SYTD Test 3

# GRPC

## Was heißt das?

Grpc

Remote

procedure

calls

## Für was benutzt man es?

* Client ó Server Kommunikation
* Server ó Client Kommunikation
* X ó Y Kommunikation

## Transport Channel

GRPC ist agnostisch über den Transport Channel – das heißt, es ist egal welches Transportprotokoll man benutzt, wir benutzen es über http

In der Informatik bedeutet agnostisch, dass etwas unabhängig von bestimmten Systemen oder Formaten funktionieren kann. Dieser Begriff kann sich auf Software, Hardware, Geschäftsprozesse oder Daten beziehen.

## Warum grpc anstatt andere?

* **Effiziente Serialisierung →** Kleine Nachrichtengröße
* **Schnittstellendefinition kann leicht geteilt werden** (z.B. .protobuf-Dateien)
* **Unterstützung für die Codegenerierung sowohl auf der Serverseite als auch auf der Clientseite in vielen Programmiersprachen**
* I**ntegration von Systemen in verschiedenen Programmiersprachen**

## Programming Model

Ein Bild, das Text, Screenshot, Schrift, Diagramm enthält.

KI-generierte Inhalte können fehlerhaft sein.

Eine Client App ruft eine Methode auf einer Server-App auf einer anderen Maschine auf, als wäre es ein lokales Objekt

## Multiple Unterschiedliche Clients

Ein Bild, das Text, Screenshot, Schrift, Reihe enthält.

KI-generierte Inhalte können fehlerhaft sein.

## Message Format

Ein Bild, das Text, Screenshot, Software, Schrift enthält.

KI-generierte Inhalte können fehlerhaft sein.

**1. service Greeter { ... }**

* Definiert einen **gRPC-Service** namens Greeter.
* Ein **Service** ist wie eine "Schnittstelle" (Interface), die Funktionen (RPCs) anbietet.

**2. rpc SayHello (HelloRequest) returns (HelloReply);**

* SayHello ist eine **Remote Procedure Call (RPC)**-Methode.
* Sie nimmt eine Nachricht vom Typ HelloRequest entgegen und gibt eine Nachricht vom Typ HelloReply zurück.

Anders gesagt:  
→ **Client** sendet eine HelloRequest →  
→ **Server** verarbeitet sie und antwortet mit einer HelloReply.

**3. message HelloRequest { ... }**

* HelloRequest ist eine **Nachricht (Message)**, die zwei Strings enthält:
  + firstName (Vorname)
  + lastName (Nachname)

Das ist das, was der Client an den Server schickt.

**4. message HelloReply { ... }**

* HelloReply ist die **Antwortnachricht** des Servers.
* Sie enthält einen String greeting, also die Begrüßung.

Zum Beispiel könnte greeting = "Hallo Max Mustermann!" sein.

## Grpc und protobuf in einer .net solution

Ein Bild, das Text, Handschrift, Screenshot, Schrift enthält.

KI-generierte Inhalte können fehlerhaft sein.

### Notwendige NuGet Packages:

Grpc.Net.Client

Google.Protobuf

Grpc.Tools

### Dann einen Ordner (Grpc) im Shared Projekt erstellen und folgendes File hinzufügen:

Ein Bild, das Text, Screenshot, Software, Multimedia-Software enthält.

KI-generierte Inhalte können fehlerhaft sein.

**1.** syntax = "proto3";

* Gibt an, dass du **Protocol Buffers Version 3** benutzt.

**2.** option csharp\_namespace = "SavingsPlanner.Shared.Grpc";

* Legt fest, in welchem **Namespace** der **C#-Code** landet.
* Verhindert Namenskonflikte und sortiert die Klassen ordentlich.

**3.** package Rewards;

* Definiert das **gRPC-Paket** mit Namen Rewards.
* Wird benutzt, um Services und Nachrichten logisch zu gruppieren.

**4.** service Rewards { ... }

* Definiert einen **gRPC-Service** namens Rewards.
* Der Service kann Methoden bereitstellen, die der Client aufruft.

