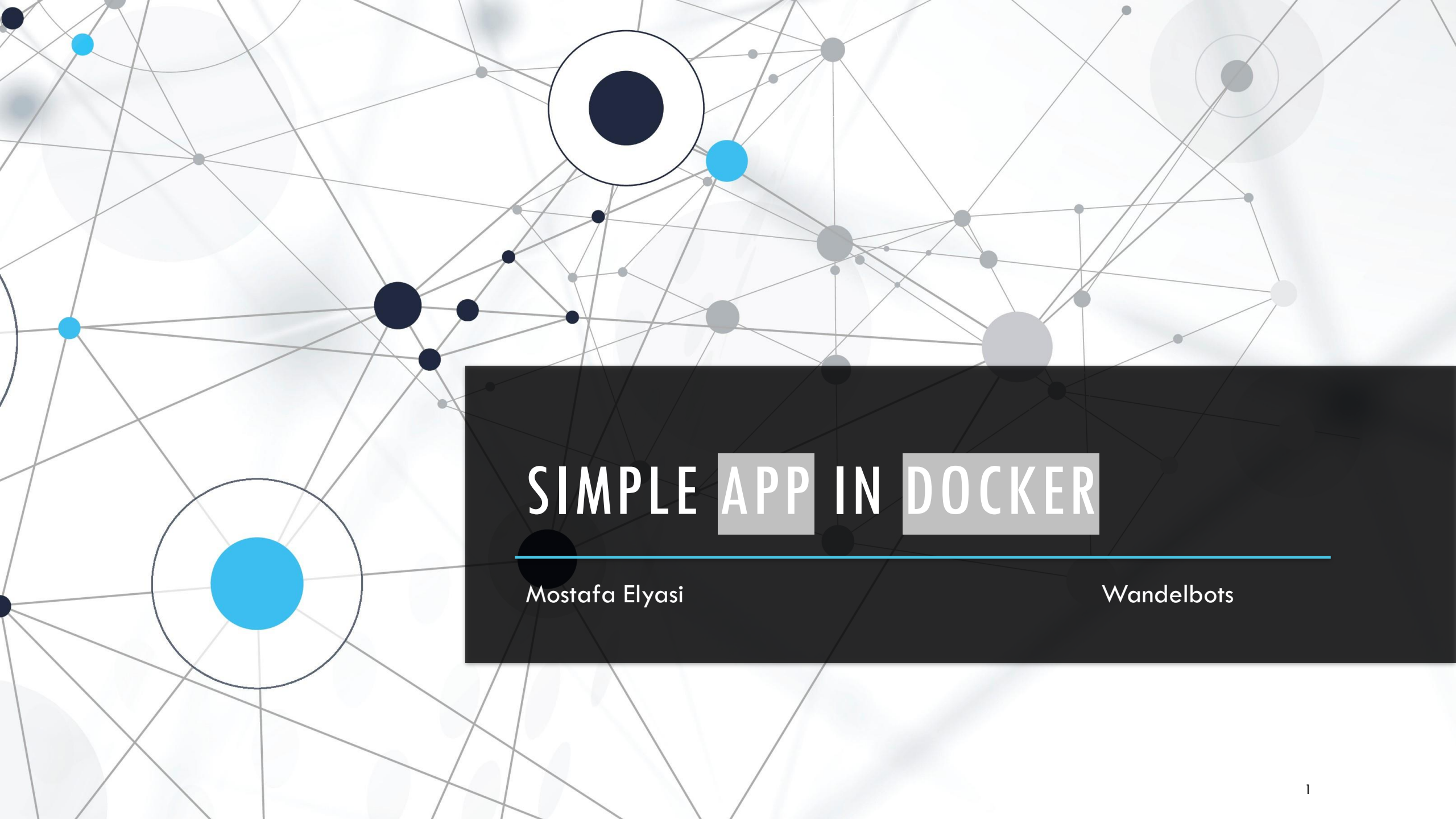


SIMPLE APP IN DOCKER

Mostafa Elyasi

Wandelbots

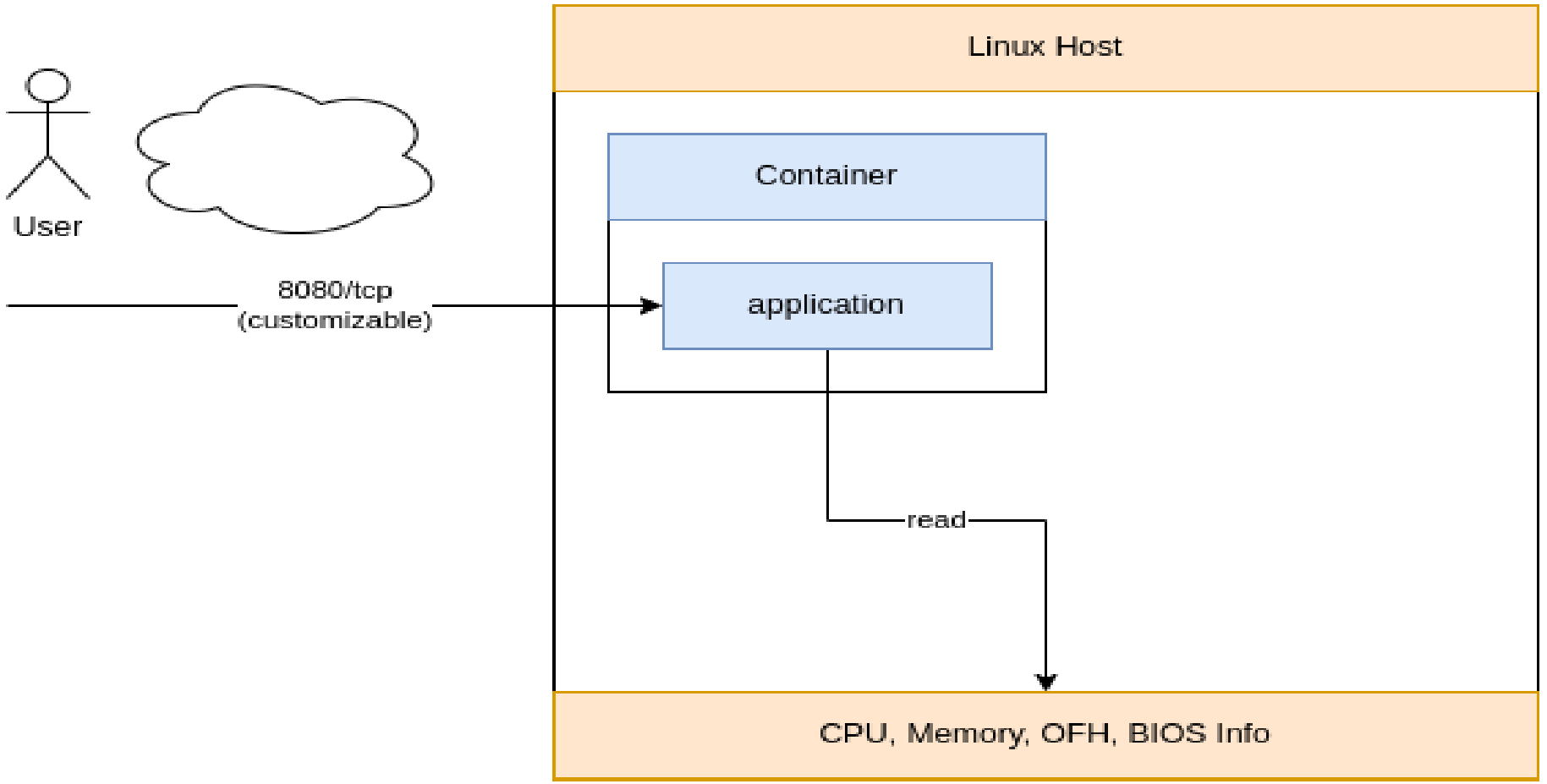
TOPICS

WHAT IS THE PROBLEM?

HOW CAN WE SOLVE IT?

FUTURE AND SCALABLE SOLUTIONS.

CHALLENGE


Linux Host
User
Container
application
8080/tcp
(customizable)
read
CPU, Memory, OFH, BIOS Info

EXPECTATIONS AND REQUESTS!

1. A minimal web application, that can be used to retrieve:
    1. The CPU count
    2. Total memory
    3. Open File Handle Limits
    4. General BIOS information (dmidecode -t 0).
2. Develop a Dockerfile that packages said application and will start it via an entrypointscript.
3. Build the Container Image
4. A way to deploy the application with a customized port in an orchestrator.
5. Deploy the container image on a Linux Host (ex. VM)
6. Develop a minimal test, to check if the application is running and providing useful data.

QUESTIONS?

1-HOW HAVE YOU SOLVED THE USE CASE?

