Da bi ustvarila program v C# za prevajanje vsebine datoteke README.md v različne jezike z uporabo **Google Translate API**, lahko uporabiš naslednji postopek:

**1. Registracija in nastavitve Google Cloud API-ja**

Najprej moraš nastaviti Google Translate API na Google Cloud Platform (GCP). Sledi tem korakom:

**a) Ustvari račun na Google Cloud Platform (če ga še nimaš):**

* Obišči Google Cloud Console.
* Registriraj se ali se prijavi.

**b) Ustvari nov projekt:**

* V konzoli klikni na gumb "Select a project" in nato "New Project".
* Poimenuj projekt (npr. "Readme Translator").

**c) Omogoči Google Translate API:**

* Pojdi v sekcijo "APIs & Services" -> "Library".
* Poišči "Cloud Translation API" in ga omogoči.

**d) Ustvari API ključ:**

* Pojdi na "APIs & Services" -> "Credentials".
* Ustvari nov API ključ z možnostjo "Create credentials" -> "API Key".
* API ključ shrani, saj ga boš potrebovala za dostop do prevajanja.

**2. Namesti potrebne knjižnice v C#**

Za uporabo Google Translate API-ja v C# boš potrebovala paket Google.Cloud.Translation.V2. Sledi tem korakom:

**a) Namesti knjižnico z uporabo NuGet:**

V Visual Studio lahko to knjižnico namestiš prek NuGet Package Manager-ja:

bash

Kopiraj kodo

Install-Package Google.Cloud.Translation.V2

Ali v datoteki .csproj dodaj paket:

xml

Kopiraj kodo

<PackageReference Include="Google.Cloud.Translation.V2" Version="2.1.0" />

**3. Napiši C# program za prevod README.md**

Nato napiši program, ki bo prebral README.md datoteko, jo prevedel in shranil v nove datoteke za vsak jezik.

**a) Primer kode za prevajanje z uporabo Google Translate API-ja:**

csharp

Kopiraj kodo

using System;

using System.IO;

using Google.Cloud.Translation.V2;

class TranslateReadme

{

static void Main(string[] args)

{

// Pot do tvojega API ključa

string apiKey = "YOUR\_GOOGLE\_TRANSLATE\_API\_KEY";

// Ustvari Google Translate clienta

var client = TranslationClient.CreateFromApiKey(apiKey);

// Pot do README.md datoteke

string filePath = "README.md";

// Preberi vsebino README.md

string readmeContent = File.ReadAllText(filePath);

// Jeziki, v katere želiš prevesti

string[] targetLanguages = { "sl", "de", "fr", "es" }; // npr. Slovenščina, Nemščina, Francoščina, Španščina

foreach (var language in targetLanguages)

{

// Prevedi vsebino

var translation = client.TranslateText(readmeContent, language);

// Ustvari ime nove datoteke

string translatedFilePath = $"README.{language}.md";

// Shrani prevedeno vsebino v novo datoteko

File.WriteAllText(translatedFilePath, translation.TranslatedText);

Console.WriteLine($"Prevedeno v {language} in shranjeno v {translatedFilePath}");

}

Console.WriteLine("Prevajanje zaključeno.");

}

}

**b) Razlaga kode:**

* **API ključ**: Vstavi svoj API ključ za Google Translate v spremenljivko apiKey.
* **Preberi README.md**: Datoteka README.md se prebere kot niz.
* **Prevedi besedilo**: Zanka prevaja besedilo v vse jezike, definirane v targetLanguages (npr. sl za slovenščino, de za nemščino itd.).
* **Shrani prevedeno datoteko**: Vsak prevod se shrani v novo datoteko z ustreznim jezikovnim oznako (npr. README.sl.md za slovenščino).