**5.** rpc GetRewards (RewardRequest) returns (RewardReply);

* Definiert eine **Service-Methode** namens GetRewards.
* Der Client sendet eine **RewardRequest**.
* Der Server antwortet mit einer **RewardReply**.

**6.** message RewardRequest { }

* Definiert die **Anfrage-Nachricht** RewardRequest.
* Diese Nachricht ist **leer** (keine Felder).
* Bedeutet: Client fragt einfach **nur** an, ohne Daten mitzuschicken.

**7.** message RewardReply { repeated RewardItem rewards = 1; }

* Definiert die **Antwort-Nachricht** RewardReply.
* Enthält eine **Liste** (repeated) von **RewardItem**-Objekten.
* → Mehrere Belohnungen können zurückgegeben werden.

**8.** message RewardItem { ... }

* Beschreibt **eine einzelne Belohnung**.
* Felder:
  + int32 id = 1; → Die ID der Belohnung.
  + string title = 2; → Der Titel/Name der Belohnung.
  + double targetAmount = 3; → Der Zielbetrag (z.B. Sparziel).
  + string description = 4; → Eine Beschreibung der Belohnung.

### Shared.csproj

Ein Bild, das Text, Screenshot, Schrift enthält.

KI-generierte Inhalte können fehlerhaft sein.

### Nuget Packages im Client Projekt:

Grpc.Net.Client.Web

### Program.cs im client project

Ein Bild, das Text, Screenshot enthält.

KI-generierte Inhalte können fehlerhaft sein.

Da brauchts glaub ich keine Erklärung, yk

### Blazor component vom Client:

Ein Bild, das Text, Screenshot, Schrift enthält.

KI-generierte Inhalte können fehlerhaft sein.

Hier auch nt

### Nuget Packages im Server project:

Grpc.AspNetCore.Web

Grpc.AspNetCore

### Jetzt einen Service implementieren, indem man von den generierten Klasssen im Shared erbt.

Ein Bild, das Text, Screenshot, Schrift enthält.

KI-generierte Inhalte können fehlerhaft sein.

* Service-Klasse wird erstellt.
* Datenbank/Repository wird benutzt, um Belohnungen zu holen.
* Die Belohnungen werden in die Antwort eingefügt.
* Die Antwort wird an den Client geschickt.

### Program.cs vom Server Projekt

Ein Bild, das Text, Screenshot, Schrift enthält.

KI-generierte Inhalte können fehlerhaft sein.

### Automapper:

Ein Bild, das Text, Screenshot, Schrift enthält.

KI-generierte Inhalte können fehlerhaft sein.

* AutoMapper lernt hier: "So wandelst du ein Reward-Objekt in ein RewardItem-Objekt."
* Für das Feld TargetAmount musst du eine Typ-Umwandlung machen ((double)).
* Der Rest der Felder (z.B. ID, Title, Description) wird automatisch gemappt, wenn die Namen gleich sind.

# Docker

## Was is da Docker?

Docker is eine Container Technologie – Man erstellt und managed Container.

## Container

Ein Container ist eine standardisierte Einheit von Software.

Ein Container bündelt den Programmcode zusammen mit allen Abhängigkeiten, die benötigt werden, um den Code auszuführen.

Zum Beispiel eine C#-Anwendung zusammen mit der .NET-Laufzeitumgebung.

## Warum überhaupt?

Container lösen das Problem von "es läuft aber auf meinem Rechner".  
Dieses Problem tritt auf, wenn man eine Anwendung entwickelt, sie auf dem eigenen Rechner testet, sie anschließend in die Produktionsumgebung überträgt – und dort plötzlich etwas nicht funktioniert, weil sich die Umgebung in irgendeinem Detail vom lokalen Rechner unterscheidet.

