Internet of Things

Programming

Week 7 – APIs and Webservers

Anas Dawod

adawod@swin.edu.au

Swinburne University of Technology

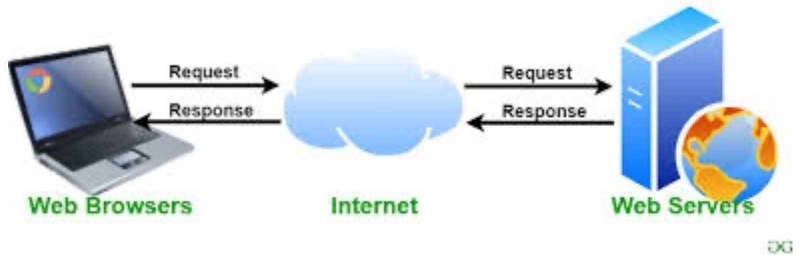
April 2023

WEB SERVERS

A Web server is responsible for accepting HTTP requests from web clients and serving them HTTP responses

It is an interface that describes a collection of operations that are network accessible through standardized XML messaging

The primary function of a web server is to store, process and deliver web pages to clients



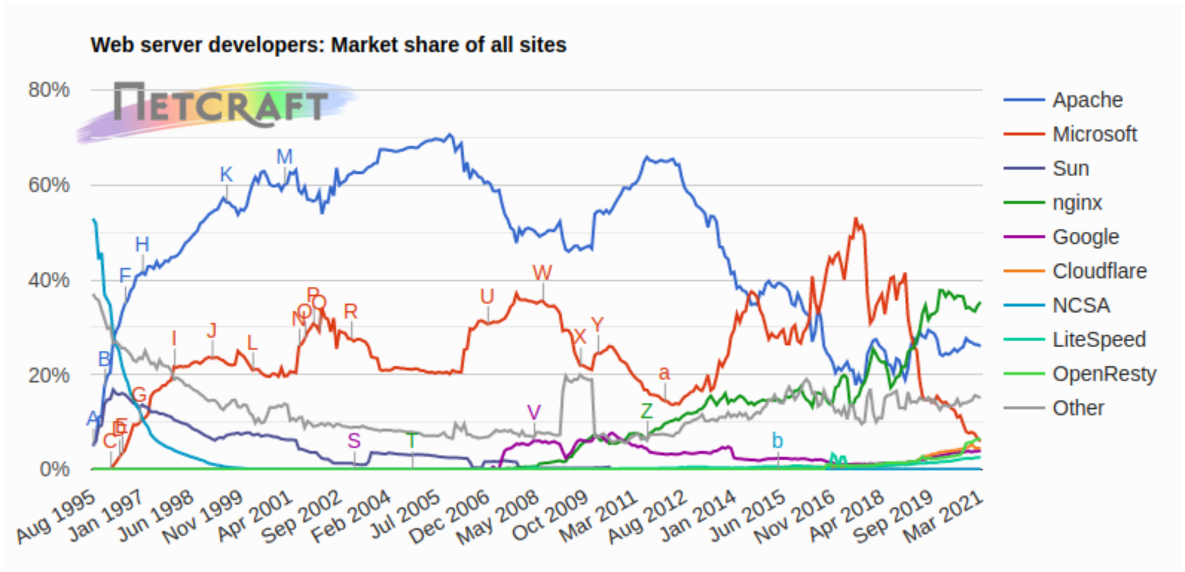
Source: geeksforgeeks.org

Important to know- Web servers can be stateless

What does this mean?

Popular Web Servers

Developer	February 2021	Percent	March 2021	Percent	Change
nginx	415,900,479	34.54%	419,637,923	35.34%	0.80
Apache	316,992,638	26.32%	308,509,042	25.98%	-0.34
OpenResty	76,623,440	6.36%	77,819,490	6.55%	0.19
Microsoft	78,331,379	6.50%	70,826,342	5.96%	-0.54



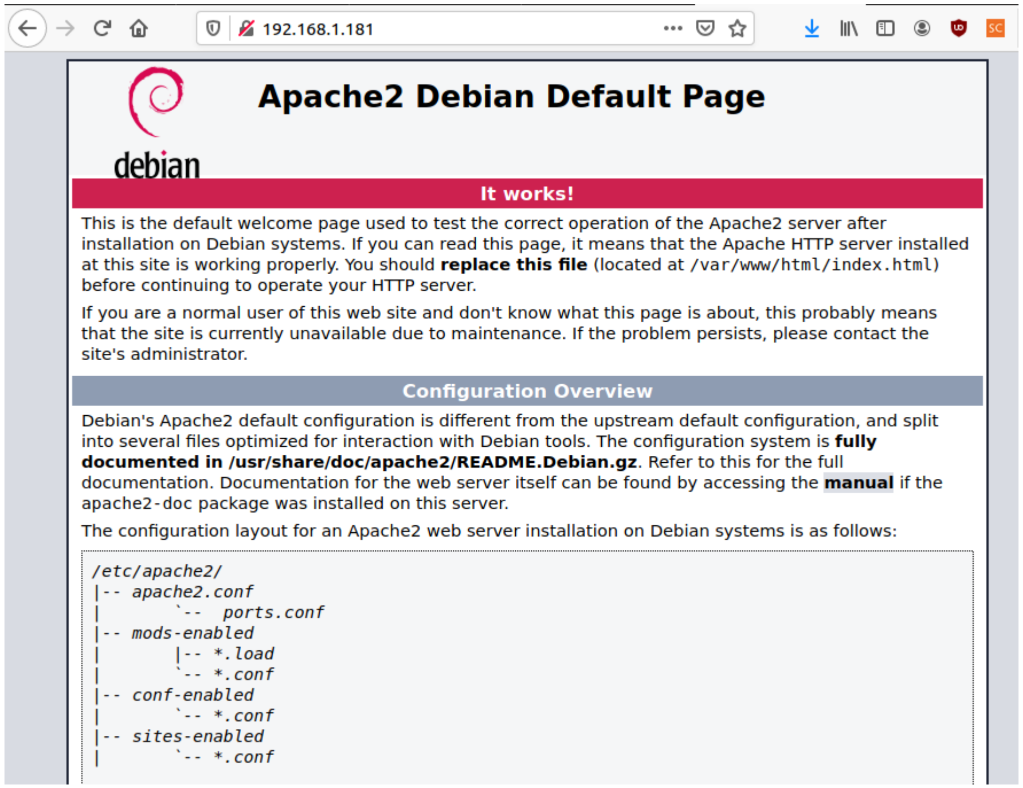
https://news.netcraft.com/archives/category/web-server-survey/

Apache Web Server

- The Apache Web server has been the most popular and widely used Web server for the last few years.
- Apache is free and open-source.
- The Apache Web server has almost has great modularity which allows it to be integrated with numerous other applications
- Apache is developed and maintained by an open community of developers under the auspices of the Apache Software Foundation

Apache Web Server Installation

- sudo apt-get install apache2
- systemctl start apache2.service
- systemctl enable apache2.service



Apache2 Debian Default Page

It works!

This is the default welcome page used to test the correct operation of the Apache2 server after installation on Debian systems. If you can read this page, it means that the Apache HTTP server installed at this site is working properly. You should **replace this file** (located at `/var/www/html/index.html`) before continuing to operate your HTTP server.

If you are a normal user of this web site and don't know what this page is about, this probably means that the site is currently unavailable due to maintenance. If the problem persists, please contact the site's administrator.

Configuration Overview

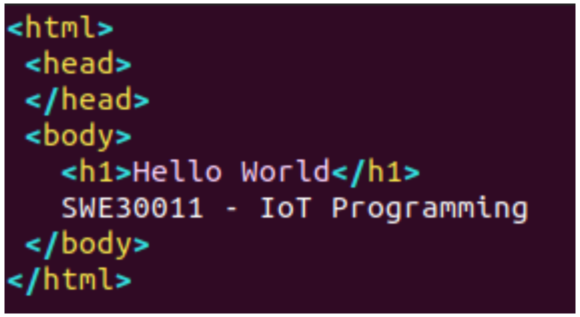
Debian's Apache2 default configuration is different from the upstream default configuration, and split into several files optimized for interaction with Debian tools. The configuration system is **fully documented in /usr/share/doc/apache2/README.Debian.gz**. Refer to this for the full documentation. Documentation for the web server itself can be found by accessing the **manual** if the apache2-doc package was installed on this server.

