	Layered Architecture	Pipeline Architecture	Service Based Architecture Style	Event-Driven Architecture Style	Space Based	Microservices Architecture
Deployability	This architecture has a low score on deployability due to its monolithic nature. The entire application or website must be deployed as one unit, making frequent deployments hard. Even the simplest change requires a complete build and deployment of the whole application.	This architecture has a low score on deployability due to its monolithic nature. The entire application or website must be deployed as one unit, making frequent deployments hard. Even the simplest change requires a complete build and deployment of the whole application.	Service-based architecture has a high deployability score. Each service is a separately deployable unit. In addition, it is possible to split the user interface and databases into multiple deployable units. A large number of small deployment units enables frequent deployments and reduces the risk of a single deployment, both of which lead to a high deployability score.	The event-driven architecture enables the decoupling of components allowing for easier deployment and updates to individual components without affecting the entire system.	SBA enables easy deployment of components, as they can be packaged into independent and self-contained units, allowing for updates and deployments without impacting the entire system.	This architecture has a high deployability rating as microservices enable easy and independent deployment of individual services, enabling faster updates without impacting the entire system. Microservices couldn't exist without the DevOps revolution and the relentless march toward automating operational concerns (automated deployment, testability, etc.)
Fault Tolerance	Given the monolithic nature, a bug (e.g., out-of-memory exception) in any part of the code risks bringing down the entire application, leading to low fault tolerance for this architecture.	Pipeline architectures don't support fault tolerance very well due to monolithic deployments and the lack of architectural modularity. If one part of a pipeline architecture causes an out-of-memory error, the entire application unit is impacted and crashes.	The decoupled nature of services in SOA allows for the isolation of failures, ensuring that a fault in one service does not necessarily lead to a complete system failure. The overall fault tolerance is lower than the microservices architecture because the services in SOA tend to be coarse-grained.	EDA's decoupled nature makes it possible to build fault-tolerant systems, as the failure of one component may not necessarily lead to a complete system failure. Additionally, the use of message brokers can help ensure message delivery even in case of temporary component failure.	Data partitioning, replication, and redundancy mean that a failure in one node can be mitigated by rerouting requests to other nodes, minimizing the impact on the overall system. On the other hand, the system relies on the eventual consistency of data leading to a medium score for fault tolerance.	The decoupled nature of microservices allows for the isolation of failures, ensuring that a fault in one service does not necessarily lead to a complete system failure, leading to a high score for fault tolerance.

	Layered Architecture	Pipeline Architecture	Service Based Architecture Style	Event-Driven Architecture Style	Space Based	Microservices Architecture
Scalability	Once this architecture has been deployed, little can be done to adjust the load and increase capacity dynamically. The scalability considerations are baked in once the application has been built and deployed. Dealing with increasing load typically requires updating the code and redeploying. Even then, given the monolithic nature of this architecture, the scalability cannot grow beyond a certain threshold.	Due to its monolithic nature, the overall design of this architecture bakes in the scalability considerations. As a result, little can be done post-deployment to react to increased load.	Service-based architecture scores a medium on scalability due to two factors. (1) The coarse-grained nature of the services, and (2) the shared database. Both of these can limit scalability under heavy loads.	Scalability is another strength of the event-driven architecture. High scalability is realized through horizontal scaling by programmatically adding event processors.	With the database no longer a bottleneck, this architecture scales extremely well by leveraging in-memory data caching and replication. The ability to dynamically bring up PUs as the load increases also contributes to a high scalability score for this architecture. Processing millions of concurrent users is possible using this architecture style.	Scalability is one of the core strengths of this architecture, making this one of the most scalable architectures. Microservices architecture supports both horizontal and vertical scalability. The granular services allow individual services to be scaled independently based on their resource requirements and workload.
Elasticity	Similar to scalability, this architecture style scores low for elasticity, as it cannot respond to increased user spikes beyond its baked-in capacity.	Similar to scalability, this architecture style scores low for elasticity, as it cannot respond to increased user spikes beyond its baked-in capacity.	Similar to scalability, the coarse-grained services and the shared database limit the elasticity of this architecture.	Similar to scalability, event-driven systems can adapt to changes in workload by dynamically scaling components up or down based on the number of events in the queue.	Same as scalability.	Similar to scalability, this architecture scores well for elasticity. The ability to horizontally scale individual services grants the system a high degree of elasticity.
Reliability	Layered architecture is the middle of the road for reliability. Given its monolithic nature, once deployed/installed successfully, it is likely to avoid problems with external dependencies (e.g., databases), which helps with reliability. On the other hand, it has low fault tolerance and a high time to recover (due to low deployability), which hurts the reliability score.	A pipeline architecture is the middle of the road for reliability. Given its medium scores on modularity testability, each stage of the pipeline can be verified independently. On the other hand, the monolithic nature, a bug (e.g., out-of-memory exception) in any part of the code risks bringing down the entire application, which hurts the reliability score for this architecture.	The isolation and independent deployment of services contribute to the overall reliability of the system, as it reduces the impact of failures and allows for easier recovery.	By using message brokers, EDA ensures the reliable delivery of messages and can even implement retries in case of failures, contributing to the system's overall reliability. However, due to its asynchronous nature, data loss can become an issue that negatively impacts the overall reliability of this architecture.	Data redundancy and replication mechanisms contribute to the system's overall reliability, ensuring data persistence and availability.	As the services are independent & single-purpose, a failure in one service does not automatically lead to the collapse of the entire system. Moreover, horizontally scaling each service enable redundancy, leading to even higher reliability. The high deployability score also helps, as it reduces the impact of failures and allows for easier recovery.

