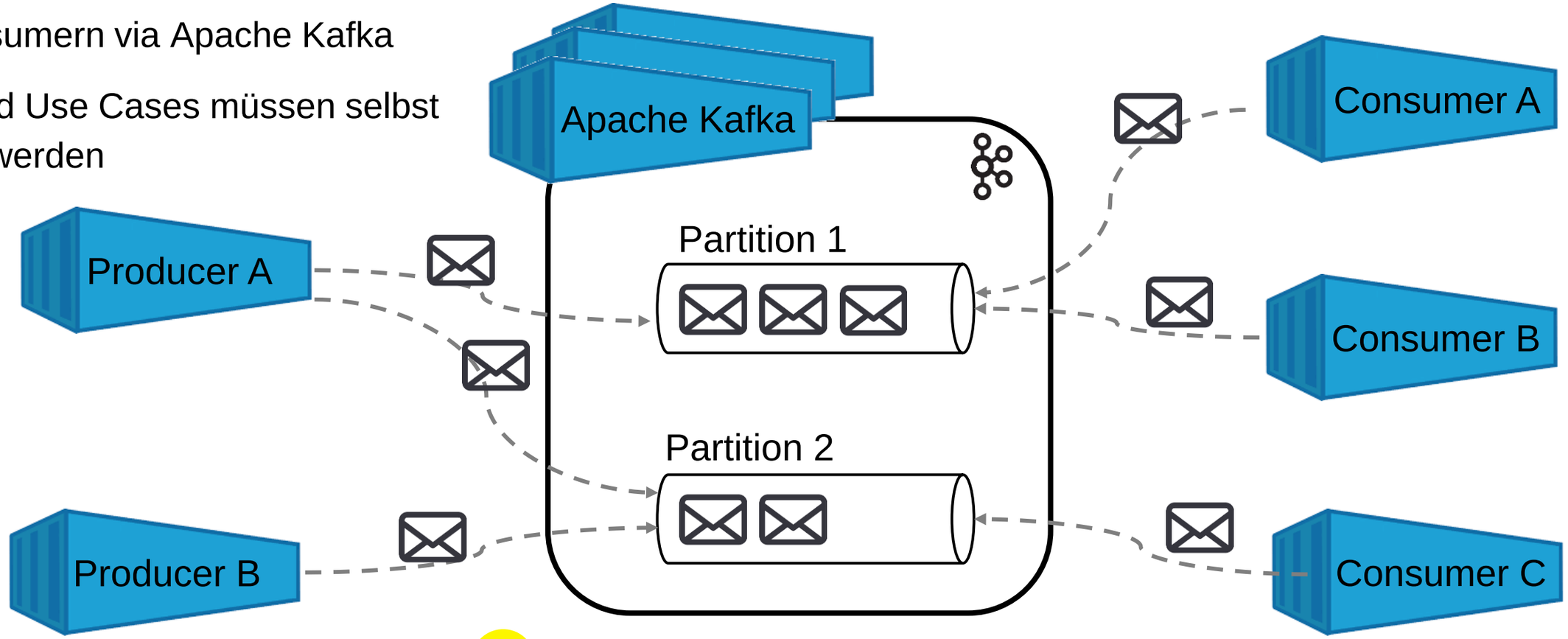
University of Applied Sciences and Arts Northwestern Switzerland
School of Engineering

