Advanced Docker Best Practices

Kostas Kalevras, DevOps Engineer, GUNet

Agenda

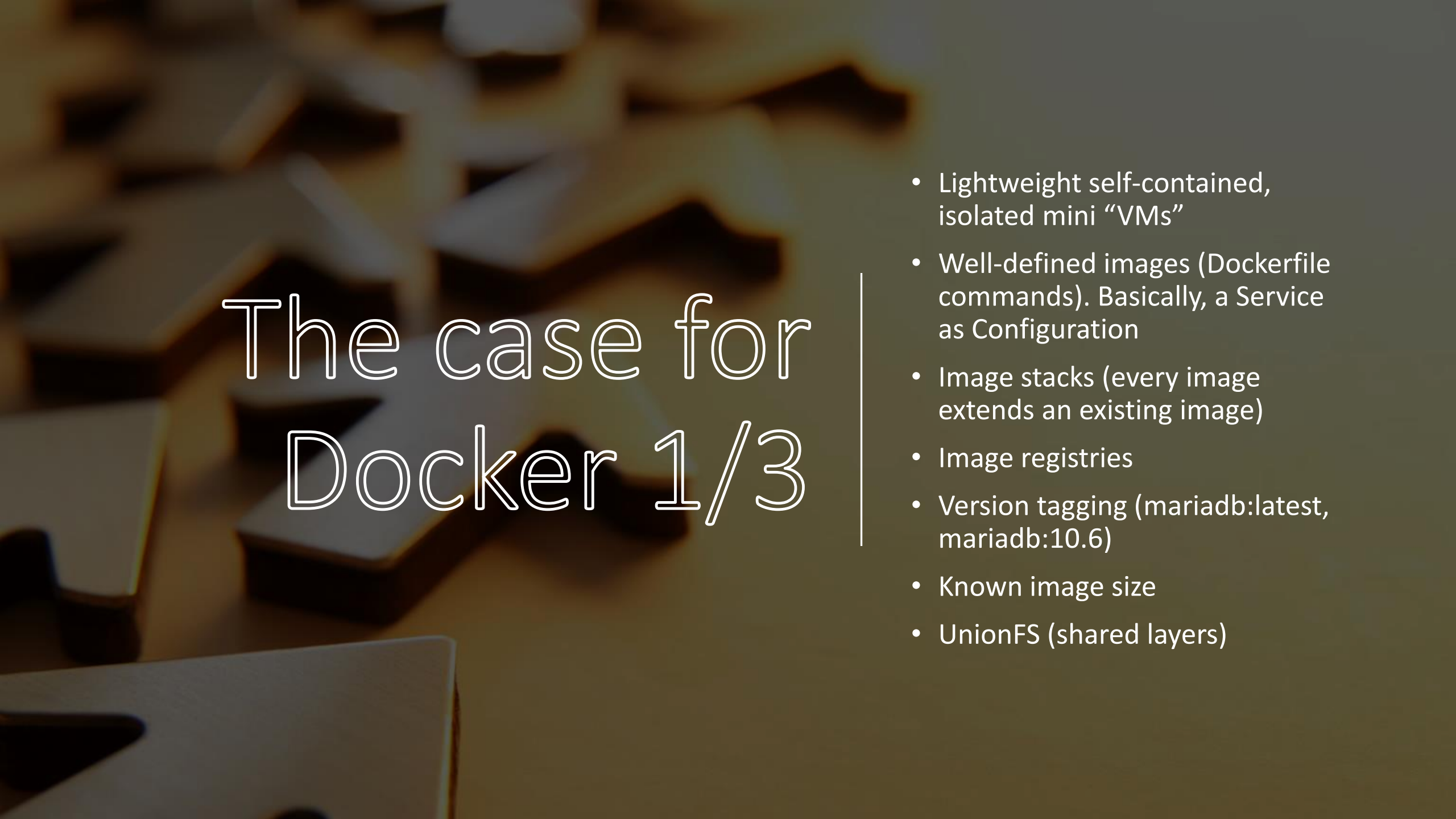
- Part 1
  - Business as Usual
  - Problems
  - The case for Docker
- Part 2
  - Docker best practices

