

# Ollama

Ve výchozím nastavení naslouchá na adrese: <http://127.0.0.1:11434/>

## ⊗ Important

Pro naslouchání aplikace je důležité mít spuštění ollama soubor. (`ollama app.exe`)

## Změna v naslouchání adresy

- Proměnné prostředí ve Windows

V `System variables`, nastavte `OLLAMA_HOST` s hodnotou adresy, kde má ollama naslouchat.

- Následně restartujte aplikaci `ollama app.exe`.

## Vypnout automatické spuštění

### i Note

Aplikace ollama je ve výchozím stavu zapnuta při spuštění počítače.

Vypnout lze provést tímto způsobem:

- `Windows` + `R`, následně napsat: `shell:startup` -> odkliknout `OK`
- Odstranit zástupce na Ollama.

## Příkazy

- ▶ [Nainstalované moduly](#)
- ▶ [Stažení modelu](#)
- ▶ [Odstranění modelu](#)

# Uživatelská konfigurace

- [Porovnání souborů přes aplikaci Meld](#)

## Klávesové zkratky

- ▶ [Parametry metody](#)
- ▶ [Procházení seznamu](#)
- ▶ [XML komentáře - zalomení řádku](#)

# Vývojové metodiky

Techniky vývoje softwaru jsou postupy, které určují, jakým způsobem se vyvíjí software.

- ▶ [Agilní metodika \(Scrum\)](#)
- ▶ [Vodopádová metodika](#)
- ▶ [Kanban](#)

## Rychlé prototypování

Proces pro vytvoření funkčního modelu projektu co nejrychleji, aby bylo možné testovat a iterovat nápady.

### Note

V kontextu Unity to znamená vytvoření základní verze hry nebo aplikace, která zahrnuje pouze klíčové mechaniky a funkce.

## Rychlá iterace

- Rychle testovat a provádět nápady.
- Pokud vytvoříte prototyp, zkuste ho co nejdříve otestovat a získat zpětnou vazbu.
- Poté můžete na základě této zpětné vazby upravit a vylepšit svůj prototyp.

### Important

**Cílem je vytvořit funkční model** vašeho projektu, **ne dokonalý produkt**. Nebojte se udělat kompromisy v kvalitě, pokud to znamená, že můžete rychleji testovat a iterovat své nápady.

## Postup

### 1. Definice konceptu

Než se začne s prototypováním, měli byste mít jasnou představu o tom, co chcete vytvořit.

### Tip

To může zahrnovat definování klíčových mechanik, funkcí a cílů vašeho projektu.

## 2. Vytvoření základní scény v Unity

### Tip

Tato scéna bude sloužit jako základ pro váš prototyp.

## 3. Přidání základních objektů

Přidejte do scény základní objekty, jako jsou krychle, koule a válce, které můžete použít k reprezentaci různých prvků ve vaší hře.

## 4. Přidání mechanik a funkcí

Použijte skriptování a vestavěné nástroje Unity k přidání mechanik a funkcí do vašeho prototypu.

## 5. Testování a iterace

Jakmile máte základní prototyp, začněte ho testovat.

### Note

Získejte zpětnou vazbu od ostatních a na základě této zpětné vazby upravte a vylepšujte svůj prototyp.

## 6. Opakování procesu

Po provedení změn na svém prototypu ho znovu otestujte a pokračujte v tomto cyklu, dokud nejste spokojeni s výsledkem.

### Tip

Rychlé prototypování je iterativní proces.

# Pojmenování BEM

BEM = "Block Element Modifier"

Metodika pro pojmenování tříd v **HTML** a **CSS**.

## Note

Pomáhá udržet váš kód organizovaný a snadno pochopitelný, a to i pro ostatní vývojáře, kteří se na váš kód podívají.

