



# Docker für ASP.NET Core Anwendungen: Best Practices von der Entwicklung bis zur Produktion

Marc Müller  
Principal Consultant



marc.mueller@4tecture.ch  
@muellermarc  
www.4tecture.ch

4tecture<sup>®</sup>  
empower your software solutions

About me:

Marc Müller  
Principal Consultant  
@muellermarc



4tecture<sup>©</sup>  
empower your software solutions

Our Products:

Multi-Tenant OpenID  
Connect Identity Provider

Enterprise Application  
Framework for .NET



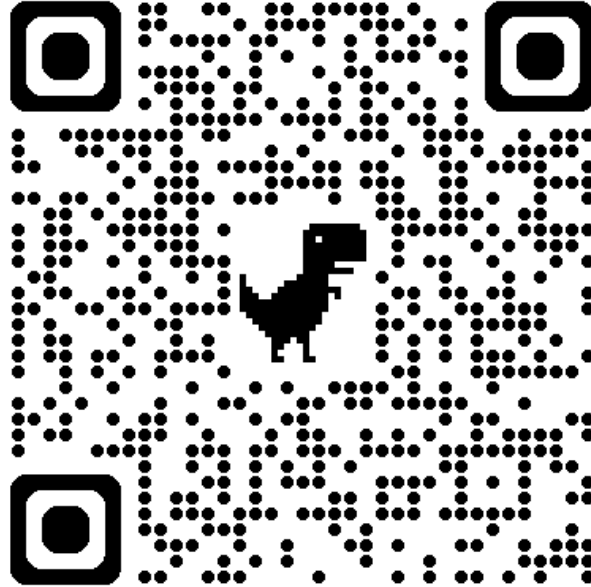
ProAuth

[www.proauth.net](http://www.proauth.net)



[www.reafx.net](http://www.reafx.net)

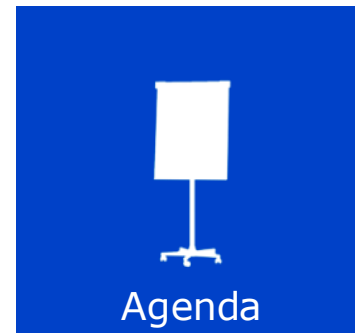
# Slide Download



<https://www.4tecture.ch/events/basta25springdockeraspnetcore>

# Agenda

- Intro
- Building Containers
- IDE Integration & Multi-Service Development
- Q&A



A background image showing a rowing team in a boat. The rowers are wearing blue and red uniforms. The focus is on the oars and the rowing mechanism, which are yellow and blue. The water is visible in the background.

Docker for ASP.NET Core Developers

# Intro

# Challenges with traditional development

- Environment inconsistencies (dev, test, prod)
- Dependency and configuration conflicts
- Complex deployment and scaling processes
- Security and resource isolation issues

# How Containers Solve These Challenges

- **Consistency**

Same container image across all environments

- **Isolation**

Containers encapsulate dependencies and runtime

- **Portability**

Run containers anywhere (local, cloud, hybrid)