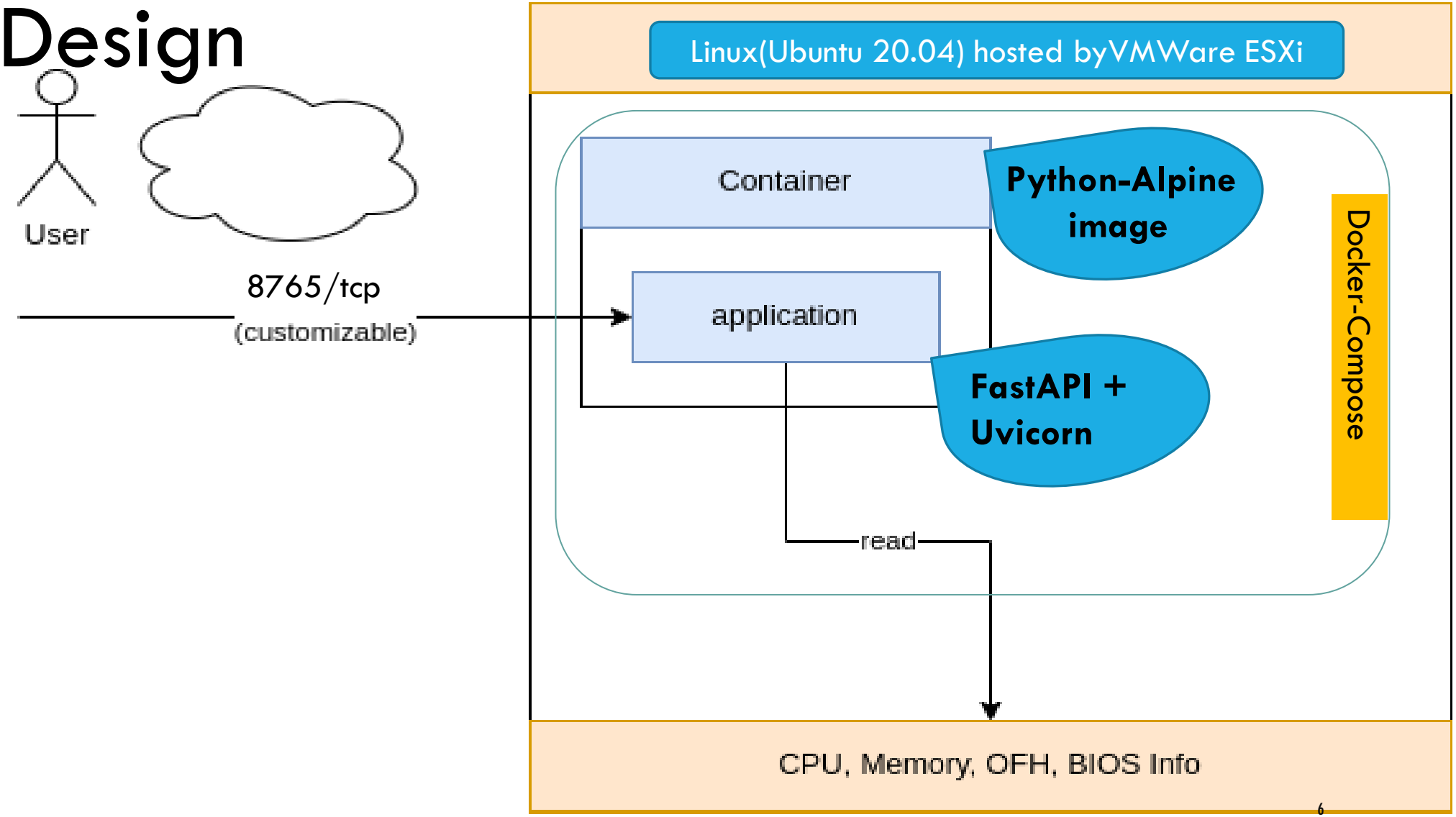
2-Why have you chosen your solution?

3-Which tools do you suggest to solve this?

4- Which aspects of your solution would you improve for a production setup?

1- How have you solved the use case?

Main Design



ENDPOINTS

1.	http://x.x.x.x:8765/hw	show processor type and number of CPU
2.	http://x.x.x.x:8765/meminfo	Memory information
3.	http://x.x.x.x:8765/dmidecode	All the information from the BIOS
4.	http://x.x.x.x:8765/ofh	Maximum Open Files Limit

2- WHY WE HAVE CHOSEN THIS SOLUTION?

SPECS:

- API-Server reacts to requests regarding the Host HardWare information
- It looks like an agent-based log collection
  - Like the Heapster monitoring component of Kubernetes
  - cadvisor agents
- For this purpose, It is preferred to use **simple**, **light** and **fast** tools

2- WHY WE HAVE CHOSEN THIS SOLUTION?

PROS AND CONS:

- Pros:
  A. Fast & Light
  B. Minimum resource usage
  C. Simple configuration and coding
  D. Save the logs history

- Cons:
  A. For some features like bios information require privileged access and permission
    A. It is possible to solve this issue with restricted and read only access
    B. Implementation on some orchestrators like SWARM require customization (doesn't support privileged command)

3- WHICH TOOLS DO YOU SUGGEST TO SOLVE THIS?