**4. Poveži Git in avtomatiziraj postopek**

Da bo postopek bolj avtomatiziran, lahko ustvariš Git hook, ki bo program zagnal ob vsaki spremembi README.md. Sledi tem korakom:

**a) Ustvari Git hook za pre-commit:**

V .git/hooks/ mapi ustvari datoteko pre-commit:

bash

Kopiraj kodo

#!/bin/sh

# Git hook skripta za prevajanje README.md ob commitu

dotnet run --project pot\_do\_tvojega\_projekta

git add README.\*

**b) Omogoči Git hook:**

Naredi datoteko pre-commit izvedljivo:

bash

Kopiraj kodo

chmod +x .git/hooks/pre-commit

Sedaj se bo tvoj C# program samodejno zagnal pred vsakim commitom, prevedel datoteko in dodal nove README.\*.md datoteke v commit.

**5. Dodatni koraki**

* **Testiranje**: Testiraj program, da preveriš, če vse deluje pravilno.
* **Nadgradnja**: Če boš želela prevod bolj prilagoditi, lahko prilagodiš vhodno in izhodno obliko, ali pa dodaš več funkcij, kot so prilagajanje naslovov v prevodu.

S tem postopkom boš lahko enostavno prevajala svojo dokumentacijo v različne jezike in shranjevala rezultate kot ločene datoteke.

Za avtomatizacijo prevajanja dokumentacije z Gitom na lokalnem računalniku (in kasneje nalaganje sprememb na GitHub) lahko uporabiš **Git hooks**, kot sem omenila prej, vendar je tukaj bolj podroben opis postopka za tvojo konkretno situacijo, kjer imaš program za prevajanje Prevedi.exe, ki se nahaja na lokalnem računalniku na poti D:\Barbara\Prevedi.exe.

Bolj podrobno:

**1. Koncept avtomatizacije z Git hooks**

Git hooks so skripte, ki jih Git samodejno izvede ob določenih dogodkih v življenjskem ciklu Git repozitorija (npr. pred commitom, po commitu, pred potegom sprememb, itd.). Tukaj boš uporabila **pre-commit** hook, ki se zažene, preden izvedeš commit.

Torej bo Git pred vsakim commitom najprej zagnal tvoj program za prevajanje in nato izvedel commit, ki bo vključeval vse prevedene datoteke.

**2. Nastavitev Git hook za pre-commit**

**a) Pojdi v lokalni repozitorij**

Na svojem računalniku odpri mapo, kjer imaš kloniran GitHub repozitorij. Git hooks so lokalne datoteke, zato boš nastavila avtomatizacijo na svojem računalniku.

**b) Ustvari ali uredi pre-commit hook**

V mapi .git/hooks/ boš našla več vzorčnih hookov (npr. pre-commit.sample).

* Ustvari nov hook pre-commit (brez končnice) ali preimenuj datoteko pre-commit.sample v pre-commit.
* Odpri datoteko pre-commit in vanjo vpiši naslednjo skripto:

bash

Kopiraj kodo

#!/bin/sh

# Git hook skripta za prevajanje README.md

# Kliči Prevedi.exe z ustrezno potjo

"D:/Barbara/Prevedi.exe"

# Dodaj prevedene datoteke v commit

git add README.\*

**c) Shrani datoteko in jo naredi izvedljivo**

Ko je skripta vpisana, shrani datoteko in nato zaženi naslednji ukaz, da datoteko narediš izvedljivo:

bash

Kopiraj kodo

chmod +x .git/hooks/pre-commit

To bo omogočilo, da se skripta samodejno zažene ob vsakem commitu.

**3. Testiranje Git hooka**

Sedaj lahko testiraš delovanje Git hooka:

1. **Uredi README.md** datoteko v svojem repozitoriju.
2. V konzoli zaženi Git commit:

bash

Kopiraj kodo

git add README.md

git commit -m "Posodobljen README"

Ko zaženeš commit, bi moral Git najprej zagnati tvoj program Prevedi.exe. Ta bo prevedel README.md v druge jezike in ustvaril datoteke, kot so README.sl.md, README.de.md, itd. Nato bo skripta z ukazom git add dodala te prevedene datoteke v commit.

