

MICROSERVICES


GITHUB



- Sourcen mit Beispielen zum Skript finden sie unter <https://github.com/florianwachs/FHRWebservices>

WAS IST EIN MICROSERVICE ?

WAS IST EIN MICROSERVICE?

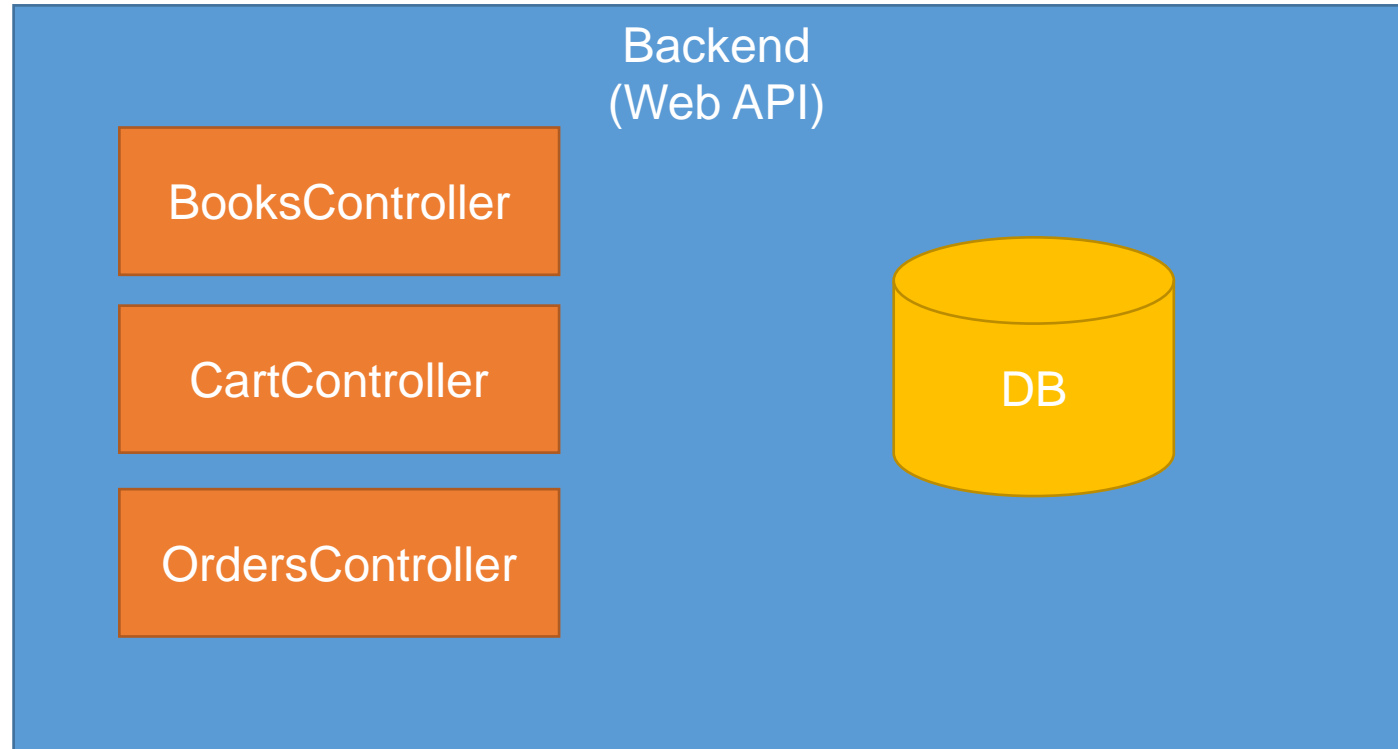


Frontend
(SPA, MVC, WPF,
Mobile)


The diagram consists of two rectangular boxes. The left box is gray and contains the text 'Frontend (SPA, MVC, WPF, Mobile)'. The right box is blue and contains the text 'Backend (Web API)'. There are no lines or arrows connecting the two boxes.

Backend
(Web API)

WAS IST EIN MICROSERVICE?



WAS IST EIN MICROSERVICE?

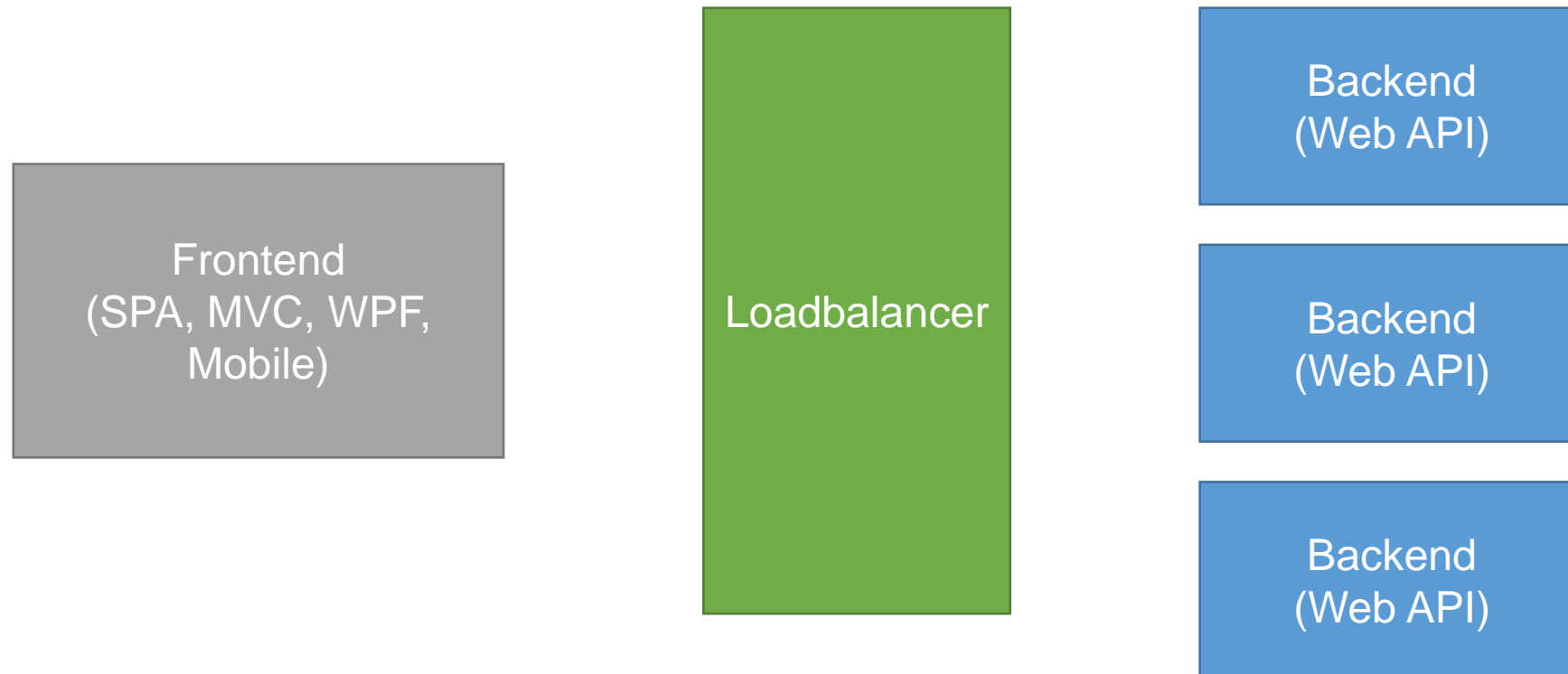


Frontend
(SPA, MVC, WPF,
Mobile)

The diagram consists of two rectangular boxes. The left box is gray and contains the text 'Frontend (SPA, MVC, WPF, Mobile)'. The right box is blue and contains the text 'Backend (Web API)'. There is no visible connection or line between the two boxes.

Backend
(Web API)

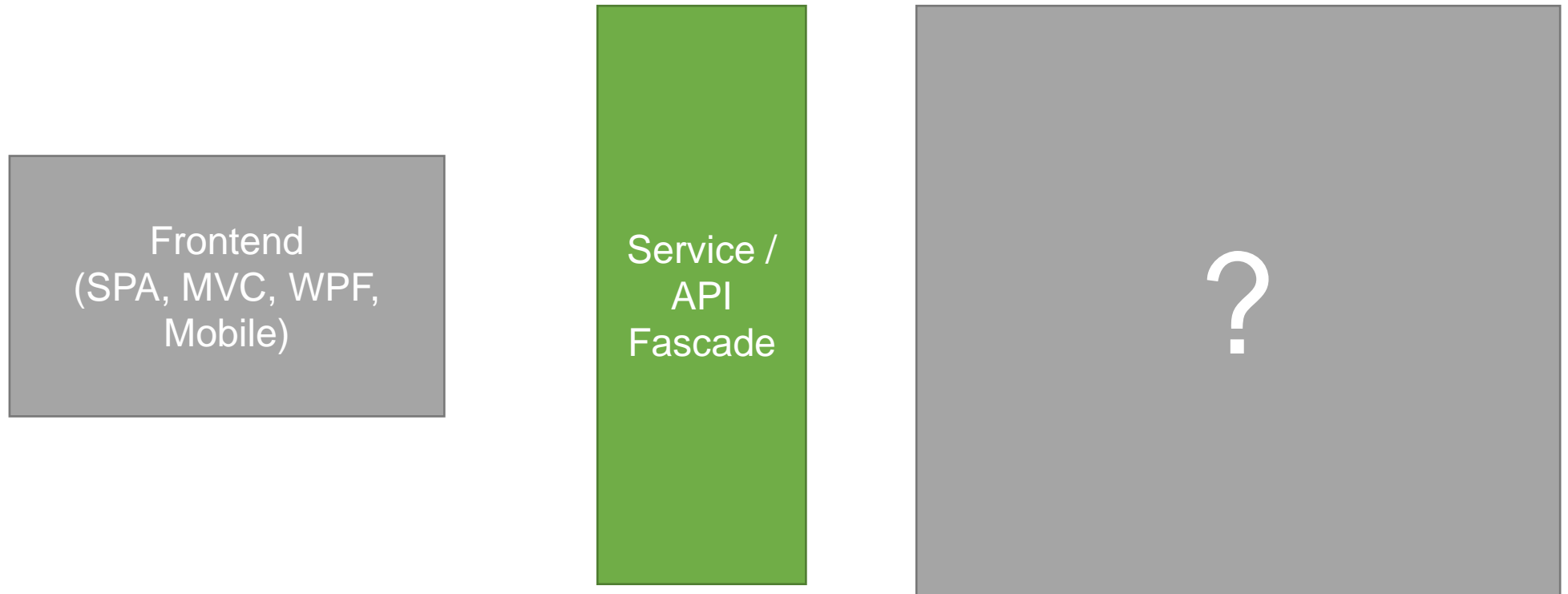
WAS IST EIN MICROSERVICE?



