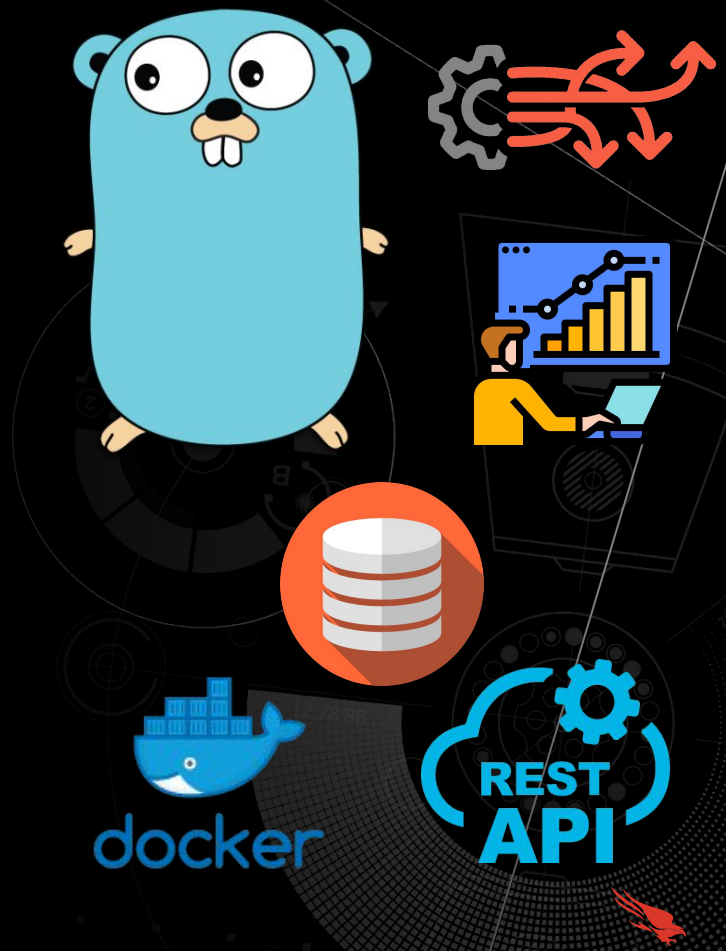
Intro to Docker

CrowdStrike HEROES - Cloud Workshop

CROWDSTRIKE

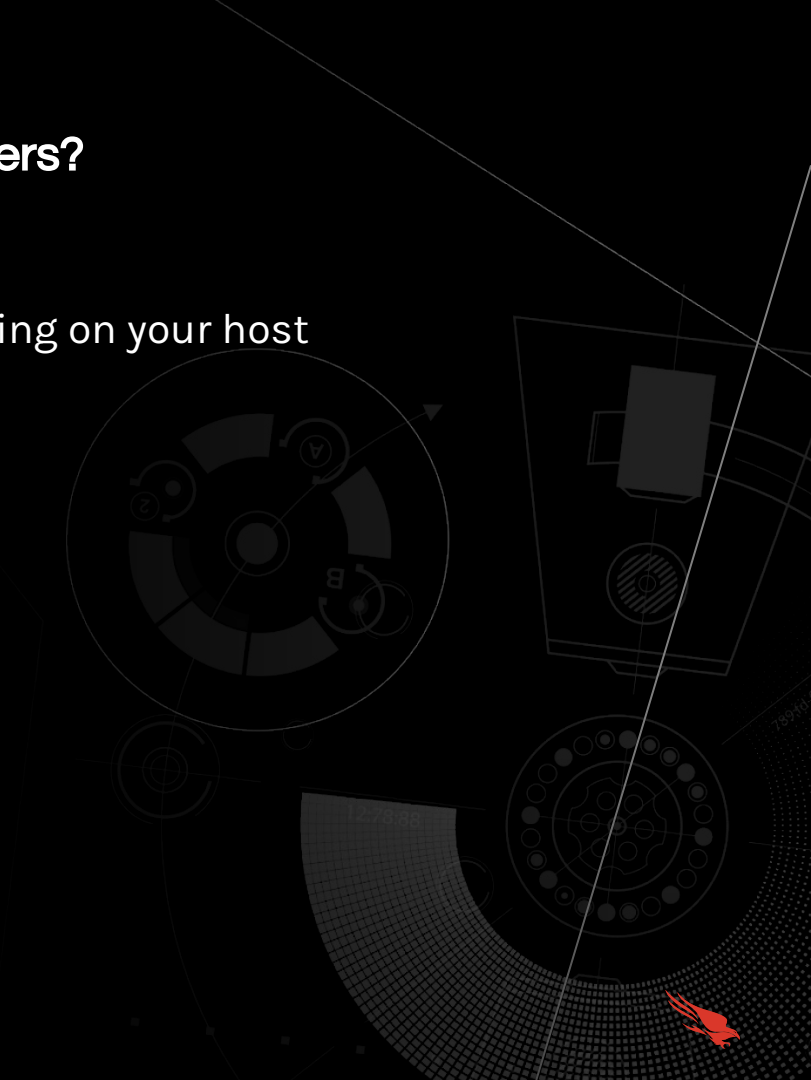
Date	Topic
July 25 - 16:00	Intro to golang
July 26 - 16:00	Intro to golang (continuation)
July 27 - 16:00	Multithreading
July 28 - 16:00	Rest API
July 29 - 16:00	Unit testing, logging and monitoring
August 1 - 16:00	Workshop and Q&A
August 2 - 16:00	**Deployments/Docker**
August 3 - 16:00	Databases
August 4 - 16:00	Databases extended
August 5 - 13:00	Microservices contest (4h with Awards)



