

API and Python training

Session 8

This session's agenda

- SDKs – Software Development Kits
- Using SDKs in Python
- SDK examples

Building simple API server

- Recap from Session 1 – Controllers
- Northbound and Southbound APIs
- Intro to Flask
- Defining routes
- Returning JSON data
- Reading data from files
- Filtering data - List comprehension
- Next steps

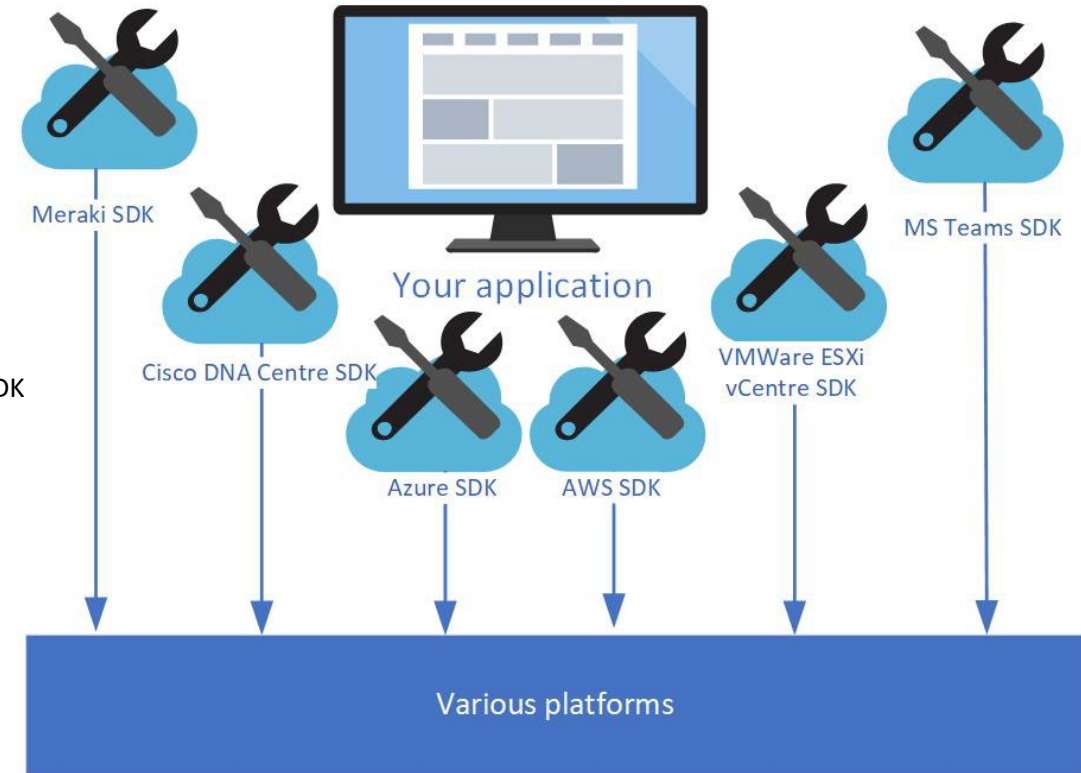
SDK – Software Development Kit

- A software development kit (SDK) is a set of software tools and programs provided by hardware and software vendors that developers can use to build applications for specific platforms. These providers make their SDKs available to help developers easily integrate their apps with their services.

<https://whatis.techtarget.com/definition/software-developers-kit-SDK>

- SDKs 'hide' the complexity of handling underlying API, providing ready to use 'building blocks'
- Most software and hardware have SDKs available – when you start working with a product/platform, check if there an SDK is available
- Most SDKs are open-source, so check the code
- Use it where available/possible – greatly simplifies the development process, allow you to consume the service and focus on application logic rather than on platform implementation details or API version
- Example: Azure Storage SDK

<https://github.com/Azure/azure-sdk-for-python/tree/master/sdk/storage/azure-storage-blob>



Containers

Scalable, cost-effective storage for unstructured data

[Learn more](#)

File shares

Serverless SMB and NFS file shares

[Learn more](#)

Tables

Tabular data storage

[Learn more](#)