The configuration layout for an Apache2 web server installation on Debian systems is as follows:

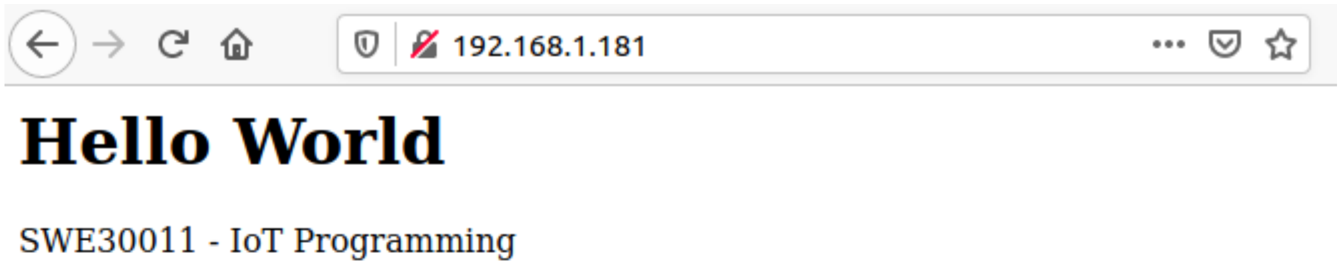
```
/etc/apache2/
|-- apache2.conf
|       `--  ports.conf
|-- mods-enabled
|       |-- *.load
|       `-- *.conf
|-- conf-enabled
|       `-- *.conf
|-- sites-enabled
|       `-- *.conf
```

Apache Web Server Installation

- sudo vim /var/www/html/index.html



```
<html>
 <head>
 </head>
 <body>
   <h1>Hello World</h1>
   SWE30011 - IoT Programming
 </body>
</html>
```



192.168.1.181

Hello World

SWE30011 - IoT Programming

Apache Web Server Service Manager

- systemctl start apache2.service
- systemctl restart apache2.service
- systemctl stop apache2.service
- systemctl enable apache2.service
- systemctl disable apache2.service

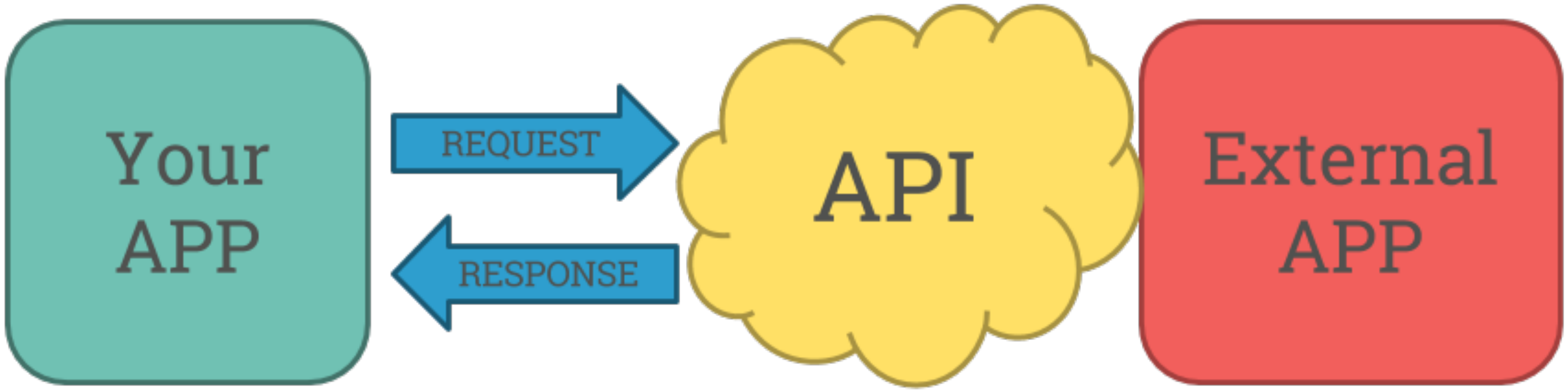
nginx

- A web server which can also be used as a reverse proxy, load balancer, mail proxy and HTTP cache.
- Can handle a high volume of connections.
- The software was created by Igor Sysoev and first publicly released in 2004.
- A company of the same name was founded in 2011 to provide support and Nginx plus paid software.
- Nginx is free and open-source software, released under the terms of a BSD-like license.
- A large fraction of web servers use NGINX often as a load balancer.

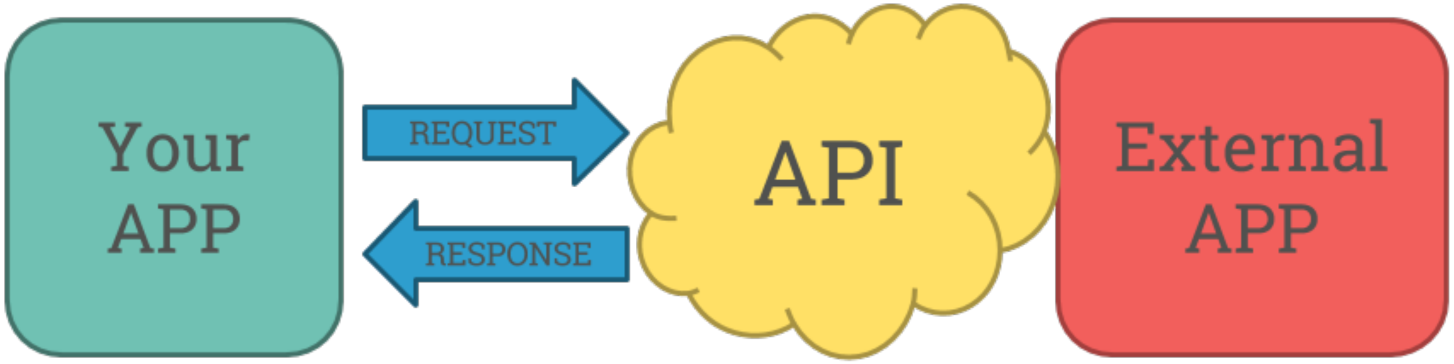
How to choose a Web Server

- Performance
- Reliability
- Easy of use
- Support
- Price
- Security
- Capabilities

Application Programming Interface



Application Programming Interface



- An API is a set of functions that allows your own application to access features of another existing application/system.
- Example of Web APIs:
  - Automated posting of Tweets.
  - Text message automation.
  - Getting location data from Google Maps.
  - Using weather data from BOM or OpenWeatherMap.
  - Etc.

REST ARCHITECTURE

REST

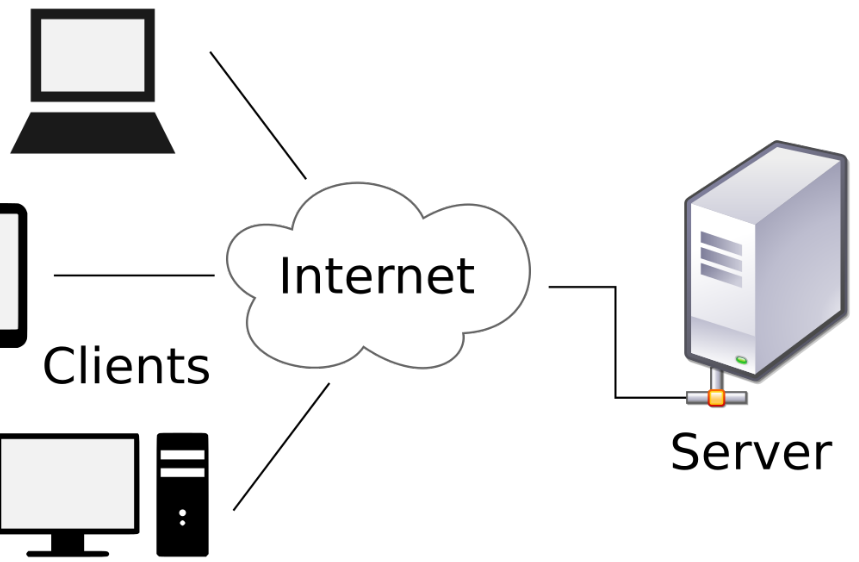
- REST: Representational state transfer
- Software architectural style using a subset of HTTP protocols
- Used to create interactive applications that use Web services
- Web services following REST guidelines are called RESTful.

REST

- simplicity, thanks to HTTP protocols.
- facilitate client-server communications and architectures.
- use a single uniform interface
- optimized for the web
- is known for excellent performance and scalability

Client-Server Communication Architecture

- The components of a RESTful system are either client or server
- The same node might act as a client in some communications and a server in others
- Web browsers are always in the client role and have the initiative to issue requests.



https://en.wikipedia.org/wiki/Client%E2%80%93server_model

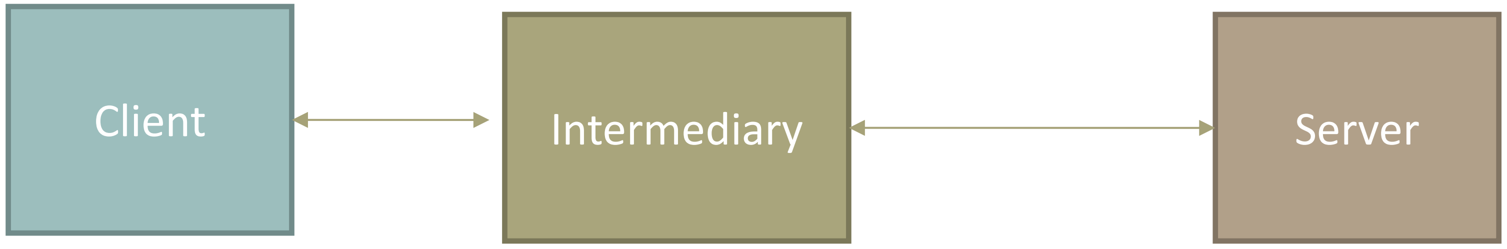
RESTful API

- The Representational State Transfer (REST) architectural style is a set of guidelines and best practices for building distributed hypermedia systems.
- A set of constraints, which when fulfilled enable desirable properties for distributed software systems such as scalability and modifiability.
- When REST principles are applied to the design of a system, the result is often called RESTful and in particular an API following these principles is called a RESTful API.
- Different protocols can be used with RESTful systems, the most common protocols are HTTP and CoAP
- Since RESTful APIs are often simple and lightweight, they are a good fit for various IoT applications

https://tools.ietf.org/id/draft-keranen-t2trg-rest-iot-05.html

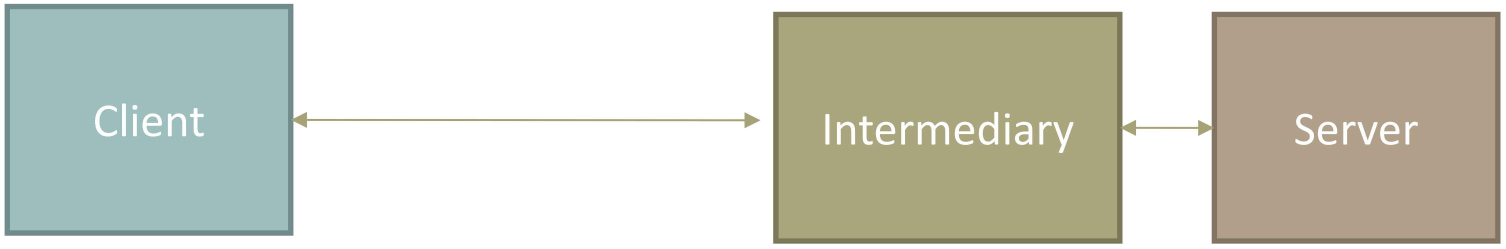
Communication with Forward Proxy

- Intermediaries (such as forward proxies, reverse proxies, and gateways) implement both roles of client and servers
- Only forward requests to other intermediaries or origin servers.
- Can also translate requests to different protocols, for instance, as CoAP-HTTP cross-proxies.