**4. Pogosta vprašanja in težave**

**a) Kako deluje Prevedi.exe?**

Git hook bo preprosto klical tvoj izvršljivi program (Prevedi.exe). Predpostavljam, da Prevedi.exe sam poskrbi za prevajanje datoteke README.md v različne jezike in ustrezno shranjevanje prevedenih datotek. Če program zahteva parametre, kot so vhodna datoteka ali ciljni jezik, jih lahko dodaš v skripto.

Primer, če Prevedi.exe potrebuje parameter za vhodno datoteko:

bash

Kopiraj kodo

"D:/Barbara/Prevedi.exe" "D:/Pot/Do/README.md"

Če pa potrebuješ dodatne parametre (npr. jezik), lahko to prilagodiš.

**b) Zagon na drugih računalnikih**

Pomembno je, da je pot do Prevedi.exe pravilno nastavljena za vsak računalnik. Če več ljudi sodeluje na projektu, morajo imeti enak ali podoben postopek in namestitev prevajalskega programa, ali pa najdeš način, da postane bolj univerzalen (npr. s relativnimi potmi ali paketnim sistemom, kot je Docker).

**5. Nalaganje sprememb na GitHub**

Ko je commit uspešno ustvarjen, lahko svoje spremembe potisneš (push) na GitHub kot običajno:

bash

Kopiraj kodo

git push origin main

Tako boš posodobil repozitorij na GitHubu, ki bo zdaj vseboval tudi prevedene README datoteke.

**6. Avtomatizacija prek CI/CD**

Če želiš proces povsem avtomatizirati na GitHubu, lahko uporabiš **GitHub Actions** za prevajanje in generiranje dokumentacije vsakič, ko posodobiš README.md. To zahteva bolj zapleteno konfiguracijo, kjer bi npr. Prevedi.exe ali prevajalski skript tekel v oblaku, ne na tvojem računalniku.

**7. Povzetek postopka**

* Nastaviš pre-commit Git hook, ki zažene tvoj lokalni program Prevedi.exe ob vsakem commitu.
* Prevedeni README se shranijo in dodajo v commit.
* Commit se lahko nato pošlje na GitHub.

Tako boš imela samodejno prevajanje lokalno urejeno in sinhronizirano z GitHubom!

Prevajanje s CI/CD

Seveda! Avtomatizacija prevajanja dokumentacije prek CI/CD (Continuous Integration / Continuous Deployment) pipeline-a, kot je **GitHub Actions**, omogoča, da prevajanje izvede strežnik, ko se README.md spremeni, namesto da to počneš lokalno na svojem računalniku. To pomeni, da prevajanje poteka v oblaku, ne glede na to, kdo posodobi datoteko.

Spodaj je podroben postopek, kako nastaviti **GitHub Actions**, da avtomatizira prevajanje dokumentacije z uporabo tvojega programa ali drugega prevajalskega orodja, ko se datoteka README.md posodobi.

**1. Koncept CI/CD avtomatizacije z GitHub Actions**

**GitHub Actions** je orodje za avtomatizacijo nalog, ki se zažene ob določenih dogodkih v repozitoriju (npr. ob push, pull request, ali ob spremembi datotek). Tu bomo ustvarili GitHub Action, ki bo ob vsaki spremembi README.md zagnal prevajalski program in prevedel dokumentacijo v druge jezike.

**2. Priprava GitHub Actions workflowa**

**a) Struktura repozitorija**

Najprej se prepričaj, da tvoj repozitorij vsebuje mapo .github/workflows. To je mesto, kjer shranjuješ vse GitHub Actions konfiguracijske datoteke.

**b) Ustvari GitHub Action YAML datoteko**

V mapi .github/workflows ustvari novo datoteko, na primer translate.yml. V tej datoteki boš definirala workflow, ki bo GitHubu povedal, kaj naj naredi, ko zazna spremembo v README.md.