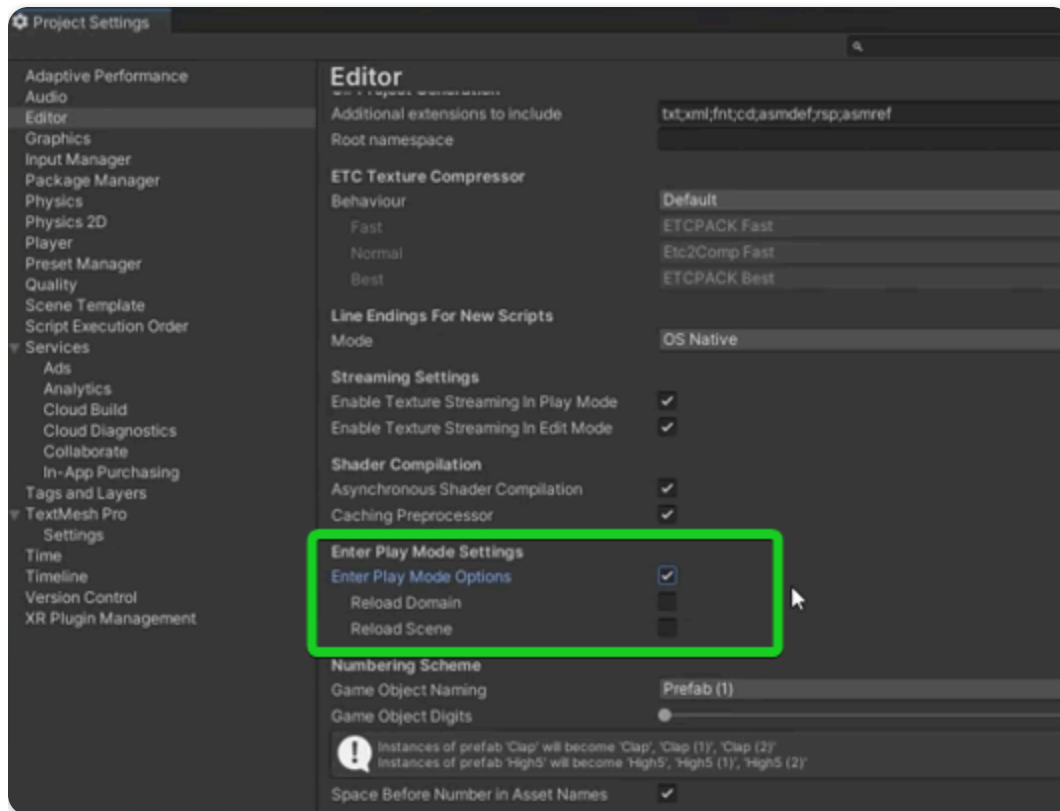
Příklad:

```
<div class="block"> <!-- Block -->
  <div class="block__element"> <!-- Element -->
  </div>
  <div class="block__element--modifier"> <!-- Element with modifier -->
  </div>
</div>
```

```
.block { ... }
.block__element { ... }
.block__element--modifier { ... }
```

- ▶ [Block](#)
- ▶ [Element](#)
- ▶ [Modifier](#)
- ▶ [Syntax BEM](#)
- ▶ [Použití v kódu](#)

# Rychlejší spuštění



- Reload Domain

Když je tato možnost povolena, všechny skripty se znovu načtou, což může trvat déle, ale zajišťuje, že se všechny změny v kódu projeví.

- Reload Scene

Když je tato možnost povolena, Unity znovu načte aktuální scénu, což může být užitečné, pokud chceš začít s "čistým" stavem.

Pokud tyto možnosti zakážeš, můžeš zrychlit vstup do režimu hry, protože Unity se vyhne některým časově náročným procesům.

## Výběr hry 2D či 3D

2D or 3D Unity Dev - What's better??



## Rychlé prototypování

Prototyping Games in Unity?



