Zürcher Hochschule
für Angewandte Wissenschaften

School of
Management and Law

AACSB ACCREDITED
Seit 2015

DevOps
Deployment mit Docker

Building Competence. Crossing Borders.

Adrian Moser
mosa@zhaw.ch, FS2024

Dockerfile

Wie entstehen die Images auf Docker Hub?

Dockerfile

Die Lösung sind **Dockerfiles**

Tutorial Dockerfile

Minimale Web-Applikation mit NodeJS

Idee: Minimale Webapp mit NodeJS und Express

Tutorial

package.json

```
{
  "name": "docker_web_app",
  "version": "1.0.0",
  "description": "Node.js on Docker",
  "author": "First Last <first.last@example.com>",
  "main": "server.js",
  "scripts": {
    "start": "node server.js"
  },
  "dependencies": {
    "express": "4.18.2"
  }
}
```

server.js

```
'use strict';

const express = require('express');

// Constants
const PORT = 8080;
const HOST = '0.0.0.0';

// App
const app = express();
app.get('/', (req, res) => {
    res.send('Hello FS2024 DevOps Course!!!');
});

app.listen(PORT, HOST);
console.log(`Running on http://${HOST}:${PORT}`);
```

Ein Hello-World REST-Service (analog Spring Boot)

Was brauchen wir für unser Image?

Tutorial

Image

- Betriebssystem
- NodeJS
- package.json und server.js

Hierarchie

Dockerfiles bauen aufeinander auf,
wir starten mit **node:20.10.0**

Dockerfile

```
FROM node:20.10.0

# Create app directory
WORKDIR /usr/src/app

# Copy app
COPY . .

# Install
RUN npm install

# Docker Run Command
EXPOSE 8080
CMD [ "node", "server.js" ]
```

Basis-Image

Files kopieren

Installieren

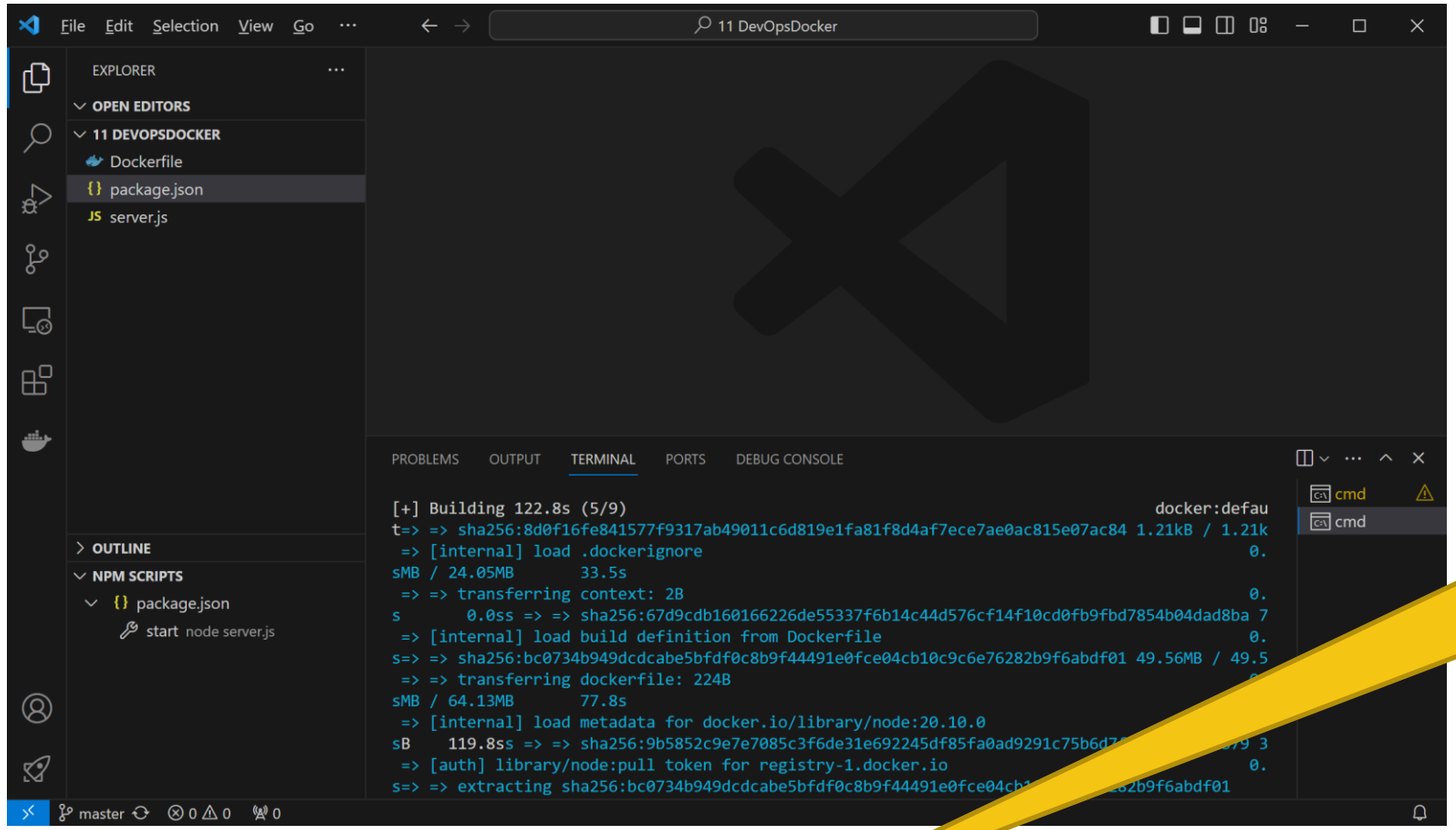
Port-Freigabe

CMD wird bei docker run ausgeführt

Dockerfile zusammenbauen

Tutorial

Terminal öffnen



Befehl in Terminal/Konsole im Verzeichnis ausführen (Punkt = aktuelles Verzeichnis)

Build-Command

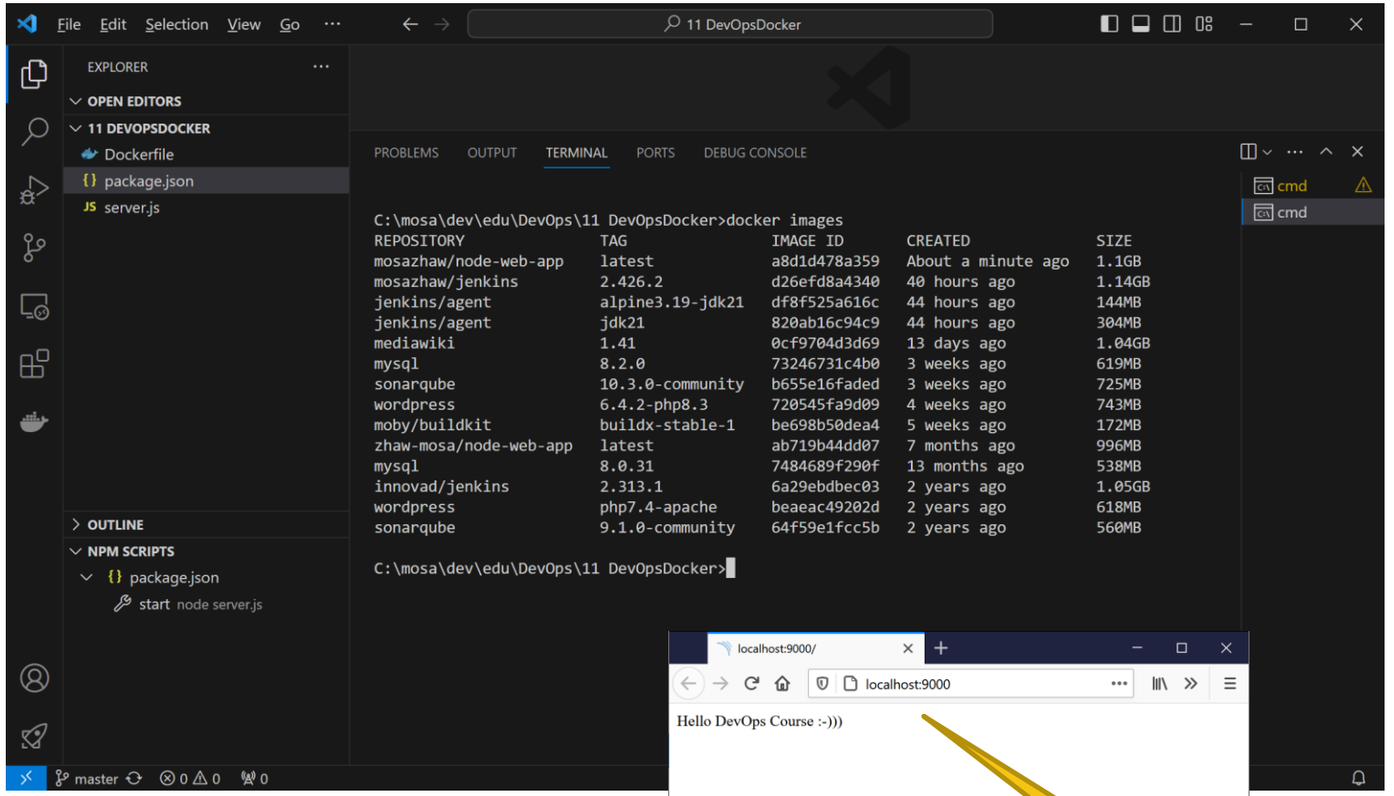
docker build -t mosazhaw/node-web-app .