- **Efficiency**

Simplified scaling and continuous integration

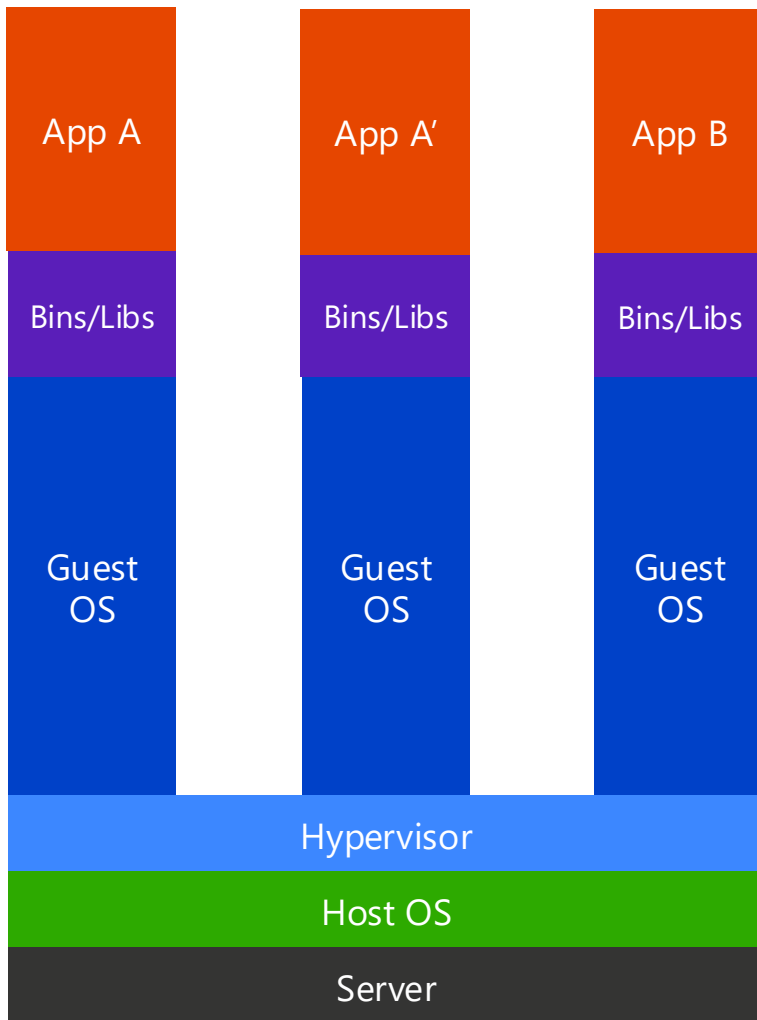


# Containers

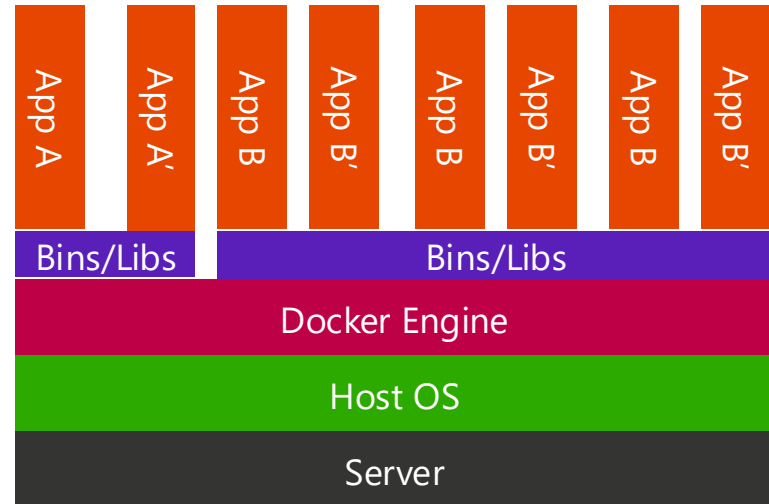
- Versioned artifact
- Isolated deployable unit
- Container image is bit by bit identical when deployed
- Abstraction of data center resources
  - Pool of compute, network and storage
- Orchestration is “Cattle Business”
  - “Forget about naming the servers and treating them like pets.”
  - “Run it for me please”







Containers are isolated, but share OS and, where appropriate, bins/libraries



# Docker Layers

**My ASP.NET Core Application**

**`mcr.microsoft.com/dotnet/core/aspnet:9.0`**

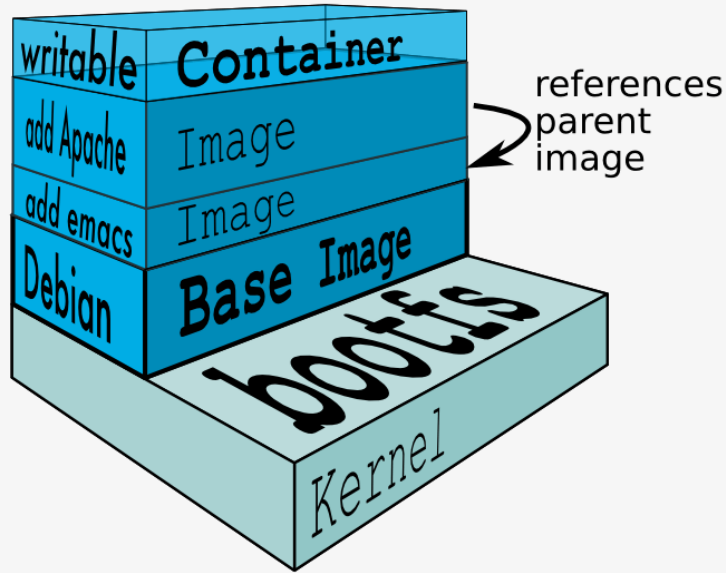
**`mcr.microsoft.com/dotnet/core/runtime`**

