Who among you has ever wasted more than half a working day trying to replicate a colleague's development environment ...only to find out that it still doesn't work?

VSCode Dev Container

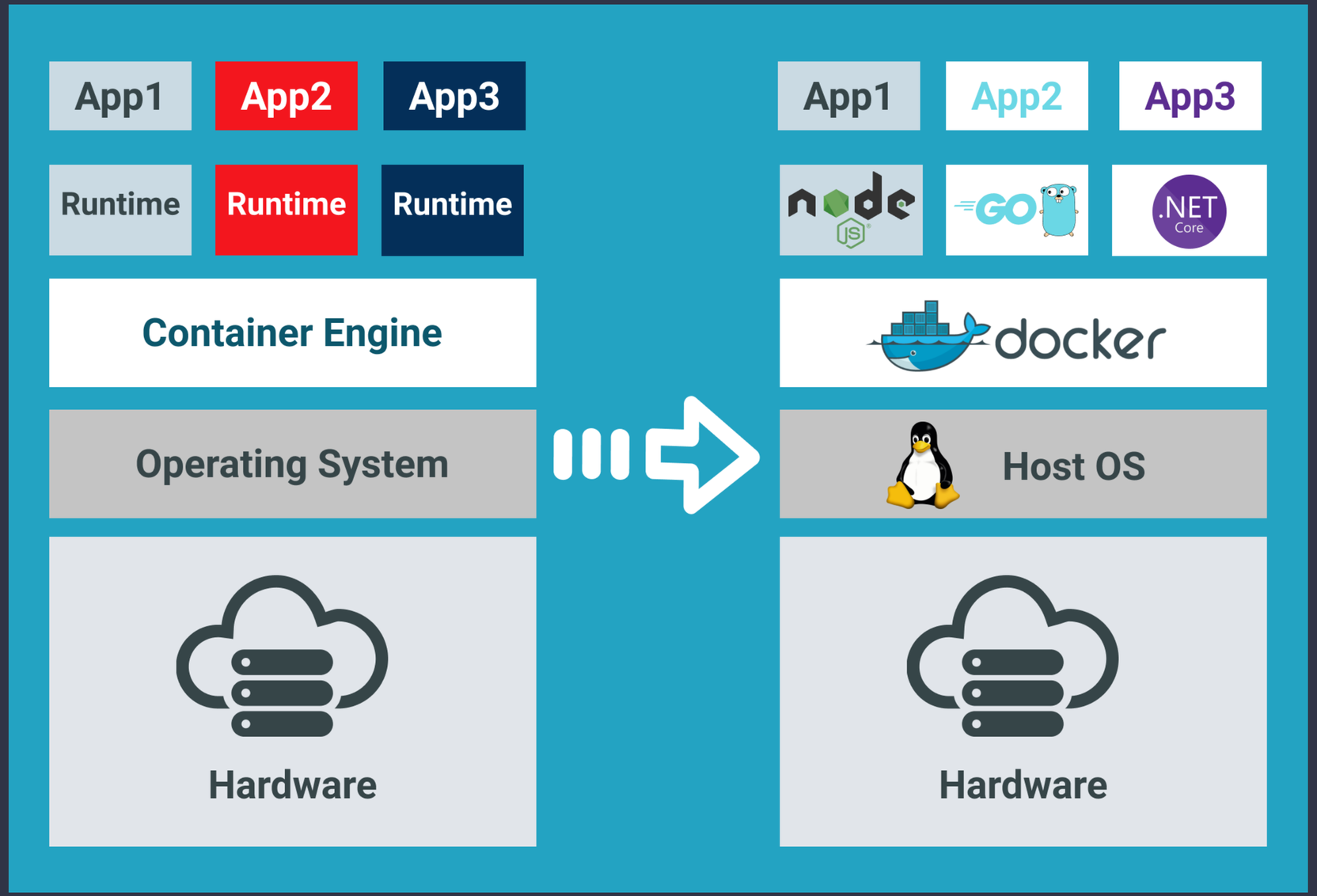
Lucasc - 28.06.2025

Agenda

1. What is docker and remote development
2. What are Dev Containers?
3. Benefits
4. Requirements
5. Setup
6. Creating your first container
7. Run an app with multiple containers

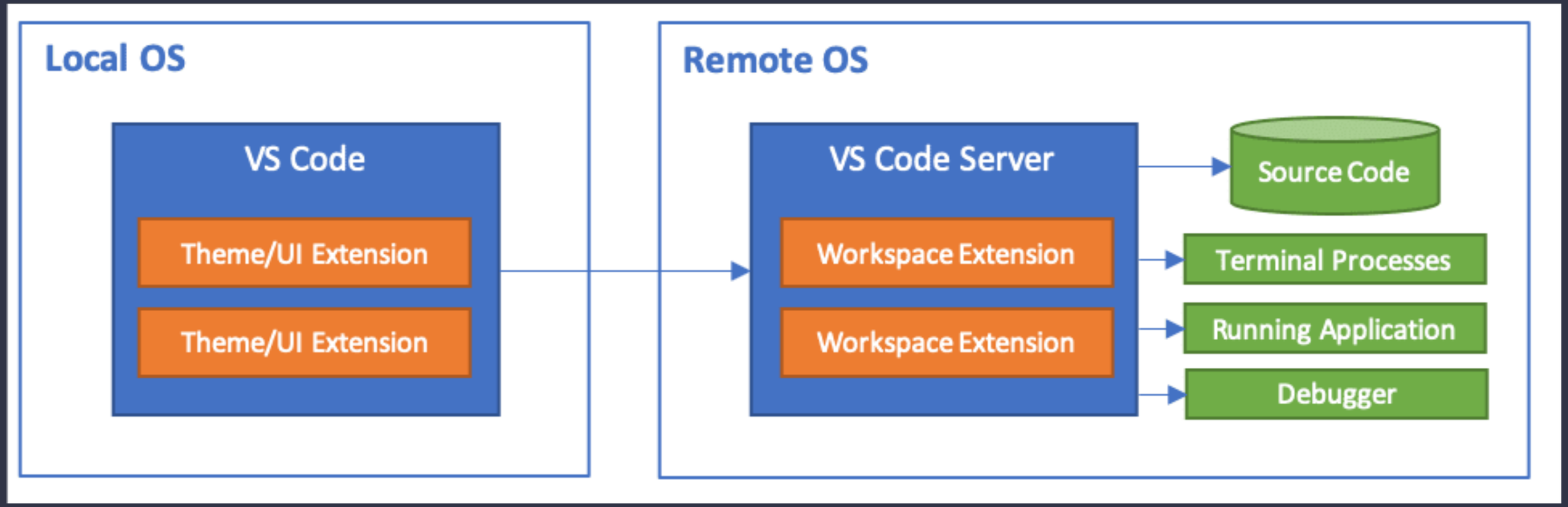
Basics

What is Docker?