Docker Container starten

Tutorial

Images auflisten

docker images



Port 9000 ist von aussen erreichbar, 8080 wurde freigegeben

Image starten

docker run -p 9000:8080 --name expressapp -d mosazhaw/node-web-app

... und im Browser testen.

Port 9000

Fazit Dockerfile

Vorgehen

– Dockerfiles werden aus anderen Dockerfiles erstellt (FROM)
– Weitere Files und Software hinzufügen (COPY)
– Befehle ausführen (RUN), z.B. Installation
– Es können Befehle hinterlegt werden, welche beim Starten ausgeführt werden (CMD)

Dockerfile

- Dockerfiles sind Scripts, welche auch z.B. in GitHub hinterlegt werden können
- Jenkins kann das zusammenbauen automatisieren

Jenkins mit Dockerfile

Ziel

Aus Git-Repository ein lauffähiges Docker-Image bauen.

Jenkins Job mit Build-Steps

- Neuer Freestyle-Job **DevOpsDockerBuild** (bekannt)
- Git Repo auschecken (bekannt)
- Webapp builden (bekannt)
- Docker File erstellen
- Docker File starten

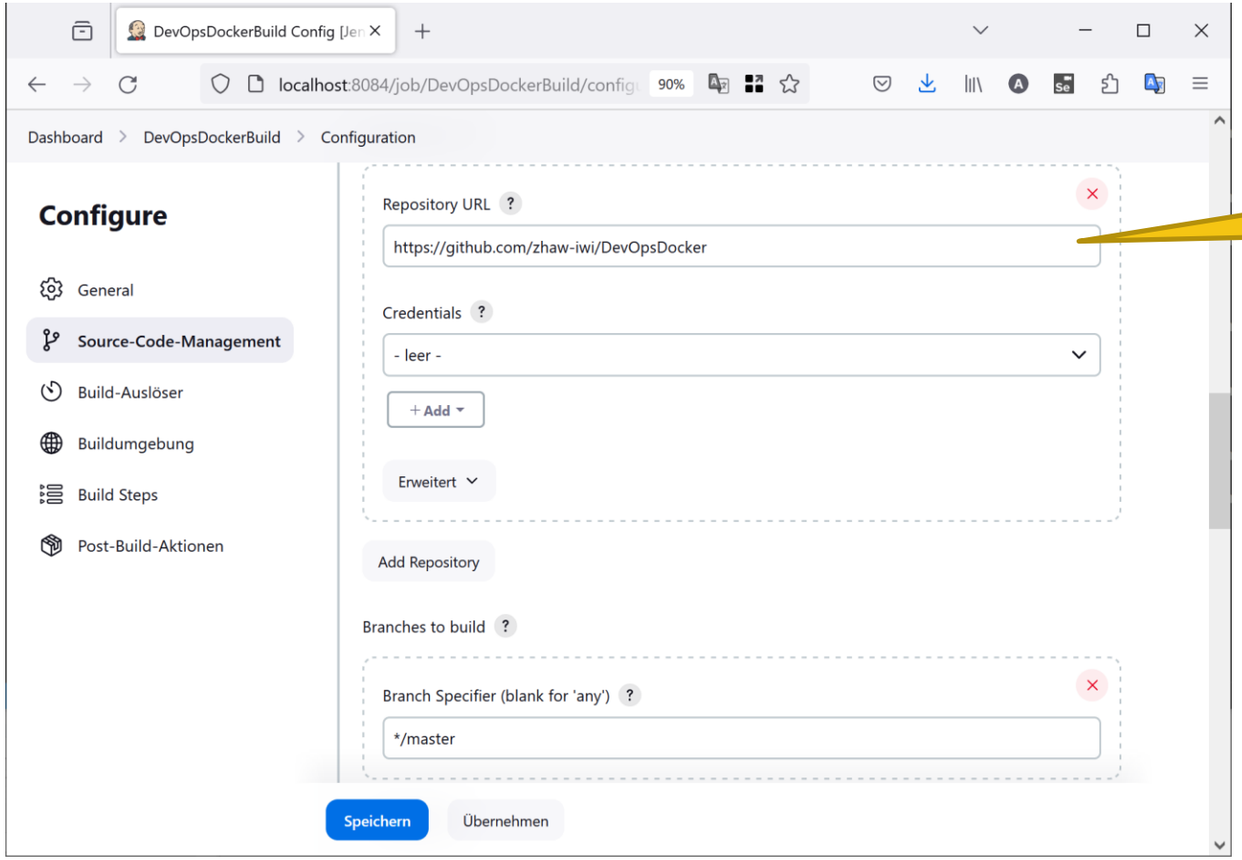
Herausforderungen

- Node innerhalb Jenkins verfügbar machen (bekannt)
- Docker innerhalb Jenkins nutzen