High Performance Computing

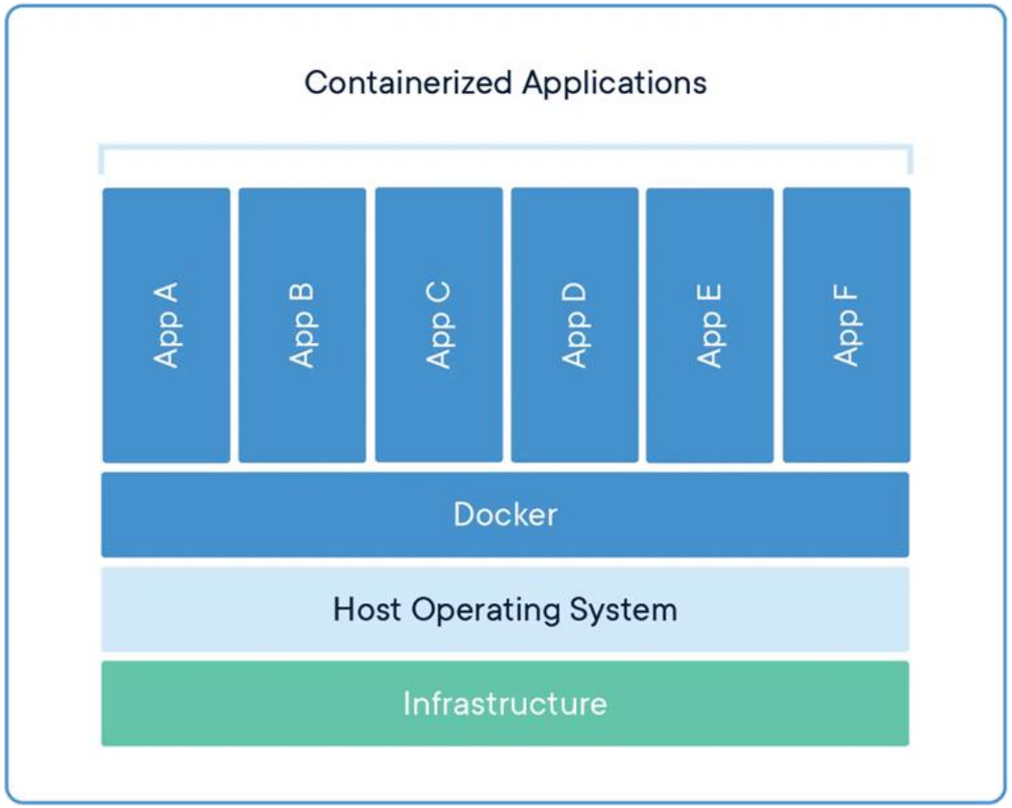
Deep Dive Mini Challenge 1

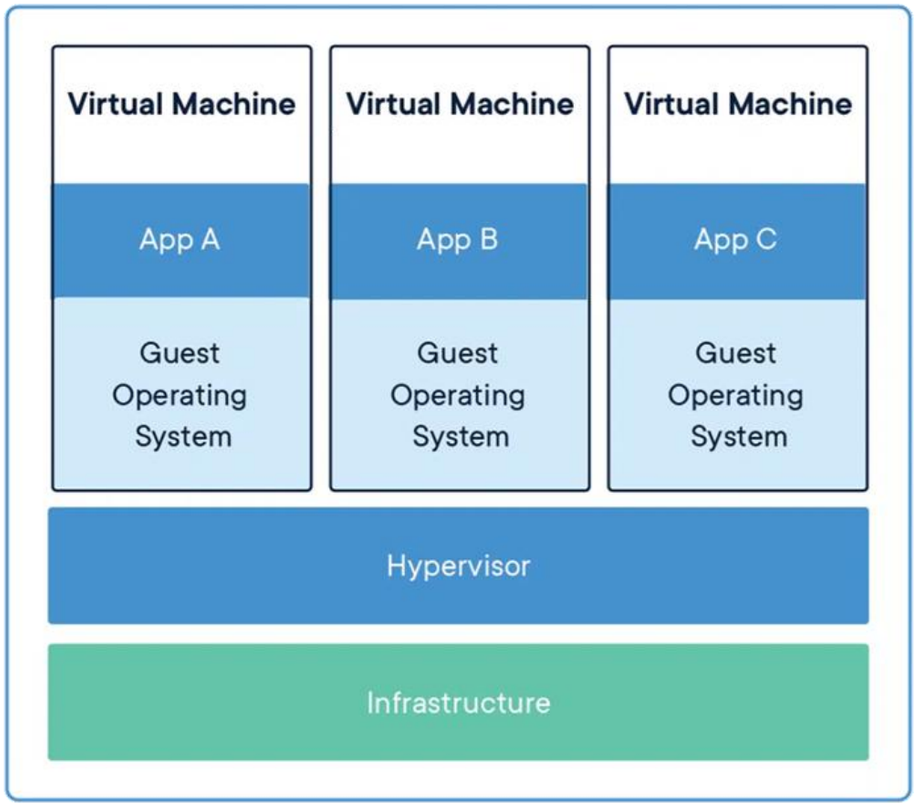
Ziel

- Applikation mit mehreren Producers und Consumern via Apache Kafka
- Daten und Use Cases müssen selbst definiert werden