Business as Usual

- 1 VM per service and institutions
- Service Stack
  - Debian OS (JeOS)
  - Debian packages
  - Custom binaries
  - Configuration files
  - Static/dynamic web pages
- Installation - Management
  - Manually
  - Scripts

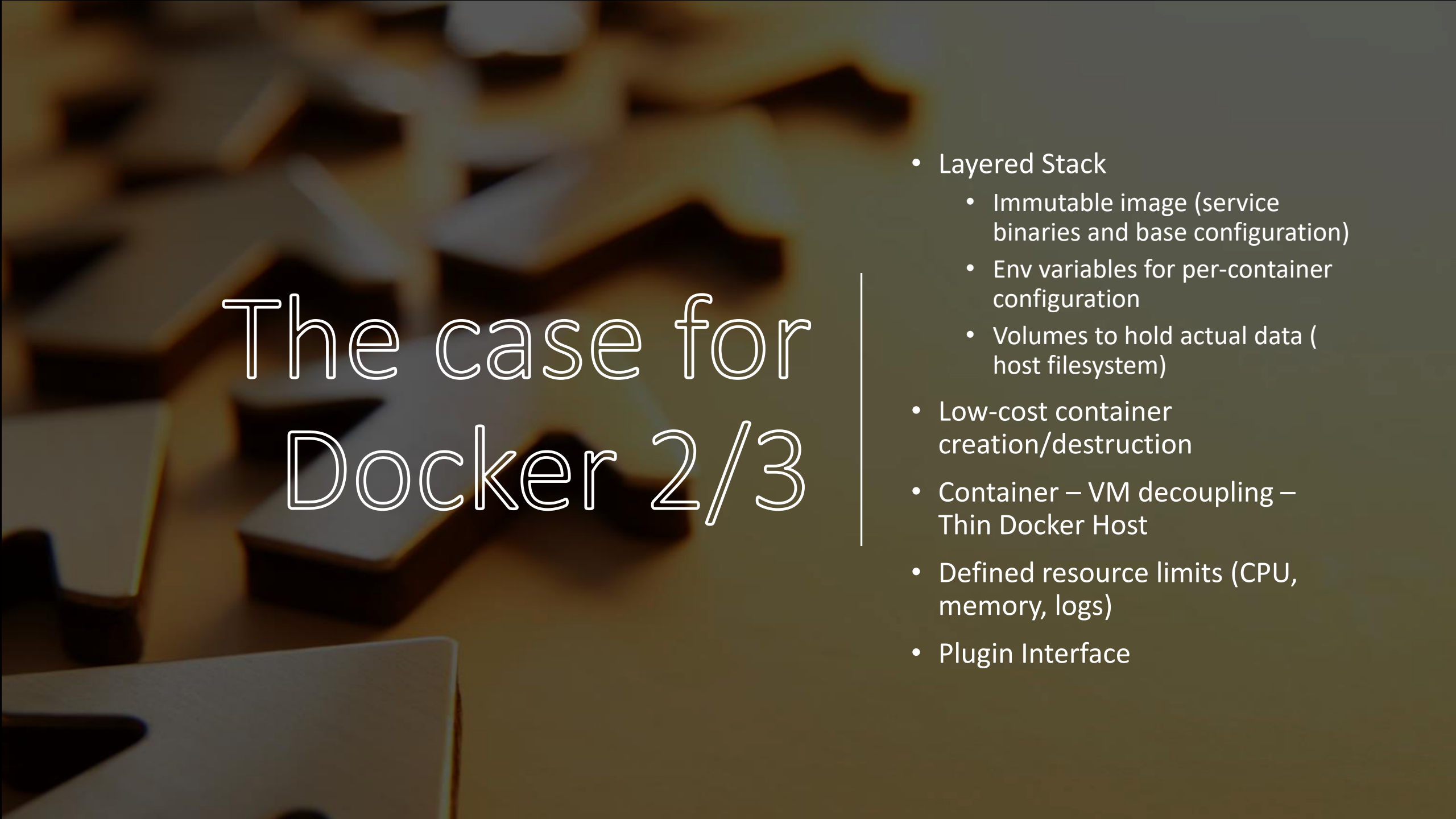
Problems

- Imperative
  - Define how to do the task, not where you want to go
  - Hard to automate
  - Hard to scale
  - Development and Operations are very hard to split
- Service Management
  - VM package management
  - Service binaries
  - Service configuration



