   AGE
replicaset.apps/nginx-6799fc88d8   1         1         1       40s
```

At this point Nginx is running on the K8s cluster, however it is only accessible from within the cluster. To expose Nginx to the outside world we should type:

```
microk8s kubectl create service nodeport nginx --tcp=80:80
```

To check your Nginx deployment type:

```
microk8s kubectl get all
```

You should see the among others the line:

```
service/nginx        NodePort    10.98.203.181   <none>
80:30155/TCP   6s
```

This means that port 80 is mapped on port 30155 of each node in the K8s cluster.

NOTE
The mapped port will be different on your deployment. Now we can access Nginx from http://IP:NODE_PORT.

You may now delete the Nginx service by using:

```
microk8s kubectl delete service/nginx
```

9.3 Deploy Web Service on K8s Cluster

To deploy a RESTful Web Service on the K8s Cluster log in the K8s Cluster create a folder

named 'service' and add this file in the folder: student_service.yaml

Open 'student_service.yaml' and replace the line:

```
image: IMAGE_NAME
```

with the name of your image as typed in the docker push command.
If you chose to integrate with an extremal database you will need to add the Deployment and service for MongoDB: mongodb.yaml

To create all the deployments and services type in the K8s folder:

```
microk8s kubectl apply -f .
```

This should create the my-temp-service deployments and services. To see what is running on the cluster type:

```
microk8s kubectl get all
```

You should see something like this:

```
NAME                              READY   STATUS    RESTARTS   AGE
pod/nginx-6799fc88d8-q7hzv        1/1     Running   0          21m
pod/service-6c75dff7db-57sw5      1/1     Running   0          53s

NAME                 TYPE        CLUSTER-IP       EXTERNAL-IP
PORT(S)          AGE
service/kubernetes   ClusterIP   10.152.183.1     <none>
443/TCP          26m
service/service      NodePort    10.152.183.176   <none>
8080:30726/TCP   53s

NAME                         READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/nginx        1/1     1            1           21m
deployment.apps/service      1/1     1            1           53s

NAME                                    DESIRED   CURRENT   READY   AGE
replicaset.apps/nginx-6799fc88d8        1         1         1       21m
replicaset.apps/service-6c75dff7db      1         1         1       53s
```

Note that in this output 'service/service' is mapped to 30726. In your case it may be a different number. Now your service should be available on http://IP:NODE_PORT/