**`mcr.microsoft.com/dotnet/core/runtime-deps`**

**`amd64/debian:bookworm`**

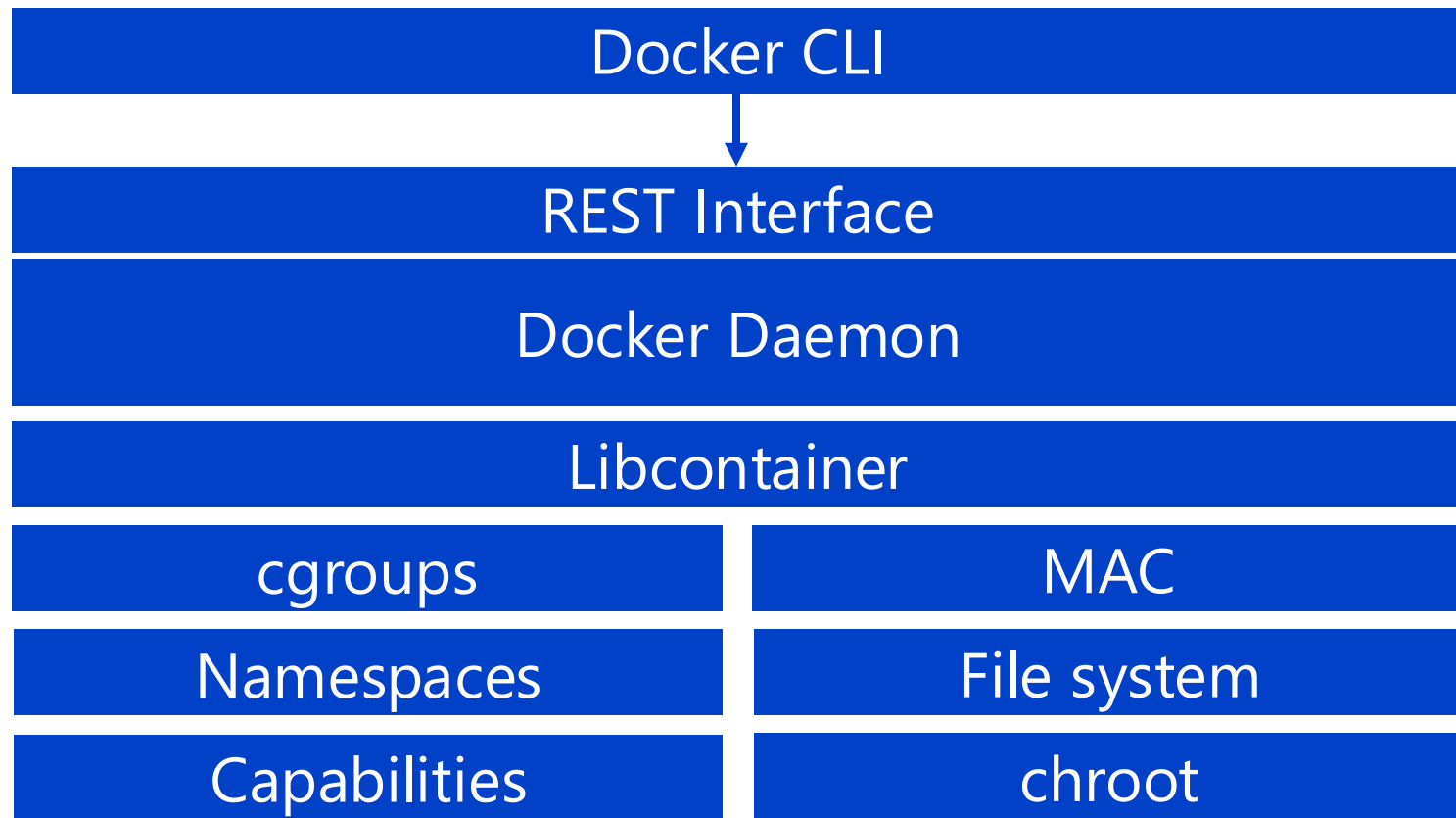


# File System Layers



- Rootfs stays read-only
- Union-mount file system over the read-only file system
- Multiple file systems stacked on top of each other
- Only top-most file system is writable
- Copy-on-write