Tools and SDKs

Storage Explorer (preview) PowerShell Azure CLI .NET Java Python Node.js

SDKs in Python

- SDKs for Python are normally libraries installed and imported
- As usual, install with pip or IDE ----->
- And import in the code:

```
from azure.storage.blob import BlobSasPermissions, generate_blob_sas

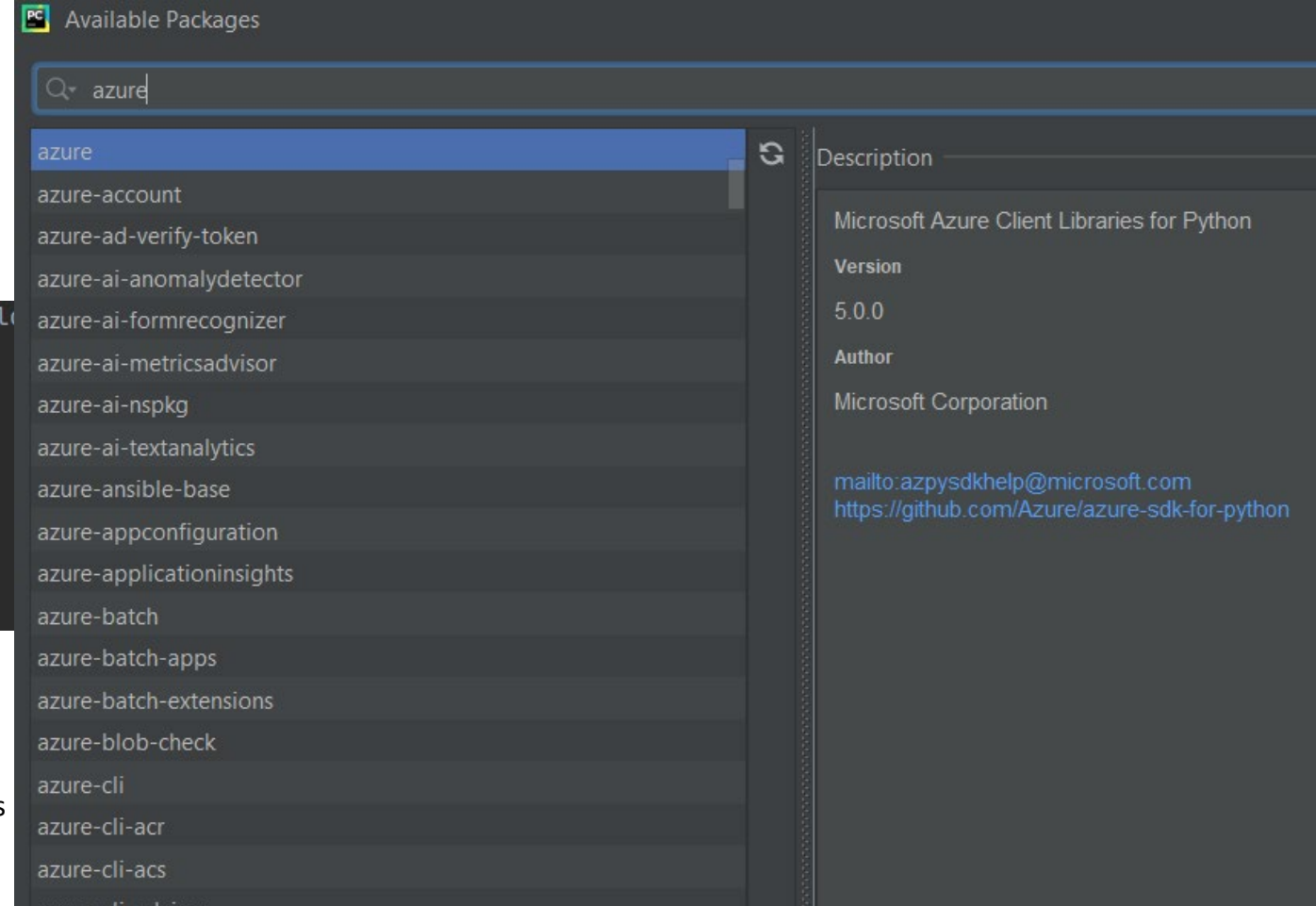
sas = generate_blob_sas(
    account_name=storage_account_name,
    container_name=container_name,
    blob_name=file_name,
    permission=BlobSasPermissions(read=True),
    expiry=datetime.utcnow() + timedelta(hours=2))
```

- Compare with using Azure REST API:

