GlobalLogic®

Strategies for

Digital Transformation

With Microservices

www.globallogic.com

BOTTOM-UP AND TOP-DOWN APPROACHES

Digital transformation is a process that relies not only on technology, but also on other digital transformation driving forces such as people and skills development, alliance development, customer experience digitalization and enhancement, value add across products, and innovation. In all digital transformation initiatives, technology is the primary enabler that provides the necessary building blocks to empower change across an organization.

Microservices have proven to be a bullet-proof approach to transforming a business by attacking existing technical debt, simplifying complex current scenarios, and using a clean and robust microservice architecture. In doing so, it effectively replaces a part of an existing legacy system with a next-gen software that supports microservices by following the strangler pattern. This pattern keeps transforming the existing solution until the whole monolithic legacy application has been migrated completely and decommissioned.

This kind of approach, in which technical debt reduction is the ultimate goal, is usually called a bottom-up approach for digital transformation. Such transformation usually occurs at the bottom of an IT pyramid, where integrations and platforms replace existing technical debt.

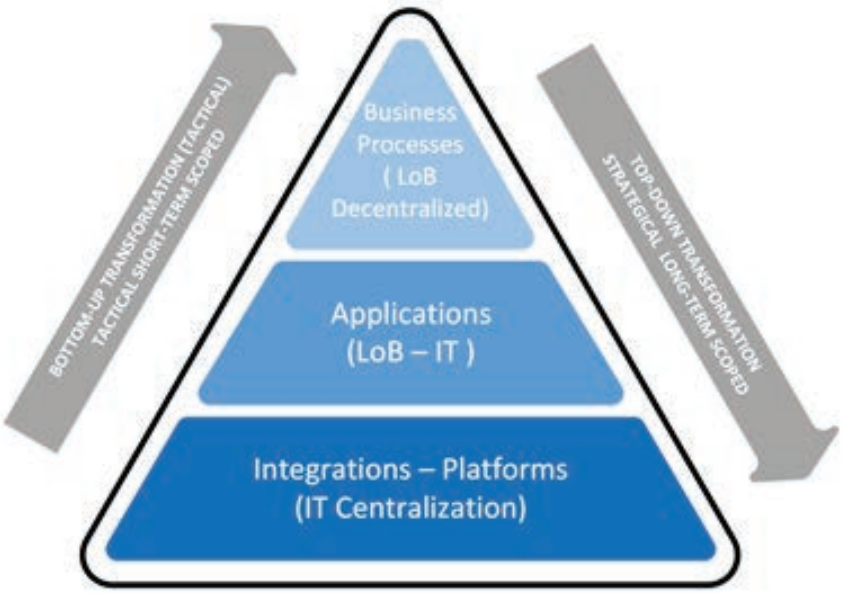

Figure 1: Approaches for Digital Transformation with Microservices

A bottom-up strategy mostly follows a brown-field approach for development, with software pieces belonging both to the legacy system and the next-gen microservice architecture. They both co-exist for some time, until the migration of corresponding legacy system to microservices, is complete. Characterised by a strangler pattern approach, the bottom-up strategy is most likely tactical and designed to deliver fast and short-term results.

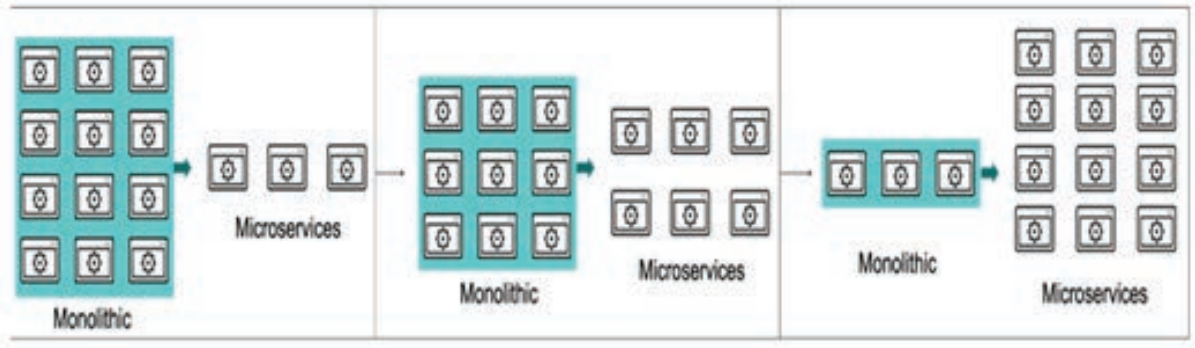

Figure 2: Strangler Pattern for Microservice Migration

For example, if scalability, availability, consistency and performance quality are compromised, a rearchitect-and-shift approach can be followed, in order to re-engineer processes and make them more robust. Once the re-engineering is complete, it can be moved to the cloud. This is one of the most widely-known approaches to start transforming a business by reducing technical debt.