# Docker Architecture



# Definitions

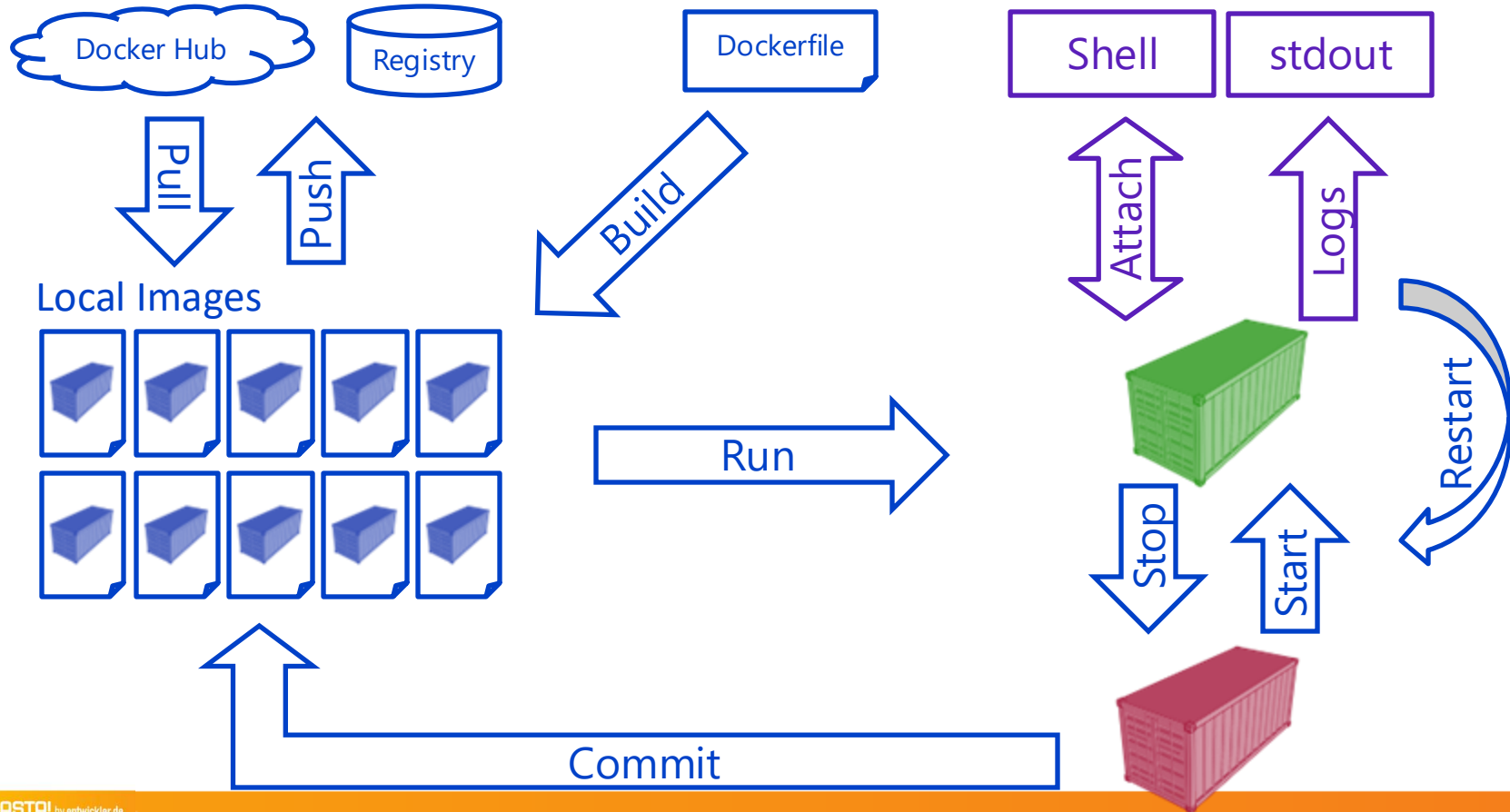
- Container

A container defines a software application and its dependencies wrapped in a complete filesystem including code, runtime, system tools, and libraries.

- Image

An image is a read-only snapshot of a Docker container.

# Docker Lifecycle



# Docker

## Build Time

- Dockerfile as definition
- Creates a read-only image
- Usually uses a base image
- Builds a docker file system by executing commands and processes

## Runtime

- Instantiate a container from an image
- Has isolated CPU, RAM, Disk and Network
- Runs processes

# Demo

A person wearing a dark hoodie is seated in an office chair, viewed from the side, working at a desk. The desk is equipped with a laptop and a large multi-monitor setup. The primary monitor displays a complex code editor with a dark theme, showing various code files and a central workspace. To the left, two smaller monitors are visible, one showing code and the other a play button icon. The desk is illuminated by two adjustable desk lamps, creating a focused and professional atmosphere. In the background, a shelf holds several white binders, and the overall lighting is a mix of cool blue tones and warm yellow light from the lamps.

Sample Application



A background image showing a rowing team in a boat, with rowers in blue and red uniforms pulling oars. The image is slightly blurred to emphasize the text overlay.

Docker for ASP.NET Core Developers

# Building Containers

# Challenges: Building and running Containers

- **Inconsistent Build Environments**

Different compilers and development setups

- **Overly Large & Insecure Images**

Unnecessary dependencies in runtime images

- **Inefficient Build Processes**

Lack of caching and redundant steps



# Demo



## Dockerfile Basics

- Multi Stage
- Caching

# Demo Recap - Dockerfile

- **Clear Separation of Concerns**

Isolate build and runtime environments

- **Optimized Dockerfile Practices**

Multi-stage builds to reduce image size

- **Efficient Layer Caching**

Minimize rebuild times with smart file copying



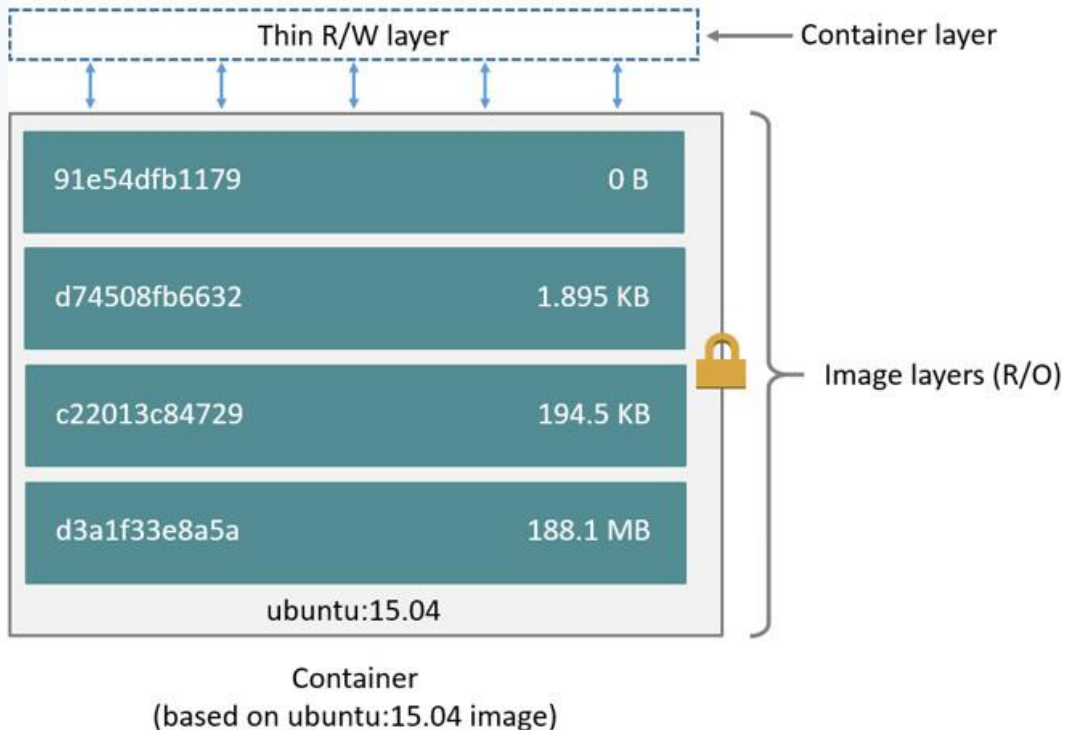
# Layer Caching

- Docker creates images using layers
- Each command in Dockerfile creates a new layer
- Each layer contains the filesystem changes  
(diff between before and after command)
- To build images faster, docker uses layer caches
- Works with RUN, COPY, ADD



