API and Python training

Session 1

Session plan

Session 1

- What is API
- A bit of history
- API examples
- Types of API
- REST API
- How to consume API
- Practice
- Controllers
- Next steps

Session 2

- API authentication
- Postman

Session 3

- Python IDE
- Python basics
- Simple data types
- Complex data types

Session 4

- Python libraries, import statements
- Requests module
- Building Python application to consume API
- Manipulating JSON

Session 5

- Python control structures
- Handling exceptions
- Python functions

Session 6

- What is SDK
- Using SDK - examples

Session 7

- Flask module
- Building Python Flask application

Session 8

- Connexion module
- Adding API to your application

API and Python training

What we will be learning

We'll learn how to use various tools to make queries to REST API endpoints.

We'll be learning Python and specifically focus how to use it with REST API.

We won't be focusing on network or systems automation specifically, but we'll use Cisco DNA Center and ServiceNow as examples

I'll be preparing and share materials for each session, you're more than welcome to suggest new topics

Some topics and definitions will be simplified, please find the time to find more details and read about what we discussed.

Similarly, we'll be doing exercises, but it would be cool if you build and try your own examples.

Goals and tools

The goal of this training to get you comfortable with REST API – what is this, use cases, how to consume.

How to build Python applications not only to consume APIs, but also to build your own.

It is supposed you have little knowledge and no experience with programming concepts and Python

When you finish this training, you should be able to start building your own applications to use in your work or home projects whether it is SD-WAN, Raspberry PI, home automation - whenever your ideas brings you ☺

Explore , Experiment, ask questions ☺

Tools we will use

- cURL
- Postman
- Python 3
- IDE - integrated development environment (PyCharm)

What we won’t cover

To stay focused, we won’t cover in detail:

- Network automation tools, such as Ansible and StackStorm
- Git
- Data Models (such as YANG)
- APIs and protocols other than REST API (such as NETCONF)

This session agenda

- What is API
- A bit of history
- API use cases and examples
- Types of API
- REST API
- How to consume API
- Practice
- Webhooks
- Controllers
- Next steps

What is API

API – Application Programming Interface

Keyword here is still Interface

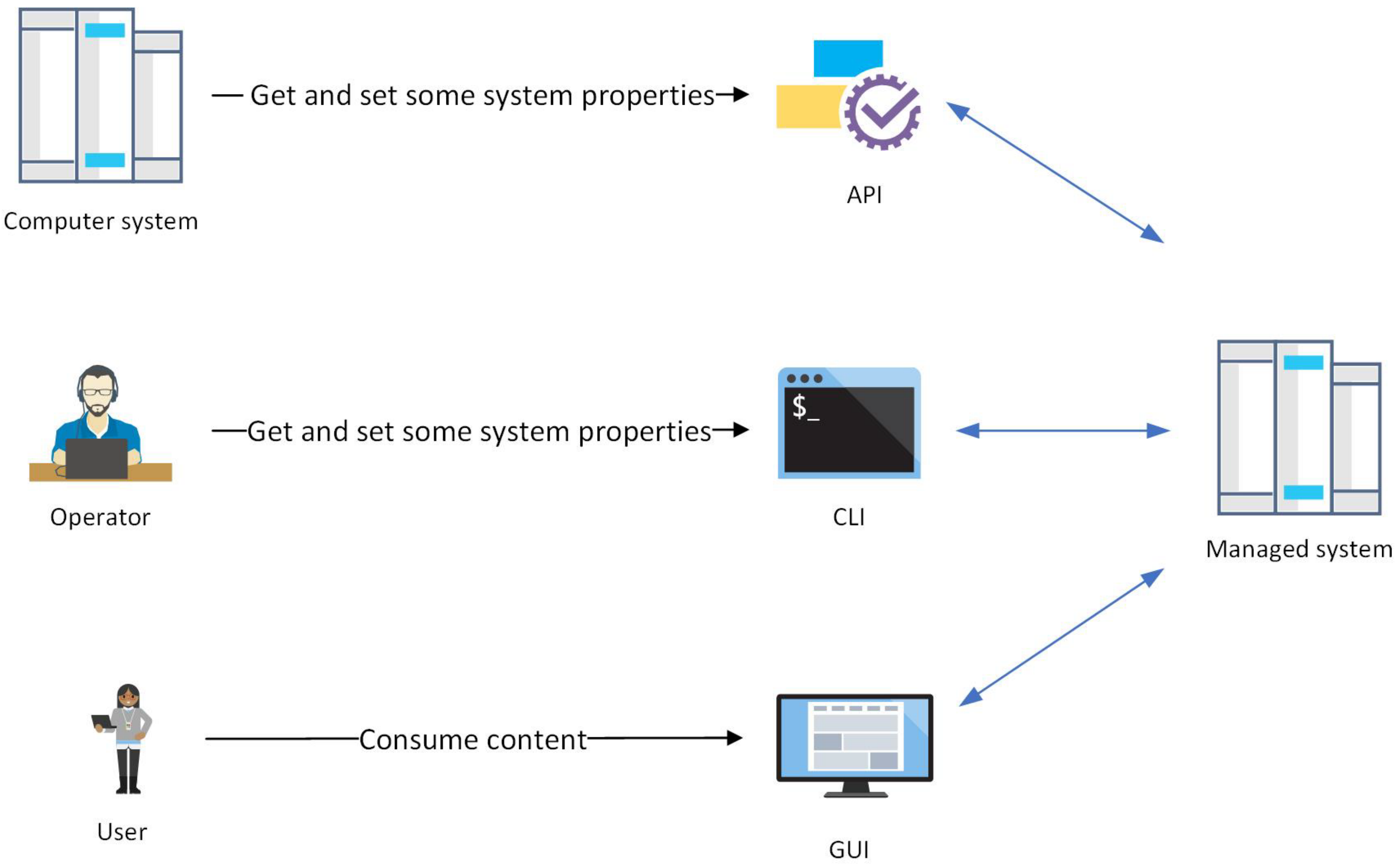
how we can interact with a system

Other types of interfaces:

CLI – Command Line Interface

GUI – Graphical Users Interface

They serve the same function, but differently



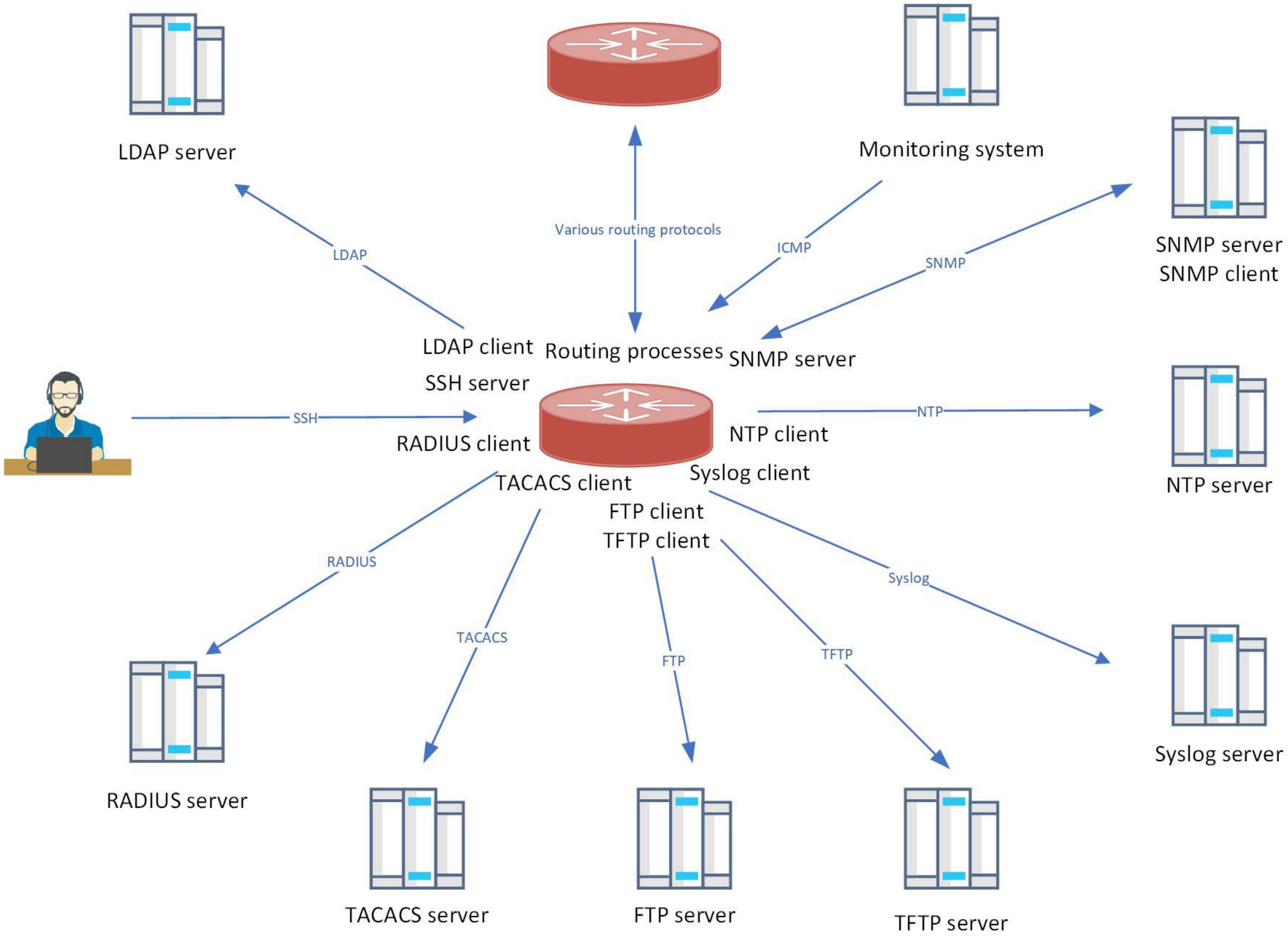
Computer system
Get and set some system properties
API
Operator
Get and set some system properties
$_
CLI
Managed system
User
Consume content
GUI

What protocols we use and what's wrong with them

Different protocols for specific purposes

Problems:

- All protocols work differently
- Each protocol has its own packet formats
- Each protocol requires its own client and server
- To add functionality, invent a new protocol or add extensions, incompatible with previous (SNMP v2/v3)
- The same protocol with added features of protocol can require using different ports (LDAP/LDAPS)
- Difficult for operators to manage all this complex infrastructure and for developers to add new features or develop a new product
- Security – multiple ports to open, difficult to track potential security issues with multiple software components



Software industry faced the same issue

Proprietary protocols, message formats, encoding rules

Monolith software, closed 3-tier architecture, very difficult to build distributed platforms

Difficult to make changes in software or implement new features

After quite a few transformations, the developer community and leading companies started adopting a new style of software architecture and building standard interfaces to their software components.

One if the most prominent works in this area is Roy Fielding’s dissertation *Architectural Styles and the Design of Network-based Software Architectures*:

https://twobithistory.org/2020/06/28/rest.html

https://www.ics.uci.edu/~fielding/pubs/dissertation/fielding_dissertation.pdf

- All teams will henceforth expose their data and functionality through service interfaces.
- Teams must communicate with each other through these interfaces.
- ……
- All service interfaces, without exception, must be designed from the ground up to be externalizable.
- Anyone who doesn’t do this will be fired.
- Have a nice day

https://gigaom.com/2011/10/12/419-the-biggest-thing-amazon-got-right-the-platform/

One protocol to rule them all

Web API

- Single transport – HTTP
- Single port – 443 (or 80)
- Simple model - Request/reply
- Encryption from the box - HTTPS
- Text-based, lightweight
- Easy to implement, easy to consume

The same approach is used for inter-systems data exchange or within the same application (microservices)

Booking and eCommerce sites, payment gateways, public clouds, social media – all of them use Web APIs
You can get data from these systems and implement a business logic or workflow