**c) Primer translate.yml datoteke**

Spodaj je primer GitHub Actions konfiguracije, ki avtomatizira prevajanje README.md ob vsaki spremembi datoteke.

yaml

Kopiraj kodo

name: Translate README

on:

push:

paths:

- 'README.md' # Sproži akcijo ob spremembi README.md

jobs:

translate:

runs-on: ubuntu-latest

steps:

- name: Checkout repository

uses: actions/checkout@v2

- name: Install .NET Core SDK

uses: actions/setup-dotnet@v3

with:

dotnet-version: '6.0.x' # Uporabi različico .NET Core, ki ustreza tvojemu prevajalskemu programu

- name: Install dependencies

run: dotnet restore # Če tvoj projekt uporablja NuGet pakete, jih bo to obnovilo

- name: Run translation program

run: |

# Tu lahko prilagodiš klic svojega prevajalskega programa

# Predpostavljamo, da imaš C# aplikacijo, ki prevaja README.md

dotnet run --project D:/Barbara/Prevedi.exe README.md

- name: Commit translated files

run: |

git config --global user.name 'github-actions'

git config --global user.email 'actions@github.com'

git add README.\*.md

git commit -m "Automated translation of README.md"

git push

**d) Razlaga YAML datoteke:**

* **name: Translate README**: Določa ime workflowa.
* **on: push**: Akcija se sproži ob push dogodku v repozitoriju, vendar samo, če se spremeni datoteka README.md.
* **jobs: translate**: Definira delovne naloge, ki jih GitHub Actions izvaja.
  + **runs-on: ubuntu-latest**: Določa, da se workflow izvaja na najnovejši različici Ubuntu Linuxa.
  + **actions/checkout@v2**: Prenese kodo iz tvojega repozitorija.
  + **actions/setup-dotnet@v3**: Namesti .NET SDK, da lahko poganjaš svoj C# program. Tu določiš, katero različico .NET uporabljaš.
  + **dotnet restore**: Obnovi vse NuGet odvisnosti tvojega projekta (če so potrebne).
  + **dotnet run --project ...**: Ta vrstica izvede tvoj prevajalski program. Lahko jo prilagodiš, če uporabljaš drugačno orodje ali različne poti.
  + **git config**: Nastavi globalne Git konfiguracije za commite, ki jih naredi akcija.
  + **git add, git commit, git push**: Po prevajanju avtomatsko doda prevedene datoteke v repozitorij in jih pošlje (push) na GitHub.

**3. Kako GitHub Actions deluje**

Ko je translate.yml datoteka dodana v repozitorij, se GitHub Actions samodejno zažene, ko nekdo izvede push, ki vključuje spremembo datoteke README.md. Akcija bo:

1. Prenesla najnovejšo različico tvoje kode (checkout).
2. Namestila potrebne odvisnosti (v tem primeru .NET SDK).
3. Zagnala tvoj prevajalski program (v našem primeru Prevedi.exe).
4. Prevedla README.md v druge jezike.
5. Naredila commit in push prevedenih datotek nazaj v GitHub repozitorij.

**4. Prilagajanje prevajalskega programa**

Če tvoj prevajalski program ne deluje na Linuxu (kar je pogosto, če uporabljaš exe datoteke), ga lahko:

* **Konvertiraš v .NET aplikacijo**, ki deluje na vseh platformah (Linux, Windows, macOS), če že uporabljaš .NET.
* **Prevedeš ali zgradiš svoj program kot križno platformsko orodje**, kar bo omogočilo delovanje na GitHub strežnikih.
* **Uporabiš drugo rešitev za prevajanje**, kot je Python skripta, ki uporablja Google Translate API ali katero koli knjižnico, ki deluje na vseh platformah.