CROWDSTRIKE

What are containers?

- Basically a piece of software, app or process running on your host
- Generally have a single purpose
- Provide all necessary dependencies to run an app
- Components: builder, engine and orchestrator
- Why use them?



CROWDSTRIKE

Deploying a http server

```
package main;

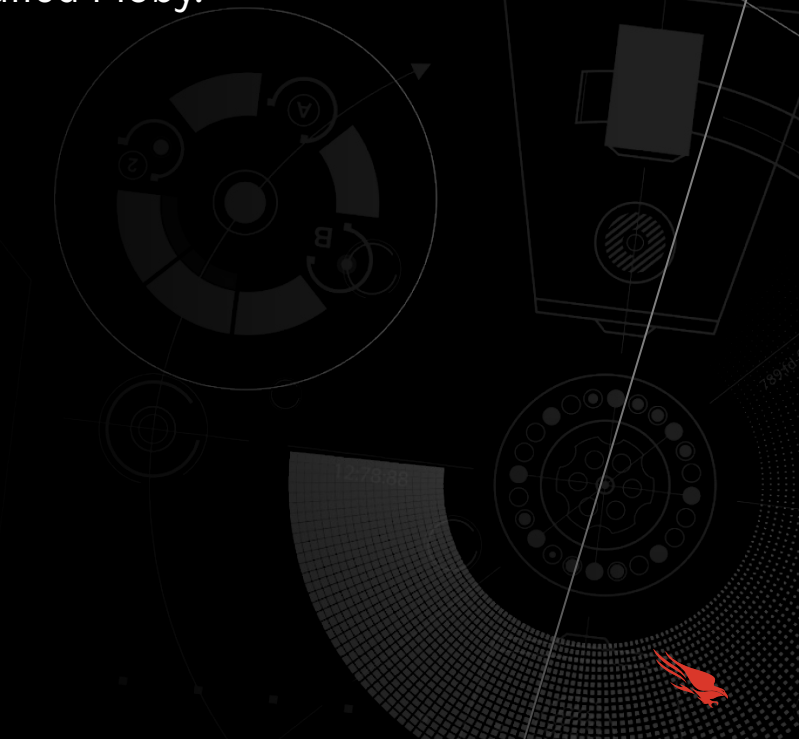
import (
   "fmt"
   "log"
   "net/http"
)

func main() {
    http.HandleFunc("/helloworld", func(w http.ResponseWriter, r *http.Request){
        fmt.Fprintf(w, "Hello, World!")
    })
    fmt.Printf("Server running (port=8080), route: http://localhost:8080/helloworld\n")
    if err := http.ListenAndServe(":8080", nil); err != nil {
        log.Fatal(err)
    }
}
```

CROWDSTRIKE

What is Docker?