Containers vs Virtual Machines

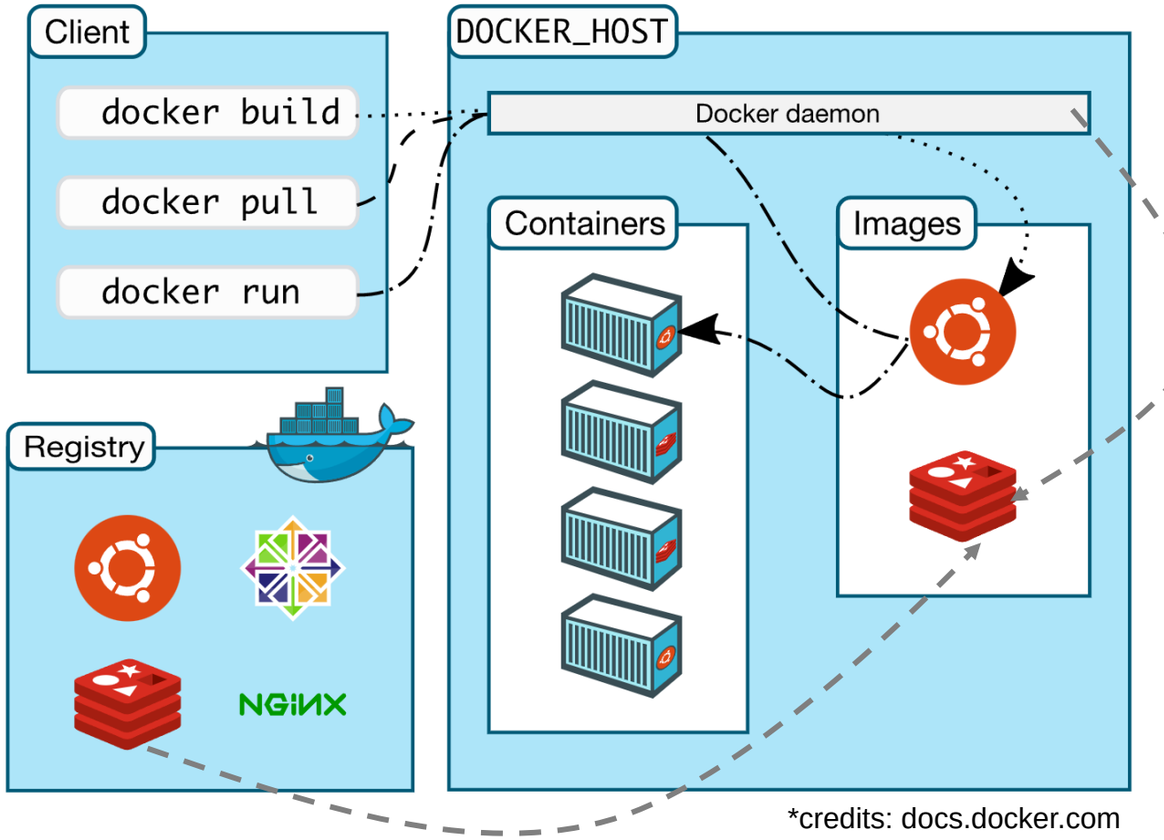




*credits: docs.docker.com

Docker Overview

- Eine Docker Registry bietet Basis-Images an
- Die Images können beliebig ergänz werden
- Docker Images haben ein 1:1 Mapping zum darunter liegenden Filesystemaufbau
- Hierarchisch aufbauendes read-only Filesystem
- Docker Containers können auf den Docker-Host zugreifen, wenn dies spezifiziert ist.
  - Netzwerk Interfaces (IP Adressen)
  - Mount points
  - Hardware wie GPUS



*credits: docs.docker.com

Docker Compose

- Mit Docker Compose können multi-container Applikationen spezifiziert und gestartet werden.
- Spezifikation erfolgt in einem .yml File
- Es können Abhängigkeiten zwischen den Containers abgebildet werden
- Die Container können miteinander kommunizieren