Unser Ziel ist es, die Anwendung in genau (!) derselben Umgebung zu entwickeln, zu testen und später auch auszuführen**.**

## Container

* Ein und derselbe Container liefert immer exakt das gleiche Anwendungsverhalten – egal wo und von wem er ausgeführt wird.
* Moderne Betriebssysteme haben bereits eingebaute Unterstützung für Container.
* Docker vereinfacht die Erstellung und Verwaltung von Containern erheblich.

## Zuverlässigkeit und reproduzierbare Umgebungen

* Wir wollen exakt die gleiche Umgebung für Entwicklung und Produktion haben →→ Das stellt sicher, dass die Anwendung genau so funktioniert, wie sie getestet wurde.
* Es soll einfach sein, eine gemeinsame Entwicklungsumgebung mit (neuen) Kolleginnen und Kollegen zu teilen.

## VMs

Ein Bild, das Text, Schrift, Screenshot, Handschrift enthält.

KI-generierte Inhalte können fehlerhaft sein.

## VMs(2)

| **Pro** |
| --- |
| Getrennte Umgebungen |
| Umgebungsspezifische Konfigurationen sind möglich |
| Umgebungen können zuverlässig geteilt und reproduziert werden |
| **Kontra** |
| Redundante Duplizierung, Platzverschwendung |
| Leistung kann langsam sein, lange Startzeiten |
| Reproduktion auf einem anderen Rechner/Server möglich, aber teilweise kompliziert |

## Container

Ein Bild, das Text, Screenshot, Schrift, Zahl enthält.

KI-generierte Inhalte können fehlerhaft sein.

| **Docker-Container** |
| --- |

|  |  |
| --- | --- |
| * Geringer Einfluss auf das Betriebssystem, sehr schnell, minimaler Speicherplatzverbrauch |  |

|  |  |
| --- | --- |
| * Teilen, Neuaufbau und Verteilung sind einfach |  |

|  |  |  |
| --- | --- | --- |
| * Kapseln Apps/Umgebungen anstelle von „ganzen Maschinen“ | |  |
| **Virtuelle Maschinen** |

* Größerer Einfluss auf das Betriebssystem, langsamer, höherer Speicherplatzverbrauch
* Teilen, Neuaufbau und Verteilung können herausfordernd sein
* Kapseln „ganze Maschinen“ anstelle nur von Apps/Umgebungen

## Docker installieren

Ein Bild, das Text, Screenshot, Schrift, Reihe enthält.

KI-generierte Inhalte können fehlerhaft sein.

Man kann Docker über Docker Desktop oder über CLI benutzen

## Images

| **Images** | **Containers** |
| --- | --- |
| Templates/Blueprints for containers | The running “unit of software” |
| Contains code + required tools/runtimes | Multiple containers can be created based on one image |

## Ein Image – viele Container

Ein Bild, das Text, Entwurf, Zeichnung, Diagramm enthält.

KI-generierte Inhalte können fehlerhaft sein.

## Image builden mit einem Dockerfile

Ein Bild, das Text, Screenshot enthält.

KI-generierte Inhalte können fehlerhaft sein.

Dann „docker build .“

## Einen Container basierend auf einem image ausführen

docker run <image\_name>

oder

docker run -p <host\_port>:<container\_port> <image\_name>

**Images sind schreibgeschützt**

Nachdem ein Image erstellt wurde, werden Änderungen nicht angezeigt

(Änderungen in den Quellcodedateien)

Multi-Stage-Dockerfiles

werden am häufigsten verwendet, um zwischen Build- und Run-Umgebungen zu unterscheiden. Das Build-Stadium enthält das SDK, während das finale Stadium nur das Runtime enthält.

Ein Bild, das Text, Screenshot, Schrift enthält.

KI-generierte Inhalte können fehlerhaft sein.

Dieses **Dockerfile** baut eine .NET Core-Anwendung in mehreren Schritten:

1. Es nutzt ein SDK-Image, um den Code zu bauen.
2. Es veröffentlicht die Anwendung (um sie für die Ausführung vorzubereiten).
3. Es kopiert die veröffentlichten Dateien in das **Runtime-Image**.
4. Schließlich wird ein Container erstellt, der die Anwendung ausführt, indem der dotnet-Befehl aufruft.