Google : **API economy**

Examples

Get current weather:

http://www.bom.gov.au/fwo/IDV60901/IDV60901.95936.json

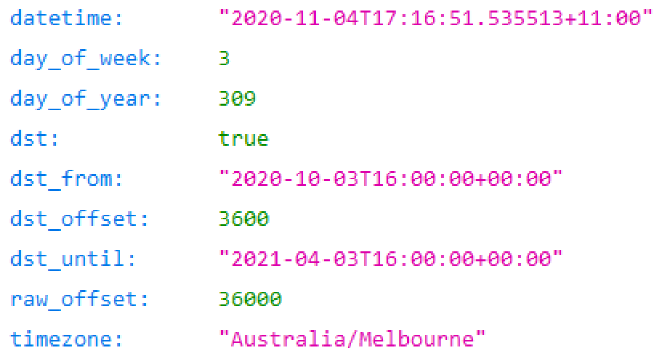
Close the blinds:

https://github.com/drbrain/indigo-powerview

Examples

Get current time:

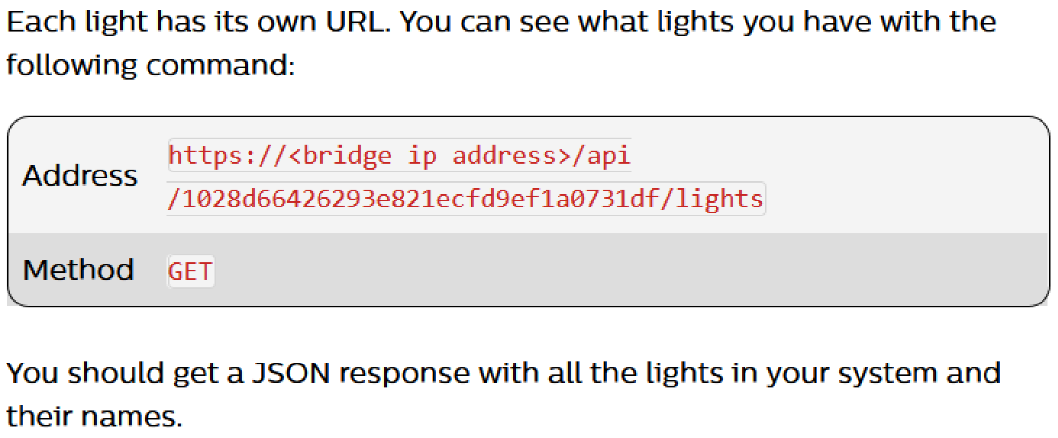
http://worldtimeapi.org/api/ip



```
datetime:       "2020-11-04T17:16:51.535513+11:00"
day_of_week:    3
day_of_year:    309
dst:            true
dst_from:       "2020-10-03T16:00:00+00:00"
dst_offset:     3600
dst_until:      "2021-04-03T16:00:00+00:00"
raw_offset:     36000
timezone:       "Australia/Melbourne"
```

Turn on the lights:

https://developers.meethue.com/develop/get-started-2/#turning-a-light-on-and-off



Each light has its own URL. You can see what lights you have with the following command:

Address	`https://<bridge ip address>/api /1028d66426293e821ecfd9ef1a0731df/lights`
Method	`GET`

You should get a JSON response with all the lights in your system and their names.

Examples

Your favourite stock has just dropped 10% ?

https://asxonline.com/public/documents/market-information-application--mia----rest-programmatic-interfa.html

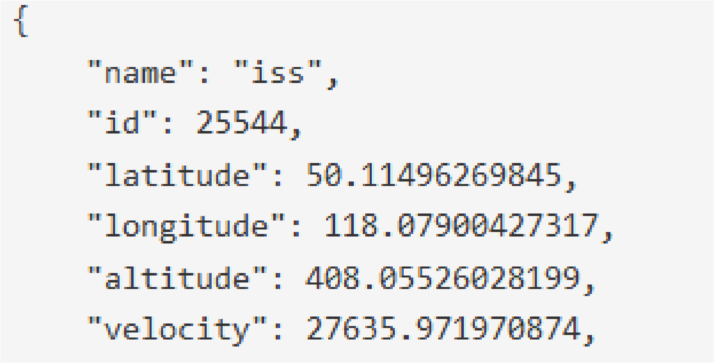
Post another kitten in Facebook:

https://developers.facebook.com/docs/graph-api/resumable-upload-api

Examples

ISS is flying above your home?

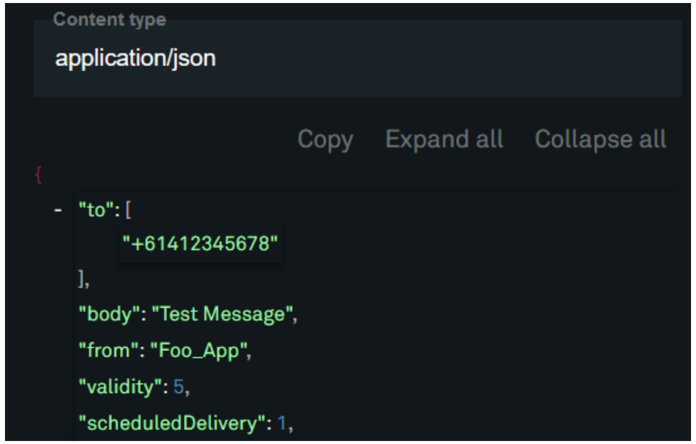
https://wheretheiss.at/w/developer



```
{
    "name": "iss",
    "id": 25544,
    "latitude": 50.11496269845,
    "longitude": 118.07900427317,
    "altitude": 408.05526028199,
    "velocity": 27635.971970874,
```