FOR DEMO

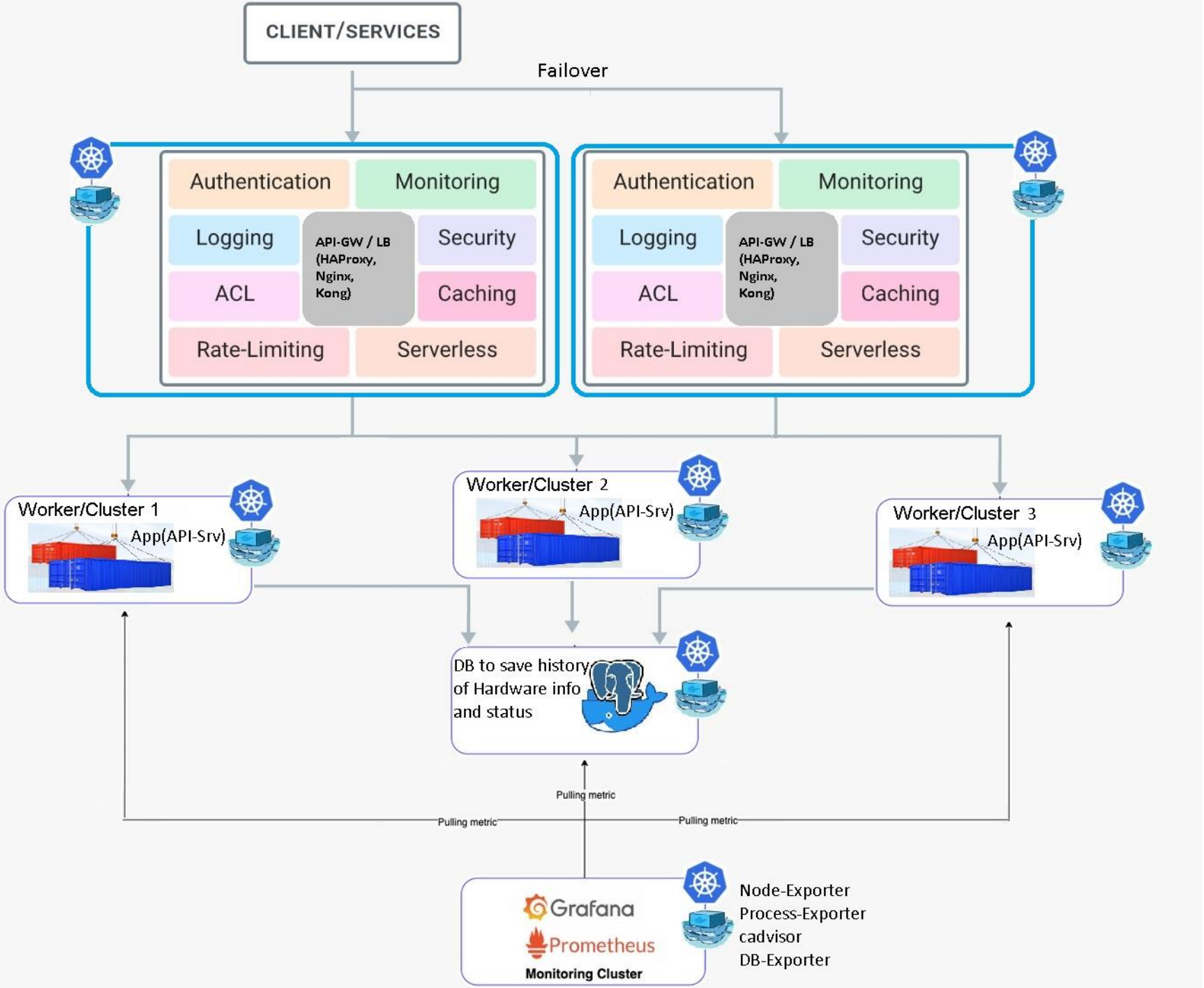
- Python
- FastAPI is the fastest and simplest framework to have an API server for whom knows python
- Docker-compose to manage the container and change the configurations(e.g. ports)
  - Docker-compose is not a full feature orchestrator but is a container manager tool
- Dockerfile to build a customized image
- python-alpine as a light and fast image
- .

FOR PRODUCTION

- For more complex codes and applications with lots of features(which require lots of packages) we can use Django
- Orchestrators like SWARM or K8S are more powerful and scalable
  - Same as docker-compose SWARM can use compose files with the minimum modification
- Dockerfile to build the required image
- Instead of python-alpine it is possible to create a minimum Unix-based and single purpose OS
- DataBases like (PSQL + Redis) to save the history of resource usage
- Monitoring systems (Visualization and alert manager)