- An opensource project part of bigger framework called Moby.
  - builder -> `docker build ...`
  - engine -> `dockerd`
  - orchestrator -> `docker swarm ...`



CROWDSTRIKE

How to run a container

- We can start a container using `docker run`:
  - `E.g. docker run ubuntu:22.04`
  - Obs. Container runs the command and exists, remember single purpose
  - By default runs in foreground, we can use `-d`
  - Options `-i` and `-t` for interactivity
- We can use `docker exec` to run a command inside a container
- If you want to remove it stop it first:
  - `E.g. docker stop quizzical_Heisenberg && docker rm quizzical_heisenberg`

Creating a docker image

- Instruct the builder
- Set of instructions a.k.a. *Dockerfile*
- Configure the environment
- Invoke using:
  - `docker image <params>`
  - `-t <name>`
- Caching

```
FROM golang:1.17

WORKDIR /app

COPY go.* ./

RUN go mod download

COPY *.go ./

RUN go build -o /hello-docker

EXPOSE 8080

ENTRYPOINT ["/hello-docker"]
```

CROWDSTRIKE

Docker images have layers

CROWDSTRIKE

Dockerfile statements (1)

- FROM – provides a base image to construct on top of it
- MAINTAINER – adds authors signature
- RUN – executes a command through shell
- ADD – adds some files/directories/remote files to our image
- COPY – similar to add but only applies to files
- ENV – configures and environment variable

CROWDSTRIKE

Dockerfile statements (2)

- ENTRYPOINT – starting of the expression when you start a container
- CMD – specifies the command to be executed or the args passed to the ENTRYPOINT
- EXPOSE – maps a port into a container
- VOLUME – used to create persistence on data
- WOKRDIR – sets the directory for the future docker commands
- USER – specifies the user the container will run as

CROWDSTRIKE

Docker Volumes (1)

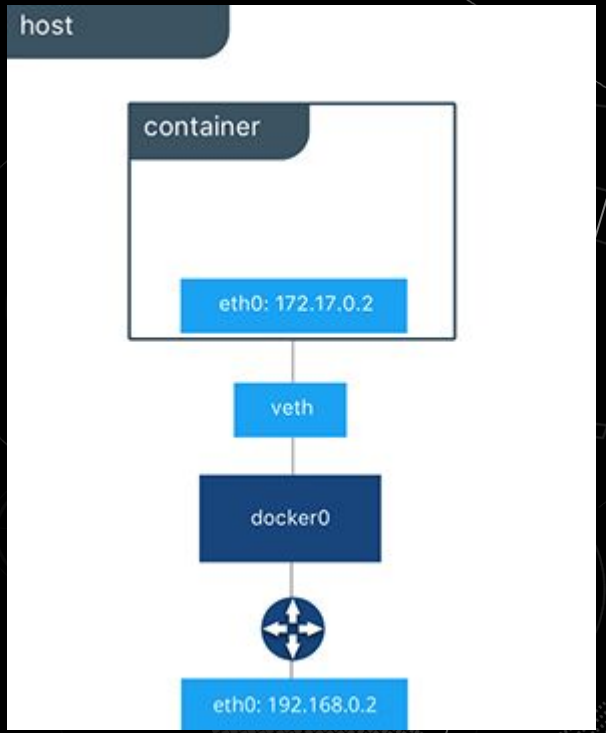
- How can we share data and configs?
- 2 options provided:
  - Bind mounts
    - Mapping a path on the hosts filesystem in our container
    - E.g. we need to initialize a database before using it
  - Docker volumes
    - Similar to bind mounts but completely handled by docker
    - Docker uses its own storage
- Permissions read-write or read-only

Docker Volumes (2)