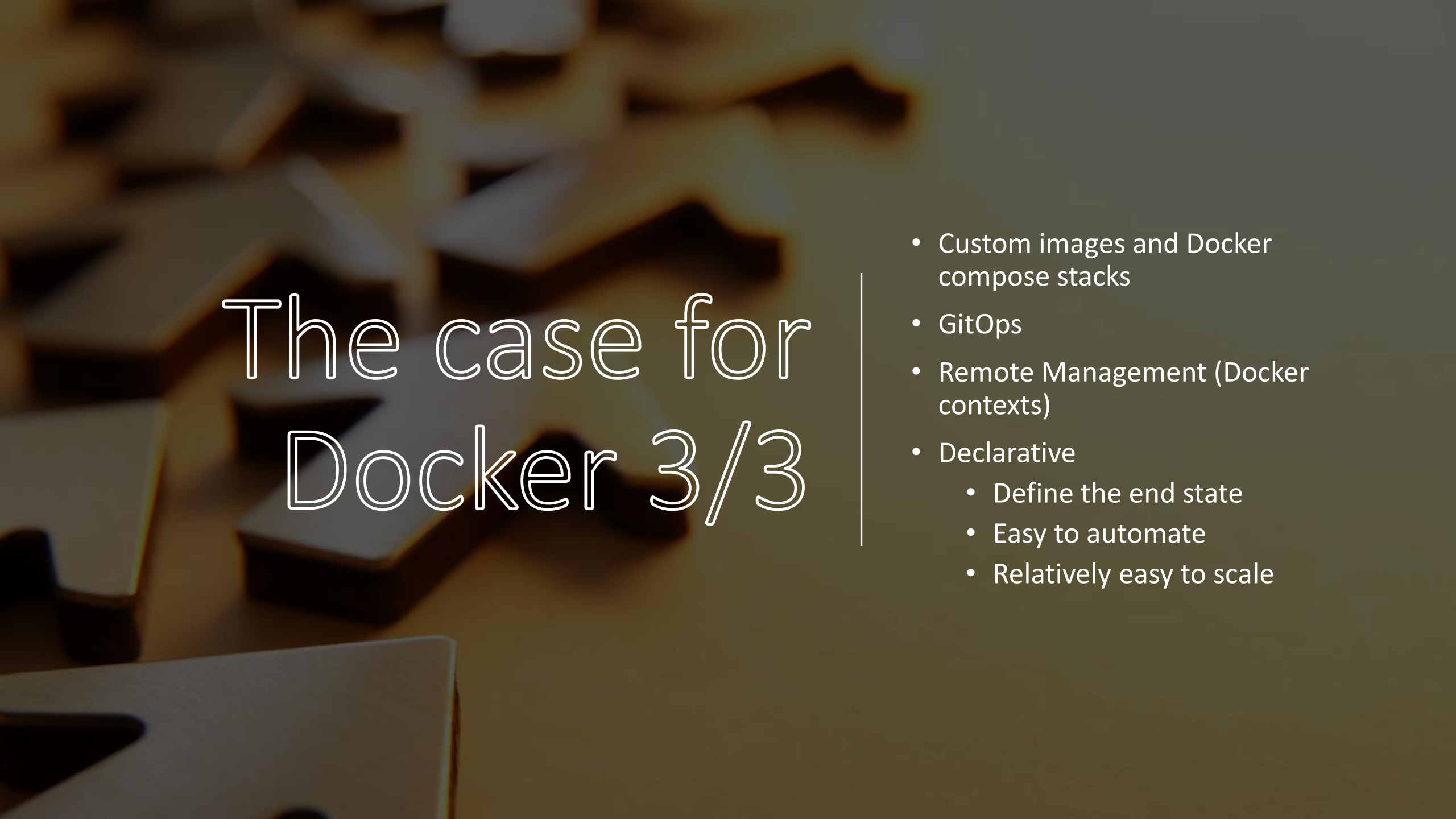
The case for Docker 1/3

- Lightweight self-contained, isolated mini “VMs”
- Well-defined images (Dockerfile commands). Basically, a Service as Configuration
- Image stacks (every image extends an existing image)
- Image registries
- Version tagging (mariadb:latest, mariadb:10.6)
- Known image size
- UnionFS (shared layers)



The case for Docker 2/3

- Layered Stack
  - Immutable image (service binaries and base configuration)
  - Env variables for per-container configuration
  - Volumes to hold actual data (host filesystem)
- Low-cost container creation/destruction
- Container – VM decoupling – Thin Docker Host
- Defined resource limits (CPU, memory, logs)
- Plugin Interface



The case for Docker 3/3

- Custom images and Docker compose stacks
- GitOps
- Remote Management (Docker contexts)
- Declarative
  - Define the end state
  - Easy to automate
  - Relatively easy to scale

Part 2

Docker best practices

Distributions – Image slim versions

- Use the distribution you know
  - ie Debian, not Alpine
  - You gain in size but use what you already know
- Try using ***slim*** versions and add any packages you need
  - Bullseye-slim vs bullseye (80MB < 125MB)
  - Python slim vs vanilla python (130MB < 920MB)
  - Node slim vs vanilla node (190MB < 910MB)

RUN wget http://<site>/package.zip RUN unzip package.zip RUN rm -f package.zip
RUN wget http://<site>/package.zip RUN unzip package.zip RUN rm -f package.zip

Layers

- Each RUN creates a **new** layer in UnionFS
  - You **cannot** delete previous layers
  - The rm below will have no effect

```
RUN wget http://<site>/package.zip
RUN unzip package.zip
RUN rm -f package.zip
```

- Run all commands in **one** layer:

```
RUN wget http://<site>/package.zip && unzip package.zip && rm
-f package.zip
```

- Start with the most basic and less changed layers first (Build caching, shared layers)
  - Install packages
  - Code
  - Configuration files
- When installing packages always clean-up in the same RUN layer

RUN wget http://<site>/package.zip RUN unzip package.zip RUN rm -f package.zip
RUN wget http://<site>/package.zip RUN unzip package.zip RUN rm -f package.zip

Multi-stage builds

- Don’t use large images or install packages only for building
- Use multi-stage building: A **separate** container is used as builder
- Example: Use eclipse JDK to build JRE while the base image is *debian:bullseye-slim*

```
FROM eclipse-temurin:17-jdk as jre-build

# Create a custom Java runtime
RUN $JAVA_HOME/bin/jlink \
         --add-modules ALL-MODULE-PATH \
         --strip-debug \
         --no-man-pages \
         --no-header-files \
         --compress=2 \
         --output /javaruntime

FROM debian:bullseye-slim

# Copy Java from OpenJDK
RUN mkdir -p /javaruntime
ENV JAVA_HOME=/opt/java/openjdk
ENV PATH "${JAVA_HOME}/bin:${PATH}"
COPY --from=jre-build /javaruntime
$JAVA_HOME
```

RUN wget http://<site>/package.zip RUN unzip package.zip RUN rm -f package.zip
RUN wget http://<site>/package.zip RUN unzip package.zip RUN rm -f package.zip

Docker entry point

- Create a Docker entrypoint to run various tasks on startup and finally call your application using *exec*
- In the *Dockerfile* you set the ***ENTRYPOINT*** to point to your entrypoint script and pass arguments using the ***CMD*** instruction

```
ENTRYPOINT [ "/usr/local/bin/docker-entrypoint.sh" ]
CMD [ "--ini", "/etc/privayidea/uwsgi/uwsgi.ini" ]
```

