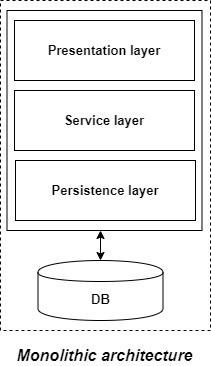
**MICROSERVICES**

**Monolithic Application**

* If all the functionalities of a project exist in a single codebase, then that application is known as a monolithic application.
* We design our application in various layers like presentation, service, and persistence and then deploy that codebase as a single jar/war file.
* **“mono”**represents the single codebase containing all the required functionalities.



**Advantages of monolithic applications:**

* Simple to develop relative to microservices.
* Easier to deploy as only a single jar/war file is deployed.
* Relatively easier and simple to develop in comparison to microservices architecture.

**Disadvantages of Monolithic applications:**

* It becomes too large with time and hence, difficult to manage.
* We need to redeploy the whole application, even for a small change.
* As the size of the application increases, its start-up and deployment time also increases.
* It is not very reliable, as a single bug in any module can bring down the entire monolithic application.
* Building Problem: - Developers has to communicate.

**What are Microservices?**

* Microservices are a modern approach to software whereby application code is delivered in small, manageable pieces, independent of others.
* Microservices are the small services that work together.
* Instead of sharing a single database with other microservices, each microservice has its own database.
* Another advantage of having a separate database per microservice is that each microservice can use the type of database best suited for its needs.
* Each service offers a secure module boundary so that different services can be written in different programming languages.
* here are many patterns involved in microservice architecture like service discovery & registry, caching, API gateway & communication, observability, security, etc.

