

CISCO *Live!*

GO BEYOND

#CiscoLiveAPJC



Begin Your Journey to Webhook Apps Using Meraki API and Docker

A story of how I started my journey on coding and automation

Fred Chua, Solutions Engineer

DEVNET-1142

Cisco Webex App

Questions?

Use Cisco Webex App to chat with the speaker after the session

How

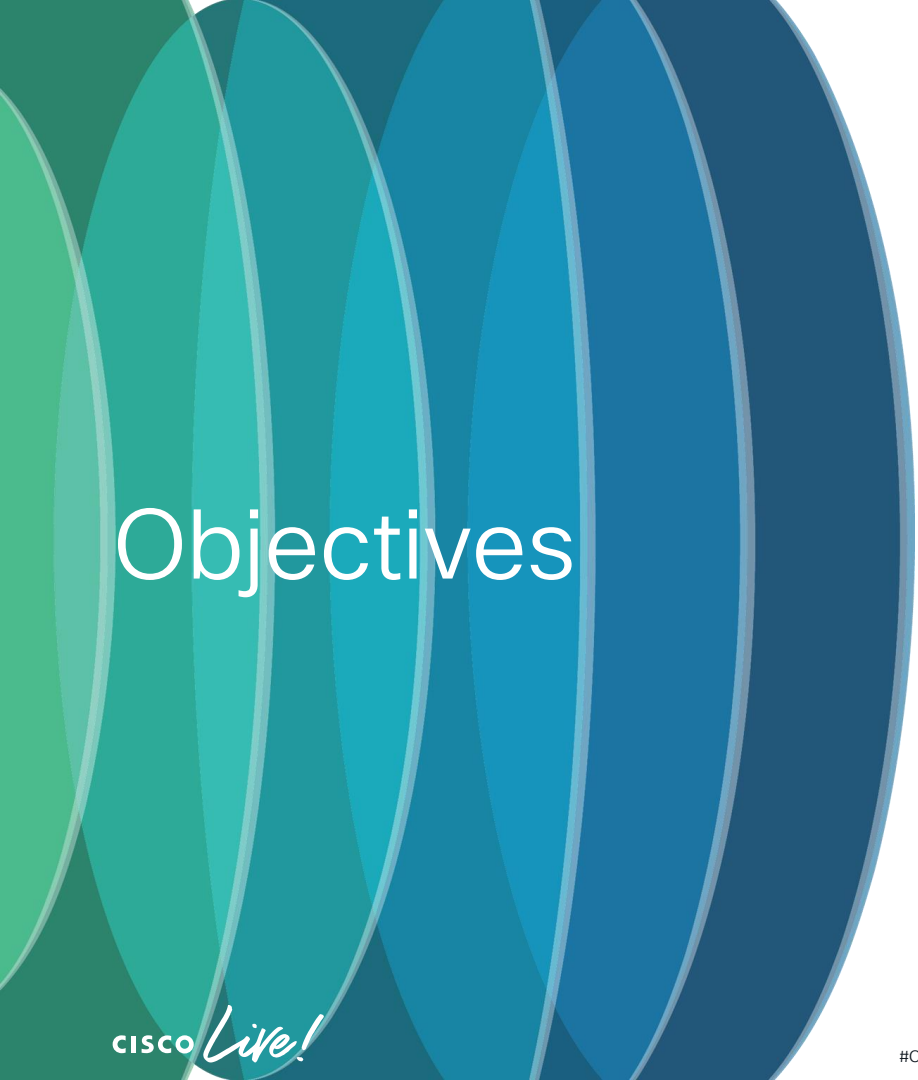
- 1 Find this session in the Cisco Live Mobile App
- 2 Click “Join the Discussion”
- 3 Install the Webex App or go directly to the Webex space
- 4 Enter messages/questions in the Webex space

Webex spaces will be moderated by the speaker until November 15, 2024.



<https://ciscolive.ciscoevents.com/ciscolivebot/#DEVNET-1142>





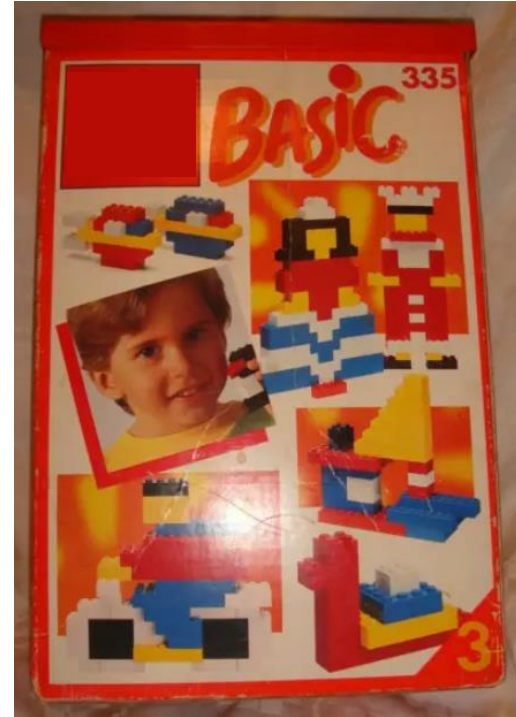
Objectives

- About Me
- Starting Line: The Challenge
 - Why I started my coding project
- Understanding What is Possible
 - What are APIs, Webhooks and Containers?
- Putting Everything Together
- Quick Demo

\$ whoami

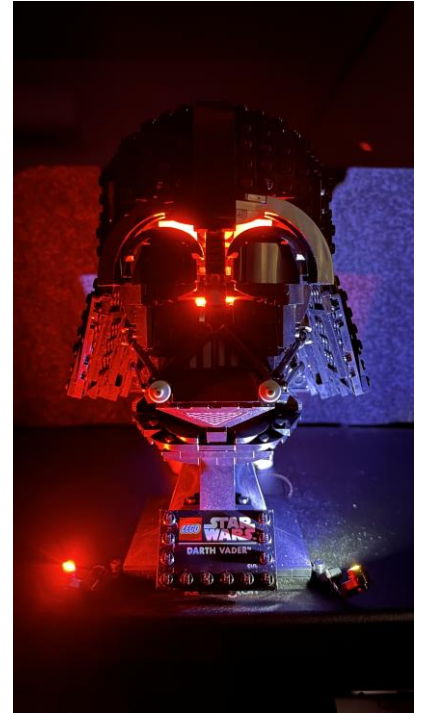
The embodiment of a nerd's mind

How it started...



The embodiment of a nerd's mind

How it's going...



Fun fact: I was NOT good at programming

I'm not kidding...