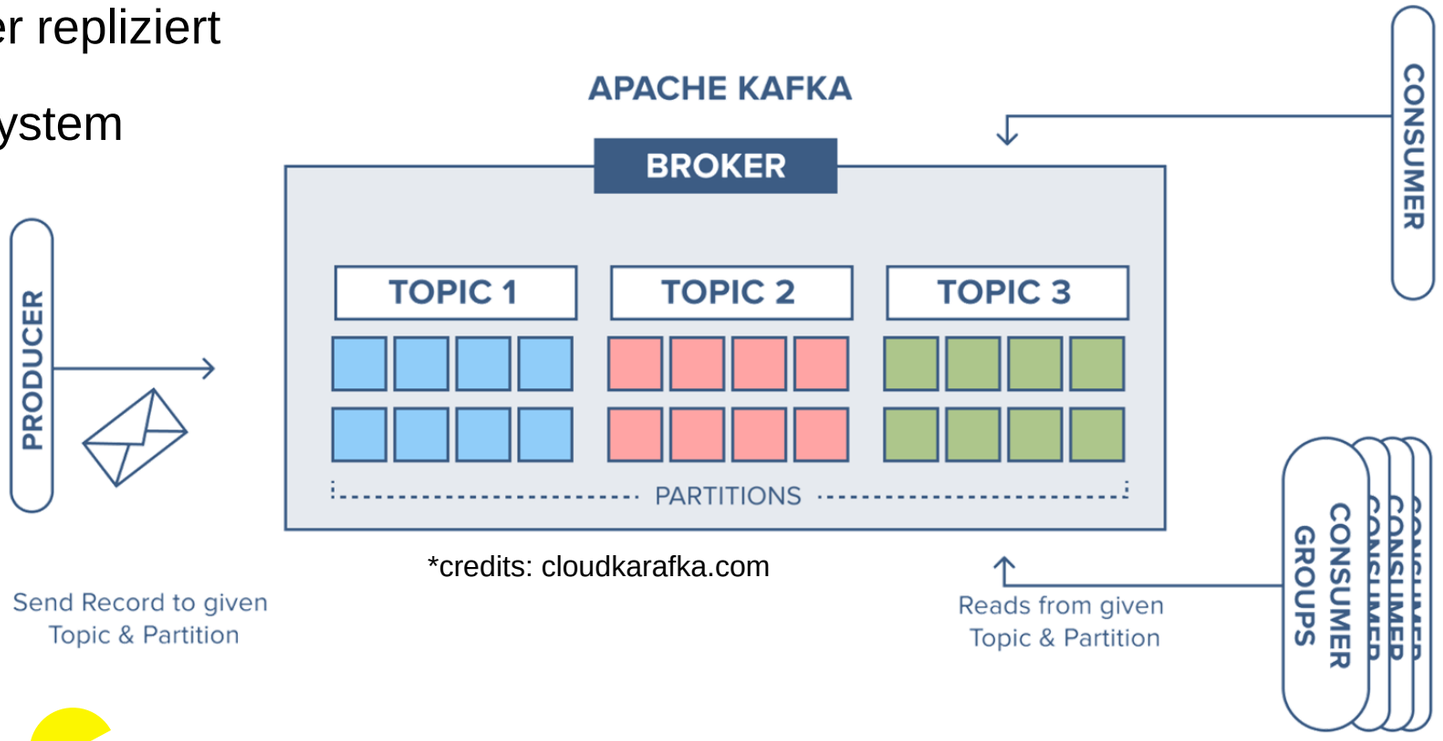
```
services:
  jupyter:
    container_name: jupyter1
    image: scipy-notebook:...
    volumes:
      - ./notebooks:/home/jovyan/
    ports:
      - "8888:8888"

  zookeeper:
    image: confluentinc/cp-zookeeper
    hostname: zookeeper
    container_name: zookeeper1
    ports:
      - "2181:2181"
    environment:
      ZOOKEEPER_CLIENT_PORT: 2181

...
```

Apache Kafka

- **Publish-subscribe** based durable **messaging system**
  - Publish-Subscribe ist ein Kommunikationspattern
  - Die Sender von Nachrichten senden ohne zu wissen wer diese Empfängt
  - Die Nachrichten werden in Topics eingeteilt
  - Die Empfänger abonnieren ein Topic und erhalten alle Nachrichten zu diesem Topic
- **Publish-subscribe based** durable messaging system
  - Kafka ist nicht Pub-Sub! Die Empfänger müssen die Nachrichten abholen (pull).
- Publish-subscribe based **durable** messaging system
  - Nachrichten werden gespeichert, auch nach dem Abholen der Empfänger
  - Nachrichten können mehrfach abgeholt werden, was auch ein «reprocessing» ermöglicht