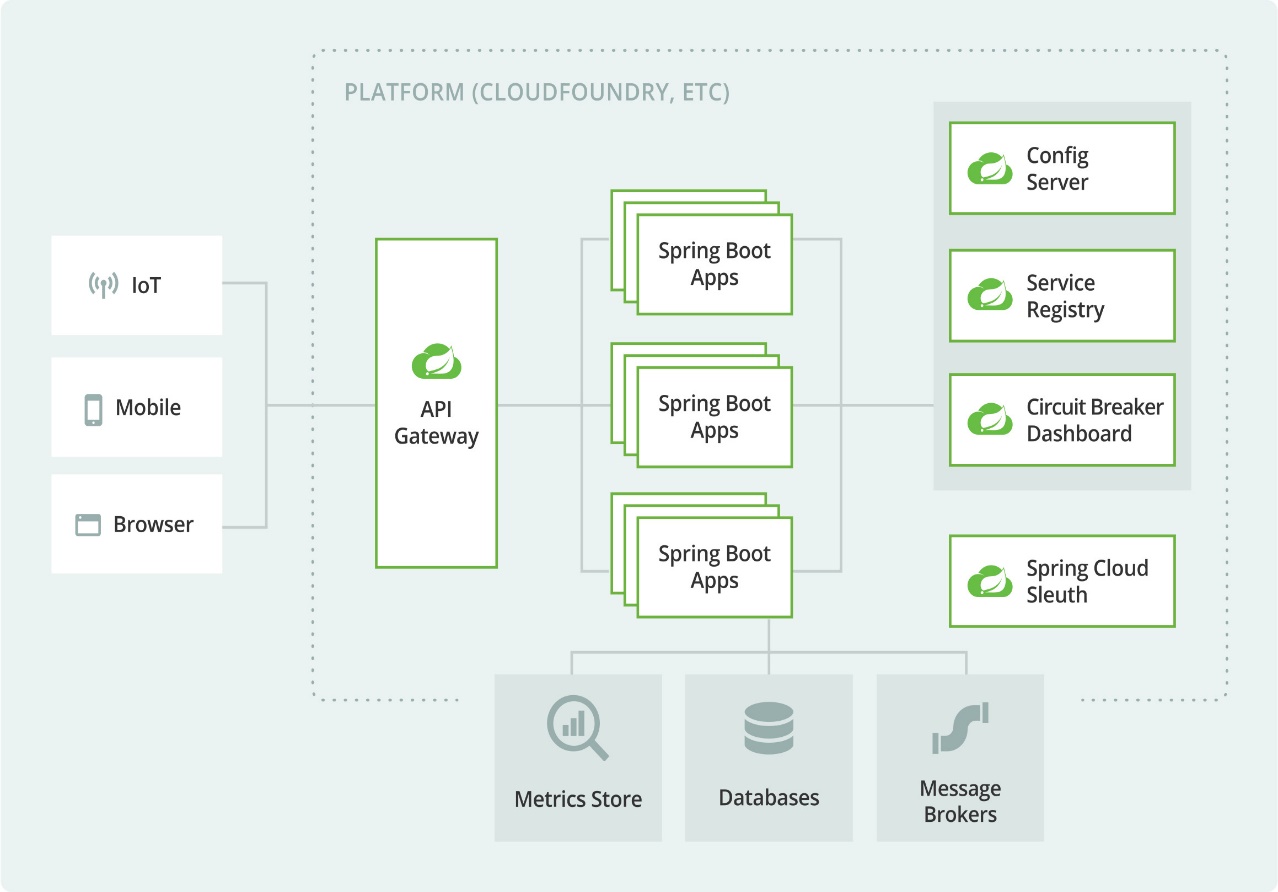
****

Figure Microservices Architecture

**Advantages of microservices:**

* It is easy to manage as it is relatively smaller.
* If there’s any update in one of the microservices, then we need to redeploy only that microservice.
* Microservices are self-contained and, hence, deployed independently. Their start-up and deployment times are relatively less.
* Each microservice can use different technology based on the business requirements.
* If a particular microservice goes down due to some bug, then it doesn’t affect other microservices and the whole system remains intact and continues providing other functionalities to the users.
* If a particular microservice is facing a large load because of the users using that functionality in excess, then we need to scale out that microservice only. Hence, the microservices architecture supports horizontal scaling.

**Disadvantages of microservices:**

* Microservices are less secure relative to monolithic applications due to the inter-services communication over the network.
* Debugging is difficult as the control flows over many microservices and to point out why and where exactly the error occurred is a difficult task.
* Skilled developers are required to work with microservices architecture, which can identify the microservices and manage their inter-communications.
* Independent deployment of microservices is complicated.

**Microservices Roadmap:**

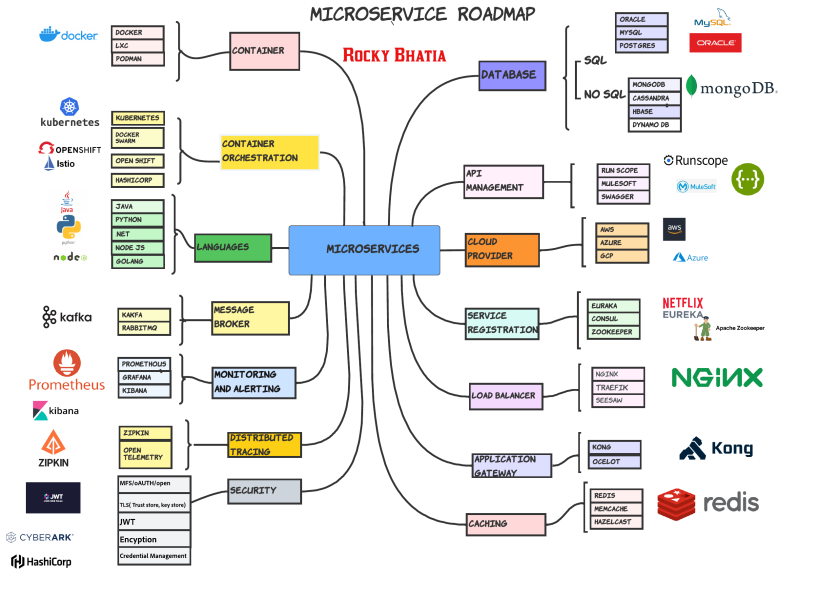




Figure Microservices Roadmap

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

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.

