The app-modernization manual

The definitive guide to intelligent apps

Table of contents

Introduction

Companies across every industry are transforming to adapt and succeed in today's fast-paced and increasingly digital world. IT plays a critical role in this environment: helping deliver breakthrough customer experiences, enabling always-on innovation cycles, and responding with unprecedented agility to marketplace changes.

But to enable digital business, you need to transform your approach to the apps that are at the core of your operations. Only then can you respond quickly to customer needs and deliver at the speed of ideation.

Modernizing means transforming existing software, taking an agile approach to development across people, processes, and technology, and embracing cloud-native development, including microservices and containerization, hybrid cloud-integration, and agile and DevOps methodologies. It also means rethinking your approach to application services to transform it into a powerful driver of innovation.

App modernization ensures you have the ability to execute quickly, respond and adapt to continuously evolving market conditions, and pivot to new business models ahead of the competition.

In this document, we'll explore the three key tenets of successful app modernization:

Discovery, which includes how to build the roadmap and business case for modernization by understanding your current state

Transformation, which focuses on how to rearchitect your applications and reap the benefits of digital agility, with core transformation pillars including embracing cloud native, harnessing the power of APIs, shifting to agile, DevOps ways of working, and leveraging POD-as-a-Service (POD) models. We'll explore transformation across three pillars:

- Architecture
- Technology
- Organizational structure.

Operations, which includes a discussion of how to optimize and scale for maximum agility and cost savings following modernization.

Part I: Discover

The fundamentals of building intelligent apps

What is app modernization?

Application modernization is the process of transforming legacy software. Broadly speaking, there are two options for application modernization: rehosting – or moving applications from an on-premises environment to a modern cloud infrastructure – and transforming, either by refactoring, rearchitecting, or replacing with SaaS.

app-modernization manual focuses on rearchitecting, which is the most complex approach to application modernization and, in many instances, the most beneficial. Subsequent chapters will dive into the different approaches and discuss what's required to make re-architecting successful.

It's crucial to remember that application modernization is not just about a technical transformation alone. In order for application modernization to be successful, organizations must consider not just technology but the people and the processes that surround it. In our experience, application modernization efforts tend to fail when not enough attention is paid to updating the organizational structure, processes, and skills alongside architecture and technology updates. Without a cultural transformation, there is no advantage of having a modernized platform. For an application-modernization project to be successful, cultural transformation cannot be an afterthought; rather, it needs to be planned for at the start of any modernization project.

Why app modernization?

All application-modernization options deliver the same fundamental advantages: shrinking the in-house footprint to reduce any associated costs across networking, compute, storage, and backup; reducing costs and making expenditures more predictable; and improving agility, speed, scalability, security, and reliability. These enable a highly scalable and always-on IT ecosystem where organizations can focus on growing their core business and launching new and innovative products instead of focusing on IT issues.

It gets organizations out of the traditional development mindset of long release cycles and siloed functions and to a modern mode of operating where IT is more aligned to the business, automation is a given, and applications can be released much more quickly. With app modernization, organizations can move from slow, multi-week deployments to ones that happen in a matter of days or even hours.

Efficiency: 25–70% reduction in platform support

Quality: <1% production code with defects

Continuity: Zero downtime and a serverless provisioning model

Self-healing: 2–10 minutes average failure recovery

Speed: 25–50x improvement in release velocity

Compatibility: No cloud provider lock-in

Flexibility: Increased configurations and infrastructure as code

TCO optimization: Optimize legacy apps to get more out of existing investments

Setting the strategy for app modernization

Understand your objectives

To get started, it's important to take a look at the entire landscape to understand what should be modernized and what approach is needed. Understand how people and processes need to transform alongside the technology to get the most out of the investments. This can be done by, for example, enabling agile, DevOps ways of working, shifting to product-centric modes of delivery, and embracing cloud-native. It's critical to build in flexibility and take a strategic approach to innovation to respond to changing customer and market needs.

