MLOps 1: KI-Modelle bereitstellen mit Docker

Additional Resources

- **Official Docker Documentation**: https://docs.docker.com/
- **Docker Hub**: https://hub.docker.com
- **More from KISZ-BB**: https://hpi.de/kisz

Why Docker

Suppose you're developing a Python application, and you've encountered a common challenge: it works on your machine but not on your colleague's. This discrepancy can arise due to differences in system configurations, library versions, and other environmental inconsistencies. Docker provides a solution to this problem by packaging the application and its environment into isolated containers, ensuring it works consistently across different setups.

Container vs Images

Docker images and containers are fundamental concepts in the Docker ecosystem.

- **Images**: An image is a static, immutable snapshot that contains the application, its dependencies, and the runtime needed to execute it. Think of it as a template or a blueprint for creating containers. It defines what the system should look like, including installed applications, environment variables, and default file systems.
- **Containers**: While an image is static, a container is a dynamic, running instance of an image. When you run an image, Docker creates a container from it. Containers are isolated from each other and from the host system, ensuring that your application behaves the same way, regardless of where you run it.

Command Quick Reference

Image Management

Images are the blueprint for containers. You can download, list, and remove them as needed.

- **Download**: `docker pull IMAGE_NAME`
- **List**: `docker image ls`
- **Remove**: `docker image rm IMAGE_ID`

Container Management

Containers are the running instances of images. You can start, list, and stop them as required.

- **Create & Run**: `docker run [--rm] [-p HOST_PORT:CONTAINER_PORT] IMAGE_NAME`
- **List**: `docker ps [--all]`
- **View logs:** `docker logs CONTAINER_ID`
- **Remove**: `docker rm CONTAINER_ID`

Dockerfile

A Dockerfile is a script that automates the creation of Docker images based on given specifications.

- **Create a file named `Dockerfile` in your project**:

```
FROM python:3.11
WORKDIR /app
COPY . .
EXPOSE 8000
CMD ["python", "model_api.py"]
```

- **Build**: `docker build -t my_model:latest .`
- **Run**: `docker run --rm -p 8000:8000 my_model:latest`

Docker Compose

Docker Compose provides a way to define and run container configurations using a simple YAML file.

- **Create a `docker-compose.yaml`**:

```
version: '3'
services:
  my_model:
    image: my_model:latest
    build:
      context: .
    ports:
      - "8000:8000"
    restart: always   # optional: restart container in case of system restart
    volumes:          # optional: persistent data storage
        - ./model:/app/model
```

- **Start**: `docker compose up [--build]`
- **Detached Mode**: `docker compose up -d`
- **Shut Down**: `docker compose down`

Deep Dive

Docker Basics

- **Understanding `docker run`**: The `docker run` command fetches the image, spawns a container, and executes the app within. It combines several operations (like `docker create` and `docker start`) into one step.
- **Resource Cleanup**: Over time, Docker accumulates old resources. Periodic cleanup with `docker system prune` can help free up valuable system resources.

Interactivity and Debugging

- **Interactive Shells in Docker Containers**: There might be scenarios where you'd like to interact directly with a specific environment or application inside a Docker container. Docker's interactive mode facilitates this. For instance, to get an interactive Python shell: `docker run --rm -it python:3.11` The `-it` flags ensure interactivity and TTY, which means you'll be dropped into an interactive Python shell.
- **Using `docker exec`**: Suppose you have a running container and you want to run additional commands inside it or spawn a new shell. The `docker exec` command is designed for this: `docker exec -it CONTAINER_ID /bin/sh` This will provide you with a shell prompt inside the running container.

Data Management and Persistence

- **Bind Mounts**: Allows you to map host directories/files directly to directories/files inside containers, reflecting immediate changes.
- **Volumes**: Managed by Docker, volumes are a more durable method for data persistence, ensuring data remains even if containers are deleted.

Outlook

In our future workshops we plan to give an overview for:

- **Networking in Docker**: Delve into intricate network configurations and their vital role in container orchestration.
- **Advanced Docker Compose**: Dive into intricate multi-container setups.
- **Kubernetes**: Beyond Docker Compose - orchestrating containers on a grand scale.