Another important benefit of the rearchitect-and-shift approach is that it leverages existing cloud capabilities; reducing time, effort and the total cost of ownership (TCO) for the solution. Some candidate use cases must be identified to further develop a rearchitect-and-shift strategy on how to re-engineer the use case with an MSA approach.

On the other hand, the top-down approach is strategic. It starts with transforming the business domains and processes, by re-designing a part of a business process using microservices. This approach is mostly considered for mid-to-long term engagements, involving complex techniques such as domain driven decomposition (DDD) and event storming. In such cases, existing business domain applications can be eventually removed, merged, increased in size, or transformed as necessary, based on high-level goals and long-term objectives.

In this business-driven approach, pain areas must be identified first, for prioritization and cost/impact analysis. The scope could extend to the whole business which may include not only technology, but also a re-design of the business processes and business re-architecture. Cost/impact analysis must be performed on a case-by-case basis.

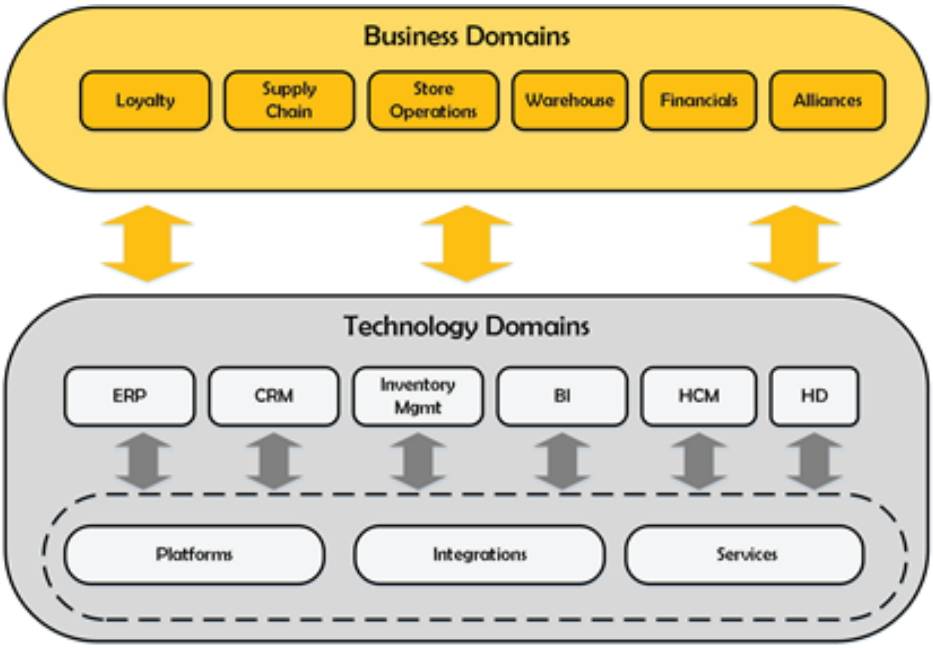

Figure 3: Sample Technology and Business Domains

BOTTOM-UP STRATEGY OR "TACTICAL" APPROACH

Following a tactical approach, the first steps towards business transformation would be to reduce technical debt and remove any factor that could be deterring business expansion. This kind of approach is usually led by digital experts.

An organization can have numerous and varied pain-points across different business processes and business units. These pain-points can be directly related to the existing technical debt. An organization is probably relying on technology which is a decade or two old and was relevant in the past. However, due to digitalization 2.0, it might not meet requisite quality attributes such as platform scalability, performance, consistency, reliability, robustness, customer experience, sensitivity and latency. Consequently, an organization struggles to meet customer expectations.

A rearchitect-and-shift approach with event-driven microservices is recommended to:

- Reduce coupling between components
- Leverage the benefits of microservice approach
- Reduce the overall time-to-deliver
- Develop scalability with independent agile teams
- Increase overall solution reusability by means of an API-led design and reduced friction
- Re-design and optimize the use case

The most effective candidate use cases for rearchitect-and-shift approach are those that do not involve business logic and are basic, scoped around backend-level integrations. If a candidate use case involves the transformation of business logic, it should be considered for the top-down approach instead.

Another strategy, lift-and-shift, focuses on moving the existing software, with minimal configuration changes, to the cloud. This strategy is suited for software monoliths that are still on-premise and are not likely to get re-architected; but, there are strong reasons for moving them to cloud. One of the probable reasons would be cost. The cost of keeping the hardware up and running on-premise, along with the indirect costs including refrigeration, power, back-up, outsourced server maintenance procedure and licensing, can easily increase the TCO.