Dieser mehrstufige Ansatz sorgt für **kleinere Container**, weil nur das benötigte **Runtime-Image** für die Ausführung der Anwendung verwendet wird.

# Docker – Data and Volumes

## Datentypen

Ein Bild, das Text, Handschrift, Schrift, Screenshot enthält.

KI-generierte Inhalte können fehlerhaft sein.

## Container und Image Daten

Ein Bild, das Text, Screenshot, Schrift, Handschrift enthält.

KI-generierte Inhalte können fehlerhaft sein.

## Container Daten

Daten können innerhalb von Containern als Dateien gespeichert werden, genau so, wie du es normalerweise auf deiner lokalen Entwicklungsmaschine tun würdest.

Allerdings gehen die Daten verloren, sobald ein Container entfernt wird.

## Volumes

Volumes sind Ordner auf der Festplatte deines Host-Rechners, die in Container eingebunden („verfügbar gemacht“, gemappt) werden.

Ein Bild, das Text, Screenshot, Schrift, Design enthält.

KI-generierte Inhalte können fehlerhaft sein.

Volumes bleiben bestehen, wenn ein Container heruntergefahren wird. Wenn ein Container (neu) gestartet wird und ein Volume einbindet, sind alle darin enthaltenen Daten im Container verfügbar.  
Ein Container kann Daten in ein Volume schreiben und Daten daraus lesen.

## Volumes erstellen

Ein Bild, das Text, Screenshot enthält.

KI-generierte Inhalte können fehlerhaft sein.

Der Befehl VOLUME gibt einen Pfad innerhalb des Containers an, der als Volume behandelt wird.  
Die Zuordnung zum Dateisystem des Hosts wird beim Starten eines Containers durch den run-Befehl festgelegt.

## Volumes vs Bind Mounts

Ein Bild, das Text, Screenshot, Schrift, Handschrift enthält.

KI-generierte Inhalte können fehlerhaft sein.

## Container / Volume Interaktion

Ein Bild, das Text, Handschrift, Schrift, Diagramm enthält.

KI-generierte Inhalte können fehlerhaft sein.

Kurzgesagt:

* Container können Daten lesen und schreiben. Volumes helfen bei der Datenspeicherung, Bind Mounts bei der direkten Interaktion mit Containern.
* Container können zwar Daten lesen und schreiben, aber geschriebene Daten gehen verloren, wenn der Container entfernt wird.
* Volumes sind Ordner auf dem Host-Rechner, die von Docker verwaltet und in den Container eingebunden werden.
* **Benannte Volumes** (Named Volumes) überleben das Entfernen eines Containers und können daher zur Speicherung von persistenten Daten genutzt werden.
* **Anonyme Volumes** (Anonymous Volumes) sind an einen Container gebunden – sie können verwendet werden, um (temporäre) Daten innerhalb des Containers zu speichern.
* **Bind Mounts** sind Ordner auf dem Host-Rechner, die vom Benutzer angegeben und in Container eingebunden werden – ähnlich wie benannte Volumes.

# Docker Übungen

UE 1 und 2 sind nur copy paste, deswegen is da ned drinnen

## Docker Übung 5

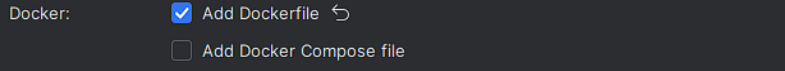
**Ausgangssituation:** Wir haben einen fertigen Code bekommen – und sollen das Ganze „dockerisieren“ (=Image erstellen, Container machen => starten & stoppen, in die Bash gehen, etc...)

### Schritt 1:

Ein Bild, das Text, Screenshot, Schrift enthält.