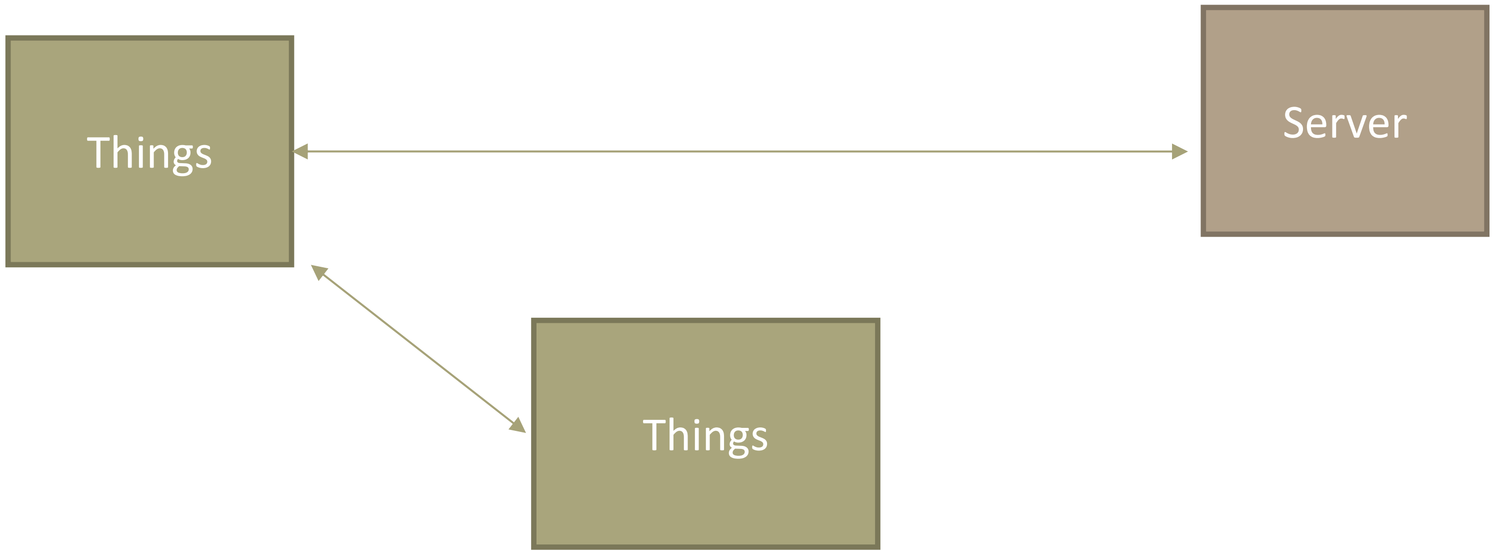
Communication with Reverse Proxy

- Reverse proxies are usually imposed by the origin server
- Can provide an interface for non-RESTful services such as legacy systems or alternative technologies such as Bluetooth ATT/GATT
- Reverse proxies are usually called gateways



IoT RESTful

- Nodes in IoT systems often implement both roles
- They can take the initiative as a client (e.g., to register with a directory, such as CoRE Resource Directory)
- Interact with another *thing*
- Act as origin server (e.g., to serve sensor values or provide an actuator interface)



Uniform Resource Identifiers (URIs)

- An important part of RESTful API design is to model the system as a set of resources whose state can be retrieved and/or modified and where resources can be potentially also created and/or deleted.
- Uniform Resource Identifiers (URIs) are used to indicate a resource for interaction, to reference a resource from another resource, to advertise or bookmark a resource, or to index a resource by search engines.
- Consists of a hierarchical sequence of five components: scheme, host, path, query, and fragment
- For RESTful IoT applications, typical schemes include “https”, “coaps”, “http”, and “coap”

```
foo:// example.com:8042/over/there?name=ferret#nose
 \_/   \______________/\_________/ \_________/ \__/
  |           |             |           |       |
scheme      host          path        query  fragment
```

Resource Representation

- Clients can retrieve the resource state from an origin server or manipulate resource state on the origin server by transferring resource representations
- Resource representations have a media type that tells how the representation should be interpreted by identifying the representation format used.
- Typical media types for IoT systems include:
  - “text/plain” for simple UTF-8 text
  - “application/octet-stream” for arbitrary binary data
  - “application/json” for the JSON format
  - “application/senml+json” for Sensor Markup Language (SenML) formatted data
  - “application/cbor” for CBOR
  - “application/exi” for EXI
- Full list of media types:

  https://tools.ietf.org/id/draft-keranen-t2trg-rest-iot-05.html#IANA-media-types

XML

- XML stands for eXtensible Markup Language
- XML is a markup language much like HTML
- XML was designed to store and transport data
- XML was designed to be self-descriptive
- XML is a W3C Recommendation

```
<note>
  <to>Tove</to>
  <from>Jani</from>
  <heading>Reminder</heading>
  <body>Don't forget me this
weekend!</body>
</note>
```

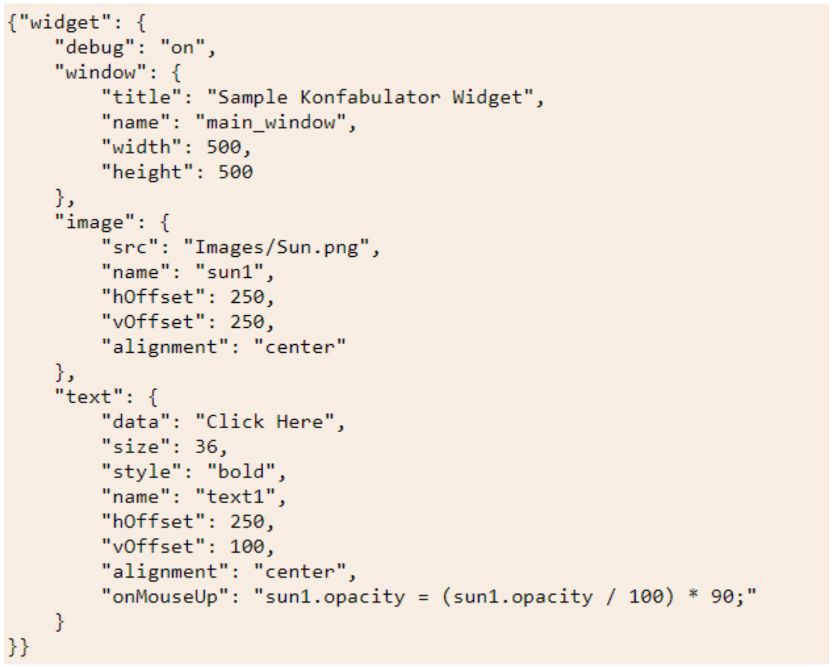
JSON

- JavaScript Object Notation (JSON)
- Open-standard file format that uses human-readable text to transmit data
- JSON is a language-independent data format

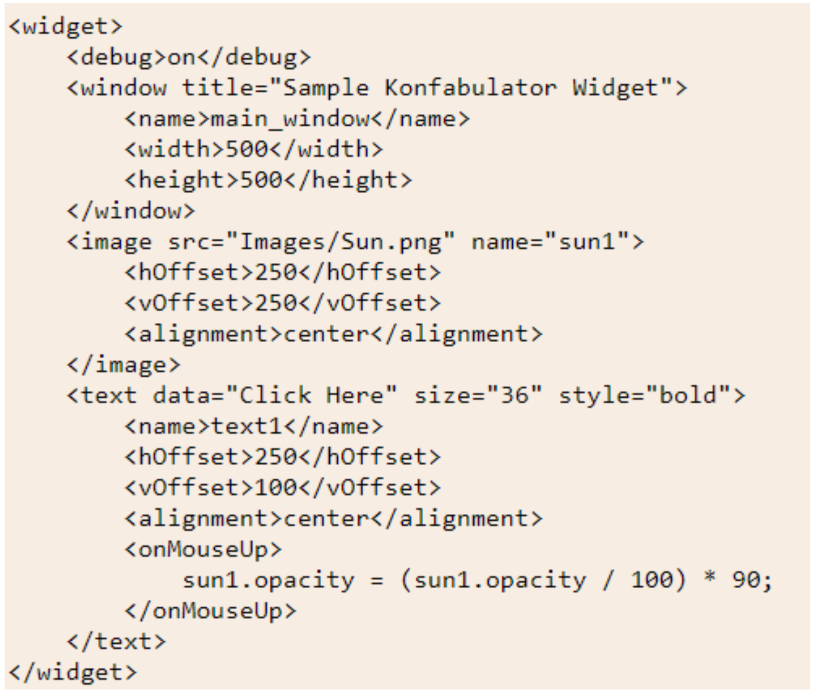
```json
{
  "firstName": "John",
  "lastName": "Smith",
  "isAlive": true,
  "age": 27,
  "address": {
    "streetAddress": "21 2nd Street",
    "city": "New York",
    "state": "NY",
    "postalCode": "10021-3100"
  },
  "phoneNumbers": [
    {
      "type": "home",
      "number": "212 555-1234"
    },
    {
      "type": "office",
      "number": "646 555-4567"
    },
    {
      "type": "mobile",
      "number": "123 456-7890"
    }
  ],
  "children": [],
  "spouse": null
}
```