**5. Shranjevanje API ključev in varnostnih informacij**

Če uporabljaš Google Translate API, boš potrebovala API ključ. Ključa ne smeš shranjevati neposredno v repozitoriju zaradi varnostnih razlogov. Uporabi GitHub Secrets, da varno shraniš API ključ:

**a) Dodaj API ključ v GitHub Secrets**

1. Pojdi v GitHub repozitorij.
2. Klikni na **Settings** > **Secrets** > **Actions**.
3. Klikni **New repository secret**.
4. Poimenuj ga (npr. GOOGLE\_TRANSLATE\_API\_KEY) in prilepi vrednost API ključa.

**b) Uporaba API ključa v workflowu**

V YAML datoteki uporabi API ključ iz GitHub Secrets:

yaml

Kopiraj kodo

- name: Run translation program

run: |

dotnet run --project D:/Barbara/Prevedi.exe README.md --api-key ${{ secrets.GOOGLE\_TRANSLATE\_API\_KEY }}

S tem zagotoviš, da je API ključ varen in ga uporabiš le v okviru GitHub Actions.

**6. Prednosti CI/CD avtomatizacije z GitHub Actions**

* **Avtomatski prevodi**: Nič več ročnega zaganjanja prevajalskega programa na tvojem računalniku. GitHub Actions vse naredi v ozadju.
* **Konsistentnost**: Vsi sodelavci bodo videli enake prevedene datoteke, saj se vse posodobi v repozitoriju.
* **Prihranek časa**: Prihranila boš veliko časa, saj bo prevajanje in nalaganje prevedenih datotek popolnoma avtomatizirano.

**Povzetek**

* Ustvari GitHub Action, ki se sproži ob spremembi README.md.
* Action samodejno zažene tvoj prevajalski program, doda prevedene datoteke in jih pošlje nazaj v repozitorij.
* Varno shrani API ključe z GitHub Secrets.

S tem bo tvoja CI/CD pipeline za prevajanje README.md popolnoma avtomatizirana in bo delovala neposredno na GitHubu!

Obstajata dve glavni možnosti:

**Možnost 1: Prevedi program za več platform (cross-platform)**

Ker je tvoj program napisan v C#, lahko prevedeš svojo aplikacijo za več platform, kot so Linux, Windows ali macOS. GitHub Actions se privzeto izvaja na Linuxu, zato bomo poskrbeli, da tvoj C# program deluje na Linuxu.

**Koraki:**

1. **Prevedi aplikacijo za Linux**: Uporabiš lahko .NET, da ustvariš samostojno aplikacijo, ki deluje na več platformah. Tukaj so koraki, kako to storiš:

a) **Odpri projekt v Visual Studio ali terminalu**.

b) **Prevedi aplikacijo za Linux (ali druge platforme)**. Uporabi naslednji ukaz v terminalu:

bash

Kopiraj kodo

dotnet publish -c Release -r linux-x64 --self-contained

Ta ukaz:

* + **-c Release**: Ustvari aplikacijo v načinu "Release" (optimizirana za produkcijo).
  + **-r linux-x64**: Cilja na Linux z 64-bitno arhitekturo.
  + **--self-contained**: Ustvari samostojno aplikacijo, ki vključuje vse potrebne odvisnosti .NET.

To bo ustvarilo izvršljivo datoteko za Linux (Prevedi brez končnice), ki jo lahko zaženeš na GitHub Actions, ki teče na Linuxu.

1. **Naloži prevedeni program na GitHub**: Prevedeno datoteko (npr. Prevedi) naloži v svoj GitHub repozitorij, morda v mapo tools ali bin.
2. **Posodobi GitHub Actions YAML datoteko**: Zdaj, ko imaš izvršljivo datoteko za Linux, posodobi svojo GitHub Actions konfiguracijo, da bo klicala to datoteko. Spodaj je primer, kako bi lahko izgledala YAML datoteka:

yaml

Kopiraj kodo

name: Translate README

on:

push:

paths:

- 'README.md'

jobs:

translate:

runs-on: ubuntu-latest

steps:

- name: Checkout repository

uses: actions/checkout@v2

- name: Install .NET Core SDK

uses: actions/setup-dotnet@v3

with:

dotnet-version: '6.0.x'

- name: Run translation program

run: |

chmod +x ./tools/Prevedi # Dovoli izvrševanje programa

./tools/Prevedi README.md # Zaženi prevedeno aplikacijo za Linux

- name: Commit translated files

run: |

git config --global user.name 'github-actions'

git config --global user.email 'actions@github.com'

git add README.\*.md

git commit -m "Automated translation of README.md"

git push

Tu smo:

* + Namesto .exe datoteke, uporabili novo prevedeno datoteko za Linux.
  + Dovolili izvrševanje (chmod +x) in nato zagnali aplikacijo.

**Možnost 2: Uporaba Dockerja**

Če ne želiš prilagoditi svojega programa za Linux, lahko ustvariš **Docker image**, ki vsebuje tvoj program in vse odvisnosti. Nato lahko GitHub Actions zažene ta Docker image, kjer se bo izvršila tvoja .exe datoteka.

**Koraki:**

1. **Ustvari Dockerfile**: V korenu svojega repozitorija ustvari datoteko Dockerfile. Ta datoteka bo vsebovala navodila za ustvarjanje Docker image-a, ki lahko zažene tvoj program.

Primer Dockerfile za poganjanje tvoje .exe datoteke na Windows:

dockerfile

Kopiraj kodo

FROM mcr.microsoft.com/dotnet/runtime:6.0 AS base

WORKDIR /app

# Kopiraj Prevedi.exe v Docker image

COPY ./Prevedi.exe .

# Kopiraj vse ostale potrebne datoteke (npr. README.md)

COPY ./README.md .

# Zaženi Prevedi.exe ob zagonu kontejnerja

CMD ["./Prevedi.exe", "README.md"]

1. **Zgradi in zaženi Docker image lokalno**: Preveri, če Docker image deluje, tako da v svojem računalniku zaženeš:

bash

Kopiraj kodo

docker build -t prevedi-app .

docker run prevedi-app

S tem boš preverila, ali Docker image pravilno zažene tvoj program.

1. **Uporabi Docker v GitHub Actions**: Nato uporabi Docker image v GitHub Actions. Spodaj je primer YAML datoteke, ki uporablja Docker image:

yaml

Kopiraj kodo

name: Translate README

on:

push:

paths:

- 'README.md'

jobs:

translate:

runs-on: ubuntu-latest

steps:

- name: Checkout repository

uses: actions/checkout@v2

- name: Build and run Docker container

run: |

docker build -t prevedi-app .

docker run prevedi-app

- name: Commit translated files

run: |

git config --global user.name 'github-actions'

git config --global user.email 'actions@github.com'

git add README.\*.md

git commit -m "Automated translation of README.md"

git push

S tem Docker pristopom lahko ohraniš .exe datoteko in njeno specifično okolje, ne da bi spreminjala kodo za Linux. Docker bo poskrbel, da tvoj program teče v ustreznem okolju.

**Povzetek:**

* **Prevedba za več platform (cross-platform .NET)** omogoča, da tvoj C# program deluje tudi na Linuxu in tako brez težav poganja na GitHub Actions.
* **Uporaba Dockerja** ti omogoča, da ohraniš svoj .exe program in ga poganjaš v prilagojenem okolju znotraj GitHub Actions.

Odvisno od tega, katera možnost ti bolj ustreza, lahko izbereš tisto, ki najbolje deluje za tvoj projekt!

### 1. ****Ustvarjanje Dockerfile****

#### a) ****Ustvari novo datoteko v svojem projektu****

Dockerfile mora biti v mapi tvojega projekta, saj bo vseboval navodila za izdelavo Docker image-a iz tvoje obstoječe aplikacije.