## Klíčová slova



2 Keywords I didn't fully understand when starting gamedev...



## Vývojové vzory

3 Game Programming Patterns WE ACTUALLY NEED.



# Microsoft SQL

## Kód

### Získat informace

---

- ▶ [Informace ze Serveru](#)
- ▶ [Informace z Tabulky](#)
- ▶ [Velikost Tabulek](#)
- ▶ [Informace o indexech na sloupcích](#)

### Hledat

---

- ▶ [Sloupec a zjistit v jaké Tabulce se nachází](#)
- ▶ [Datový typ Sloupce z Tabulky](#)
- ▶ [Hodnoty ve všech textových a číselných sloupcích databáze](#)
- ▶ [Nejnovější a Nejstarší záznam](#)
- ▶ [Nejčastěji se vyskytující hodnoty](#)
- ▶ [Port na kterém je spuštěn Server](#)

### Výkon

---

- ▶ [Efektivita dotazů](#)

### Konfigurace

---

- ▶ [Vzdálený přístup](#)

Izoluje aplikace se všemi jejími knihovnami, konfiguračními soubory a dalšími závislými soubory do kontejnerů.

### *i* Note

Kontejnery zajišťují, že aplikace mohou být spuštěny v jakémkoli prostředí.

Docker se stará o celý životní cyklus kontejnerů.

**Kontejner → Vytvoření → Spuštění → Zastavení**

### *i* Tip

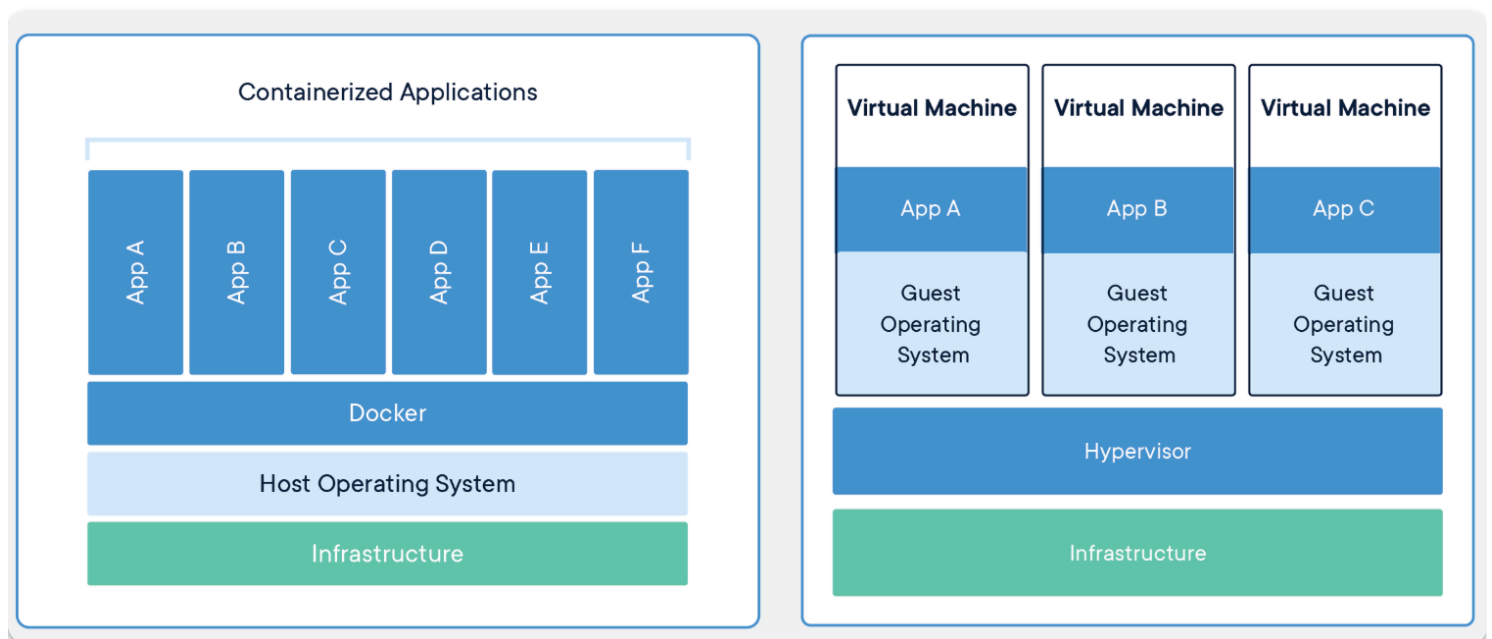
- Kontejnerizace je virtualizací jádra operačního systému.

