E-BOOK

Scale up Digital Transformation with Microservices Architecture

Contents

Introduction

Unlike the name suggests, digital transformation relies on technology, but also on various other digital transformation drivers such as people and skills development, alliance development, customer experience, digitalization and enhancement, value add across products, and innovation. In all the initiatives around digital transformation, technology is the primary enabler providing the necessary building blocks to empower change across the organization.

Microservices have proven to be the most appropriate approach towards digitally transforming a business by attacking existing technical debt, simplifying complex current scenarios, and using a clean and robust microservice architecture. Typically, the legacy applications are monolithic with a 3-tier architecture that results in the lack of scalability. But today, adopting microservices architecture is the need of the hour.

Learn more in this white paper on strategies for using microservices to achieve digital transformation.

Technological disruptions are happening and being adapted at a faster pace than ever. Every industry and businesses are transforming to change the way they operate and improve customer experience. Organizations have started adopting new and modern ways to streamline operations and provide the business with an agile and futuristic approach.

What Industry Leaders Say

"For many organizations, legacy systems are seen as holding back the business initiatives and business processes that rely on them," says **Stefan Van Der Zijden**, VP Analyst, Gartner. "When a tipping point is reached, application leaders must look to application modernization to help remove the obstacles."

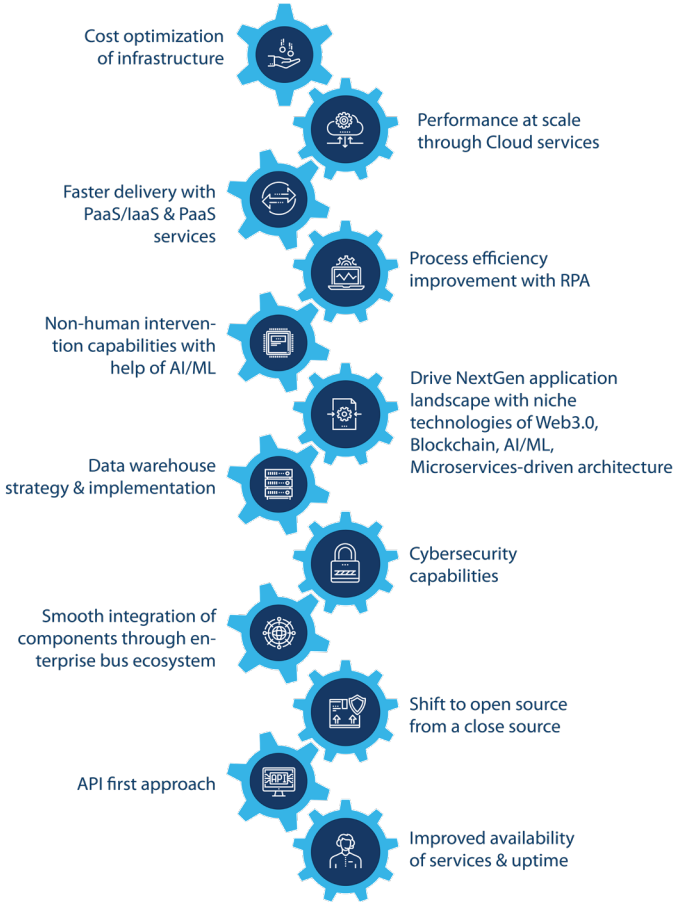
"Application architecture impacts how well an application can perform in a volatile environment," says **Aashish Gupta**, Team Manager, Gartner. "An investment in application architecture competency can yield powerful returns."

"The evolution of software engineering has expanded the scope of a software engineering leader's job," says **Mike Gilpin**, Managing Vice President, Gartner. "It's not just about delivering code. On a daily basis, software engineering leaders are concerned with resilience, scaling, security, team morale, open source, low code, managing stakeholders, emergent technologies, and legacy solutions."

"The evolutionary architect is one who understands that pulling off this feat is a constant balancing act. Forces are always pushing you one way or another and understanding where to push back or where to go with the flow is often something that comes only with experience. But the worst reaction to all these forces that push us toward change is to become more rigid or fixed in our thinking." – **Sam Newman**, Building Microservices

Digital Transformation Value Drivers

Now that we know what all factors digital transformation relies on, let us have a look at the digital transformation drivers of many organizations:



What are Microservices?

