WHITEPAPER

Microservices: The Next Logical Step to Grow Your SaaS Business

aspire SYSTEMS

attention. always.

www.aspiresys.com

Monolithic applications have been around for decades and the IT industry has been content with them until the early part of the last decade. Prominent software developers around the world had been urging their counterparts to switch to service oriented architectures for years. Their discontent with monoliths is understandable because they are riddled with complexities. Monolithic applications are large and they are slow to start. Their sheer size makes it difficult for testing teams to understand the impact of any changes made in them. If there is an error in the code, the entire application should be brought down and they are difficult to scale up. These issues have been plaguing ISVs until recently.

Microservices and related architectures have gained a lot of attention among IT-experts. Although, the underlying concept of a distributed architecture based on services is not really new, it potentially means a revolution within the IT-department when combined with a DevOps approach. This approach supports the demand for increased business agility, resulting in the best possible time-to-market. But since both the business and IT departments will have to collaborate with each other, it demands a different mindset.

In this whitepaper we discuss the origins of Microservices, the concept, the advantages and challenges you might face while implementing. Besides, we discuss why Microservices migration is essential for growing ISVs and how they can embrace it.

Microservices providing immediate business value

There is a growing urge in the industry for agility, cost optimization, and shifts between Capital Expenses (CapEx) and Operating Expenses (OPEX). Microservices Architecture is proving to be the facilitator for all of these. Also, this is convenient for the testing strategies and DevOps paradigm.

A successful ISV running its business based on its self-developed solution will most likely be facing the challenges of a monolithic architecture. A potential next step could be to move to Microservices.

Microservices Defined

Microservices are service-oriented architecture patterns where applications are built as a collection of various small independent service units. It is a software engineering approach that develops an application with single-function modules and well-defined interfaces. These modules are independently deployed and operated by small teams who own the entire lifecycle of the service.

Vision and Goals

Before talking about Microservices migration, it is important to explain how the benefits of Microservices migration relate to an ISV's business vision.

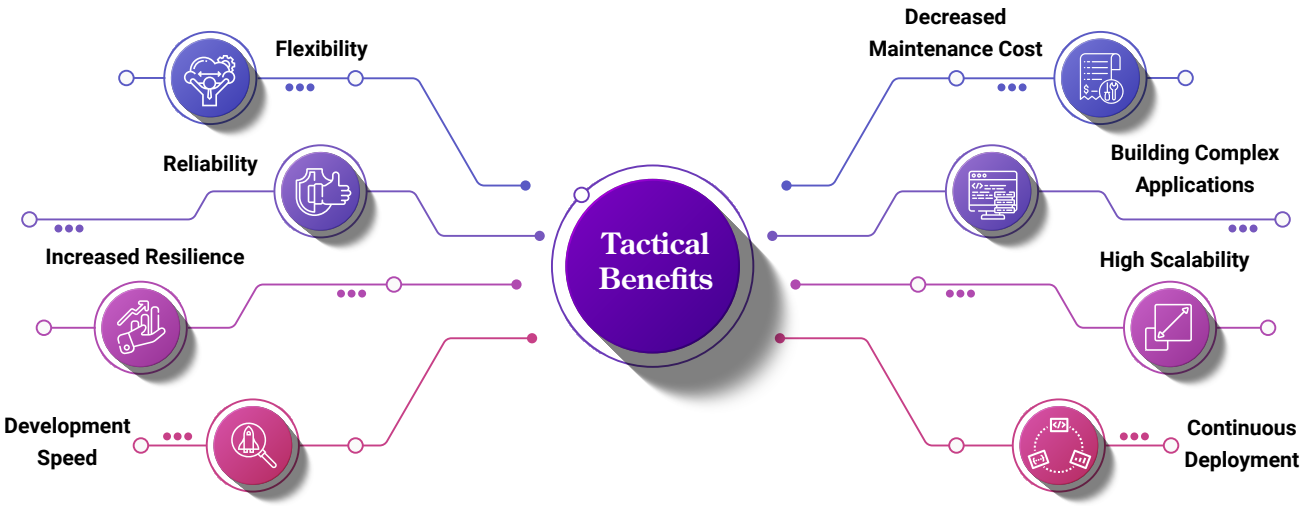
Following table provides a sample structure of vision, goals, and action:

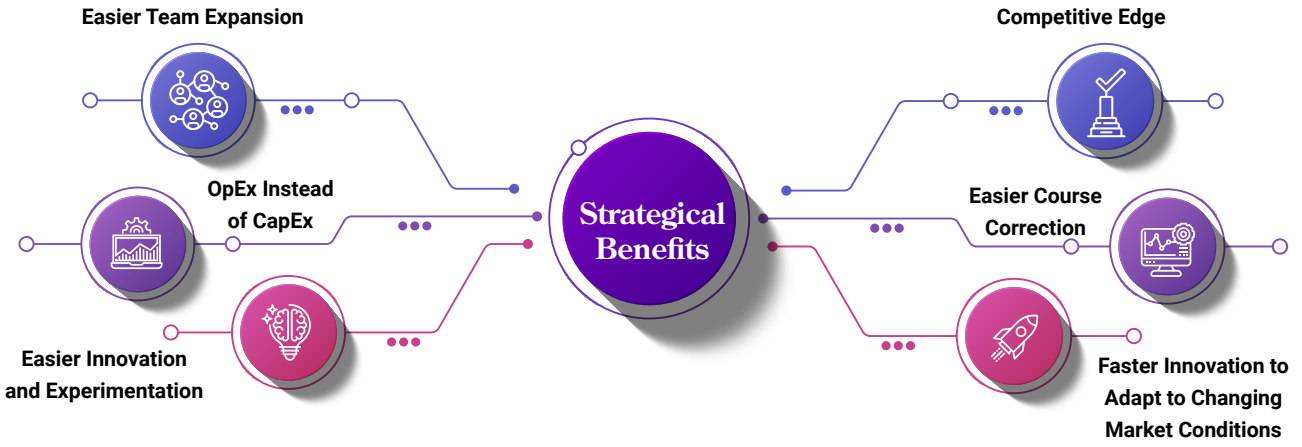


Vision	Goals	Action
Be a market leader in the space in the 4 years	Improve customer satisfaction index by **10%**	Switch to an architecture that supports continuous deployment easily
	Decrease operational cost by **30%**	Move to an architecture that reduces maintenance cost significantly
	Grow by **25%** every year	Shift to an architecture that can scale exponentially and offers economy of scale

Table 1: Sample vision, goals and action

Microservices Tactical and Strategical Benefits





Tactical Benefits

Flexibility:

Microservices architecture is quite flexible. Different Microservices can be developed by using different technologies. Since Microservices are small, the code base is quite less, so it's not that difficult to upgrade the technology stack versions. Also, we can incrementally adopt a newer technology without much difficulty.

Reliability:

Microservices architecture is very reliable. If one feature goes down, the entire application doesn't go down. We can easily fix the issue in the corresponding microservice and immediately deploy it.

Increased Resilience:

Microservices are decentralized and decoupled and the services act as separate entities. With Microservices, the entire application decentralizes and decouples into services that act as separate entities. Unlike the monolithic architecture wherein a failure in the code affects more than one service or function, there is very little impact of a failure in microservices. Even if several systems are brought down for maintenance, your users won't notice it.

Development Speed:

Development is fast in this architecture. Since the volume of code is much less for Microservices, it's not difficult for new team members to understand and modify the code. They become productive right from the start and the code quality is high. The IDE is much faster. Microservices take less time to start up. All of these factors considerably increase developers'

productivity.

Decreased Maintenance Cost:

Microservices architecture enforces stability and hence its maintenance cost can be cut down to half.

Building Complex Applications:

With Microservices architecture, it is easy to build complex applications. If the business features are analyzed properly, we can break it down into independent components that can deploy independently. Deciding the boundaries of Microservices can be quite challenging. It is an evolutionary process, but once we decide, it is easy to develop, as there is no limitation in technologies.

High Scalability:

Scalability is a major advantage. Each Microservice scales individually.

Continuous Deployment:

Continuous deployment becomes easier. In order to update one component, we must redeploy only that Microservice.

Strategical Benefits

Easier Team Expansion:

It is easier to find developers and application teams to work on new applications rather than to work on an existing application. Microservices are almost like a new application and hence it is easier to add new teams much faster to work on them. Also, Microservices are small enough for anyone new to understand, making it easier to induce teams to work on it.

OpEx Instead of CapEx:

Once you have the base, it is very easy to add more and more Microservices at scale and hence instead of looking at the cost per application, the costing changes to per Microservice. This allows incurring on-demand development costs with ease rather than a huge development cost upfront.

Easier Innovation and Experimentation:

Since it is very easy to add or discard Microservices, it is possible to quickly and cost-effectively build a new feature.

Competitive Edge:

As innovation and experimentation are easier, it provides the ISVs with a high competitive edge.

Easier Course Correction:

Many applications contain tons and tons of features that nobody uses. These exist in the code because of a fear that it would break something else. With Microservices, it is very easy to continuously assess business value and retire features that do not add significant value. This leads to a significant saving in maintenance costs.

Faster Innovation to Adapt to Changing Market Conditions:

Microservices can also help you to adapt more quickly to changing market conditions. Since Microservices allow applications to be updated and tested quickly, you can follow market trends and adapt your products faster.

