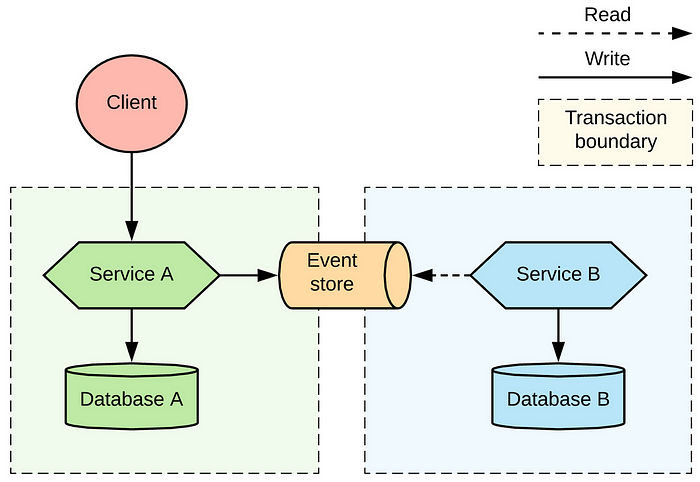
**What is Event Sourcing Design Pattern in Microservice Architecture? How does it work?**

**An Overview of Event Sourcing Design Pattern and Its Implementation in Microservice Architecture.**



Hello folks, In last a couple of years , due to increased cloud adoption, there has been a growing trend towards the adoption of [**microservices architecture**](https://medium.com/javarevisited/difference-between-microservices-and-monolithic-architecture-for-java-interviews-af525908c2d5) for building large-scale software systems, not just in Java world but also in .NET and other programming languages.

With this shift, new design patterns and techniques have emerged to help developers manage the complexity and scalability of these systems. One such pattern is **Event sourcing**, which is gaining popularity as a way to build systems that can **easily track changes and recover from failures.**

In last few articles, I have explained you about various Microservice design patterns like [**SAGA**](https://medium.com/javarevisited/what-is-saga-pattern-in-microservice-architecture-which-problem-does-it-solve-de45d7d01d2b), [**API Gateway**](https://medium.com/javarevisited/difference-between-api-gateway-and-load-balancer-in-microservices-8c8b552a024), [**Circuit-Breaker**](https://medium.com/javarevisited/what-is-circuit-breaker-design-pattern-in-microservices-java-spring-cloud-netflix-hystrix-example-f285929d7f68), and [**CQRS**](https://medium.com/javarevisited/difference-between-saga-and-cqrs-design-patterns-in-microservices-acd1729a6b02) and in this article, I will tell you about Event sourcing pattern, another important pattern for [*Microservice interview questions*](https://medium.com/javarevisited/50-microservices-interview-questions-for-java-programmers-70a4a68c4349) perspective.

**Event sourcing** is a Microservice design pattern that involves capturing all changes to an application’s state as a **sequence of events, rather than simply updating the state itself**. Each event represents a discrete change to the system and is stored in **an event log,** which can be used to reconstruct the system’s state at any point in time.

This approach allows for easy auditing and provides a reliable source of truth for the system’s data.

As I said, in this article, you will learn about event sourcing pattern in more detail and discuss how it can be used in [Microservices architecture](https://medium.com/javarevisited/difference-between-microservices-and-monolithic-architecture-for-java-interviews-af525908c2d5) to build robust and resilient systems.

You will also learn about the key components of an event sourcing system, how events are stored and retrieved, and how this approach can be used to simplify the development of distributed systems.