Send a text:

https://dev.telstra.com/content/messaging-api



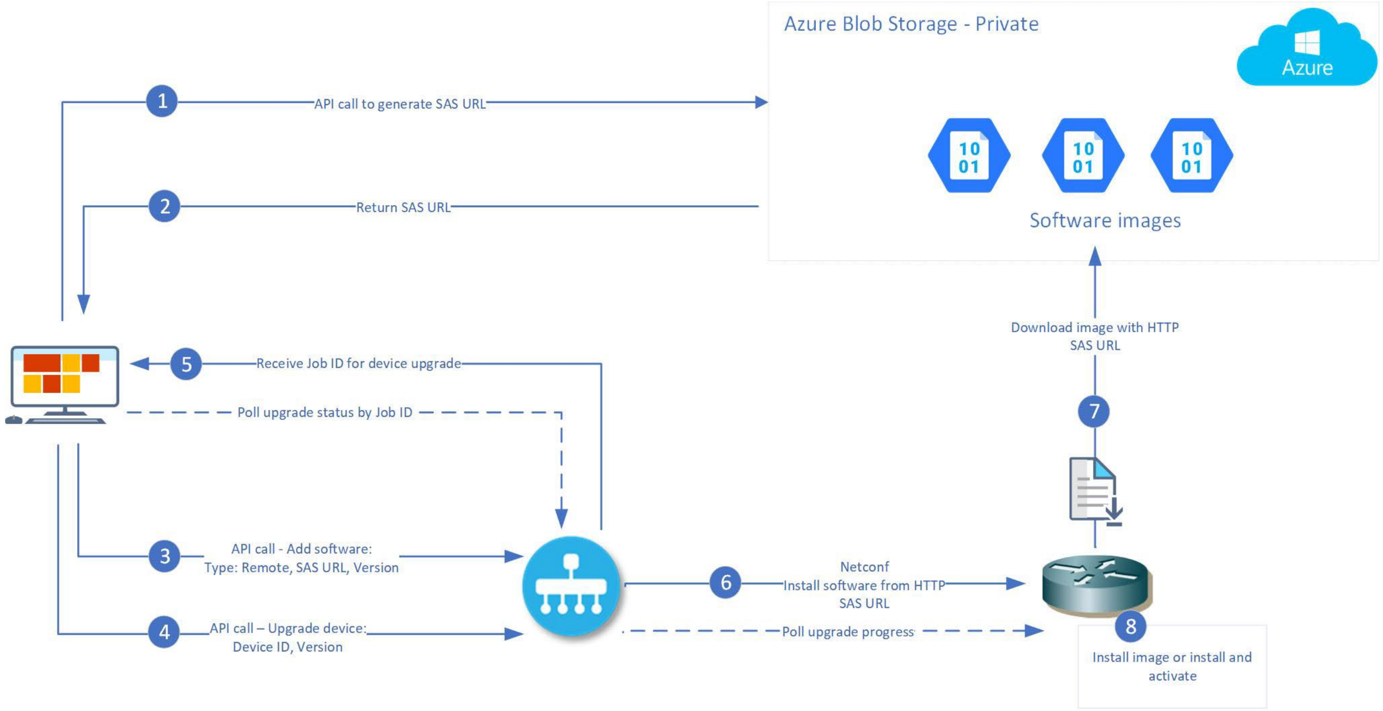
Content type

application/json

```
{
 -  "to": [
        "+61412345678"
    ],
    "body": "Test Message",
    "from": "Foo_App",
    "validity": 5,
    "scheduledDelivery": 1,
```

How we use API

- **ServiceNow** retrieve Alerts from **vManage** API and raises tickets
- **StackStorm** uses **vManage** API
- **LogicMonitor** retrieves cellular signal monitor from vManage
- Ansible modules to communicate with Storage


Azure Blob Storage - Private
Azure
10 01
10 01
10 01
Software images
1
API call to generate SAS URL
2
Return SAS URL
5
Receive Job ID for device upgrade
Poll upgrade status by Job ID
3
API call - Add software:
Type: Remote, SAS URL, Version
4
API call – Upgrade device:
Device ID, Version
6
Netconf
Install software from HTTP
SAS URL
Poll upgrade progress
7
Download image with HTTP
SAS URL
8
Install image or install and activate