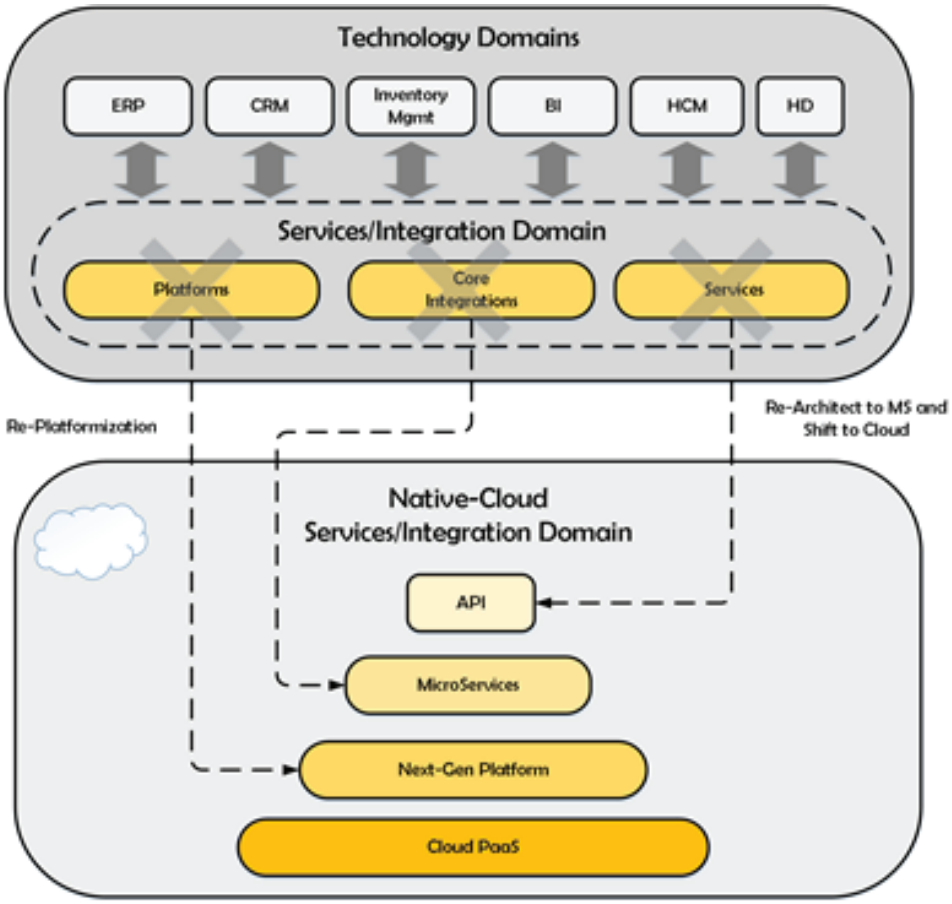

Figure 4: Rearchitect-and-Shift Approach

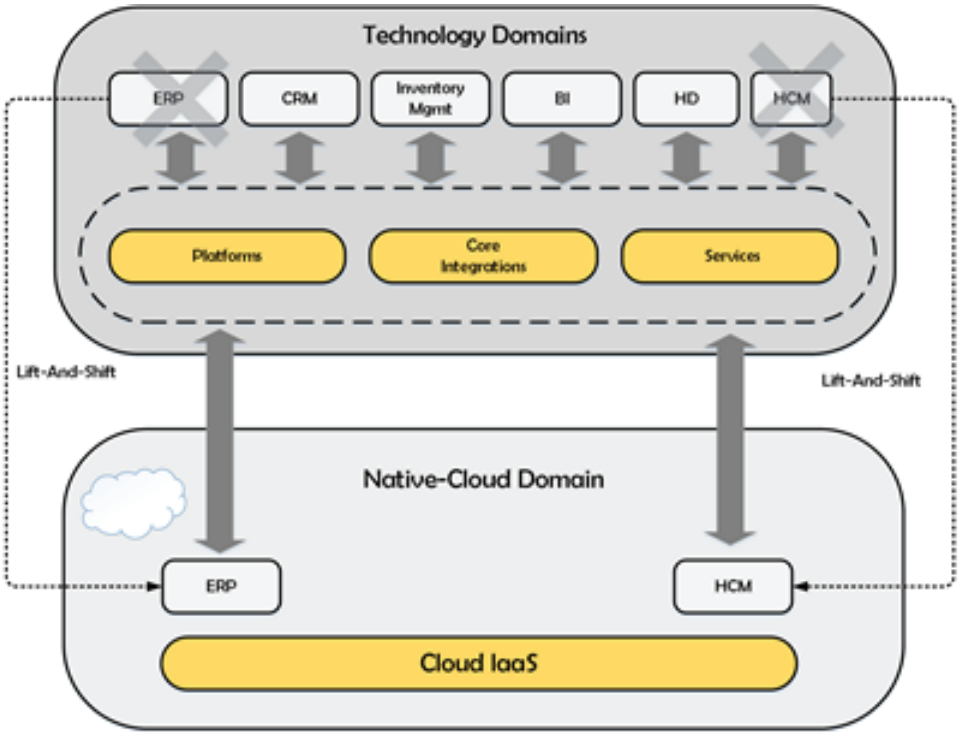

Figure 5: Lift and Shift Approach

BROWN-FIELD AND IMPLEMENTATION REUSABILITY APPROACHES

When rearchitecting or lifting an existing use case, a brown-field approach should be followed. The re-engineered piece(s) of software may coexist with the existing legacy functionalities, even when the code has been refactored to microservices. These microservices must still interact with existing services and components to some extent.

