ProcessMaker

Revolutionize with Microservices Process Orchestration

WHITEPAPER

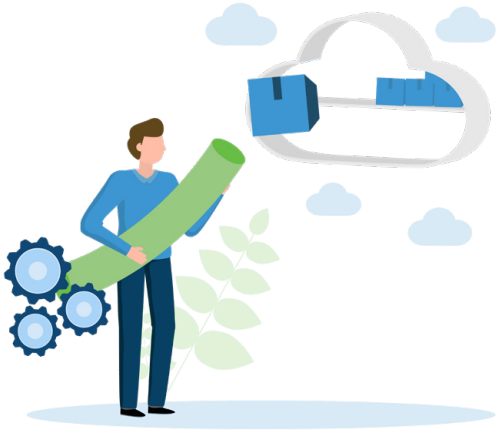
+1.919.289.1377

PROCESSMAKER.COM

Table of Content

Introduction

Microservices came about as a natural progression from monolithic architecture. Traditional monolithic systems have become too time-consuming to update and maintain. During the Waterfall days of software development, a monolithic architecture made sense. In today's Agile environments, monolithic applications are just too clunky and complicated. **With microservices expected to grow globally at 22.5% between 2019 and 2025**, companies who do not transition to microservices will struggle to retain a competitive edge. Now is the time to revolutionize your business processes with microservices process orchestration.



+ 22.5%

expected microservices growth

Monolithic architecture

It only takes a single code base with a varying number of modules to create a monolithic application. The number of modules depends on the number of technical features desired. Further, the entire application is built on a singular system with deployment based on a single executable binary. There are fewer cross-cutting concerns since the basis of a monolithic architecture means linking multiple components.

Monolith testing is not very complex, but entanglement makes it difficult for independent scaling or isolating services in need of upgrades. Monolithic applications are also more challenging to understand when new layers are added to scale the application. In terms of upgrades, a monolithic application may need a completely new re-code. And it's difficult to break up tightly-integrated applications. If you need to use more than one programming language, monolithic applications are limited. Moreover, it is a challenge to add self-contained third-party components and tools.

Microservices architecture

Within the microservices architecture, you get a suite of modular services and components to facilitate extensive application development.

Microservices offer independently-deployable units that are organized based on business capabilities. Each unit is decoupled, meaning it can upgrade it separately without requiring a re-code of the entire application. Besides, microservices are as flexible as you need to be with independent scaling. So then, new developers only need to master the microservice unit they are responsible for instead of the full application architecture. When adding new components, you can scale each microservice separately. As a result, a fault in one module does not affect the entire application. Replacing faulty modules is easy.

For the most part, there are two types of microservices:

Stateless

The stateless form of microservice does not maintain any session state between requests. If a service is removed, it will not affect the processing logic. When using a distributed system, stateless microservices are recommended.

Stateful

When two or more microservices interact and communicate, they maintain a service request state as they store session data within the code. In environments where you want to store session information, stateful microservices are recommended.

Let's take a look at booking an airline ticket. Using the monolithic approach would include a single application process to "Book the ticket." However, "book ticket" also incorporates smaller operations such as making ticket reservations, charging a credit card, and deploying a customer's confirmation.

When using a microservices approach, each process is broken down into separate services. One service is booking the ticket, and another service is taking and charging the credit card. Moreover, each service communicates through a specified interface.

The rise of microservices

As you can see, microservices helps applications get to market much faster since it can scale better than a monolithic architecture and retain an agile environment. Decomposing a system into microservices offers more autonomy for teams to deploy at any time since each component is isolated. Invariably, microservice architectures are now mainstream.

Microservices scalability

If you wanted to scale a monolith architecture, you could run multiple route requests and instances. You could also use non-blocking I/O and execute multiple threads. However, you can do all of these and more in microservice architectures -- and you wouldn't need nearly as many resources. In fact, each resource can manage a larger number of connections.

You get more throughput with microservices and more precise scaling. When a monolith uses up all its resources, then you might consider adding a second instance. On the other hand, when a microservice uses all its resources, you only need to add more instances to the one service.

Scaling individual services is quite precise and fine-grained. Independent microservices can and should work independently to relate to the following components:

Independent life cycle

You can start, stop, and make changes to individual microservices independently from other microservices. Each team can own a microservice without having to coordinate with other departments.

Stable interfaces

You can make interface changes by re-versioning the interface. Otherwise, microservices offer a stable interface environment.

Local data storage

Local copies of information are always stored by microservices that enable the fulfillment of designated services.

Communication

Asynchronous messaging can facilitate communication between microservices. Moreover, a bulkhead architecture ensures that a service outage does not affect any other service processes.

Independent scalability

Every microservice may need different resources and can scale independently from other services.

Fault tolerance

The entire architecture will continue to run despite any faulty services.