With Microservices architecture, organizations can try out a new technology stack on an individual service with fewer concerns about dependencies, and you can roll back changes much easier if necessary. And if a module fails, it doesn't impact the rest of the application.

Now, we have clearly established the advantages of Microservices, the above table of mission, goal and action transforms to:

Vision	Goals	Action
Be a market leader in the space in the 4 years	Improve customer satisfaction index by **10%**	Switch to Microservices architecture that supports continuous deployment easily
	Decrease operational cost by **30%**	Move to Microservices architecture that reduces maintenance cost significantly
	Grow by **25%** every year	Shift to Microservices architecture that can scale exponentially and offers economy of scale

Table 2: Vision, Goals, and action after migrating to Microservices

Achieving Microservices Migration

Once an ISV decides that the Microservices Architecture is the right way forward, the next step is to act. It is very important to look at the end to end factors in this movement.

Many think that Microservices migration is only about dealing with technical changes. Though technical alignment is very important, there are other factors that will account for when an ISV moves towards Microservices architecture.

Team Structure Alignment:

The structure of your software should reflect the structure of your teams. Teams should align with the services and must organize in a way that allows them to own what they're responsible for, end to end. Amazon calls this as *"you build it, you own it"*, or *"build and run"* teams, responsible for development and production throughout the entire lifecycle for a chunk of software. The team size for one build and run team should not be more than 10. Since ISVs are going to end up with multiple smaller teams at any points, they should also build an overall governance team which spans across all the teams for providing governance and also align individual goals of the team to an overall goal.

Process Alignment

While the overall agile methodology holds good for Microservices development as well, there are many things that we need to bring into the process that might not be relevant for monolithic applications. Following points cover the most important ones:

At the beginning of the Microservices development, dedicate **at least three to four sprints** for defining the software architecture, infrastructure architecture, DevOps architecture, standard guidelines and practices for the Microservices. This is very important to avoid massive reworks and failures later. A general naming, we follow for these sprints are Foundation sprints.

Once the development has started, multiple parallel scrum teams need to run for the Microservices that are in development respectively. **A scrum of scrums** needs to run aligning the different delivery goals of the independent Microservices teams to an overall goal of the product release.

The **Testing strategy** needs to be different for the Microservices approach. You should not rely on manual testing. Since Microservices are smaller in scope, the time required for this automation testing is generally small and it is simple. **Automated testing** is the one that helps majorly in the faster release to market. Microservices without automated testing is a road to failure. The types of automated tests that should be developed for all Microservices are:

Unit Tests

API Tests

Event driven Integration tests

Saga automation tests

Contract tests

End to end tests

Again, like any other application, these tests should follow a pyramid with the highest number of unit test cases to lowest end-to-end tests.

A standardized DevOps process defined for all the Microservices. IAC and pipeline as a code need to be adhered. It is highly counter-productive to define this process for each Microservice. IAC and Pipeline code library needs to be available from which each of the Microservices can choose from. If it is very much necessary, Microservices can alter it for their needs. This indicates specifically as a task during scrum planning.

Dependency management is crucial for microservices architecture. Developers need the habit of tracking and ensuring that their versioning strategy does not affect the dependents. It is highly recommended to use a specialized Microservice dependency tracking system to do this.

The release process needs to be an end-to-end automated process. The stages for the release pipeline and the approval authorities need to be set in place from the beginning.

Periodic audits should be conducted to ensure that Microservices are adhering to the right guidelines.

Technology Alignment

Designing, delivering and operating Microservices application needs to follow a different mindset as opposed to monolithic. We will discuss the topmost technical principles below:

API, Events, bounded context as the first–class citizen:

People who are working on monolithic applications tend to jump directly to database structures and classes when designing the solution. While working with Microservices, the team should think about APIs, Events and the bounded context it represents. The underlying implementation should follow only after these details ironed out.

Authentication and Access Control:

Authentication and access control in Microservices has different needs as opposed to a monolithic application. As there are many Microservices that are widespread, it is advisable to have a single gateway for all the external endpoints for the Microservices. Granular access control enforced by an individual Microservice and an overall authentication check enforced at the gateway. One other authentication need is for the inter-service Microservices call. In this case, either the Microservices can adopt a policy of full trust between them which does not need authentication or adopt a policy to carry forward the authentication for interservice communication as well.

Data Consistency:

Eventual consistency is the norm in Microservices. Because of the widespread nature of Microservices, a single business process and transactions can span multiple Microservices. There are patterns to achieve this by relying on eventual consistency.