My grades sucked BIG time



2003 - 2004	
COMPUTER PROGRAMMING 1 (LAB)	2.50
COMPUTER PROGRAMMING 1 (LEC)	2.50

This grade is just above passing ^^

Starting Line: Solving a Challenge

Begin with the end in mind



- Why? What? How?
- Can I automate it?
- Starting with small ideas are totally okay

Why: My reason for starting the project

I wanted focused notifications:

- MV Motion Alert
- Sensor Readings
- Device Outages



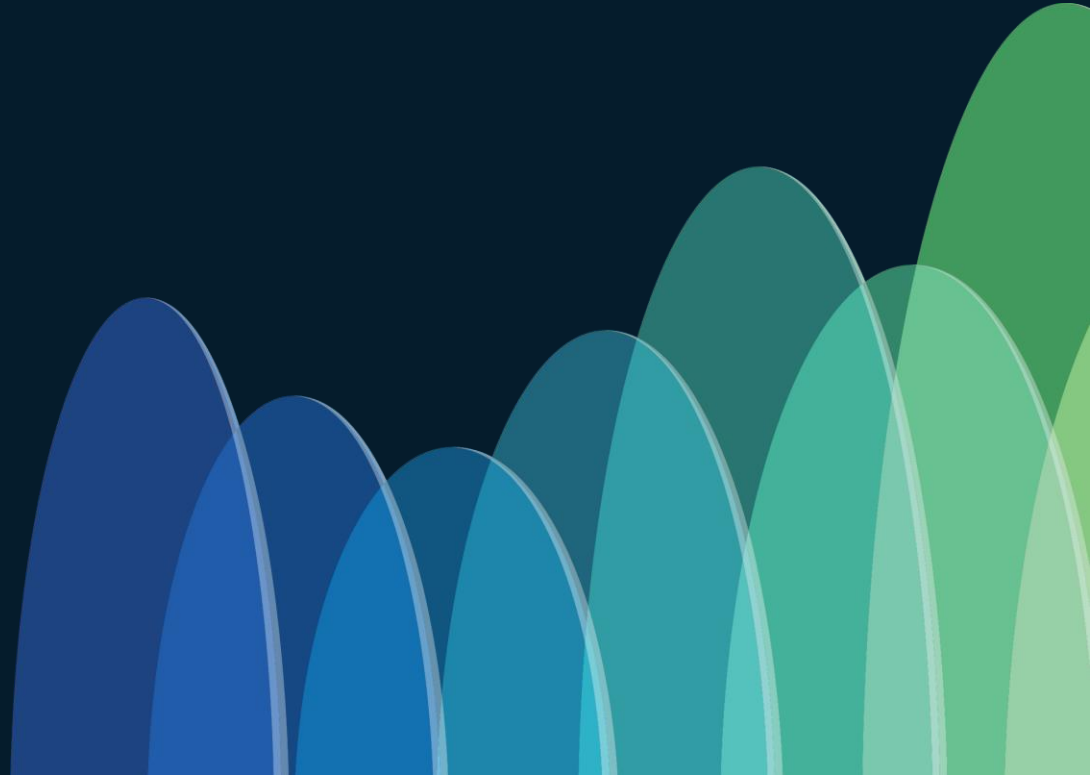
The starting line: my problem statement

“I want to receive real-time event alerts from my Meraki dashboard.”

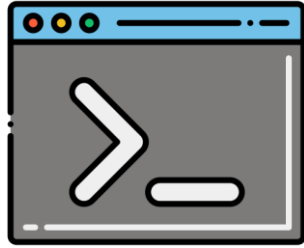
...followed by:

“Email/SMS alerts are not good enough. How can I achieve this outcome using **Webex** or other channels?”

Understanding What is Possible



Review: computer science 101

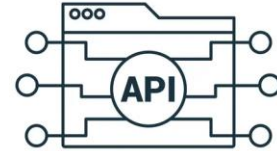


Command
Line
Interface



GUI

Graphical
User
Interface



API

Application
Programming
Interface

So...what is an API?

The term “API” has been generically used to describe **connectivity interfaces** to an application.

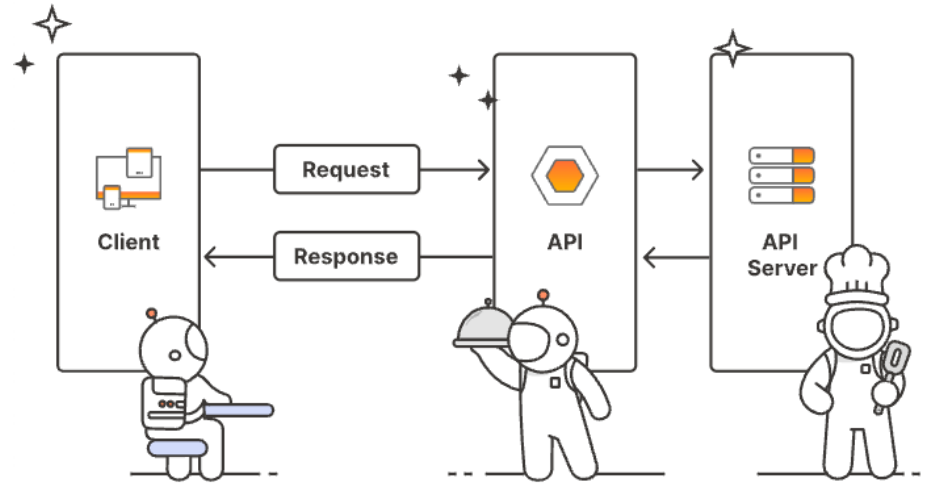
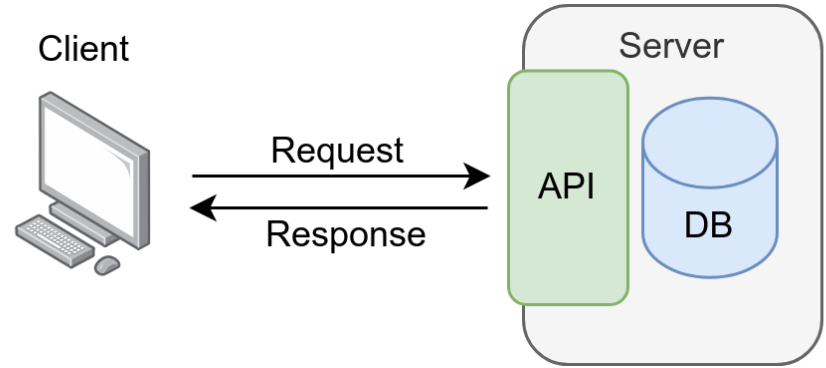


Image credits: [Postman](#)

Again...what is an API?

Application refers to any **software** with a **distinct function**. (i.e. a computer program)

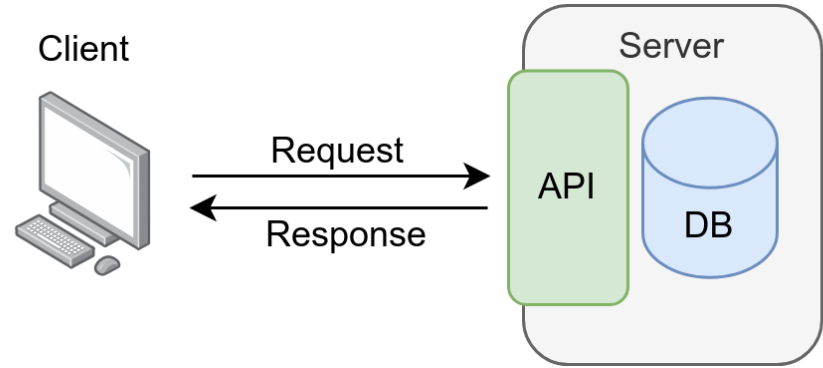


Interface can be thought of as a contract of service between two applications. This contract defines how the two communicate with each other using requests and responses.

Source: <https://aws.amazon.com/what-is/api/>

Again...what is an API?