In the case of rearchitect-and-lift strategy, a part of the existing use case implementation may be re-used, to some extent. However, the intention is to transform into microservices as much as possible but there are cases where some components may still remain unchanged due to different reasons.

While planning, three different reusability approaches can be considered, depending on the extent of usage of the existing implementation:

- **Black-Box Reuse:** Reuse: In black-box reuse, a component is directly reused without modification. A component can be reused as it is or reused through inheritance if the programmer creates a specialized subclass of an existing class component. This approach helps with easy maintenance and evolution of software systems.

- **White-Box Reuse:** In white-box reuse, programmers reuse the component after they have modified the components to their needs. White-box reuse does not facilitate easy maintenance and evolution of software systems but it can reduce development time.

- **Glass-Box Reuse:** In glass-box reuse, programmers do not directly reuse the component; instead, they use it as an example for their own development. For instance, programmers can look at examples to find out how a program plan is realized and build their own system through analogy.

TOP-DOWN STRATEGY OR "STRATEGICAL" APPROACH

The strategical approach is targeted at the business domain level. Following this approach, business domains and specific areas should be identified as candidates for digital transformation. Additionally, it requires a technology shift and a complete business re-design. This approach is usually led by business experts followed by digital experts.

Such a transformation demands different capabilities such as:

- **Business/Enterprise Architecture Areas:** In order to understand how the current domain area can be re-architected for business and aligned with the overall business vision and goals, not only enterprise/business architects, but also specific business domain stakeholders must be involved at this stage. This will help get consensus and ensure consistency. Business architects should also deal with the inherent risk and find ways to mitigate it over time. Enterprise architects must have knowledge of the business domain and the business logic under revision.

- **Business Analysis Areas:** In order to understand the impact of the changes being proposed, analysis is essential. Some changes can indirectly and unintentionally impact other business domain areas and complicate the initiative. Also, business analysts should be involved in order to foresee how the proposed change will affect the business value chain and take appropriate actions based on the outcome.

- **Solution Architecture:** Solution architecture represents the IT/technical mapping for the change initiative that is being proposed by the enterprise architecture team. It is probable that a proposal will be challenged, based on concerns. Such factors must be resolved before embarking on a business change. A solution architect has a wider vision, including constraints, risks, limitations, technology stack and budgeting concerns. A solution architect works on building the solution and technical blueprints based on the high-level enterprise architecture requirements, considering all the factors involved, to reduce any existing gap that may lead to a poor execution. So, maintaining high standards is a must for them.