- Benefits
  - Signals go to your application, not the shell
  - Setup things on first run
  - Substitute environment vars in configuration files using *envsubst*
  - Decrypt keys using a passphrase passed as env var

```
#!/bin/bash

# Create database if it does not exist (in the case of sqlite)
grep SQLALCHEMY_DATABASE_URI ${PI_CONFIG} | grep -q sqlite && \
    [ ! -f /etc/privacyidea/db/data.sqlite ] && \
    echo "Creating sqlite database.." && ${PI_MANAGE} create_tables

# Decrypt keys
ENCKEY=$(grep PI_ENCFILE ${PI_CONFIG}| awk '{print $3}'|tr -d \')
ENCKEY_ENC=$(grep PI_ENCFILE ${PI_CONFIG}| awk '{print $3}'|tr -d \'|sed -e
's/$/.aes/')

PRIVKEY=$(grep PI_AUDIT_KEY_PRIVATE ${PI_CONFIG}| awk '{print $3}'|tr -d \')
PRIVKEY_ENC=$(grep PI_AUDIT_KEY_PRIVATE ${PI_CONFIG}| awk '{print $3}'|tr -d
\'|sed -e 's/.pem$/.key/')

if [[ -f ${ENCKEY_ENC} ]]; then
    echo "Decrypting enckey '${ENCKEY_ENC} to ${ENCKEY} .."
    openssl enc -aes-256-cbc -md sha512 -pbkdf2 -iter 100000 -d -salt -pass
pass:${ENC_PASSPHRASE} -in ${ENCKEY_ENC} -out ${ENCKEY}
fi

if [[ -f ${PRIVKEY_ENC} ]]; then
    echo "Decrypting private key ${PRIVKEY_ENC} to ${PRIVKEY} .."
    openssl rsa -passin pass:${ENC_PASSPHRASE} -in ${PRIVKEY_ENC} -out
${PRIVKEY}
fi

# Use template to substitute env variables
envsubst </etc/privacyidea/templates/main.json >/etc/privacyidea/main.json

exec /usr/local/bin/uwsgi "$@"
```

RUN wget http://<site>/package.zip RUN unzip package.zip RUN rm -f package.zip

Dependency stack

- In Docker Compose we can setup a dependency stack
  - A service only starts if the dependencies have started
  - We can add a **healthy** requirement

```
depends_on:
  db:
    condition: service_healthy
  proxy:
    condition: service_healthy
```

- Cron Jobs
  - Docker does not include Cron jobs
  - Kubernetes does
  - We can approximate with a script that sleeps between runs

```
-- docker-compose.rotateaudit.yaml --
version: '3.7'

services:
  rotateaudit:
    image: gunet/privacyidea:${INSTITUTION:-latest}
    entrypoint: /usr/local/bin/rotate_auditlog.sh
    environment:
      - ROTATE_DAYS=120
      - ROTATE_SLEEP=86400
    restart: unless-stopped
    network_mode: "none"

-- rotate_auditlog.sh --
#!/bin/bash

if [[ ! -v ROTATE_DAYS ]]; then
    echo "No ROTATE_DAYS env variable available!"
    exit 1
fi
if [[ ! -v ROTATE_SLEEP ]]; then
    echo "No ROTATE_SLEEP env variable available!"
    exit 1
fi

while true
do
    echo "Rotating audit log, trimming anything older than ${ROTATE_DAYS}"
    ${PI_MANAGE} audit rotate_audit --age ${ROTATE_DAYS}
    echo "Sleeping for ${ROTATE_SLEEP} secs.."
    sleep ${ROTATE_SLEEP
done
```