Apache Kafka

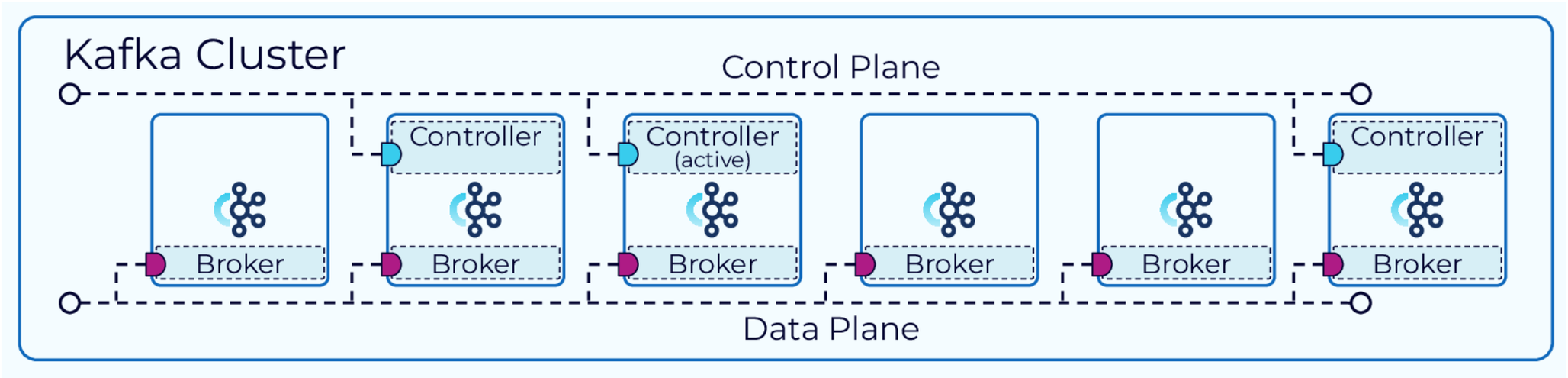
- Nachrichten werden innerhalb eines Topics in mehreren Partitionen unterteilt
- Dies erlaubt das parallele Entnehmen/Verarbeiten von Nachrichten
- Partitionen werden auf mehrere Broker repliziert
- Dies ermöglicht ein fehlertolerantes System



*credits: cloudkarafka.com

Apache Kafka

- Wir verwenden die neue Kafka Architektur KRaft (alt brauchte noch Zookeeper Nodes)
- Ein Node im Cluster kann Controller oder Broker oder Beides sein
- Alle anderen Eigenschaften des Kafka Clusters können – respektive müssen - der eigenen Applikation angepasst werden



