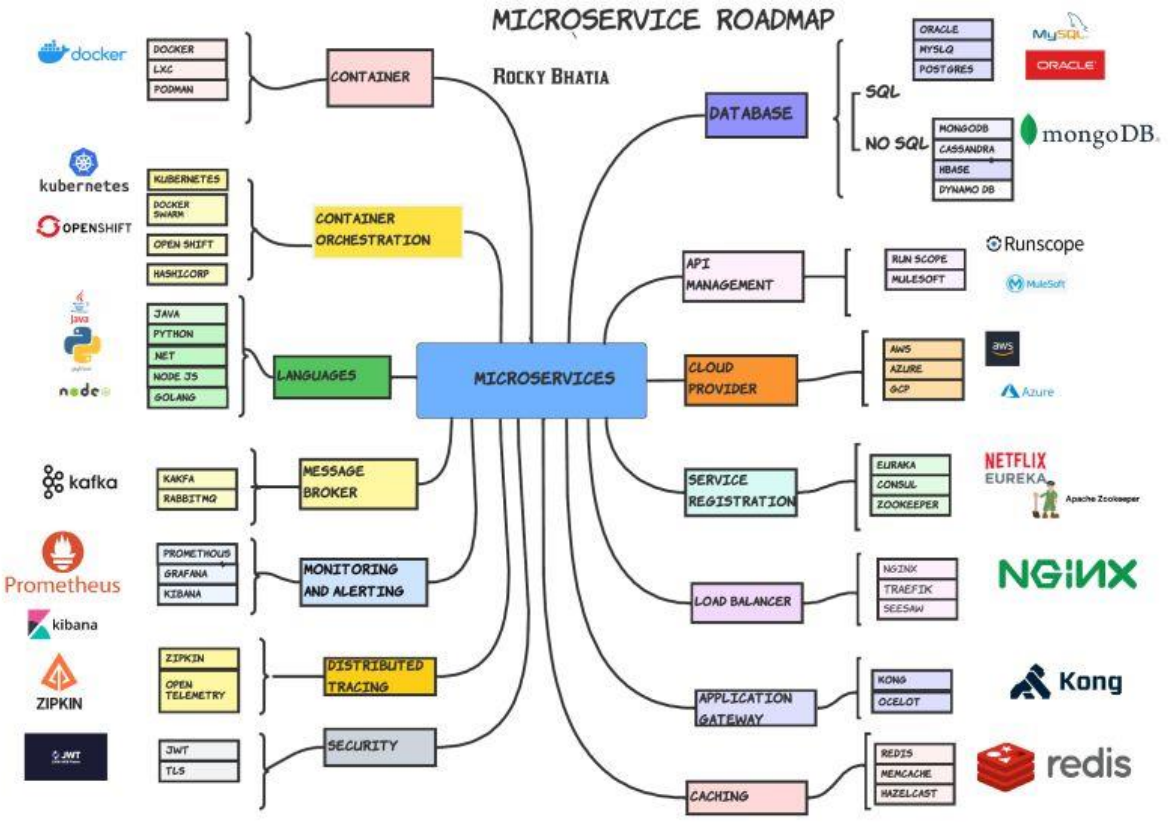
# Microservices




One Diagram to Refer for **Microservices** Roadmap

Companies like Netflix, Amazon, and others have adopted the concept of microservices in their products due to large benefits offered by microservices.

As we understand, many developers want to know how they should start this journey. So I decided to make this journey clearer by defining a road map for this learning curve.

**Container** - A container is a standard unit of software that packages up code and all its dependencies so the application runs quickly and reliably from one computing environment to another.

**Container orchestration** automates containers' deployment, management, scaling, and networking. Enterprises that need to deploy and manage hundreds or thousands of Linux® containers and hosts can benefit from container orchestration.

**Load balancer** is a device that acts as a reverse proxy and distributes network or application traffic across several servers.

Load balancers are used to increase the capacity (concurrent users) and reliability of applications.

**Monitoring and Alerting** : In a microservice architecture, if you want to have a reliable application or service, you have to monitor the functionality, performance, communication, and any other aspect of your application in order to achieve a responsible application. Prometheus is widely popular.

**Distributed Tracing -** when it comes to microservice architecture, a request may be passed through different services, which makes it difficult to debug and trace because the codebase is not in one place, so here distributed tracing tool can be helpful.

**Message Broker** - A message broker is software that facilitates the exchange of messages between applications, systems, and services.

**Database** - in most systems, we need to persist data, because we would need the data for further processes or reporting, etc.