"Microservices are independently deployable services modelled around a business domain. They communicate with each other via networks, and as an architecture choice offer many options for solving the problems you may face. It follows that a microservice architecture is based on multiple collaborating microservices."

– Sam Newman

Monolith to Microservices Evolutionary Patterns to Transform Your Monolith

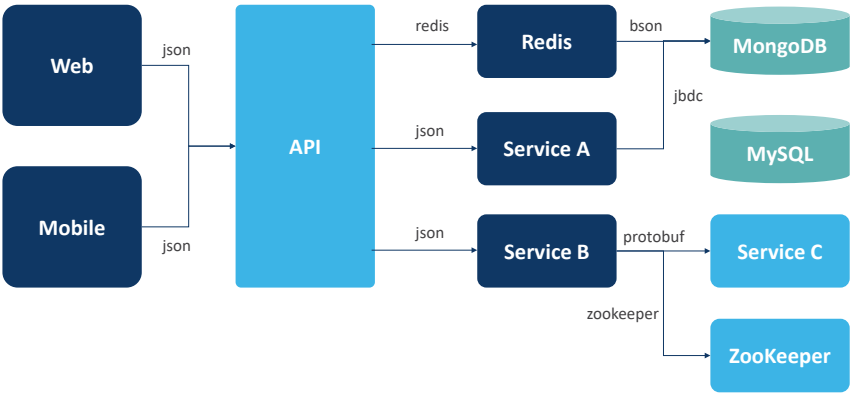
Gartner defines microservice as an application component that is tightly scoped, strongly encapsulated, loosely coupled, independently deployable, and independently scalable.

Microservices are a type of Service Oriented Architecture (SOA). It deals with one functionality and possibly one DB (database), deployed as containers running on a container orchestrator that can run multiple instances of the container as per the scalability needs. Compared to building entirely new applications, adding the functionalities are quicker and with smaller test cycles.

A microservices style is an approach to developing cloud-native software applications as a suite of small components, or services, which are designed around a single business workflow and work together.

In an Internet-driven world, where speed is the primary differentiator, companies cannot afford to waste time waiting to introduce new services and products. APIs and microservices allow teams to quickly develop new ser-

vices and get features out easily. Microservices and APIs also allow teams to break down old systems into task-specific modules.



The Microservice architecture pattern corresponds to the Y-axis scaling of the Scale Cube model of scalability.

The Emergence of Microservices

Talking about the emergence of microservices, there have been numerous claims as to the origin of microservices. Whilst vice president of Thought-Works in 2004, Fred George began working on prototype architectures based on what he called the "Baysean Principles" named after Jeff Bay.

In the year 2005, Peter Rodgers introduced the term "Micro-Web-Services" during a presentation at the Web Services Edge conference. As opposed to the traditional thinking and SOA architecture hype, he argued for REST services and discussed "Software components are Micro-Web-Services".

In May 2011, people in a workshop near Venice for software architects used the word "microservice" to describe an architectural style several of them had recently explored. In May 2012, they decided microservices was the most appropriate name for their work and formally adopted it. They had been experimenting with building continuously deployed systems while

incorporating the DevOps philosophy**.** This form of architecture quickly gained popularity.

The Cloud Microservices Market Research Report of February of 2020 has predicted the size of the global microservice architecture market will increase with a compound annual growth rate of 21% from 2019 to 2026 with the market reaching a value of $3.1 billion by 2026.

The Monolith

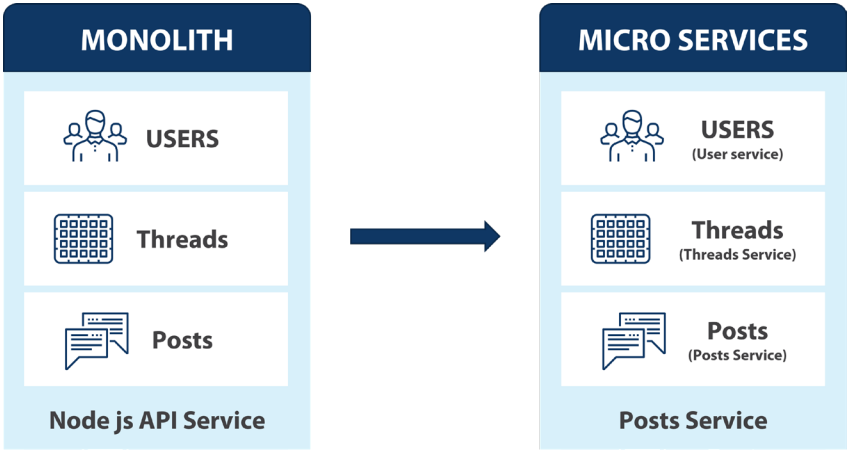
A Monolith is an application that is built as a single unified unit. Monolithic architecture is the traditional model of software program that is self-contained and independent from other applications. A monolithic architecture is a singular, large computing network inclusive of a single code base that couples all the business concerns together. To make a change to this sort of application requires updating the entire stack by accessing the code base and building and deploying an updated version of the service-side interface. This makes updates restrictive and time-consuming.