Source: https://medium.com/dbconvert/building-docker-image-for-windows-desktop-applications-58236b2307f3

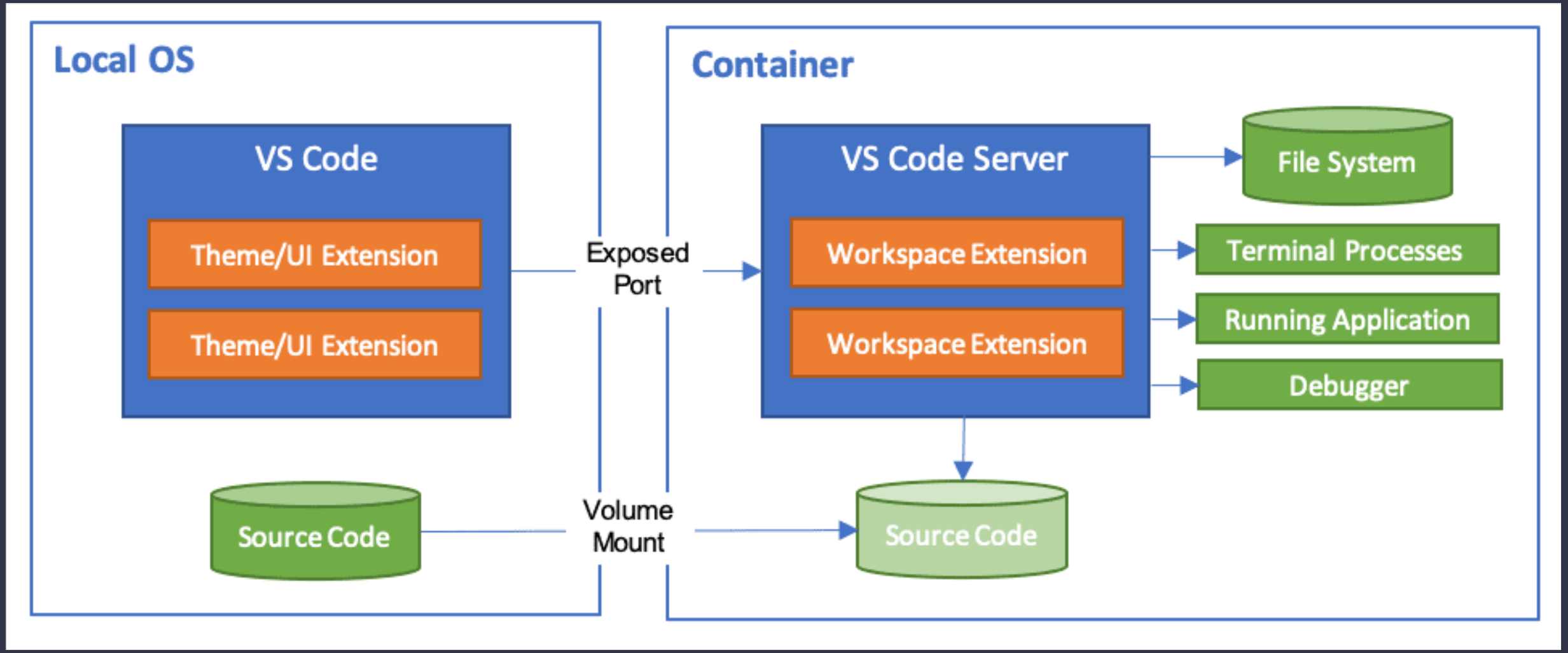
What is remote development?



Source: https://code.visualstudio.com/docs/remote/remote-overview

What are Dev Containers?

- **Dev Environment** in a **Container**
- Reproducible
  - No "*It works on my machine*"



Source: https://code.visualstudio.com/docs/devcontainers/containers

Benefits

- Reproducible
- Versionable
- Easier dependency management
- Isolated environments

Creating your first container

Requirements

- Visual Studio Code
- Dev Containers extension
- Docker or Kubernetes
  - Local: Docker Desktop / Docker installed
  - Github Codespaces
  - Remote:
    - A machine with an SSH-Server and Docker installed
    - Remote Explorer and Remote SSH extensions

Setup

1. Install all required extensions
2. Install an SSH-Server on your remote machine
3. Setup login using SSH-Keys (optional, recommended)
4. Install Docker on your remote machine
5. Connect to your remote machine

Creating your first container

1. Open the Command Palette (Press F1)
2. Select `Dev Containers: New Dev Container`
3. Select the template (e.g deno)
4. Select `create dev container`
5. Create a file called `app.ts` (and insert the code from next slide)
6. Open a terminal inside of VSCode
7. Run `deno run app.ts`

```
const fruits: string[] = ["apple", "banana", "grape", "kiwi", "pear"];

console.log("Number of items: " + fruits.length);
printList(fruits);

fruits.push("mango");
console.log("Number of items: " + fruits.length);
printList(fruits);

function printList(fruits: string[]): void {
    console.log("---------------");
    fruits.forEach((fruit) => {
        console.log(fruit);
    });
    console.log("---------------");
}

```

```
Number of items: 5
---------------
apple
banana
grape
kiwi
pear
---------------
Number of items: 6
---------------
apple
banana
grape
kiwi
pear
mango
---------------
```