<https://docs.microsoft.com/en-us/rest/api/storageservices/service-sas-examples>

- Example using Azure blob storage SDK:

<https://github.com/supro200/sdwan-auto-upgrade>



SDKs examples

- During our recent sessions we already built a simple SDK
Compare with Real Cisco DNA SDK, should be similar ☺

<https://blogs.cisco.com/developer/using-cisco-dna-center-sdk>

Other examples:

- Meraki SDK - <https://developer.cisco.com/meraki/>

- Vmware SDK:

<https://github.com/vmware/vsphere-automation-sdk-python>

- AWS SDK

<https://boto3.amazonaws.com/v1/>

[documentation/api/latest/guide/quickstart.html](https://boto3.amazonaws.com/v1/documentation/api/latest/guide/quickstart.html)

```
1  from dnac_library import Controller
7  controller = Controller(URL, user, password)
8
9  controller.get_dnac_device()
10 controller.get_dnac_interface()
```

```
import meraki
dashboard = meraki.DashboardAPI(API_KEY)

response = dashboard.organizations.getOrganizations()
```

```
# Connect to a vCenter Server using username and password
vsphere_client = create_vsphere_client(server='<vc_ip>', username='<vc_username>')

# List all VMs inside the vCenter Server
vsphere_client.vcenter.VM.list()
```

```
import boto3

# Let's use Amazon S3
s3 = boto3.resource('s3')

# Print out bucket names
for bucket in s3.buckets.all():
    print(bucket.name)
```

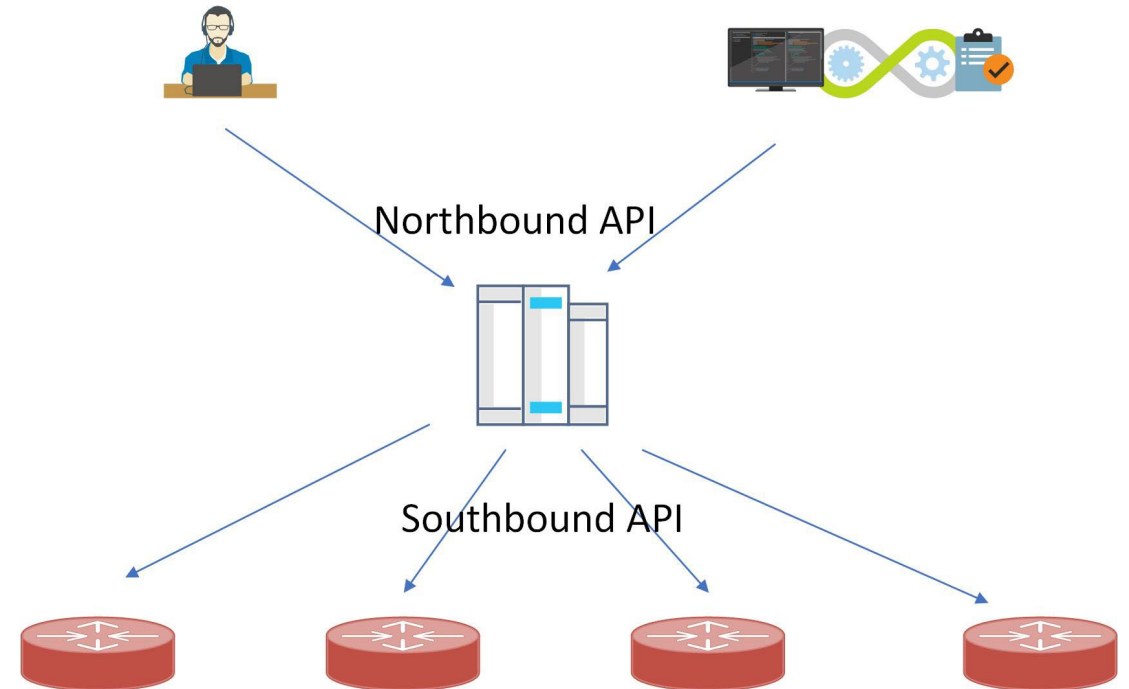
Looking at REST API from different side:

Building API Server



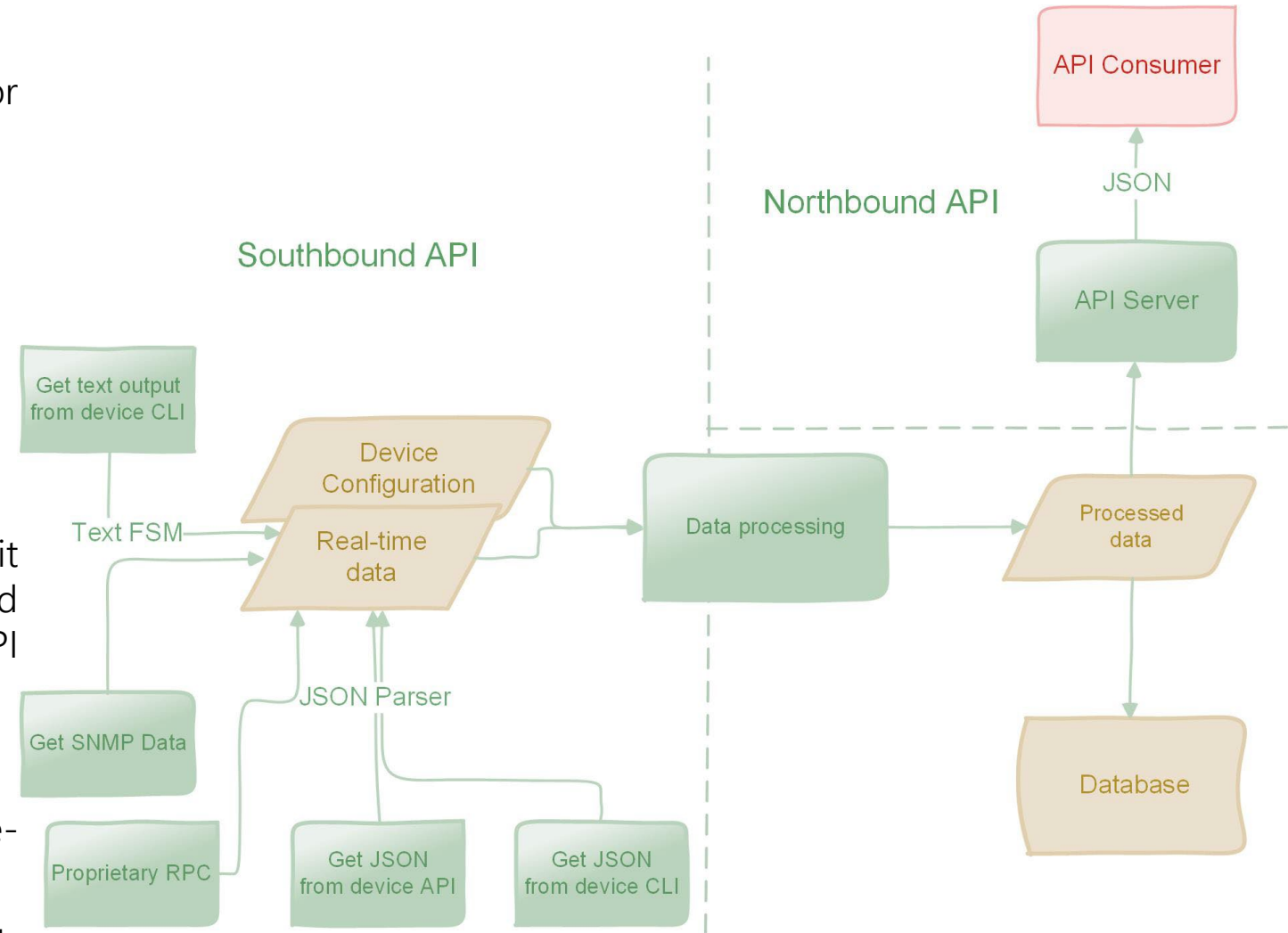
Recap from session 1 - Controllers

- Remember session 1 ?
- Controllers offers consumable API, doing authentication, RBAC, etc. This type of API exposed to be used is called northbound API.
- Controllers maintain connections to identify the operational state of controlled devices, push configuration, pull logs. Controlled devices also can push alerts, state changes. This type of comms is called southbound API. This API can be proprietary, maybe even SSH or SNMP.
- Example of controllers:
Cisco vManage NSX Manager Kubernetes Master
Philips Hue Lighting Google Smart Home
- Public clouds also provide API endpoints to control resources, so they can be also considered as controllers to some extent.