Dependency Tracking:

At any point, multiple teams are going to be working on Microservices. It is very important to minimize the dependencies by design. Though we can achieve it to a great degree by properly compartmentalizing the Microservices, there are going to be dependencies between them. ISVs need to have a different mindset and depend on proper tools to visualize, track and validate the dependencies. Failing this will lead to dependency chaos in a short while.

Infrastructure Management:

Instead of deploying and managing a single Microservice, there are multiple Microservices being provisioned and maintained. Hence, automate the deployments and use a single pane where DevOps can provision and manage the full infrastructure using best practices. The entire release automates via DevOps. Monitoring should be enabled to monitor all the microservices via a single pane.

Observability:

User calls can span multiple Microservices and end-to-end tracing should be enabled to troubleshoot. The core design of the microservices architecture reflects this

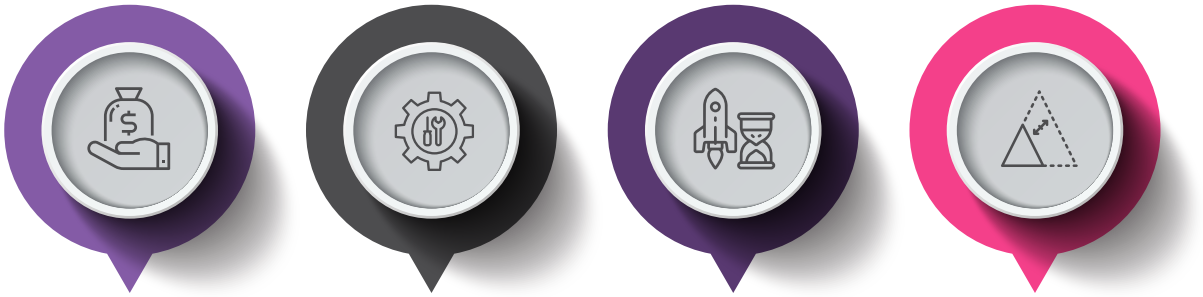
Skill Alignment

Microservices are different in the way of developing compared to monolithic and hence, the engineering team needs to get trained and their thought process needs to realign. There is a lot to learn and unlearn for every role in the engineering team right from the product owners to architects, testers, developers, and operations teams.

Optimizing Microservices Implementation

Once the Microservices migration has commenced, it is important to track the goals. If there is a significant difference between what we expect and what we achieve, there are possibilities that the ISV has missed out certain important principles. ISVs must optimize and take corrective actions.

Following are some of the factors that generally lead to issues



Operational Cost

- Improper infra management
- Gaps in skill alignment of Infra and DevOps team

Maintenance Cost

- Improper testing Strategy
- Improper release process
- Improper dependency tracking
- Insufficient Observability across microservices
- Insufficient Data handling

Release time

- Improper DevOps process
- Scrum of Scrums team is not set up efficiently
- Gaps in developer skill alignment

Scaling Issues

- Improper infra management
- Improper Microservices boundary
- Gaps in skill alignment of Infra and DevOps team

Conclusion

Microservices architecture offers a unique kind of modularization, makes big solutions easier, increases productivity, offers flexibility in choosing technologies and is great for distributed teams. ISVs must analyze their data, innovate, and launch new products and services better and faster than their competitors. They need to be flexible to meet the changing needs of their customers. We have seen how an ISV can achieve its goal of growth using Microservices and how to adopt the right approach for migration. Microservices enablement does not stop with the first step of migration. ISVs need to measure the benefits and optimize the implementations to eradicate the gaps.

We hope this white paper provides you with ideas to develop your business based on MicroServices. Aspire Systems is most willing to share their experience in software engineering with you. Please feel free to go through our website and reach out to our experts.

Contact Us

aspire SYSTEMS
attention. always.

Our core philosophy of "Attention. Always." communicates our belief in lavishing care and attention on our customers, employees and the society. We are CMMI Maturity Level 3, ISO 9001:2015 and ISO 27001: 2013 certified. In recognition of our employee-centric and flexible work culture, we have been ranked among the Top 100 Best companies to work for by the Great Place to Work (GPW) Institute for the 10th year in a row. Our customers span from 150+ VC funded start-ups to Fortune 1000 companies such as Apple, Macy's, Pearson, Hot Topic, Jack Henry & Associates, etc. We have strategic partnerships with Oracle, Microsoft, Amazon, Dell Boomi, Temenos, ServiceNow, JDA, WSO2, EpiServer, Microstrategy, Hortonworks, Yellowfin and Information Builders.

For more info contact

info@aspiresys.com or visit www.aspiresys.com

NORTH AMERICA +1 630 368 0970
POLAND +44 203 170 6115
INDIA +91 44 6740 4000
MIDDLE EAST +971 50 658 8831
EUROPE +44 203 170 6115
SINGAPORE +65 3163 3050