**Caching** - Caching reduces latency in service-to-service communication of microservice architectures.

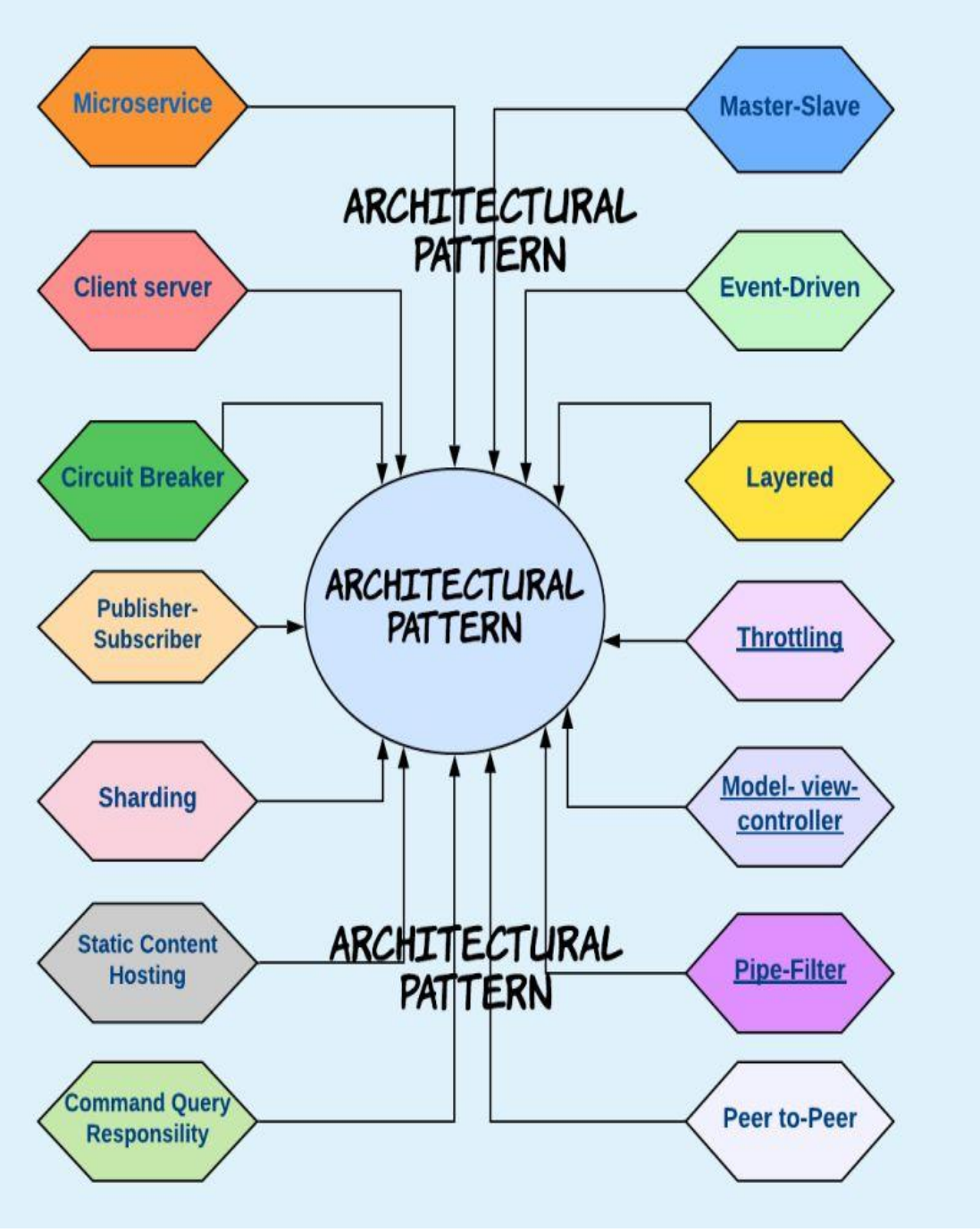
**Cloud service provider** -is a third-party company offering a cloud-based platform, infrastructure, application, or storage services.

**API Management**: API management is the process of designing, publishing, documenting, and analyzing APIs in a secure environment.

**Application Gateway** -An application gateway or application-level gateway (ALG) is a firewall proxy that provides network security. It filters incoming node traffic to certain specifications, meaning only transmitted network application data is filtered.

**Service Registration** -A service registry is a database used to keep track of the available instances of each microservice in an application. The service registry needs to be updated each time a new service comes online and whenever a service is taken offline or becomes unavailable.

# Microservices Architecture




One Diagram for Software Architectural Design Pattern

In software engineering, a software design pattern is a general, reusable solution of how to solve a common problem when designing an application or system. Unlike a library or framework, which can be inserted and used right away, a design pattern is more of a template to approach the problem at hand.

14 Important and widely used design Patterns to follow:

**1**. **Static Content Hosting**: used to optimize webpage loading time. It stores static content (information that doesn't change often, like an author's bio or an MP3 file) separately from dynamic content (like stock prices).

**2**. **Peer-to-Peer**: Involves multiple components called Peers, where a pear may function both as a client, requesting services from other peers, and as a server, providing services to other peers.

**3**. **Publisher-Subscriber**: Used to send (publishes) relevant messages to places that have subscribed to a topic.

**4**. **Sharding Pattern** – Used to partition data in a database to speed commands or queries.

**5**. **Circuit Breaker**: Helps make systems more fault tolerant by minimizing the effects of a hazard by rerouting traffic to another service.

**6**. **Layered Pattern**: Generally used to develop applications that include groups of subtasks that execute in a specific order.

**7**. **Client-Server**: A peer-to-peer architecture that is comprised of a client, which requests a service, and a server, which provides the service.

**8**. **Master-Slave**: Consists of two components; the master distributes the work among identical slaves and computes a final result from the results which the slaves return.

**9**. **Pipe-Filter**: Used to structure systems that produce and process a stream of data.

**10**. **Event-Driven**: Uses events to trigger and communicate between decoupled services.

**11**. **Model-View-Controller**: Divides an application into three components. The model contains the application's data and main functionality; the view displays data and interacts with the user; and the controller handles user input and acts as the mediator between the model and the view.

**12**. **Throttling** - pattern controls how fast data flows into a target. It's often used to prevent failure during a distributed denial of service attack or to manage cloud infrastructure costs..

**13**. **Microservices**: Used to create multiple services that work interdependently to create a larger application.

**14**. **Command Query Responsibility Segregation**: Used to separate read and write activities to provide greater stability, scalability, and performance.