Though there are many companies in this area, but docker has become the de facto standard in the industry

A Docker container image is a lightweight, standalone, executable package of software that includes everything needed to run an application: code, runtime, system tools, system libraries and settings.

Container images become containers at runtime and in the case of Docker containers – images become containers when they run on Docker Engine. Available for both Linux and Windows-based applications, containerized software will always run the same, regardless of the infrastructure. Containers isolate software from its environment and ensure that it works uniformly despite differences for instance between development and staging.

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.

Kubernetes is the de facto standard for container orchestration

𝗞𝘂𝗯𝗲𝗿𝗻𝗲𝘁𝗲𝘀 is a portable, extensible, open-source platform for managing containerized workloads and services, that facilitates both declarative configuration and automation. It has a large, rapidly growing ecosystem. Kubernetes services, support, and tools are widely available.

The name Kubernetes originates from Greek, meaning helmsman or pilot. K8s as an abbreviation result from counting the eight letters between the "K" and the "s". Google open-sourced the Kubernetes project in 2014. Kubernetes combines over 15 years of Google's experience running production workloads at scale with best-of-breed ideas and practices from the community.

Containers are a good way to bundle and run your applications. In a production environment, you need to manage the containers that run the applications and ensure that there is no downtime. For example, if a container goes down, another container needs to start. Wouldn't it be easier if this behaviour was handled by a system?

That's how Kubernetes comes to the rescue! Kubernetes provides you with a framework to run distributed systems resiliently. It takes care of scaling and failover for your application, provides deployment patterns, and more. For example: Kubernetes can easily manage a canary deployment for your system.

C𝐥𝐨𝐮𝐝 𝐬𝐞𝐫𝐯𝐢𝐜𝐞 𝐩𝐫𝐨𝐯𝐢𝐝𝐞𝐫 is a third-party company offering a cloud-based platform, infrastructure, application, or storage services. Much like a homeowner would pay for a utility such as electricity or gas, companies typically have to pay only for the amount of cloud services they use, as business demands require.

Besides the pay-per-use model, cloud service providers also give companies a wide range of benefits. Businesses can take advantage of scalability and flexibility by not being limited to physical constraints of on-premises servers, the reliability of multiple data centres with multiple redundancies, customization by configuring servers to your preferences, and responsive load balancing that can easily respond to changing demands. Though businesses should also evaluate security considerations of storing information in the cloud to ensure industry-recommended access and compliance management configurations and practices are enacted and met.

An important factor to consider during this process is, what cloud technologies will you be able to handle within your enterprise, and which should be delegated to a cloud service provider?

Having infrastructure, platforms, or software managed for you can free your business to serve your clients, be more efficient in overall operations, and improve or expand your development operations (DevOps) strategy.

Many public cloud service providers have a set of standard support contracts that include validating active software subscriptions, resolving issues, maintaining security, and deploying patches. Managed cloud service providers' support could be relegated to simple cloud administration or it can serve the needs of an entire IT department.

𝗟𝗼𝗮𝗱 𝗯𝗮𝗹𝗮𝗻𝗰𝗶𝗻𝗴 refers to efficiently distributing incoming network traffic across a group of backend servers, also known as a server farm or server pool.

Modern high‑traffic websites must serve hundreds of thousands, if not millions, of concurrent requests from users or clients and return the correct text, images, video, or application data, all in a fast and reliable manner. To cost‑effectively scale to meet these high volumes, modern computing best practice generally requires adding more servers.

A load balancer acts as the “traffic cop” sitting in front of your servers and routing client requests across all servers capable of fulfilling those requests in a manner that maximizes speed and capacity utilization and ensures that no one server is overworked, which could degrade performance. If a single server goes down, the load balancer redirects traffic to the remaining online servers. When a new server is added to the server group, the load balancer automatically starts to send requests to it.