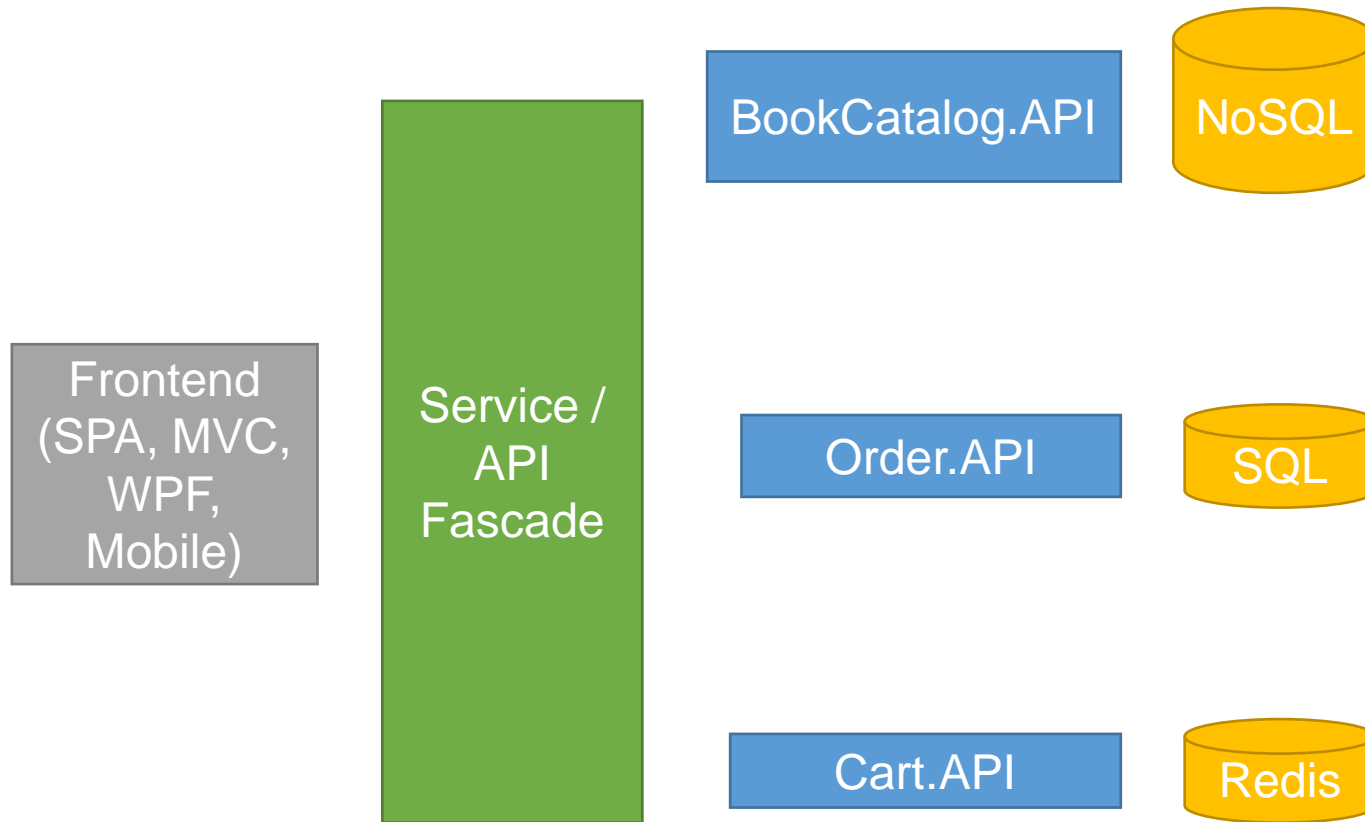
Probleme mit Monolithen

- Können nur als ganzes „skalieren“
- Bei Code-Änderungen muss die komplette Software getestet werden (☺)
- Für eine neue Version muss meist das Produktivsystem angehalten werden
- Oft lange Entwicklungszyklen gerade in größeren Teams
- Müssen sich bei der Technologieauswahl für die Gesamtanwendung festlegen
- Modellierung der Daten oft nicht optimal möglich da Rücksicht auf die zugrundeliegende Speichertechnologie genommen werden muss (NoSQL vs. Relational)

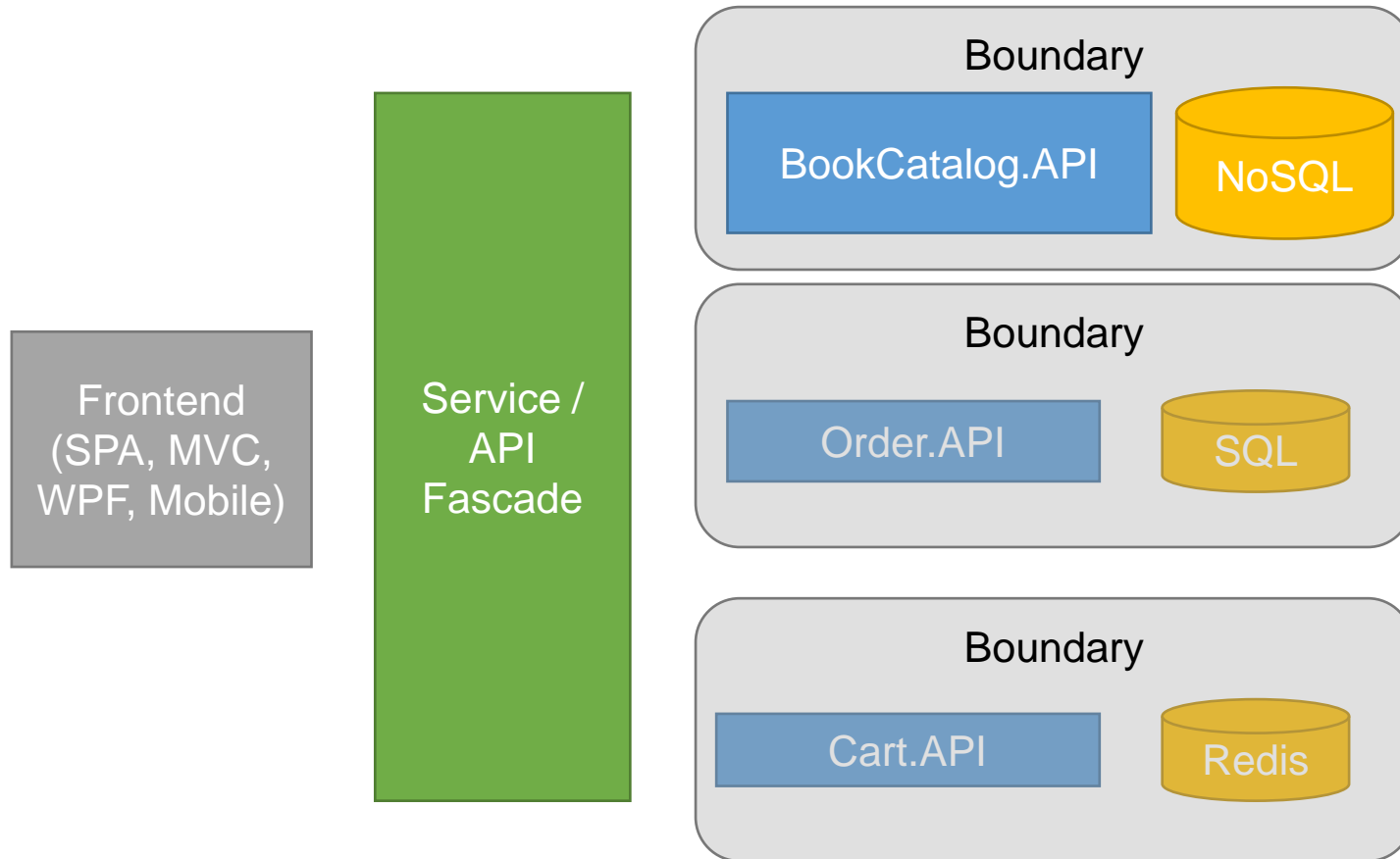
WAS IST EIN MICROSERVICE?



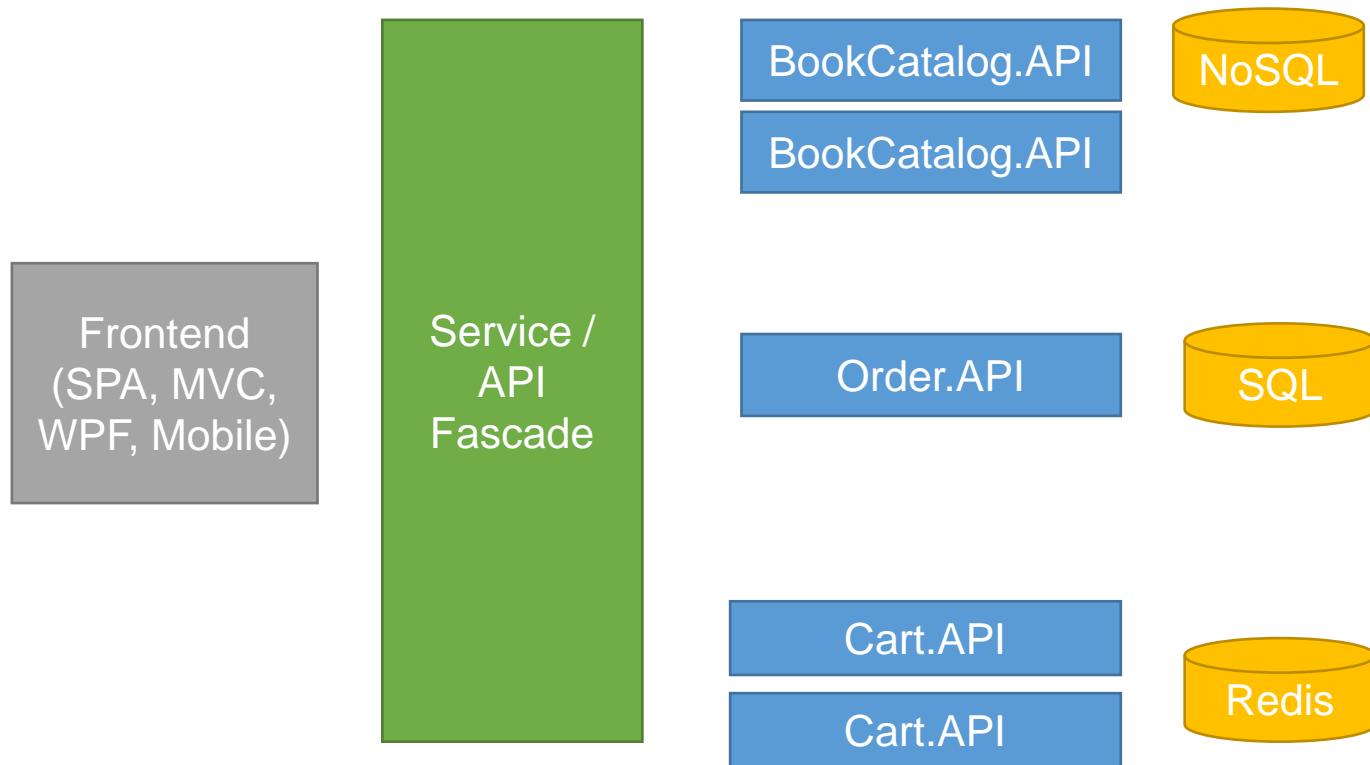
WAS IST EIN MICROSERVICE?



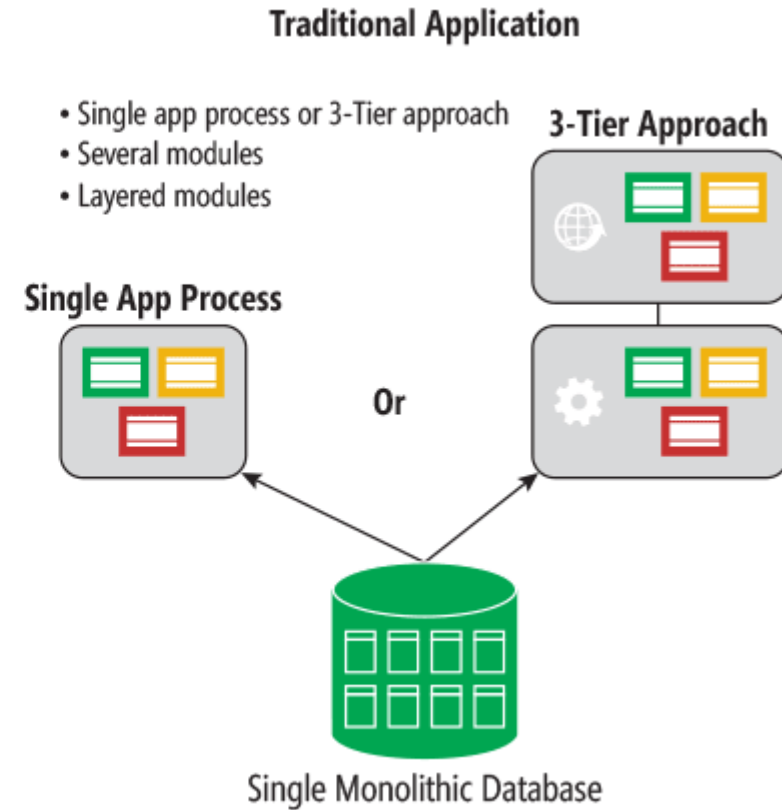
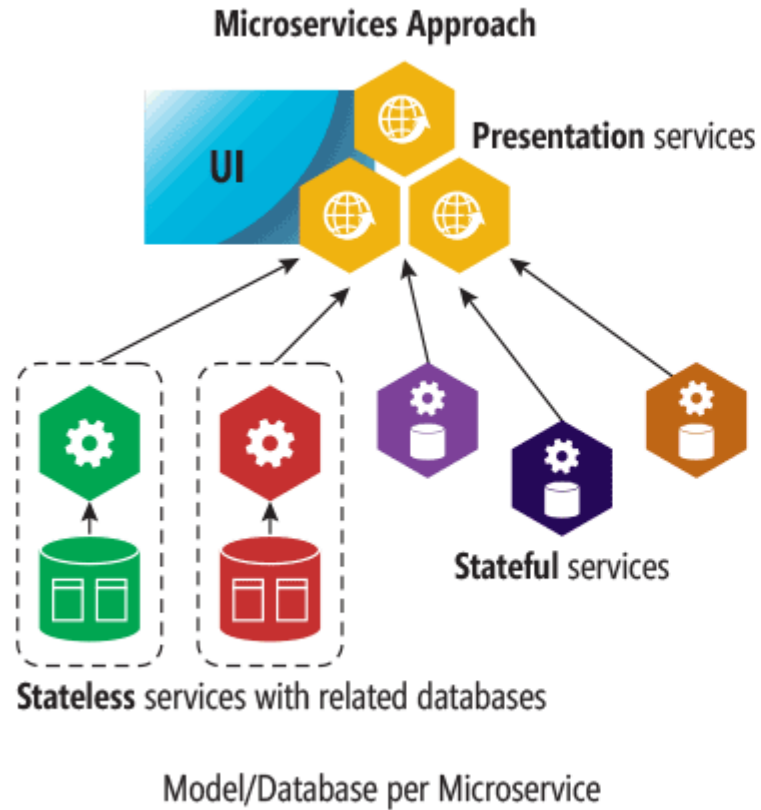
WAS IST EIN MICROSERVICE?