And if you haven’t joined Medium yet then I also suggest you to [**join Medium**](https://medium.com/@somasharma_81597/membership) to read your favorite stories without interruption and learn from great authors and developers. You can **join Medium** [**here**](https://medium.com/@somasharma_81597/membership)

**What is Event Sourcing Pattern in Microservices Architecture? How does it work?**

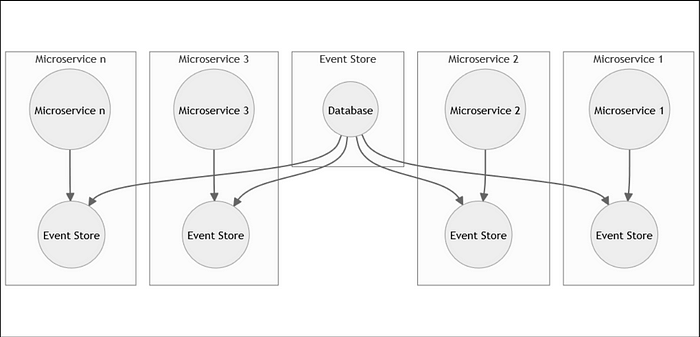
As I said, Event sourcing is a design pattern that stores the state of an application as a sequence of events in the order that they occurred, rather than simply storing the current state of the application. In microservice architecture, this pattern can help in **achieving better scalability, fault tolerance, and resilience.**

Here’s how it works:

1. An event is generated whenever a change occurs in the system.
2. The event is persisted to an event store, which is essentially a log of all events that have occurred in the system.
3. The current state of the system can be reconstructed at any time by replaying all of the events in the event store, in the order that they occurred.
4. Each service in the microservice architecture can have its own event store, which can be used to maintain its own state.
5. Services can subscribe to events that are relevant to them and update their own state accordingly.

**By using event sourcing, it is possible to achieve better fault tolerance and resilience.** Because the system can be reconstructed from the event store, it is possible to recover from failures more easily. Additionally, because the event store is a**ppend-only**, it is more difficult for data to be lost or corrupted.

Here is a diagram which *shows several microservices, each with its own event store*, connected to a shared database where the events are stored.



One of the main advantages of Event sourcing is that it also makes it easier to support auditing and compliance requirements, as a complete record of all changes to the system is maintained.

However, *implementing event sourcing can be more complex than other approaches,* as it requires additional infrastructure to store and manage events. Additionally, because the state of the system is reconstructed from events, it can be more difficult to query and analyze the current state of the system.

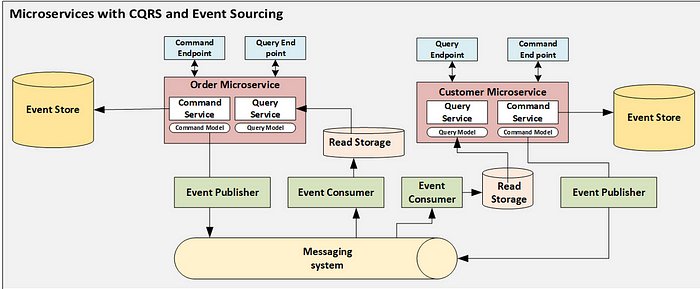
But, don’t worry, let’s first learn about this pattern in depth and its different component then we will take a detailed look into pros and cons of Event sourcing pattern in Microservices architecture.

**6 Key Components of Event sourcing system in Microservices**

Here are the key components of a Event sourcing System which are important to implement Event sourcing pattern in any Microservices architecture:

1. **Event Store**  
   The event store is a **database** that stores all the events that have occurred in the system. It is an immutable log of all the changes that have occurred to the state of the system over time.
2. **Aggregate**  
   An aggregate is a **domain object** that is responsible for processing the commands received by the system and generating events. It represents the current state of the domain object and provides a way to apply the events to rebuild the current state.
3. **Command**  
   A command is a request to the system to perform some action. When a command is received, it is processed by the aggregate, which generates one or more events as a result.
4. **Event**An event is a representation of a change that has occurred to the state of the system. It contains all the necessary information to reconstruct the current state of the system.
5. **Event Bus**  
   The event bus is responsible for distributing events to all the interested parties in the system. It ensures that all the interested parties receive the events in the same order.
6. **Read Model**  
   The read model is a de-normalized view of the data that is optimized for querying. It is built by consuming the events from the event store and updating the view accordingly. It is used to answer queries in a fast and efficient way.

Here is a nice diagram which shows how a Event Sourcing pattern is implemented in Microservices, you can see that we have event publishers and event consumer as well as Event stores where events are stored.



