1. What are the main benefits of SOA?

Service Oriented Infrastructure (SOA) is the organization of the components of a system according to the services they provide. The provider of the service and the requester are loosely coupled within the architecture allowing the services to operate independently of the platform on which they will be used. It is an approach to designing software in which components are arranged according to business processes.

The main benefits of SAO are as follows

* Reliability: SOA services are complete and self-contained programs. This makes it easy for testing, debugging or any form of maintenance.
* Scalability: Different organizations have different levels of complexity in their architecture. So, the ability of services to be successfully run on different servers within an environment increases the scalability and availability of the service.
* **High Quality Service:** Servers, as we have seen, may have many applications to which they may be applied. As such, there will be no functional redundancy. Data is always consistent and the service codes are constantly maintained resulting in continuous high-quality applications.
* Platform Independence: This platform independence feature allows the integration of different requested services from different vendors to run harmoniously, regardless of the platform.

1. How can you achieve loose coupling in SOA?

Loose coupling is one of the principle of SOA. Loose coupling is avoiding or at least encapsulating temporal, technology and organizational constraints in the information system design.

Loosely coupled system support for late or dynamically binding to other components while running, and can mediate the difference in the component’s structure, security model, protocols, and semantics, thus abstracting volatility.

Loose coupling in SOA is how the services are implemented without impacting other services or application. The only interaction between the application and services is through the publish interfaces. This means application doesn’t interested how the services been implemented

1. Are web services and SOA the same?

* SOA is an architectural style for building software applications that use services available in a network such as the web. It promotes loose coupling between software components so that they can be reused. Applications in SOA are built based on services.

SOA allows for the reuse of existing assets where new services can be created from an existing IT infrastructure of systems

Services are software components with well-defined interfaces that are implementation-independent. An important aspect of SOA is the separation of the service interface (the what) from its implementation (the how).

* Web services are software systems designed to support interoperable machine-to-machine interaction over a network. This interoperability is gained through a set of XML-based open standards, such as WSDL, SOAP, and UDDI. These standards provide a common approach for defining, publishing, and using web services. They can be combined in a loosely coupled way in order to achieve complex operations. Programs providing simple services can interact with each other in order to deliver sophisticated added-value services.

Web services communicate over a network through HTTP between the two systems. Many web services are identical to SOA (Services Oriented Architecture) and mainly rely on standards such as XML-RPC and SOAP (Simple Object Access Protocol.

1. What is a reusable service?

The service reusability principle is a design principle, applied within the service-orientation design paradigm, to create services that can be reused across a business. These reusableservices are designed so that their solution logic is independent of any particular business process or technology.

Service reusability is typically measured by how much extra functionality a service contains that could be reused in future, and how much of the service’s functionality goes beyond the current requirements. This encourages services that contain extra capabilities built around possible future service usage scenarios. However, little is done in designing the service logic in a manner that it could be reused to automate multiple business processes.

1. What are the disadvantages of SOA?

**Extra overload**:

In SOA, all inputs are validated before it is sent to the service. If you are using multiple services then it will overload your system with extra computation.

High cost:

SOA is costly in terms of human resource, development, and technology.

High bandwidth server:

As some web services send and receives messages and information frequently it easily

reaches a million request per day. So it involves high speedserver with lot of data bandwidth

to run a web service.

1. What is ESB and where does it fit in?

An enterprise service bus (ESB) implements a communication system between mutually interacting software applications in a [service-oriented architecture](https://en.wikipedia.org/wiki/Service-oriented_architecture) .

It represents a [software architecture](https://en.wikipedia.org/wiki/Software_architecture) for distributed computing, and is a special variant of the more general [client-server](https://en.wikipedia.org/wiki/Client-server) model, wherein any application may behave as server or client.

ESB promotes agility and flexibility with regard to high-level protocol communication between applications. Its primary use is in [enterprise application integration](https://en.wikipedia.org/wiki/Enterprise_application_integration) (EAI) of heterogeneous and complex service landscapes.