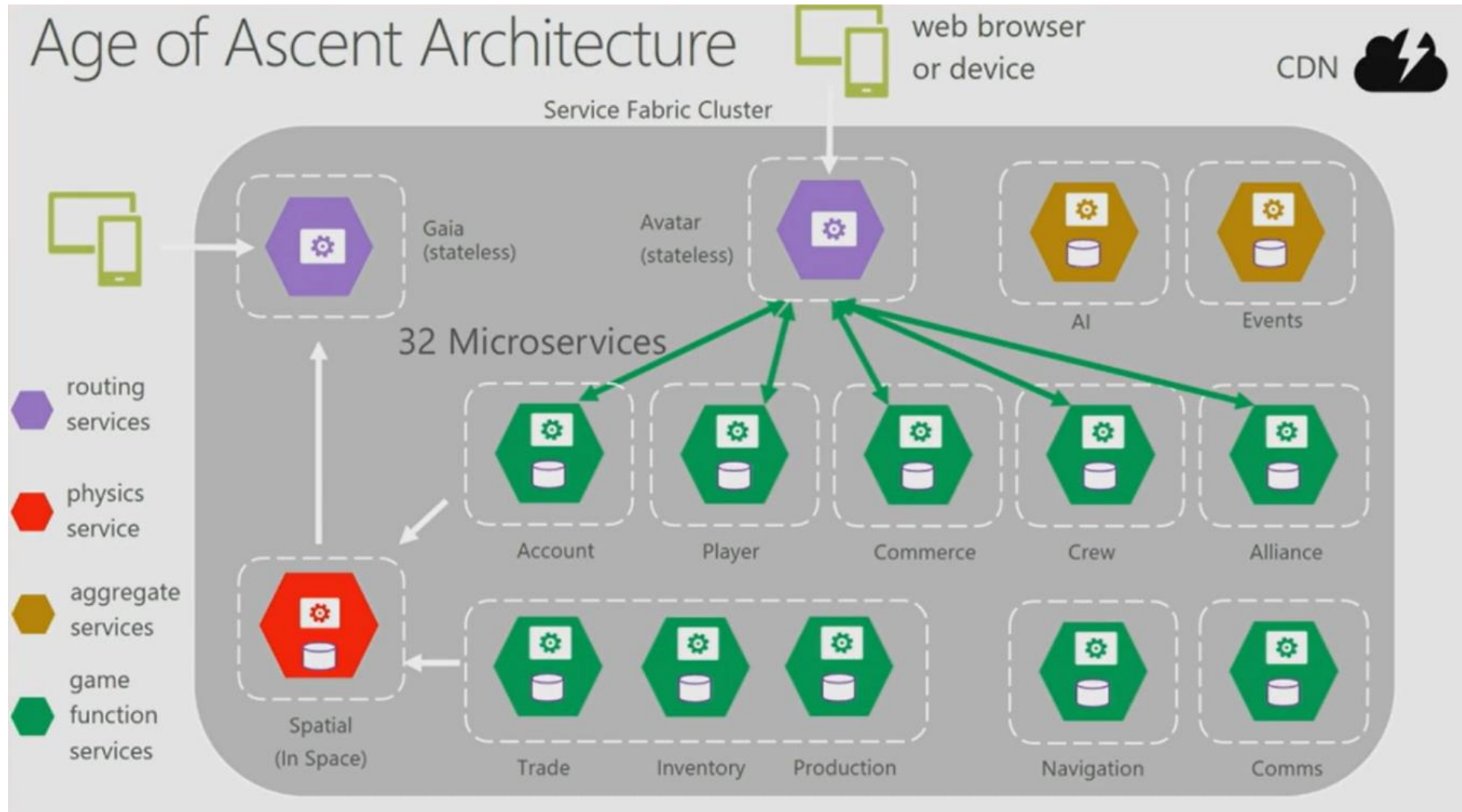
WAS IST EIN MICROSERVICE?



WAS IST EIN MICROSERVICE?



WAS IST EIN MICROSERVICE?



©Microsoft

MICROSERVICE ARCHITECTURE

- Erfüllen exakt eine begrenzte Aufgabe
 - Bestellung platzieren
 - Produktbewertungen
 - Preise für Artikel berechnen
- Läuft unabhängig von anderen Diensten in seinem eigenen Prozess
- Verwaltet seine Daten selbst
- Kann unabhängig von anderen Microservices deployed werden
- Nutzt meist selbst andere Microservices um seine Aufgabe zu erledigen

MICROSERVICE ARCHITECTURE

Microservices are...

Autonomous

- A microservice is a self-contained unit of functionality with loosely coupled dependencies on other services.

Isolated

- A microservice is a unit of deployment that can be modified, tested and deployed as a unit without impacting other areas of a solution

Elastic

- A microservice can be stateful or stateless and can be scaled independently of other services

Resilient

- A microservice is fault tolerant and highly available

Responsive

- A microservice responds to request in a reasonable amount of time

MICROSERVICE ARCHITECTURE

Microservices are... (cont.)

Intelligent

- The intelligence in a system is found in the endpoints not on the wire. ESB is an anti-pattern to Microservices.

Message Oriented

- Microservices rely on asynchronous message-passing to establish a boundary between components

Programmable

- Microservices provide API's for access by developers and administrators and Applications are composed from multiple microservices

Configurable

- Microservices provide an API and/or a console that provides access to administrative operations

Automated

- The lifecycle of a microservice is managed through automation that includes dev, build, test, staging, production and distribution

MICROSERVICE ARCHITECTURE

Benefits of a Microservices Approach

Evolutionary	<ul style="list-style-type: none">• Can be developed alongside existing monolithic applications providing a bridge to a future state
Open	<ul style="list-style-type: none">• Language agnostic APIs• Highly decoupled
Resilient	<ul style="list-style-type: none">• No monolith to fall over• Designed for failure
Speed of Development	<ul style="list-style-type: none">• Adding, updating and maintaining services and can be done at velocity
Reuse	<ul style="list-style-type: none">• Reusable and Composable

MICROSERVICE ARCHITECTURE

Benefits of a Microservices Approach (cont.)

Deployment Governance	<ul style="list-style-type: none">• Services are deployed independently
Scale Governance	<ul style="list-style-type: none">• On-demand scaling of smaller services leads to better cost control
Replaceable	<ul style="list-style-type: none">• Services can be rewritten and replaced with minimal downstream impact
Versioned	<ul style="list-style-type: none">• New API's can be released without impacting clients that are using previous API's
Owned	<ul style="list-style-type: none">• Microservices are typically owned by one team from development through deployment

MICROSERVICE ARCHITECTURE| Herausforderungen

- Kommunikation
 - Zwischen Teams
 - Zwischen Services
 - Message Bus
 - REST
 - Binär
- Verwaltung (Orchestrierung)
 - Health Management
 - Deployment
 - Vertikale Skalierung
- Versionierung
- Performance

CONWAY'S LAW

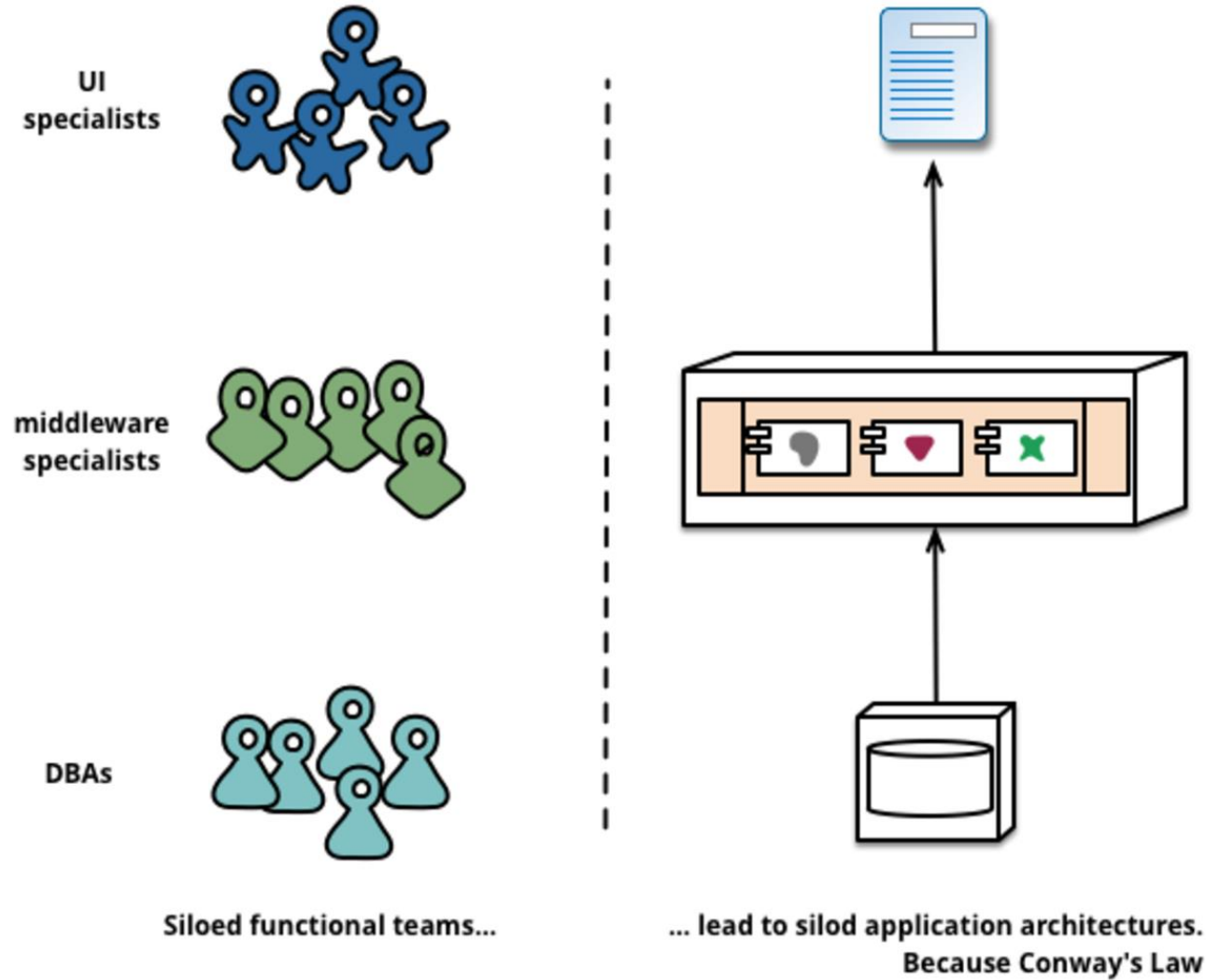
„Any organization that designs a system (defined broadly) will produce a design whose structure is a copy of the organization's communication structure“

CONWAY'S LAW

tightly-coupled organizations => the architecture becomes more **tightly-coupled**