Neuer Freestyle-Job, Git Repo auschecken

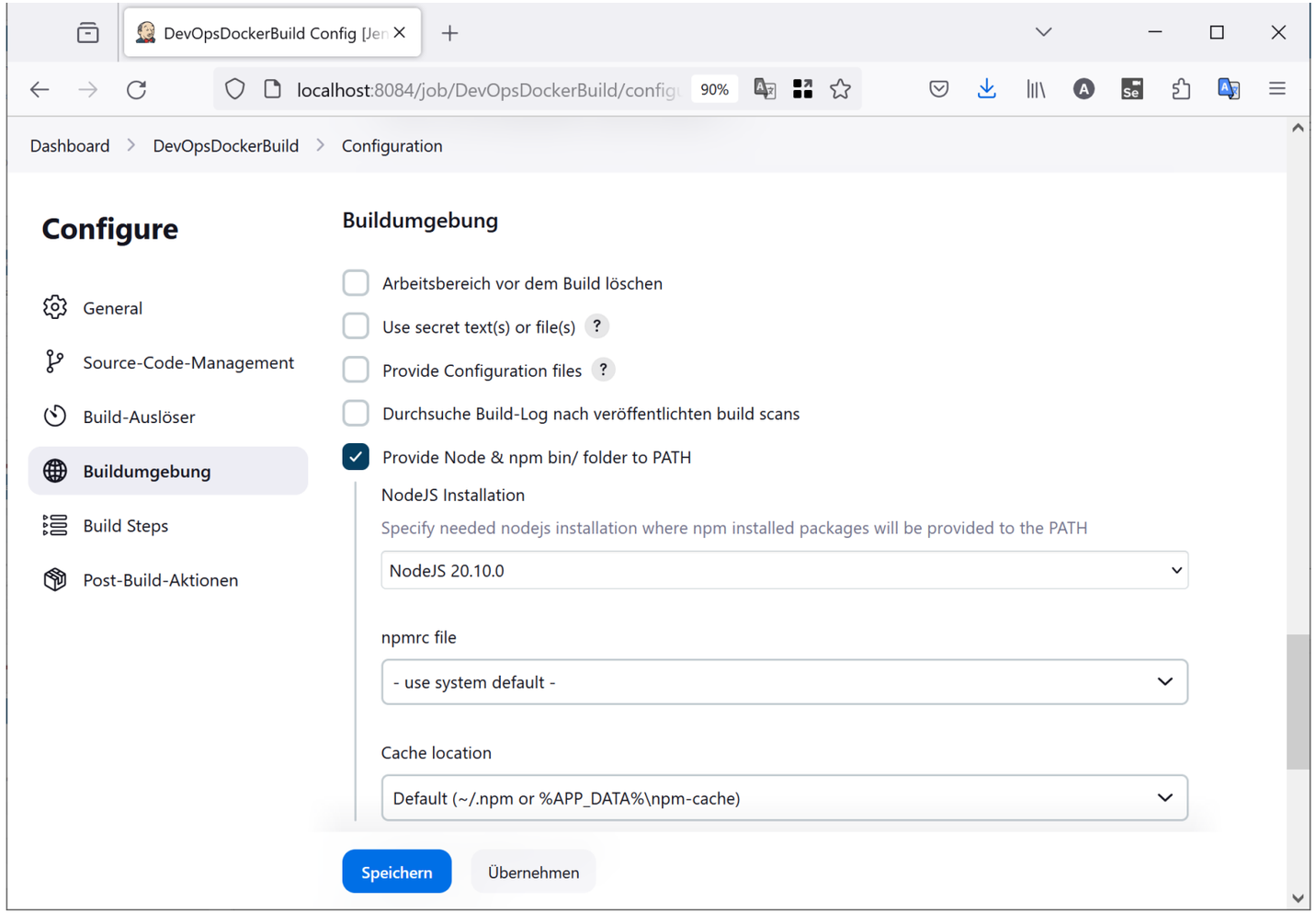
Tutorial

Repository URL: https://github.com/zhaw-iwi/DevOpsDocker



Buildumgebung: NPM zur Verfügung stellen

Tutorial



Buildverfahren «Shell ausführen»: npm install

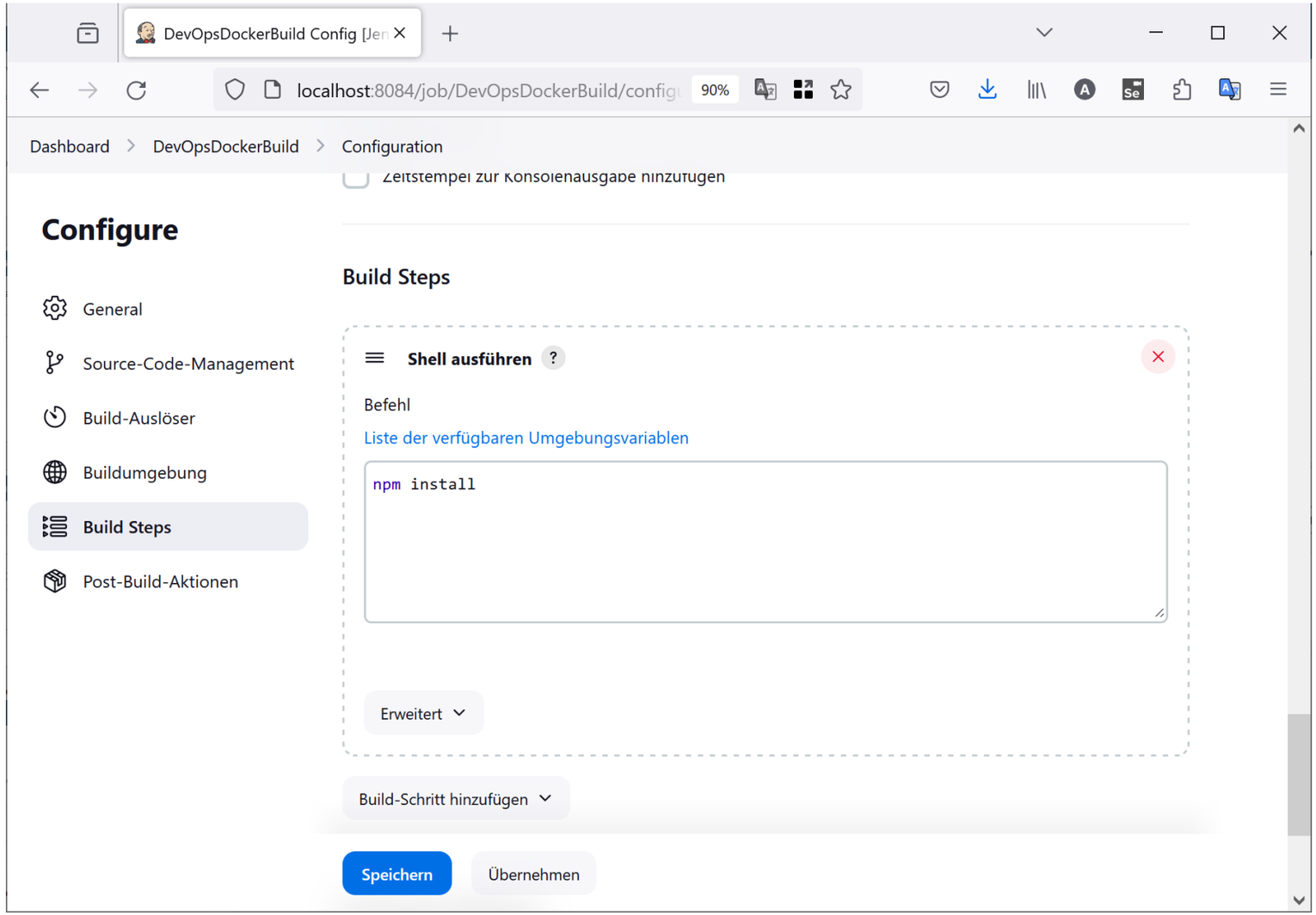
Tutorial

install

npm install

build

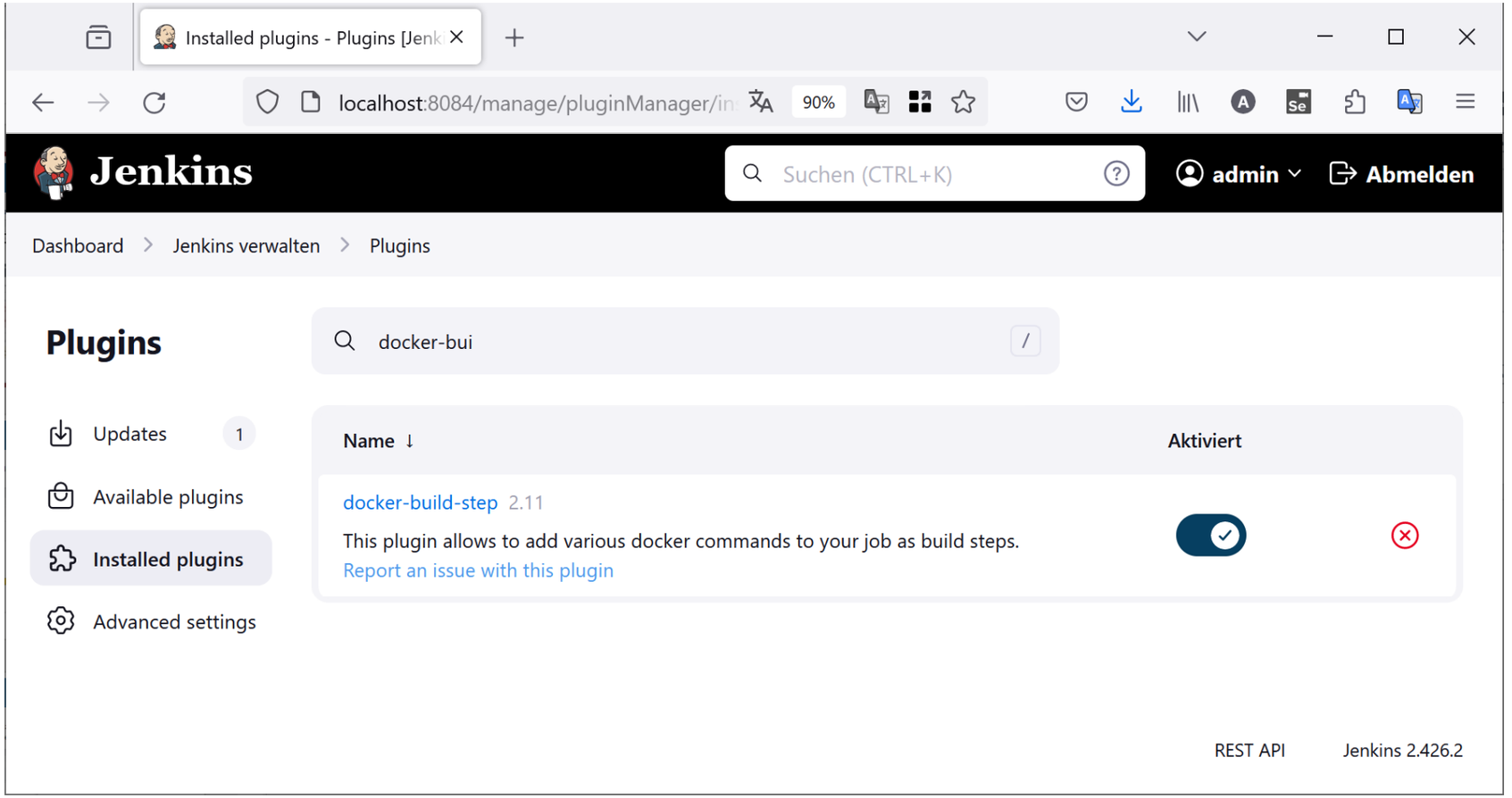
Da es sich um ein sehr einfaches JavaScript-Projekt handelt, ist ein «npm run build» hier nicht notwendig bzw. möglich.



Docker als Build-Step ausführen

Tutorial

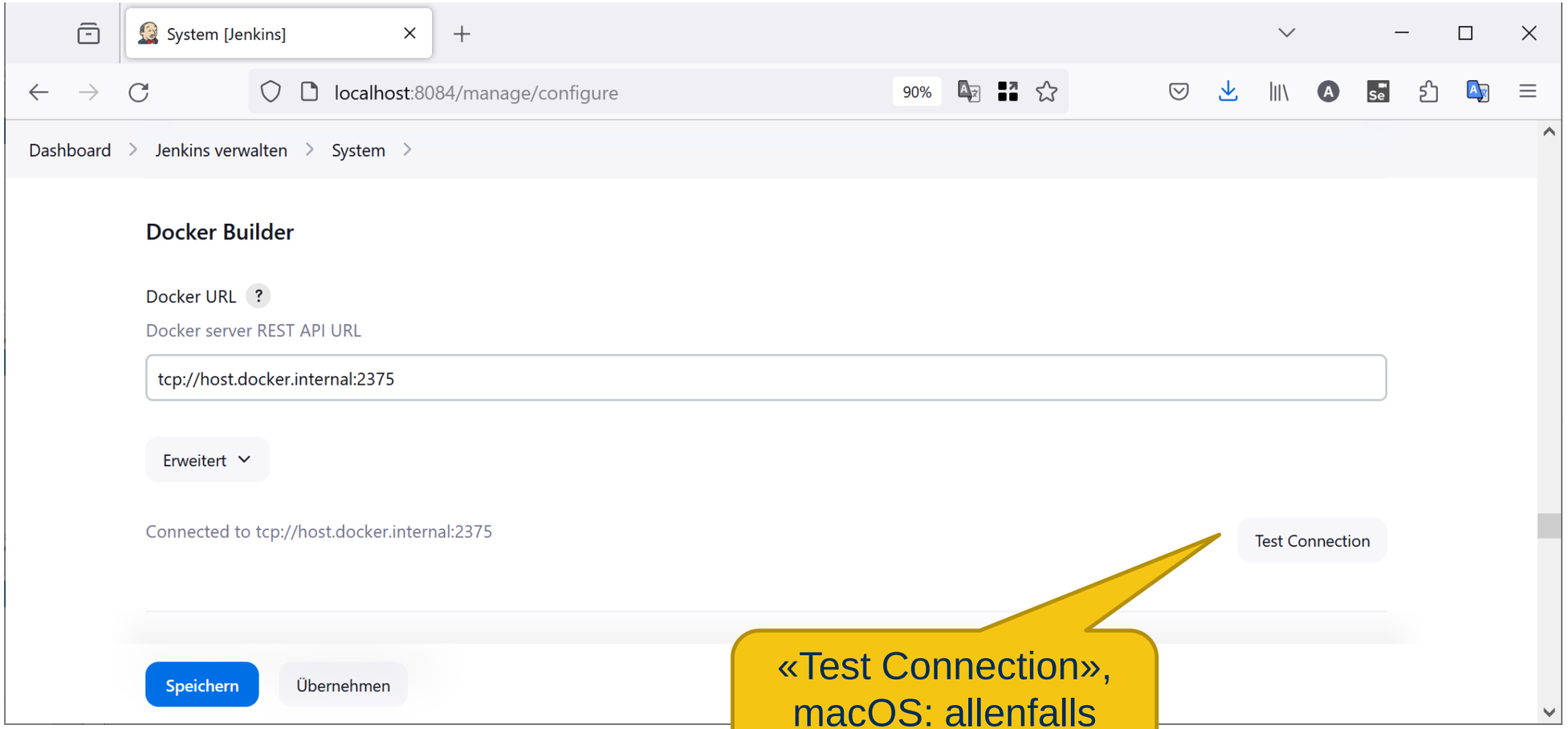
docker-build-step (Version 2.11) Plugin für Jenkins vorhanden (bereits vorinstalliert)



