# Questions for self-check:

1. What are the cons and pros of the Monolith architectural style?

**Pros**:

* Simple to develop
* Simple to test
* Simple to deploy
* Simple to scale horizontally by running multiple copies behind a load balancer.

**Cons**:

* The 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.
* Each update you must relaunch the entire application.
* Continuous deployment is difficult.
* Difficult to scale when different modules have conflicting resources requirements.
* Has a barrier to adopting new technologies. Since changes in frameworks or languages will affect an entire application.

1. What are the cons and pros of the Microservices architectural style?

**Pros**:

* 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.
* Enables each service to be developed independently by a team that is focused on that service.
* 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.
* Enables each service to be scaled independently.

**Cons**:

* It adds a complexity to the project just by the fact that microservices application is a distributed system.
* Due to microservices has the partitioned database architecture, 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 microservices is also more complex.
* It is more difficult to implement changes that span multiple services.
* Deploying is also more complex.

1. What is the difference between SOA and Microservices?

SOA is an integration architectural style and an enterprise-wide concept. It enables existing applications to be exposed over loosely coupled interfaces, each corresponding to a business function, that enables applications in one part of an extended enterprise to reuse functionality in other applications.

Microservices architecture is an application architectural style and an application-scoped concept. It enables the internals of a single application to be broken up into small pieces that can be independently changed, scaled, and administered. It does not define how applications talk to one another.

The main difference comes down to scope. SOA has an enterprise scope, while the microservices architecture has an application scope.

Another difference is that microservices use lightweight messaging protocols like HTTP/REST and SOA are more open to heterogeneous messaging protocols such as SOAP, AMQP, MSMQ.

1. What does hybrid architectural style mean? Think of your current and previous projects and try to describe which architectural styles they most likely followed.

It means to mix two or more architectural patterns to take advantage of their benefits and reduce the drawbacks of each one according to the application requirements.

For example, we can create a whole system with the microservices architecture, but each microservice can be designed in layers or in another pattern adapting to the requirements in a better way.

1. Name several examples of the distributed architectures. What do ACID and BASE terms mean.
   * Service-oriented
   * Microservices
   * Serverless
   * Peer-to-peer

**ACID** (Atomic consistent isolated durable) model provides a consistent system. The ACID database transaction model ensures that a performed transaction is always consistent.

**BASE** (Basically available soft stat eventually consistent) model provides high availability. Rather than enforcing immediate consistency, BASE-modelled NoSQL databases will ensure availability of data by spreading and replicating it across the nodes of the database cluster. The BASE model breaks off with the concept of a database which enforces its own consistency, delegating that responsibility to developers.

The fact that BASE does not enforce immediate consistency does not mean that it never achieves it. However, until it does, data reads are still possible (even though they might not reflect the reality).

1. Name several use cases where Serverless architecture would be beneficial.

In those cases where you don’t want to manage servers, databases and even application logic.

Also, in cases where you want to reduce development cost such Auth0. Firebase’s database service. Those services allow us to integrate ready-built authentication and database functionality into our application.

Cases where you want to reduce costs in scaling as the provider makes horizontal scaling and completely automatic and elastic.

# Home task

Diagram

Description automatically generated

**Task 1**

Determine the architectural style of the solution. Provide an explanation for your choice.

In the example we have three monolithic applications.

* Warehouse App
* Backoffice
* Online Shop

A two-tier application; Mobile App Service first tier and the mobile apps as the second tier.

And a 3rd party reporting service which connects to the same database of all the other applications.

It is not an architecture style pattern defined for the whole system so it can be improved in that aspect.

**Task 2**

Draw the architectural diagram of the solution as if you were designing from scratch. Provide pros and cons of the two solutions.

Diagram

Description automatically generated

* In the purposed microservices architecture now we have consistent Business Logic defining a service for each part of the system and reusing it where it is required. For example, in the first version if an entity like the order changes, we need to update all applications in the Data Access Layer. And in the second diagram now the orders have their own service.
* Now each part of the business is more manageable and easier to understand and maintain.
* Each module or service can be scaled independently. For example, Order Service and Authentication are used by several services so it could be a good idea to scale them independently.
* With the edge service for the 3er party reporting tool now we can change the provider if needed.
* The problem now is that we have increased the complexity so testing will be a little more difficult. Also deploying the whole application is more complex than in the previous one.
* Now in web applications we can change easier ASP .NET for any other Javascript framework like Angular and React if we want because Authentication and business logic are part of another service.
* In the second version architecture we can use BASE databases for many services like in the reporting service, delivery service and shop service. But in the previous model we got an ACID consistent data due all application are connected to a single sql database.