Southbound vs Northbound API

- **Southbound API** is to talk to specific vendor devices and can be any, for example:
 - SSH
 - Netconf
 - RESTConf
 - Any proprietary RPC – Remote Procedure calls
 - SNMP
- When the controller gets data from the devices it processes it, can store the raw and/or processed data in a database and presented to the API consumer
- We'll build simple **Northbound API** using pre-processed data in CSV format
- REST API server is essentially a lightweight web server



Flask – getting started

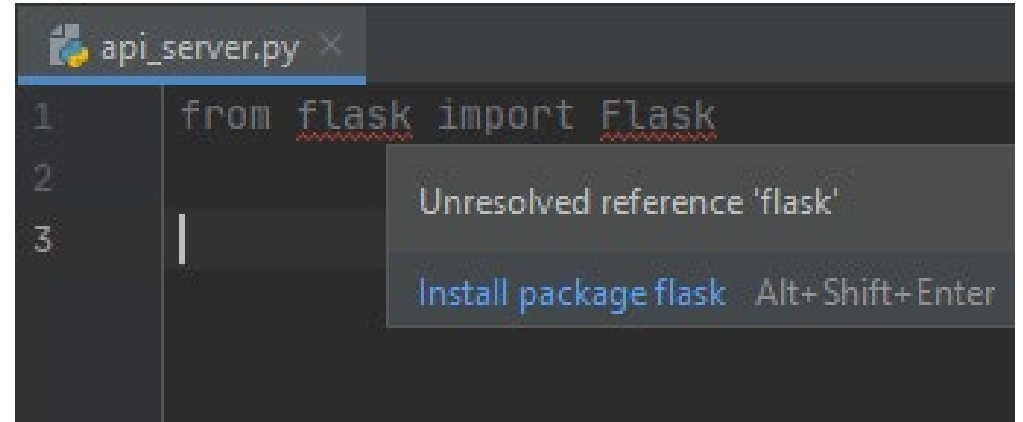
- There are many web frameworks for Python, most popular are Flask and Django
- Flask is a lightweight, easy to use, good documentation and community support
- To get started, install flask package using **pip**

or simply with IDE ----->

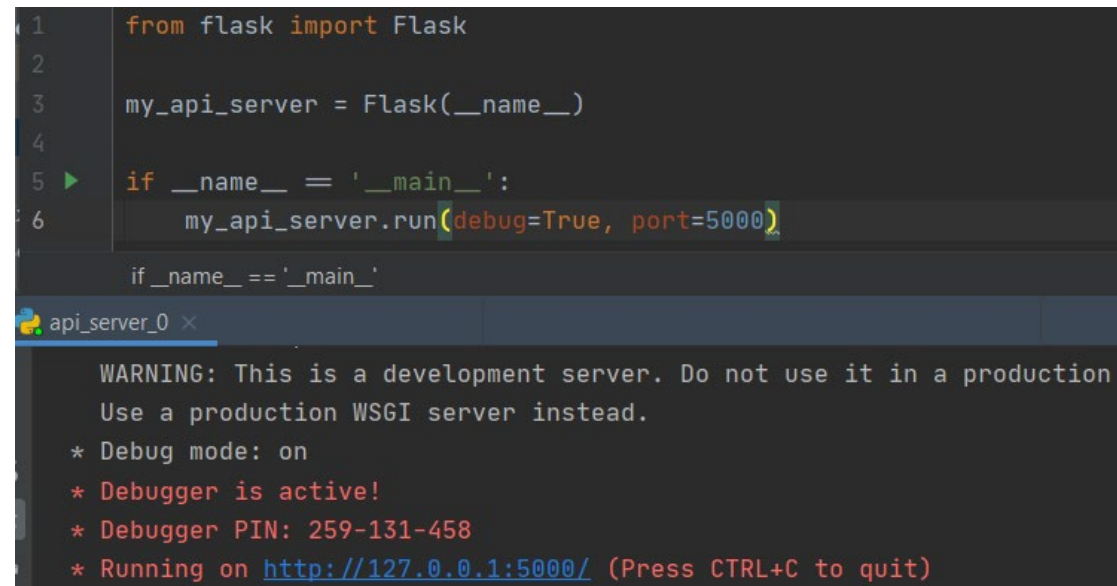
- Minimal code to run the server is ----->
- Note you can specify the port
- **debug=True** mean you can change your code and don't need to restart the server, it will detect the changes **when you save the file** and restart automatically – convenient while you're developing
- By default, it only accepts connections from localhost, to override it, use parameter **host='0.0.0.0'** (or whatever network you originate requests from)
- Refer to the doc for full description

