**Answers:**

1. Read about docker swarm and add a brief overview about it in your own words.

Docker swarm is a tool that lets you manage and deploy containers across multiple servers as if they were one. It handles things like distributing workloads, scaling up or down, and keeping services running even if some servers fail.

1. Make a comparison between Docker Swarm and Kubernetes.

Docker swarm is simpler than Kubernetes and more integrated with docker, although k8s offers more features, flexibility, its more powerful and widely adopted. Docker swarm is often preferred for simpler use cases and smaller setups, while k8s is used for large, complex deployments.

1. Start docker swarm on your own computer and deploy mongoDB on the swarm. You will use this mongo on the next exercise (development).

The commands I used:

* + Docker swarm init
  + service create --name mongo --replicas 1 --publish published=27017,target=27017m--mount type=volume,source=mongo-data,target=/data/db mongo:latest

1. Install robomongo (the exe to your computer, not the swarm) and connect it to the robo.

Deployment:

1. Asynchronous programming allows your code to perform tasks in the background, like fetching data or reading files, without waiting for them to complete before moving on. This keeps your program responsive and efficient.
2. A thread is a basic unit of execution within a process. It's like a lightweight version of a process that can run tasks in parallel with other threads within the same application, sharing the same memory space. Threads allow programs to perform multiple operations at once, improving efficiency and responsiveness.
3. A process is an instance of a running program. It includes the program's code, data, and its own memory space. Each process runs independently, with its own resources, and can have multiple threads. Operating systems manage processes to run multiple programs simultaneously.
4. A coroutine is a special type of function that can pause its execution and resume later, allowing for cooperative multitasking. Unlike regular functions, coroutines can be suspended and restarted, making them useful for asynchronous programming, where you want to handle tasks like I/O operations without blocking the main program flow.
5. An async function is a function that allows you to write asynchronous code in a more readable, synchronous-looking way. It returns a promise and can use the await keyword to pause execution until a promise is resolved, making it easier to handle tasks like fetching data without blocking the rest of your code.
6. The await keyword is used inside an async function to pause execution until a promise is resolved, making asynchronous code look and behave like synchronous code. It simplifies handling asynchronous operations by allowing you to write code that waits for a result without blocking other operations.
7. asyncio is a Python library for writing concurrent code using the async and await keywords. It helps manage asynchronous tasks and I/O operations, allowing you to run multiple tasks simultaneously and efficiently without blocking the main program flow.
8. An event loop is a programming construct that continuously checks for and executes tasks or events, such as handling asynchronous operations or I/O events. It manages and schedules the execution of tasks, allowing programs to perform non-blocking operations and handle multiple tasks concurrently.

-----------------------------------------------------------------------------------------------------------------

1. An API (Application Programming Interface) is a set of rules and tools that allows different software applications to communicate with each other. APIs define the methods and data formats that applications use to request and exchange information.

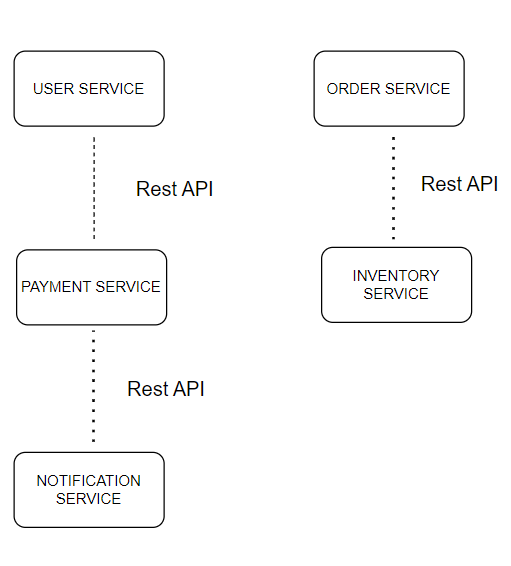
Types of APIs:

* + Web APIs: accessible over the internet, usually using HTTP/HTTPS.
  + Library/Framework APIs: provided by libraries or frameworks that offer predefined functions and classes.
  + Operating System APIs: provided by an operating system to interact with hardware or system services.
  + Database APIs: allow applications to interact with databases to perform operations like queries and updates.

Implementation examples:

* + Rest API: Uses HTTP requests to perform CRUD operations on resources.
  + Graph API: Allows clients to request exactly the data they need and nothing more.

1. An SDK (Software Development Kit) is a collection of tools, libraries, and documentation that helps developers create software applications for a specific platform or service. It typically includes APIs, code samples, and development environments to streamline the process of building and integrating applications.
2. Microservice architecture is a design approach where a software application is divided into small, independent services, each responsible for a specific function or business capability. These services communicate over a network and can be developed, deployed, and scaled individually, allowing for more flexibility and easier maintenance compared to monolithic architectures.
3. The Microservice architecture works that way:
   * **Service Decomposition**: The application is split into discrete services, each handling a specific aspect of the application.
   * **Independent Deployment**: Each service can be deployed independently. This allows for continuous delivery and easy updates without affecting the whole application.
   * **Communication**: Services communicate with each other over a network using APIs (typically REST or messaging queues).
   * **Data Management**: Each service manages its own data storage, which avoids tight coupling and improves scalability.
   * **Scalability**: Services can be scaled independently based on demand.
   * **Fault Isolation**: Failures in one service do not necessarily impact others, improving the overall resilience of the system.