Monoliths can be convenient early on in a project's life for ease of code management, cognitive overhead, and deployment. This allows everything in the monolith to be released at once.

Monolith v/s Microservices

- Whether it is a single process monolith or a distributed monolith, it is more vulnerable to the coupling. Especially, implementation and deployment of coupling. When an organization grows, the number of teams grows, hence developers wanting to change the same piece of code and the teams wanting to push the functionality live at same time can create chaos and result in delayed delivery.
- With the monolith, you must be ready to face the challenges of delivery contention. But a microservice architecture will offer concrete boundaries in a system around which the ownership lines can be drawn, while offering the flexibility on how to reduce the problem.

- With microservices, the idea is to split your application into a set of smaller, interconnected services instead of building a single monolithic application. Each microservice is a small application that has its own hexagonal architecture consisting of business logic along with various adapters. Some microservices would expose a REST, RPC or message-based API and most services consume APIs provided by other services.



With microservices projected to grow globally at a 22.5% rate between 2019 and 2025, the choice between monolithic and microservices architectures needs to be carefully considered

Defining the boundaries of Microservices — Coupling and Cohesion

"Understanding the balancing forces between coupling and cohesion is important when defining microservice boundaries. Coupling speaks to how changing one thing requires a change in another; cohesion talks to how we group related code. These concepts are directly linked. Constantine's law articulates this relationship well: A structure is stable if cohesion is high, and coupling is low."

—Larry Constantine

What is Cohesion?

Cohesion is the degree to which the members of a certain class belong together. Simply put, the code that changes together, stays together.

It is a measure of how deeply each piece of device module functionality relates (Fenton and Bieman, 2014).

In microservices-oriented systems, a low degree of cohesion is accomplished by pooling specific business processes together, such that, if developers need to change actions, only a single microservice must be modified (Newman, 2015).

What is Coupling?

As the term says, coupling is the situation when a part of the code changes, every other thing needs to change. Tight coupling leads to changes in the structure of code, which can be quite expensive to deal with. Having to make changes across one or more independently deployable services, perhaps dealing with the impact of breaking changes for service contracts, is likely to be huge.

There are different types of coupling, but that is a topic for some other time.

Shift to Microservices from Monoliths

Deciding to shift to microservices might need a good amount of understanding of its business case scenarios, but to begin with it requires more attention and research. During planning, you should at least be able to discuss the goal with your technical partner.

1 Understanding the Goal

Understand that Microservices aren't the goal. Adopting microservices architecture should be a decision, not the goal. It will help you achieve something that isn't currently achievable with your existing system architecture. Without properly defining what you are trying to achieve, it would be difficult to inform the decision-making process about the options to choose. What you are trying to achieve by adopting microservices will greatly change where you focus your time, and how you prioritize your efforts. It will also help you avoid becoming a victim of analysis paralysis—being overburdened by choices. You also risk falling into a cargo cult mentality, just assuming that: **"If microservices are good for Netflix, they're good for us!"**

2 Benefits of Microservices

- **Independent deployment:** Each service can be modified and upgraded without touching the entire system. They can be scaled independently from one another as well if one feature is under too much load because of excess requests.