What about communication and interaction? We cover that below:

Asynchronous communication

Microservices can exchange asynchronous messages either via a message broker or an event bus. You don't need dedicated topics for a single communication channel. Instead, you can use one large pipeline for all events. Microservices will then communicate and consume events resulting in decoupling. This does not require a centralized "brain" to control every component.

Synchronous remote calls

Another communication approach is via synchronous remote calls using REST as a fundamental call chain. You also get a retry mechanism and state handling. For example, if a customer is trying to make a purchase and their credit card is declined, the entire order does not have to be canceled. On the contrary, a simple message is sent to the customer to retry or use another card within a specified time frame. If the customer updates their credit card data within the period given, they can still complete their purchase successfully. In this example, every order has a persistent state.

To achieve persistence requires simple state handling frameworks, actor frameworks, or custom entities. If the customer does not respond in the time allotted, it helps to monitor order processes, timeouts, and adequate reporting.

Benefits of using a workflow engine

Unquestionably, a workflow engine comes with many advantages, such as having explicit processes that can be modified easily and swiftly. You also get the benefit of the workflow engine handling the persistence of every order instance. The status of process instances is transparent with monitoring available. And visibility helps to ensure the process remains optimized.

Microservices process orchestration

Consider how end-to-end business processes can stretch across independent microservices that can communicate to achieve the intended results. Even within a single architecture, microservices can execute various communication styles. Microservices may communicate via REST or place messages on an event bus.

For instance, a customer interacts with a hypothetical mobile stock trading app. The customer downloads the app and receives a credit to purchase their first stock. The stock app is also on alert for any other opportunities to offer promotions that induce customer loyalty.

There are several steps involved for the customer, such as purchasing their first stock, receiving other promotions, and using those promotions on future purchases.

From the application perspective, every transaction is managed by a separate microservice. But, what if an error occurs during the purchase? What if an Internet interruption occurs? What if the system failed to receive the callback? What if the customer was charged twice for the same product? The possibilities are endless. To avoid an unhappy customer, perhaps you cancel one of the transactions.

Nonetheless, this is where process orchestration plays a critical role. The credit can't be used until it reaches the customer's account, and the stock can't be purchased until the customer has the credit. Using process orchestration is what coordinates each microservice.

BPMN orchestration engines offer a microservice process orchestration framework using the BPMN standard. As a result, it is much easier to comprehend the logic and define the flow. Moreover, many workflow and BPM platforms already support the BPMN standard using graphical notation to define the microservices process orchestration. Since every process is clearly shown, it becomes easier to set timers, define exception handling, and more.

You can also define how to handle errors and manage mistakes. To illustrate, "Do B -> Do C; if C fails, undo B." With BPMN, you don't have to hand-code or hard-code any logic. Sequential processes are also transparent. You can revolutionize microservices process orchestration to coordinate applications, robots, and humans.

Further, BPMN facilitates the automation of any service managed through APIs while it integrates with legacy and specialty systems for a seamless transition.

Use a lightweight and embeddable engine

Workflow engines do have a reputation for being part of large, zero-code BPM suites. However, they can also feel like a development library and feel very lightweight, running embedded within the microservice. For instance, every microservice can have its engine, and the team who owns that service can decide what version they prefer. The engine initiates swiftly and does not require external dependencies to run automated unit tests.

There is another crucial component: With Processmaker, you can express processes in code.

In conclusion

End-to-end business processes can be executed using the microservices process orchestration. Therefore, business processes can be distributed to various microservices depending on capabilities. Nonetheless, it is vital to use proper event command transformations and introduce them to a microservice and bounded context.

With Processmaker, you get a lightweight workflow and BPM platform agile enough to work with microservices process orchestration. You do not have to run from a centralized component. Instead, you can combine flows programmatically and seamlessly. When applying microservices, it's essential to align the business side with technology. And transparency is a must. If you're ready to take on the challenge, we can assure you it will be worth your time.

About ProcessMaker

ProcessMaker is low-code BPM and workflow software. ProcessMaker makes it easy for business analysts to collaborate with IT to automate complex business processes connecting people and existing company systems. Headquartered in Durham, North Carolina in the United States, ProcessMaker has a partner network spread across 35 countries on five continents. Hundreds of commercial customers, including many Fortune 100 companies, rely on ProcessMaker to digitally transform their core business processes enabling faster decision making, improved compliance, and better performance.

Ready to learn more about how intelligent process automation can serve your organization?
Learn more at **processmaker.com**

Request a Free Demo
+1.919.289.1377

Momentum Leader WINTER 2021
Easiest To Use Enterprise WINTER 2021
High Performer Enterprise WINTER 2021
//CODiE// 2020 SIIA CODiE WINNER
Leader Enterprise WINTER 2021
Users Most Likely To Recommend Enterprise WINTER 2021