1. In SOA do we need to build a system from scratch?

No, if we need to integrate any existing system you just can loosely couple wrappers which help in wrapping all customer services and expose all functionalities in a generic manner.

1. What is the most important skill needed to adopt SOA technical or cultural?

SOA implementing will likely need [enterprise architects](https://www.cio.com/article/101401/The_Rising_Importance_of_the_Enterprise_Architect), data architects, security specialists, process modelers, integration specialists, process analysts on the business side, and various types of developers.

If there is a need to purchase software such as an ESB, BPMS, service management tools, etc., there will be a need to hire people to administer the software.

Testers and infrastructure people will need to understand the concepts of SOA. It would be wise to consider bringing in an expert or two in this area as well.

SOA requires cultures that value enterprise architecture, governance, and standards will have a much easier time adapting to SOA then companies that do not have controls in place. Building services that are abstract, loosely coupled, extensible, and truly reusable requires a consistent and well governed design process.

1. List down the advantages of Microservices Architecture.

* Microservices Are Easier to Build and Enhance
* Microservices Are Easier to Deploy
* Microservices Are Easier to Maintain, Troubleshoot, and Extend
* Microservices Simplify Cross-Team Coordination
* Microservices Deliver Performance and Scale
* Microservices Simplify Real-Time Processing

1. .What are the best practices to design Microservices?

The best practices to design Microservices are

1. **Create separate data store for microservice**

The team needs to choose database that best suits the service.  Moreover, with a single data store it’s too easy for microservices written by different teams to share database structures, perhaps in the name of reducing duplication of work. You end up with the situation where if one team updates a database structure, other services that also use that structure have to be changed too.

1. **Keep Code at a Similar Level of Maturity**

Keep all code in a microservice at a similar level of maturity and stability. In other words, if you need to add or rewrite some of the code in a deployed microservice that’s working well, the best approach is usually to create a new microservice for the new or changed code, leaving the existing microservice in place.

This way you can iteratively deploy and test the new code until it is bug free and maximally efficient, without risking failure or performance degradation in the existing microservice.

### Do a Separate Build for Each Microservice

### Do a separate build for each microservice, so that it can pull in component files from the repository at the revision levels appropriate to it. This sometimes leads to the situation where various microservices pull in a similar set of files, but at different revision levels. That can make it more difficult to clean up your codebase by decommissioning old file versions.

### Deploy in Containers

### Deploying microservices in containers is important because it means you just need just one tool to deploy everything. As long as the microservice is in a container, the tool knows how to deploy it. It doesn’t matter what the container is. That said, Docker seems very quickly to have become the de facto standard for containers.

### Treat Servers as Stateless

### Treat servers, particularly those that run customer‑facing code, as interchangeable members of a group. They all perform the same functions, so you don’t need to be concerned about them individually. Your only concern is that there are enough of them to produce the amount of work you need, and you can use autoscaling to adjust the numbers up and down.

### How does Microservice architecture works ?

### A microservice architecture consists of “suites of independently deployable services” organized “around business capability, automated deployment, intelligence in the endpoints, and decentralized control of languages and data.

### The main idea behind a microservice architecture is that applications are simpler to build and maintain when broken down into smaller pieces that work seamlessly together. When using microservices, you isolate software functionality into multiple independent modules that are individually responsible for performing precisely defined, standalone tasks. These modules communicate with each other through simple, universally accessible application programming interfaces (APIs).

1. What are the pros and cons of Microservice Architecture?

### Pros

1. Service enabled, independently running components.
2. Independently running components classified around some business capabilities.
3. Product mentality over the project.
4. Smart components using simple communication channels like simple RESTish protocol or lightweight messaging queue.
5. Decentralize standards. Each independent component can use its exclusive standard for development and deployment.
6. Decentralized data management. Observe in the above diagram, how individual components have their own data storage.
7. Automated infrastructure management. For the deployment of independent components, we need to rely on automated infrastructure management to reduce complexity.
8. Application design considering failure in mind. There are several independent moving parts in applications. In the event of the receiver not getting a response, it should be handled gracefully.
9. Evolutionary design for getting the best possible decomposed system, which can be replaced and upgraded without affecting its collaborator.