Application refers to any **software** with a **distinct function**. (i.e. a computer program)

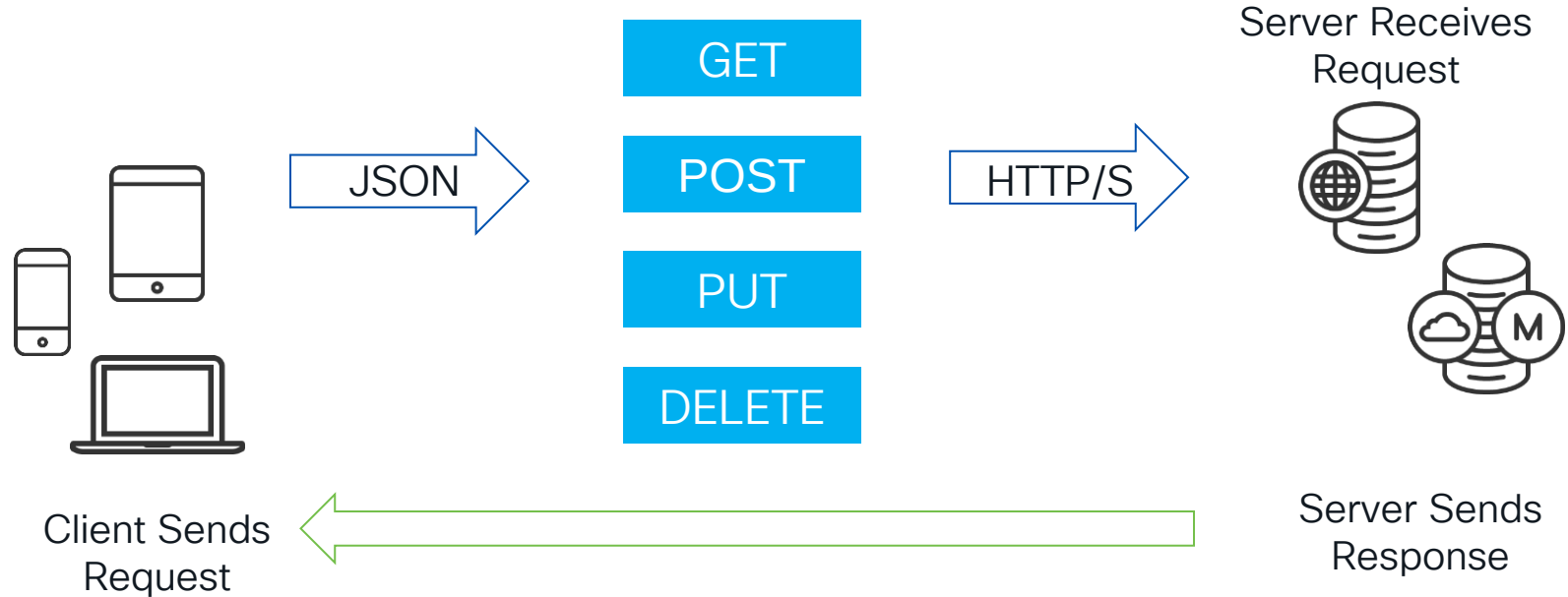


Interface can be thought of as a **contract of service** between two applications. This contract defines **how the two communicate** with each other **using requests and responses**.

Source: <https://aws.amazon.com/what-is/api/>

Examples of APIs:
SOAP, RPC, REST

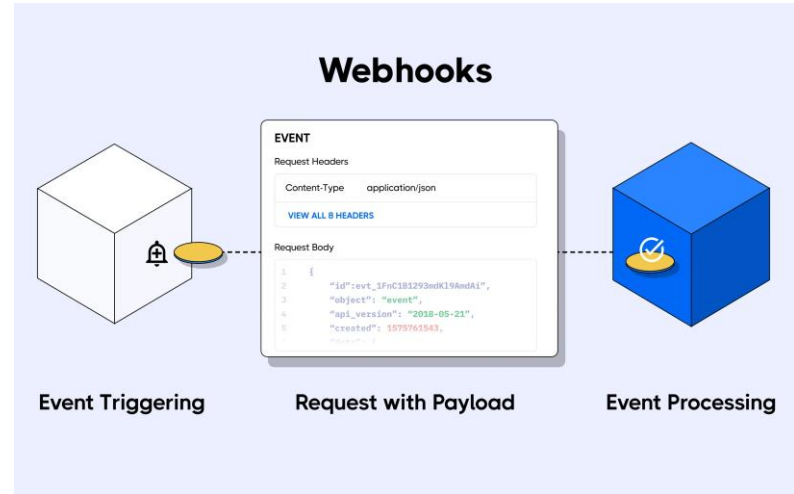
REST APIs: What we are here to discuss



What is a Webhook?

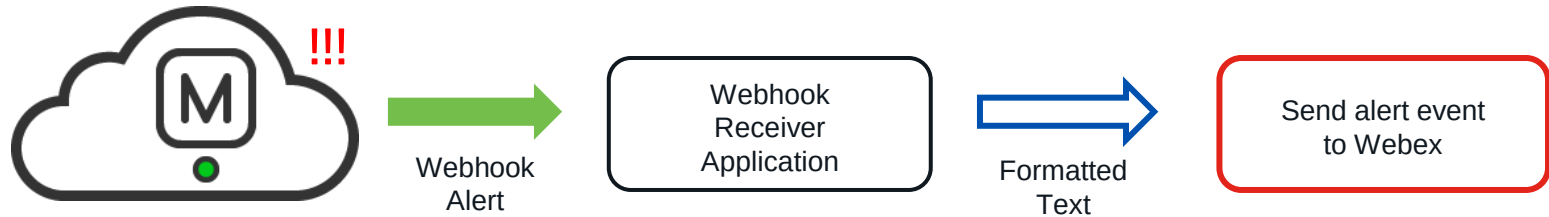
A webhook is an HTTP request, **triggered by an event** in a source system and **sent to a destination system**, often with a payload of data.

Webhooks are automated, i.e. they are automatically sent out when their event is fired in the source system.



Source: <https://hookdeck.com/webhooks/guides/what-are-webhooks-how-they-work>

My webhook receiver app: simplified high-level flow



Where to begin?



It all begins with a computer program (or code)

What is a Program?

Program is a **set of instructions** that a machine follows.

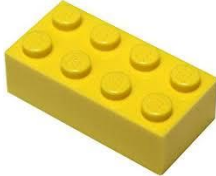
What is Programming?

Programming is to create **instruction sets** that a machine follows **to execute a task.**



An oversimplified representation of computer code

Data In →  → Output

Data In →  → Action

Data In →  → Other

Code in Blocks

An oversimplified representation of computer code coming together

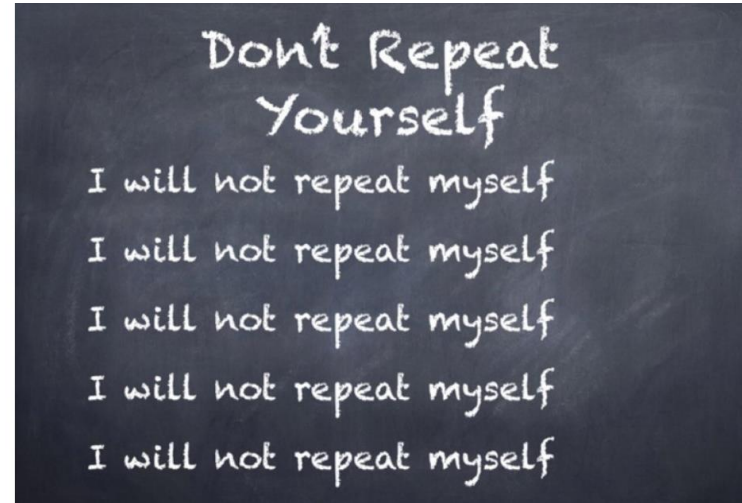


💡 Tip: Remember to stay D-R-Y

Don't Repeat Yourself

Repetitive **blocks of code** which serve a single purpose can be written as **functions**, then called and reused as often as necessary.