Docker URL konfigurieren

Tutorial

Das Docker-Plugin muss Verbindung zu Docker herstellen können:



«Test Connection», macOS: allenfalls socat starten

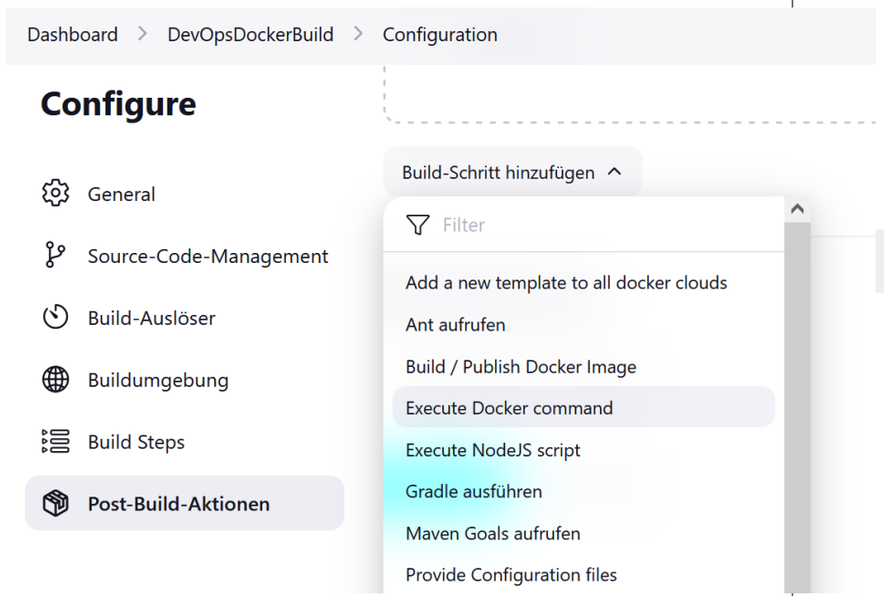
Docker URL

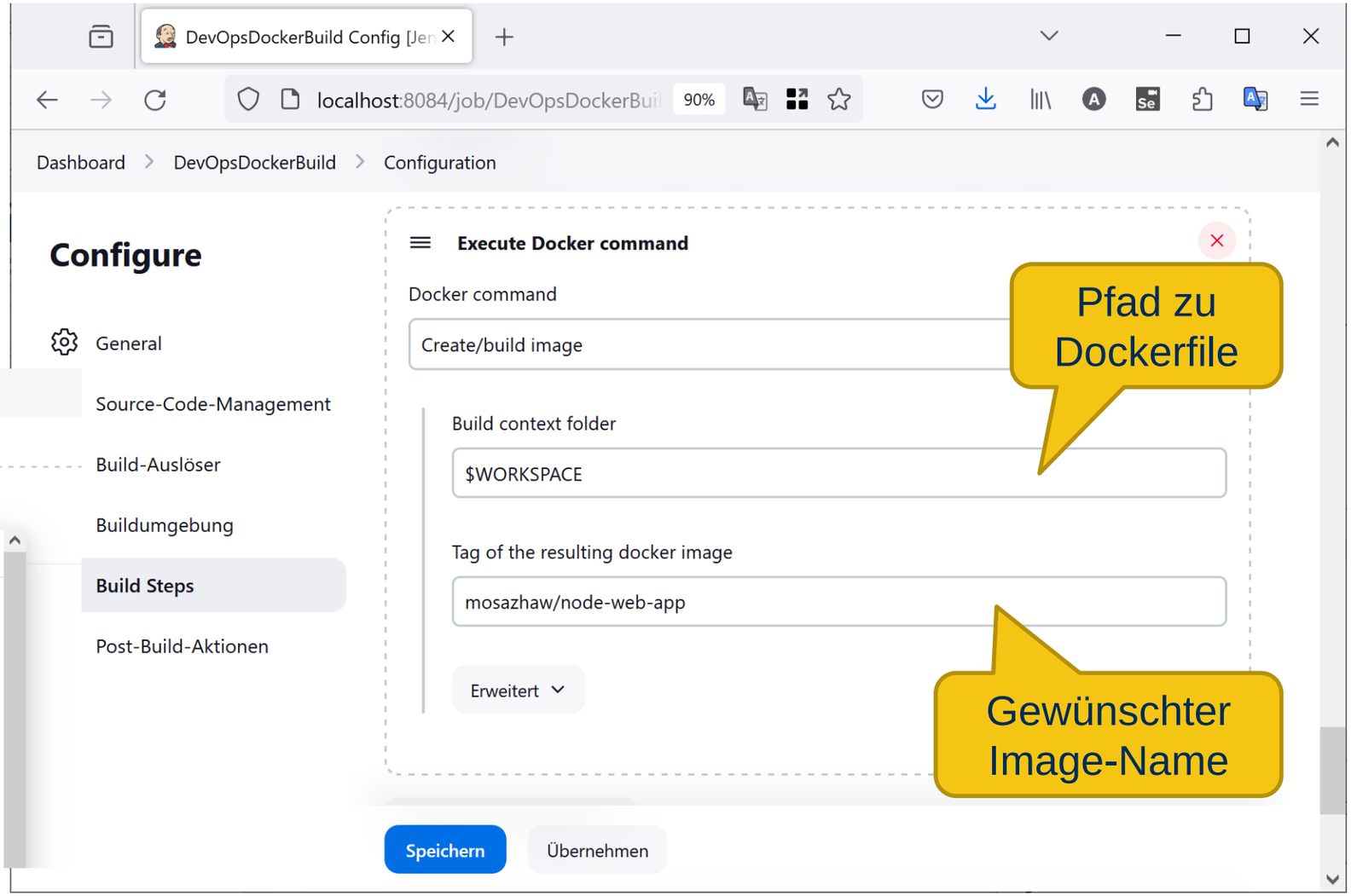
tcp://host.docker.internal:2375

Docker image: Docker build

Tutorial

Nun erstellen wir das Docker Image. Der Name könnte z.B. auch die Build-Nummer beinhalten, um jede Version festzuhalten. Im Beispiel wird ein statischer Name verwendet:

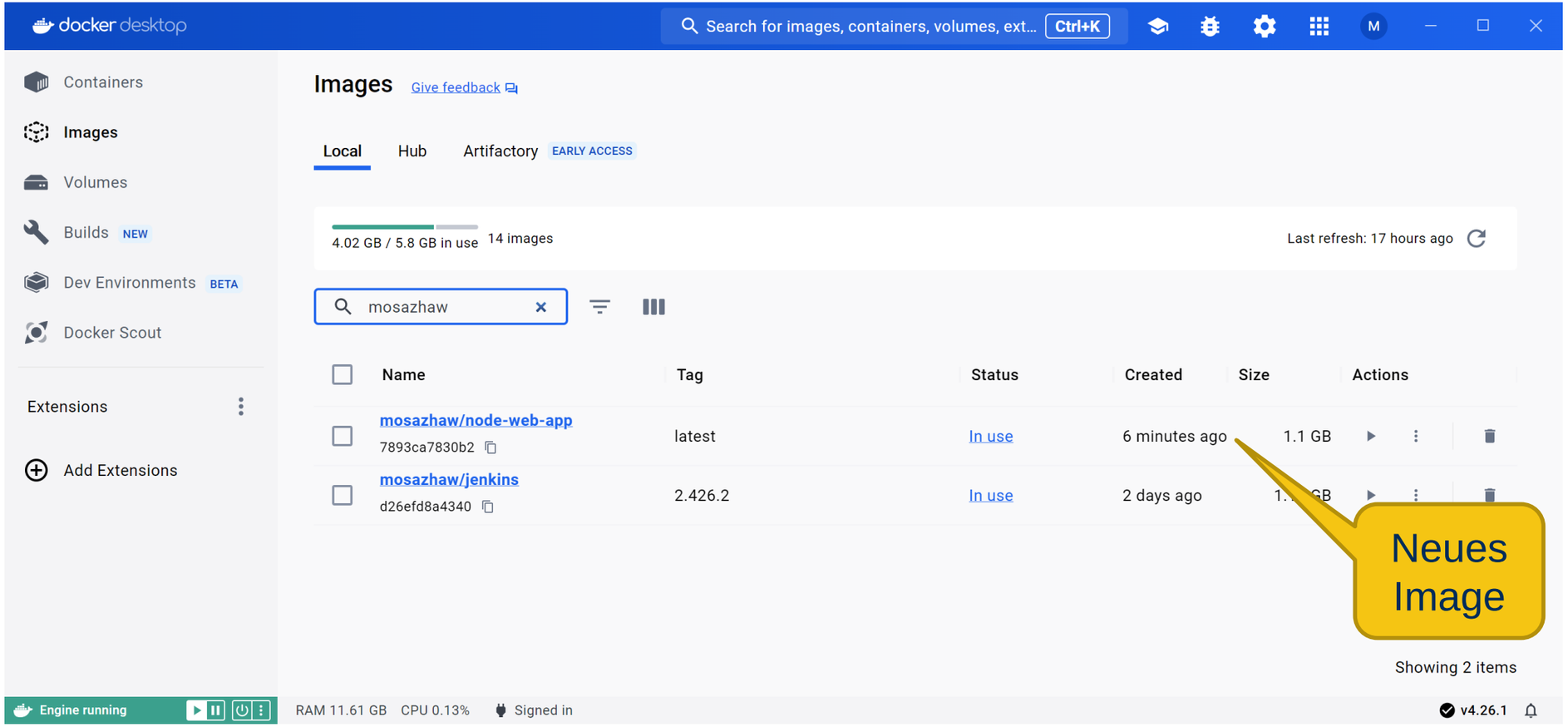




Neues Docker image

Tutorial

Befehl **docker images** oder Docker Desktop sollte das neue Image auflisten.



Deploy image with Jenkins

Ziel: Das Image automatisch starten und die Applikation verfügbar machen

Jenkins Build Triggers

Variante «ein Job»

- Build und Deploy in einem Job unterbringen
- Entwicklung und Testing ist aufwändiger (Durchlaufzeit)

Variante «mehrere Jobs»

- «separation of concerns» - Trennung der Verantwortlichkeiten
- Build Triggers können Jobs verbinden
- Deployment Job nur starten wenn Build (und Test) erfolgreich

Ziel

Zweiter Job welche das vom ersten Job erstellte Image deployed.

Jenkins Job mit Build-Steps

- Neuer Freestyle-Job **DevOpsDockerDeploy** (Vorgehen bekannt)
- Allenfalls bestehenden alten Container stoppen und entfernen
- Neuen Container erstellen aus Image
- Neuen Container starten

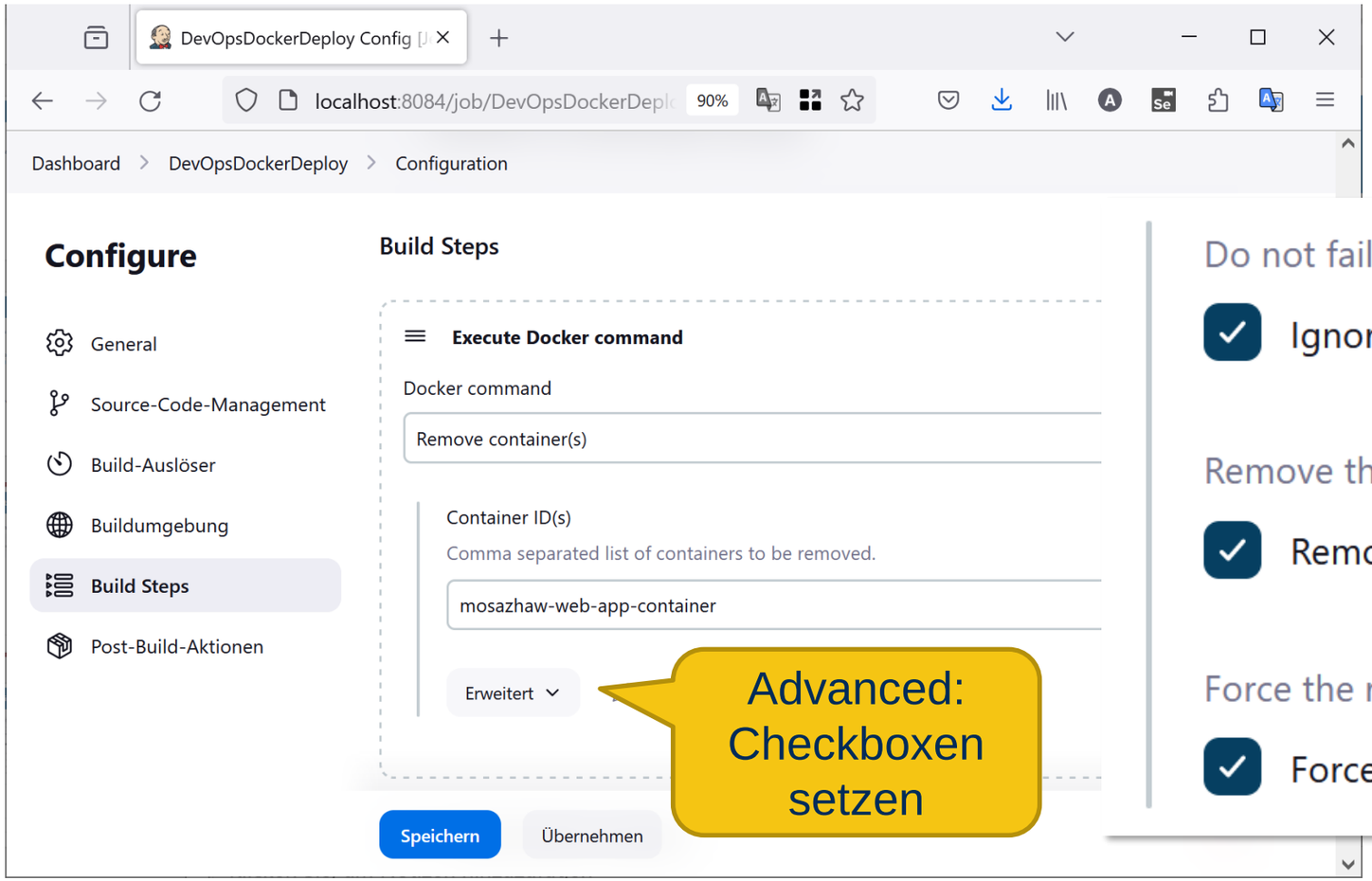
Herausforderungen

- Variante «mehrere Jobs»: Beide Jobs miteinander verknüpfen

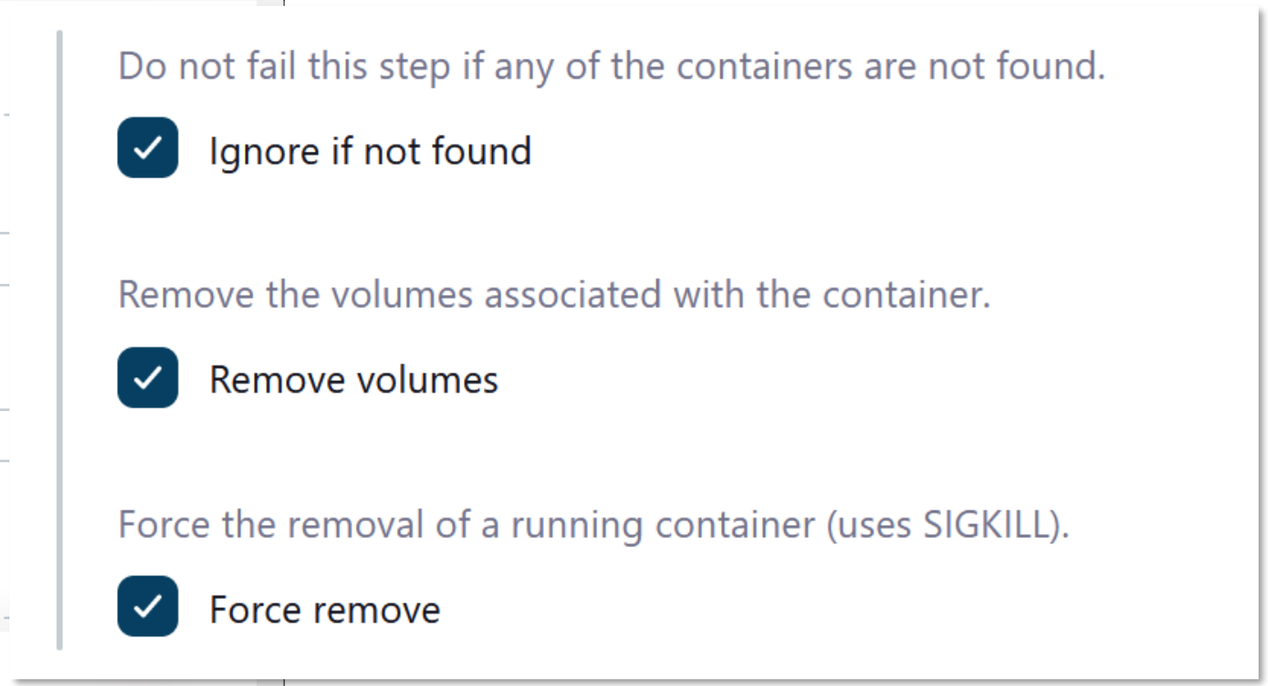
Build Step: Remove (and stop) old container