Building the business case and roadmap

The strategy phase is a critical element of any application-modernization project. The value lies in thinking strategically, becoming a trusted advisor to business, and planning across domains. It can help reduce issues like unexpected delays and ensure optimal value is delivered for the business. These are the elements to include when planning a modernization initiative:

> *"Getting to the right approach will include a combination of workshops, interviews, assessments, analyses, and meetings to gain alignment across the organization."*

Start with an assessment. It can be a big task to "see" everything in the IT enterprise but understanding what's in the portfolio goes side by side with a strategy. An assessment will give you an in-depth understanding of the current IT landscape and provide the critical information that will form the basis for a strong business case and an outcomes-oriented roadmap. Gaining visibility into your application portfolio can help identify the buckets of strategic decisions needed, from cloud strategy to consolidation. It should include analyses across multiple dimensions, including application portfolio, infrastructure, costs, cloud and digital readiness, and resource allocation. It helps identify what to eliminate, consolidate, modernize, replace, or remove, as well as where to invest in the future. By aggregating data into an analytics platform, it's possible to group apps across themes to drive stability and velocity – for example, grouping by criticality, problem volume, risk, and cloud suitability.

An important component of a successful assessment is the decision framework. It includes the organizational strategy, vision, and guiding principles to drive the decision-making process and provide parameters for how IT will engage with the business. For example, as an organization shifts towards agile and decides whether to be product-driven or services-driven, a decision framework will lay out the implications of the selected choice.

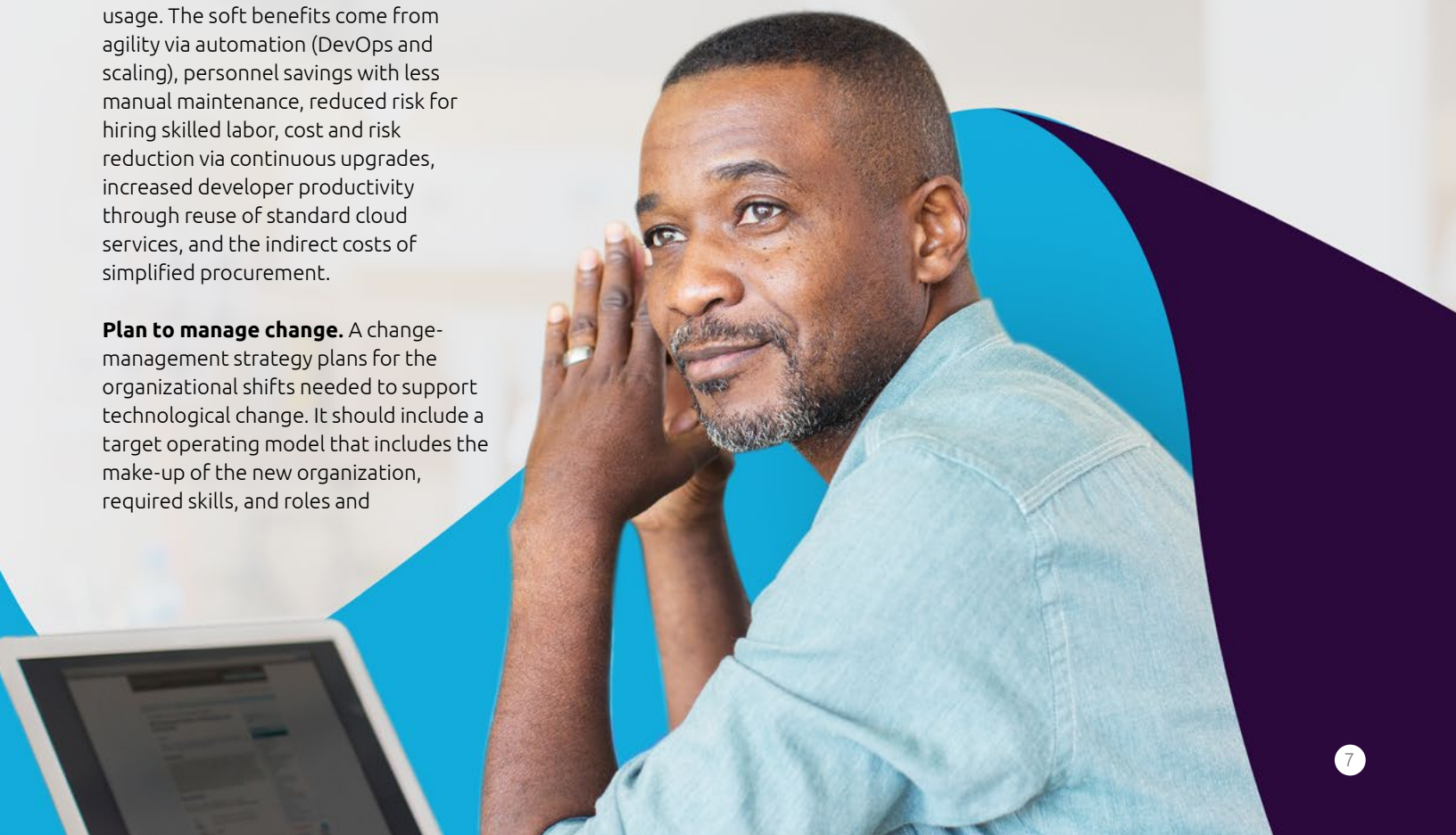
Building the approach. Develop the business case and roadmap. An execution roadmap prioritizes initiatives like application modernization, cloud, automation, and DevOps by considering timelines, costs, and findings of the assessment phase. To build the roadmap, a good place to start is with Gartner's PACE-layered Application Strategy for aligning with the business to categorize the value of applications and the pace of change, as well as on what to invest and tolerate or eliminate and consolidate.
When categorizing applications, organizations need to think about systems of innovation, systems of business differentiation, and systems of record, as that will have an impact on your strategy. For example, for applications that fall into the systems-of-record and tolerate categories, you'll likely need to focus on investing in some incremental transformation to get them cloud ready. For systems of innovation and differentiation that are likely already on the cloud, you'll want to ensure you're transforming people and processes to ensure maximum business agility and responsiveness.

