

Craft demo

The Accounting team would like to quickly look up country codes to speed up processing of expense reports, they currently have a website bookmarked and the process is manual and tedious. Help them speed things up by creating a simple cli.

- Write a country lookup service using your favorite programming language.
- Explore the following API <https://www.travel-advisory.info/api>
- Given country code -> return country name example:

```
lookup --countryCode=AU
Australia
```

- Save the data from <https://www.travel-advisory.info/api> to a file called `data.json` and add functionality to your program to work with a file instead of real api endpoint
- Make sure your program supports multiple country codes as input
- Be ready to demo execution of your program in the terminal

Bonus

Demand for your tool is increasing and there is an ask from multiple teams to be able to use county code lookup automation. Decision has been made to extend your tool to expose its functionality via REST.

- Convert code written in Craft demo to a REST service with two routes
- `/health` returns health of your service
- `/diag` check returns status of the api <https://www.travel-advisory.info/api> return `{"api_status":{... "code":200,"status":"ok"}}` that you obtained from hitting an API
- `/convert` – converts country name to country code
- Create local k8s cluster on your workstation
- Deploy your service to local k8s cluster

The Operations team likes your service, but to make their job easier and keep things operating at an optimal level

- Setup basic monitoring

Solution:

We can achieve this with a simple Flask application that exposes three endpoints and utilizes the Flask web framework for handling HTTP requests. Let me explain the code in sections:

main.py file

Flask: The main Flask class that is used to create the web application.

jsonify: A Flask function that helps in creating JSON responses.

request: Allows access to incoming request data.

json: Python's built-in module for working with JSON data.

requests: A popular HTTP library for making requests to APIs

data.py

There is a function `fetch_data` written in `data.py`

`fetch_data`: A function that takes an API URL as a parameter, sends an HTTP GET request to that URL using the `requests.get` method, and checks if the response status code is 200 (OK). If the status code is 200, it returns the JSON content of the response using `response.json()`. If the status code is not 200, it prints an error message and returns `None`.

Then there is a `save_to_file` function which will save the data in `data.json` file that will be used as source of information in the `main.py` (application)

`save_to_file`: A function that takes data and a filename as parameters and writes the data to a file in JSON format.

It uses the `json.dump` method to serialize the data to a JSON-formatted string and writes it to the specified file.

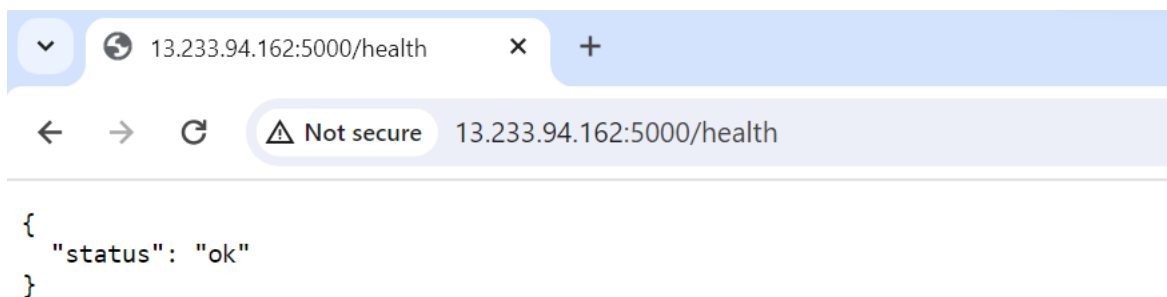
The `indent=2` argument adds indentation to make the JSON file more human-readable.

We can now run the `main.py` file using the following command: #I am using a AWS EC2 instance to test the application.