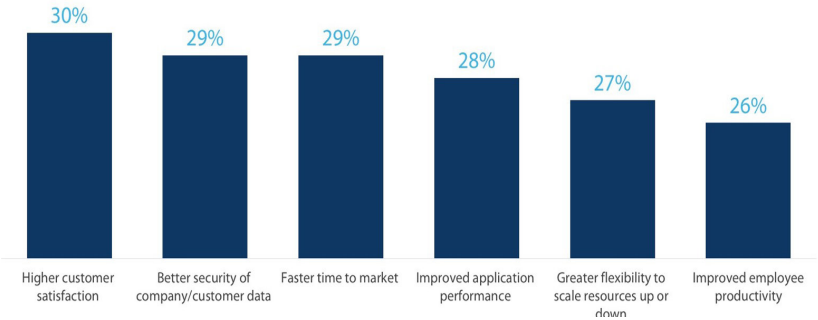
- **Lower blast radius of failure:** Clear boundaries between the services, coupled with their micro size, limit the impact of new releases and ensure fault isolation. A failure in one service does not take down unrelated functionality, with the rest of the system remaining intact and continuing to provide services to users.
- **Data Isolation:** Data sovereignty per component is an essential feature of microservices. A microservices architecture makes it possible to clearly delineate services that touch data.
- **Use of the right technology:** Microservices also simplify adopting the latest technology or integrating with third-party tools as needed.
- **Efficiency:** This model minimizes handoffs when one team needs to wait for another to complete their task, be it deployment or testing before they can start their work. With no dependence on another team, the speed of development accelerates.
- **Improved agility:** It facilitates agile DevOps and continuous delivery practices, enabling software engineering teams to increase the cadence at which they can deploy new features.
- **Improve robustness:** By using microservices, we can implement a more robust architecture because functionality is decomposed—that is, an impact on one area of functionality need not bring down the whole system.
- **Scaling the development team:** With microservices architecture, you can clearly identify the boundaries and limit the coupling with each other while having the pieces of code that can be worked on independently. Therefore, we hope we can scale the number of developers by reducing the delivery contention.
- **Embracing new tech:** With Monoliths, the technology choices are quite limited. There is just one programming language on the backend. Generally, we are fixed to a single deployment platform, one operating system, one type of database. With a microservice architecture, we

get the option to vary these choices for each service. By isolating the technology change in one service boundary, we can understand the benefits of the new technology in isolation and limit the impact if the technology turns out to have issues.

- **Reduced Security threats:** Platforms that are no longer supported and maintained can lead to system and data vulnerabilities.
- **Elimination of Technical threats:** The inability to integrate with modern applications can limit a business's ability to operate and serve customers.
- **No Business threats:** Aging systems can experience technical failures and frequent breakdowns, interrupting business operations.

Business Benefits

Adopters are reporting multiple benefits of using microservices, according to the IMB survey of 1,200 IT executives and developers. The most important microservices advantages they've felt include:



3 When Not to Choose Microservices

There are a few situations when microservices might be a bad idea. In situations of having an unclear domain, for start-ups, while developing customer-installed and managed software, and when you don't have a good reason to shift to microservices.

- **Start ups**: This may sound weird and pointless as many well-known organisations for their use of microservices are well-considered d start-ups. But as a reality check, many of these companies such as Netflix, Airbnb, etc moved towards microservice architecture later in their evolution.
- **Customer-Installed and Managed Software:** If you create a software that is being shipped to customers who then operate it themselves, it can surely be a bad choice. When you migrate to a microservice architecture, you push a lot of complexity into the operational domain.
- **Not Having a Good Reason:** This is probably one of the biggest reasons to not adopt microservices. If you do not have a clear idea of what exactly it is that you're trying to achieve.
- **Not Having Clear Domain:** When there is a lack of clear and precise domain ideas, , choosing microservices can lead to a disaster. Even worse than having a single monolithic system.

When Enterprise Should Shift from Monolithic to Microservices

Even after the adoption of mini services and agile DevOps and continuous delivery practices, if you still can't achieve the software engineering cadence goals, then it may be time to adopt a microservices architecture. The perfect time for an organization to migrate from monolithic to microservices is when your organization is growing and is facing productivity issues.

When an organization is at a smaller scale it usually practices following monolithic architecture. In today's world of nimble competition, everything as a service (XaaS) has brought significant challenges to the established organizations that once dominated their markets.

For any organization to have a distinctive and comprehensive architecture, shifting to microservices is one of the finest choices.

Expectations from Microservices Architecture While Considering the Shift from Monolith

1. Componentization via Services

A component is a unit of software that is independently replaceable or upgradeable. By using microservice, the application is divided into separate components as services that can be deployable independently and run on their own process means that each component can be scalable independently and failure on one component will not affect other components in an application.

2. Organized around Business Capabilities and/or Sub Domains

Usually, architecture is organized around technology layers (i.e., UI layer, server-side logic, database) but in microservice, architecture is organized over business capabilities and subdomains.