**A diagram** **example**: Each service operates independently but communicates with other services via APIs, allowing for a flexible and scalable architecture.

1. Pros:
   * Scalability: Services can be scaled independently based on demand.
   * Flexibility: Allows the use of diverse technologies and easy updates.
   * Resilience: Failures in one service don’t necessarily affect others.
   * Deployability: Services can be deployed independently, enabling continuous delivery.

Cons:

* + Complexity: Managing multiple services can be complex, requiring sophisticated orchestration and monitoring.
  + Communication Overhead: Inter-service communication can introduce latency and network issues.
  + Data Management: Ensuring consistency across services can be challenging.
  + Deployment: Requires more infrastructure and coordination for deployment and testing.

1. A monolith architecture is a single, unified application where all components are interconnected and deployed together. It's typically a single codebase handling various functionalities.

The key differences between it and microservice architecture:

* Structure:
  + **Monolith**: Single, tightly-coupled application.
  + **Microservices**: Multiple, loosely-coupled services.
* Scalability:
  + **Monolith**: Scaled as a whole unit.
  + **Microservices**: Individual services can be scaled independently.
* Deployment:
  + **Monolith**: Deployed as one unit.
  + **Microservices**: Services can be deployed independently.
* Flexibility:
  + **Monolith**: Limited flexibility in technology and updates.
  + **Microservices**: Allows for diverse technologies and easier updates.
* Complexity:
  + **Monolith**: Simpler to develop initially but can become complex as it grows.
  + **Microservices**: More complex to manage but can be more manageable in large systems.

1. Monolith architecture is suitable for:
   * **Small to Medium Applications**: Easier to manage and develop.
   * **Rapid Prototyping**: Quick to build and deploy.
   * **Single Technology Stack**: When only one technology is used.
   * **Tightly Coupled Components**: Components that work closely together.
   * **Limited Scale**: Applications with small scaling needs.
2. These are the pros and cons of this architecture:

Pros:

**Simplicity**: Easier to develop and deploy as a single unit.

**Unified Codebase**: All components are in one place, simplifying management.

**Performance**: Direct function calls within the same process can be faster.

Cons:

**Scalability**: Difficult to scale individual components independently.

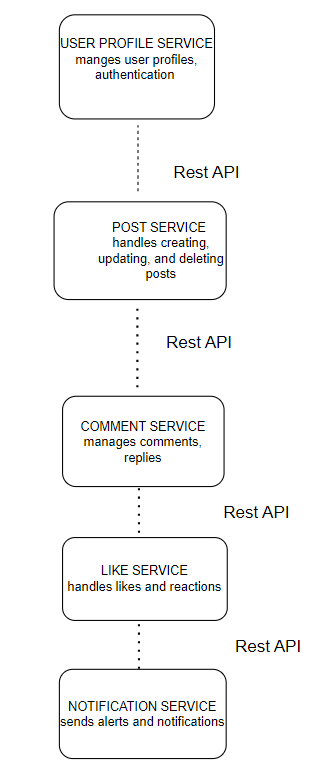
**Flexibility**: Harder to update or modify specific parts without affecting the whole application.

**Complexity**: Can become complex and unwieldy as the application grows.

**Deployment**: Entire application must be redeployed for any changes.

1. A RESTful API uses standard HTTP methods (GET, POST, PUT, DELETE) to interact with resources via URLs. It is stateless, meaning each request must contain all necessary information, and it typically returns data in formats like JSON or XML.
2. 3 examples are:

* **Netflix**: Uses microservices to handle different aspects of streaming, such as user management, recommendations, and content delivery.
* **Amazon**: Employs microservices for various functions like product catalog, order management, and payment processing.
* **Uber**: Utilizes microservices to manage ride requests, driver tracking, payment systems, and customer service.

11.

1. CRUD stands for Create, Read, Update, Delete. It represents the four basic operations performed on data in a database or data storage system.
2. Besides API calls, services can communicate using various methods, including:
   * Message queues: services send messages to a queue, where other services can consume and process them asynchronously.
   * Service buses: a centralized bus facilitates communication between services by routing messages and handling complex interactions.
   * Event streaming: services publish events to a stream, and other services subscribe to and react to these events in real-time.
   * Shard databases: services read from and write to a shared database to exchange data. This method can lead to tight coupling and is less preferred for scalable systems.
   * Remote procedure: services directly invoke methods on other services as if they were local, often using binary protocols for efficiency.
   * File base communication: services read from and write to files in a shared storage location to exchange data.