No need for repetition.



Code Snippet: Send Message to Webex App

Builds message payload
from received formatted
text

Sends constructed
message to a Webex
space using POST API
call

Error handling and
logging

```
def outbox_str_only(md_body):
    tx_runtime = RuntimeLoader()

    mp_payload: MultipartEncoder = MultipartEncoder(
        {
            "roomId": tx_runtime.WX_ROOM_ID,
            "markdown": md_body
        })

    headers: dict = ({
        'Content-Type': mp_payload.content_type,
        'Authorization': f'Bearer {tx_runtime.WX_TOKEN}'
    })

    try:
        response = requests.post(tx_runtime.WX_API_URL,
                                headers=headers, data=mp_payload)
        # Feedback: print response body
        if response and response.status_code == 200:
            response_dict: dict = response.json()
            created_at: str = utc_iso_to_tz_offset(utc=(response_dict.get('created_at', '')))
            logger.info(f"Webex Sender: Message sent {created_at}")

            return response_dict

        logger.error(f'POST Error: {tx_runtime.WX_API_URL}')
        raise HTTPRequestExceptionError(f'POST Error: {tx_runtime.WX_API_URL}')

    except HTTPRequestExceptionError:
        logger.error(f'HTTP Exception: {response.status_code} - {response.get('status_code', response.status_code)}')
        raise HTTPRequestExceptionError(status_code=response.status_code, message=f'HTTP Exception: {response.status_code} - {response.get('status_code', response.status_code)}')
```

Oversimplified representation of a typical API app

Input Data
(usually in JSON)



API Application

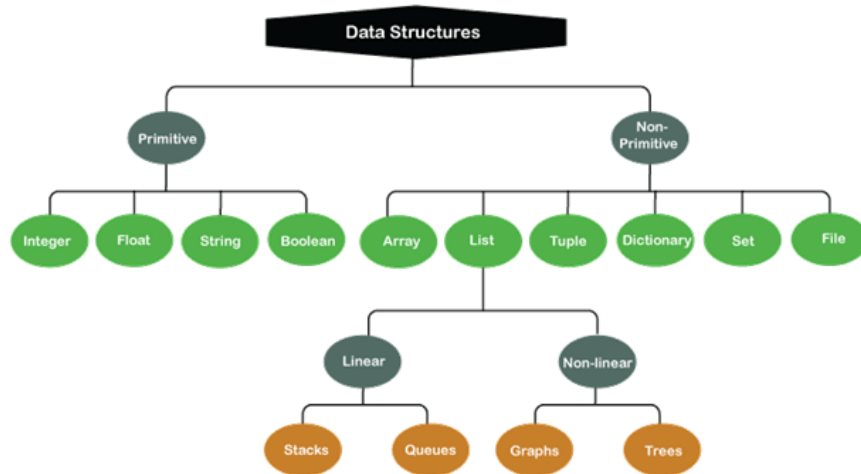


Output
and/or
Action

Understand data types and formats

Data Type

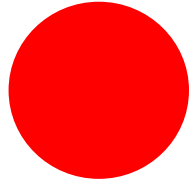
It tells about the type of data a variable can accept. i.e. **string**, **integer**, **float**, etc.



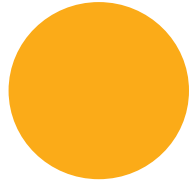
Data Formats

It is a representation of structured data.

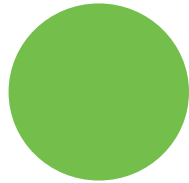
Data formats: Using a simple analogy



STOP

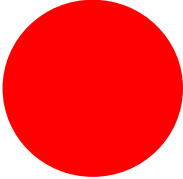
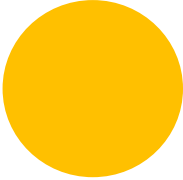
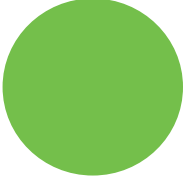


CAUTION or
SLOW DOWN



GO

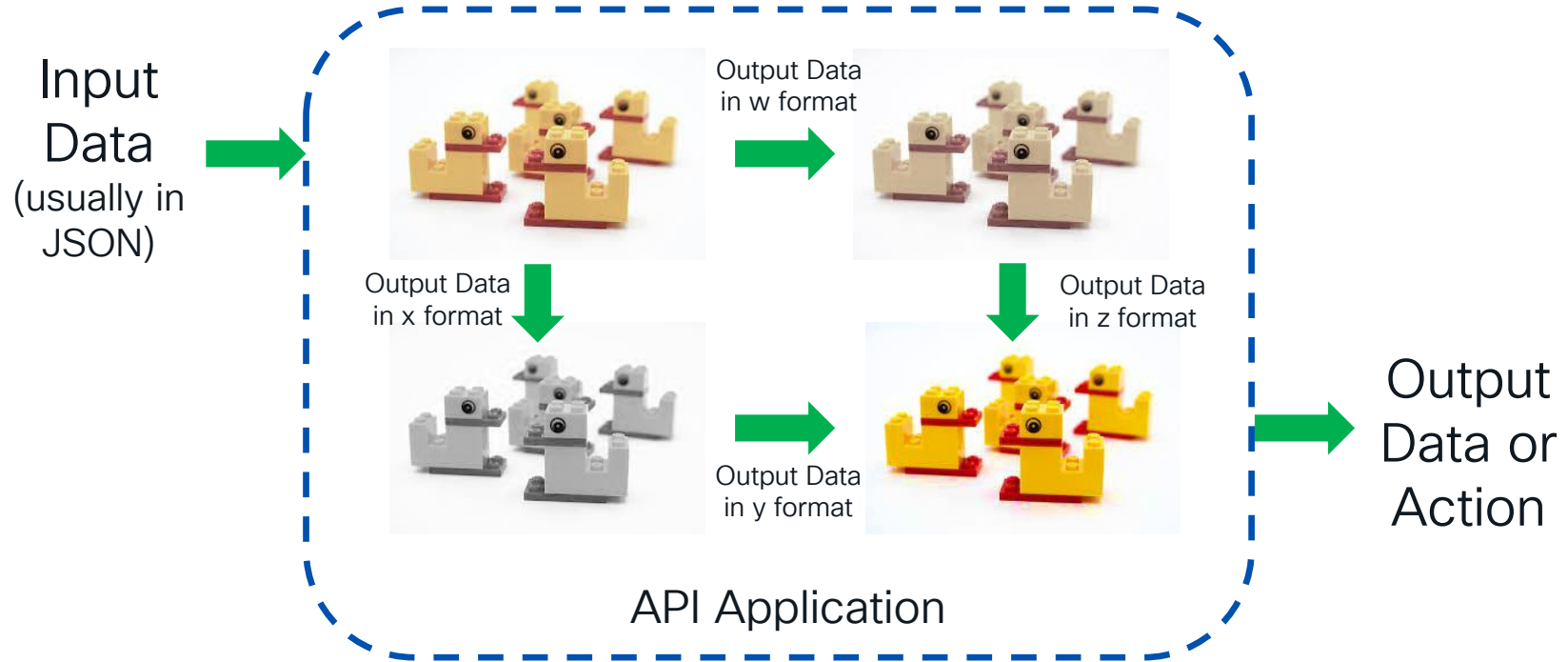
How I visualise a dictionary object in JSON format

```
{  "   "     "SLOW DOWN"]  ,  
  " 
```