# Images and Layers

```
FROM ubuntu:18.04
COPY . /app
RUN make /app
CMD python /app/app.py
```



# Layer Caching - RUN

- RUN executes a command in the Docker image
- If the layer generated by the run command already exists, it is reused, and the command is not executed
- Cache Hash: the command itself

# Layer Caching - COPY

- COPY imports one or more external files into a Docker image
- If the contents of all the external files are the same, the layer cache is used
- Cache Hash: hash on all the file contents



# Layer Caching - ADD

- ADD imports one or more external files into a Docker image
- If the contents of all the external files are the same, the layer cache is used
- Cache Hash: hash on all the file contents

Note: COPY vs ADD → ADD allows external URLs and can extract tar files to the image



# Taking Advantage of Layer Caching

- If a layer has a cache miss, then all subsequent layers will be built and are not cached
- Structure your Dockerfile in a way to get most out of the caching and therefore build performance

# Challenges: Image Size

- **Impact on Deployments**

Larger images slow down downloads and startup times  
Increased resource usage during storage and transfer

- **Variability by Base Image**

Standard, Alpine, and Chiseled images differ greatly in size



# Demo

A person is seen from behind, sitting in an ergonomic chair at a desk in a dimly lit office. They are working on a large multi-monitor setup. The primary monitor displays a code editor with a dark theme and syntax-highlighted code. To its left, a laptop also shows code. Further left, another desk with two more monitors is visible, one of which has a play button icon. The scene is illuminated by desk lamps, creating a focused and professional atmosphere. In the background, there are shelves filled with binders or files.

Image Size

# Demo Recap – Image Size

- Optimization Strategies

Tailor your base image choice to your application's needs

- Balancing Trade-offs

Consider features versus size – minimal images for runtime efficiency

Multi-Language Support (ICU)

AOT Compilation – no dynamic loading, no reflection

- Base image is also relevant for security

Hardening, dependencies / tools, distro-less



# Challenges: Dependencies of Base Images

- **Dependency Overload**

Base images include numerous libraries and dependencies

- **Risk Exposure**

Each dependency may have vulnerabilities

- **Need for Proactive Security**

Continuous monitoring and tracking are essential





# Demo

A person wearing a dark hoodie is seated in an office chair, viewed from the side, working at a desk. The desk is equipped with a laptop and a large multi-monitor setup. The screens display lines of code in a dark-themed editor. A desk lamp provides focused light on the workspace. In the background, there are shelves with binders and another desk with more monitors, all in a dimly lit environment.

Security Scans

# Demo Recap – Security Scan

- **SBOM: Your Dependency Inventory**

Generate a Software Bill of Materials to list all components

- **Automated Vulnerability Scanning**

Utilize Docker Scout and Trivy for security analysis

- **Proactive Security Management**

Identify and remediate issues before deployment

Integrate security scans in your CI builds as well as run them periodically.





# Challenges: Container Security

- **Challenge: Elevated Risks**

Containers running as root with open permissions

Multiple dependencies increase the attack surface

- **Not all base images are secure by default**

Most base images run with root user

Large images with many dependencies



# Demo

A person wearing a dark hoodie is seated in an ergonomic chair, viewed from the side, working at a desk in a dimly lit office. The desk is equipped with a laptop and a large multi-monitor setup. The primary monitor displays a complex code editor with a dark theme, showing various code snippets and a file explorer on the left. A secondary monitor to the left also shows code. A third monitor further back displays a large white play button icon. The person's hands are on a keyboard. A desk lamp with a green shade provides focused light on the workspace. In the background, there are shelves filled with binders and another desk with a lamp. The overall atmosphere is professional and tech-oriented.

Container Hardening

# Demo Recap – Hardening

- **Hardened Base Image**

Improved security with adjusted file permissions and non-root user  
Offers balance: enhanced security while still allowing debugging tools

- **Chiseled Image**

Distro-less design minimizes extraneous components  
Focus on maximum security and performance for production



# Challenges: Extract Artifacts

- **More artifacts than just the application**

Build produce test results, database schemas, client packages, etc.

The final image does not contain them, but they are needed for the process.

- **Intermediate image identification**

We need to identify intermediate images and extract artifacts from them.





# Demo

A person is seen from behind, sitting at a desk in a dimly lit office. They are working on a computer with multiple monitors. The main monitor displays code or test results. A laptop is also open on the desk. A desk lamp is on the left, and a shelf with binders is on the right. The overall atmosphere is professional and focused.

Test Results Extraction

# Demo Recap – Extract artifacts

- **Label Images**

Images – also intermediate images – can be labelled and therefore easily be identified.

Unique label values are needed if multiple builds are performed on the same environment.

- **Copy files between Container and Host**

Files can be copied from the container to the host.

The host (i.e. CI build) can then work with those files.

# Challenges: Secrets in Build

- Private Feeds

Private feeds need authentication.

The docker build does not know anything about the outer environment

- Use secrets in container build

Make sure the secrets are not persisted in the produced image.





# Demo

A person wearing a dark hoodie is seated in an office chair, viewed from the side, working at a desk. The desk is equipped with a laptop and a large multi-monitor setup. The primary monitor displays a complex code editor with a dark theme, showing various code files and snippets. A secondary monitor to the left also shows code. A third monitor further back displays a large white play button icon. The workspace is illuminated by two adjustable desk lamps, creating a focused and professional atmosphere. In the background, a tall shelving unit holds numerous white binders or folders. The overall scene suggests a technical or development environment.