### Cons

1. **Team communication overhead** — Microservices architecture reduces team management complexity, but it is not able to diminish the need for team communication. Teams need to make sure that updates in one team's service do not break another's team functionality. We find this problem in monolith architecture applications, too.
2. **Formal documentation overhead** — every individual running a component application needs to keep updated schemas and interface documents all the time. It helps other teams who are using the service.
3. **Dev-Ops complexity** — We need to have a mature DevOps team to handle the complexity involved in maintaining microservices-based applications. Due to several moving parts of the application, it becomes complex and requires a level of expertise.
4. **Increased resource use** — Initial investment to run these applications is high because all the independently running components need their runtime containers with more memory and CPU.
5. **Increase network communication** — Independently running components interact with each other using the network. Such systems require reliable and fast network connections.
6. **Network security**— Inter-service communication needs to be secured to avoid any security breach. Due to several moving parts, these applications are more prone to security vulnerabilities.
7. **Testing** — Testing such applications is harder in comparison to monolith applications.
8. **Production monitoring** — Cost to monitor such applications is higher. Unavailability of the right tools is also an issue to be considered.
9. **High upfront cost** — Running multiple applications will incur more cost in comparison to monolith applications.
10. What is the difference between Monolithic, SOA and Microservices Architecture?

**Microservices:**

Services are built in small units and expressed formally with business-oriented APIs.

Services exposed with a standard protocol, such as a RESTful API, and consumed/reused by other services and applications

Services exist as independent deployment artifacts and can be scaled independently of other services.

**SOA**:

Services can range in size anywhere from small application services to very large enterprise services including much more business functionality

Services exposed with a standard protocol, such as SOAP and consumed/reused by other services – leverage messaging middleware.

Dependencies between services and reusable sub-components can introduce scaling challenges.

**Monolithic:**

Monolithic applications evolve into huge size, a situation where understanding the entirety of the application is difficult.

Limited re-use is realized across monolithic applications.

Scaling monolithic applications can often be a challenge.

1. What are the challenges you face while working Microservice Architectures?

The challenges while working microservice architectures as follows

**Managing Microservices**:

As the number of microservices increases, managing them gets more challenging. It is important that management is planned before or while microservices are being built. While the modularity helps, things can very quickly get out of hand if not managed well.

**Monitoring:**

The traditional forms of monitoring and diagnostics will not align well with microservices since you have multiple services making up the same functionality previously supported by a single application. When a problem arises in the application, finding the root cause can be challenging if you do not have a means of monitoring and tracking the path a specific request took, like how many and which microservices were traversed for a specific request coming from a user interface.

**Embracing DevOps Culture:**

Separate teams need agility, autonomy, and continuous delivery to be able to deliver initial releases and subsequent iterative changes. A lack of DevOps culture can bottle up releases and impact the overall time to market and the response to business requests and issues.

**Fault Tolerance:**

Fault tolerance at the service level, and more importantly, at the overall solution level, is critical. Given the complexity of a microservices environment and the complex dependency chains, failure is inevitable. Microservices need to be able to withstand both internal and external failures. Robust resiliency testing is key to successful issue preparedness.

Testing:

Testing is much more complex in a microservices environment due to the different services, their integration, and interdependencies. The team members responsible for quality assurance need to be knowledgeable on the order and channels of communications between services to have full coverage in their test cases. The asynchronous aspect of microservices also makes it harder to test in lower environments.

1. What are the characteristics of Microservices?

The characteristics of Microservices are

* 1. Microservice architectural style is an approach to developing a single application as suite of small services.
  2. Services are built around business capabilities, independently deployable and packaged running in its own process.
  3. Each service have separate database layer.
  4. Each Service can be tested in isolation without dependent on other services.
  5. Each Service can be tested in isolation without dependent on other services.
  6. Each Service should have monitoring and troubleshooting capabilities for operation team.
  7. Services should implement Retry functionality in case of network failure or system failure.