Advanced

Run an app with multiple containers

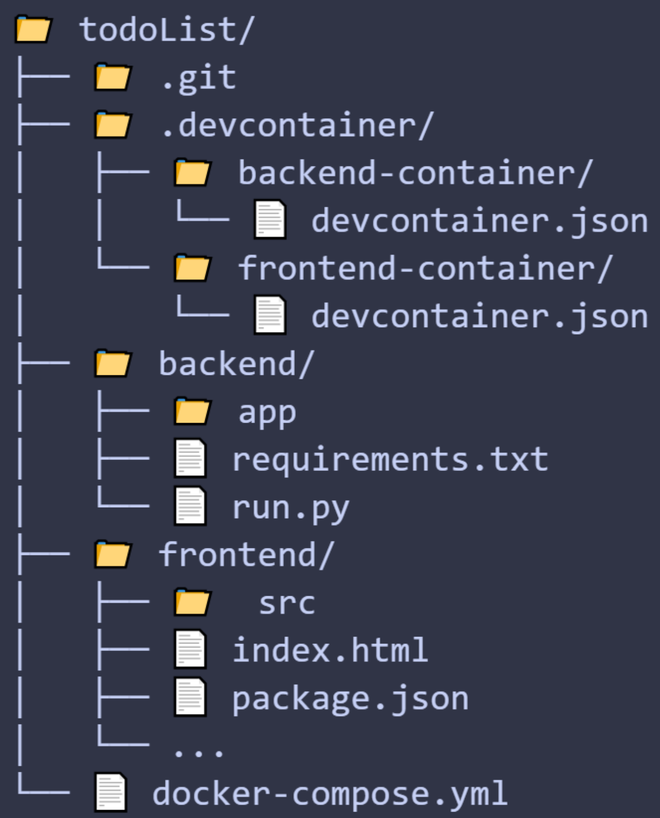
Frontend

Backend

Database

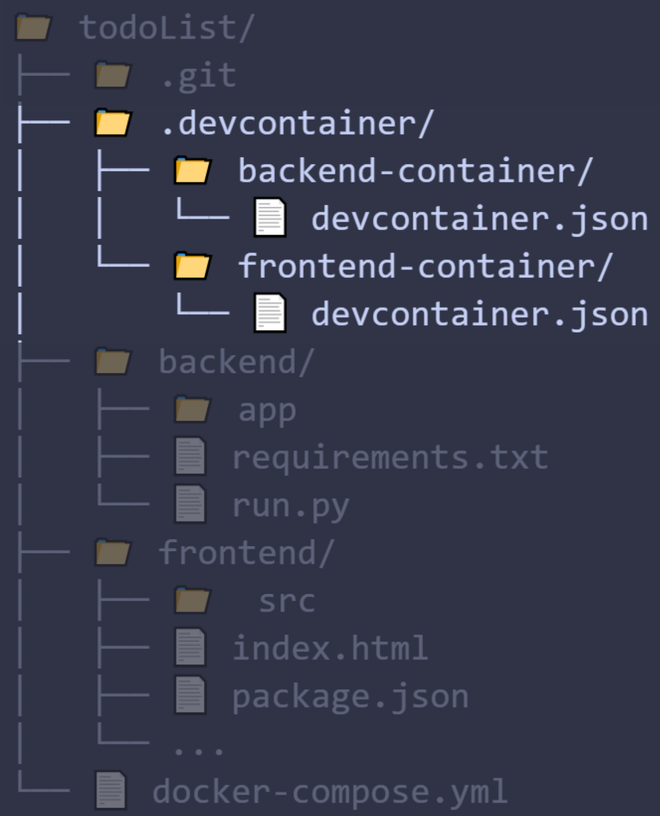
Sources: Vue: https://github.com/vuejs/art ; NodeJS: https://nodejs.org/en/about/branding ; FastAPI:https://fastapi.tiangolo.com/ ; Python: https://www.python.org/community/logos/ ; Postgres: https://wiki.postgresql.org/wiki/Logo

Run an app with multiple containers



Run an app with multiple containers

devcontainer.json



devcontainer.json

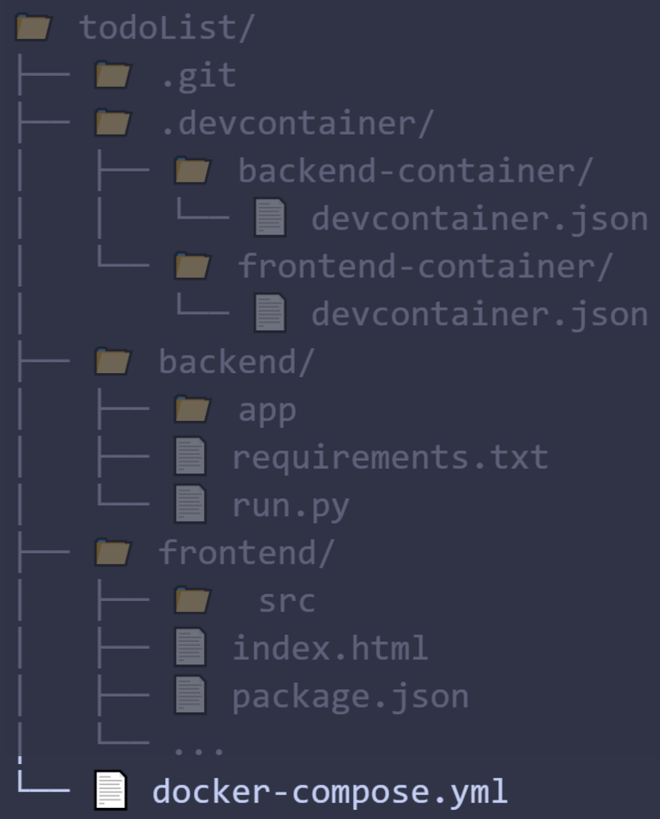
- Includes **metadata** and **settings**
  - Image/Dockerfile specific
  - Docker Compose specific
  - Lifecycle hooks
  - Minimum host requirements
  - Port attributes
- **Installed extensions**

devcontainer.json

```
{
    "name": "Backend",
    "service": "backend",
    "dockerComposeFile": [
        "../../docker-compose.yaml"
    ],
    "forwardPorts": [
        8000
    ],
    "postCreateCommand": "cd backend ; python -m pip install -r requirements.txt",
    "shutdownAction": "none",
    "workspaceFolder": "/workspace"
}
```