Všechny kontejnery běží v rámci jednoho operačního systému a sdílejí paměť, knihovny a další zdroje.

- Zdroje se tímto způsobem využívají efektivněji než klasická virtualizace.

Spuštění kontejneru je navíc mnohem rychlejší než spuštění virtuálního stroje s instalací operačního systému.

- Malá režie a na stejném hardwaru můžete spustit více docker kontejnerů než virtuálních strojů



## Soubory dockeru

## **dockerd.exe a docker.exe**

- dockerd.exe:

Spouští Docker Daemon, což je hlavní služba, která spravuje kontejnery a poslouchá na socketu či TCP portu.

- docker.exe

Klientský nástroj, který posílá příkazy daemonu (např. `docker run`, `docker ps`).

## **docker-compose.exe a docker-compose.yml**

- docker-compose.exe:

Umožňuje definovat a spouštět více Docker kontejnerů jako součásti jedné aplikace.

Pomocí souboru `docker-compose.yml` můžete definovat všechny služby (kontejnery), které mají běžet, včetně jejich konfigurací, závislostí a propojení mezi nimi.

- docker-compose.yml:

Konfigurační soubor, který popisuje, jaké kontejnery (služby) mají být spuštěny, jaké obrazové soubory mají používat, jaké porty mají být mapovány a jaké další nastavení kontejnery potřebují.

Tento soubor je zpracován příkazem `docker-compose up`, který automaticky spustí všechny definované kontejnery.

## **Základní pojmy**

### **dockerfile**

Textový soubor s instrukcemi k vytvoření `Docker image`.

Specifikuje operační systém, na kterém bude běžet kontejner, jazyky, lokace, porty a další komponenty.

### **docker image**

Komprimovaná, samostatná část softwaru vytvořená příkazy v `Dockerfile`.

Je to "šablona" (aplikace plus požadované knihovny a binární soubory) potřebná k vytvoření a spuštění Docker kontejneru.

### **docker run**

Příkaz, který spouští kontejnery.

#### Note

Každý kontejner je instancí jednoho image.

## docker hub

Oficiální úložiště pro sdílení `docker image`.

#### Tip

Obsahuje oficiální `docker image` z open-source projektů a neoficiální od komunity.

Možnost pracovat i s lokálními docker úložišti.

## docker engine

Jádro softwaru docker.

Technologie na principu klient-server, která vytváří a provozuje kontejnery.

## docker compose

Definice ke spuštění více kontejnerů.

# Příkazy

## Stáhnout docker image

```
docker pull <Image name>
```

#### Note

Je název `docker image`.

Například: `mcr.microsoft.com/dotnet/core/sdk:3.1`

Umístění docker image po stažení:

- Linux:

```
/var/lib/docker/
```

- Windows:

```
C:\ProgramData\DockerDesktop
```

- macOS:

```
~/Library/Containers/com.docker.docker/Data/vms/0/
```

## Sestavení docker image

- `docker build [-t] customFolder`

Sestaví kontejner pro docker image ve vybraném adresáři.

### Note

`customFolder`

Název vybraného adresáře k sestavení docker image.

Může být například aktuální adresář: `.`, nebo jakkoli jinak.

`-t`

Pojmenování image a tagu. (Pokud není zadán parametr -t, použije se tag: `latest`)

- Příklad:

```
docker build -t myapp .
```

**i Tip**

`myapp`

Název pro nově sestavený kontejner. (Může být jakýkoli.)

.

Pracovní adresář v dockeru. (V tomto příkladu kořenový adresář.)