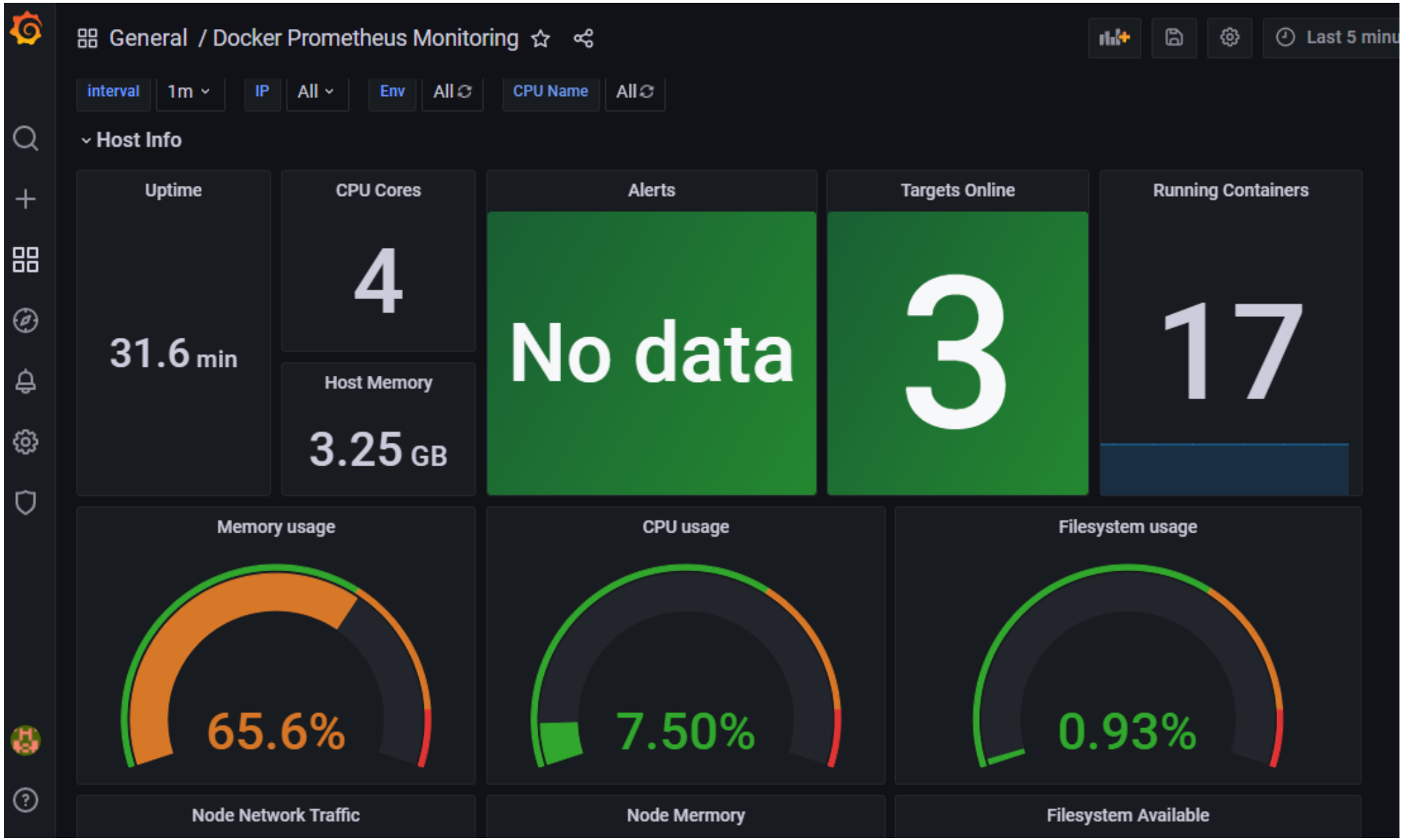
4- WHICH ASPECTS OF YOUR SOLUTION WOULD YOU IMPROVE FOR A PRODUCTION SETUP?

ARCHITECTURE



4- WHICH ASPECTS OF YOUR SOLUTION WOULD YOU IMPROVE FOR A PRODUCTION SETUP?

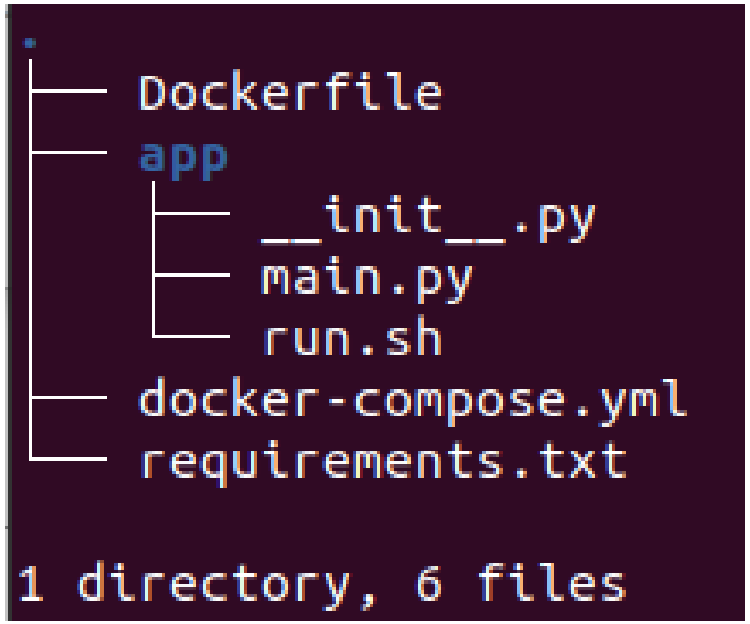
MONITORING



DEPLOYMENT

FILES STRUCTURE

- Management and docker Files are in root folder and the application files are in the app subfolder



FILES CONTENT

- Dockerfile

```
From python:3.10-alpine3.15
MAINTAINER m.elyasi at Wandelbots

# set env variables
ENV PYTHONDONTWRITEBYTECODE 1
ENV PYTHONUNBUFFERED 1

# install dependencies
RUN apk --no-cache add dmidecode && apk --no-cache add curl
RUN /usr/local/bin/python -m pip install --upgrade pip
COPY ./requirements.txt /requirements.txt
RUN pip install --no-cache-dir --upgrade -r /requirements.txt

# set work directory
RUN mkdir /app
WORKDIR /app

#Copy app files
COPY ./app /app

#Run the application
#CMD ["uvicorn", "main:app", "--host", "0.0.0.0", "--port", "80", "--reload"]
ENTRYPOINT [ "sh","/app/run.sh"]

#Create user
#RUN adduser -D user
#USER user
```