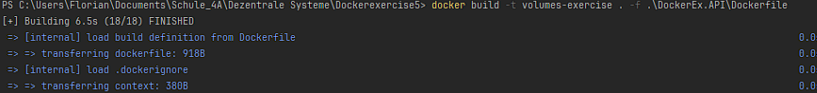
KI-generierte Inhalte können fehlerhaft sein.

Diesen Schritt kann man sich sparen, wenn man bei der Erstellung des Projektes folgenden Geheimtrick kennt:



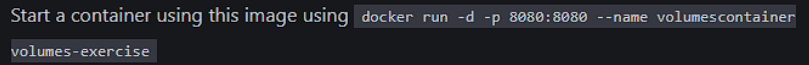
### Schritt 2:



Mit docker build erstellt man ein Image. Ich hatte das Problem, dass das Dockerfile, egal wo ich war, nie gefunden wurde und ich deswegen mit -f (File) den Pfad explizit angeben musste. 

Nun kann man in Docker-Desktop das neue Image begutachten:



Schritt 3: 

Container starten, -d für „detached mode“ (Im Hintergrund laufen lassen), -p für Port :, --name eh klar.



Lange Zeichenkette = Container-ID, falls der im Detached-Mode ist.

### Schritt 4:

Ein Bild, das Text, Screenshot, Software, Computersymbol enthält.

KI-generierte Inhalte können fehlerhaft sein.

Swagger aufrufen, Port vom Container nehmen!!! Ausführen und dann im Terminal nachschauen:



Und jetzt in /persistent nachschauen, ob die Datei generiert wurde:

Ein Bild, das Text, Screenshot, Schrift enthält.

KI-generierte Inhalte können fehlerhaft sein.

### Schritt 5:

Ein Bild, das Text, Schrift, Screenshot enthält.

KI-generierte Inhalte können fehlerhaft sein.

Neustarten und nachschauen, ob das generierte File noch da ist. Ja.

### Schritt 6:

Ein Bild, das Text, Schrift, Screenshot enthält.

KI-generierte Inhalte können fehlerhaft sein.

Container stoppen & removen

### Schritt 7:



Container wieder erstellen + Parameter „--rm“ ergänzen: Wenn man den Container stoppt, wird der auch gleich automatisch gelöscht



File ist nicht mehr da.

### Schritt 8:

Ein Bild, das Text, Screenshot, Schrift enthält.

KI-generierte Inhalte können fehlerhaft sein.

VOLUME“ ergänzen und Image neu erstellen (Es wird da dann nicht alles neu erstellt, sondern nur die Änderungen)

### Schritt 9:

 Ein Bild, das Text, Schrift, Screenshot enthält.

KI-generierte Inhalte können fehlerhaft sein.

Volume ist defined & mit „inspect“ kann man das Ganze auch begutachten#

### Schritt 10:

Ein Bild, das Text, Schrift, Screenshot enthält.

KI-generierte Inhalte können fehlerhaft sein.

Die zwei Run’s ergänzen, Image neu erstellen + Container ins Leben rufen und in Docker Desktop nachschauen, ob ein Volume erstellt wurde

Ein Bild, das Text, Screenshot, Schrift, Software enthält.

KI-generierte Inhalte können fehlerhaft sein.

Unter „Inspect“ in Docker Desktop sieht man auch meine Mounts:

Ein Bild, das Text, Screenshot, Schrift enthält.

KI-generierte Inhalte können fehlerhaft sein.

### Schritt 11:

Container löschen + Nachschauen, ob die Dateien noch da sind + ob das Volume noch da ist. ➔ Volume ist nicht mehr da, Files auch verloren gegangen

### Schritt 12:



Neuen Container starten, mit -v geben wir unser Volume an.

Ein Bild, das Screenshot, Text, Schwarz, Reihe enthält.

KI-generierte Inhalte können fehlerhaft sein.

Port hat sich auch geändert, wir haben nach außen hin jetzt 8081.

### Schritt 13:

Daten generieren, Container stoppen + löschen, ist Volume noch da? + Container noch mal starten, ob Files da sind.