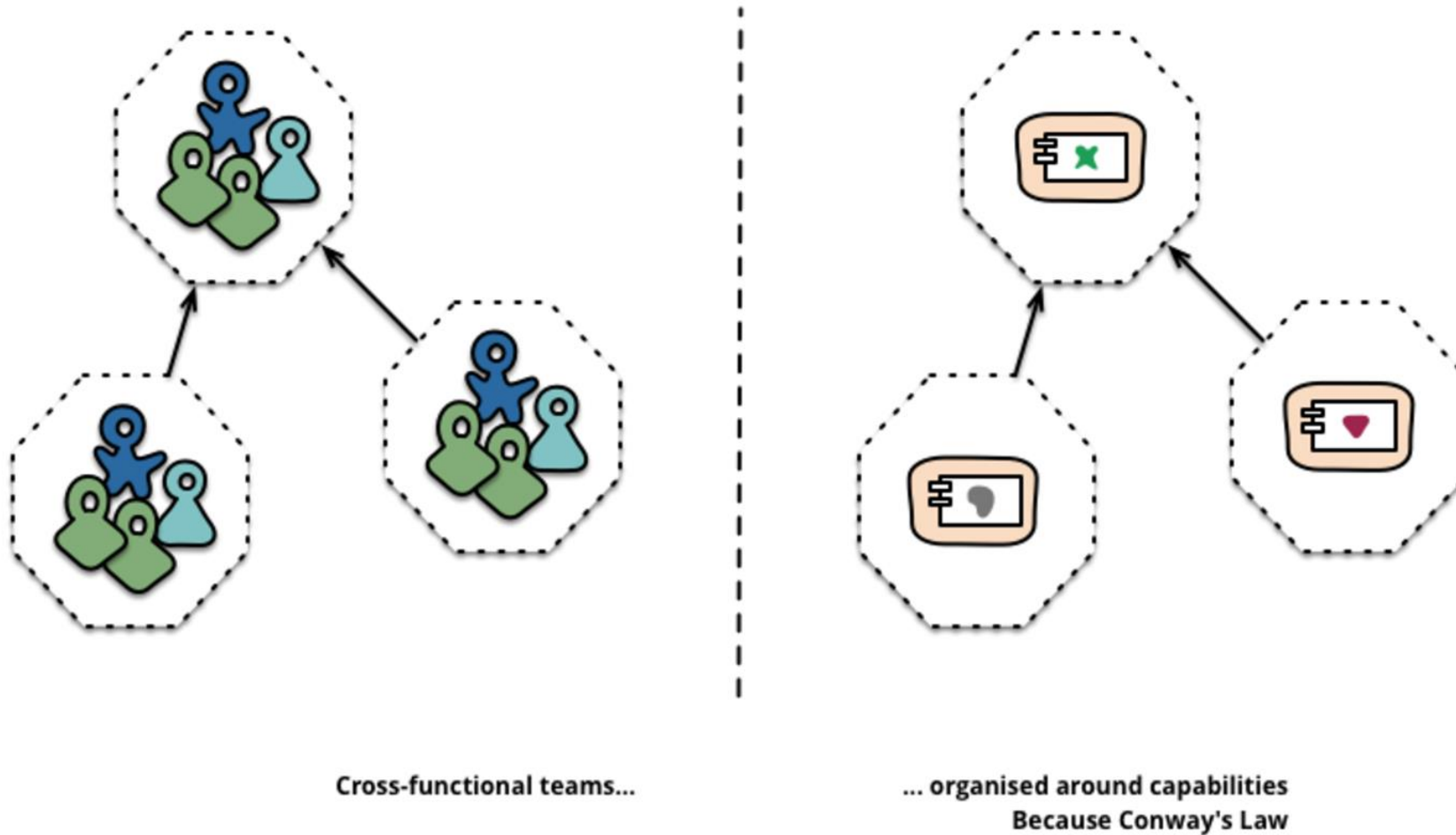
loosely-coupled organizations => the architecture becomes more **modular**

CONWAY'S LAW



©<https://martinfowler.com/articles/microservices.html>

CONWAY'S LAW



©<https://martinfowler.com/articles/microservices.html>

WANN SIND MICROSERVICES DEN AUFWAND WERT?

- It Depends...
- Microservices erlauben schnelle Entwicklungszyklen in autonomen Teams und große Flexibilität bei Deployment und Skalierung im Austausch für größeren Aufwand bei der Architektur, Schnittstelle und Kommunikation zwischen den Services
- Microservices sind keine „Silver Bullet“, ein sauber strukturierter Monolith kann oft genauso die richtige Architektur sein („Your not Netflix“)

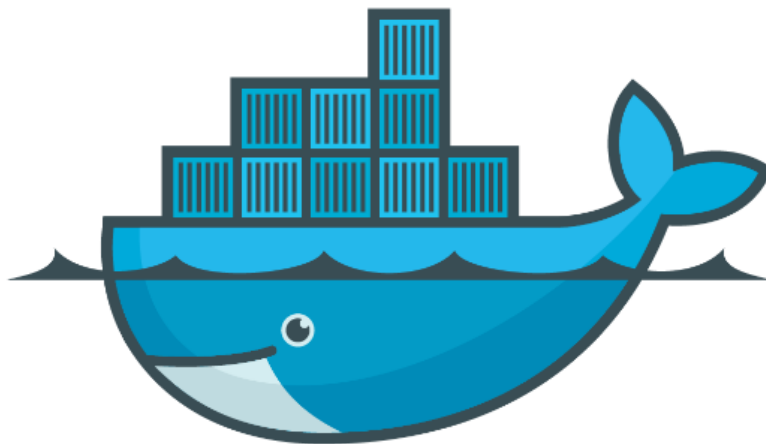
IMPLEMENTIERUNGSMÖGLICHKEITEN MIT .NET

- ASP.NET Core MVC
- ASP.NET Core Middleware
- ASP.NET Core Endpoint Routing
- Carter







WAS IST BEI DER ENTWICKLUNG ZU BEACHTEN?








- Müssen separat deploybar sein
- Keine Abhängigkeiten zu anderen Service-Projekten
 - „Building Blocks“ werden „geduldet“
- Fehlertoleranz ist extrem wichtig
 - Aufrufe können / werden fehlschlagen
- Performance as a Feature
 - Latenzzeiten
 - Caching (und Cache-Invalidation)
- Nachvollziehbarkeit
 - Logging mit Correlation-ID um einen Request verfolgen zu können

DOCKER



DOCKER

	Static website	?	?	?	?	?	?	?
	Web frontend	?	?	?	?	?	?	?
	Background workers	?	?	?	?	?	?	?
	User DB	?	?	?	?	?	?	?
	Analytics DB	?	?	?	?	?	?	?
	Queue	?	?	?	?	?	?	?
		Dev VM	QA Server	Single Prod Server	Onsite Cluster	Azure	Contributor laptop	Customer Servers

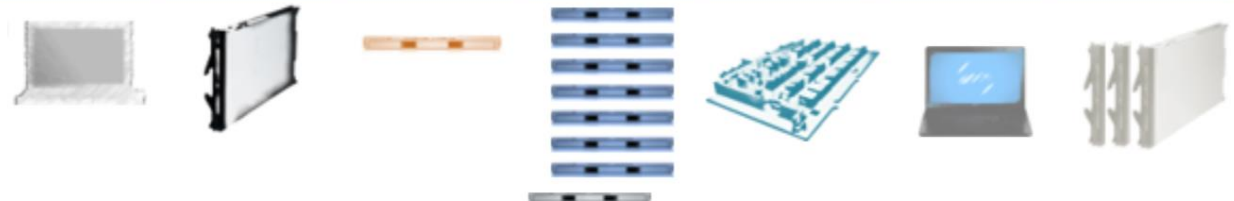








©Microsoft

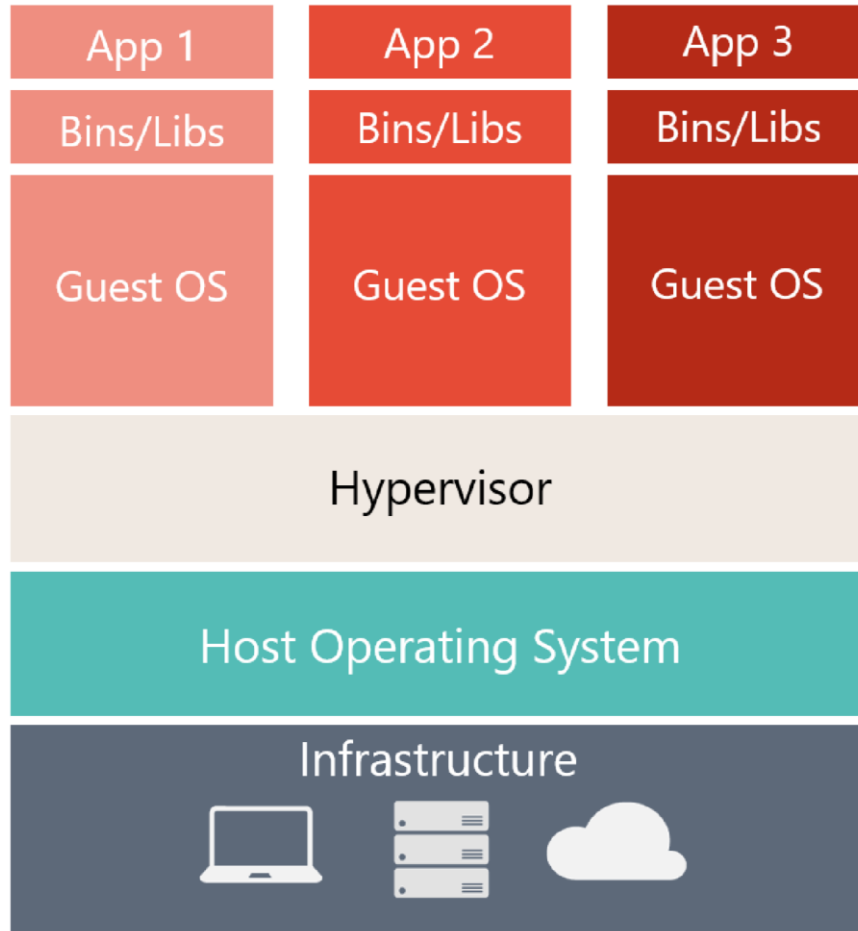
DOCKER

	Static website							
	Web frontend							
	Background workers							
	User DB							
	Analytics DB							
	Queue							
		Dev VM	QA Server	Single Prod Server	Onsite Cluster	Azure	Contributor laptop	Customer Servers