## Spuštění kontejneru z docker image

- `docker run <docker image>`

Spustí kontejner pro docker image.

```
docker run kitematic/hello-world-nginx
```

**i Tip**

Spustí docker kontejner s docker image: `kitematic/hello-world-nginx`

## Spustit na jiném portu

```
docker run -p 70:80 kitematic/hello-world-nginx
```

### Note

`-p`

Mapuje port 70 na hostitelském stroji na port 80 uvnitř kontejneru. (To znamená, že pokud aplikace uvnitř kontejneru poslouchá na portu 80, bude přístupná na portu 70 hostitelského stroje.)

`kitematic/hello-world-nginx`

Název docker image ke spuštění.

## Spustit v interaktivním módu

```
docker run -it kitematic/hello-world-nginx
```

### Note

Užitečné, pokud chcete spustit kontejner a poté v něm spustit další příkazy, například při ladění nebo vývoji.

## Odstranit po zastavení

```
docker run --rm kitematic/hello-world-nginx
```

### Note

`--rm`

Docker automaticky odstraní kontejner, když je běh kontejneru přerušen.

### Tip

Užitečné, pokud nechcete, aby se vaše lokální úložiště naplnilo zastavenými kontejnery.



## Spuštění více kontejnerů z docker image najednou

### Note

Musíte použít soubor YAML k definování služeb vaší aplikace.

Následně pomocí jediného příkazu `docker-compose up` můžete vytvořit a spustit všechny služby definované ve vašem souboru `docker-compose.yml`.

### Tip

Automaticky použije lokální `docker image`, pokud je k dispozici.

Příklad souboru `docker-compose.yml`:

```
# Verze Docker Compose souboru
version: '3.4'

# Definice služeb
services:
  # Název služby
  webapp:
    # Obraz, který se má použít pro tuto službu
    # Tento obraz je vzorová aplikace ASP.NET Core od Microsoftu
    image: mcr.microsoft.com/dotnet/core/samples:aspnetapp
    # Instrukce pro sestavení obrazu
    build:
      # Kontext pro sestavení, obvykle je to adresář obsahující Dockerfile
      context: .
      # Cesta k Dockerfile
      dockerfile: Dockerfile
    # Mapování portů mezi hostitelem a kontejnerem
    # Formát je "host:kontejner"
    # Toto nastavení říká Dockeru, aby přesměroval port 8000 na hostiteli na port 80
    # v kontejneru
    ports:
      - "8000:80"
```

## Dockerfile

V dockeru **není žádná výchozí složka**.

#### Tip

Když vytváříte Dockerfile, můžete nastavit pracovní adresář v kontejneru pomocí příkazu `WORKDIR`.

#### Tip

Pokud není nastaven `WORKDIR`, vztahuje se vše na kořenový adresář (/) kontejneru.

## Příklad pro .NET Core

```
# Používáme oficiální .NET Core runtime image z Docker Hub
# 'dotnet' je jméno image a '3.1' je tag, který specifikuje verzi
FROM mcr.microsoft.com/dotnet/core/runtime:3.1

# Nastavíme pracovní adresář v kontejneru na /app
# Pokud tento adresář neexistuje, docker ho vytvoří
WORKDIR /app

# Kopírujeme výstup buildu z našeho stroje do kontejneru
# 'publish' je cesta k výstupu buildu na našem stroji
# '.' znamená aktuální (pracovní) adresář v kontejneru
COPY ./publish .

# Nastavíme spustitelný soubor pro kontejner
# 'myapp.dll' je název naší aplikace
ENTRYPOINT ["dotnet", "myapp.dll"]
```

#### Note

Vytvoří `docker image` pro vaši aplikaci .NET Core.

Když spustíte kontejner z této image, vaše aplikace se automaticky spustí.