JSON vs. XML

- Both JSON and XML are "self describing" (human readable)
- Both JSON and XML are hierarchical (values within values)
- Both JSON and XML can be parsed and used by lots of programming languages
- XML is much more difficult to parse than JSON.



```
{"widget": {
    "debug": "on",
    "window": {
        "title": "Sample Konfabulator Widget",
        "name": "main_window",
        "width": 500,
        "height": 500
    },
    "image": {
        "src": "Images/Sun.png",
        "name": "sun1",
        "hOffset": 250,
        "vOffset": 250,
        "alignment": "center"
    },
    "text": {
        "data": "Click Here",
        "size": 36,
        "style": "bold",
        "name": "text1",
        "hOffset": 250,
        "vOffset": 100,
        "alignment": "center",
        "onMouseUp": "sun1.opacity = (sun1.opacity / 100) * 90;"
    }
}}
```



```
<widget>
    <debug>on</debug>
    <window title="Sample Konfabulator Widget">
        <name>main_window</name>
        <width>500</width>
        <height>500</height>
    </window>
    <image src="Images/Sun.png" name="sun1">
        <hOffset>250</hOffset>
        <vOffset>250</vOffset>
        <alignment>center</alignment>
    </image>
    <text data="Click Here" size="36" style="bold">
        <name>text1</name>
        <hOffset>250</hOffset>
        <vOffset>100</vOffset>
        <alignment>center</alignment>
        <onMouseUp>
            sun1.opacity = (sun1.opacity / 100) * 90;
        </onMouseUp>
    </text>
</widget>
```

w3schools.com and json.org

CRUD Operations

- In computer programming, Create, Read, Update, and Delete (CRUD) are the four basic functions of persistent storage.
- The term was likely first popularised by James Martin in his 1983 book Managing the Data-base Environment.
- CRUD principles of persistent data storage are often mapped to REST.

HTTP/CoAP Methods

- GET: to read a resource.
- POST: to create a resource.
- PUT: to update a resource.
- DELETE: to delete a resource.

https://tools.ietf.org/html/draft-irtf-t2trg-rest-iot-07

GET

- The GET method is used to read (or retrieve) a representation of a resource.
- GET returns a representation in XML or JSON and an HTTP response code.
- GET requests are used only to read data and not change it. Therefore, when used this way, they are considered safe. That is, they can be called without risk of data modification or corruption.
- GET is idempotent, which means that making multiple identical requests ends up having the same result as a single request.

https://www.restapitutorial.com/lessons/httpmethods.html

POST

- The POST verb is most-often utilised to create new resources (subordinate resources).
- When creating a new resource, POST to the parent and the service takes care of associating the new resource with the parent, assigning an ID (new resource URI).
- On successful creation, return HTTP status 201, returning a Location header with a link to the newly-created resource with the 201 HTTP status.
- Making two identical POST requests will most-likely result in two resources containing the same information.

https://www.restapitutorial.com/lessons/httpmethods.html

PUT

- PUT is most-often utilised for update capabilities, PUT-ing to a known resource URI with the request body containing the newly-updated representation of the original resource.
- PUT can also be used to create a resource in the case where the resource ID is chosen by the client instead of by the server.
- On successful update, return 200 (or 204 if not returning any content in the body) from a PUT. If using PUT for create, return HTTP status 201 on successful creation.
- Put modifies the state of the server (not a safe operation), but it is idempotent (Updating a resource with the same call, the resource should have the same state)

https://www.restapitutorial.com/lessons/httpmethods.html

DELETE

- DELETEs a resource identified by an URI.
- On successful deletion, return HTTP status 200 (OK) along with a response body, or return HTTP status 204 (NO CONTENT) with no response body.
- DELETE operations are idempotent. If you DELETE a resource, it's removed. Repeatedly calling DELETE on that resource ends up the same

https://www.restapitutorial.com/lessons/httpmethods.html

Rest Constraints

- The REST architectural style defines a set of constraints for the system design. When all constraints are applied correctly, REST enables architectural properties of key interest:
  - Performance
  - Scalability
  - Reliability
  - Simplicity
  - Modifiability
  - Visibility
  - Portability

Client-Server

- RESTful system components have clear roles in every interaction.
- Clients have the initiative to issue requests, intermediaries can only forward requests, and servers respond requests, while origin servers are the ultimate recipient of requests that intent to modify resource state.
- This improves simplicity and visibility, as it is clear which component started an interaction.
- Furthermore, it improves modifiability through a clear separation of concerns.

Stateless

- The Stateless constraint requires messages to be self-contained
- They must contain all the information to process it, independent from previous messages. This allows to strictly separate the client state from the resource state
- This improves scalability and reliability, since servers or worker threads can be replicated. It also improves visibility because message traces contain all the information to understand the logged interactions

Cache

- This constraint requires responses to have implicit or explicit cache-control metadata.
- This enables clients and intermediary to store responses and re-use them to locally answer future requests.
- The cache-control metadata is necessary to decide whether the information in the cached response is still fresh or stale and needs to be discarded.
- Cache improves performance, as less data needs to be transferred and response times can be reduced significantly.
- Less transfers also improves scalability, as origin servers can be protected from too many requests.
- Local caches furthermore improve reliability, since requests can be answered even if the origin server is temporarily not available.

Uniform Interface

- All RESTful APIs use the same, uniform interface independent of the application.
- This simple interaction model is enabled by exchanging representations and modifying state locally, which simplifies the interface between clients and servers to a small set of methods to retrieve, update, and delete state – which applies to all applications.
- A REST interface is fully defined by:
  - URIs to identify resources
  - representation formats to represent (and retrieve and manipulate) resource state
  - self-descriptive messages with a standard set of methods (e.g., GET, POST, PUT, DELETE with their guaranteed properties)

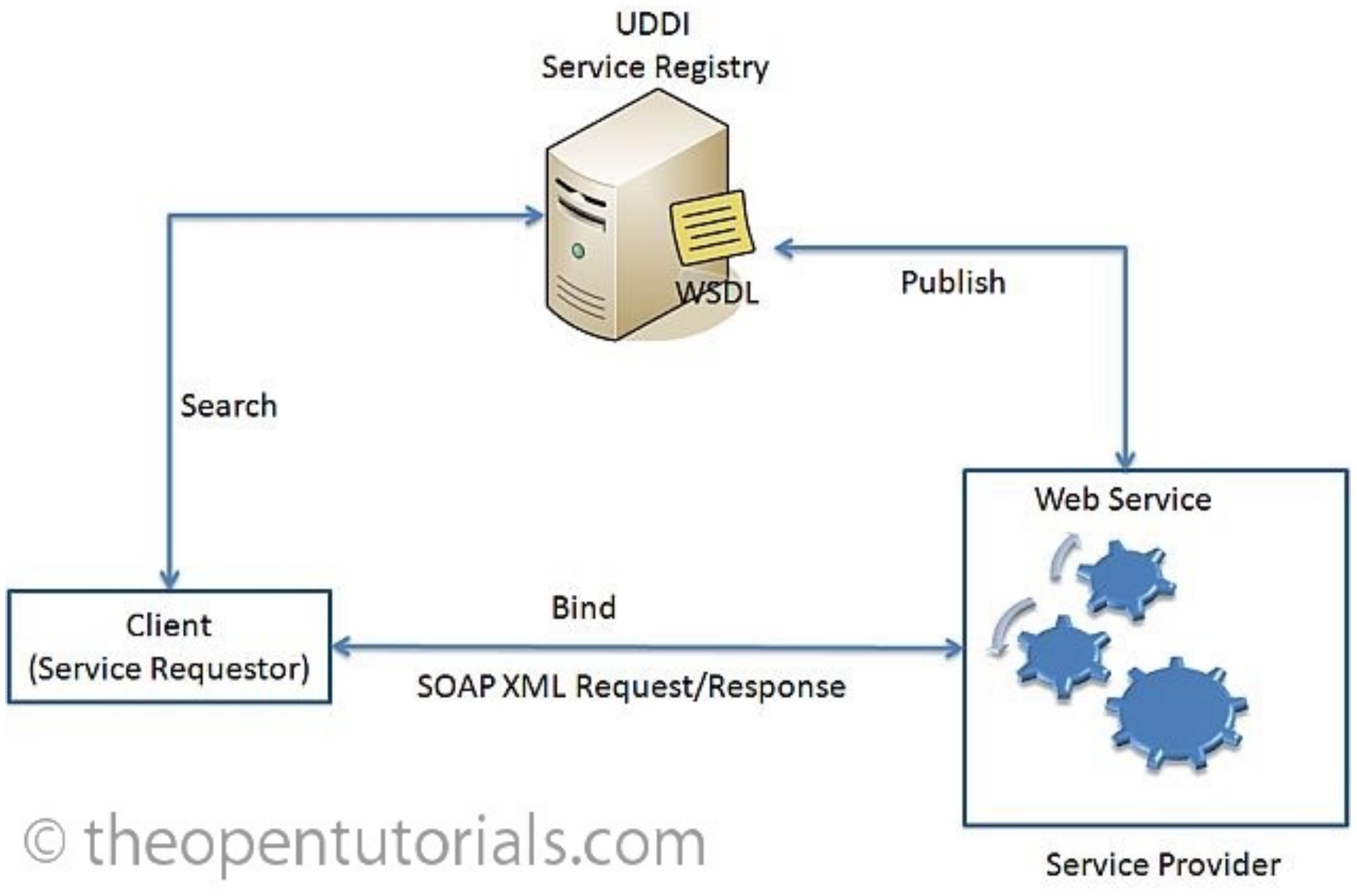
Layered System

- This constraint enforces that a client cannot see beyond the server with which it is interacting
- A layered system is easier to modify, as topology changes become transparent.
- Helps scalability, as intermediaries such as load balancers can be introduced without changing the client side.
- The clean separation of concerns helps with simplicity.