Run an app with multiple containers

docker-compose.yml



docker-compose.yaml

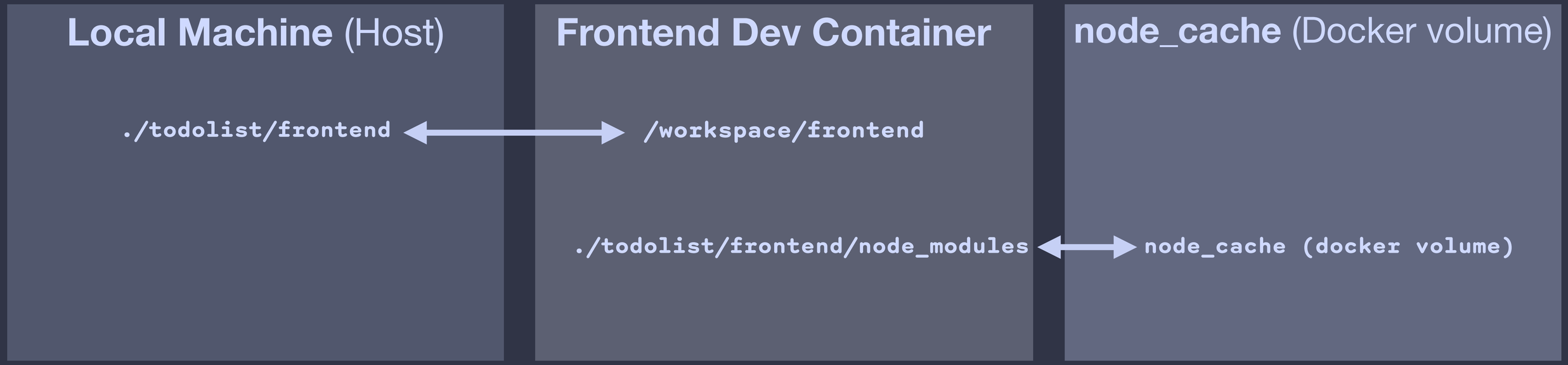
- Defines
  - Containers
  - Networks
  - Volumes
- Sets start oder

docker-compose.yaml

```yaml
services:
  backend:
    image: mcr.microsoft.com/devcontainers/python:3.11-bookworm
    volumes:
      - ./:/workspace
    command: sleep infinity
    ports:
     - 8000:8000
    networks:
     - devcontainers

  postgres:
    image: postgres:15
    container_name: todo_postgres
    environment:
      POSTGRES_USER: postgres
      POSTGRES_PASSWORD: postgres
      POSTGRES_DB: todo_db
    volumes:
      - postgres_data:/var/lib/postgresql/data
    networks:
      - devcontainers
```

Volumes


Local Machine (Host)
./todolist/frontend
Frontend Dev Container
/workspace/frontend
./todolist/frontend/node_modules
node_cache (Docker volume)
node_cache (docker volume)

Volumes

- **Persistent storage** managed by docker
- Uses cases:
  - **Caching** dependencies
  - **Local databases**
  - **Share files** between containers

Volumes

```
21  "mounts": [
22      "source=node_cache,target=/workspace/frontend/node_modules"
23  ]
```

Start everything

1. Open the folder `todoList` in the repo
2. Open the Command Palette (Press F1)
3. Select `Dev Containers: Reopen in Container`
4. Select to which container you want to connect
5. Run
   - `python run.py` for the backend
   - `npx vite dev --host` for the frontend

Lifecycle hooks

- **Commands** that run on a **different points** in the **container lifecycle**
  - for example to install dependencies, start services, set permissions
- e.g **postCreateCommand, postStartCommand, postAttachCommand**

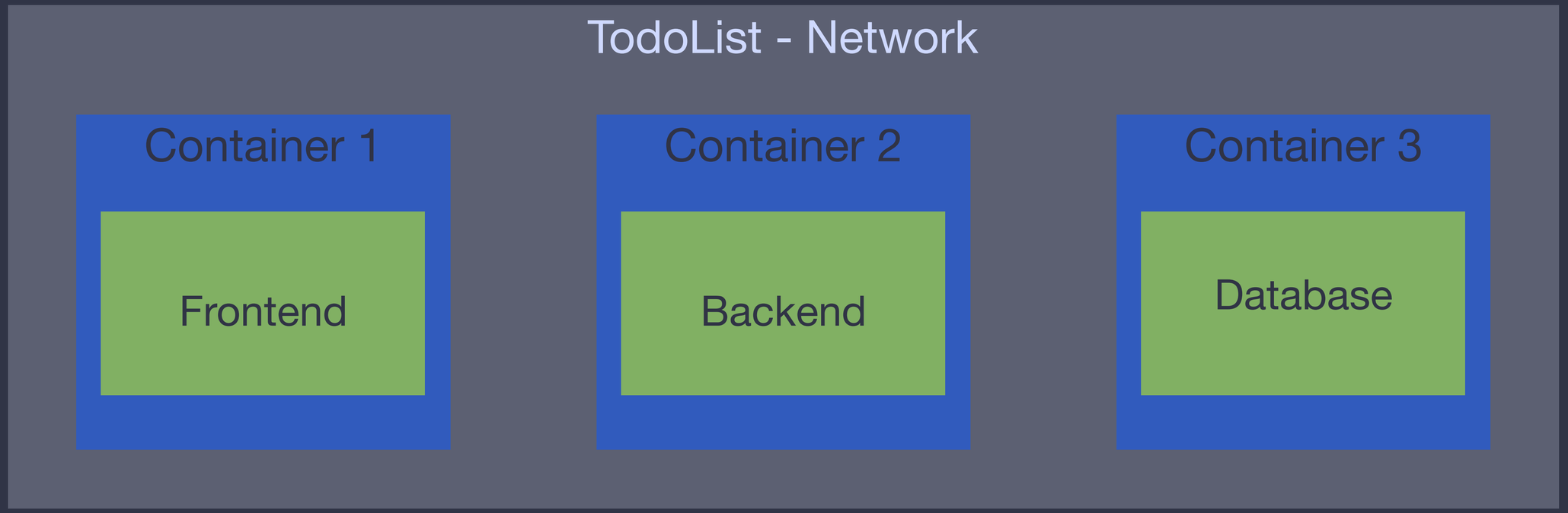
Example:

```
{
    "postCreateCommand": "cd frontend/ ; npm install"
}
```