## Příklad pro C# Aplikaci

```
# Používáme oficiální .NET Core SDK image z Docker Hub
# 'dotnet' je jméno image a '3.1' je tag, který specifikuje verzi
```

```
FROM mcr.microsoft.com/dotnet/core/sdk:3.1

# Nastavíme pracovní adresář v kontejneru na /app
# Pokud tento adresář neexistuje, docker ho vytvoří
WORKDIR /app

# Kopírujeme všechny soubory z našeho stroje do kontejneru
# '.' znamená aktuální adresář na našem stroji
# '.' znamená aktuální (pracovní) adresář v kontejneru
COPY . .

# Spustíme příkaz 'dotnet restore', který stáhne všechny potřebné NuGet balíčky
RUN dotnet restore

# Spustíme příkaz 'dotnet publish', který vytvoří výstup buildu naší aplikace
RUN dotnet publish -c Release -o out

# Nastavíme spustitelný soubor pro kontejner
# 'myapp.dll' je název naší aplikace
ENTRYPOINT ["dotnet", "out/myapp.dll"]
```

#### Note

Tento Dockerfile vytvoří docker image pro vaši aplikaci C#.

Když spustíte kontejner z této image, vaše aplikace se automaticky spustí.

## Příklad .NET Core a lokálních NuGet balíčků

```
# Používáme oficiální .NET Core SDK image z Docker Hub
FROM mcr.microsoft.com/dotnet/core/sdk:3.1

# Nastavíme pracovní adresář v kontejneru na /app
WORKDIR /app

# Kopírujeme všechny soubory z našeho stroje do kontejneru
COPY . .

# Spustíme příkaz 'dotnet restore', který načte všechny potřebné NuGet balíčky z
lokálního úložiště
# Předpokládáme, že všechny potřebné NuGet balíčky jsou uloženy v adresáři 'nuget'
našeho projektu
RUN dotnet restore --source ./nuget
```

```
# Spustíme příkaz 'dotnet publish', který vytvoří výstup buildu naší aplikace
RUN dotnet publish -c Release -o out

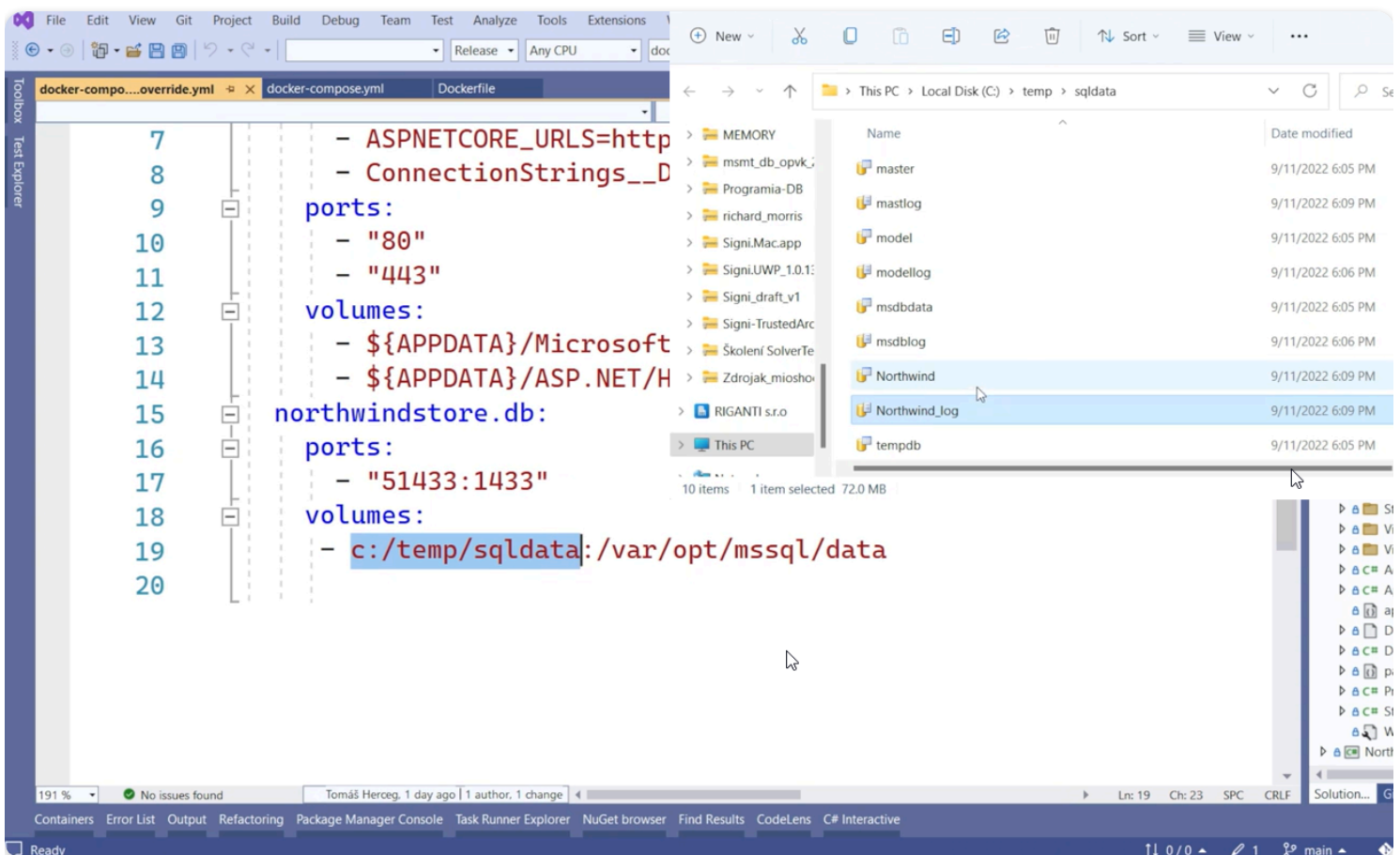
# Nastavíme spustitelný soubor pro kontejner
ENTRYPOINT ["dotnet", "out/myapp.dll"]
```