Code on Demand

- This principle enables origin servers to ship code to clients.
- Code-on-Demand improves modifiability, since new features can be deployed during runtime (e.g., support for a new representation format).

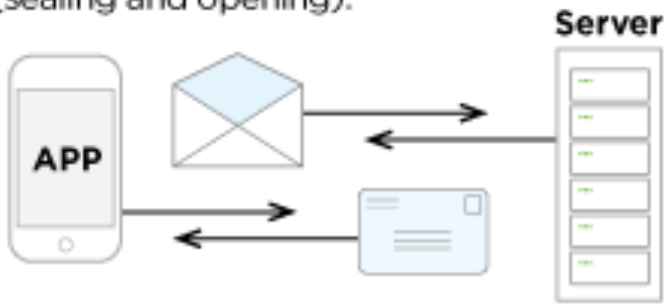
SOAP (Simple Object Access Protocol)



SOAP vs. REST APIs

SOAP is like using an envelope

Extra overhead, more bandwidth required, more work on both ends (sealing and opening).



REST is like a postcard

Lighterweight, can be cached, easier to update.

SOAP vs. REST Comparison: Which is Right for You?

Difference	SOAP	REST
Style	Protocol	Architectural style
Function	Function-driven: transfer structured information	Data-driven: access a resoruce for data
Data format	Only uses XML	Permits many data formats, including plain text, HTML, XML, and JSON
Security	Supports WS-Security and SSL	Supports SSL and HTTPS
Bandwidth	Requires more resources and bandwidth	Requires fewer resources and is lightweight
Data cache	Can not be cached	Can be cached
Payload handling	Has a strict communication contract and needs knowledge of everything before any interaction	Needs no knowledge of the API
ACID compliance	Has built-in ACID compliance to reduce anomalies	Lacks ACID compliance

Visit upwork.com to learn more

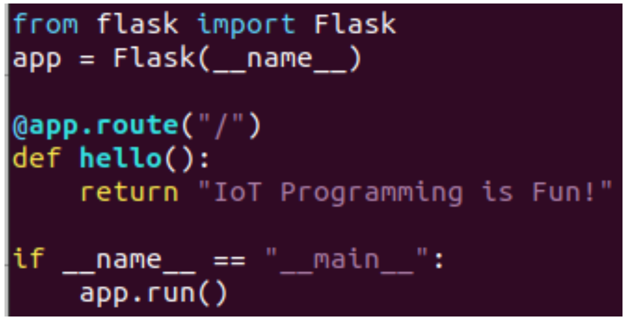
Coding a RESTfull API in Python

- You need the following packages:
  - Python: already installed in the Raspberry Pi
  - Flask: sudo apt-get install python-flask
  - Flask-Restful: sudo apt-get install python-flask-restful
  - Jsonify (optional – depending on how you code the response. More info: https://www.fullstackpython.com/flask-json-jsonify-examples.html)

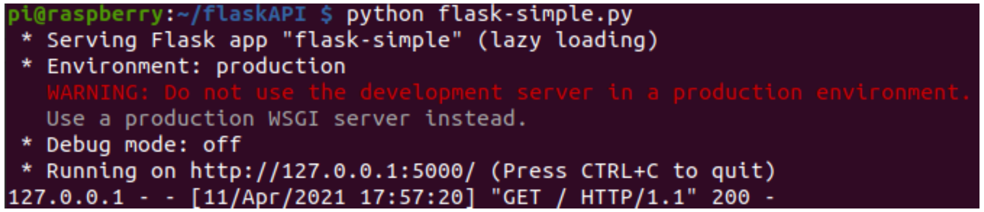
https://www.codementor.io/sagaragarwal94/building-a-basic-restful-api-in-python-58k02xsiq

Flask

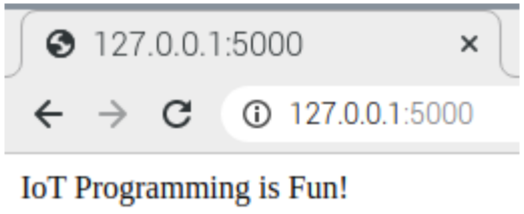
- Flask is a micro web framework written in Python
- It is classified as a micro framework because it does not require particular tools or libraries
- Applications that use the Flask framework include Pinterest, LinkedIn, and the community web page for Flask itself
- We used it in week 7 lab to control the edge device



```python
from flask import Flask
app = Flask(__name__)

@app.route("/")
def hello():
    return "IoT Programming is Fun!"

if __name__ == "__main__":
    app.run()
```



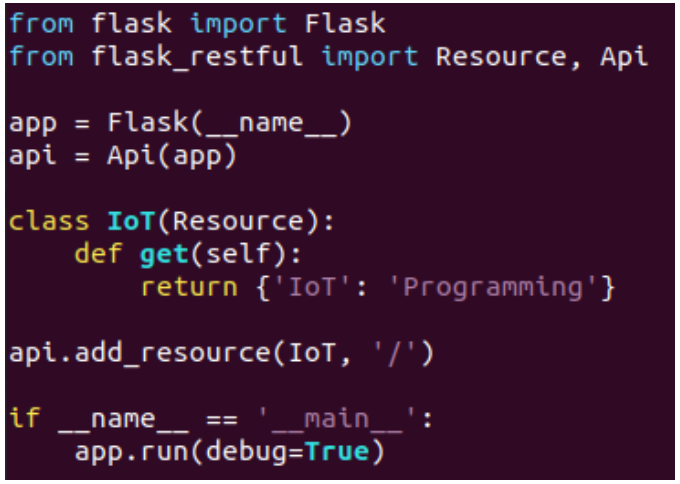
```
pi@raspberry:~/flaskAPI $ python flask-simple.py
 * Serving Flask app "flask-simple" (lazy loading)
 * Environment: production
   WARNING: Do not use the development server in a production environment.
   Use a production WSGI server instead.
 * Debug mode: off
 * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
127.0.0.1 - - [11/Apr/2021 17:57:20] "GET / HTTP/1.1" 200 -
```



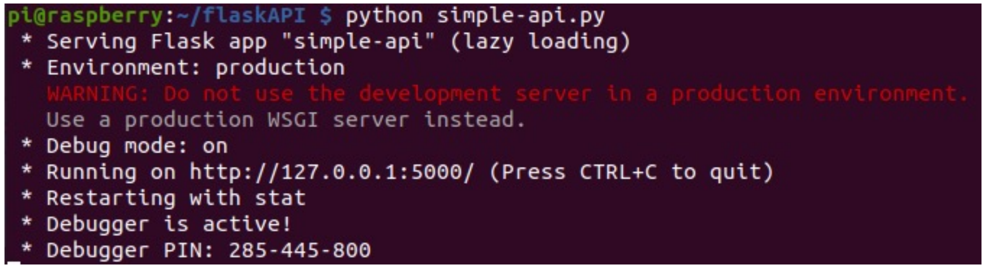
IoT Programming is Fun!

Flask-RESTful

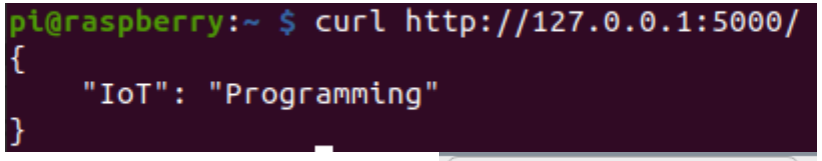
- Flask-RESTful is an extension for Flask that adds support for quickly building REST APIs
- It is a lightweight abstraction that works with your existing ORM/libraries.
- Flask-RESTful encourages best practices with minimal setup
- Task: change ‘IoT’ for ‘Potentiometer’ and ‘Programming’ for the value read via serial bus



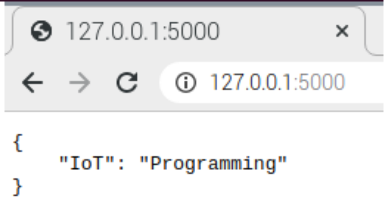
```python
from flask import Flask
from flask_restful import Resource, Api

app = Flask(__name__)
api = Api(app)

class IoT(Resource):
    def get(self):
        return {'IoT': 'Programming'}

api.add_resource(IoT, '/')

if __name__ == '__main__':
    app.run(debug=True)
```



```
pi@raspberry:~/flaskAPI $ python simple-api.py
 * Serving Flask app "simple-api" (lazy loading)
 * Environment: production
   WARNING: Do not use the development server in a production environment.
   Use a production WSGI server instead.
 * Debug mode: on
 * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 285-445-800
```



```
pi@raspberry:~ $ curl http://127.0.0.1:5000/
{
    "IoT": "Programming"
}
```



```
{
    "IoT": "Programming"
}
```