Secrets in Build



# Demo Recap – Secrets

- **Multi-Stage Build**

Make sure that no build related files and configs make it into the target image

- **Use build args**

Provide arguments for the build which could contain secrets

- **Use Secrets**

Credentials are injected at build time, never stored in the final image.

Use the --secret option to securely access private feeds (e.g., NuGet) without hardcoding sensitive data.

Eliminates risk of exposing authentication details during runtime or in image layers.

Best practices for secure CI/CD pipelines by separating code and secrets.



A background image showing a rowing team in blue uniforms and red accents, pulling oars with yellow handles. The scene is outdoors on water, with a white wake visible. The text is overlaid on a semi-transparent white box.

Docker for ASP.NET Core Developers

# IDE Integration & Multi-Service Development

# IDE Integration (Visual Studio)

- Seamless Integration
- Fast Mode Debugging
- Full Developer Experience
- Configurable Options



# Container Tools Build Properties

Property name	Description
ContainerDevelopmentMode	Controls whether "build-on-host" optimization ("Fast Mode" debugging) is enabled. Allowed values are <b>Fast</b> and <b>Regular</b> .
ContainerVsDbgPath	The path for VSDBG debugger.
DockerDebuggeeArguments	When debugging, the debugger is instructed to pass these arguments to the launched executable.
DockerDebuggeeProgram	When debugging, the debugger is instructed to launch this executable.
DockerDebuggeeKillProgram	This command is used to kill the running process in a container.
DockerDebuggeeWorkingDirectory	When debugging, the debugger is instructed to use this path as the working directory.
DockerDefaultTargetOS	The default target operating system used when building the Docker image.
DockerImageLabels	The default set of labels applied to the Docker image.
DockerFastModeProjectMountDirectory	In <b>Fast Mode</b> , this property controls where the project output directory is volume-mounted into the running container.
DockerfileBuildArguments	Additional arguments passed to the <a href="#">Docker build</a> command.
DockerfileContext	The default context used when building the Docker image, as a path relative to the Dockerfile.
DockerfileFastModeStage	The Dockerfile stage (that is, target) to be used when building the image in debug mode.
DockerfileFile	Describes the default Dockerfile to use to build/run the container for the project. This value can be a path.
DockerfileRunArguments	Additional arguments passed to the <a href="#">Docker run</a> command.
DockerfileRunEnvironmentFiles	Semicolon-delimited list of environment files applied during Docker run.
DockerfileTag	The tag to use when building the Docker image. In debugging, a ":dev" is appended to the tag.

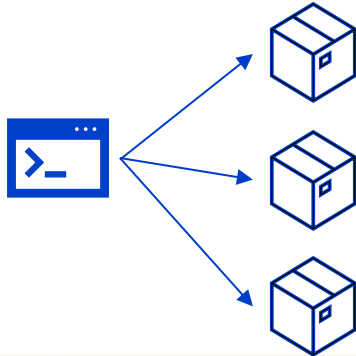
# Sample

```
1 <Project Sdk="Microsoft.NET.Sdk.Web">
2
3   <PropertyGroup>
4     <TargetFramework>net8.0</TargetFramework>
5     <Nullable>enable</Nullable>
6     <ImplicitUsings>enable</ImplicitUsings>
7     <UserSecretsId>8c7ab9a5-d578-4c40-8b6d-54d174002229</UserSecretsId>
8     <DockerDefaultTargetOS>Linux</DockerDefaultTargetOS>
9     <!-- In CI/CD scenarios, you might need to change the context. By default, Visual Studio uses the
10      | folder above the Dockerfile. The path is relative to the Dockerfile, so here the context is
11      | set to the same folder as the Dockerfile. -->
12     <DockerfileContext>.</DockerfileContext>
13     <!-- Set `docker run` arguments to mount a volume -->
14     <DockerfileRunArguments>-v ${MSBuildProjectDirectory}/host-folder:/container-folder:ro</DockerfileRunArguments>
15     <!-- Set `docker build` arguments to add a custom tag -->
16     <DockerfileBuildArguments>-t contoso/front-end:v2.0</DockerfileBuildArguments>
17   </PropertyGroup>
18
19   <ItemGroup>
20     <PackageReference Include="Microsoft.VisualStudio.Azure.Containers.Tools.Targets" Version="1.20.1" />
21   </ItemGroup>
22
23 </Project>
```

# Docker Compose – Multi Container Applications

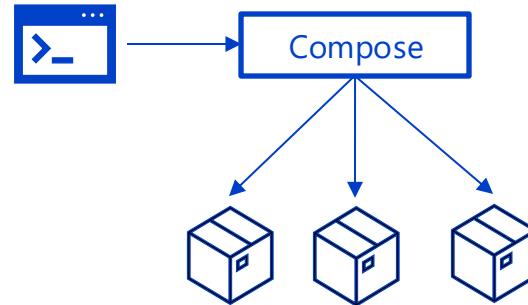
## Without Compose

- Build and run one container at a time
- Must be careful with dependencies and startup order
- Manually connect containers together



## With Compose

- Single file to define multi container applications
- Single command to deploy the entire application
- Compose takes care of dependencies
- Works with Docker Swarm, Networking, Volumes, Universal Control Plane



# What is Docker Compose

- A tool for running multi-container applications
- A YAML file definition to configure your application
- Works on all stages: development, testing, production
- Single command to create, start and stop services based on the configuration



# Demo

A person is seen from the side, sitting in an ergonomic chair at a desk in a dimly lit office. They are working on a multi-monitor setup. The main monitor displays a code editor with a dark theme and syntax-highlighted code. To the left, a laptop also shows code. In the background, another desk with more monitors and a desk lamp is visible. A shelf with binders is on the right. The scene is illuminated by desk lamps, creating a focused and professional atmosphere.