Erkenntnisse:

➔ Volume nach Container-Löschung trotzdem noch da

➔ „File3“ auch nach dem Löschen + Neu-Erstellen des Containers noch da



## Docker Übung 6

**Instructions**

Your goal is to have a docker container running that uses your most recent files on

your host system as the application code that will be executed when you visit the web

page running in the container. This way you can have a well-defined execution

environment (using docker for managing the dependencies an environment), but still

be able to make rapid changes to your code and test it out while developing.

server.mjs

Ein Bild, das Text, Schrift, Screenshot enthält.

KI-generierte Inhalte können fehlerhaft sein.

Package.json

Ein Bild, das Text, Screenshot, Schrift, Design enthält.

KI-generierte Inhalte können fehlerhaft sein.

Dockerfile

Ein Bild, das Text, Schrift, weiß, Design enthält.

KI-generierte Inhalte können fehlerhaft sein.

Ein Bild, das Text, Schrift, weiß, Screenshot enthält.

KI-generierte Inhalte können fehlerhaft sein.

.dockerignore

Ein Bild, das Text, Design, Algebra, Guide enthält.

KI-generierte Inhalte können fehlerhaft sein.

.env

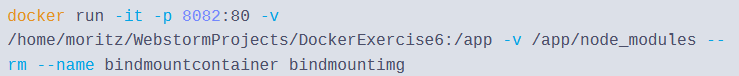


**Build an image using the Dockerfile.**

Give it a tag of bindmountimg



**Run a container based on the image using**

Parameter:

-i = interaktiv (hält STDIN offen, auch wenn keine Verbindung besteht)  -t = erstellt ein pseudo-TTY (terminal) – (Nützlich für interaktive Shells wie „bash“ oder „nodemon“)

-v = Volume mounten (Also Ordner C:\Users... mit /app verknüpfen)

-v /app/node\_modules = Dies ist ein anonymer Volume-Mount, der das Verzeichnis /app/node\_modules im Container vom Host trennt.

Warum? Um zu vermeiden, dass node\_modules vom Host überschrieben wird – z. B. wenn es im Host-Verzeichnis nicht vorhanden ist.

**Access the site in your browser**

**Ein Bild, das Text, Screenshot, Schrift enthält.

KI-generierte Inhalte können fehlerhaft sein.**

**Make changes to your code and change the HTML output of your**

**site.**

server.mjs

Ein Bild, das Text, Schrift, Screenshot enthält.

KI-generierte Inhalte können fehlerhaft sein.

After that, go into the container and type “rs”

**Refresh the page in the browser**

**Ein Bild, das Text, Screenshot, Schrift enthält.

KI-generierte Inhalte können fehlerhaft sein.**

**Try to answer the following questions:**

**How is the container port on which the application listens defined and managed?**

It is defined via an local environment file and then changed in the Dockerfile.

Lastly it is redirected to Port 8080

**What does the ARG command do in your Dockerfile and how can you use that**

**when building an image?**

It is used to define a variable in the build process. It can be used with a dollar sign.

**Why does the application listen on the port defined in the environment variable?**

So you can use it in the container. (goofy question)

**How can you set environment variables when running a container? (There are 2**

**ways. One might be connected to the .env file. But you should try it out to make**

**sure.)**

**.env File:**

****

other way



**What is nodemon and how did we use it in that example?**Nodemon is a tool that helps develop Node.js applications by automatically  
restarting the server whenever file changes are detected. It saves developers from  
manually stopping and restarting the server after each code change.  
**Which part of the docker run command defines a bind mount?**  
-v /home/moritz/WebstormProjects/DockerExercise6:/app  
and nodemon makes it that the server restarts automatically  
**Why did we have to specify an anonymous volume using -v /app/node\_modules ?  
What would happen if we left that out? (This last question is probably the most  
important question for you to understand how volumes and bind mounts interact  
with each other.)**If we don't specify an anonymous volume for node\_modules, it would be empty  
because it is excluded in .dockerignore file