



Understanding Microservices Architecture

Maaïke van Putten
Software developer & trainer
www.biltup.com



Y

Session 1

Introduction to
microservices

d

Session 2

Comparing
microservices and
monolithic
architecture

d

Session 3

Building and
containerizing
microservices

E

Session 4

Managing data in
microservices



Building and containerizing microservices

Learning objectives



d

Be able to **build a basic microservice in Python** and **containerize it using Docker**.

s

Implement **interservice communication** within Docker.



Create a simple API for microservices that **communicate** with each other **using REST**.

Implementing microservices

- Actual creation of the microservices



Designing microservices



Domain-driven design principles



Clear boundaries between the services

M

Choosing the appropriate language and frameworks



Designing RESTful APIs

D





API documentation

RESTful APIs

- A.k.a. REST API
- Used by two systems to exchange information
- API that conforms to REST architecture principles, includes:
 - Client/server style
 - Stateless
 - Cacheable
 - Uniform interface
 - Resource base (URI)
- HTTP Methods: GET, POST, PUT, DELETE.
- Using JSON for data exchange.



Fundamentals of Python Flask

-  Web applications framework (lightweight)
-  Can be used for writing small web application, including the interface combined with (for example) template engine jinja2
-  Great for creating REST APIs
-  Easier to learn than the obvious heavier alternative Django

Demo

Create two microservices



Containerization



Wrapping an application plus its environment in a container



Allows the application to run in a virtualized environment



The one that we'll be working with is Docker



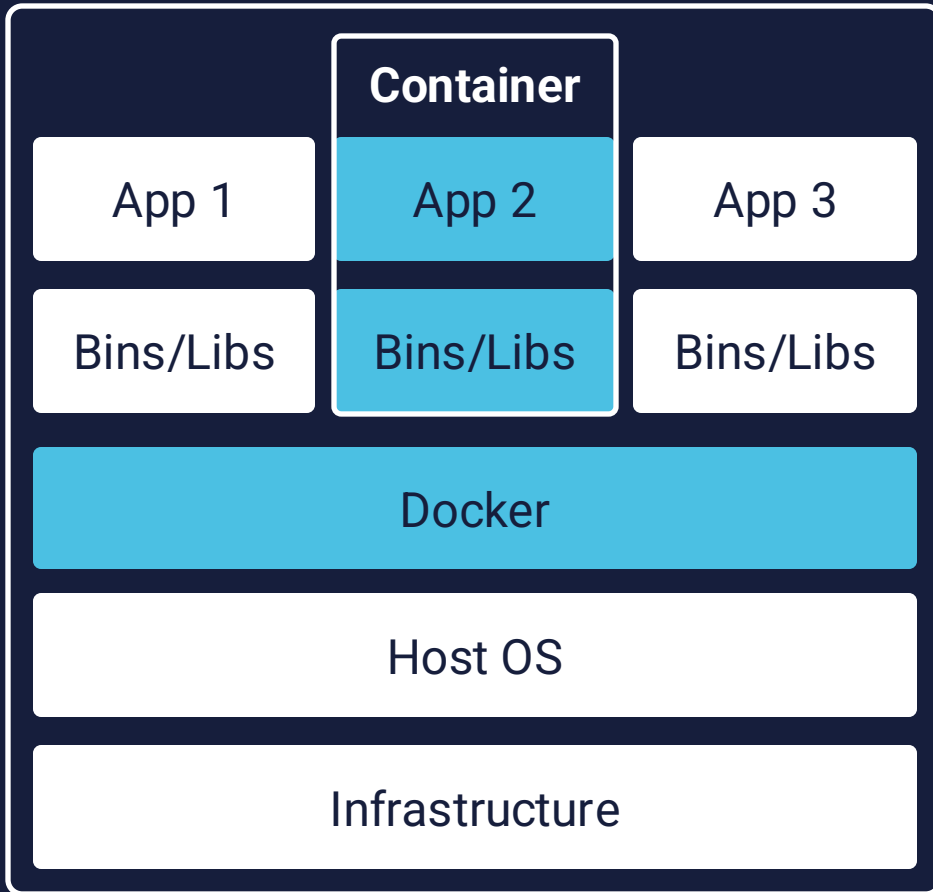
In docker you create images that contains the app, properties, etc.