In this manner, a load balancer performs the following functions:

* Distributes client requests or network load efficiently across multiple servers
* Ensures high availability and reliability by sending requests only to servers that are online
* Provides the flexibility to add or subtract servers as demand dictates

𝗠𝗼𝗻𝗶𝘁𝗼𝗿𝗶𝗻𝗴 𝗮𝗻𝗱 𝗔𝗹𝗲𝗿𝘁𝗶𝗻𝗴 : Understanding the state of your infrastructure and systems is essential for ensuring the reliability and stability of your services. Information about the health and performance of your deployments not only helps your team react to issues, it also gives them the security to make changes with confidence. One of the best ways to gain this insight is with a robust monitoring system that gathers metrics, visualizes data, and alerts operators when things appear to be broken.

Metrics, monitoring, and alerting are all interrelated concepts that together form the basis of a monitoring system. They have the ability to provide visibility into the health of your systems, help you understand trends in usage or behavior, and to understand the impact of changes you make. If the metrics fall outside of your expected ranges, these systems can send notifications to prompt an operator to take a look, and can then assist in surfacing information to help identify the possible causes.

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. Promethous is widely popular.

𝗗𝗶𝘀𝘁𝗿𝗶𝗯𝘂𝘁𝗲𝗱 𝗧𝗿𝗮𝗰𝗶𝗻𝗴 - Moving your applications from a monolithic design to a microservices-oriented design introduces several advantages during development and in operations. However, that move has a price. New challenges are introduced, as traditional metrics and log information tend to be captured and recorded in a component and machine-centric way. When your components are spread across machines and physical locations and are subject to dynamic horizontal scaling over transient compute units, traditional tools to capture and analyze information become powerless.

Distributed tracing is a technique that addresses logging information in microservice-based applications. A unique transaction ID is passed through the call chain of each transaction in a distributed topology. One example of a transaction is a user interaction with a website. The unique ID is generated at the entry point of the transaction. The ID is then passed to each service that is used to finish the job and written as part of the services log information. It's equally important to include timestamps in your log messages along with the ID. The ID and timestamp are combined with the action that a service is taking and the state of that action.

Unique identifiers, such a transaction IDs and user IDs, are helpful when you gather analytics and debug. Unique IDs can point you to the exact transaction that failed. Without them, you must look at all the information that the entire application logged in the time frame when your problem occurred. After you implement the generation and usage of the unique ID in your logs, you can use the unique ID in several ways and 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 enables applications, systems, and services to communicate with each other and exchange information. The message broker does this by translating messages between formal messaging protocols. This allows interdependent services to “talk” with one another directly, even if they were written in different languages or implemented on different platforms.

Message brokers can validate, store, route, and deliver messages to the appropriate destinations. They serve as intermediaries between other applications, allowing senders to issue messages without knowing where the receivers are, whether or not they are active, or how many of them there are. This facilitates decoupling of processes and services within systems.

In order to provide reliable message storage and guaranteed delivery, message brokers often rely on a substructure or component called a message queue that stores and orders the messages until the consuming applications can process them. In a message queue, messages are stored in the exact order in which they were transmitted and remain in the queue until receipt is confirmed.

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

Choosing which database to use is one of the most important decisions we can make when starting working on a new app or website,microservice or any backend system.

If you realize down the line that you’ve made the wrong choice, migrating to another database is very costly and sometimes more complex to do with zero downtime.

Taking time to make an informed choice of database technology upfront can be a valuable early decision for your application.

Data can be structure semi-structure(json, xml, etc.), and unstructure (blob, image).In the case of structure, they can be relational or columnar, while in the case of semi-structured, there is a wide range of possibilities, from key-value to graph.

RDBMS scale vertically. We need to upgrade hardware (more powerful CPU, higher storage capacity) to handle the increasing load.

NoSQL datastores scale horizontally. NoSQL is better at handling partitioned data, so you can scale by adding more machines.