Tutorial

Ignorieren wenn es den Container nicht gibt (z.B. erste Ausführung) bzw. beenden wenn er läuft



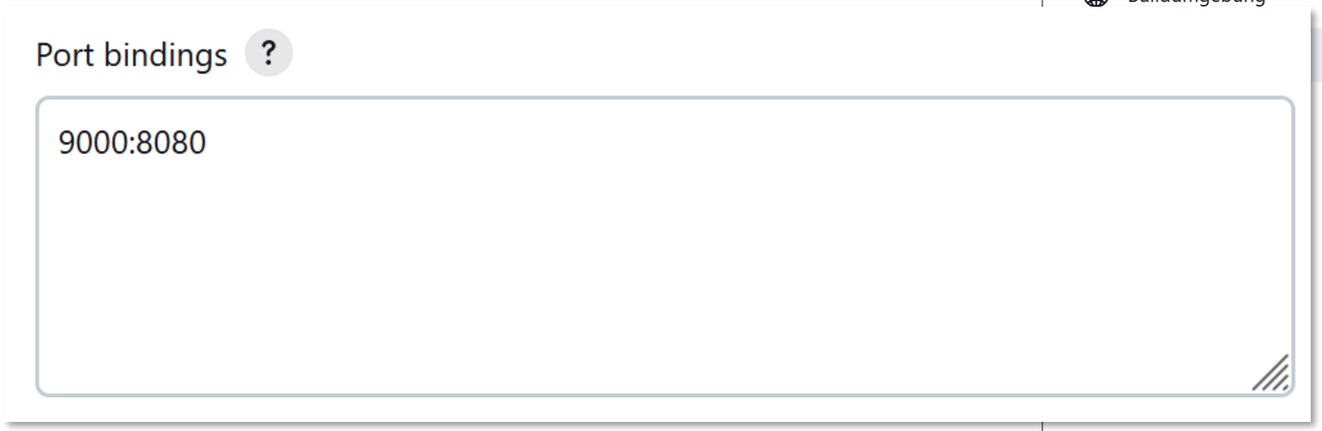
Advanced: Checkboxen setzen

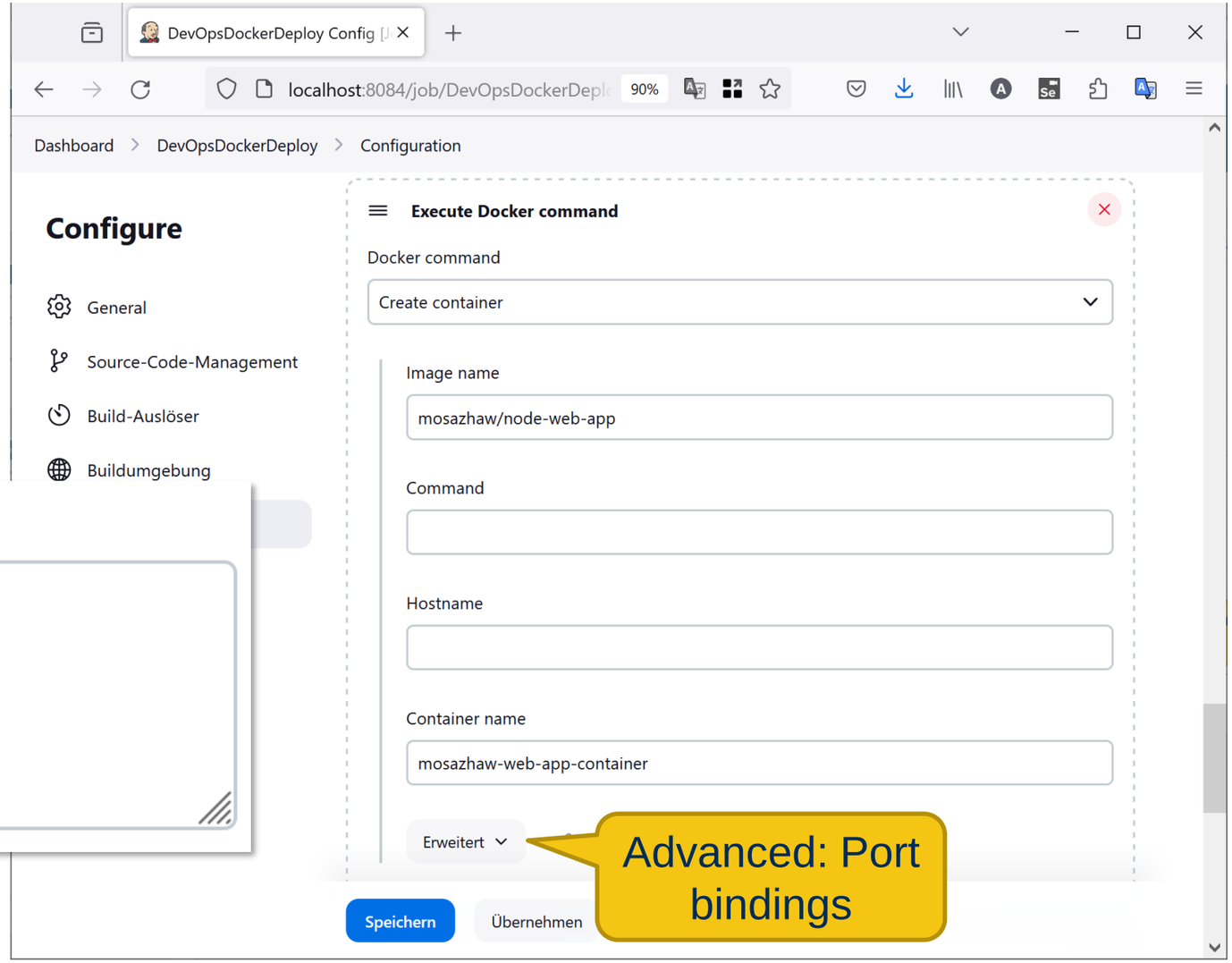


Build Step: Create new container from image

Tutorial

Container name setzen damit Container beim nächsten Durchlauf über den gleichen Namen wieder gestoppt werden kann (Ziel: immer nur ein Container soll laufen)





Start new container

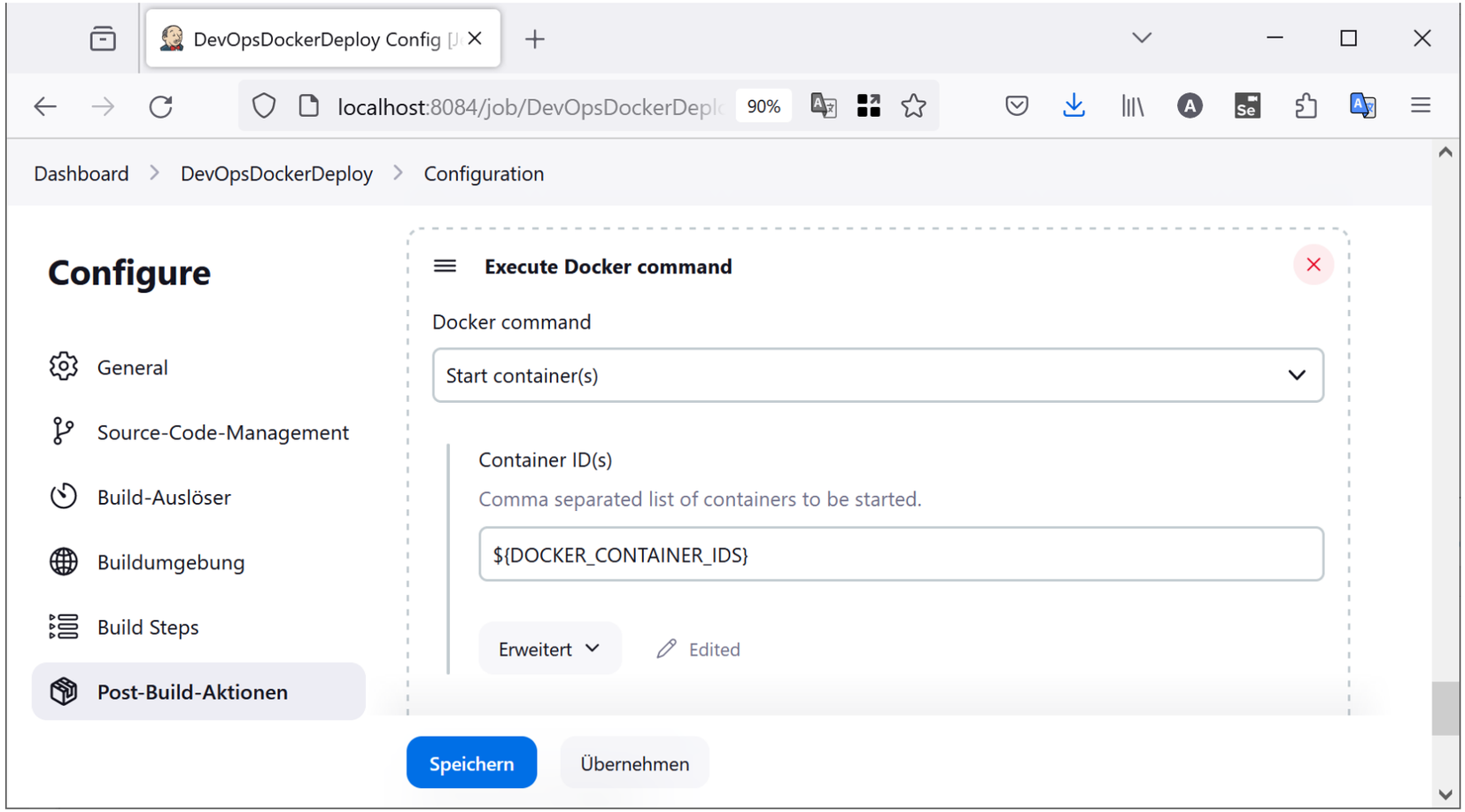
Tutorial

Container ID

Woher wissen wir die im letzten Schritt erzeugte Container ID?