Code snippet: Meraki webhook alert sample

```
{
  "version": "0.1",
  "sharedSecret": "secret",
  "sentAt": "2021-10-01T09:27:29.615483Z",
  "organizationId": "2930418",
  "organizationName": "My organization",
  "organizationUrl": "https://dashboard.meraki.com/o/VjjsAd/manage/organization/overview",
  "networkId": "N_24329156",
  "networkName": "Main Office",
  "networkUrl": "https://n1.meraki.com/n//manage/nodes/list",
  "networkTags": [],
  "deviceSerial": "Q234-ABCD-5678",
  "deviceMac": "00:11:22:33:44:55",
  "deviceName": "My camera",
  "deviceUrl": "https://n1.meraki.com/n//manage/nodes/new_list/000000000000",
  "deviceTags": [ "tag1", "tag2" ],
  "deviceModel": "MV",
  "alertId": "0000000000000000",
  "alertType": "Motion detected",
  "alertTypeId": "motion_alert",
  "alertLevel": "informational",
  "occurredAt": "2018-02-11T00:00:00.123450Z",
  "alertData": {
    "imageUrl": "https://cataas.com/cat",
    "timestamp": 1563499479.547,
    "motionRecapEnabled": true,
    "motionDetectorAllowsRecap": true,
    "isSpyglass": false,
    "imageEnabled": true
  }
}
```

Oversimplified representation of an API app data flow



Next Up: Containers



The starting line: my problem statement

...followed by:

“I want to receive real-time event alerts from my Meraki dashboard.”

“Email/SMS alerts are not good enough. How can I achieve this outcome using **Webex** or other channels?”

Finally:

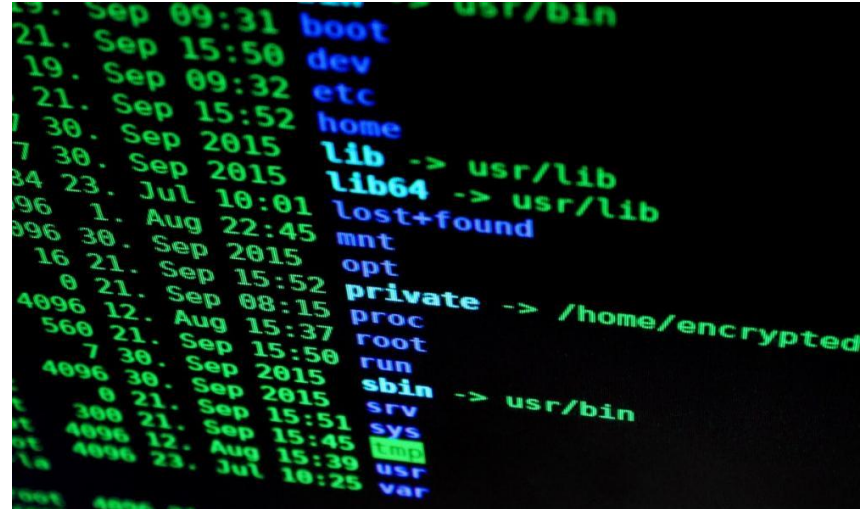
“I want my application running all the time”



New roadblocks ahead!



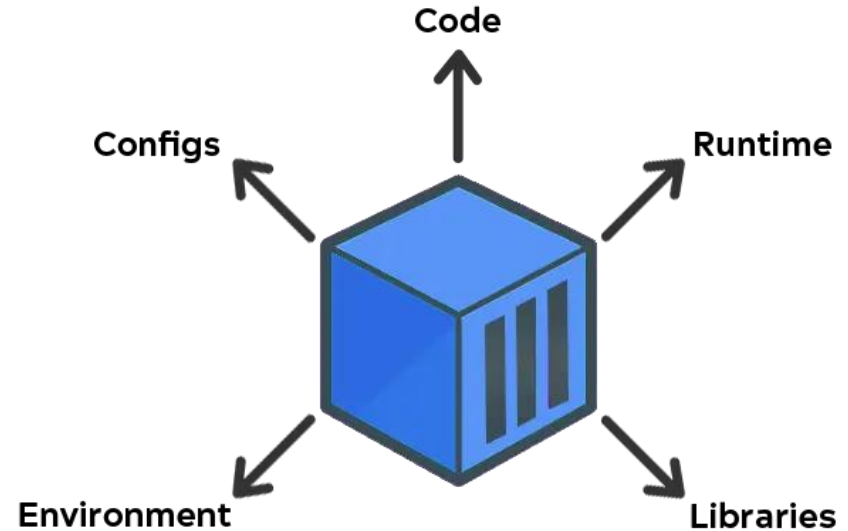
- What I Want: Have my script running 24/7
- What runtime OS do I use? Windows? Mac? Linux?
- In a terminal environment, the Python script terminates once the console window shuts down.



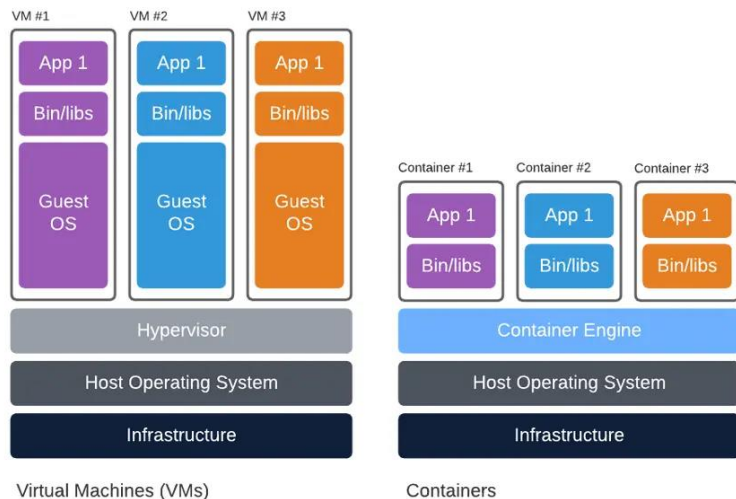
What is a container?

- Small, **standalone** executable **package** -- contains everything needed to run your application (libraries, dependencies, etc.)
- Eliminates the “**it runs on my machine**” problem
- Building blocks for **microservices** architecture

What's included in a Container?



Virtualisation: VM vs Container



I asked GPT to explain the difference:

- Imagine you have a big toy box, and you want to play with your friends
- A **VM** is like a separate toy box for each friend, with its own space and resources.
- A **container** is like a special bag inside the same big toy box, sharing space and resources but keeping things organized.

💡 Tip: Make AI your best friend

Getting code ideas from ChatGPT or Github CoPilot is NOT cheating. We're not taking an exam.

“Use ChatGPT like how you use Google.”