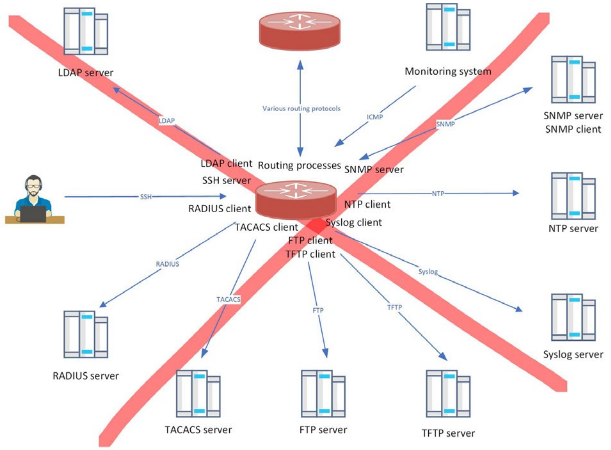
- Automatic routers upgrade

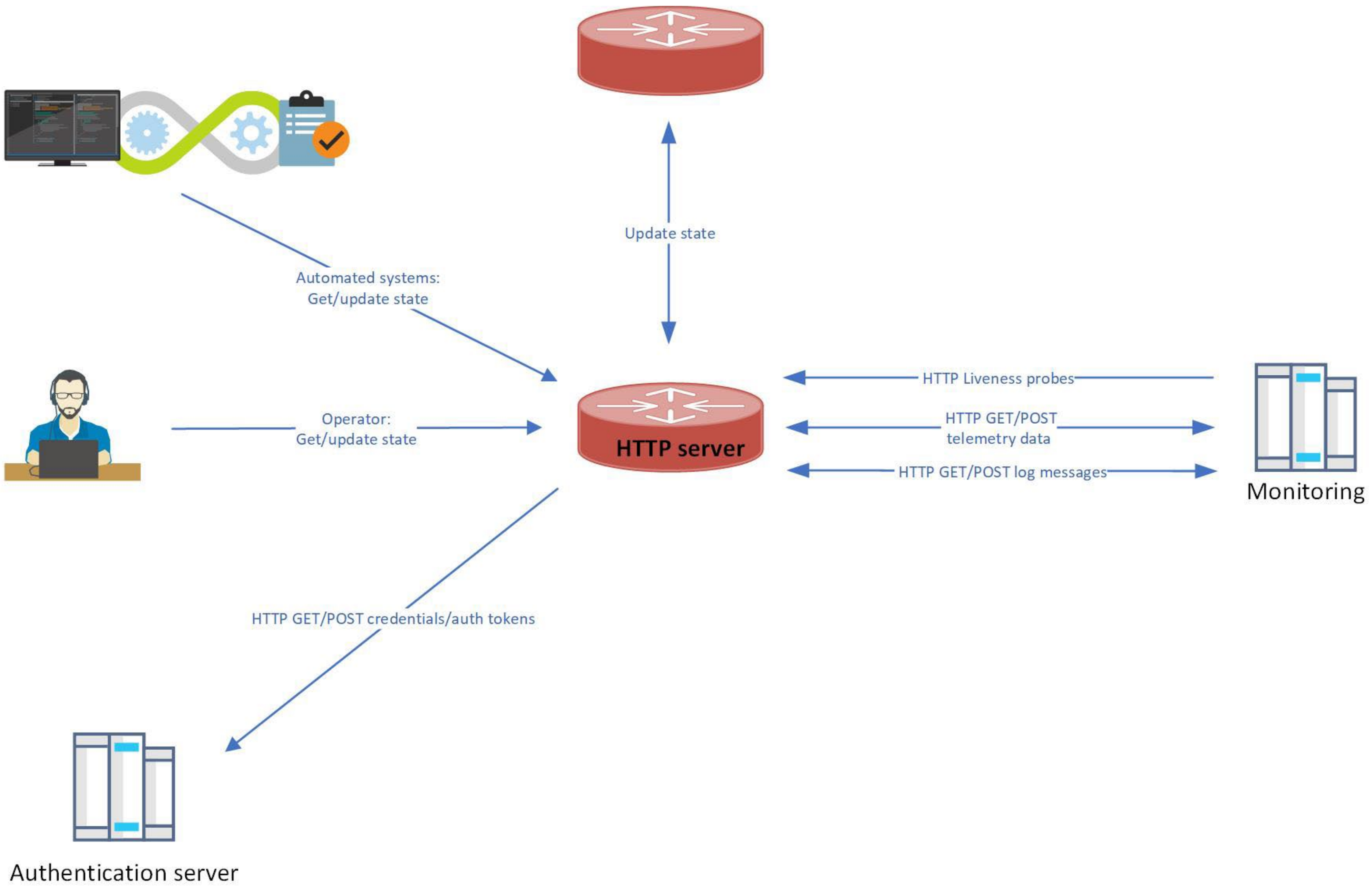
make API call to **Azure to** generate unique URL

make API call to **vManage**

instructing it to start upgrade process of routers

API is getting more and more common in modern networks





Different APIs

As mentioned before, API is a broad term, but we'll use **Web API** which is the most common meaning of the term API

There two main types of Web APIs:

- SOAP

  still used, but not very common these days. W3C Standard, transport-independent, uses XML

- REST

  the key part of most modern web applications, most commonly used, leverages HTTP transport and JSON as data format

https://www.upwork.com/resources/soap-vs-rest-a-look-at-two-different-api-styles

We'll focus of **REST API.**

What is REST API

REST is actually not a protocol, but software architecture style and Web service APIs that adhere to the REST architectural constraints are called RESTful APIs

It's worth mentioning some of these constraints here, please make sure to check all six of them: https://en.wikipedia.org/wiki/Representational_state_transfer

Uniform Interface:
Resource identification in requests, for example, https://vmanage/device or https://vmanage/policy/policy-id - here **device** and **policy-id** are called **resources**

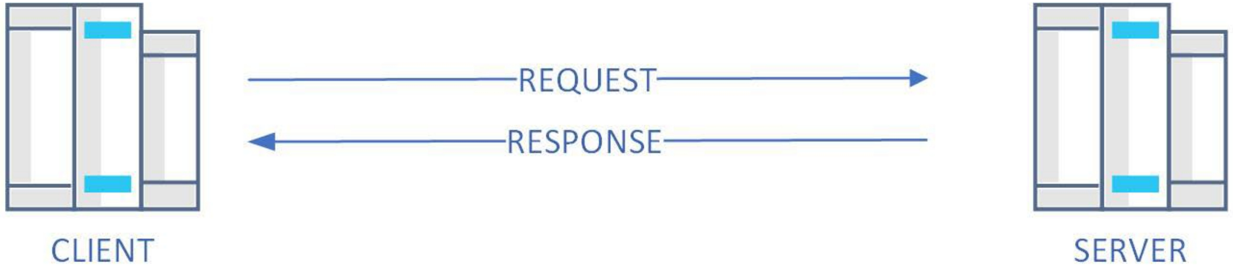
Manipulation of Resources Using **Representations** - client can request data in JSON or XML format, you can see this later in Headers: Accept: application/json
This data (text) we receive doesn't reflect server's internal data or logic; they are independent.

REST API – Client-Server

Client-Server: REST application should have a client-server architecture.

Client doesn’t need to know anything about business logic and server doesn’t need to know anything about client, as long as a request is in correct format.

A client can be anything – mobile app, IoT device, web page, router, etc….



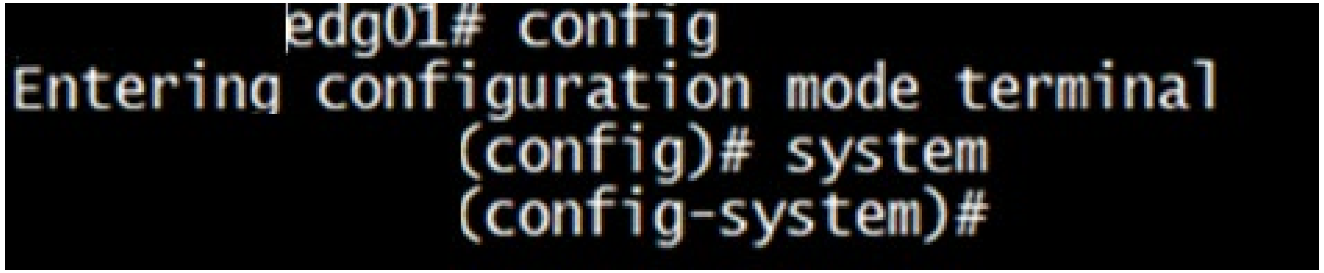
An example of system interaction **not** using client-server architecture – routing protocols or peer-to-peer networks

REST API - Stateless

Statelessness

Each request from any client contains all the information necessary to service the request, and the session state is held in the client.