- Run.sh

```
#!/bin/bash
/usr/local/bin/uvicorn main:app --host 0.0.0.0 --port 80
```

- Requirements file

```
fastapi>=0.78.0,<0.79.0
uvicorn>=0.17.6,<0.18.0
pydantic>=1.9.1,<1.10.0
py-dmidecode>=0.1.0,<0.2.0
```

DEPLOYMENT

FILES CONTENT

- Main app

```
from fastapi import FastAPI
from dmidecode import DMIDecode
import subprocess
import os
app = FastAPI(title="Simple API")

@app.get("/{item_id}")
async def read_item(item_id: str):
    result={}
    itemid = item_id.lower()
    host_name = ''
    dmi = DMIDecode()
    if os.path.exists('/host_name'):
      host_name = os.popen("cat /host_name").read().strip()
    if itemid == "meminfo" :
      total_memory = os.popen("cat /proc/meminfo | grep -i 'memtotal' | grep -o '[[:digit:]]*' ").rea
      free_memory = os.popen("cat /proc/meminfo | grep -i 'memavailable' | grep -o '[[:digit:]]*' ").
      result = {"Total Memory":total_memory,"Free Memory":free_memory}
    elif itemid == "hw" :
      bios_info = {'Manufacturer:': dmi.manufacturer(), \
              'Model:': dmi.model(), \
              'Firmware:': dmi.firmware(), \
              'Serial number:': dmi.serial_number(), \
              'Processor type:': dmi.cpu_type(), \
              'Number of CPUs:': dmi.cpu_num(), \
              'Cores count:': dmi.total_enabled_cores(), \
              'Total RAM:':'{} GB'.format(dmi.total_ram())}
      result = {"HW_Info from BIOS:":bios_info}
    elif itemid == "dmidecode":
      dmi_info = os.popen("dmidecode -t 0").readlines()
      list_info = []
      for line in dmi_info:
        list_info.append(line.strip())
      result = {"DMIDecode:":list_info}
    elif itemid == "ofh" :
      ofh_info = os.popen("cat /proc/sys/fs/file-max").read().strip()
      result = {"Maximum Open Files Limit:":ofh_info}
    return {"Info of {}".format(host_name):result}
```

{ It is also possible to use 'cat /proc/cpuinfo' }

DEPLOYMENT

FILES CONTENT

- Docker-compose yaml file

```
version: "3.8"
services:
  app:
    build:
      context: .
      dockerfile: ./Dockerfile
#    image: hesaba/simple_api:1.0.1
    restart: "on-failure"
    privileged: true
    cap_add:
      - SYS_RAWIO
    ports:
      - "8765:80"
    volumes:
      - /dev/mem:/dev/mem
      - /:/rootfs:ro
#       - ./app:/app
      - /etc/hostname:/host_name:ro
#    command:
#      - uvicorn app.main:app --host 0.0.0.0
    healthcheck:
      test: curl "http://localhost:80/hw" | grep -iq "Info"
      interval: 10s
      timeout: 5s
      retries: 5
```

```
[redacted] docker-compose ps
  Name         Command          State                          Ports
-------------------------------------------------------------------------------
wa_app_1    sh /app/run.sh    Up (healthy)    0.0.0.0:8765->80/tcp,:::8765->80/tcp
```

DEPLOYMENT

PROCESS STATUS OUTPUT

- Run with docker-compose

```
[redacted] docker-compose ps
  Name         Command          State                    Ports
-------------------------------------------------------------------------------
wa_app_1   sh /app/run.sh   Up (healthy)   0.0.0.0:8765->80/tcp,:::8765->80/tcp
```

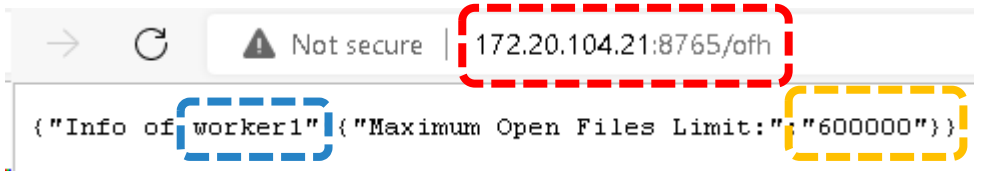
- Docker swarm

```
[redacted] docker stack ps hw_mon
ID             NAME          IMAGE                      NODE         DESIRED STATE   CURRENT STATE                ERROR     PORTS
ovjk5tellsjv   hw_mon_app.1  hesaba/simple_api:1.0.1    k8s-master   Running         Starting about a minute ago
4fp3l1cec4gk   hw_mon_app.2  hesaba/simple_api:1.0.1    worker1      Running         Starting about a minute ago
ou7asjhbr4j0   hw_mon_app.3  hesaba/simple_api:1.0.1    k8s-master   Running         Starting about a minute ago
```