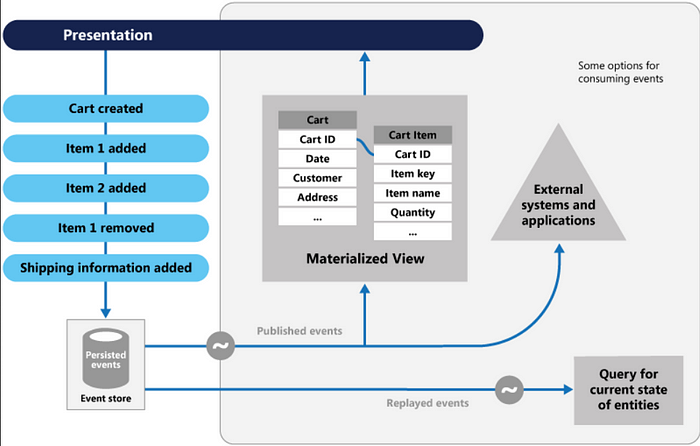
**How Events are stored and retrieved in Event Sourcing Design Pattern?**

In the Event Sourcing pattern, events are stored in an event log or an append-only store in the order in which they occur. Each event represents a change in the state of the system. The event log acts as the single source of truth for the system’s state.

When retrieving the state of the system at a particular point in time, all the events that occurred before that point in time are read from the event log and used to compute the current state of the system. This process is known as event replay.

In other words, the state of the system is derived from the sequence of events that occurred. Since the events are stored in an immutable store, they cannot be modified or deleted, which ensures that the system’s state can be reconstructed at any point in time.

Here is another diagram which shows how events are published, stored and retrieved in Event sourcing pattern in Microservice architecture:



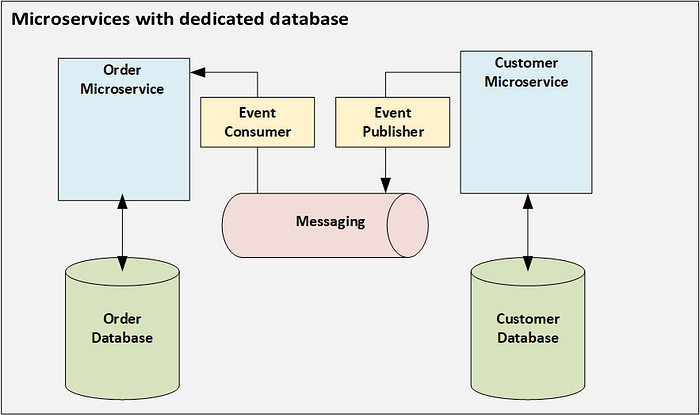
**How Event sourcing approach simplifies development of distributed systems**

Event sourcing can simplify the development of distributed systems in several ways. First, it enables developers to break down complex systems into smaller, **more manageable services**. Each service can then be designed to handle a specific type of event, making it easier to test and maintain.

Second, event sourcing allows for **easier integration between services since each service is simply publishing and consuming events**. This means that changes to one service can be easily propagated to other services, without the need for complex integration logic.

Third, event sourcing can help **ensure consistency across services**. By maintaining a single source of truth for events, developers can avoid issues where different services have conflicting data. This can be especially important in systems where multiple services are making changes to the same data.

And, event sourcing can also **provide a powerful audit trail for distributed systems.** By recording every event, developers can easily trace the flow of data through the system, and identify the source of any issues that arise. This can be critical for debugging complex systems, especially in production environments.