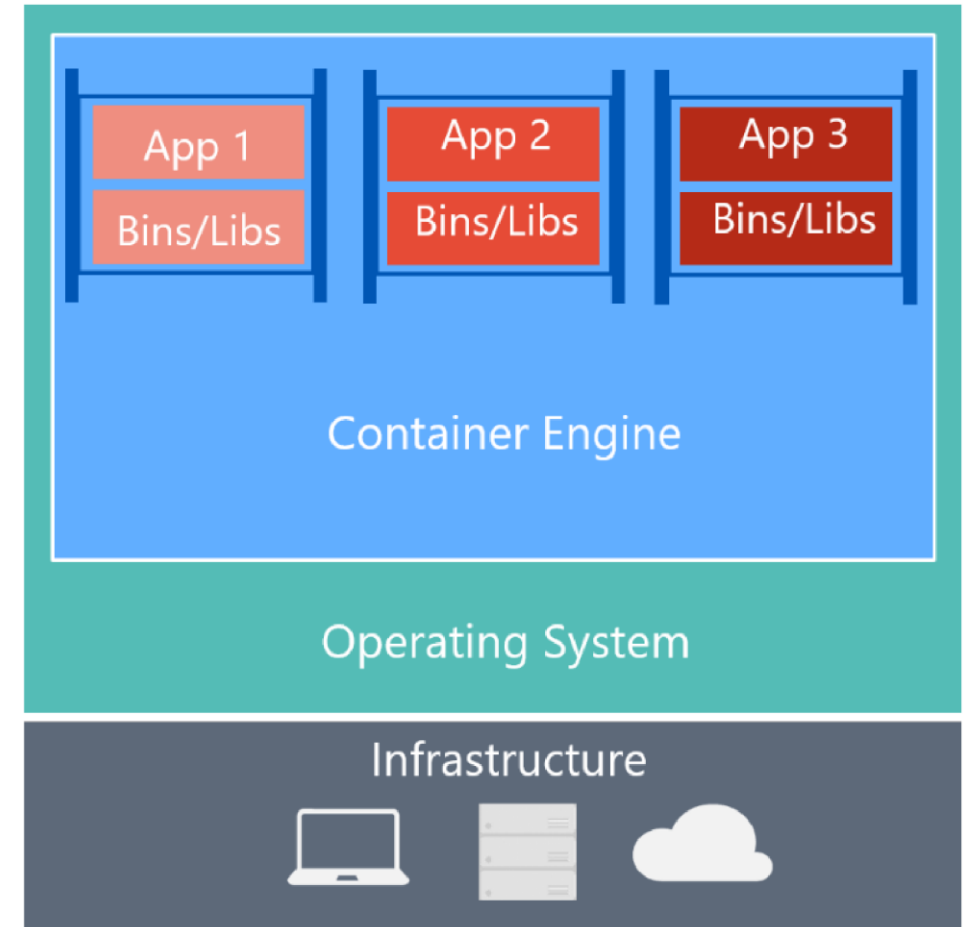
©Microsoft



DOCKER



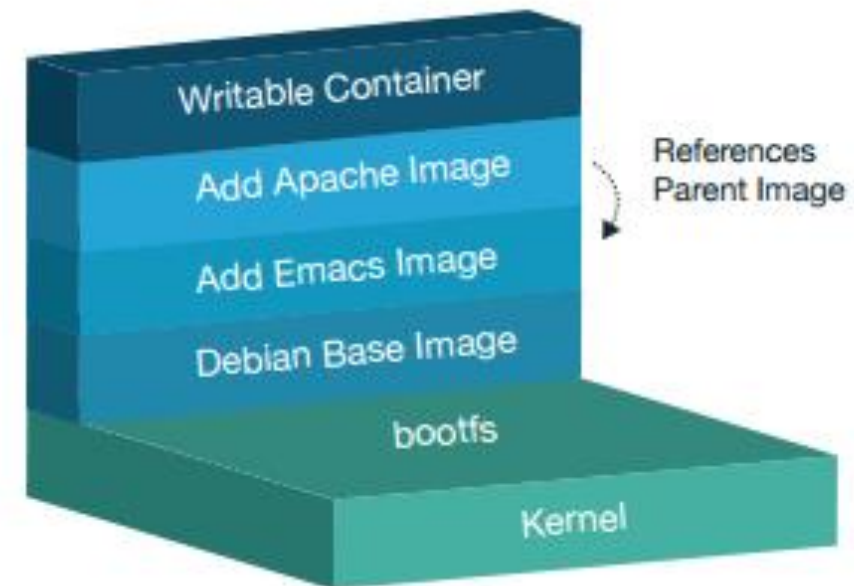
©Microsoft



©Microsoft

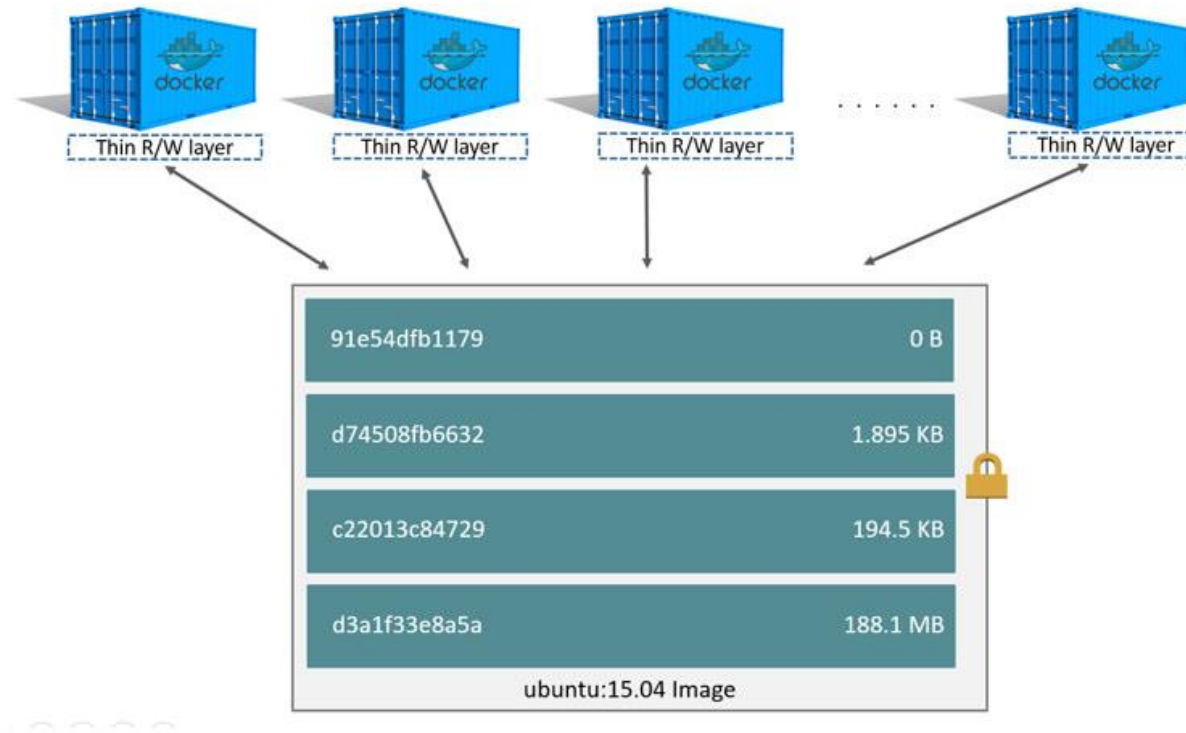
DOCKER|Container-Image

- Paket das alle benötigten Abhängigkeiten wie Programme und Frameworks enthält um einen Container zu erzeugen (Template).
- Images sind unveränderlich und basiert auf (mehreren) Basisimages (Layering)



DOCKER| Container

- Eine Instanz eines Containerimages und stellt die Laufzeitumgebung für die Applikation(en) bereit



©Docker

DOCKER| Volume

- Images sind Read-Only
- Container enthalten ein Filesystem das aber nur für unwichtige Daten benutzt werden darf
- Volumes sind Ordner / Dateien außerhalb eines Containers auf dem Host-System
- Volumes können beim Starten eines Containers mit diesem verbunden werden

DOCKER| Tag

- Images können mit einer Bezeichnung (Tag) gekennzeichnet werden, z.B. für Versionierung. Images erhalten eine GUID die sie eindeutig kennzeichnet, ein Tag hilft bei der Identifizierung / Suche des Images

DOCKER| Dockerfile

- Eine Textdatei welche Instruktionen darüber enthält, wie ein Image zusammengesetzt ist
- **Automatisiert die Erstellung von Images**

```
FROM mcr.microsoft.com/dotnet/core/aspnet:3.1-buster-slim AS base
WORKDIR /app
EXPOSE 80
EXPOSE 443
```

```
FROM mcr.microsoft.com/dotnet/core/sdk:3.1-buster AS build
WORKDIR /src
COPY ["MyMicroservice/MyMicroservice.csproj", "MyMicroservice/"]
RUN dotnet restore "MyMicroservice/MyMicroservice.csproj"
COPY . .
WORKDIR "/src/MyMicroservice"
RUN dotnet build "MyMicroservice.csproj" -c Release -o /app/build
```

```
FROM build AS publish
RUN dotnet publish "MyMicroservice.csproj" -c Release -o /app/publish
```

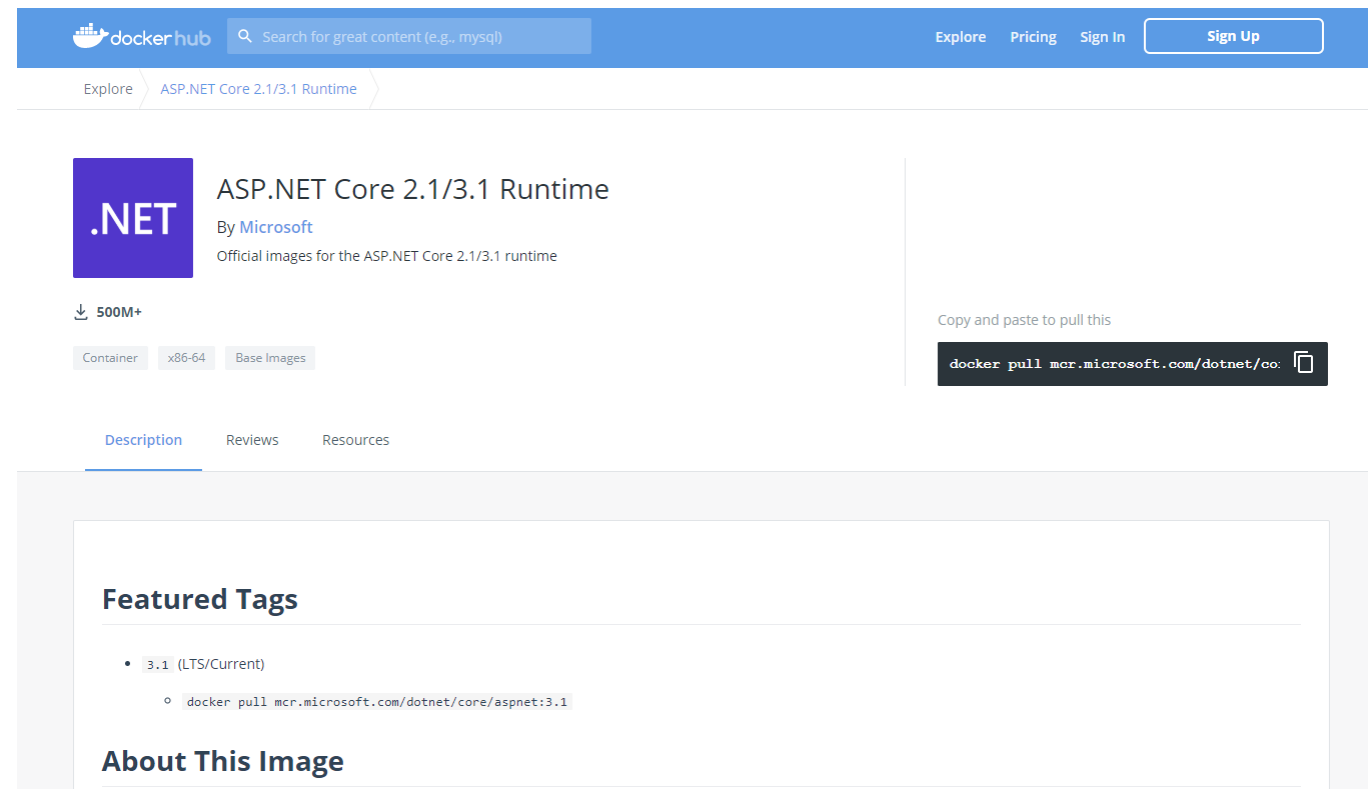
```
FROM base AS final
WORKDIR /app
COPY --from=publish /app/publish .
ENTRYPOINT ["dotnet", "MyMicroservice.dll"]
```

DOCKER| Build

- Images müssen vor ihrer Verwendung den build-Prozess durchlaufen
- Docker stellt dazu den Befehl `docker build` bereit

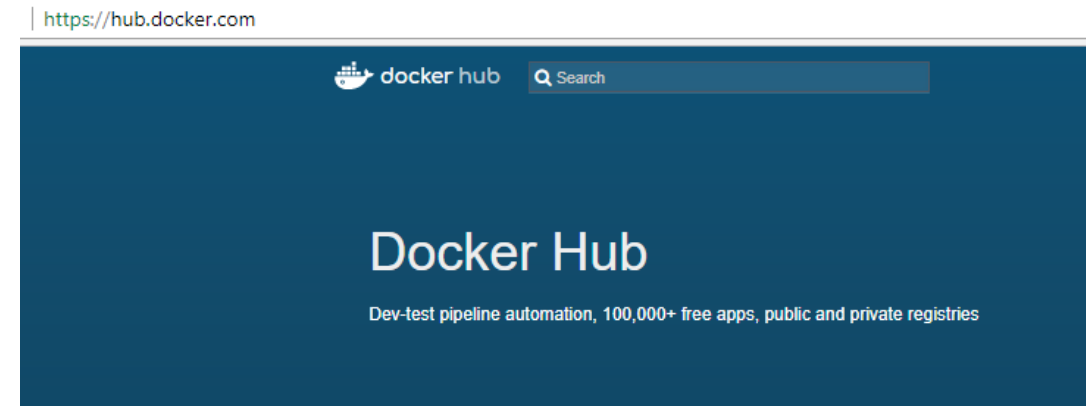
DOCKER| Repository

- Sammlung von Dockerimages mit Tags welche Auskunft über die Version geben
- Enthalten oft auch verschiedene Ausprägungen wie z.B. Linux / Windows basiert, oder stable / beta



DOCKER| Registry

- Registry
 - Ein Dienst der Zugriff auf Repositories bietet und Funktionalität, um diese zu verwalten
- Docker Hub
- Docker Trusted Registry
- Azure Container Registry



DOCKER| Compose

- Docker compose ist ein Tool welches eine YAML-Datei einliest, welche Metadaten für die Ausführung von Multi-Container-Applikationen enthält
- Mit docker compose up können die Container gestartet werden
- Compose baut die Container falls dies noch nicht geschehen ist
- **Automatisiert den Prozess der (Multi-)Containererzeugung**

```
version: '2'

services:
  bookcatalog.api:
    image: bookcatalog.api
    build:
      context: ./BookCatalog.API
      dockerfile: Dockerfile

  recommendations.api:
    image: recommendations.api
    build:
      context: ./Recommendations.API
      dockerfile: Dockerfile
    depends_on:
      - sql.data

  sql.data:
    image: microsoft/mssql-server-linux

  nosql.data:
    image: mongo
```