	Layered Architecture	Pipeline Architecture	Service Based Architecture Style	Event-Driven Architecture Style	Space Based	Microservices Architecture
Performance	Layered architecture avoids some of the costs associated with distributed architecture. However, the lack of parallel processing, closed layering, and the sinkhole architecture anti-pattern lock in its overall performance.	The use of pipes for communication between filters can introduce latency. Performance degradation can scale linearly with the number of filters in the path, leading to worse performance than tightly integrated monolithic designs. In some cases, creating parallel pipelines allows multiple stages to work on different data segments concurrently. This option leads to a medium overall score for performance.	The overhead of inter-service communication and potential latency introduced by network calls between services negatively impacts the performance of this architecture style. On the other hand, the distributed nature of the services presents opportunities for parallel processing, leading to a medium overall score.	Asynchronous communications combined with highly parallel processing leads to a high score for performance.	Performance is another core strength of this architecture. Parallel processing and in-memory caches for reading data lead to a high-performance score.	The performance of this architecture suffers due to the high degree of inter-service communication, which increases the latency of a request.
Modularity	The modularity is low modularity as each layer is built and deployed as part of the monolith. The layers are not standalone *modules* and are not reusable.	Even though the overall application is monolithic, the filters are typically built as separate modules and thus help with the modularity score of this architecture.	This architecture style promotes the separation of concerns, making it easier to develop, maintain, and scale individual services independently, leading to a high score on modularity.	EDA encourages modularity by decoupling components, allowing them to evolve independently.	The components within space-based architecture are built and deployed as independent, self-contained units. Often, off-the-shelf components are used as sub-components of the virtualized middleware, helping the modularity of the overall system.	The microservice architecture is highly modular by definition. The system is built up of small, independently developed, and maintained services, leading to a high modularity score.

	Layered Architecture	Pipeline Architecture	Service Based Architecture Style	Event-Driven Architecture Style	Space Based	Microservices Architecture
Extensibility	The extensibility score is low due to low modularity and deployability. New functionality has to be added by changing and redeploying the entire application.	We can add additional functionality via additional filter and transformer nodes, which makes it easier to add new paths of processing within this architecture. However, this cannot be done dynamically due to the monolithic nature of this architecture, resulting in a medium score.	The service-based architecture gets a medium on extensibility. On the plus side, new functionality is added by building new services without impacting existing services. On the negative side, the services are coarse-grained, which means often adding new functionality requires modifying and redeploying existing services which hurts the score.	Adding new features through existing or new event processors is relatively straightforward, particularly in the broker topology. The architecture makes it easy to add, modify or remove components without a major impact on the overall system, leading to a high rating for extensibility.	The architecture's data partitioning and parallel processing capabilities make it easier to accommodate new data-intensive tasks or additional processing requirements. Similarly, the reliance of this architecture on message-passing or event-driven communication between components makes it easier to build and integrate new functionality.	As the services are granular and independent, adding new functionality becomes easy. Often, new workflows can be added by simply combining existing services. Updating an existing service limits the scope of the change, making it easy to test and deploy.
Testability	Given the monolithic nature of this architecture, even a simple change often requires running the entire test suite before deployment, which is time consuming and lowers the testability of this architecture.	The fact that the filters can be built as independent modules helps with the testability of this architecture. However, the score is pegged as a medium due to the monolith nature and low deployability.	This architecture scores high on testability as high modularity limits the scope of the domain for a given service, and high deployability means that frequent deployments are less risky.	Testing individual components can be straightforward, but testing interactions between components (e.g., event flow) can be more challenging due to the asynchronous and nondeterministic event flows.	Testing individual components can be straightforward, but testing interactions between the nodes and ensuring data consistency across nodes is challenging.	Microservices can be tested independently, allowing for more focused and efficient testing of individual components.

	Layered Architecture	Pipeline Architecture	Service Based Architecture Style	Event-Driven Architecture Style	Space Based	Microservices Architecture
Simplicity	Simplicity is one of the strengths of this architecture. This architecture is easy to understand, and the concept of layering code is familiar to all software engineers. Often, each layer grows organically due to code organization best practices. Finally, due to its monolithic nature, this architecture does not have to deal with the complexities associated with distributed architectures.	The pipeline architecture simplifies the design of individual stages, making it conceptually easy to understand.	As a distributed architecture, service-based architecture is more complex than monolithic architecture. However, this is one of the most straightforward distributed architectures, leading to a medium overall rating.	EDA can simplify the overall design of a system by decoupling components. However, the asynchronous and distributed nature of the architecture can introduce complexity in understanding event flows and handling failures.	Space-based architecture is a highly complex architecture style due to caching and eventual consistency of the primary data store, which is the ultimate system of record.	While microservices can simplify the design of individual services, the overall architecture is highly complex due to inter-service communication, data consistency, and coordination.
Overall Cost	Cost is another advantage of this architectural style. Its simplicity, low barrier to entry, and familiarity make it relatively low-cost to build and maintain.	Pipeline architecture can reduce development and maintenance costs through modularity and extensibility. Due to its monolithic nature, it is relatively easy to build and maintain, which helps with its costs.	All distributed architectures have higher costs than monolithic architectures due to higher complexity and more resource usage. Specifically, distributed architectures increase infrastructure and operational costs due to the need for managing and maintaining multiple services, load balancing, and monitoring.	High modularity and extensibility help with the overall maintenance costs of this architecture. However, the need for message brokers and managing distributed components increases operational and infrastructure costs.	Compared to other architecture styles, space-based architecture is expensive, primarily due to infrastructure costs associated with licensing fees and managing and maintaining multiple nodes.	Microservices can reduce development and maintenance costs by enabling modularity and extensibility. However, they may increase infrastructure and operational costs due to the need for managing and maintaining multiple services, load balancing, and monitoring.