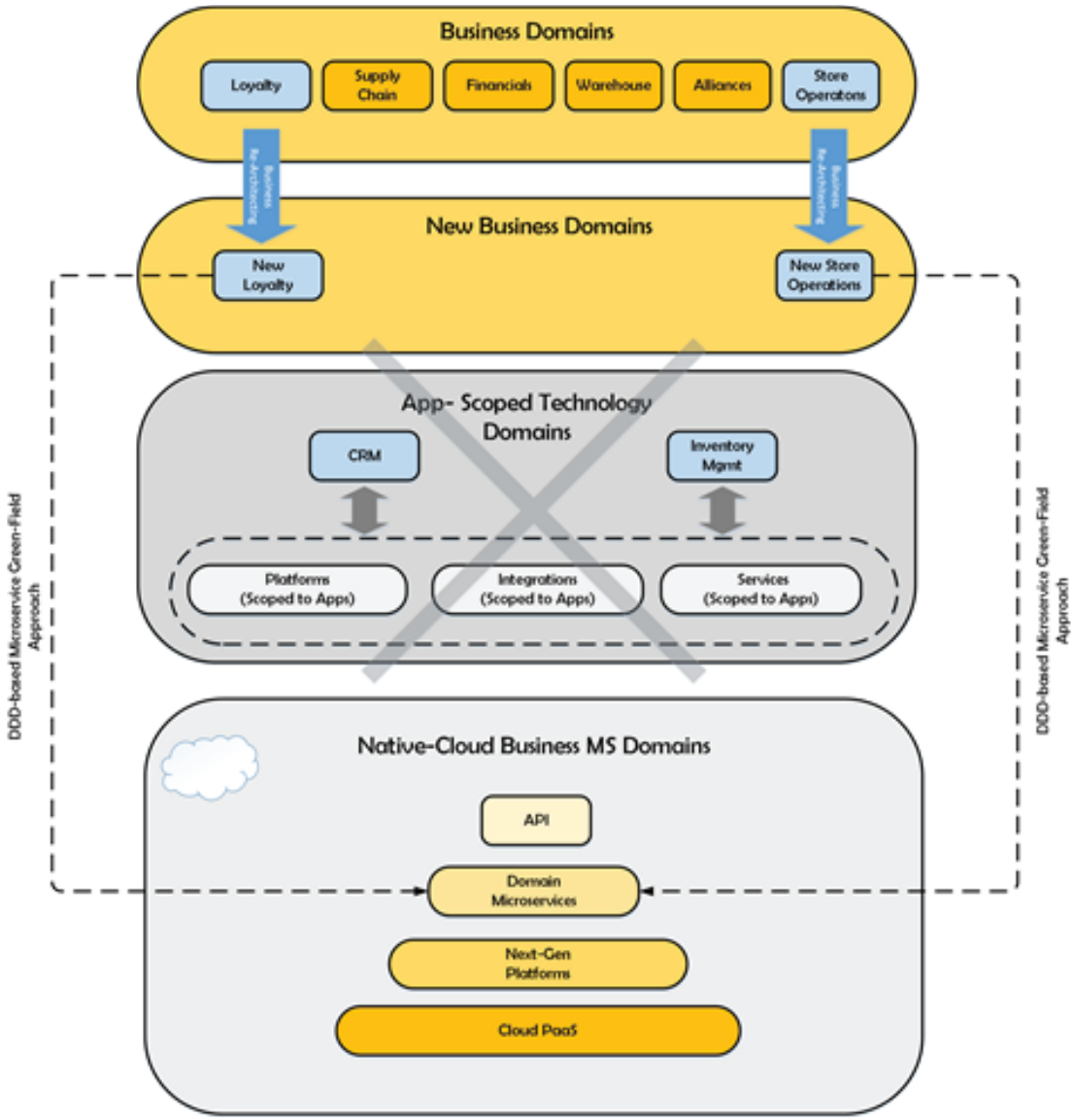

Figure 6: Top-Down Approach

DEVOPS CHALLENGES FOR MICROSERVICE APPROACH

Any digital transformation engagement with microservices is tied, to some extent, to the existing DevOps capability in the organization. This is required from the very beginning of any initiative, in order to set the basic infrastructure on which microservices can be deployed, evolved, monitored and eventually retired.

There can be multiple duties that are typically delegated to the DevOps capability, depending on the specific stage of the software delivery life cycle (SDLC):

- **Continuous integration/continuous deployment (CI/CD) Pipeline Setup:** A robust SDLC relies on a robust CI/CD pipeline to ensure that only solid, robust and secure codes are being pushed. It is DevOps' responsibility to set it up properly, either by leveraging some components on the cloud or by deploying a complete CI/CD on-premises, by following specific requirements. This is usually where DevOps invests most time and to some extent, can be considered a bottleneck, leading to delays in development.

- **Environment Setup:** While migrating existing business capabilities to microservices, specific environments may be required to deploy and test them. The exact number of environments have to be provisioned for – either on-premises or cloud-native, depending on specific project requirements and constraints. However, it's not unusual to find multiple environments being allocated for Development, Testing, Staging, QA and Production. Setting up each environment, either manually or programmatically, is the responsibility of the DevOps team, and they must make provision for any necessary resource as required, such as elastic filesystems, queues, database tables, bucket storage, compute capacity, or even a complete analytics platform. Environment setup is time intensive and clearly requires maximum effort from the DevOps team to set it properly, and keep it running.

- **Troubleshooting CI/CD components:** As microservice code gets pushed into a central repository, it is built and deployed in a target environment by means of an automated CI/CD pipeline. If problems arise through the pipeline itself, it would be the DevOps team's responsibility to perform proper root cause analysis (RCA) and resolve the issue. In the process, the CI/CD pipeline improves over time and becomes a mature CI/CD pipeline, which can detect issues sooner and avoid major outages once the pace of digital transformation increases.

- **Troubleshooting target environments:** It's not unusual to start disclosing issues on the infrastructure already provisioned for by DevOps when the first microservice or a couple of microservices are being deployed to a target environment. In fact, it is expected to happen to some extent. Since resources were provisioned completely from scratch, there could be multiple reasons for which they could stop working such as mis-configurations, bad resource provisioning on the platform, and bugs on either the on-premise or on-cloud platform. Issues can deter project progression, preventing features from being delivered by the due date and that makes troubleshooting one of the most important duties of a mature DevOps team.

- **Identifying code issues at runtime:** Even with a robust CI/CD in place, with CAS and SCA well implemented, a problematic code can get pushed and delivered to the pipeline, and eventually, the target environment. A bad code can cause multiple issues like ridiculously spiking CPU, memory footprint increasing over time due to memory leak condition, and race condition in threading caused by unwanted interactions of code with non-compatible libraries. Sometimes, such issues require time and a deep-dive RCA to figure out the real problem and not just the symptom/s.

- **Set Basic Monitoring**: Real-time monitoring of component and resources is essential to react to notifications and alerts rapidly, based on pre-configured monitoring template, deployed either as application performance management (APM) or other cross-platform monitoring solutions. Monitoring is especially important for upper environments such as user acceptance testing (UAT) or Production, in order to detect metrics that might go out-of-thresholds and possibly, correct the condition either automatically, in the form of corrective control, or manually. While APM is the responsibility of an architect, DevOps is responsible for installing, deploying and setting up the appropriate monitoring templates, based on requirements from SecOps or Operations.