<https://flask.palletsprojects.com/en/1.1.x/quickstart/>

- Then test with CURL, and get 404 – Not Found which means the server is running, but there are no routes, as expected



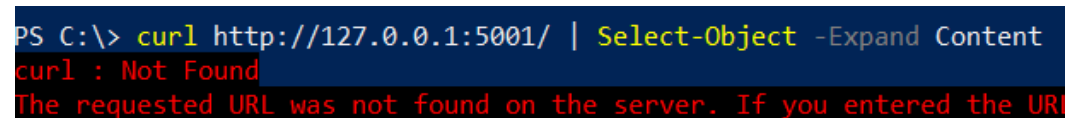
```
api_server.py x
1 from flask import Flask
2
3 |
   Unresolved reference 'flask'
   Install package flask Alt+ Shift+ Enter
```



```
1 from flask import Flask
2
3 my_api_server = Flask(__name__)
4
5 if __name__ == '__main__':
6     my_api_server.run(debug=True, port=5000)

if __name__ == '__main__':

api_server_0 x
WARNING: This is a development server. Do not use it in a production
Use a production WSGI server instead.
* Debug mode: on
* Debugger is active!
* Debugger PIN: 259-131-458
* Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
```



```
PS C:\> curl http://127.0.0.1:5001/ | Select-Object -Expand Content
curl : Not Found
The requested URL was not found on the server. If you entered the URL
```

Flask routing

- Add routes using **decorators** - @
- Decorators are in fact functions which are called immediately before the function defined next
- In flask we use decorator **route** with parameter of path
- In next line define the function which will implement the application logic
- In return statement you return the API response body

<flask_object>.route(<path>)

def <your function name> ():

code.....

return <something to return in API response>

- Make API call again using resource/path ----->

```
1  from flask import Flask
2
3  my_api_server = Flask(__name__)
4
5  @my_api_server.route('/')
6  def root_path():
7      return 'Accessed root'
8
9  @my_api_server.route('/device')
10 def device_path():
11     return 'Accessed device'
12
13 if __name__ == '__main__':
14     my_api_server.run(debug=True, port=5001)
```

```
PS C:\> curl http://127.0.0.1:5001/ | Select-Object -Property Content
Content
-----
Accessed root

PS C:\> curl http://127.0.0.1:5001/device | Select-Object -Property Content
Content
-----
Accessed device
```

Flask – returning JSON

- In API response you normally expect data in JSON format
- You can send the text string in as JSON
- Alternatively, Flask has embedded function **jsonify** which converts Python **dictionary** to JSON string
- Define the dictionary and supply it as a parameter to jsonify function Lines 11-12 and 16-17
- Change path to /api/<resource> to look more professionally ☺
- Make API call again using resource/path ----->

```
1  from flask import Flask, jsonify
2  |
3  my_api_server = Flask(__name__)
4  |
5  @my_api_server.route('/')
6  def root_path():
7      return 'Accessed root'
8  |
9  @my_api_server.route('/api/device')
10 def api_device_path():
11     device_data = {'name': 'device1', 'capatibility': 'host'}
12     return jsonify(device_data)
13 |
14 @my_api_server.route('/api/sensor')
15 def api_sensor_path():
16     device_data = {'name': 'sensor1', 'temperature': '22'}
17     return jsonify(device_data)
18 |
```

api_server x

```
* Debugger is active!
* Debugger PIN: 259-131-458
* Running on http://127.0.0.1:5001/ (Press CTRL+C to quit)
127.0.0.1 - - [19/Jan/2021 16:52:05] "GET /api/device HTTP/1.1" 200 -
127.0.0.1 - - [19/Jan/2021 16:52:11] "GET /api/sensor HTTP/1.1" 200 -
* Detected change in 'C:\\dev\\session8_demo\\api_server.py', reloading
```

```
PS C:\> curl http://127.0.0.1:5001/api/device | Select-Object -Expand Content
{
  "capatibility": "host",
  "name": "device1"
}

PS C:\> curl http://127.0.0.1:5001/api/sensor | Select-Object -Expand Content
{
  "name": "sensor1",
  "temperature": "22"
}
```