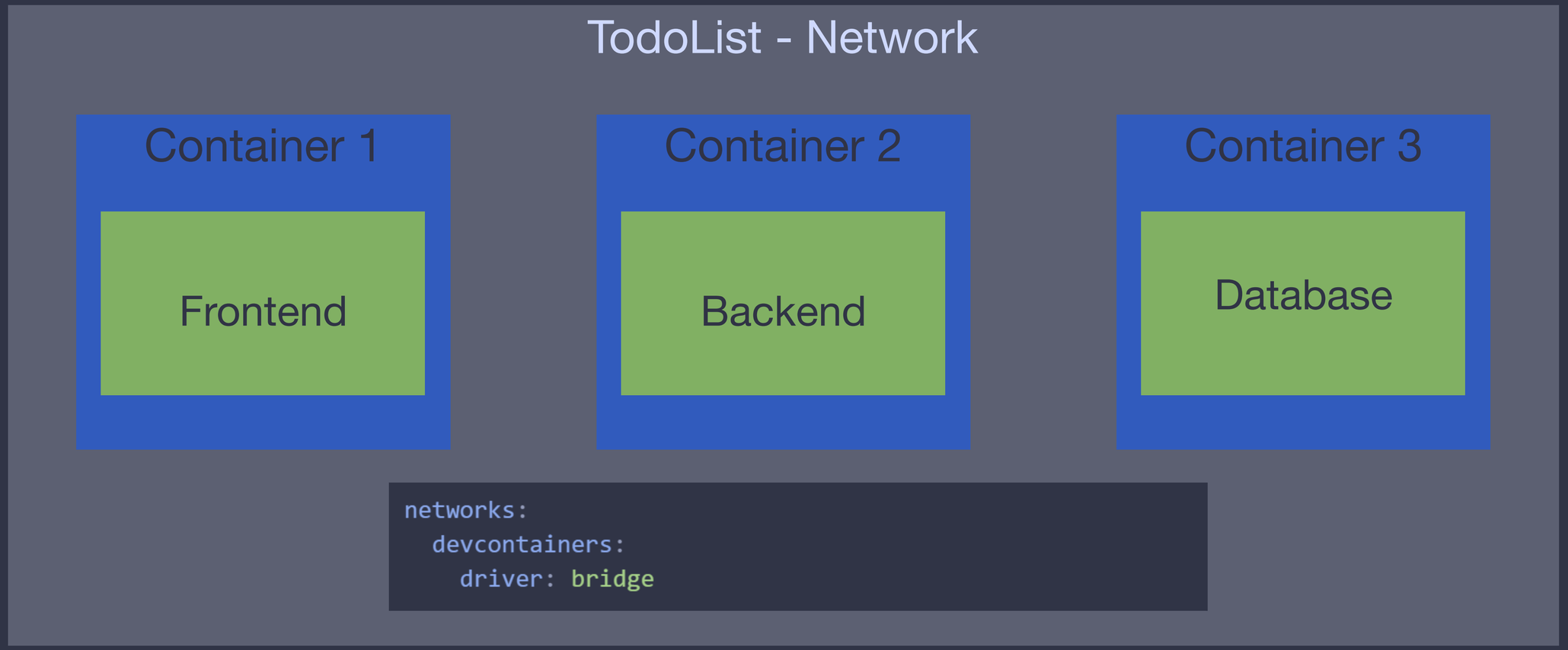
Connect to multiple containers

- One VSCode window per container
- Multiple windows possible
  - `File -> New Window`
  - Press `F1`
  - Select `DevContainers: Reopen in Container`
  - Select the container