IMPORTANCE OF DEVOPS ROLE IN DIGITAL TRANSFORMATION

In large digital transformation initiatives, following Agile or scaled-agile (SAFe) methodologies, with multiple scrum teams running across Agile Trains, hundreds of developers could be divided into small scrum teams of 5-10 individuals each, with each team being responsible for the development of one or two microservices. Such teams may have multiple needs from the DevOps teams, not only in terms of CI/CD and code (which increases in complexity and lines committed over time), but also infrastructure/components. In such a scenario, a single DevOps team can hardly cope with the increasing demand and the only option is to scale-up.

DevOps can face limitations in the case of smaller projects as well. The most common would be the lack of organizational DevOps capability planning for the long-term. It is usual to find many engagements where DevOps capability starts growing but due to one reason or another, gets downsized or even dismantled completely. This could be due to budgetary or high-level business management decisions.

It is important to point out that DevOps is a discipline in itself. DevOps professionals generally have significant experience in Operations, Development and QA, such as an SME in the Ops side rather than on the Dev side. Undoubtedly, DevOps engineers must have knowledge about coding languages, debugging codes and performing deep-dive on performance issues. DevOps teams also have a strategic role to play as they become the first point of interaction for a Dev team for coding best practices and coding patterns.

Projects can even get dismantled because of multiple performance issues, lack of code quality, unstable platforms, and a decrease in software delivery pace. Following this, the DevOps capability can be diminished or dismantled. Unfortunately, this happens too often, especially in projects where the vision and importance of a solid DevOps discipline hasn't been properly introduced to the client in the early stages.

DevOps not only supports Dev teams, by looking into the details of infrastructure/environments and platforms operations, but also takes care of issues that arise on a day-to-day basis. They can also check the quality of a software being produced, by becoming coding advisors and developers, besides SME on Operations.

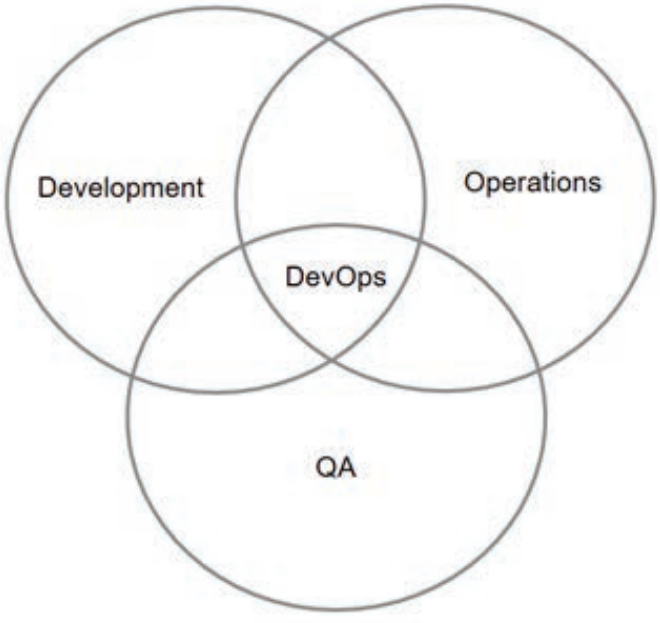

Figure 7: DevOps as the convergence of Dev, Ops and QA

DEVOPS LIMITATIONS ON DIGITAL TRANSFORMATION COMMITMENT

One of the most important yet undervalued tasks of DevOps is to get involved in setting up an environment. When we say environment, we can refer to anything – from a complete CI/CD pipeline, which is required to set up SDLC automation, to a whole set of PaaS resources, frameworks or platforms, required by the software. In our case, it is microservices.

A microservice can be resource-intensive. If you are going to deploy a microservice toperform event-based communication, you need:

- To make provision for an entity partition for entity persistence
- An event hub and topics for event-based communication
- An API Proxy if you want to expose the microservice
- A Kubernetes cluster's namespace for processing
- Different components that make up the microservice chassis

Some other microservices may require an analytics engine instance, a data warehouse instance, a database with some pre-existing tables and a real-time speed data layer.

Deploying and maintaining such components can be time consuming and lead to issues as the project scales up, and the number of Dev teams and microservice applications increases. As the process of digital transformation progresses, the pressure on DevOps increases, often reaching an inflection point, where DevOps cannot deal with the backlog. If DevOps cannot scale, it can become a bottleneck and stall the digital transformation commitment (DT Commitment). This is the risk that should be avoided because delaying a DT Commitment (percentage of the whole digital transformation, at the program level, that has been accomplished already) can prove to be expensive.

GETTING RID OF DEVOPS LIMITATIONS BY MEANS OF PLATFORMIZATION

Instead of provisioning, customizing and tailoring the infrastructure stack to the specific needs of a project manually, an automated delivery methodology can be used.