Compare to CLI:



```
        edg01# config
Entering configuration mode terminal
              (config)# system
              (config-system)#
```

So the command you are executing depends on the context you're in, and the server tracks the current context

In REST API the **client** is responsible to provide all the necessary information, so the server can fulfill the request. This includes URI, headers, authentication credentials and any parameters.

Each request is processed independently from any other

As a result there is no concept such as 'commit' – the request is processed immediately

REST API messages

Four most commonly used messages (also called as verbs):

- **GET** – get list of resources, or details about particular resource. Remember resources are specified in a request URI, for example:

  GET *https://router/configuration/acls* - get router's access-list

  GET *https://router/configuration/acls/id-123456* - get details about a particular access-list

Note there can't be request methods like GET-ROUTER-ACCESS-LIST

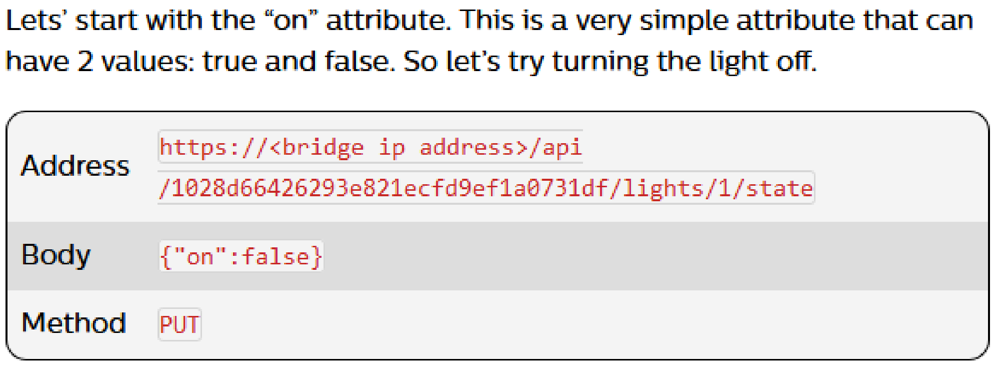
- **POST** - create a resource, for example, create a new access-list

- **PUT** - update a resource

- **DELETE** – deletes resource

Less commonly used, we won't consider them:

PATCH, HEAD, OPTIONS



Lets' start with the "on" attribute. This is a very simple attribute that can have 2 values: true and false. So let's try turning the light off.

Address	`https://<bridge ip address>/api/1028d66426293e821ecfd9ef1a0731df/lights/1/state`
Body	`{"on":false}`
Method	`PUT`

https://en.wikipedia.org/wiki/Hypertext_Transfer_Protocol#Request_methods

Request URL

https://api.exchangeratesapi.io/latest?base=AUD&symbols=USD

^ ^ ^ ^ ^ ^ ^ ^ ^ ^ ^ ^ ^ ^ ^   ^ ^ ^ ^   ^ ^ ^ ^ ^ ^ ^ ^ ^ ^ ^ ^ ^ ^ ^ ^

Host Resource Parameters

Host + Resource is **API endpoint**

Always contain **Headers**, most common:

- Authorization – username/password, key, bearer, etc, we'll cover it later
- Content-Type - data format of the request body, for example *application/json*
- Accept – data format requested from the server, for example *application/json*

Optionally can have **Payload** (body) – can be json, binary (for file upload), etc

Response

Always contains response code and status message, for example:

Code	Message	Meaning	
200	OK	Success	- 2xx responses - Success
401	Unauthorized	Authentication missing or incorrect	- 4xx responses – Client error
404	Not Found	Resource not found/wrong URL	
500	Internal Server Error		- 5xx responses – Server error
503	Service Unavailable	Server is unable to complete request	

See:

https://en.wikipedia.org/wiki/List_of_HTTP_status_codes

Response

Always contain **Headers**

Always contain **Body – this is what we want to receive from the server (if request completed)**

Most common is JSON:

```
{
    "base": "AUD",
    "date": "2020-11-04",
    "rates": {
        "USD": 0.7170561605
    }
}
```

Client’s (requester’s) can then parse this data and do something with them

REST API message formats - GET

```
Request      Verb -------->           > GET http://worldtimeapi.org/api/ip HTTP/1.1
Headers ------------------>           > User-Agent: curl/7.29.0
                                      > Host: worldtimeapi.org
                                      > Accept: */*
                                      > Proxy-Connection: Keep-Alive
                                      >
 Response   Status ----------->       < HTTP/1.1 200 OK                    <<<<  Status code and text
 Headers      ----------------->      < Access-Control-Allow-Credentials: true
                                      < Access-Control-Allow-Origin: *
                                      < Access-Control-Expose-Headers:
                                      < Cache-Control: max-age=0, private, must-revalidate
                                      < Content-Length: 399
                                      < Content-Type: application/json; charset=utf-8
                                      < Cross-Origin-Window-Policy: deny
                                      < Date: Wed, 04 Nov 2020 23:31:47 GMT
                                      < X-Request-Id: b89182b8-a667-4c67-90a5-54f0aacace5c
                                      {"abbreviation":"AEDT","client_ip":"118.127.80.40","datetime":"2020-11-
 Response Body (Payload)              05T10:31:48.549066+11:00","day_of_week":4,"day_of_year":310,"dst":true,"dst_from":
                                      "2020-10-03T16:00:00+00:00","dst_offset":3600,"dst_until":"2021-04-
                                      03T16:00:00+00:00","raw_offset":36000,"timezone":"Australia/Sydney","unixtime":16
                                      04532708,"utc_datetime":"2020-11-
                                      04T23:31:48.549066+00:00","utc_offset":"+11:00","week_number":45}
```

How to consume API

What you need to know:

API documentation - Normally API contains documentation and examples, so the first step is to read the doc to see auth and methods are supported, what format is expected, etc

https://wheretheiss.at/w/developer http://worldtimeapi.org/ https://ipwhois.io/documentation

https://developers.facebook.com/docs/graph-api/overview/ https://<vManage>/apidocs