In Business capabilities, the application focuses on business values and objects (i.e., Project management, Document Management). In Domain-driven design (DDD) subdomains, the application focuses on problem space - the business as a domain. A domain consists of multiple subdomains. Each subdomain corresponds to a different part of the business.

And organizing architecture around business capabilities and/or sub domains gives:

- Stable architecture since Business capabilities and subdomains are relatively stable.
- Incremental development and evolving architecture continuously (i.e., possible to implement continual architecture—building an architecture that has no end state and is designed to evolve with an ever-changing software development ecosystem.)

3. Products not Projects

The product mentality ties in with the linkage to business capabilities. Rather than looking at the software as a set of functionalities to be completed, there is an ongoing relationship where the question is how software can assist its users to enhance business capability.

The smaller granularity of services can make it easier to create personal relationships between service developers and their users.

4. Decentralized Governance

Systems should be language agnostic. One component can be developed in Java, and another can be developed in .NET. The decision of choosing a technology platform for a particular service should not affect the application architecture.

In other words, microservices can allow you to embrace different technologies more easily. But embracing multiple technologies does not come without overhead, so we should place some constraints on language or platform selection.

5. Decentralized Data Management

Systems should have a decentralized database. Ideally, each component or microservice should have its own database, with whom only that service is interacting. No other component or service can fetch or modify the data in that database.

This isolates the impact of schema changes. Development teams can be more in control of the impact when planning releases.

6. Infrastructure Automation

Embrace the concept of DevOps. Build it, Run it.

Each component in the system should be cohesive, independent, and self-deployable. It should not be dependent on any other component

or resource to work or deploy. It should have Continuous Integration/ Continuous Deployment (CI/CD) in place to ship faster.

The system should have automated testing in place. Speed is one of the most desirable features of microservice architecture. In the cycle of build, test, and ship, if automated testing is not in place, then it cannot meet the goal.

7. Design for Failure

Any component/service failure should be in isolation. Failure of one service should not make the whole application go down. If it fails, it should not affect other components/services. A failure rollback mechanism should be in place. That means if one service fails, it should be easy to roll it back to an older working version.

A Few Approaches for the Migration of Monolithic to Microservices

1 Domain Driven Design

The migration of a big monolithic application to a microservice architecture is a promise made by IT to business to reduce cost, increase operational efficiency, and gain competitive advantage. Architecture design holds the key to success with a forward-looking vision with a 5 to 8-year timeline.

In domain driven design approach, the functionalities are divided based on the domains they serve with a bounded domain context where the entities talk to each other. This way, defining a clear set of domain objects [aggregates] and the entities to avoid chatty services is possible. In subdomain decomposition, we will go a step deeper, where we decompose the applications based on subdomains in each business domain with loosely coupled services adhering to common closure principle. For example, a business in the manufacturing and retail domain may have major domains, such as manufacturing, warehouse and retail. In retail, it may have order, price, customer, loyalty, etc.

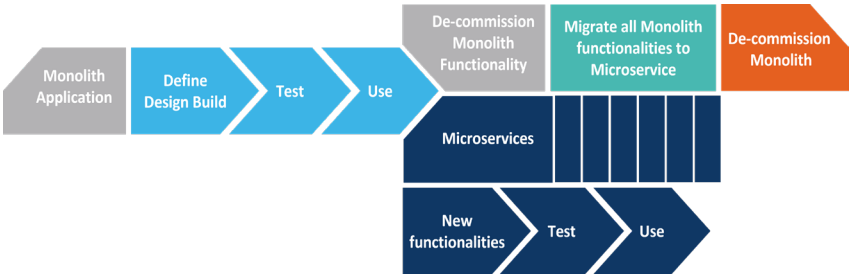
To decompose monolith to microservices in domains and subdomains, it is good to follow the below mentioned steps at a high level:

- Detailed domain analysis
- Draw clear boundaries between domains (Make it crisp and avoid smoky boundaries in the definition stage, do not rush as this step may consume some time)

- Entity definitions bound to the domain and made simple
- Identify access patterns of the domain entities
- Identify orchestration needs of the entities as per access patterns
- As per all the above, define a microservice

Implementing DDD approach directly in a big bang way has a high initial risk approach. The chance of failure is higher when converting the existing production application to microservice based architecture in a single shot. The risk increases multiple folds, considering the non-clean legacy coding patterns and multiple bug fixes applied over the time by relatively less domain specific development teams.