Docker Compose

# Demo

A person is sitting at a desk in a dimly lit room, working on a computer. The desk has a laptop and a large monitor displaying code. A desk lamp is on the desk, and another lamp is on a shelf in the background. The person is wearing a hoodie and is looking at the monitor. The room has shelves with binders and a clock on the wall.

Debugging in IDE



# Demo

A person wearing a dark hoodie is seated in an ergonomic office chair, viewed from the side. They are working at a desk in a dimly lit office environment. The desk is equipped with a laptop and a large multi-monitor setup. The primary monitor displays a complex code editor with a dark theme, showing various code files and snippets. A secondary monitor to the left also displays code. A third monitor further back shows a play button icon. The person's hands are on a keyboard. A desk lamp with a green shade is positioned above the desk, casting a focused light. In the background, there are shelves filled with binders and another desk with a lamp. The overall atmosphere is professional and tech-oriented.

Networking

# Demo

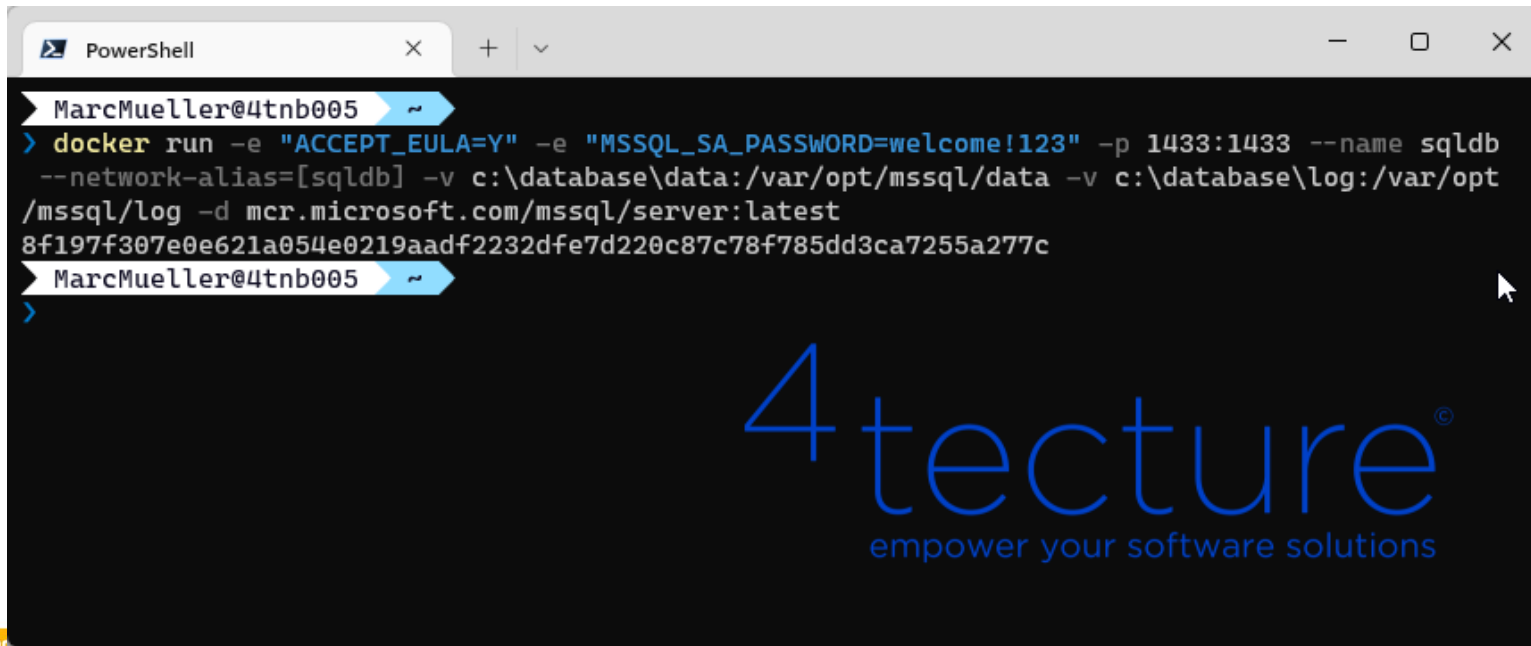
A person wearing a dark hoodie is seated in an ergonomic chair, viewed from the side, working at a desk in a dimly lit office. The desk is equipped with a laptop and a large multi-monitor setup. The primary monitor displays a complex code editor with a dark theme, showing various code files and a central pane with code snippets. To the left, two smaller monitors also display code. A desk lamp with a warm glow is positioned to the left of the person, and another lamp is positioned above the main monitor. On the desk, there is a black can, a mouse, and some papers. In the background, a shelf holds several white binders or folders. The overall atmosphere is professional and focused on software development.