Jenkins Variablen

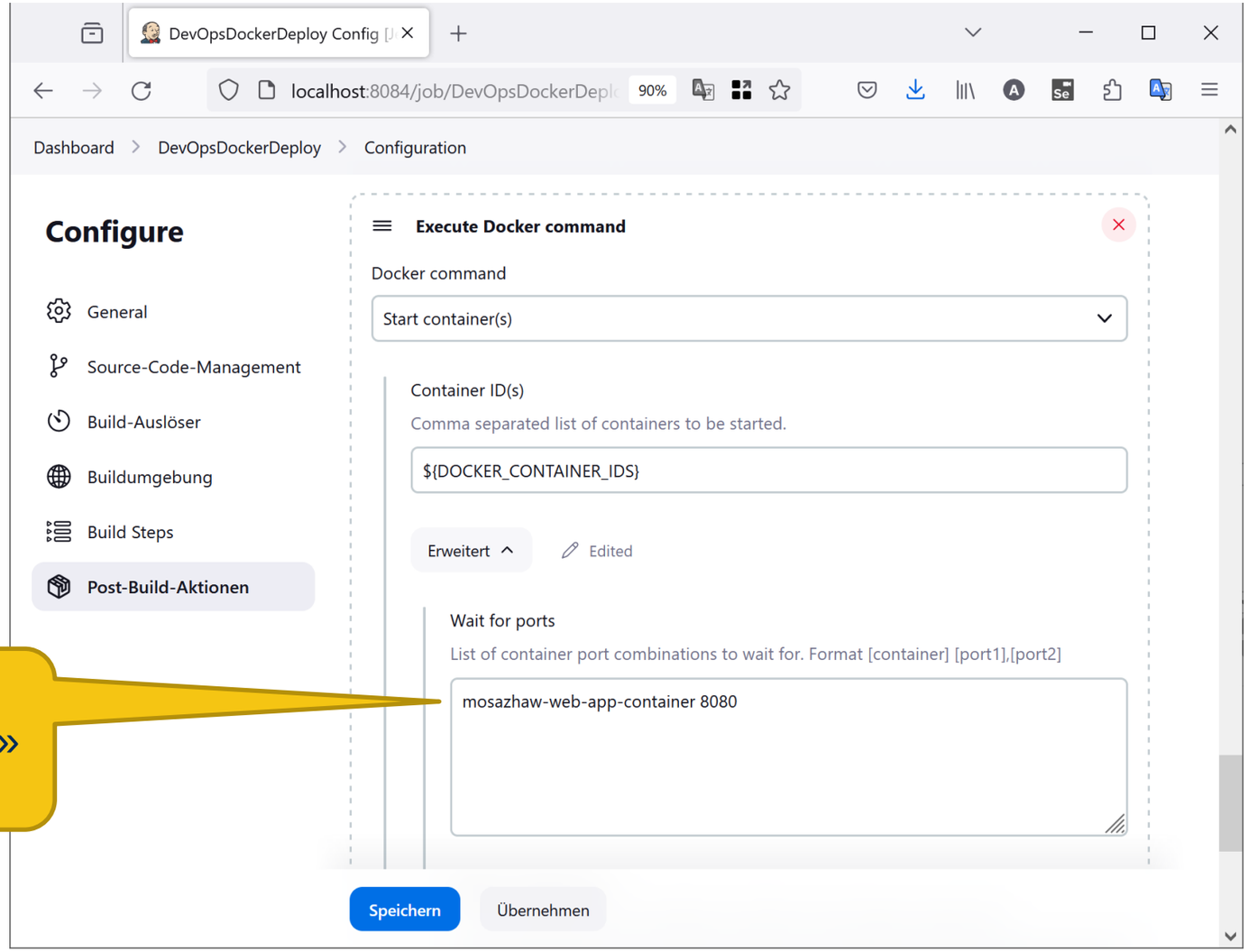
In Jenkins können Variablen verwendet werden, z.B. Von Plugins oder aus dem Environment: ${DOCKER_CONTAINER_IDS}



Start new container: Warten auf Port

Tutorial

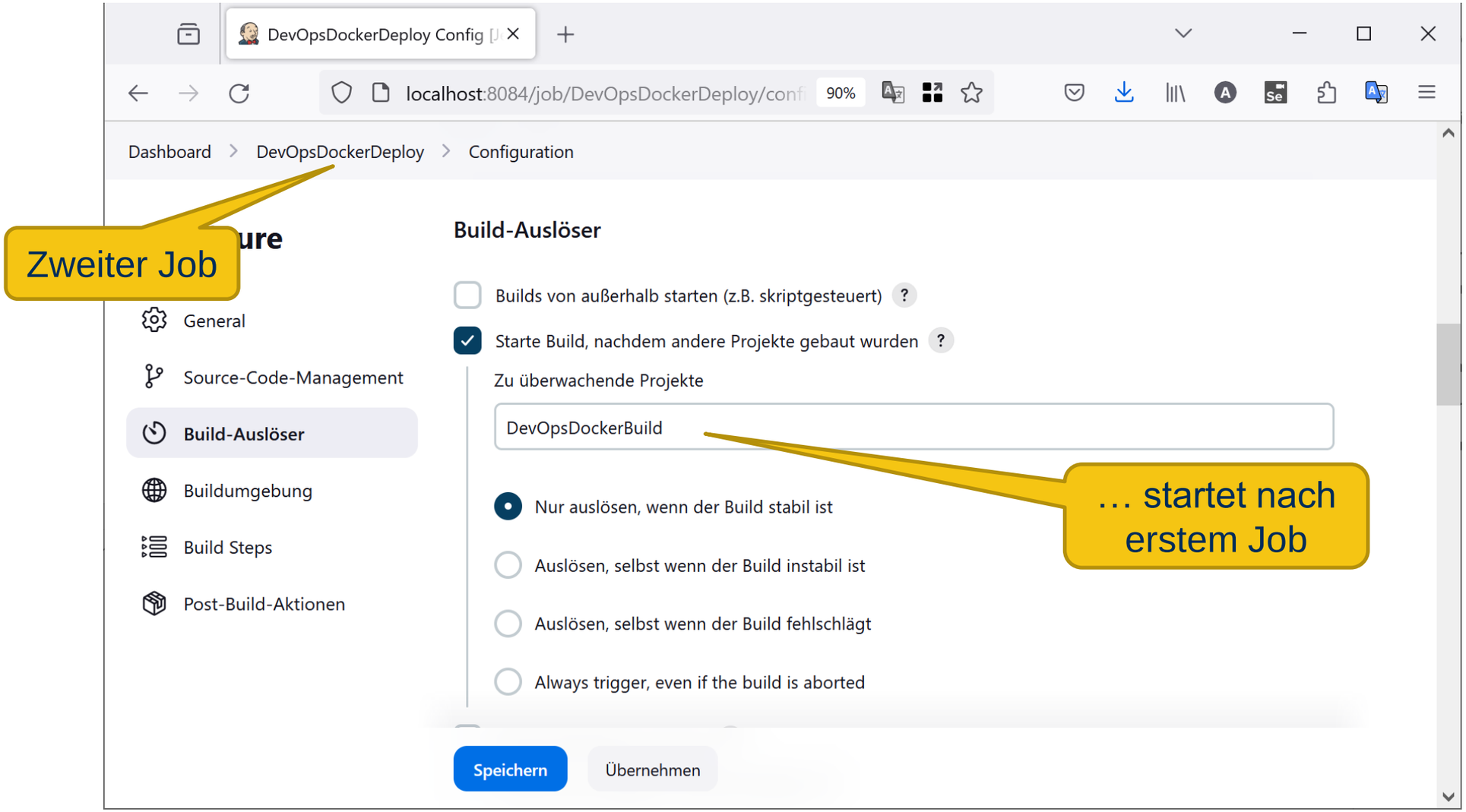
Warten bis Port 8080 / 9000 freigegeben ist



Auf diesen «Container Port» wird gewartet

Jobs verknüpfen: Build-Triggers (Build-Auslöser)