An automated infrastructure stack delivery platform (IDP) takes care of programmatically defining the stack by means of a specification that needs to be written by the end user. Once defined, it can be repeated as many times as required, creating multiple stacks based on the same definition, reducing the effort required to create different environments with the similar stack, components and configurations in place. This saves time and cost, and consequently reduces the time-to-deliver for new solutions because the infrastructure can be created by means of a specification instead of creating each component manually.

In terms of IDP, a specification can be defined through a template, which, written in a specific pre-defined normalized language, can be used to create any necessary resource following the infrastructure-as-code (IaC) paradigm

Platformization is the concept for which an automated infrastructure delivery platform is leveraged in order to speed up the SDLC, by reducing the time required to provision for environments and components, both on-premises and on cloud (PaaS). This includes PaaS components, configurations and ready-made codes that can be used as an accelerator to get something up and running on time. The platformization approach has proven to drastically reduce TCO for SDLC, ensuring timely ramp-up of solutions. It also reduces time-to-deliver for new features to be introduced and increases the delivery rate for software and digital transformation.

Platformization also empowers business by leveraging technology for digital transformation, leading to a business-driven technology approach instead of the technology-driven business approach. The latter, although the most popular approach for years now, limits business expansion due to the existing infrastructure and technology.

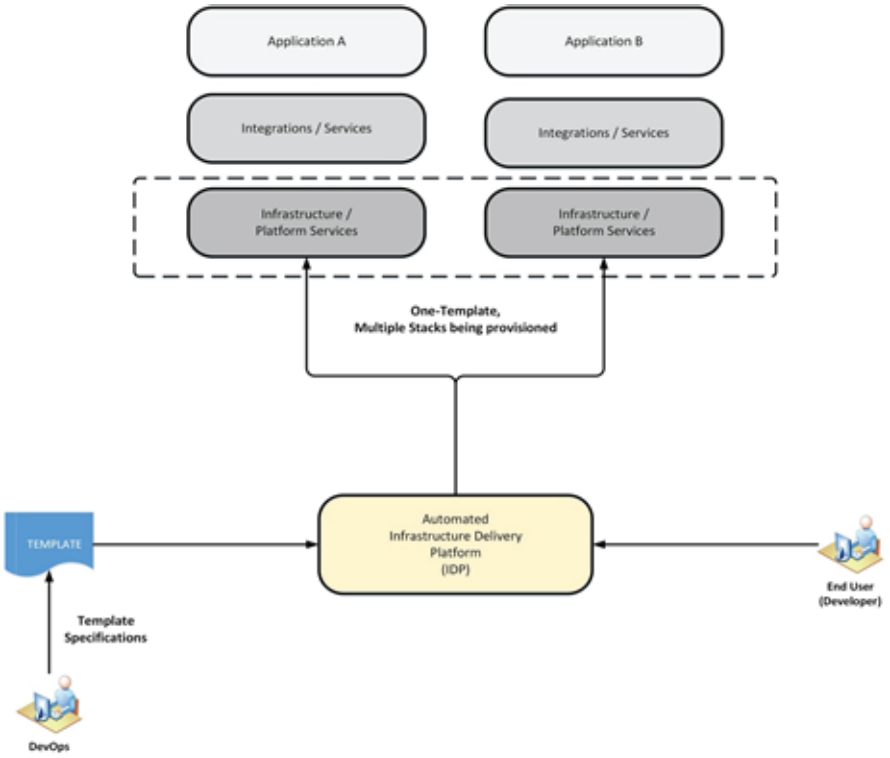

Figure 8: Platformization Concept

The diagram on next page (Figure 9) represents the total DT Commitment with passage of time. The gold curve represents the real DevOps backlog. The red line represents the DT Commitment without platformization – the plateau areas represent time required by DevOps to fully provision artifacts and resources. Once provisioned, the DT Commitment is increased. The blue line represents the DT Commitment employing the platformization approach. It would be good to note that this line stays close to the gold line which implies that DT Commitment stays close to backlog. The dotted green line represents the DT Commitment threshold, that is reached once the DevOps team reaches the inflection point.