- For volume management docker volume …
  - `docker volume create ana-are-mere`
  - `docker volume inspect ana-are-mere`
- Option `-v` or `--mount`:
  - `docker run -v $(pwd):/var/opt/project ubuntu:22.04 bash -c "ls /var/opt/project"`
  - `docker run -v ana-are-mere:/var/opt/project ubuntu:22.04 bash -c "ls /var/opt/project"`
  - ```
    docker run -v ana-are-mere:/var/opt/project:ro ubuntu:22.04 \
    bash -c "echo something > /var/opt/project/message.txt"
    ```
    ```

Docker Networking (1)

- Allowing network isolation between services
- Exposing ports to outer world
  - Using `-p <port-on-host>:<port-on-container>`
- Docker bridge
- Command `docker network …`:
  - `docker network create -d bridge myBridge`
  - `docker network inspect myBridge`



Docker Networking (2)

- Attach when booting:
  - `docker run -d --network myBridge ubuntu:22.04 sleep infinity`
- Attach after container is created:
  - `docker run -d --network myBridge ubuntu:22.04 sleep infinity`
  - `docker network connect myBridge 958dc5fb71e0`
- Disconnect afterwards using:
  - `docker network disconnect myBridge 958dc5fb71e0`
- Use `docker network prune` to get rid of unused networks

CROWDSTRIKE

Docker-compose (1)

- Docker CLI doesn’t scale
- Imagine writing the same command again and again
- Automating the deployments with docker-compose
  - YAML files specifying the configs
  - Instructs docker what operations to do
- Syntax
  - Always start with version (e.g. “`version: '3.8'`”)
  - key:value pairs
  - value can be associated with a string, integer, list, etc

Docker-compose (2)

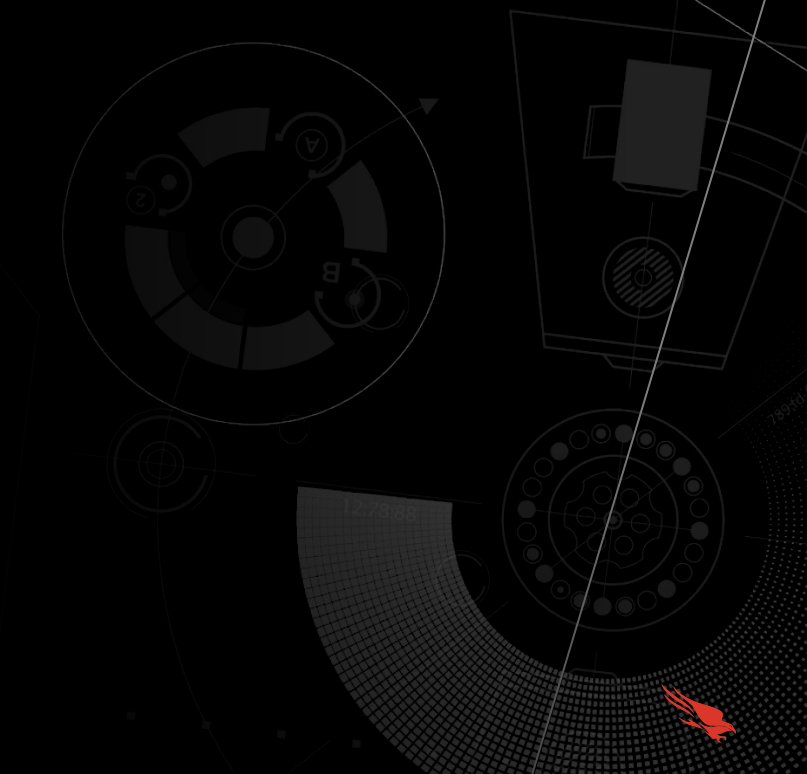
```
services:
    api:
        build: .
        image: hello-docker:latest
        environment:
            NODE_ENV: "development"
            ENVIRONMENT_VARIABLE: "value"
        ports:
            - "18080:8080"
        networks:
            - network-docker
        depends_on:
            - postgres
```

```
postgres:
        image: postgres:latest
        environment:
            POSTGRES_PASSWORD: admin
            POSTGRES_USER: admin
            POSTGRES_DB: admindb
        volumes:
            -
volume-docker:/var/lib/postgresql/data
        networks:
            - network-docker
```

Docker-compose (3)

```
volumes:
    volume-docker: {}

networks:
    network-docker: {}
```

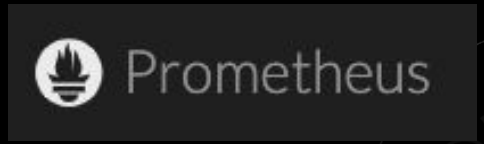


CROWDSTRIKE

Monitoring (1)

- Why is important?
  - A new release show new error logs
  - Maybe there is an increase in the consumed resources
- Inspecting logs:
  - `docker logs <container-id|container-name>`
- Inspecting resources:
  - `docker stats <container-id|container-name>`
  - Without params will display all the stats

Monitoring (2)



- Even though we can do better
  - Centralized metrics collector
  - Centralized aggregator
  - A SPOG to visualise values
- Possible solutions:
  - cAdvisor
  - Prometheus
  - Grafana

Prometheus (1)

- Time series database
- Monitoring and alerting
- Real-time metrics
- HTTP pull model
- [Go package]

Prometheus (2)

- Metrics:
  - Gauge
  - Counter
  - Histogram

```
scrape_configs:
 - job_name: 'prometheus'
   scrape_interval: 5s
   static_configs:
     - targets: ['prometheus:9090']
```

CROWDSTRIKE

Any questions?