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

A microservice needs to know the location (IP address and port) of every service it communicates with. If we don’t employ a Service Discovery mechanism, service locations become coupled, leading to a system that’s difficult to maintain. We could wire the locations or inject them via configuration in a traditional application, but it isn’t recommended in a modern cloud-based application of this kind.

Dynamically determining the location of an application service isn’t a trivial matter. Things become more complicated when we consider an environment where we’re constantly destroying and distributing new instances of services. This may well be the case for a cloud-based application that’s continuously changing due to horizontal autoscaling to meet peak loads, or the release of a new version. Hence, the need for a Service Discovery mechanism.

𝗖𝗮𝗰𝗵𝗶𝗻𝗴 - Caches reduce latency and service-to-service communication of microservice architectures. A cache is a high-speed data storage layer that stores a subset of data. When data is requested from a cache, it is delivered faster than if you accessed the data’s primary storage location.

you should cache only data that is frequently accessed and / or relatively stale. Typically, you should cache objects and application-wide settings. The objects can be business entities or objects that hold frequently accessed and relatively stale data. You can also cache application-specific settings.

Cache strategies could be

* Read through
* write-through.
* Write-back

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

Cloud service providers are companies that establish public clouds, manage private clouds, or offer on-demand cloud computing components (also known as cloud computing services) like Infrastructure-as-a-Service (IaaS), Platform-as-a-Service (PaaS), and Software-as-a-Service(SaaS). Cloud services can reduce business process costs when compared to on-premise IT.

Using a cloud service provider is a helpful way to access computing services that you would otherwise have to provide on your own, such as:

Infrastructure: The foundation of every computing environment. This infrastructure could include networks, database services, data management, data storage (known in this context as cloud storage), servers (cloud is the basis for serverless computing), and virtualization.

Platforms: The tools needed to create and deploy applications. These platforms could include operating systems like Linux®, middleware, and runtime environments.

Software: Ready-to-use applications. This software could be custom or standard applications provided by independent service providers.

An important factor to consider during this process is, what cloud technologies will you be able to handle within your enterprise, and which should be delegated to a cloud service provider?

Having infrastructure, platforms, or software managed for you can free your business to serve your clients, be more efficient in overall operations, and improve or expand your development operations (DevOps) strategy.

Many public cloud service providers have a set of standard support contracts that include validating active software subscriptions, resolving issues, maintaining security, and deploying patches. Managed cloud service providers' support could be relegated to simple cloud administration or it can serve the needs of an entire IT department.

𝗔𝗣𝗜 𝗠𝗮𝗻𝗮𝗴𝗲𝗺𝗲𝗻𝘁:it is the process of designing, publishing, documenting and analyzing APIs in a secure environment. Through an API management solution, an organization can guarantee that both the public and internal APIs they create are consumable and secure.

API management solutions in the market can offer a variety of features; however, the majority of API management solutions allow users to perform the following tasks:

API management solutions in the market can offer a variety of features; however, the majority of API management solutions allow users to perform the following tasks:

API design - API management solutions provide users – from developers to partners – the ability to design, publish and deploy APIs as well as record documentation, security policies, descriptions, usage limits, runtime capabilities and other relevant information.

API gateway - API management solutions also serve as an API gateway, which acts as a gatekeeper for all APIs by enforcing relevant API security policies and requests and also guarantees authorization and security.

API store - API management solutions provide users with the ability to keep their APIs in a store or catalog where they can expose them to internal and/or external stakeholders. This API “store” then serves as a marketplace for APIs, where users can subscribe to APIs, obtain support from users and the community and so on.

API analytics - API management allow users to monitor API usage, load, transaction logs, historical data and other metrics that better inform the status as well as the success of the APIs available.

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.

Application Gateway can make routing decisions based on additional attributes of an HTTP request, for example URI path or host headers. For example, you can route traffic based on the incoming URL. So if /images is in the incoming URL, you can route traffic to a specific set of servers (known as a pool) configured for images. If /video is in the URL, that traffic is routed to another pool that's optimized for videos.