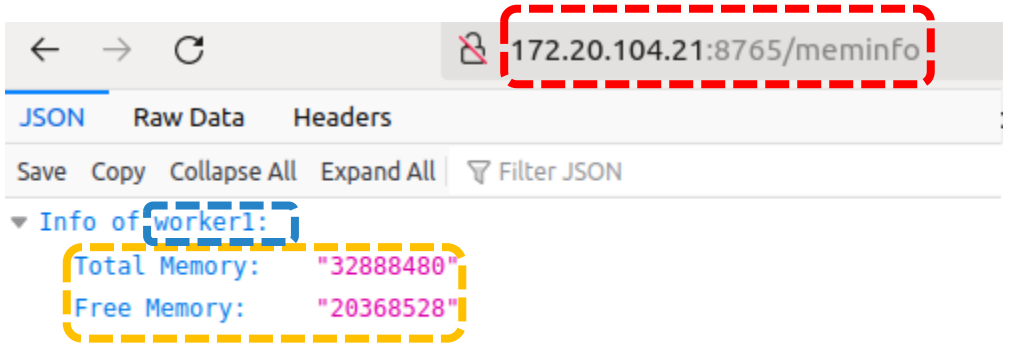
HOW TO USE

ENDPOINTS (/OFH & /MEMINFO)

- open file handler



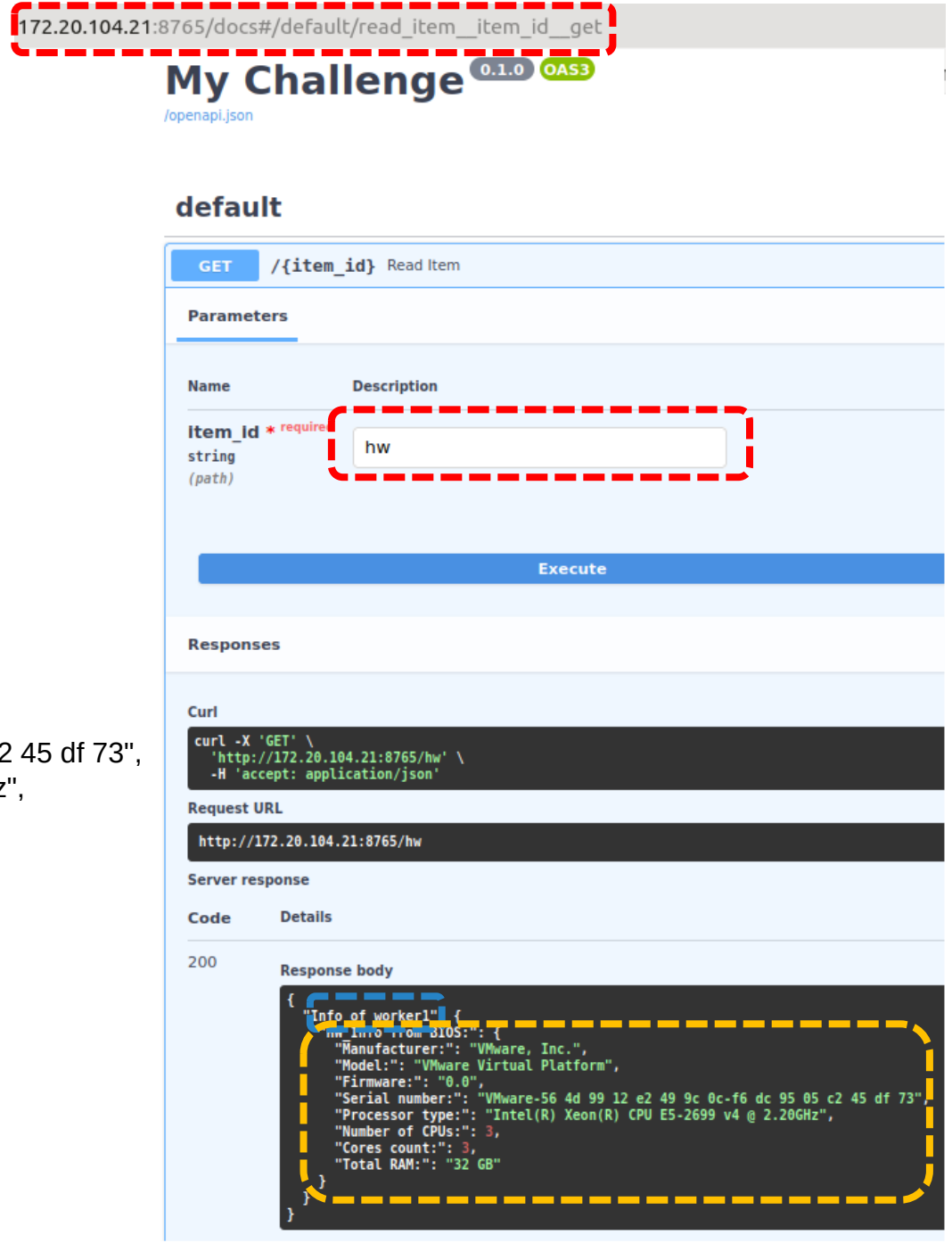
- memory information



HOW TO USE

ENDPOINTS (/HW)