Networks

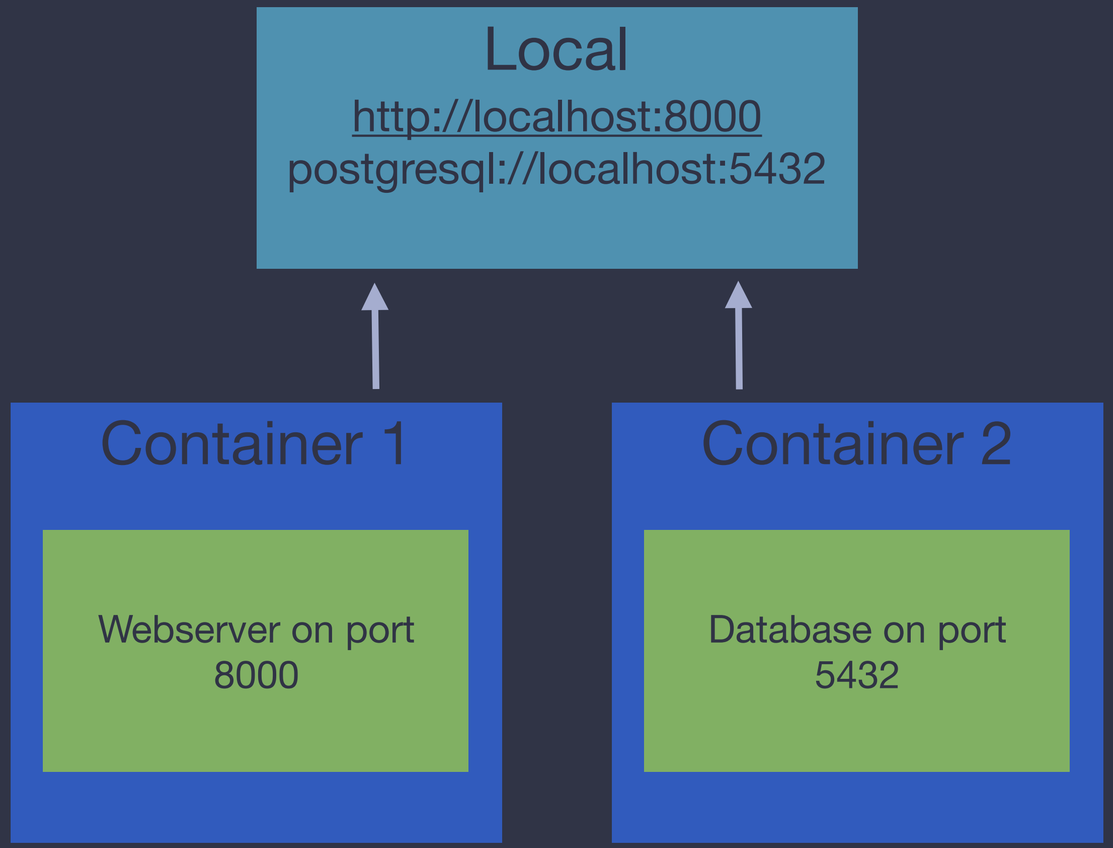

TodoList - Network
Container 1
Frontend
Container 2
Backend
Container 3
Database

Networks

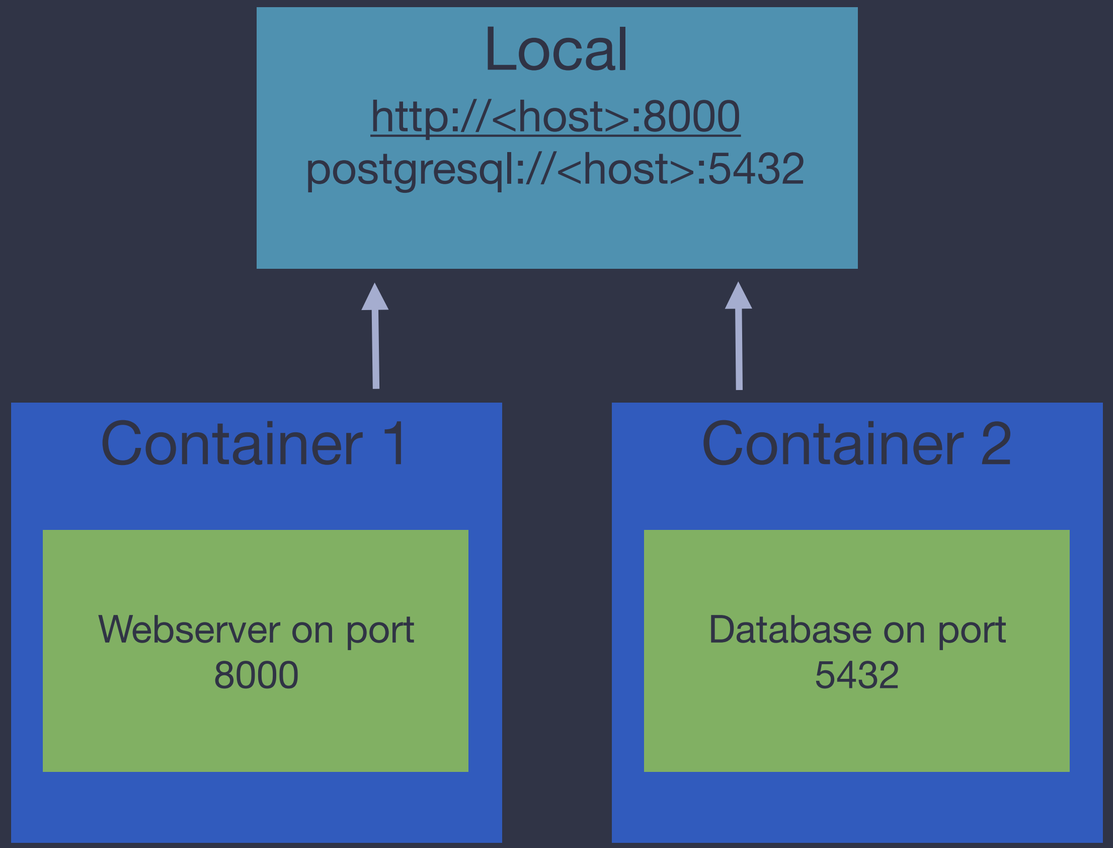

TodoList - Network
Container 1
Frontend
Container 2
Backend
Container 3
Database
networks:
devcontainers:
driver: bridge