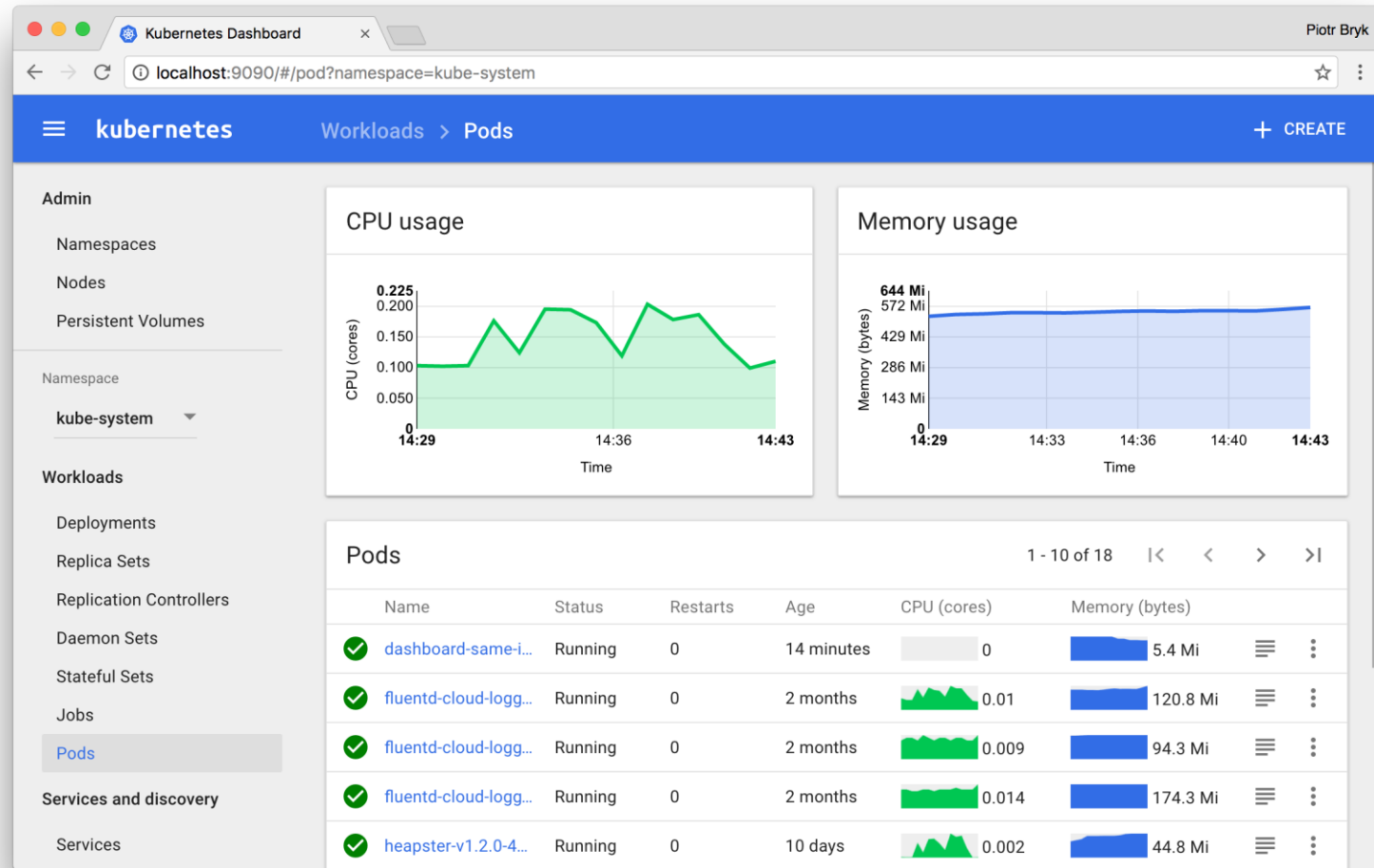

DOCKER TERMS| Cluster

- Cluster (Swarm)
 - Mehrere Docker Host werden zu einem logischen Host zusammengefasst
 - Innerhalb des Clusters können Docker Container über mehrere Hosts skaliert werden
 - Spezielle Managementsoftware steht zur Erzeugung des Clusters bereit
 - Docker Swarm
 - Mesosphere DC/OS
 - Kubernetes
 - Azure Service Fabric

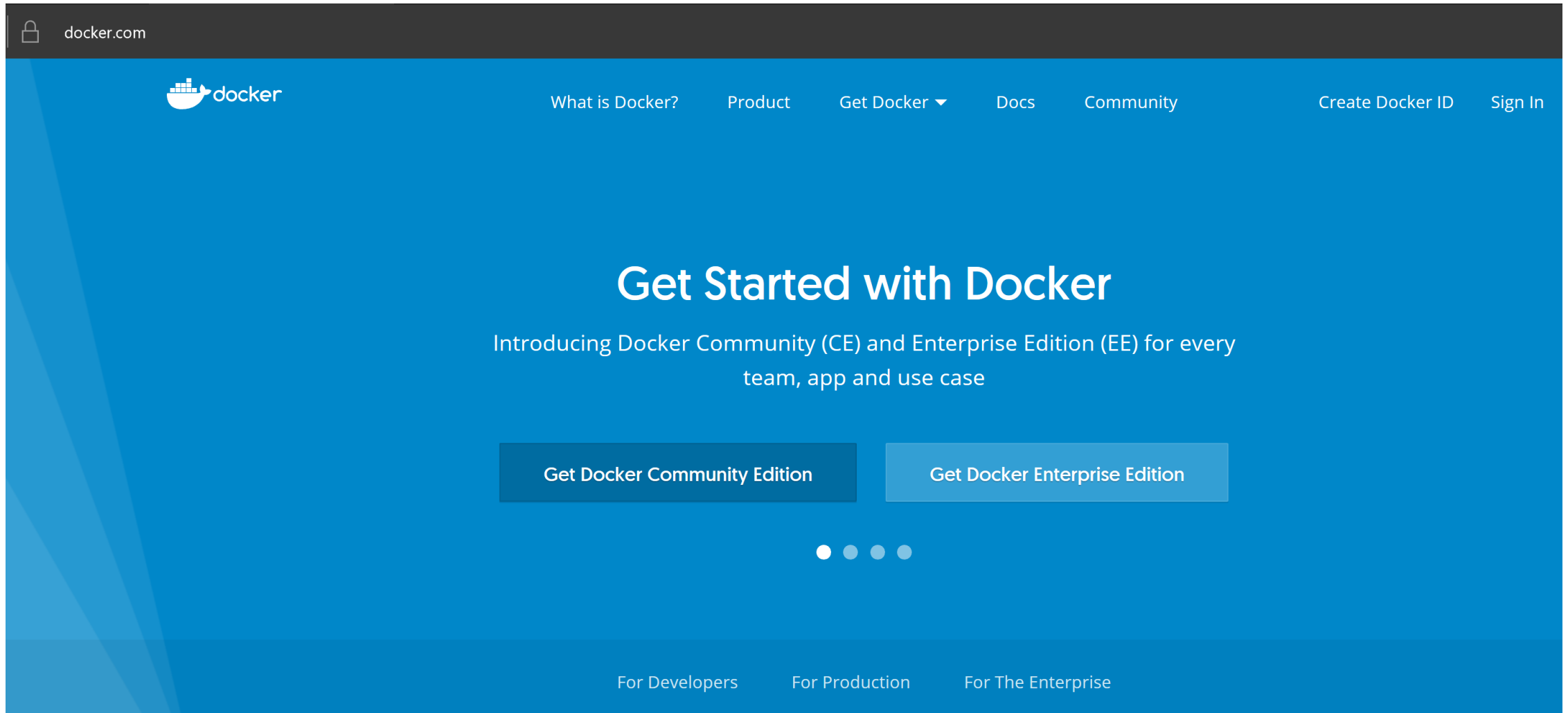
DOCKER| Orchestrator

- Software zur Verwaltung des Clusters
- Container / Image Management
- Health Checks
- Load Balancing
- Service Discovery
- Spezielle Managementsoftware steht zur Verwaltung des Clusters bereit
 - Docker Swarm
 - Mesosphere DC/OS
 - **Kubernetes**
 - Azure Service Fabric