- CPU and platform information

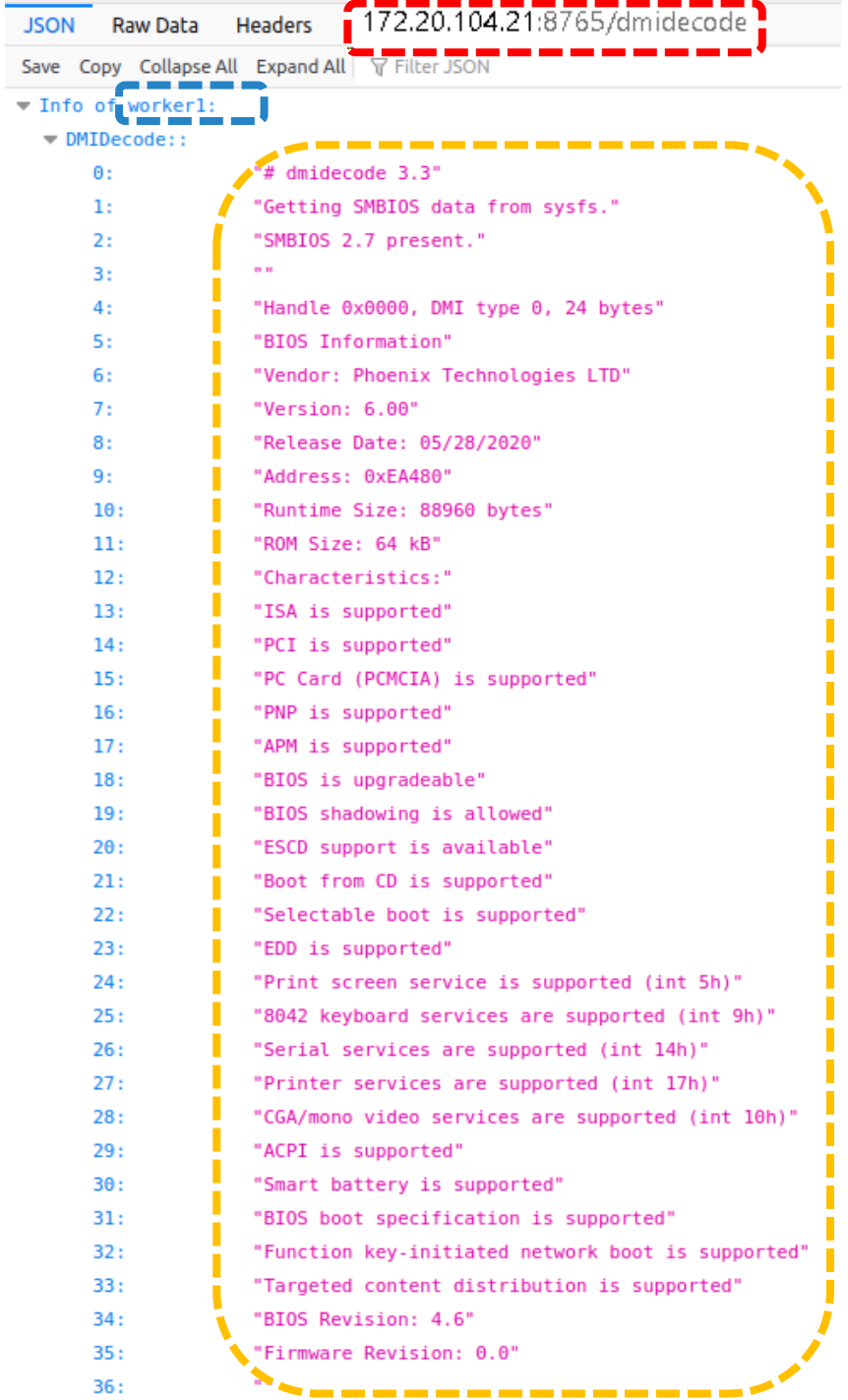


{"Info of worker1":{"HW_Info from BIOS:":
{"**Manufacturer**:":"VMware, Inc.",
"**Model**:":"VMware Virtual Platform",
"**Firmware**:":"0.0",
"**Serial number**:":"VMware-56 4d 99 12 e2 49 9c 0c-f6 dc 95 05 c2 45 df 73",
"**Processor type**:":"Intel(R) Xeon(R) CPU E5-2699 v4 @ 2.20GHz",
"**Number of CPUs**:":3,
"**Cores count**:":3,
"**Total RAM**:":"32 GB"}}}

HOW TO USE

ENDPOINTS (/DMIDECODE)

- BIOS information



THANKS FOR YOUR TIME AND CONSIDERATION

ANYONE...ANYONE...ANYONE...

- Repository on Github
  - https://github.com/mostafaelyasi/simple_api

Contact Information:
Mostafa Elyasi
DevOps Engineer
Tel: +989378152433
Email: mostafa83@gmail.com