SQL Server Container



# Local Development Environment

- Fast
- Multiple versions side-by-side
- External persistency

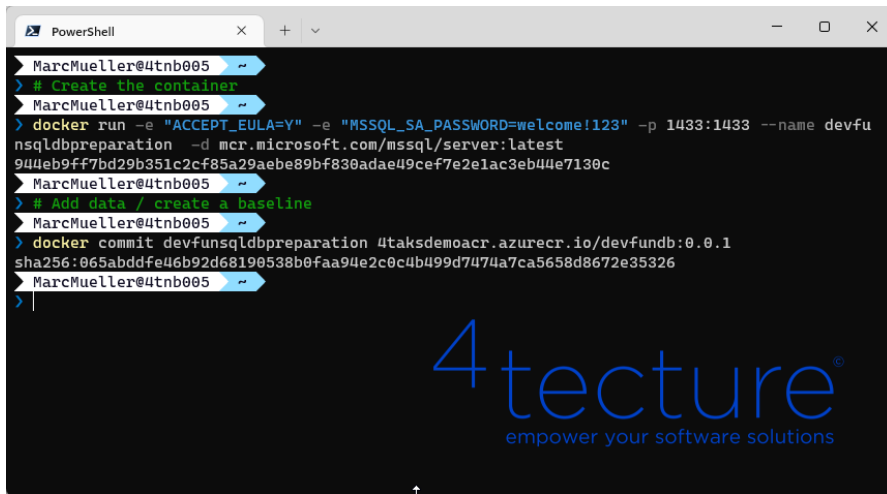


```
PowerShell
MarcMueller@4tnb005 ~
> docker run -e "ACCEPT_EULA=Y" -e "MSSQL_SA_PASSWORD=welcome!123" -p 1433:1433 --name sqlldb
--network-alias=[sqlldb] -v c:\database\data:/var/opt/mssql/data -v c:\database\log:/var/opt
/mssql/log -d mcr.microsoft.com/mssql/server:latest
8f197f307e0e621a054e0219aadf2232dfe7d220c87c78f785dd3ca7255a277c
MarcMueller@4tnb005 ~
>
```

4tecture<sup>®</sup>  
empower your software solutions

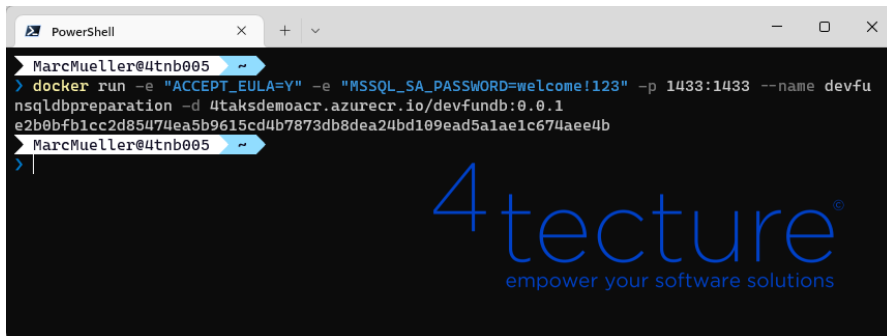
# Fast Baseline for Testing

- Backup / Restore may increase execution time drastically
- Container images can be built with database baselines
- Multiple application data versions for migration testing
- Specific data baseline for corresponding tests



```
MarcMueller@4tnb005 ~
> # Create the container
MarcMueller@4tnb005 ~
> docker run -e "ACCEPT_EULA=Y" -e "MSSQL_SA_PASSWORD=welcome!123" -p 1433:1433 --name devfu
nsqldbpreparation -d mcr.microsoft.com/mssql/server:latest
944eb9ff7bd29b351c2cf85a29aebef830adae49cef7e2e1ac3eb44e7130c
MarcMueller@4tnb005 ~
> # Add data / create a baseline
MarcMueller@4tnb005 ~
> docker commit devfunsqldbpreparation 4taksdemoacr.azurecr.io/devfundb:0.0.1
sha256:065abddfe46b92d68190538b0faa94e2c0c4b499d7474a7ca5658d8672e35326
MarcMueller@4tnb005 ~
>
```

4tecture®  
empower your software solutions



```
MarcMueller@4tnb005 ~
> docker run -e "ACCEPT_EULA=Y" -e "MSSQL_SA_PASSWORD=welcome!123" -p 1433:1433 --name devfu
nsqldbpreparation -d 4taksdemoacr.azurecr.io/devfundb:0.0.1
e2b0bfb1cc2d85474ea5b9615cd4b7873db8dea24bd109ead5a1ae1c674aee4b
MarcMueller@4tnb005 ~
>
```

4tecture®  
empower your software solutions

A background image showing a rowing team in a boat. The rowers are wearing blue and red uniforms and are captured in a synchronized rowing motion. The focus is on the oars and the rowers' hands, with the water visible in the foreground.

Docker for ASP.NET Core Developers

Q & A

4tecture<sup>®</sup>  
empower your software solutions



# Recap

- Containers are the ideal artifact
- Multi-Stage Container Builds include your full CI tooling and an optimized runtime image
- Optimize your final container images – size, dependencies, and security
- Seamless IDE integration with debugging support



# Thank you for your attention!

If you have any questions do not hesitate to contact us:

4tecture GmbH  
Industriestrasse 25  
CH-8604 Volketswil

+41 44 508 37 00  
info@4tecture.ch  
www.4tecture.ch

Marc Müller  
Principal Consultant

[www.powerofdevops.com](http://www.powerofdevops.com)



A background image showing several hands of different skin tones reaching towards the center, where they are assembling a small cluster of four interlocking wooden puzzle pieces. The pieces are colored light brown, white, red, and green. The overall scene is softly blurred, focusing attention on the hands and the puzzle pieces.

4tecture<sup>©</sup>

empower your software solutions