Reading from CSV files

- We'll use pre-processed data to return in API response
- Python has standard CSV module (which means you don't need to install it using *pip*)
- CSV file with some example data ----->
- How to read data from the file:
 - Line 17 – open file, this is the recommended way of working with filesystems
 - Line 18 – uses csv module standard function DictReader to convert CSV to Python dictionary using headers as keys
 - Line 19 – convert dictionary to Python string

```
15 @my_api_server.route('/api/sensor')
16 def api_sensor_path():
17     with open('device_data.csv', 'r') as file:
18         data = list(csv.DictReader(file, delimiter=',', quotechar='"'))
19     return jsonify(data)
```

- Make API call again using resource/path,
receive the data in JSON format ----->
- If you're using Windows powershell:
curl http://127.0.0.1:5000/api/sensor | Select-Object -Expand Content

```
device_data.csv X
1 Device name,Device type,Device IP,State
2 L0sensor1,Temperature Sensor,10.3.4.3,Green
3 L0sensor2,Humidity Sensor,10.3.4.15,Red
4 L0sensor3,Humidity Sensor,10.3.4.25,Green
5 L1sensor1,Temperature Sensor,10.3.4.55,Green
6 L1sensor2,Temperature Sensor,10.3.4.51,Red
7 L1sensor3,Humidity Sensor,10.3.4.50,Green
8 L1sensor4,Humidity Sensor,10.3.4.65,Yellow
9 L1sensor5,Temperature Sensor,10.3.4.95,Yellow
```

```
PS C:\> curl http://127.0.0.1:5000/api/sensor | Select-Object -Expand Content
[
  {
    "Device IP": "10.3.4.3",
    "Device name": "L0sensor1",
    "Device type": "Temperature Sensor",
    "State": "Green"
  },
  {
    "Device IP": "10.3.4.15",
    "Device name": "L0sensor2",
    "Device type": "Humidity Sensor",
    "State": "Red"
  },
  {
    "Device IP": "10.3.4.25",
    "Device name": "L0sensor3",
    "Device type": "Humidity Sensor",
    "State": "Green"
  },
  {
    "Device IP": "10.3.4.55",
    "Device name": "L1sensor1",
    "Device type": "Temperature Sensor",
    "State": "Green"
  },
  {
    "Device IP": "10.3.4.51",
    "Device name": "L1sensor2",
    "Device type": "Temperature Sensor",
    "State": "Red"
  },
  {
    "Device IP": "10.3.4.50",
    "Device name": "L1sensor3",
    "Device type": "Humidity Sensor",
    "State": "Green"
  },
  {
    "Device IP": "10.3.4.65",
    "Device name": "L1sensor4",
    "Device type": "Humidity Sensor",
    "State": "Yellow"
  },
  {
    "Device IP": "10.3.4.95",
    "Device name": "L1sensor5",
    "Device type": "Temperature Sensor",
    "State": "Yellow"
  }
]
```

Using variables in URL

- In Flask routes you can use variables, and pass these variables as parameters to a function
- Add a new route with variable in <>
- Line 25 – **List comprehension** – build a new list from another list using conditions, there are 3 parts:
 - *item* – takes the element from the original list to the new list
 - *for item in data* - iterate though the list data
 - *if item['Device name'] == name* - for the element taken from the list (element here is a dictionary), compare if value of key Device name matches the requested value

```
21 @my_api_server.route('/api/sensor/<name>')
22 def api_get_sensor_by_name(name):
23     with open('device_data.csv', 'r') as file:
24         data = list(csv.DictReader(file, delimiter=',', quotechar='"'))
25         filtered_data = [item for item in data if item['Device name'] == name]
26         return jsonify(filtered_data)
```

- Make API call again
using resource name ----->

```
PS C:\> curl http://127.0.0.1:5001/api/sensor/L1sensor1 | Select-Object -Expand Content
[
  {
    "Device IP": "10.3.4.5",
    "Device name": "L1sensor1",
    "Device type": "Temperature Sensor",
    "Status": "Green"
  }
]
```

Demo

Summary and next steps

- Summary

SDK

Flask

Next time – Final session

- Open API specification/Swagger
- Connexion module
- Training summary