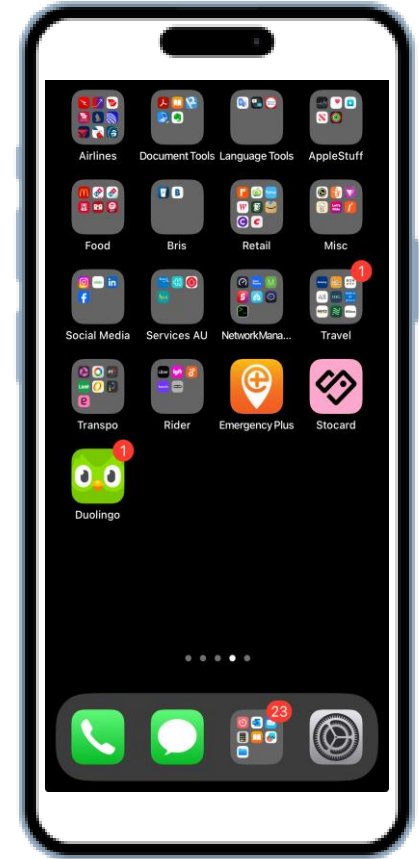
– Y. Terada

A friendly reminder from:



Next up: packaging your application

- Think **IOS/Android** apps: Containers are similar standalone runtimes
- Build your application's image via a Dockerfile (think: **installer script**)
- Publish your image via Docker's public repository (think: **App Store / Play Store**)



Code snippet: my webhook receiver's Dockerfile

```
# syntax=docker/dockerfile:1

# Use latest official Python 3.11 image as the base image
FROM python:3.11-slim-bookworm

# Set the working directory inside the container
WORKDIR /Receiver

# Copy the entire current directory into the container
COPY . .

# Install the Python dependencies
RUN pip3 install --no-cache-dir -r requirements.txt

# Set the command to run your Python application
CMD ["python3", "app.py"]
```

Finally: my app container is ready to deploy

```
$ docker run --env-file [path/to/env/file] \  
  --network [network_name] --volume [volume_name] \  
  -p [host_port:container_port] -d [image name]
```



This command runs ONE container

A simpler way with Docker Compose

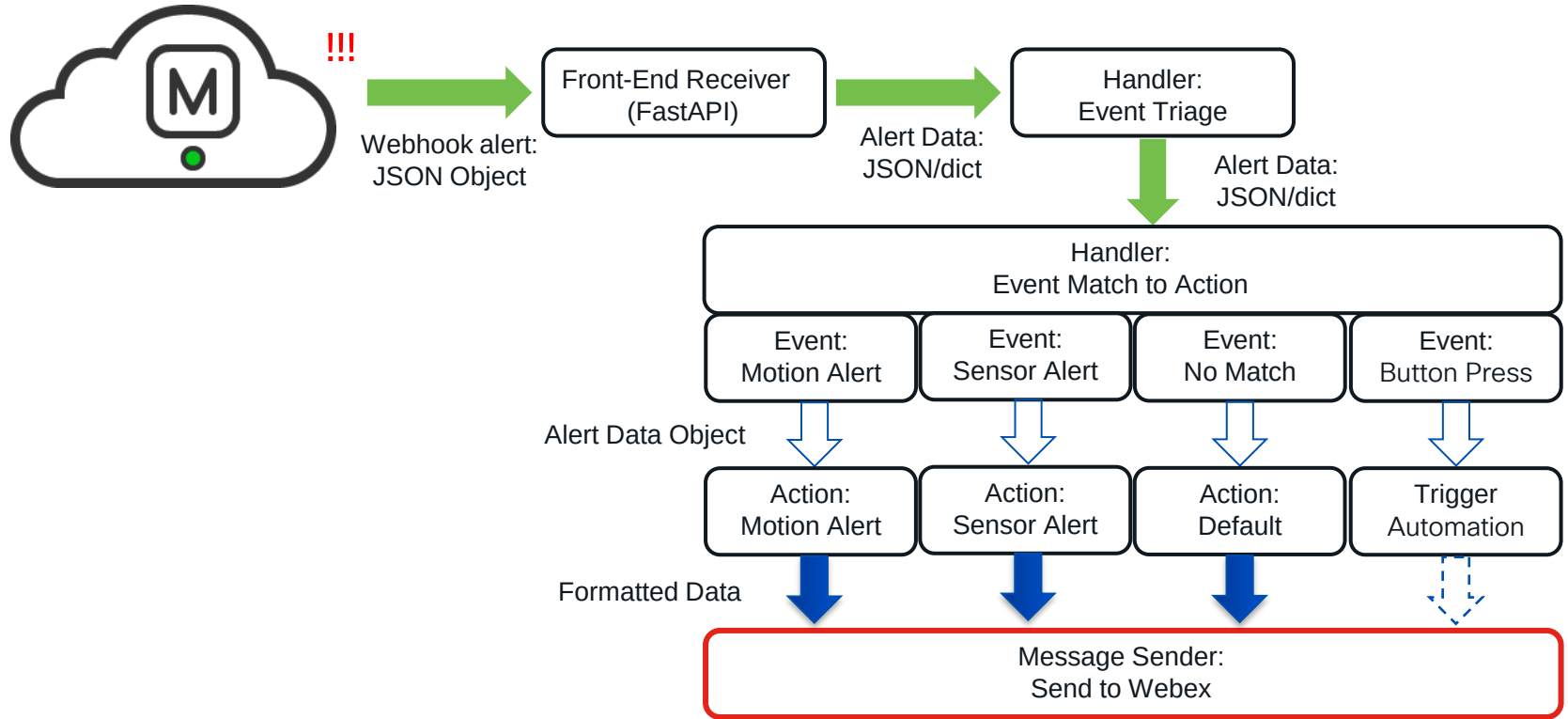
- Docker Compose is a tool for **defining** and **running** multi-container applications.
- With a single command, you create and start all the services from your configuration file.

\$ docker compose up

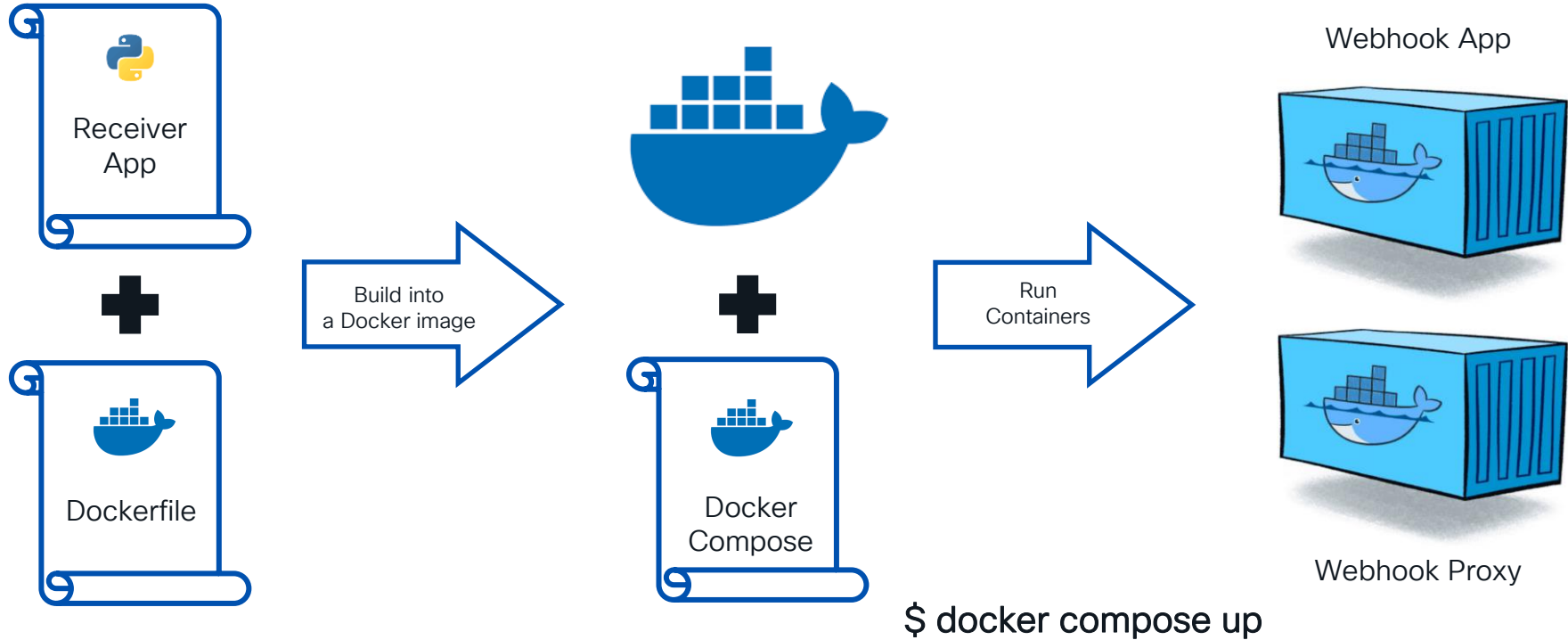
```
1  services:
2      whfast:
3          image: "*****/whfast:v0.8"
4          container_name: whfast
5          env_file:
6              - .env.prod
7          ports:
8              - "${HOST_PORT}:${APP_PORT}"
9          volumes:
10             - snaps-data:/Receiver/snaps
11          networks:
12             - network1
13
14  networks:
15      network1:
16          name: webhook
17
18  volumes:
19      snaps-data:
20
```