Ports

Port forwarding

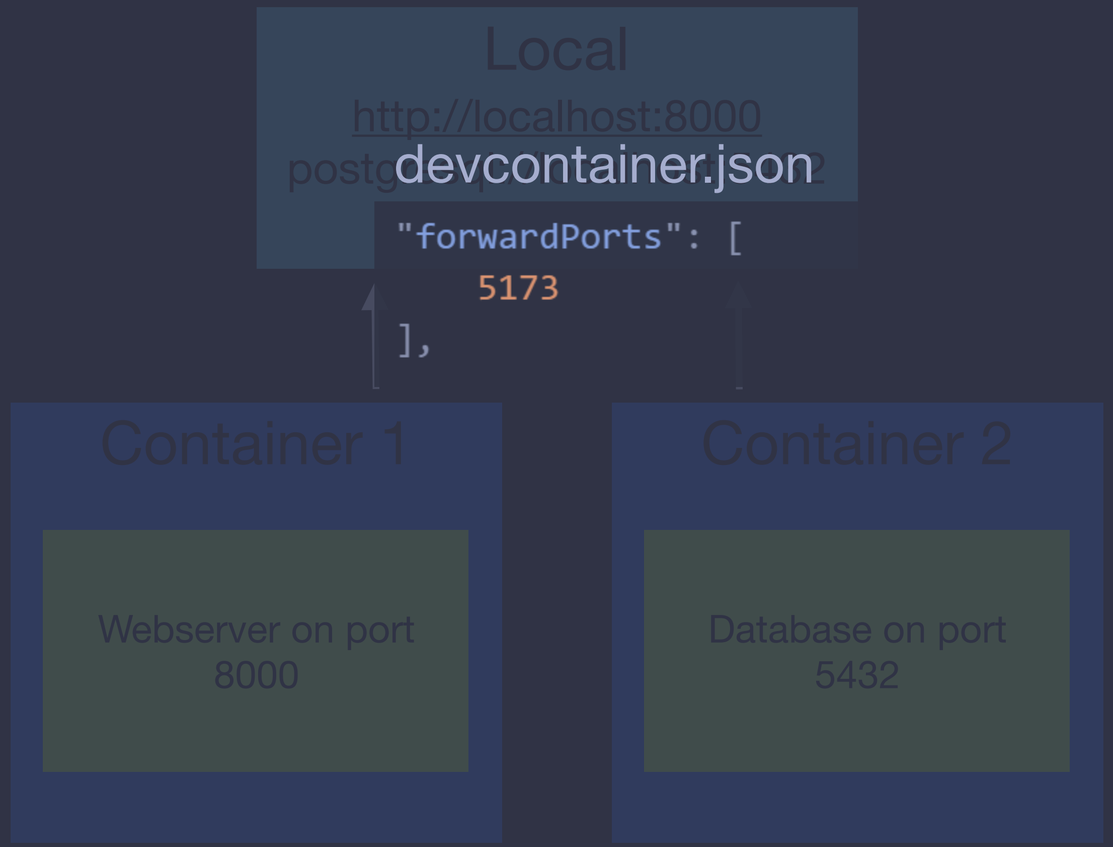


Publishing

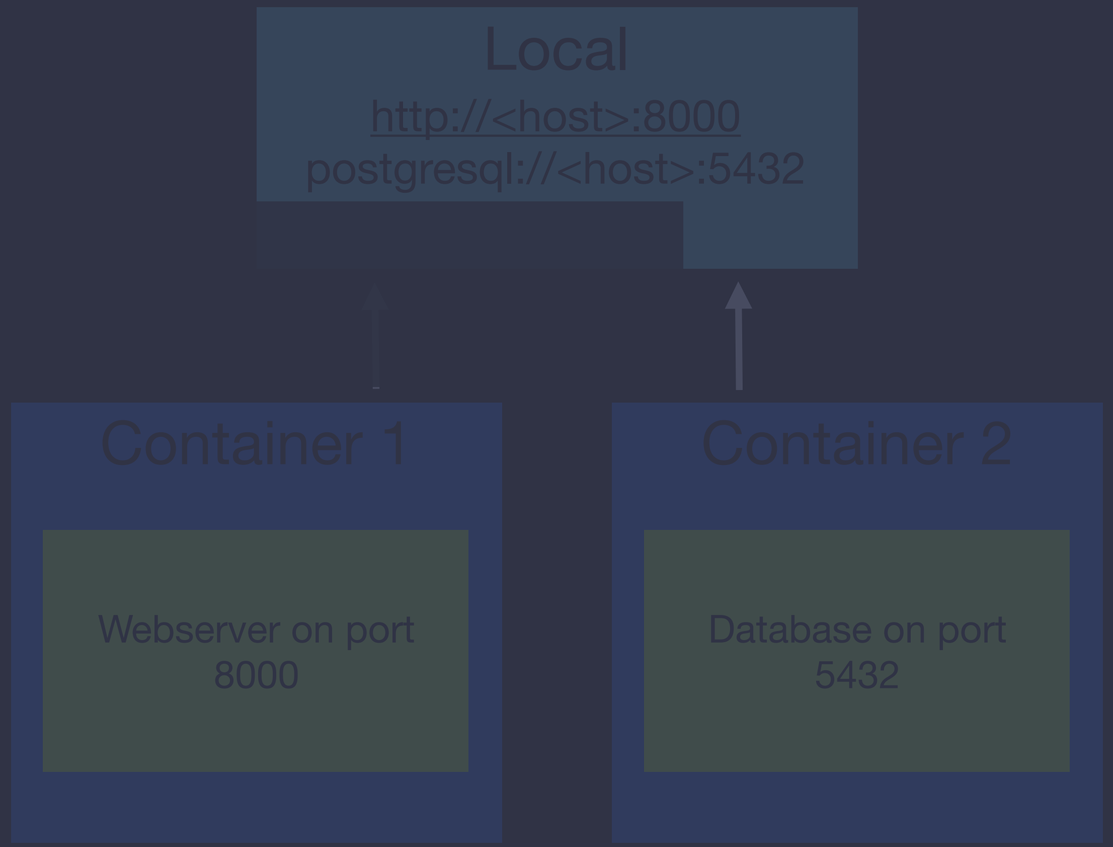


Ports - Forwarding

Port forwarding

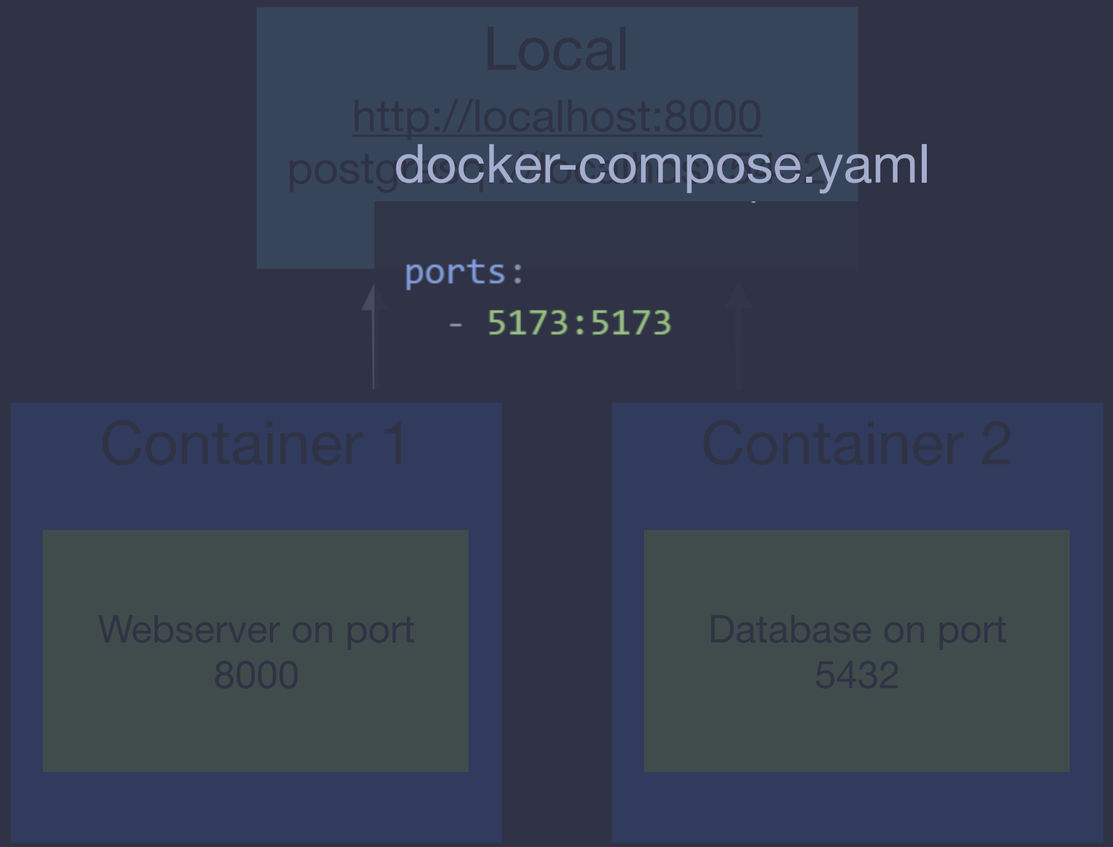


Publishing

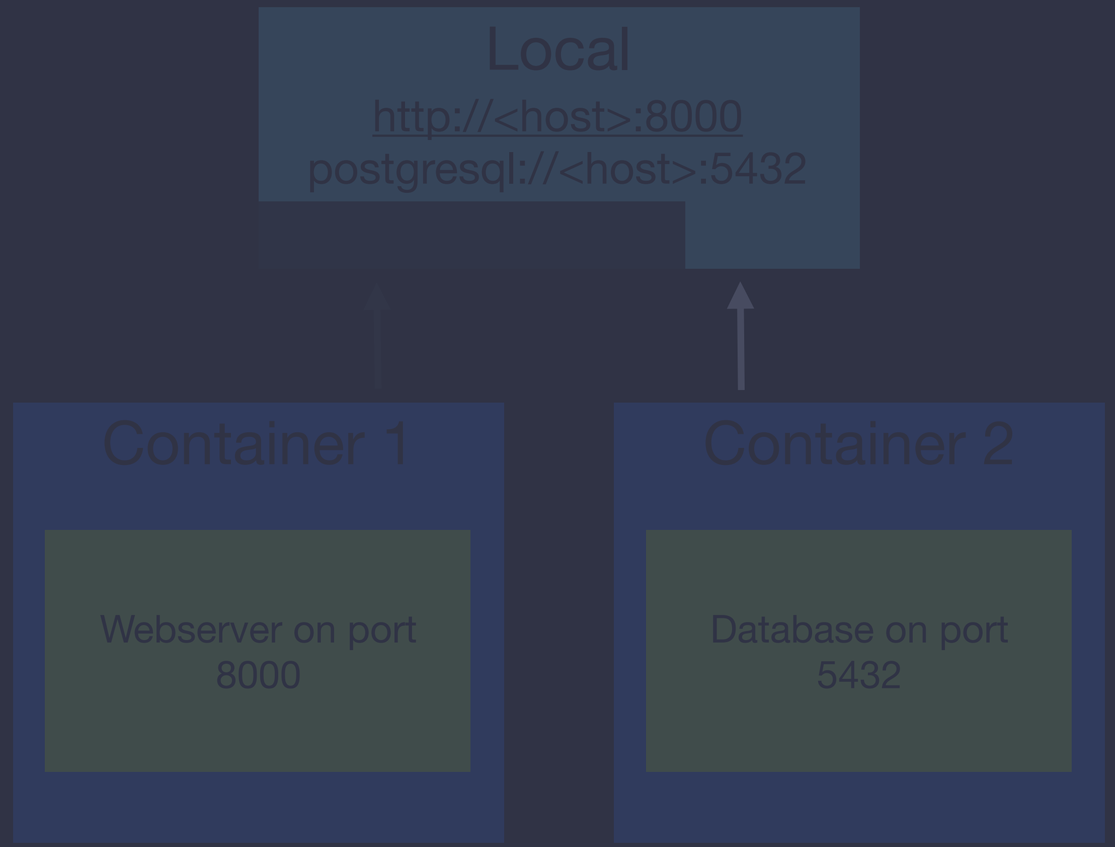


Ports - Publishing

Port forwarding



Publishing



Source: Generated with Chat GPT. The prompt is on the next slide.

Prompt for that image

- A photorealistic scene showing two laptops side by side on a dark wooden desk with soft ambient lighting. The left laptop is old and severely damaged: the screen is cracked with a bullet-like hole, the casing is dirty and scratched, and multiple keys are missing or scattered around. The screen displays a broken Node.js logo with a sad cartoon-style face drawn on top, expressing frustration or disappointment. The right laptop is new, modern, and clean, with an illuminated screen showing a smiling cartoon-style face. On its screen, there's a Node.js and Visual Studio Code logo, both encapsulated inside a stylized container or devcontainer illustration, emphasizing safety and functionality. No glass case is present. The atmosphere is professional yet fun, highlighting a clear contrast between decay and well-maintained productivity tools.

- Style:
- Photorealistic
- Balanced lighting with soft shadows
- Clean, neutral background
- Mood: optimistic, light-hearted, tech-oriented

- Visual elements:
- Cracks, dirt, and missing parts on the broken laptop
- A glowing screen with icons and a smile on the working one
- Node.js and VS Code branding should be clear and centered
- Faces on the screens should have a slight cartoon touch

https://github.com/cz-lucas/vscode-devcontainers-workshop

More resources

- https://code.visualstudio.com/docs/devcontainers/containers
- https://learn.microsoft.com/en-ca/shows/beginners-series-to-dev-containers/introduction-1-of-8--beginners-series-to-dev-containers
- https://www.zweitag.de/blog/how-to-develop-with-dev-containers
- https://code.visualstudio.com/docs/remote/ssh
- https://www.digitalocean.com/community/tutorials/how-to-use-visual-studio-code-for-remote-development-via-the-remote-ssh-plugin
- https://medium.com/@techsuneel99/docker-from-beginner-to-expert-a-comprehensive-tutorial-5efec10c82ab