## Tip

V tomto příkladu předpokládáme, že všechny potřebné NuGet balíčky jsou uloženy v adresáři `nuget` vašeho projektu.

Příkaz `dotnet restore --source ./nuget` pak načte tyto balíčky z lokálního úložiště místo stahování z internetu.

## Zachování dat z kontejneru na lokálním disku



The screenshot displays the Visual Studio IDE with a Docker Compose configuration file open in the editor. The configuration defines two services: a web application and a database. The web application service is configured with environment variables for ASPNETCORE\_URLS and ConnectionStrings, and ports 80 and 443. The database service is configured with a volume mapping from the host to the container. The file explorer on the right shows the local disk path `C:\temp\sqldata`, which contains several folders and files, including `Northwind` and `Northwind_log`.

```
7 - ASPNETCORE_URLS=http
8 - ConnectionStrings__D
9
10 ports:
11 - "80"
12 - "443"
13
14 volumes:
15 - ${APPDATA}/Microsoft
16 - ${APPDATA}/ASP.NET/H
17
18 northwindstore.db:
19 ports:
20 - "51433:1433"
21
22 volumes:
23 - c:/temp/sqldata:/var/opt/mssql/data
```

## Použití mřížky (grid)

1. View -> Show Grid (zobrazí mřížku)
2. View -> Grid and Axis... (nastavení mřížky)

- [Výpočet obrázku pro zarovnání na střed](#)

## FAQ

- [Modrý čtverec uvnitř stránky](#)

# OBS (Open Broadcast Software)

► [Argumenty](#)

# SSH

SSH je bezpečnější než používání uživatelského jména a hesla, protože využívá veřejný a soukromý klíč.

► [Přípojení na GitHub](#)

# Zobrazení

- [Sloučení/Oddělení panelů kalendáře](#)



# Instalace

- ▶ [Řešení problému s neviditelným diskem při instalaci Windows](#)

## Základní nastavení

- ▶ [Zobrazení sekund v dolním panelu](#)

## Klávesnicové zkratky

- ▶ [Minimalizace/Maximalizace všech oken](#)
- ▶ [Skočení na adresní řádek](#)

## Chybějící klávesy na klávesnici

- ▶ [Kontextová klávesa](#)