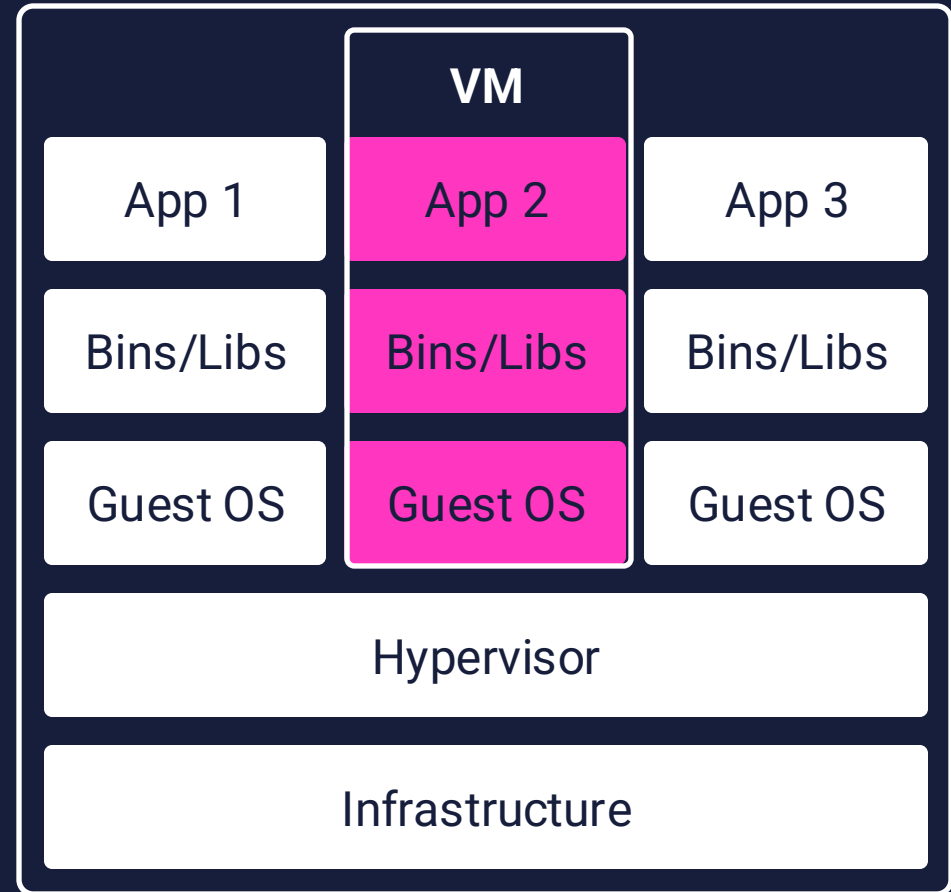
You can then run that image, and a running image is called a container

Containers vs VMs

Docker containers



Virtual machines



Docker image

- Executable package that includes everything needed to run an application
- For example, a Java microservice docker image might have:
 - JVM
 - Web server (such as Tomcat)
 - Any additional dependencies (such as database drivers)
 - Your JAR with the actual applications



Images vs containers

- Running images results in:
 - An in-memory instance of the image
 - This instance is called a container
 - One image can create many containers



Demo

Containerize two microservices



Exercise

Implement and containerize two microservices from the design



Service communication

- Multiple options, including:
 - RESTful APIs
 - gRPC
 - Message queues
 - Event streaming platforms
- Synchronous vs asynchronous
- Performance vs complexity
- Data format
- Service coupling
- Resilience





What we'll use

F

RESTful calls between services

D

Environment variables and docker networking

V

Adding docker compose for easier and less error prone spinning up

Docker compose

- So far, we've been spinning up the containers manually (and we only have 2!)
- It's kind of error prone, especially if there are more containers
- Docker compose can be used to manage multiple containers, the config for docker compose holds:
 - List of services (build image and run container)
 - Volumes or mounts that are needed by the containers
 - Linking the containers together if applicable



Exercise

Implementing interservice
communication



Which of the following describes the purpose of containerizing microservices using Docker?

{A}

To package microservices and their dependencies for consistent deployment across different environments.

{B}

To improve the performance of microservices by compiling them into native code.

{C}

To convert monolithic applications into microservices automatically.

{D}

To enable microservices to share the same memory space for faster communication.

Which of the following describes the purpose of containerizing microservices using Docker?

{A}

To package microservices and their dependencies for consistent deployment across different environments.

{B}

To improve the performance of microservices by compiling them into native code.

{C}

To convert monolithic applications into microservices automatically.

{D}

To enable microservices to share the same memory space for faster communication.

What is the role of Docker Compose in managing microservices?

{A}

It replaces the need for Dockerfiles by automating image creation.

{B}

It provides a way to define and run multicontainer Docker application.

{C}

It converts Docker images into virtual machines for deployment.

{D}

It automatically spins up failing containers with a new instance to increase uptime.

What is the role of Docker Compose in managing microservices?

{A}

It replaces the need for Dockerfiles by automating image creation.

{B}

It provides a way to define and run multicontainer Docker application.

{C}

It converts Docker images into virtual machines for deployment.

{D}

It automatically spins up failing containers with a new instance to increase uptime.

Why would you use an environment variable in a Dockerfile or Docker Compose configuration?

{A}

To hardcode sensitive information like passwords into the container image.

{B}

To define the file system hierarchy within a container.

{C}

To make sure the information is not lost when the container crashes.

{D}

To store and manage configuration data external to the application code.

Why would you use an environment variable in a Dockerfile or Docker Compose configuration?

{A}

To hardcode sensitive information like passwords into the container image.

{B}

To define the file system hierarchy within a container.

{C}

To make sure the information is not lost when the container crashes.

{D}

To store and manage configuration data external to the application code.



Next up:

Managing data in microservices



Questions or suggestions?

maaikejvp@gmail.com

See you tomorrow!



www.biltup.com