*credits: developer.confluent.io

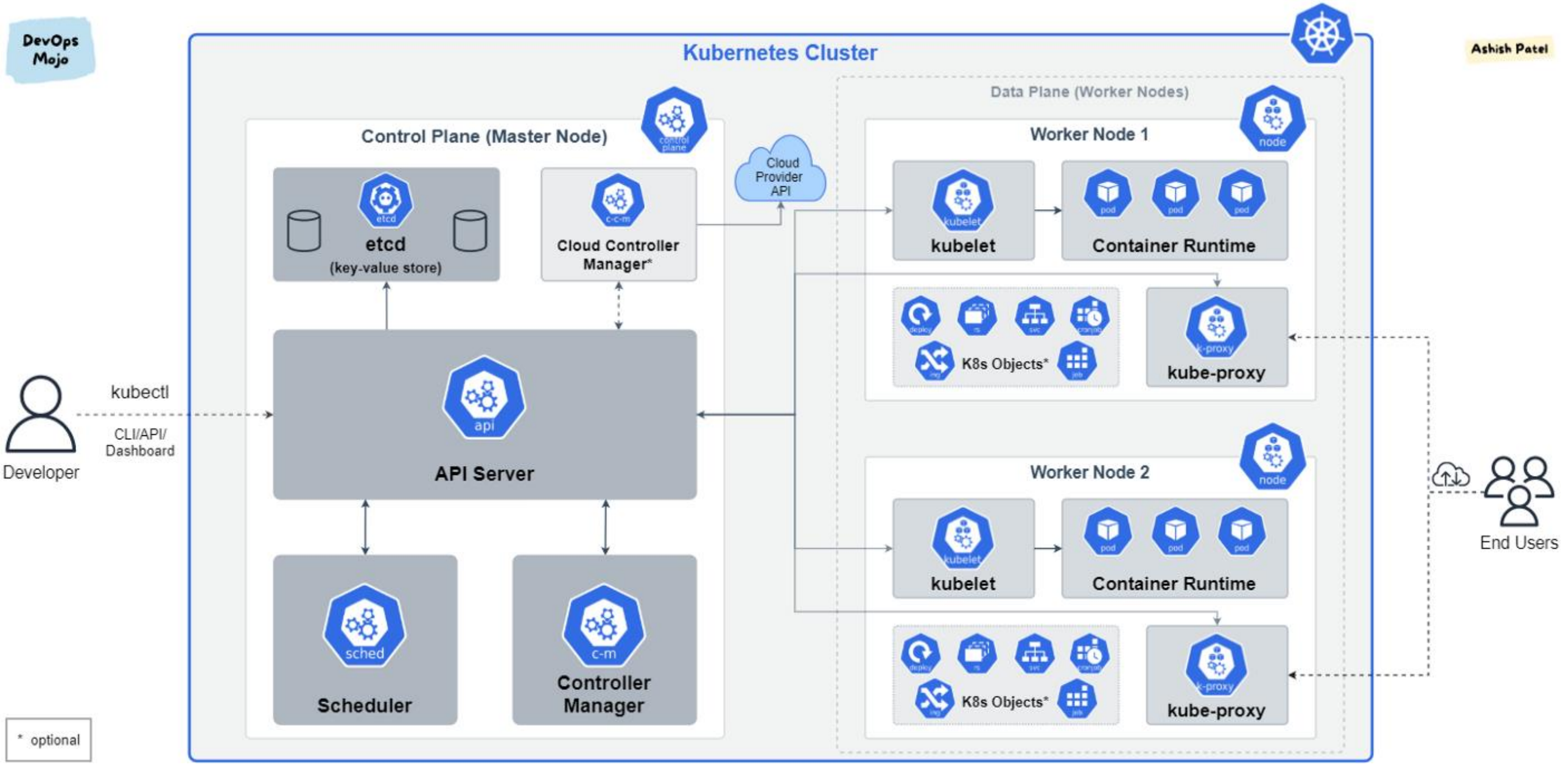
Messaging in HPC

- Classic HPC (grosse homogene Cluster mit low-latency Interconnect)
  - Message Passing Interface (**MPI**)
  - 1:1 Kommunikation
  - Gruppen- und Broadcast Kommunikation
  - Patterns wie Scatter und Gather, Reduce
- HTC oder Cloud Computing (viele loosely-coupled Jobs)
  - Kommunikation über persistente Speicher (z.B. auch Kafka)
  - Micro-Service Architektur mit
    - Dezentraler Kommunikation
    - Via zentrale Broker(-Infrastruktur)
    - Spezielle Kommunikationsservices (AWS SQS, …)

Kubernetes Overview

- Service discovery and load balancing
- Storage orchestration
- Automated rollouts and rollbacks
- Self-healing
- Extremly customizable

- **Not** a CI/CD system
- Does **not** provide application-level service



*credits: medium.com/devops-mojo

Profiling

A **dynamic** program analysis that measures:

- Measure the space (memory) or time complexity of a program
- The usage of particular instructions
- The frequency and duration of function calls.

Challenges:

- Observer effect: The profiling has its own overhead. Can change timings and make the code slower
- Distributed systems

Why do we profile

- **Identify Bottlenecks:** Profiling helps identify the slowest parts of the code that are affecting overall performance.
- **Optimize Performance:** By understanding where the code spends most of its time, developers can optimize these areas for improved performance.
- **Memory Usage Analysis:** Profiling is not just about speed, but also about efficient memory usage, helping prevent memory leaks and excessive memory consumption.
- **Improve Code Quality:** Profiling leads to cleaner, more efficient code, enhancing maintainability and readability.

How do we profile

- **Timers**: like the time and timeit standard library modules
- **Deterministic profilers**: like profile, cProfile, and line_profiler
- **Statistical profilers**: like Pyinstrument and the Linux perf profiler

Good python guide: https://realpython.com/python-profiling/

- **Focus on hotspots**
- **Amdahl’s law is real**