2 Strangler Pattern



Talking about another approach, strangler implementation approach is one of the better and safer implementation approaches. In the beginning, one must consider refactoring and building specific functionality of the big monolithic application to the modern microservice architecture, which can still be a part of a business domain as per our DDD approach.

Once the functionality is developed, tested and ready to use, strangle the same functionality in monolithic application and route the requests to the newly built microservice.

The Advantages of this approach are:

- Less risky and more stable application migration
- Zero business impact, gradual and silent migrations
- Can implement test driven development methodologies
- Co-existence of monolithic and microservices
- Easy failover

Cygnet Digital's Approach to Digital Transformation with Microservices

Most of the time, when Cygnet Digital engage with clients to support them in their Digital transformation journey, the clients are usually looking to address the digital transformation drivers/pain points mentioned above. However, they end up most of the time in an engineering activity unless it's just a human process inefficiency, but even while they address human inefficiency and if their business landscape allows, they move to process automation tools for redundant processes and can improve their process efficiency with reduced cost.

Carry out your digital transformation initiatives effectively and maximize the value of your technology investments.

Find out the hidden potential in your business, systems, and processes.

GET CONSULTED BY OUR ADVISORS

Discovery

1. First phase of the digital transformation mandate starts with:
   - Understanding the client's business environment
   - Management commitment on the client side
   - Agreement on the scope
   - Architecture principles
   - Governance
2. Transformation vision as **"THE END GOAL"**.
3. Market research on micro-trends, competitive dynamics, and technology advancements to ensure Roadmap is in line with current market needs.
4. Identification and articulation of business scenarios.
5. Gap analysis—Process, Data & Code.
6. Assessment and audit of the existing tech stack:
   - **Architecture:** High-level architecture and components
   - **Code:** Audit of source code to detect code errors, vulnerability, and compatibility with the new platform
   - **UI/UX:** Assessment of user interfaces, supported operations, and process
7. Choosing the right migration approach for shifting to microservices:
   - **Big Bang:** Lift and shift approach where the entire application is re-hosted to the cloud in a single milestone. The Big Bang approach offers a shorter implementation time and is considered perfect for organizations that utilize smaller, non-complex workloads.

- **Phased:** Here, the application workloads are shifted to the cloud in multiple, smaller milestones implemented over a period.

8. Choosing the right Tech Stack/Cloud offerings: Pragmatically choosing the right cloud service:
   - **IaaS:** To acquire infrastructure resources such as storage, networks, processors, and servers on demand
   - **PaaS:** Managing the hardware and operating systems to focus on developing codes and automating deployment pipelines
   - **SaaS:** Development of applications to be offered over the web
9. Envision the future state and define the IT ecosystem roadmap
10. RACI matrix definition for right governance
11. Performance metrics/KPIs to measure outcome
12. Documentation of architecture both "AS-IS" and "TO-BE".
13. Alignment of data dictionary with application should be interwoven to avoid data leakage.
14. Document NFR's with right KPI's.
15. Establishment and development of the transformation roadmap covering build vs buy decision-making.

POC development

1. Foundation building blocks
2. Delivery of POCs
3. Re-assess long-term goals and IT ecosystem roadmap
4. Updated IT ecosystem roadmap

Incremental development

1. Requirement analysis
2. Foundation building blocks
3. Release plan
4. Incremental deliveries
5. Re-assess long-term Goals and IT ecosystem roadmap
6. Updated IT ecosystem roadmap

Data migration (Brown Field Perspective)

1. Understand "AS-IS" entity's / table & record of business
2. Overlap with application migration strategy, data follow's service for business continuity
3. Parallel run approach irrespective of going big-bang or incremental
4. Post go-live assessment for identify data leaks

Cygnet Infotech firmly believes that Digital transformation is a journey that should be incremental in nature and digital engineering should be in place to fulfil this journey to ensure both go in tandem to fulfil the need of businesses and reach the end goal for the purpose-driven organization.

During the process, the clients are also suggested to keep evaluating their processes and IT landscape and identify what is working and what is not, and at right time take the necessary steps to get on track and emerge as a leader in their respective business domains.

Gain a detailed inside roadmap, achieve architectural excellence, and areas of improvement

Book for Technical Due Diligence

The steps we take during the migration phase— Shift to Microservices from Monolith

1. Everything is changed by breaking up the monolith and altering the way all developers and ops teams work in the organization. The new processes are explained to the concerned stakeholders to get the team on board.

