When developing a server-side application you can start it with a modular hexagonal or layered architecture which consists of different types of components:

- Presentation — responsible for handling HTTP requests and responding with either HTML or JSON/XML (for web services APIs).
- Business logic — the application's business logic.
- Database access — data access objects responsible for access the database.
- Application integration — integration with other services (e.g. via messaging or REST API).

Despite having a logically modular architecture, the application is packaged and deployed as a monolith. **Benefits of Monolithic Architecture**

- Simple to develop.
- Simple to test. For example, you can implement end-to-end testing by simply launching the application and testing the UI with Selenium.
- Simple to deploy. You just have to copy the packaged application to a server.
- Simple to scale horizontally by running multiple copies behind a load balancer.

In the early stages of the project it works well and basically most of the big and successful applications which exist today were started as a monolith.

Drawbacks of Monolithic Architecture

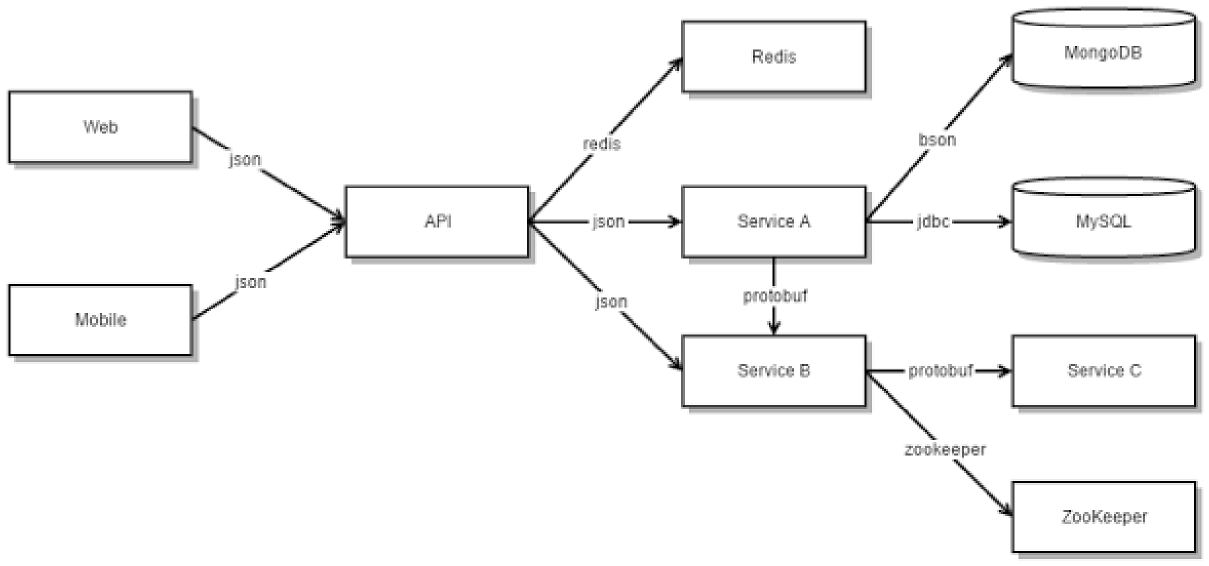
- This simple approach has a limitation in size and complexity.
- Application is too large and complex to fully understand and made changes fast and correctly.
- The size of the application can slow down the start-up time.
- You must redeploy the entire application on each update.
- Impact of a change is usually not very well understood which leads to do extensive manual testing.
- Continuous deployment is difficult.
- Monolithic applications can also be difficult to scale when different modules have conflicting resource requirements.
- Another problem with monolithic applications is reliability. Bug in any module (e.g. memory leak) can potentially bring down the entire process. Moreover, since all instances of the application are identical, that bug will impact the availability of the entire application.
- Monolithic applications has a barrier to adopting new technologies. Since changes in frameworks or languages will affect an entire application it is extremely expensive in both time and cost.

Microservices Architecture

The idea is to split your application into a set of smaller, interconnected services instead of building a single monolithic application. Each microservice is a small application that has its own hexagonal architecture consisting of business logic along with various adapters. Some microservices would expose a REST, RPC or message-based API and most services consume APIs provided by other services. Other microservices might implement a web UI.

The Microservice architecture pattern significantly impacts the relationship between the application and the database. Instead of sharing a single database schema with other services, each service has its own database schema. On the one hand, this approach is at odds with the idea of an enterprise-wide data model. Also, it often results in duplication of some data. However, having a database schema per service is essential if you want to benefit from microservices, because it ensures loose coupling. Each of the services has its own database. Moreover, a service can use a type of database that is best suited to its needs, the so-called polyglot persistencearchitecture.

Some APIs are also exposed to the mobile, desktop, web apps. The apps don’t, however, have direct access to the back-end services. Instead, communication is mediated by an intermediary known as an API Gateway. The API Gateway is responsible for tasks such as load balancing, caching, access control, API metering, and monitoring.



The Microservice architecture pattern corresponds to the Y-axis scaling of the Scale Cube model of scalability.

Benefits of Microservices Architecture

- It tackles the problem of complexity by decomposing application into a set of manageable services which are much faster to develop, and much easier to understand and maintain.
- It enables each service to be developed independently by a team that is focused on that service.
- It reduces barrier of adopting new technologies since the developers are free to choose whatever technologies make sense for their service and not bounded to the choices made at the start of the project.
- Microservice architecture enables each microservice to be deployed independently. As a result, it makes continuous deployment possible for complex applications.

- Microservice architecture enables each service to be scaled independently.

Drawbacks of Microservices Architecture

- Microservices architecture adding a complexity to the project just by the fact that a microservices application is a distributed system. You need to choose and implement an inter-process communication mechanism based on either messaging or RPC and write code to handle partial failure and take into account other fallacies of distributed computing.
- Microservices has the partitioned database architecture. Business transactions that update multiple business entities in a microservices-based application need to update multiple databases owned by different services. Using distributed transactions is usually not an option and you end up having to use an eventual consistency-based approach, which is more challenging for developers.
- Testing a microservices application is also much more complex then in case of monolithic web application. For a similar test for a service you would need to launch that service and any services that it depends upon (or at least configure stubs for those services).
- It is more difficult to implement changes that span multiple services. In a monolithic application you could simply change the corresponding modules, integrate the changes, and deploy them in one go. In a Microservice architecture you need to carefully plan and coordinate the rollout of changes to each of the services.
- Deploying a microservices-based application is also more complex. A monolithic application is simply deployed on a set of identical servers behind a load balancer. In contrast, a

microservice application typically consists of a large number of services. Each service will have multiple runtime instances. And each instance need to be configured, deployed, scaled, and monitored. In addition, you will also need to implement a service discovery mechanism. Manual approaches to operations cannot scale to this level of complexity and successful deployment a microservices application requires a high level of automation.