##### Kako:

1. V raziskovalcu datotek (ali v terminalu) se premakni do mape, kjer imaš projekt. To je lahko ista mapa, kjer imaš .exe datoteko ali izvorno kodo.
2. Ustvari novo datoteko z imenom **Dockerfile** (brez končnice!).

#### b) ****Primer vsebine Dockerfile****

Če želiš, da Docker image vsebuje tvoj .exe program in vse odvisnosti, bo datoteka izgledala nekako takole:

dockerfile

Kopiraj kodo

# Izberi osnovno sliko .NET Runtime, ki podpira poganjanje .exe datotek

FROM mcr.microsoft.com/dotnet/runtime:6.0 AS base

# Nastavi delovno mapo znotraj kontejnerja

WORKDIR /app

# Kopiraj lokalno .exe datoteko v Docker image

COPY ./Prevedi.exe .

# Kopiraj README.md (ali druge datoteke, ki jih program potrebuje)

COPY ./README.md .

# Zaženi .exe datoteko, ko kontejner zaženeš

CMD ["./Prevedi.exe", "README.md"]

V tem primeru:

* **FROM** določa, da uporabljamo uradno .NET Runtime sliko (to je okolje, kjer lahko poganjaš .NET aplikacije in .exe datoteke).
* **WORKDIR** nastavi delovno mapo znotraj kontejnerja.
* **COPY** kopira tvoje lokalne datoteke (npr. Prevedi.exe in README.md) v kontejner.
* **CMD** določa ukaz, ki ga Docker zažene, ko se kontejner zažene (v tem primeru poganja tvoj prevajalski program).

### 2. ****Kako zgraditi Docker image****

Ko imaš Dockerfile v mapi svojega projekta, je naslednji korak ustvariti Docker image.

#### a) ****Odpri terminal ali ukazno vrstico****

* Na **Windows** lahko uporabiš **PowerShell** ali **Command Prompt**.
* Na **macOS** ali **Linux** odpri **Terminal**.

Prepričaj se, da si v isti mapi, kjer je tvoj Dockerfile. Do mape se lahko premakneš z ukazom:

bash

Kopiraj kodo

cd pot/do/tvoje/mape

#### b) ****Zgradi Docker image****

V terminalu zaženi naslednji ukaz za gradnjo Docker image-a:

bash

Kopiraj kodo

docker build -t prevedi-app .

* **-t prevedi-app**: To določa ime Docker image-a (v tem primeru prevedi-app).
* **.**: Pika pomeni, da Docker najde Dockerfile v trenutni mapi.

Docker bo prebral tvoj Dockerfile, skopiral tvoje datoteke in ustvaril Docker image. Ko je postopek zaključen, boš imela Docker image s tvojim programom, pripravljen za zagon.

### 3. ****Kako zaženem Docker container****

Ko imaš zgrajen Docker image, lahko zaženeš svoj program znotraj Docker kontejnerja.

V terminalu zaženi naslednji ukaz:

bash

Kopiraj kodo

docker run prevedi-app

Ta ukaz ustvari in zažene kontejner, ki temelji na tvojem Docker image-u prevedi-app, ter izvede tvoj program (Prevedi.exe).

### 4. ****Preverjanje ali vse deluje****

Docker bo zagnal kontejner in izvedel tvoj prevajalski program. Preveri izhod v terminalu in preveri, ali je tvoj program uspešno prevedel README.md.

### 5. ****Uporaba Docker image-a na GitHub Actions****

Ko si preverila, da tvoj Docker image deluje lokalno, lahko to rešitev uporabiš tudi v GitHub Actions.

#### a) ****Posodobi GitHub Actions YAML****

V tvoji YAML datoteki za GitHub Actions lahko zgradiš in zaženeš Docker container:

yaml

Kopiraj kodo

name: Translate README

on:

push:

paths:

- 'README.md'

jobs:

translate:

runs-on: ubuntu-latest

steps:

- name: Checkout repository

uses: actions/checkout@v2

- name: Build Docker container

run: docker build -t prevedi-app .