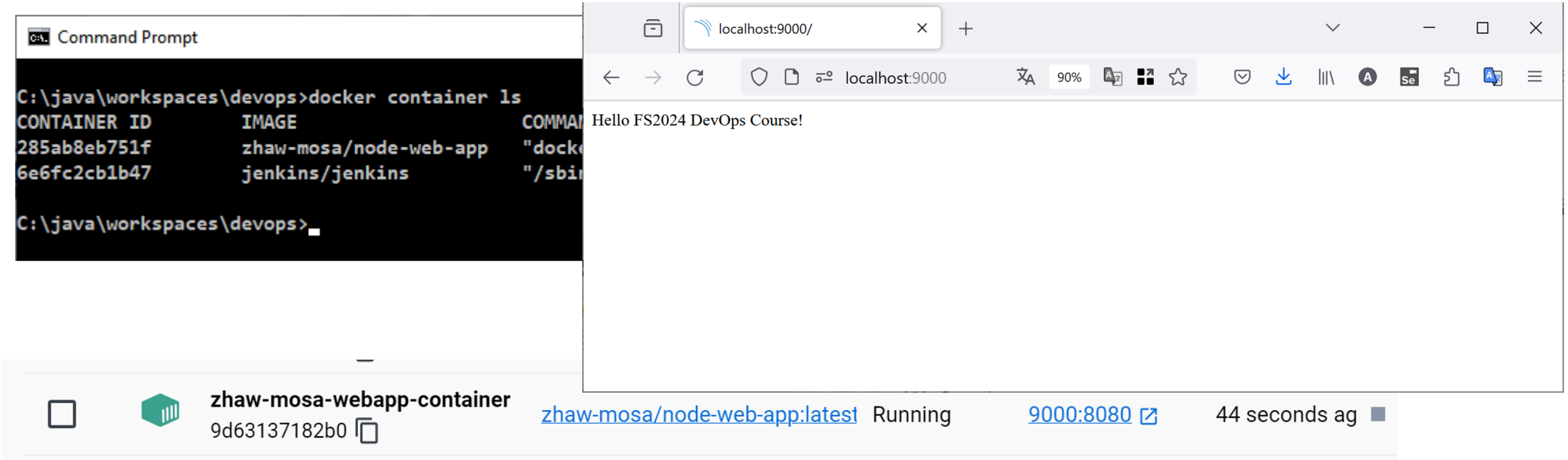
Tutorial



Test: Container und Webapp sind verfügbar

Tutorial

Container läuft und Webapp ist im Browser verfügbar



DevOps Pipeline vervollständigen: Build nach Push automatisch starten

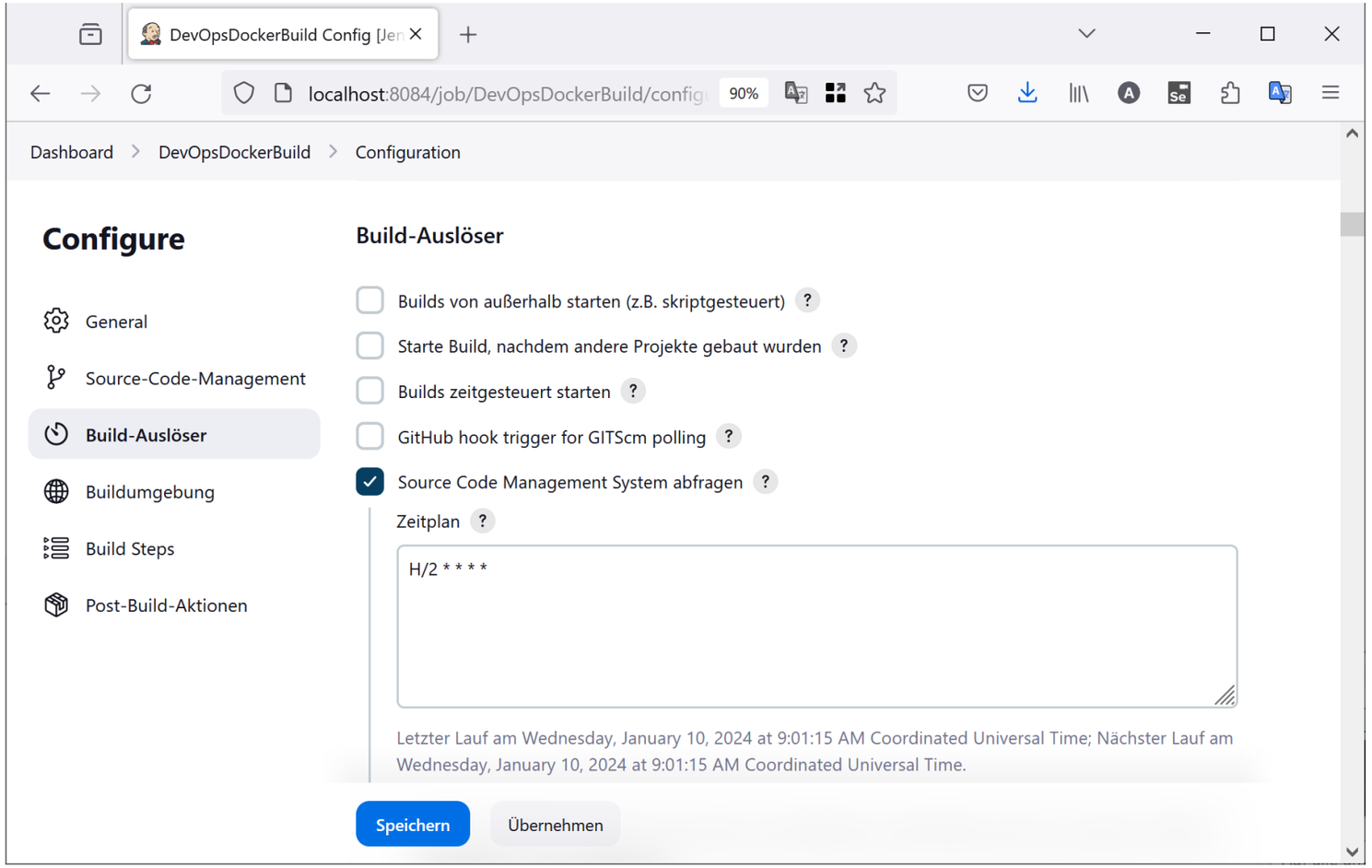
Tutorial

Build-Auslöser für DevOpsDockerBuild

– Source Code Management System abfragen
– Format: siehe Hilfe

Effizientere Lösung

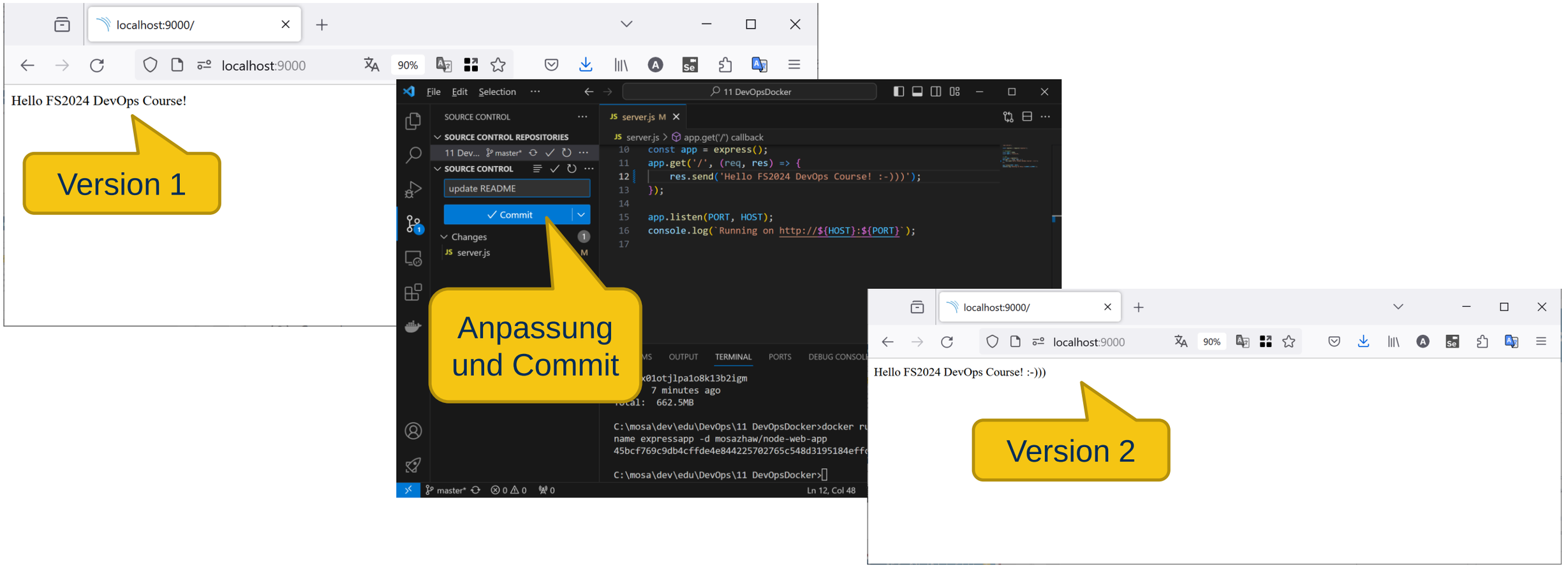
- GitHub Trigger, welcher Build bei uns auslöst
- Nur möglich, wenn Netzwerkzugriff möglich



Test unserer DevOps Pipeline: Commit, Push, Build, Deployment

Tutorial

Nach Anpassung und Commit/Push wird Version 2 automatisch gebaut, (getestet) und deployed.



Lernjournal

Lernjournal «Deployment Docker»

Ziele

- Dockerfile «manuell» bauen und starten
- Vorgang mit Jenkins automatisieren, Docker-Befehle in Jenkins konfigurieren
- Mehrere Jobs in Jenkins verknüpfen können
- Build-Auslöser in Jenkins verstehen

Checkliste

- ✓ Repository mit Applikation(en) existieren (Minimum: Node-Express-App, Advanced: DevOpsDemo)
- ✓ Manueller Build und manuelles Starten erfolgreich
- ✓ Vorgang mit Jenkins automatisiert
- ✓ Mehrfaches Starten funktioniert
- ✓ DevOps-Pipeline funktional: Code-Anpassung, Commit/Push, neues Deployment sichtbar
- ✓ Es werden mehrere verknüpfte Jenkins-Jobs verwendet
- ✓ Build-Auslöser vorhanden