2. Clients are advised to accomplish this migration incrementally. This allows the monolith code to run alongside the new application that consists of microservices. Decouple the services with a domain-driven design. Try to gain a good comprehension of the problem space prior to moving into the solution space. Before redirecting them to the new service, extracting the capability's data, logic, and user-facing components is essential. Also, evaluate the cost versus benefits.

3. Prioritize the services for migration based on the type and scale of dependency and ensure that while migrating the services incrementally, communication between services and monolith is configured through well-defined API contracts.

4. Over time, the functionality implemented by the monolithic application reduces. It either transforms into another microservice or disappears altogether.

5. Services in a microservice architecture is planned around business concerns and not technical concerns. So, there may be a need to create larger services instead of smaller services and defining service boundaries may be an iterative process.

About Cygnet Digital

Cygnet Digital is a leading Digital Engineering Services company that works with its client to achieve their digital transformation goals with a core focus on modernizing their systems while leveraging Microservices architecture. We aim to give clients an ideal technology partner experience that empowers them to earn customer delight at each stage of their customers' journey.

We make sure the capabilities delivered by software products are modular, rapid, and safely assembled, disassembled, and recomposed as per evolving business, customer and market needs with faster delivery while being flexible. We help scale businesses to thrive in the competitive market using a consultative, customer-centric approach. Our solutions range from standalone bespoke development and managed services to building connected ecosystems across the enterprise and developing smart systems by leveraging emerging technologies like AI, Blockchain, and automation.

Closing Summary

Adopting microservice architecture involves various steps and many decisions must be taken on various layers. We suggest that organization must plan it properly, select the right tools, select the right cloud partners, use cloud features to avoid re-inventing the wheel, calculate the cost including cloud usage costs, and as far as possible, be open to using cloud environments to optimize usage. The microservice maturity model should be considered along with the IT roadmap.

Cygnet Digital

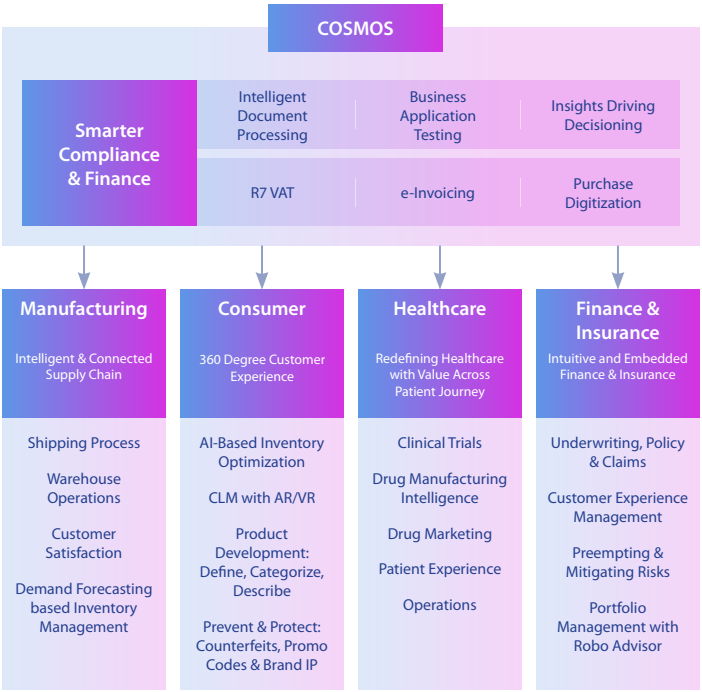
LIVING THE TRUST

Cygnet Digital is your dedicated digital transformation partner. At Cygnet, through our proprietary framework, Cygnet COSMOS, we empower organizations across the spectrum of business intelligence, and customer experience.

Our comprehensive suite of services encompasses Domain Consulting, Digital Engineering, and Enterprise Applications, with a keen emphasis on Digital Commerce and Experience. We harness the potential of Data, Analytics, AI, IoT, and Automation, coupled with AI-powered Testing as a Service, to drive your transformation agenda.

With flagship products like Tax Transformation, Cygnet IRP, and Finance Transformation, we are dedicated to a digitally enabled future for your enterprise.

What sets us apart is our unwavering commitment to a **"Business First"** approach. We're not just developing applications; we're co-innovating with you to forge the future.



GET IN TOUCH