- name: Run translation program in Docker

run: docker run prevedi-app

- name: Commit translated files

run: |

git config --global user.name 'github-actions'

git config --global user.email 'actions@github.com'

git add README.\*.md

git commit -m "Automated translation of README.md"

git push

Ta GitHub Action bo:

1. Zgradil Docker image iz tvojega Dockerfile.
2. Zažene Docker container, ki bo izvedel tvoj prevajalski program.
3. Prevedene datoteke bo naložil nazaj v repozitorij.

### 6. ****Nalaganje Docker image-a (če je potrebno)****

Če bo tvoj Docker image prevelik ali pa potrebuješ ponovno uporabo, lahko ustvariš račun na **Docker Hub** ali uporabljaš GitHub Container Registry, kamor naložiš svoj image, da ga lahko pozneje preprosto preneseš in uporabiš. Za zdaj pa bo lokalno ustvarjanje image-a dovolj za testiranje.

### Povzetek

* Ustvari Dockerfile v korenski mapi tvojega projekta.
* Zgradi Docker image z docker build.
* Zaženi image lokalno z docker run.
* Ko je vse pripravljeno, lahko uporabiš Docker z GitHub Actions za avtomatizacijo.

Če želiš, da se datoteka a1.txt, ki jo ustvariš znotraj kontejnerja, shrani tudi na tvoj lokalni računalnik, moraš uporabiti **"volumes"**. Z volumni lahko mapiraš lokalno mapo na mapo v kontejnerju, kar omogoča shranjevanje in deljenje datotek med kontejnerjem in tvojim računalnikom.

**Koraki za pridobitev datoteke iz Docker kontejnerja**

**1. Uporabi Docker volume za mapiranje lokalne mape**

Ko zaženeš Docker container, lahko dodaš možnost -v, s katero poveš Dockerju, da mapira lokalno mapo v mapo v kontejnerju. Na primer, če želiš shraniti datoteko a1.txt iz kontejnerja v lokalno mapo, uporabiš naslednji ukaz:

bash

Kopiraj kodo

docker run -v D:/Challenger/izhod:/app/izhod prevedi-app

Kaj ta ukaz pomeni:

* **-v D:/Challenger/izhod:/app/izhod**: Tukaj mapiraš lokalno mapo D:/Challenger/izhod na mapo /app/izhod znotraj kontejnerja. Vse, kar program zapiše v /app/izhod, bo na voljo v lokalni mapi D:/Challenger/izhod.
* **prevedi-app**: To je ime tvojega Docker image-a, ki ga poganjaš.

**2. Posodobi program, da shrani datoteko v pravilno mapo**

V tvojem C# programu moraš zagotoviti, da datoteka a1.txt nastane v mapi, ki je mapirana z Dockerjem. Če uporabljaš mapo /app/izhod v kontejnerju, poskrbi, da program shrani datoteko v to mapo.

Na primer, v tvojem C# programu uporabi naslednjo pot:

csharp

Kopiraj kodo

string filePath = "/app/izhod/a1.txt";

File.WriteAllText(filePath, "Testna vsebina datoteke");

Ko program teče znotraj Docker kontejnerja, bo datoteka a1.txt shranjena v mapo /app/izhod, kar pomeni, da jo boš našla tudi v lokalni mapi D:/Challenger/izhod.

**3. Preveri lokalno mapo**

Po zagonu Docker kontejnerja preveri mapo D:/Challenger/izhod na svojem računalniku. Tam bi morala najti datoteko a1.txt, ki jo je tvoj program ustvaril.

**Povzetek**

1. Uporabi opcijo -v, da povežeš mapo znotraj Docker kontejnerja z lokalno mapo.
2. Posodobi program, da shrani datoteke v mapirano mapo.
3. Po zagonu Docker containerja najdi datoteko v lokalni mapi.

dotnet publish -c Release