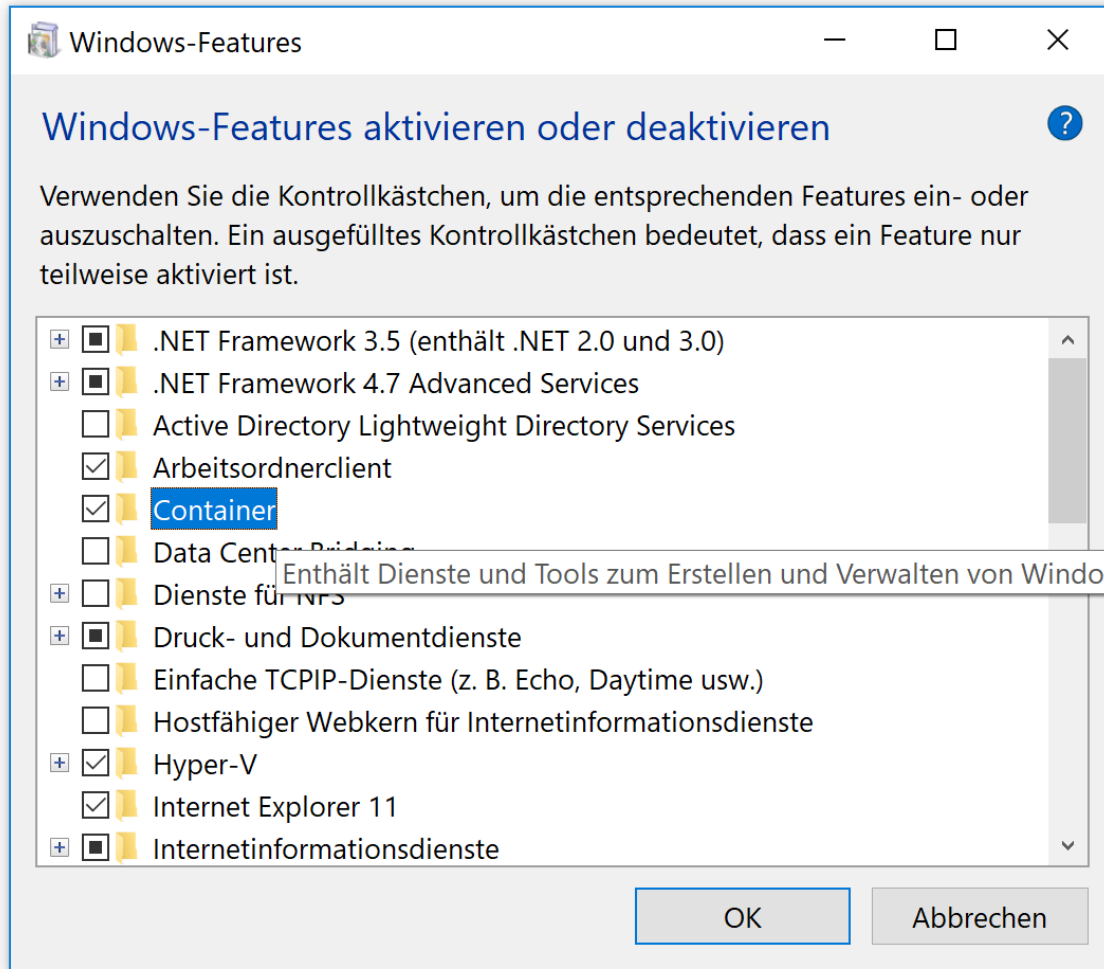
DOCKER| Orchestrator



DOCKER FOR WINDOWS



DOCKER FOR WINDOWS



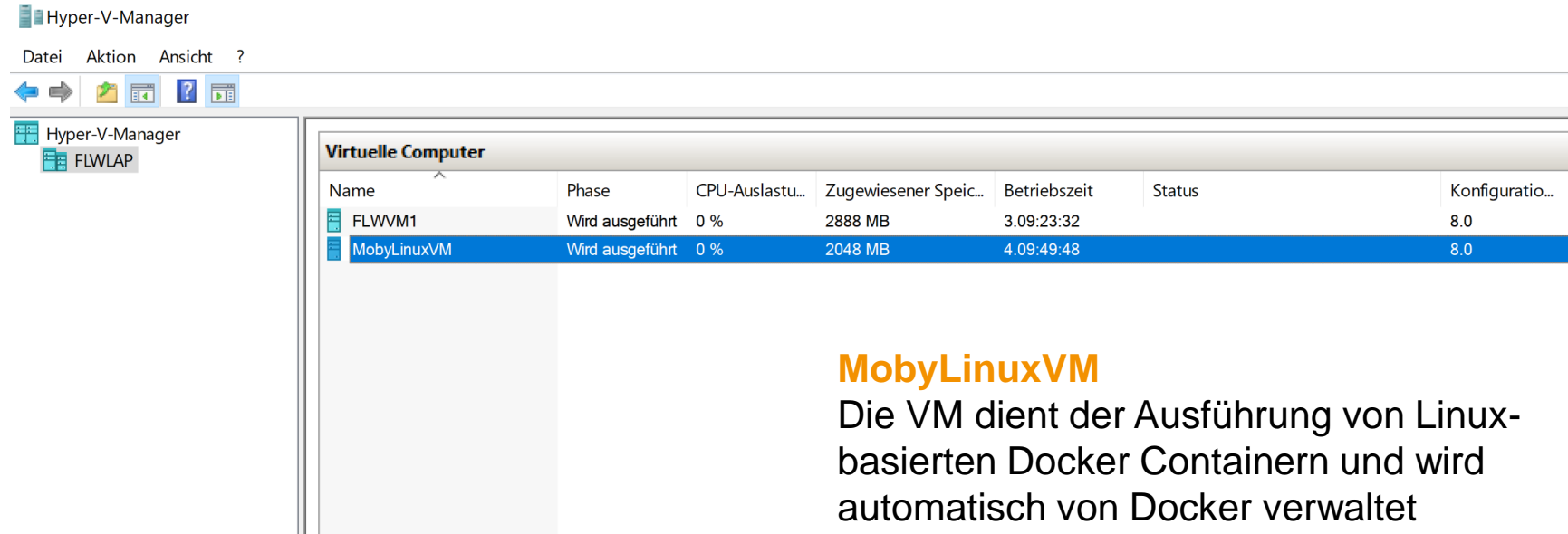
Native Windows-Container in Win 10+ / Server 2016

Für den Einsatz von Windows-basierten Containern muss das entsprechende Windows-Feature aktiviert werden.

Hyper-V für Linux Container (WSL 1)

Hyper-V wird für Linux-Container und eine Spezialvariante von Windows-Container (Isolation) verwendet

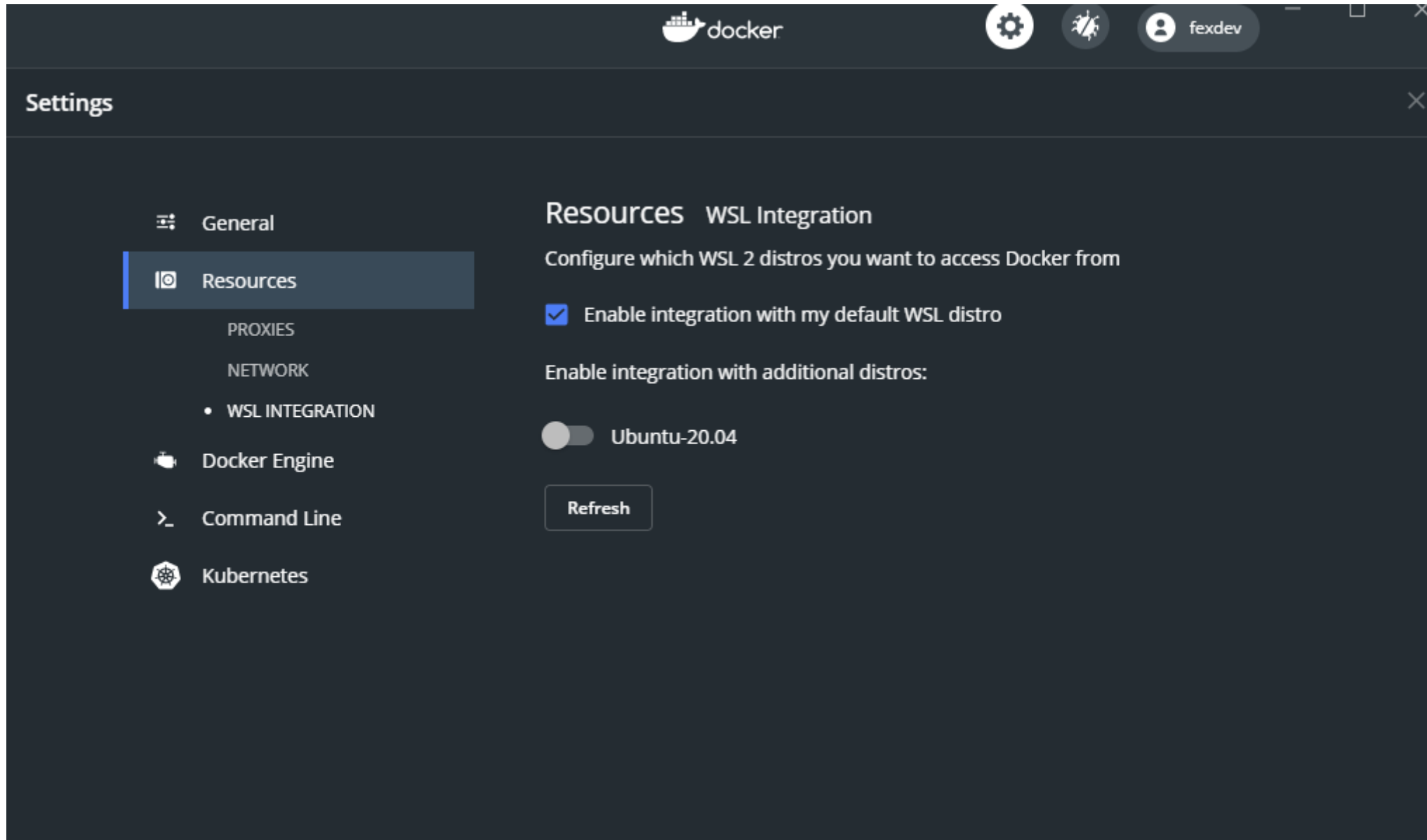
DOCKER FOR WINDOWS (WSL 1)



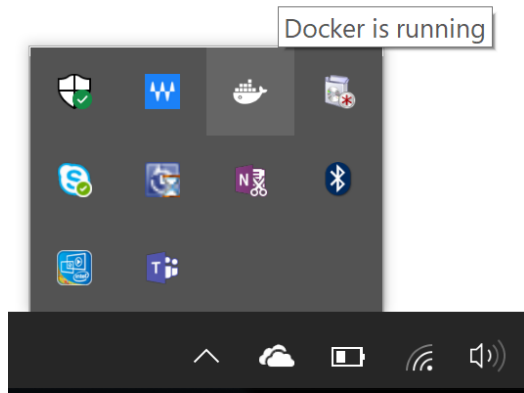
MobyLinuxVM

Die VM dient der Ausführung von Linux-basierten Docker Containern und wird automatisch von Docker verwaltet

DOCKER FOR WINDOWS (WSL 2)

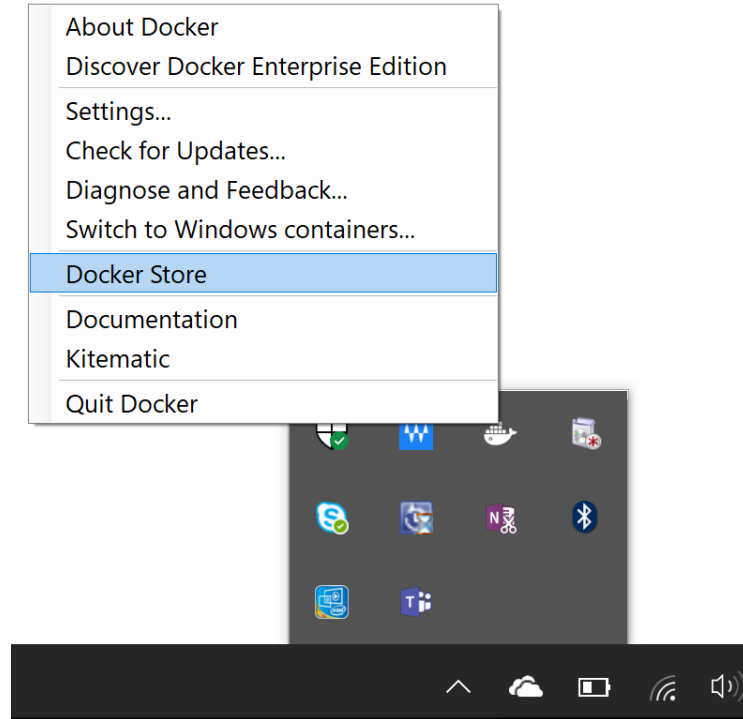


DOCKER FOR WINDOWS



Tray-Icon

Nach der Installation ist Docker über die Taskleiste verfügbar



DOCKER FOR WINDOWS

```
PowerShell
logout      Log out from a Docker registry
logs        Fetch the logs of a container
pause       Pause all processes within one or more containers
port        List port mappings or a specific mapping for the container
ps          List containers
pull        Pull an image or a repository from a registry
push        Push an image or a repository to a registry
rename      Rename a container
restart     Restart one or more containers
rm          Remove one or more containers
rmi         Remove one or more images
run         Run a command in a new container
save        Save one or more images to a tar archive (streamed to STDOUT by default)
search      Search the Docker Hub for images
start       Start one or more stopped containers
stats       Display a live stream of container(s) resource usage statistics
stop        Stop one or more running containers
tag         Create a tag TARGET_IMAGE that refers to SOURCE_IMAGE
top         Display the running processes of a container
unpause     Unpause all processes within one or more containers
update      Update configuration of one or more containers
version     Show the Docker version information
wait        Block until one or more containers stop, then print their exit codes

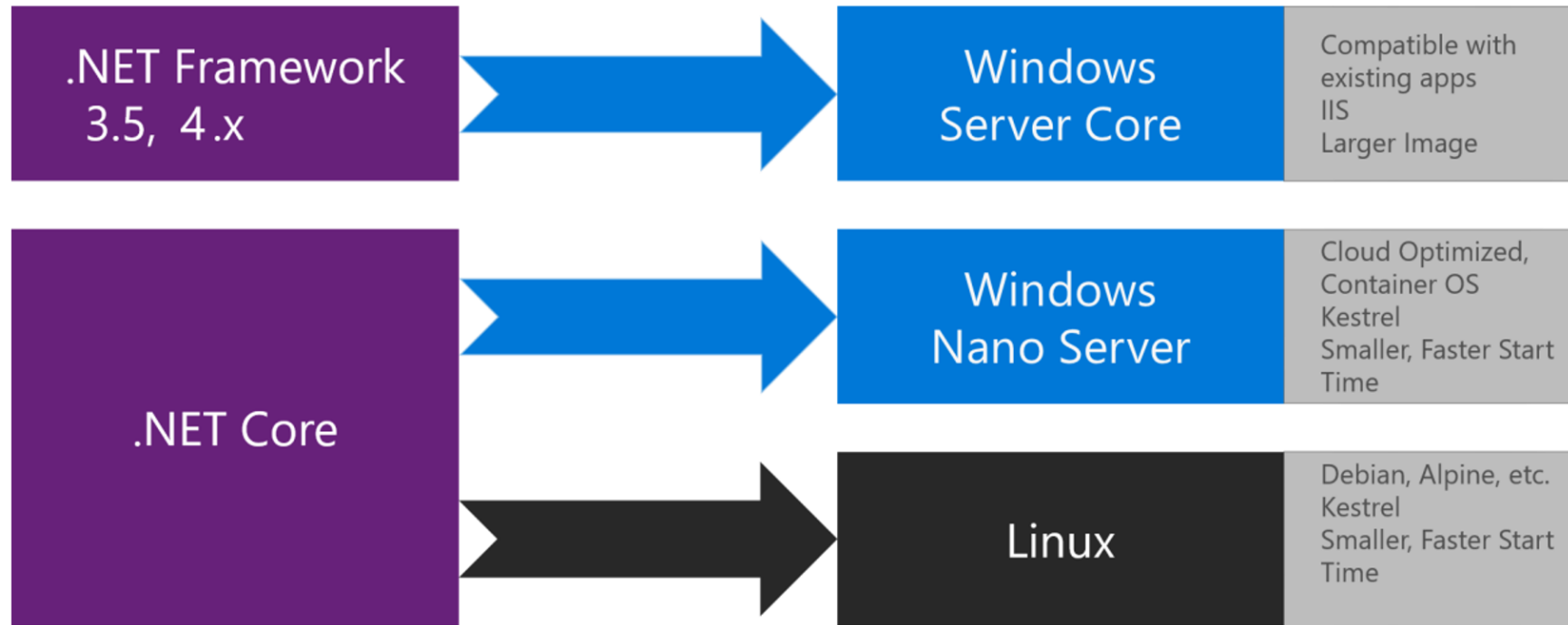
Run 'docker COMMAND --help' for more information on a command.
Florian.Wachs@CLIENT650100 ~
> |
```