RUN wget http://<site>/package.zip RUN unzip package.zip RUN rm -f package.zip

Dynamic Routing

- Docker Compose stack with multiple services
- Traefik dynamic routing
- Configuration using Docker Compose labels
- Routing
  - HTTP (Host header)
  - HTTPS (SNI)
- Client-IP
  - HTTP headers (X-Forwarded-For)
  - PROXY2 protocol (SSL)
  - Apache supports both
- Middleware
  - BasicAuth
  - IP Whitelisting

```
guest:
  image: ghcr.io/gunet/wifi-guest
  labels:
    # Explicitly instruct Traefik to expose this service
    - traefik.enable=true
    # Router configuration
    ## One should replace the `guest` name with the name of their service
    ## Enable TLS and passthrough
    - traefik.tcp.routers.guest.tls=true
    - traefik.tcp.routers.guest.tls.passthrough=true
    ## Listen to the `web` entrypoint
    - traefik.tcp.routers.guest.entrypoints=https
    ## Rule based on the HostSNI of the request
    - traefik.tcp.routers.guest.rule=HostSNI(`www.gunet.gr`)
    # Service configuration
    ## 443 is the port that the guest container is listening to
    - traefik.tcp.services.guest.loadbalancer.server.port=443
```

RUN wget http://<site>/package.zip RUN unzip package.zip RUN rm -f package.zip
RUN wget http://<site>/package.zip RUN unzip package.zip RUN rm -f package.zip

GitHub integration

- GitHub Container Registry (GHCR)
- Integrate repo with Docker images
- Inherit access rights
  - Private repo has private packages
  - Only users with access to repo can access packages
  - Easy integration in GitHub Actions

RUN wget http://<site>/package.zip RUN unzip package.zip RUN rm -f package.zip
RUN wget http://<site>/package.zip RUN unzip package.zip RUN rm -f package.zip

GitHub Actions

- Build images using GitHub actions
- Build on
  - Push
  - Pull Request
  - Manual invocation
- Common restrictions
  - Branch
  - Folders/files
- Integrated with GHCR
- Workflow stacks
- Can be used for
  - Tests
  - Linting

RUN wget http://<site>/package.zip RUN unzip package.zip RUN rm -f package.zip
RUN wget http://<site>/package.zip RUN unzip package.zip RUN rm -f package.zip

GitHub Actions Example

```yaml
name: build-grafana
run-name: Build Grafana Docker image

on:
  workflow_dispatch:
  push:
    branches:
      - "main"
    paths:
      - 'grafana/**'

jobs:
  build-grafana:
    uses: ./.github/workflows/all_imagesbuild_worker.yml
    with:
      image-name: grafana
      directory-name: grafana
      dockerfile: Dockerfile
   secrets: inherit
```

```yaml
jobs:
  build-base:
    runs-on: ubuntu-latest
    strategy:
      matrix:
        institution: [aegean, asfa, aueb, ihu, ionio, latest, staging, unipi, uoi, uop, uowm]
    steps:
      - name: Print inputs
        run: |
          echo "image-name: ${{ inputs.image-name }}"
          echo "directory-name: ${{ inputs.directory-name }}"
          echo "institution: ${{ matrix.institution }}"
          echo "dockerfile ${{ inputs.dockerfile }}"
      - name: Set timezone
        uses: zcong1993/setup-timezone@master
        with:
          timezone: "Europe/Athens"
      - name: Get date
        id: date
        run: echo "push_date=$(date '+%H:%M@%d/%m/%Y')" >> $GITHUB_ENV
      - name: Checkout
        uses: actions/checkout@v3
      - name: Login to GitHub Container Registry
        uses: docker/login-action@v2
        with:
          registry: ghcr.io
          username: ${{github.actor}}
          password: ${{secrets.GITHUB_TOKEN}}
      - name: Set up Docker Buildx
        uses: docker/setup-buildx-action@v2
      - name: Build and push Base image
        uses: docker/build-push-action@v3
        with:
          context: ./${{ inputs.directory-name }}
          file: ./${{ inputs.directory-name }}/${{ inputs.dockerfile }}
          build-args: |
            INSTITUTION=${{ matrix.institution }}
          push: true
          tags: ghcr.io/gunet/${{ inputs.image-name }}:${{ matrix.institution }}
          labels: gr.gunet.${{ inputs.image-name }}.pushdate=${{ env.push_date }}
          cache-from: type=gha
          cache-to: type=gha,mode=max
```

Thank you

Comments - Questions