Putting It All Together

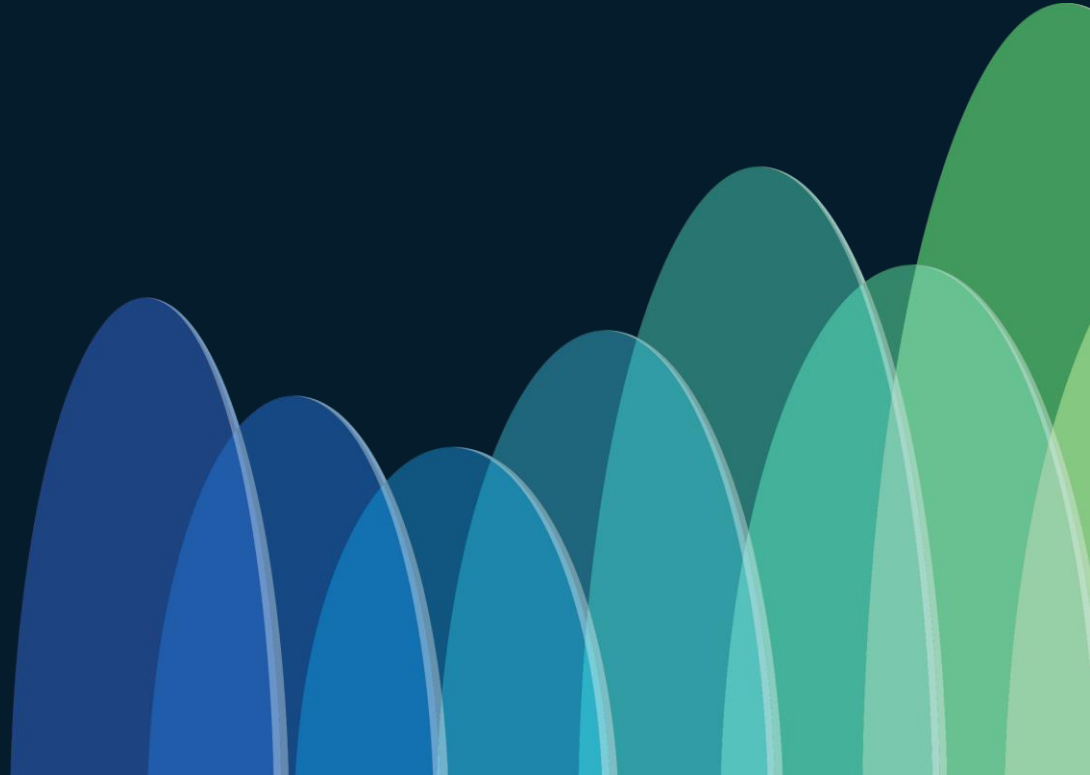
Webhook receiver app workflow



Webhook Receiver App: Putting it together

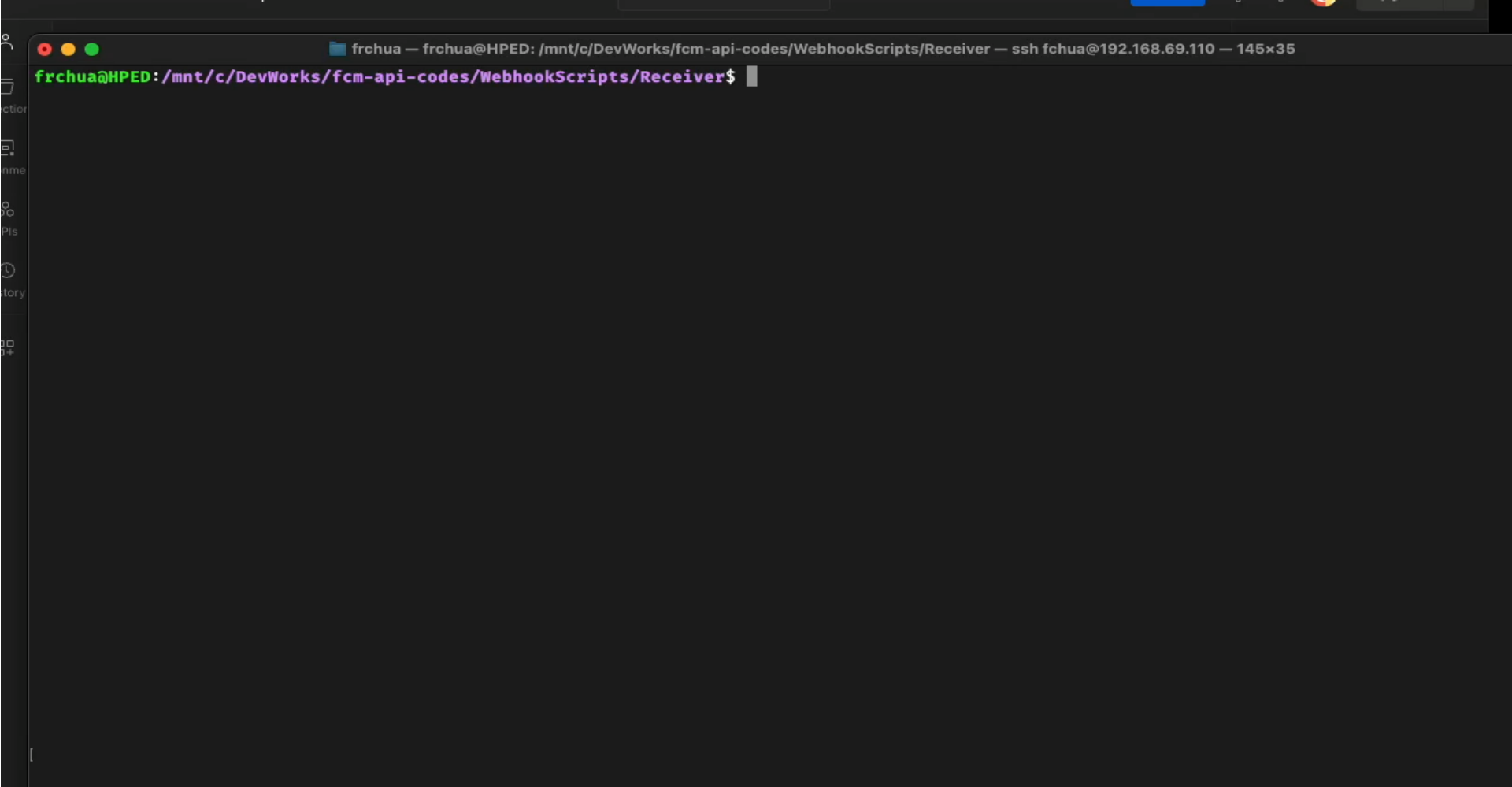


Demo Time



Demo scenarios

- Show working code assisted by Postman
- Illustrate use of environment variables