https://flask-restful.readthedocs.io/en/latest/quickstart.html#a-minimal-api

Connecting to APIs

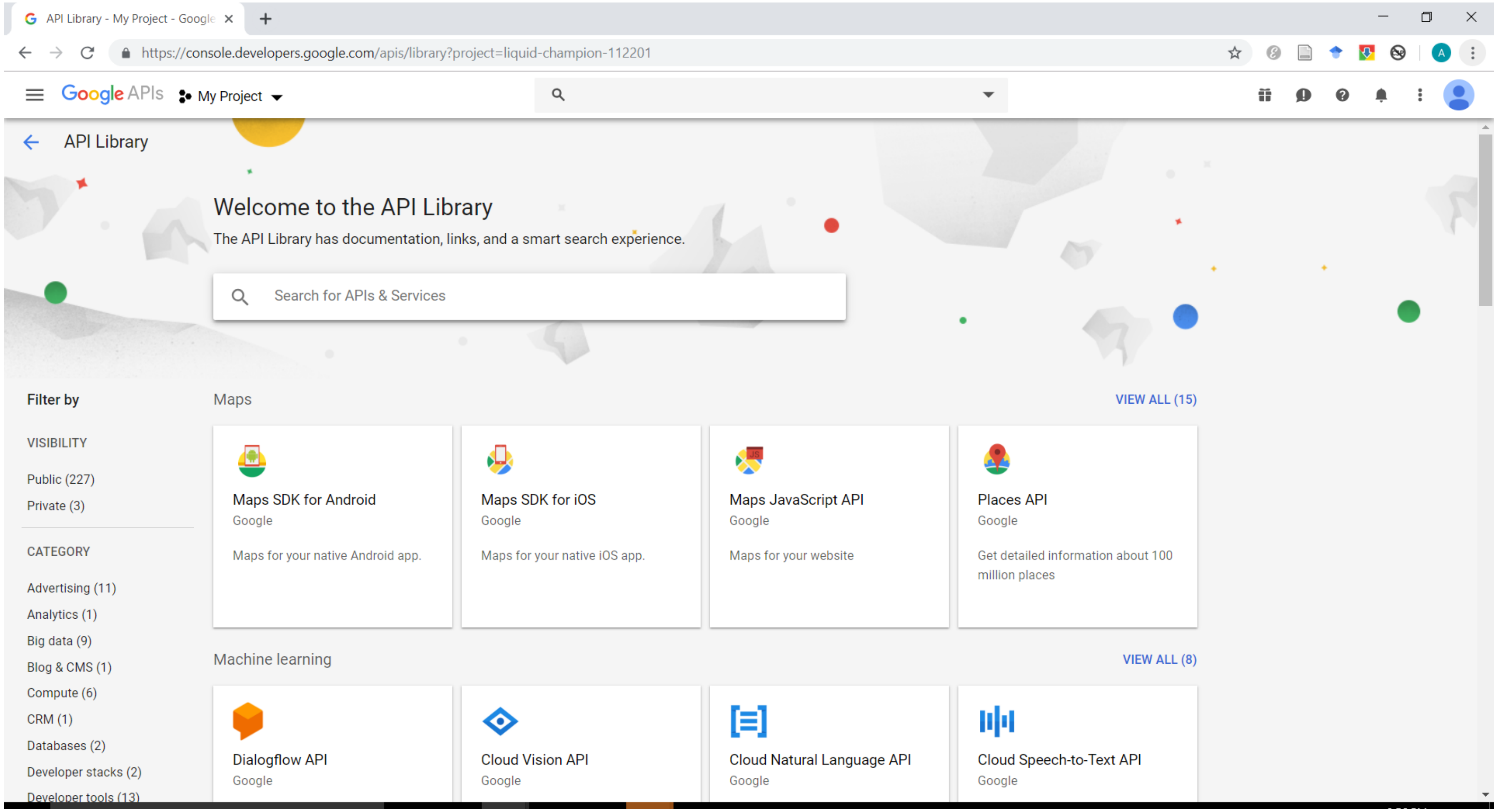
Connecting to Google Map API

- Google Map API Key
  - API keys are freely available with a Google Account at https://developers.google.com/console
  - The type of API key you need is a Server key

- $ pip install -U googlemaps

```
import googlemaps
gmaps = googlemaps.Client(key='Add Your Key here')
geocode_result = gmaps.geocode(John Street, Hawthorn, Melbourne')
```

Convert addresses into geographic coordinates


API Library - My Project - Google

https://console.developers.google.com/apis/library?project=liquid-champion-112201

API Library

Welcome to the API Library

The API Library has documentation, links, and a smart search experience.

Search for APIs & Services

Filter by

VISIBILITY

Public (227)
Private (3)

CATEGORY

Advertising (11)
Analytics (1)
Big data (9)
Blog & CMS (1)
Compute (6)
CRM (1)
Databases (2)
Developer stacks (2)
Developer tools (13)

Maps — VIEW ALL (15)

Maps SDK for Android
Google
Maps for your native Android app.

Maps SDK for iOS
Google
Maps for your native iOS app.

Maps JavaScript API
Google
Maps for your website

Places API
Google
Get detailed information about 100 million places

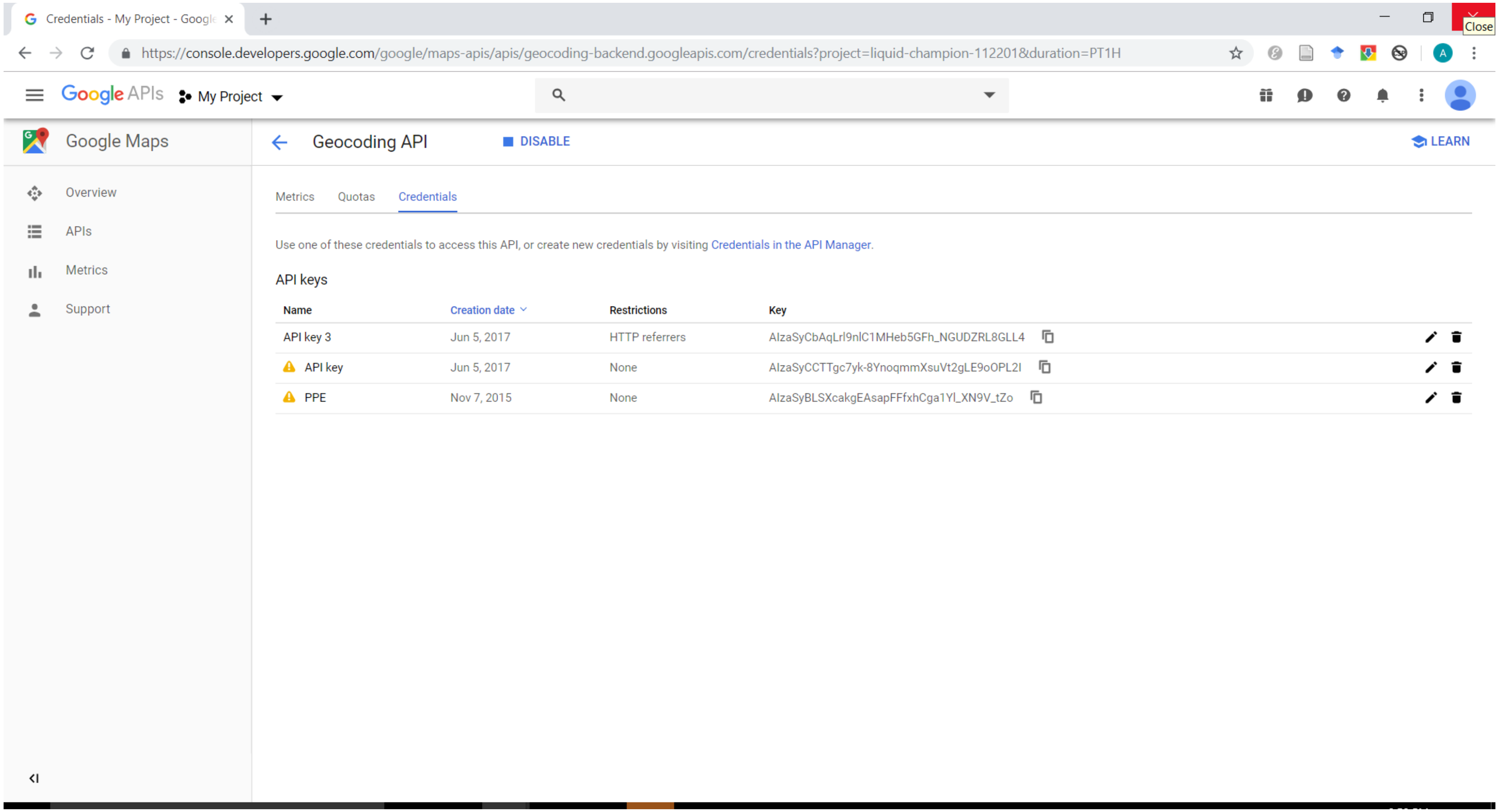
Machine learning — VIEW ALL (8)

Dialogflow API
Google

Cloud Vision API
Google

Cloud Natural Language API
Google

Cloud Speech-to-Text API
Google



Credentials - My Project - Google

https://console.developers.google.com/google/maps-apis/apis/geocoding-backend.googleapis.com/credentials?project=liquid-champion-112201&duration=PT1H