Docker CLI

Docker for Windows installiert Management Tools wie docker und docker-compose für die Verwaltung von Docker Containern

.NET CONTAINER

What OS to target with .NET containers



©Microsoft

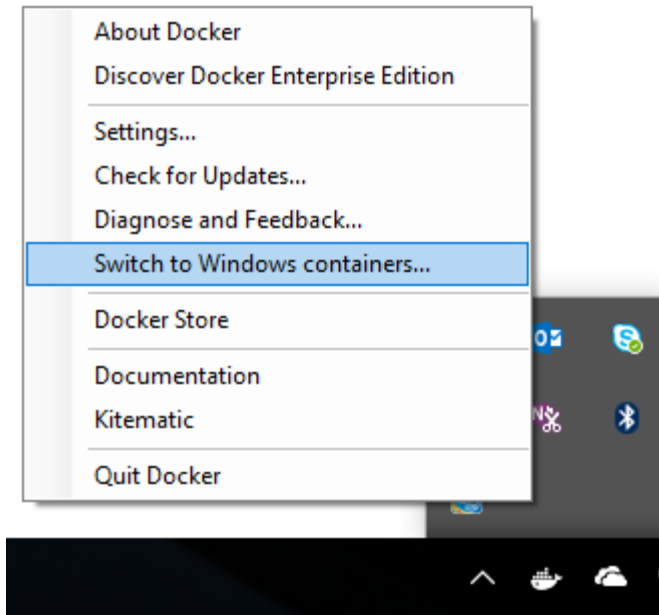
VORTEILE VON WINDOWS CONTAINERN

- Laufen nativ auf Windows Server 2016 oder in einer speziellen Hyper-V Umgebung (keine „echte“ VM) und damit auf Windows Systemen performanter
- Zukünftige Container Features von Windows
- Ausführung von .NET Full Framework Anwendungen
- Migration von Legacy-Systemen
- Tool zum Erstellen von Containern aus bestehenden Systemen in Entwicklung

NACHTEILE VON WINDOWS CONTAINERN

- Deutlich größere Imagegröße
- Immer noch nicht vollständig mit Docker vor Windows integriert (z.B. Netzwerk)
- Weniger Optionen bei Orchestratoren und noch wenig ausgereift

WINDOWS CONTAINER



Wechsel zwischen Container Typen

Aktuell muss Docker mitgeteilt werden ob Linux oder Windows Container verwendet werden sollen. Bekommt man beim Installieren oder Starten von Containern „merkwürdige“ Fehlermeldungen, sollte man diese Einstellungen prüfen

WELCHES CONTAINER OS?

- Windows Container wenn,
 - Legacy Applikationen in eine Container Infrastruktur integriert werden sollen
 - .NET Apis benötigt werden die in .NET Core (noch) nicht vorhanden sind oder nie implementiert werden (WCF, Workflows)
 - Frameworks / Libraries die nicht für .NET Core verfügbar sind
- Linux Container für,
 - Cross-Platform Entwicklungen mit .NET Core
 - Maximale Performance
 - Effizienz bei Imagegröße und Ressourcenverbrauch
 - Microservices Architekturen (Asp.Net Core / Middleware)
 - Mehr Entwickleroptionen da Linux Container auf Linux, MacOS und Windows entwickelt werden können

VORTEILE VON CONTAINERN

- Benötigen weniger Ressourcen als eine VM
- Keine „Works on MyMachine“-Probleme mehr*
- Starten und Beenden sehr schnell (scale-out)
- „Higher Density“ -> viele Container auf der gleichen Maschine
- Klare Definition von Abhängigkeiten
- Keine unterschiedlichen Patchlevel (Image wird immer als ganzes betrachtet)

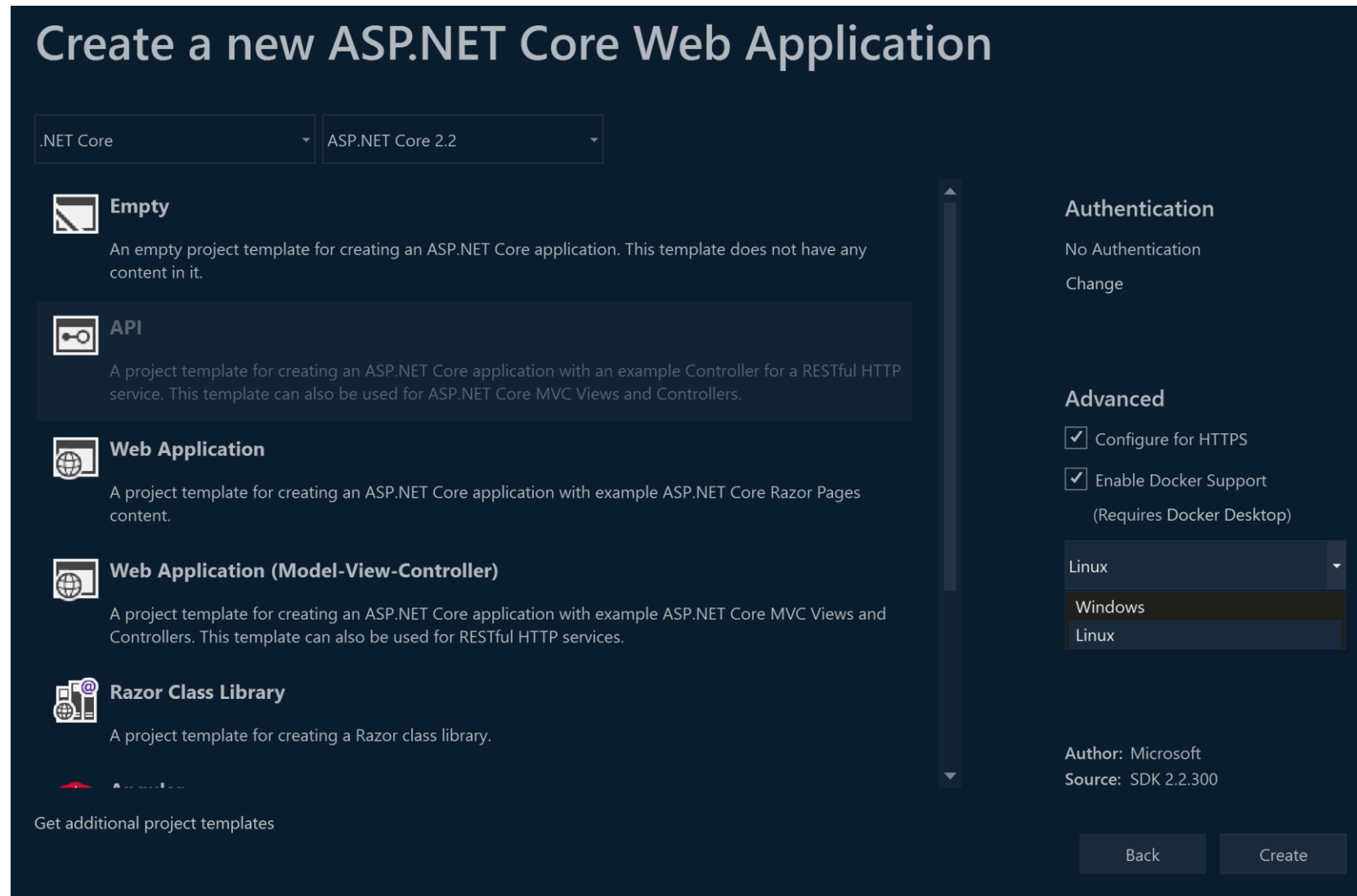
NACHTEILE VON CONTAINERN

- Theoretisch nicht so sicher wie eine VM
 - Root, Kernel-Sharing
- Man benötigt eine Registry für die Images
- Mehr „bewegliche Teile“ (Container, Registry, Orchestrator)

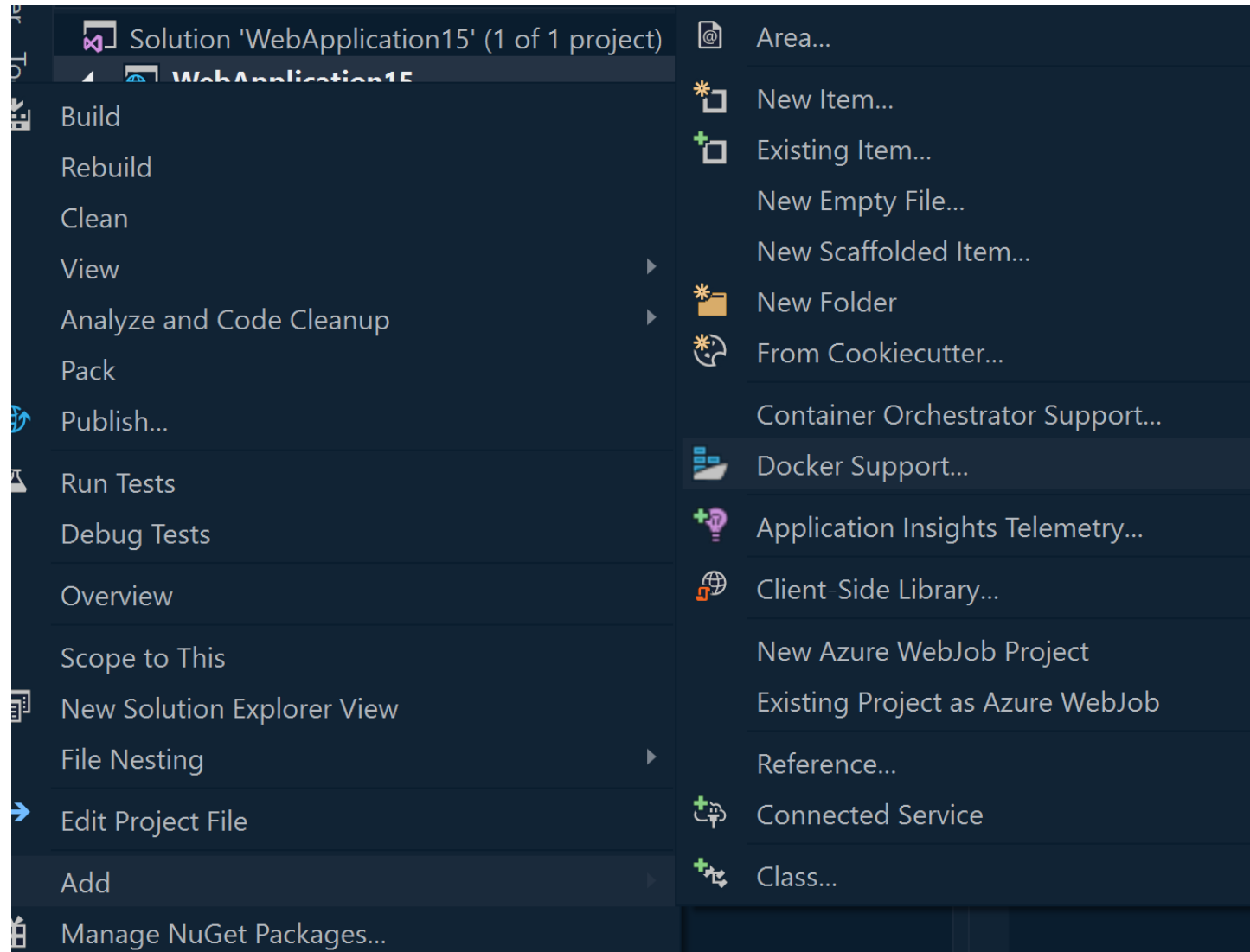
Ausblick Docker auf Windows

- In Windows existiert aktuell die Windows Subsystem for Linux (WSL), welche eine Linuxumgebung simuliert. Es werden OS-Aufrufe nach Windows übersetzt, Anwendungen wie Docker können damit aber nicht ausgeführt werden.
- In WSL 2 (aktuell in Preview 2019) wurde die Emulation durch eine virtualisierte Implementierung eines vollständigen Linux-Kernels ersetzt. Damit ist es möglich auch Docker auszuführen. Docker selbst hat bereits mit der Implementierung für WSL 2 begonnen (<https://engineering.docker.com/2019/06/docker-hearts-wsl-2/>)

Docker in Visual Studio



Docker in Visual Studio



DEMO

- Docker for Windows
- Visual Studio Docker Tools