Bonus: Some Best Practices I Learned

💡 Tip: environment variables improve security

In addition to making your code flexible and portable, using environment variables also mean **private information** (like API keys) are **not hard-coded**

```
MERAKI_API_URL: str = os.getenv("MERAKI_API_URL")
M_API_KEY: str = os.getenv("M_API_KEY")
M_ORG_ID: str = os.getenv("M_ORG_ID")
WX_API_URL: str = os.getenv("WX_API_URL")
WX_ROOM_ID: str = str(os.getenv("WX_ROOM_ID"))
WX_TOKEN: str = os.getenv("WX_TOKEN")
```

Code Snippet: Env Variables over Sensitive Information

Declarations with
environment
variables

```
MERAKI_API_URL: str = str(runtime_env.MERAKI_API_URL)
M_API_KEY: str = str(runtime_env.M_API_KEY)

time_occurred_iso: str = payload.get('occurredAt')
url_gen_snap: str = f"{MERAKI_API_URL}/devices/{payload.get('deviceSerial')}/camera/generateSnapshot"
```

API request
header and body

```
tx_payload = json.dumps(
    {
        "timestamp": time_occurred_iso,
        "fullframe": "false"
    }
)

tx_headers: dict = {
    'X-Cisco-Meraki-API-Key': M_API_KEY,
    'Content-Type': 'application/json'
}

# Start Generate snapshot function##
logger.debug("get_snap: Request generate snapshot ... ")
# send get_snap request action
try:
    response = requests.post(url_gen_snap, headers=tx_headers, data=tx_payload)
    response_dict: dict = response.json()
    url: str = response_dict.get('url')
    logger.info(f"get_snap Success: snapshot URL generated")
    return (url)
```

Generate snapshot on MV
from timestamp received

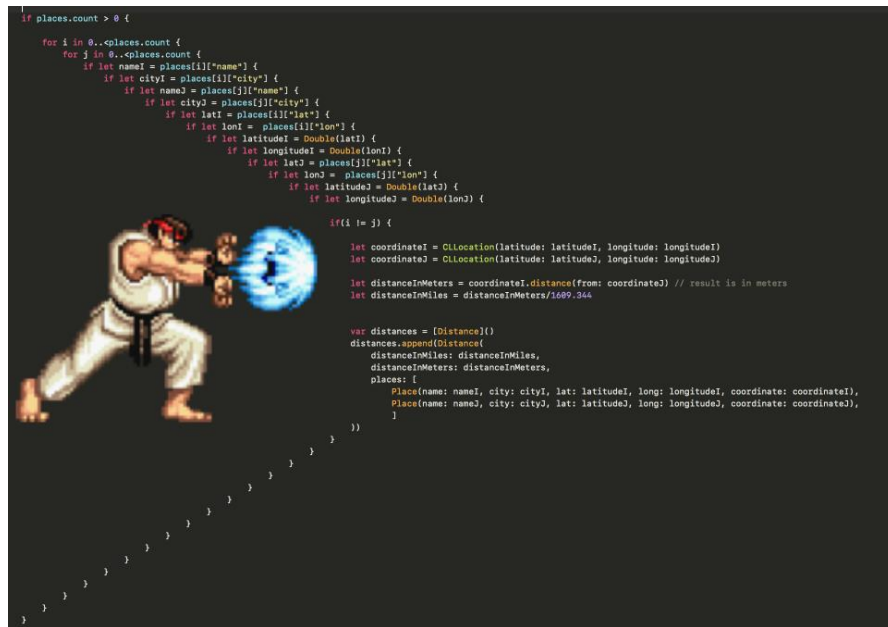
Code snippet: environment file template

```
1 TZ=Australia/Sydney
2 TZ_OFFSET=10
3 APP_PORT=8220
4 HOST_PORT=8220
5 WX_API_URL=https://webexapis.com/v1/messages
6 MERAKI_API_URL=https://api.meraki.com/api/v1
7 WX_ROOM_ID=YOUR_WX_ROOM_ID
8 WX_TOKEN=YOUR_WX_USER_TOKEN
9 M_API_KEY=YOUR_MERAKI_API_KEY
10 M_ORG_ID=YOUR_MERAKI_ORG_ID
```



Tip: Make code modular and readable

- Modular code is flexible and easier to maintain
- Learn design patterns to make code readable



Code Snippet: Event Matching Logic Design

Event classification
dictionary

Event action function call

Event matching logic

```
class eventTypes:
    def __init__(self, alertType: str):
        self.alertType = alertType
        self.event_dict = {
            "motion_alert": self.motion_alert,
            "sensor_alert": self.sensor_alert,
            "settings_changed": self.settings_changed,
            "sensor_automation": self.sensor_automation
        }

    def motion_alert(self, payload: dict):
        import src.motionAlert
        logger.info(f'Motion alert event')
        return src.motionAlert.event_processor(payload)
```

```
def event_match(self, payload: dict):
    event_matched = self.event_dict.get(self.alertType)
    if event_matched:
        return event_matched(payload=payload)
    else:
        return event_handler(payload) #user event_handler as default
```

```
## Triage the incoming payload based on alert type
def webhook_triage(payload: dict):
    logger.info(f'Webhook Triage')

    runtime_env = RuntimeLoader()
    logger.info(f'{runtime_env.env_check()} | {runtime_env.payload_check(payload)}')

    event_type = eventTypes(payload.get('alertTypeId'))
    return event_type.event_match(payload) # Event processing
```

Resources to get you started

- Meraki Developer Portal:
<https://meraki.io>



- Cisco DevNet Portal:
<https://developer.cisco.com>



- Fred's Github Repo:
<https://github.com/MerakiFC/fcm-api-codes>



Keynote Deep Dives

Wednesday
10:30am – 11:30am



Experiences Amplified:
How AI Can Fuel Better Employee and Customer Experiences

Level 1
Room 106



Smart, Secure, Seamless:
Transforming Experiences with Next-Generation Networking

Level 2
Room 204



Harness a Bold New Era:
Transform Data Centre and Service Provider Connectivity

Level 2
Room 203



Securing User to Application and Everything in Between

Level 2
Melbourne Room 2



Unlocking Digital Resilience through Unified Observability

The HUB
Centre Stage

Complete Your Session Evaluations



Complete a minimum of 4 session surveys and the Overall Event Survey to claim a **Cisco Live T-Shirt**.



Complete your surveys in the **Cisco Live mobile app**.



Continue your education



- Visit the Cisco Stand for related demos
- Book your one-on-one Meet the Expert meeting
- Attend the interactive education with DevNet, Capture the Flag, and Walk-in Labs
- Visit the On-Demand Library for more sessions at www.CiscoLive.com/on-demand



Thank you

CISCO *Live!*

#CiscoLiveAPJC

CISCO *Live!*

GO BEYOND

#CiscoLiveAPJC