Know API endpoint which consists of:

- Server URL - API server – just a Web server, lightweight, as there no GUI, for example, NGNIX or Flask. In prod is usually behind a load-balancer
- Resource - path like /device/interface/id or /client/payments/

Client

- Your browser
- CURL
- Postman
- Programming language

Practice time! – Basic GET requests with cURL

Raw ouput:

```
curl -v "http://worldtimeapi.org/api/ip"
```

```
curl -v "http://worldtimeapi.org/api/ip"
```

Pretty output:

```
curl -v "http://worldtimeapi.org/api/ip" | python -m json.tool
```

No parameters:

```
curl -v https://ipwhois.app/json/8.8.8.8| python -m json.tool
```

Let’s include parameters:

```
curl -v https://ipwhois.app/json/8.8.8.8?objects=country,city,timezone | python -m json.tool
```

Single parameter:

```
curl https://api.exchangeratesapi.io/latest?base=AUD | python -m json.tool
```

Multiple parameters: NOTE – here URI is in ‘’ as & is a command in Linux

```
curl 'https://api.exchangeratesapi.io/latest?base=AUD&symbols=USD'  | python -m json.tool
```

More complex response:

```
curl "https://endpoints.office.com/endpoints/worldwide?clientrequestid=b10c5ed1-bad1-445f-b386-b919946339a7" -x 172.21.40.146:8080
```

Body in the request:

```
curl -H 'Content-Type: application/json' -d '{"text": "Test message via Webhook"}' https://outlook.office.com/webhook/82e433c98-a978-465f-8254-9d541ee73c/IncomingWebhook/cb6d6416f49aaba3bd52357266
```

Webhooks

Last command we done is made to MS Teams webook

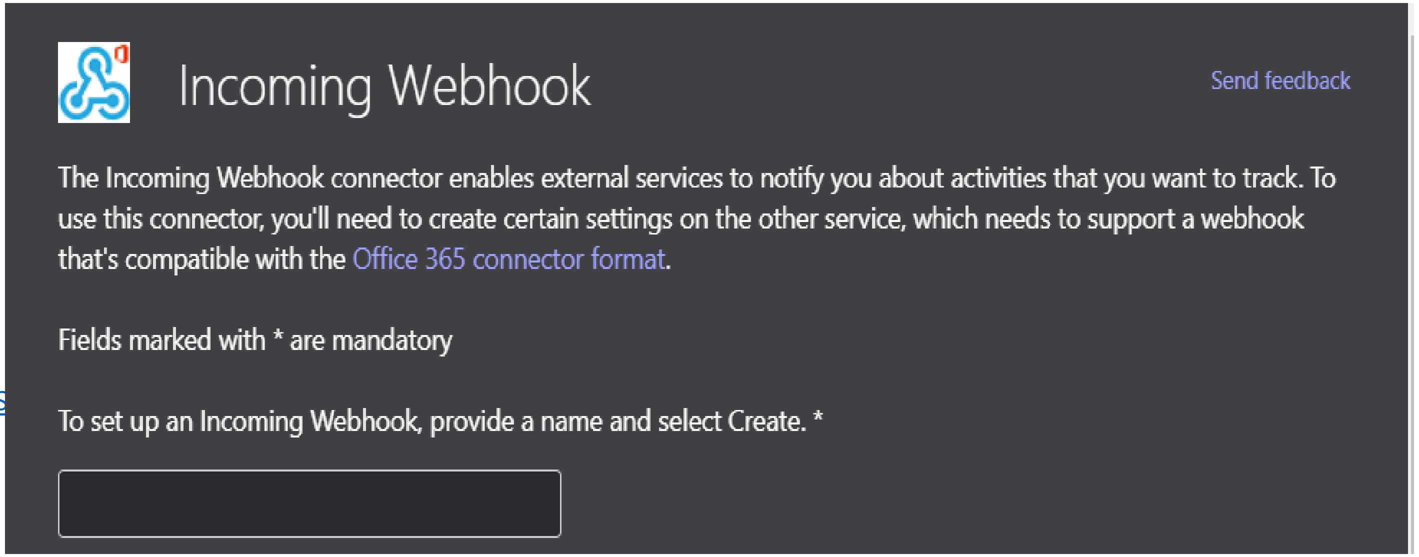
Webhook = **API endpoint**

- Used to communicate to the application in the **real-time**
- In most cases works like a **trigger** for the app to do some **actions**
- Usually consumed with POST commands

In our case this is webhook to MS Teams app which posts messages to the channel

The same approach for Webex or Slack

https://developer.webex.com/docs/api/guides/webhooks



Incoming Webhook

The Incoming Webhook connector enables external services to notify you about activities that you want to track. To use this connector, you'll need to create certain settings on the other service, which needs to support a webhook that's compatible with the Office 365 connector format.

Fields marked with * are mandatory

To set up an Incoming Webhook, provide a name and select Create. *

https://docs.microsoft.com/en-us/microsoftteams/platform/webhooks-and-connectors/how-to/connectors-using#example-connector-message
https://docs.github.com/en/free-pro-team@latest/developers/webhooks-and-events/about-webhooks

Controllers

It’s possible to make API calls individually to each router, node, or lighting bulb.