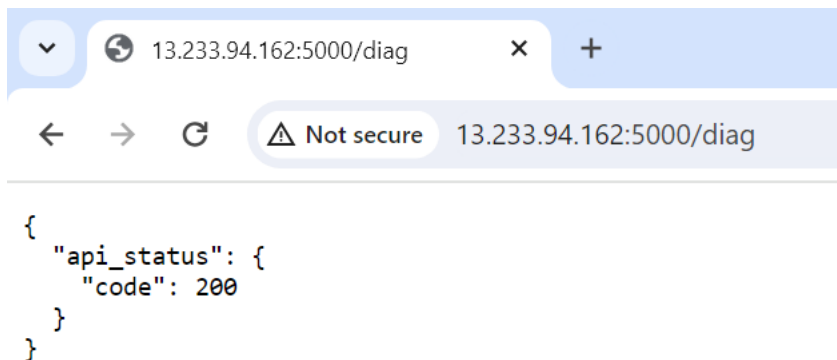
```
#python3 main.py
```

```
ubuntu@ip-172-31-33-171:~/ankit$ python3 main.py
* Serving Flask app 'main' (lazy loading)
* Environment: production
  WARNING: This is a development server. Do not use it in a production deployment.
  Use a production WSGI server instead.
* Debug mode: on
* Running on all addresses.
  WARNING: This is a development server. Do not use it in a production deployment.
* Running on http://172.31.33.171:5000/ (Press CTRL+C to quit)
* Restarting with stat
* Debugger is active!
* Debugger PIN: 714-810-253
103.164.211.193 - - [09/Jan/2024 06:42:24] "GET /convert?countryName=Australia HTTP/1.1"
```

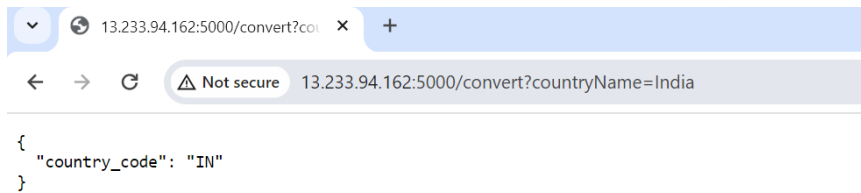
Then we go to the browser to browse the application:



A screenshot of a web browser window. The address bar shows the URL `13.233.94.162:5000/health`. The page content displays a JSON response: `{ "status": "ok" }`.



A screenshot of a web browser window. The address bar shows the URL `13.233.94.162:5000/diag`. The page content displays a JSON response: `{ "api_status": { "code": 200 } }`.



So now the application is running successfully.

Now we can dockerize the application and run this as a container in Kubernetes or any other orchestration tool.

Dockerfile

```
FROM python:3
WORKDIR /usr/src/app
COPY requirements.txt ./
RUN pip install --no-cache-dir -r requirements.txt
EXPOSE 5000
COPY . .
CMD ["python", "./main.py"]
```

I have installed docker on my EC2.

Command used to build the image:

#docker build and docker run command and -p to do port mapping to expose it for outside world.

```
ubuntu@ip-172-31-33-171:~/ankit$ sudo docker run -p 5000:5000 62f6f07e2368
* Serving Flask app 'main'
* Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment
* Running on all addresses (0.0.0.0)
* Running on http://127.0.0.1:5000
* Running on http://172.17.0.2:5000
Press CTRL+C to quit
* Restarting with stat
* Debugger is active!
* Debugger PIN: 293-704-684
```

This container can also be run on Kubernetes.

We can deploy a service manifest file

Service.yaml

```
apiVersion: v1
kind: Service
metadata:
  name: country-lookup-service
spec:
  selector:
    app: country-lookup
  ports:
    - protocol: TCP
      port: 80
      targetPort: 5000
  type: LoadBalancer
```

port: 80: Exposes port 80 on the service.

targetPort: 5000: Forwards incoming traffic on port 80 to port 5000 on the selected pods.

type: LoadBalancer: Specifies the type of service as a LoadBalancer. This implies that the Kubernetes cluster should provision an external load balancer to handle incoming traffic and distribute it to the pods selected by the service.

A Kubernetes Service named "country-lookup-service" that directs incoming traffic on port 80 to pods labeled with "app: country-lookup" using the TCP protocol. The service type is set to LoadBalancer, indicating that an external load balancer should be provisioned to distribute the traffic.