**How can you use Event sourcing pattern to manage distributed transactions in Microservices?**

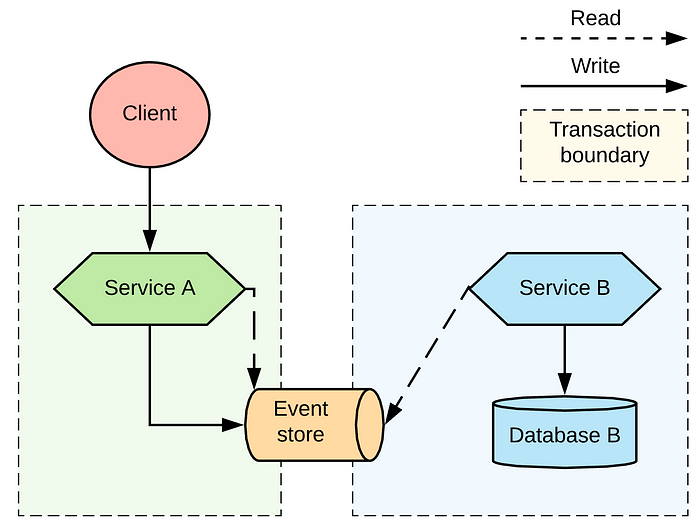
As I explained in my earlier article about [**3 ways to manage distributed transaction**](https://medium.com/javarevisited/how-to-manage-transactions-in-distributed-systems-and-microservices-d66ff26b405e) in Microservices, Event sourcing can be used to manage distributed transactions by keeping track of all the events that occur during a transaction.

Each microservice involved in the transaction can generate events that reflect the changes made to its local state, and those events can be stored in a centralized event store.

When the transaction is complete, a **coordinator microservice can examine the events generated by each microservice** and determine if the transaction was successful or if it needs to be rolled back. The coordinator can use compensating events to undo any changes made by the transaction, if necessary.

By using event sourcing in this way, **distributed transactions can be managed without the need for a two-phase commit protocol,** which can be expensive and lead to performance issues.

Instead, the **event sourcing pattern allows for a more flexible and scalable approach to** [**managing distributed transactions in a microservice architecture**](https://medium.com/javarevisited/how-to-manage-transactions-in-distributed-systems-and-microservices-d66ff26b405e)**.**



**What are Pros and Cons of Event Sourcing Pattern in Microservices?**

Like any other [Microservices pattern](https://medium.com/javarevisited/top-10-microservice-design-patterns-for-experienced-developers-f4f5f782810e), Event sourcing pattern in microservices architecture also has its own advantages and disadvantages. Here are main pros and cons of using Event sourcing pattern in Microservices

**Pros:**

* Event sourcing provides an audit log of all the events, which can be used for **debugging and reconciliation**.
* It also makes it easier to **track changes** and maintain a history of events.
* Event Sourcing enables scalability and resilience by allowing the system to be **partitioned** and **distributed**.
* Helps to simplify the development of distributed systems by providing a clear **separation of concerns.**
* Event Sourcing pattern also enables the system to be event-driven, which can improve the responsiveness and agility of the system.
* Helps to ensure data consistency and integrity by using a write-once, read-many approach.
* Enables developers to easily add new functionality by reacting to events instead of building complex queries.

**Cons:**

* Event Sourcing pattern can be more complex to implement than traditional database-driven systems.
* It can also be more resource-intensive, as it requires storing and processing a large number of events.
* It also requires careful consideration of the event schema and versioning.
* More importantly, it can be difficult to debug, as events are stored and processed asynchronously.
* Event sourcing pattern may require more complex recovery strategies in the event of system failures.
* It can be more difficult to ensure data privacy and security, as events are stored in an unencrypted form.

It is important to consider these pros and cons while deciding whether to **use event sourcing pattern** in a [microservices architecture](https://medium.com/javarevisited/what-is-api-gateway-pattern-in-microservices-architecture-what-problem-does-it-solve-ebf75ae84698). The benefits of event sourcing can be significant, but they must be weighed against the challenges and complexities involved in implementing and maintaining such a system.

**Conclusion**

That’s all about w**hat is Event Sourcing pattern and how does it work?** We have also seen how it simply development of distributed system and its various pros and cons.

In conclusion, the *Event Sourcing pattern is a powerful tool for building highly scalable, fault-tolerant, and reliable microservices systems*. By using this pattern, developers can easily manage the state of their services, simplify distributed transactions, and reduce complexity in their systems.

However, **it is important to carefully consider the trade-offs before deciding to use Event Sourcing.** While the benefits are significant, the additional complexity and overhead required to implement and maintain the pattern must be taken into account.

And , if you like this article and you are not a Medium member then I highly recommend you to join Medium and read great stories from great authors from real field. You can **join Medium** [**here**](https://medium.com/@somasharma_81597/membership)