Spickzettel: Docker – Logging & Debugging

Ziel

Container-Probleme systematisch analysieren und beheben – zur Laufzeit, im Fehlerfall oder bei Entwicklungsproblemen.

Logging mit Docker

Logs eines Containers anzeigen

```
docker logs <containername>
```

- `-f`: live folgen (tail -f)
- `--since` / `--until`: Zeitfenster eingrenzen

Beispiel

```
docker logs -f webapp
```

Logs analysieren

- Fehlerausgaben (stderr)
- Timestamps & Events
- Logging-Framework im Container nutzen (`stdout`/`stderr` an Docker übergeben)

Interaktive Fehleranalyse

In Container einsteigen

```
docker exec -it <container> /bin/bash
```

- Alternativ: `sh` bei minimalistischen Images

Laufende Prozesse anzeigen

```
docker top <container>
```

Dateisystem durchsuchen

```
docker exec -it <container> ls /app/logs
```

Container-Lifecycle analysieren

Status & Exit Code

```
docker ps -a
```

- Exit Code ≠ 0 → Fehler beim Start oder Crash

Details anzeigen

```
docker inspect <container>
```

- Zeigt Volumes, Netzwerke, Mounts, Restart-Gründe etc.

Weitere Tools & Tipps

- **`docker events`** → Live-Stream aller Ereignisse
- **`docker diff`** → Änderungen am Container-Dateisystem
- **`docker stats`** → Laufende Ressourcenverbrauch (RAM, CPU)
- **`docker cp`** → Dateien aus dem Container holen
- **`docker-compose logs`** → Mehrere Logs gleichzeitig

Best Practices

- Container-Logs immer via stdout/stderr ausgeben
- Keine direkten `log.txt`-Dateien im Container – schwer zugänglich
- Bei Problemen: zuerst `docker logs`, dann `inspect`, dann `exec`
- In CI: Logs bei Fehlern automatisiert anhängen (z. B. `--no-start`, `--rm`, `tee`)

Effizientes Logging & Debugging macht Container-Entwicklung robuster und spart Zeit bei Fehlersuche.