However, this is not scalable, not very secure architecture, so the most common method is to use a **controller.**

Controllers offers consumable API, doing authentication, RBAC, etc. This type of API exposed to be used is called **northbound API.**

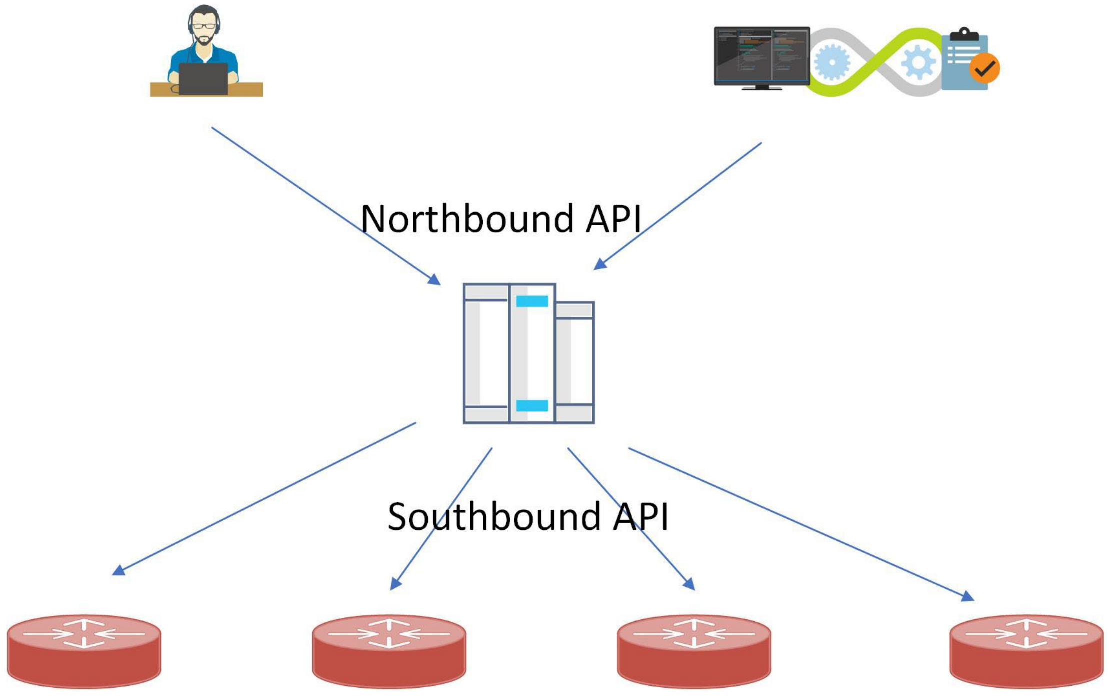
Controllers maintain connections to identify the operational state of controlled devices, push configuration, pull logs. Controlled devices also can push alerts, state changes. This type of comms is called **southbound API**. This API can be proprietary, maybe even SSH or SNMP.

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.



These APIs and the controller what makes **Software Defined** in any SDN including SD-WAN

Summary and home work

Today was an introduction session and we learned:

- What is API
- What is REST API
- Request and responses
- How to use cURL

Homework: ☺

- Read more about REST API
- Think of use cases
- Ask questions, I'm happy to answer them

Get familiar with Meraki API - Explore resources, requests and responses

https://documentation.meraki.com/General_Administration/Other_Topics/The_Cisco_Meraki_Dashboard_API

vCentre API

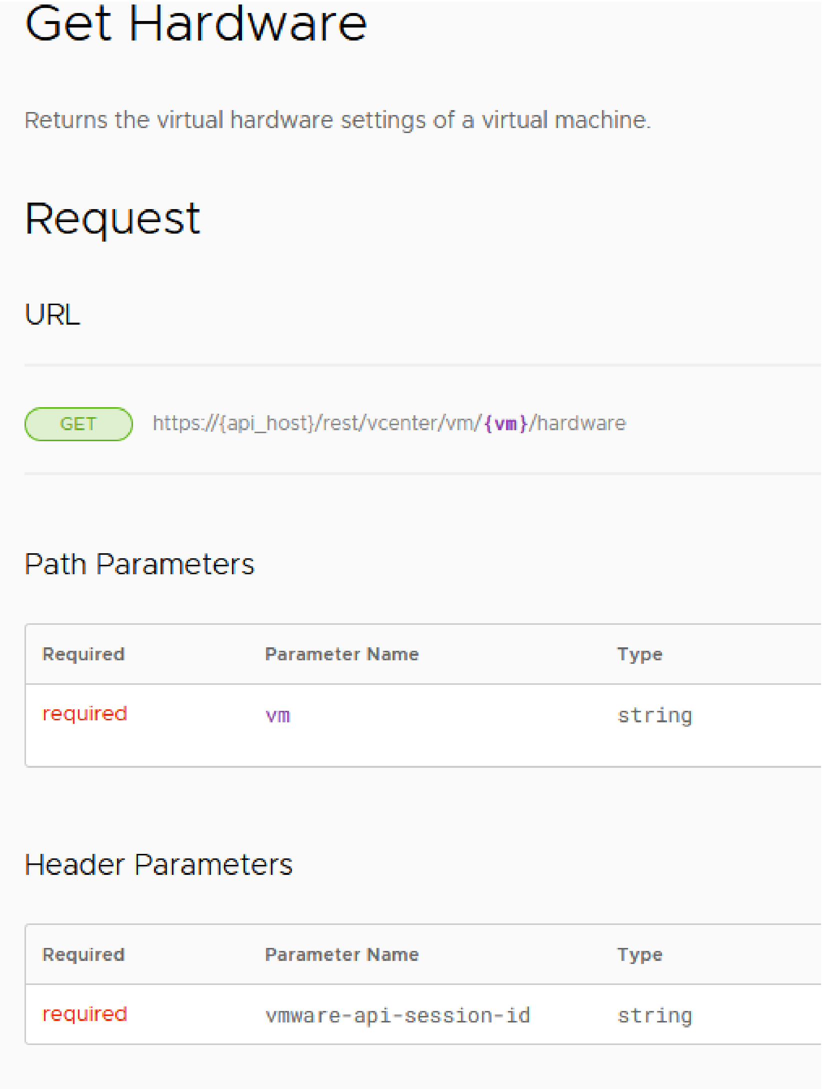
Have a look at the REST API documentation

- https://developer.vmware.com/docs/vsphere-automation/latest/vcenter/rest/vcenter/vm/vm/hardware/get/

This is a typical API documentation

Notice how the documentation looks like, example of

Requests, Responses, Parameters, Headers, Errors



Get Hardware

Returns the virtual hardware settings of a virtual machine.

Request

URL

GET https://{api_host}/rest/vcenter/vm/{vm}/hardware

Path Parameters

Required	Parameter Name	Type
required	vm	string

Header Parameters

Required	Parameter Name	Type
required	vmware-api-session-id	string

What we’ll do next time

- Explore and try different types of authentication
- Practice with Postman using free public API and vManage

- After that we’ll focus on Python