Geocoding API

Metrics | Quotas | Credentials

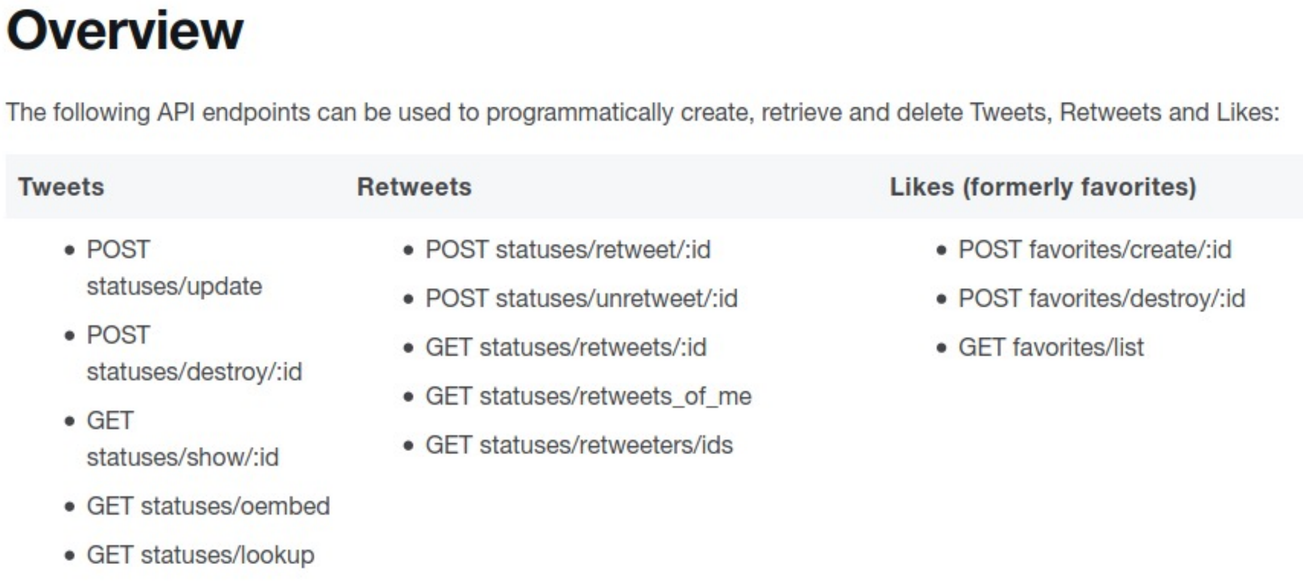
Use one of these credentials to access this API, or create new credentials by visiting Credentials in the API Manager.

API keys

Name	Creation date	Restrictions	Key
API key 3	Jun 5, 2017	HTTP referrers	AIzaSyCbAqLrl9nlC1MHeb5GFh_NGUDZRL8GLL4
⚠ API key	Jun 5, 2017	None	AIzaSyCCTTgc7yk-8YnoqmmXsuVt2gLE9oOPL2I
⚠ PPE	Nov 7, 2015	None	AIzaSyBLSXcakgEAsapFFfxhCga1Yl_XN9V_tZo

Twitter API

- https://developer.twitter.com/en/docs/twitter-api
  - Analyse conversations
  - Explore user’s tweets
  - Post, retrieve, and engage with Tweets



Overview

The following API endpoints can be used to programmatically create, retrieve and delete Tweets, Retweets and Likes:

Tweets	Retweets	Likes (formerly favorites)
• POST statuses/update	• POST statuses/retweet/:id	• POST favorites/create/:id
• POST statuses/destroy/:id	• POST statuses/unretweet/:id	• POST favorites/destroy/:id
• GET statuses/show/:id	• GET statuses/retweets/:id	• GET favorites/list
• GET statuses/oembed	• GET statuses/retweets_of_me	
• GET statuses/lookup	• GET statuses/retweeters/ids	

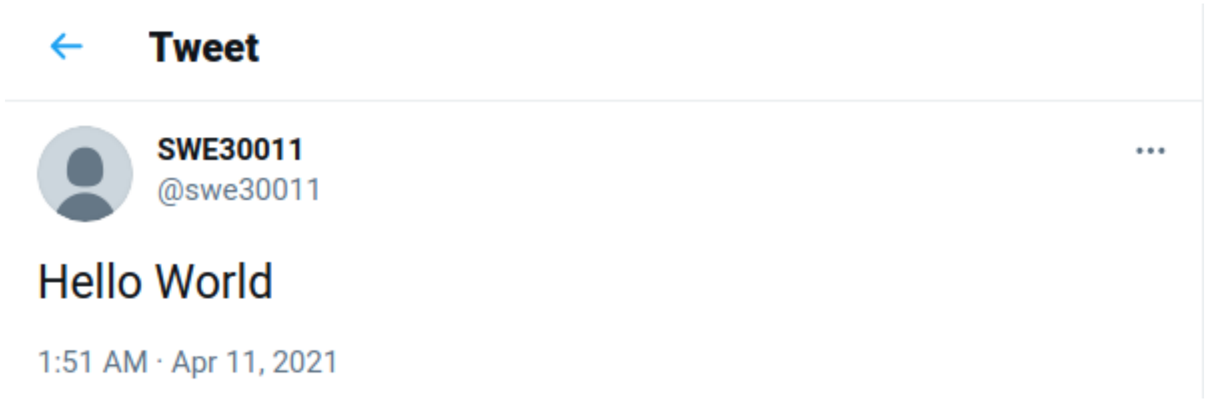
https://developer.twitter.com/en/docs/twitter-api/v1/tweets/post-and-engage/overview

Twitter API

- If you don't want to deal directly with the API, you can find third party software interface
- Twitter CLI
  - https://github.com/sferik/t
- Twitter Python
  - https://python-twitter.readthedocs.io/en/latest/
  - https://www.tweepy.org/
  - Etc.

Twitter API: Connecting to the edge

- https://github.com/sferik/t
- sudo apt-get install ruby-dev
- sudo gem install t
- t authorize
- t update "Hello World"

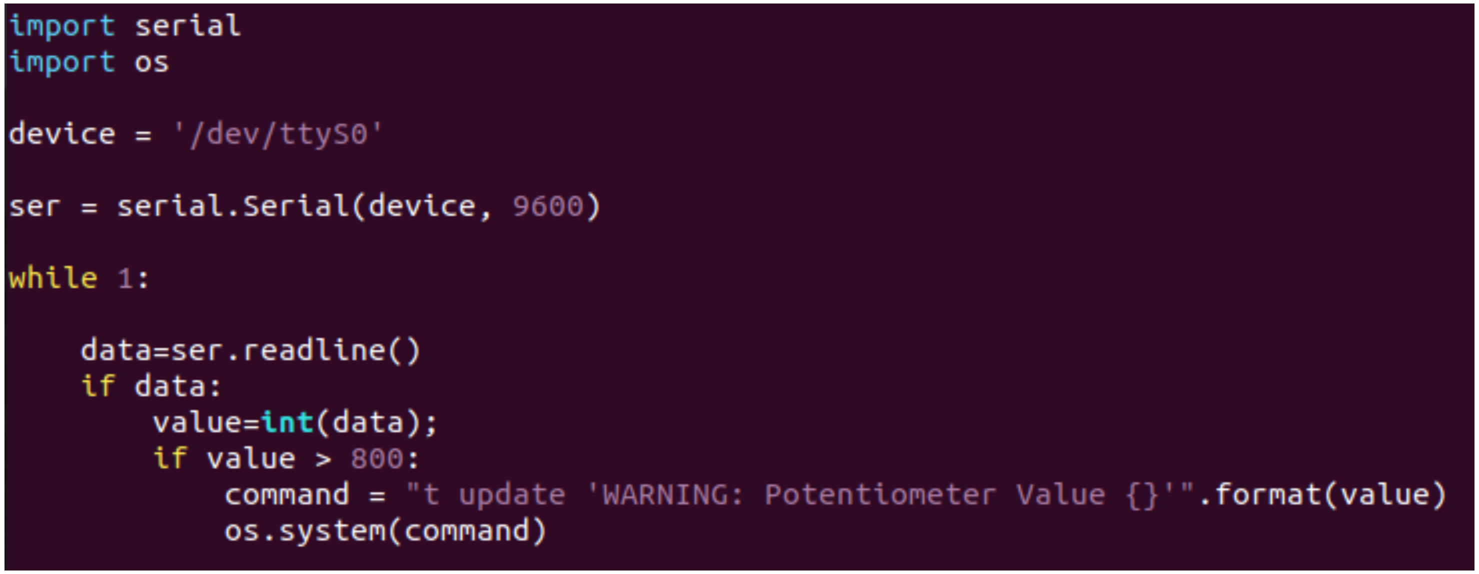


SWE30011
@swe30011

Hello World

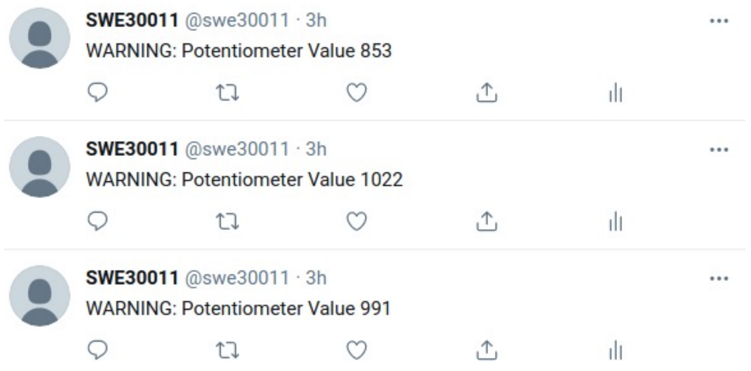
1:51 AM · Apr 11, 2021

Twitter API: Connecting to the edge



```python
import serial
import os

device = '/dev/ttyS0'

ser = serial.Serial(device, 9600)

while 1:

    data=ser.readline()
    if data:
        value=int(data);
        if value > 800:
            command = "t update 'WARNING: Potentiometer Value {}'".format(value)
            os.system(command)
```

https://twitter.com/swe30011/



SWE30011 @swe30011 · 3h
WARNING: Potentiometer Value 853

SWE30011 @swe30011 · 3h
WARNING: Potentiometer Value 1022

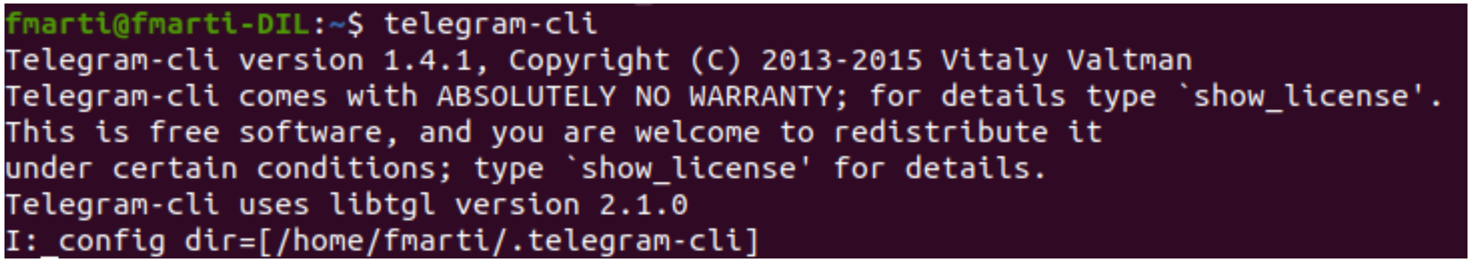
SWE30011 @swe30011 · 3h
WARNING: Potentiometer Value 991