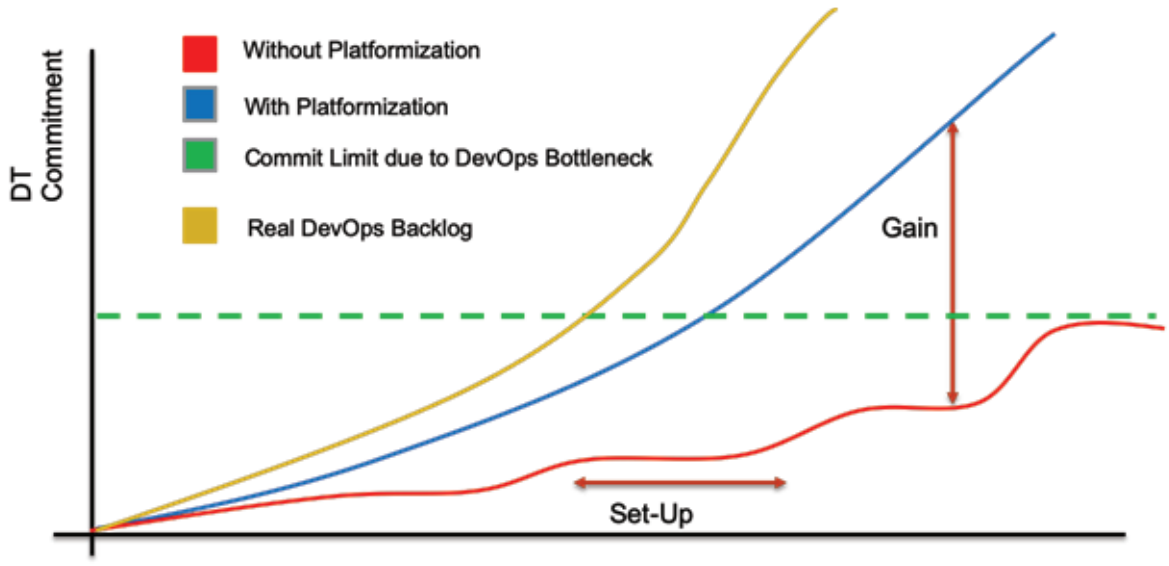

Figure 9: Platformization vs Non-Platformization

PLATFORMIZATION ADOPTION

One of the most important goals of platformization is to empower developers and help them create necessary infrastructure in a short span of time. On the other hand, in traditional N-tier deployment models (as shown in Figure 155), full DevOps capability is required to get all the necessary artifacts provisioned across different applications, in a timely manner. Shortening the provisioning cycles decreases the overall time-to-deliver and shortens SDLC as well.

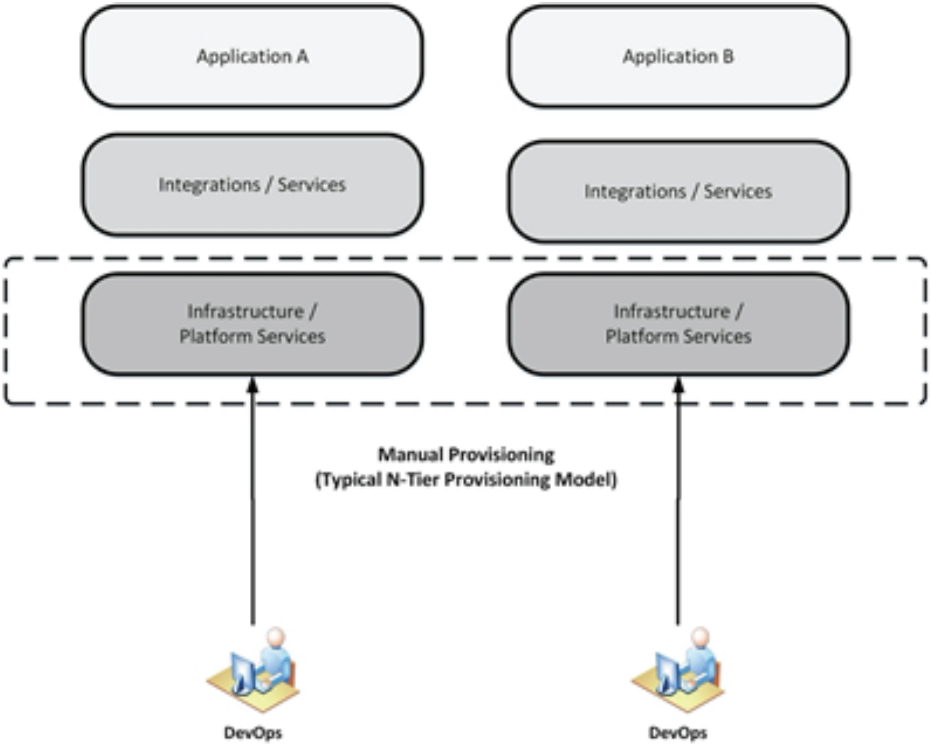

Figure 10: Traditional N-Tier Provisioning Model

Having said that, it is important to mention that in order to guarantee success in platformization approach, it should be adopted across an organization, especially for large-scale digital transformation engagements.

Simply put, IDP should be promoted within the organization through formal pre-sales activities to ensure that all the teams are aligned with the vision and value of such a platformization effort. Without complete adoption, outcome can be compromised and unsatisfactory. When adopted completely and implemented properly, an organization can benefit from the overall digital transformation approach, by reducing TCO, relieving DevOps backlog pressure, and maximizing the pace of DT Commitment.

About GlobalLogic

GlobalLogic is a leader in digital product engineering. We help our clients design and build innovative products, platforms, and digital experiences for the modern world. By integrating strategic design, complex engineering, and vertical industry expertise—we help our clients imagine what's possible and accelerate their transition into tomorrow's digital businesses.

Headquartered in Silicon Valley, GlobalLogic operates design studios and engineering centers around the world, extending our deep expertise to customers in the communications, automotive, healthcare, technology, media and entertainment, manufacturing, and semiconductor industries.

www.globallogic.com