Telegram API

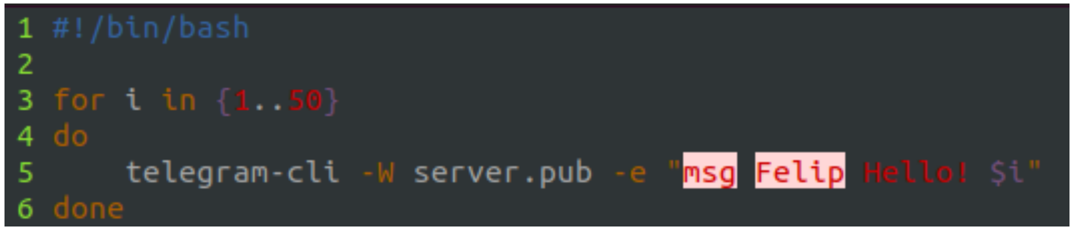
- https://core.telegram.org/
- Two types of API:
  - The Telegram API and TDLib allow you to build your own customised Telegram clients. E.g., telegram-cli
  - The Bot API allows you to easily create programs that use Telegram messages for an interface
    - Heaps of bots sample code in Python, Ruby, C++, PHP, etc.
    - https://core.telegram.org/bots/samples

Telegram API: telegram-cli

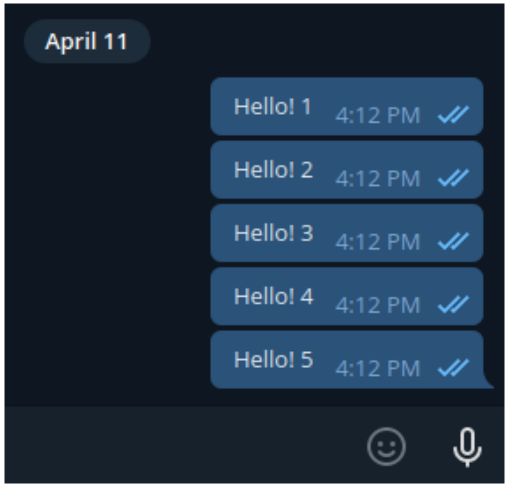
- https://github.com/vysheng/tg
- Uses the telegram API to send telegram messages via command line interface.



```
fmarti@fmarti-DIL:~$ telegram-cli
Telegram-cli version 1.4.1, Copyright (C) 2013-2015 Vitaly Valtman
Telegram-cli comes with ABSOLUTELY NO WARRANTY; for details type `show_license'.
This is free software, and you are welcome to redistribute it
under certain conditions; type `show_license' for details.
Telegram-cli uses libtgl version 2.1.0
I: config dir=[/home/fmarti/.telegram-cli]
```



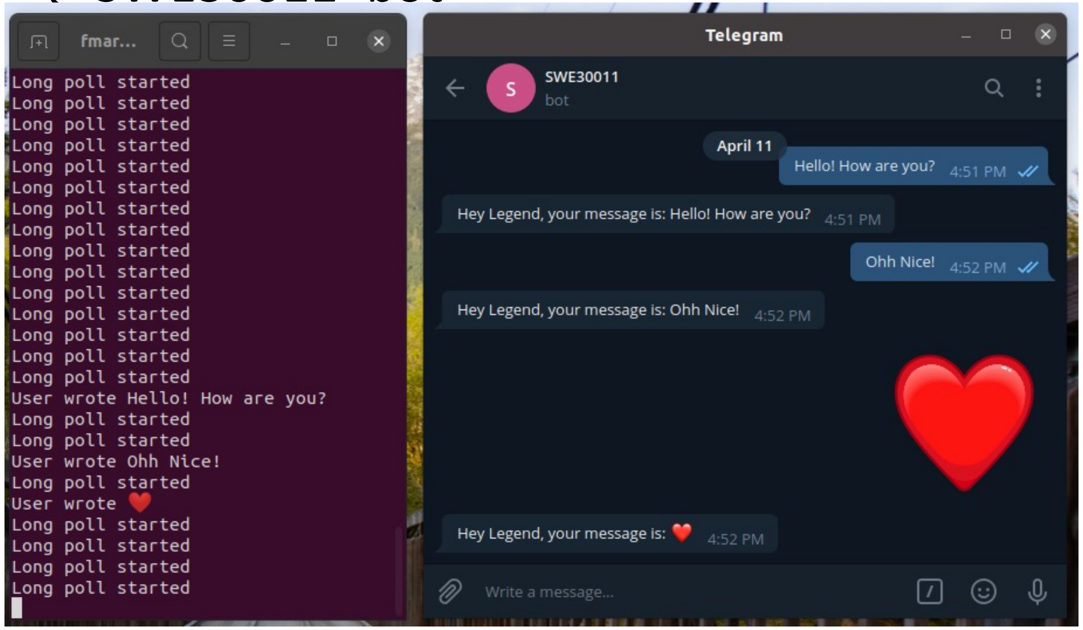
```
#!/bin/bash

for i in {1..50}
do
    telegram-cli -W server.pub -e "msg Felip Hello! $i"
done
```



```
April 11
Hello! 1   4:12 PM
Hello! 2   4:12 PM
Hello! 3   4:12 PM
Hello! 4   4:12 PM
Hello! 5   4:12 PM
```

Telegram API: bots

- You don't need to put your personal credentials
- It works with access tokens
- Echo bot in C++: https://github.com/desmoteo/telegram-bot-api
- @SWE30011_bot



```
Long poll started
Long poll started
Long poll started
Long poll started
Long poll started
Long poll started
Long poll started
Long poll started
Long poll started
Long poll started
Long poll started
Long poll started
Long poll started
Long poll started
Long poll started
User wrote Hello! How are you?
Long poll started
Long poll started
User wrote Ohh Nice!
Long poll started
User wrote ❤️
Long poll started
Long poll started
Long poll started
Long poll started
```

SWE30011
bot

April 11

Hello! How are you? 4:51 PM

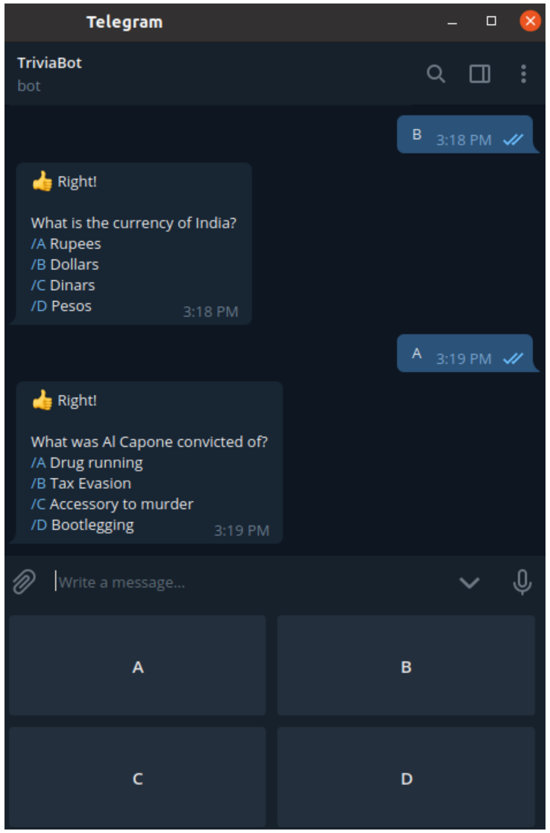
Hey Legend, your message is: Hello! How are you? 4:51 PM

Ohh Nice! 4:52 PM

Hey Legend, your message is: Ohh Nice! 4:52 PM

❤️

Hey Legend, your message is: ❤️ 4:52 PM



TriviaBot
bot

B 3:18 PM

👍 Right!

What is the currency of India?
/A Rupees
/B Dollars
/C Dinars
/D Pesos 3:18 PM

A 3:19 PM

👍 Right!

What was Al Capone convicted of?
/A Drug running
/B Tax Evasion
/C Accessory to murder
/D Bootlegging 3:19 PM

Telegram trivia bot with fast response (commands) buttons

Other APIs

- Discord: https://discord.com/developers/docs/intro
- Instagram: https://www.instagram.com/developer/
- Australian Weather Data (using bom.gov.au):
  - http://www.bom.gov.au/catalogue/data-feeds.shtml
  - Data in ftp server: ftp://ftp.bom.gov.au/anon/gen/
  - https://pypi.org/project/weather-au/
- The RESTful Pokémon API: https://pokeapi.co/
- Spotify:
- https://developer.spotify.com/documentation/web-api/
- Etc.

Questions?