The business case includes the financial rationale behind a transformation, including cost, savings, and returns. It provides a forecast for return on investment and connects IT changes to business KPIs that ensure ongoing project funding. To build the business case, organizations need to first understand the focus and motivations around cost savings. They then need to develop a current state cost model per app to get to an average monthly cost per virtual machine and per database that can be compared with cloud hosting and drive decisions around IT spend alignment to business criticality.

Savings from cloud adoption and modernization include both hard and soft benefits. Hard savings come from the right-sized infrastructure, autoscaling, and elasticity of cloud along with the reduction in software licenses (think VMWare or open-source databases) as well as pay-as-you-go usage. The soft benefits come from agility via automation (DevOps and scaling), personnel savings with less manual maintenance, reduced risk for hiring skilled labor, cost and risk reduction via continuous upgrades, increased developer productivity through reuse of standard cloud services, and the indirect costs of simplified procurement.

Plan to manage change. A change-management strategy plans for the organizational shifts needed to support technological change. It should include a target operating model that includes the make-up of the new organization, required skills, and roles and responsibilities. IT transformation projects aren't just about technology; they're also about people, processes, and culture. IT leaders need to think about the skill sets they need and how existing skills are going to change to support the future. They need a plan for upskilling existing teams as well as recruiting for hard-to-find skills. Because various components are required to create a well-structured plan, the strategy phase is critical to any transformation program. Getting to the right approach will include a combination of workshops, interviews, assessments, analyses, and meetings to gain alignment across the organization. By aligning principles, understanding the IT enterprise from bottom-up, and building an initiative roadmap, the organization is able to manage and track transformation at an intentional program level to drive efficiency.

> *"To get started, it's important to take a look at the entire landscape to understand what should be modernized and what approach is needed."*



Approaches to application modernization

Your approach to application modernization depends on your landscape. It's not a one-size-fits-all situation, and each approach has different outcomes and ROI and varies by different business needs. Here, we'll dive into the possibilities for modernizing your applications.

Rehosting

Although rehosting doesn't technically fall into the category of application modernization, it is the first step many companies take towards modernizing their application portfolios. Rehosting is simply moving an application from its on-premises environment to a modern cloud infrastructure. Companies choose to rehost an application when they are pleased with its performance and functionality but need the benefits of an agile cloud infrastructure. They retain the familiarity and functionality of the original application while reducing their data-center footprint and networking, compute, and storage costs. They also reduce their capital investment and maintain business continuity. It is the most straightforward of the four options and generally delivers significant boosts in performance.

Now, let's shift to true application modernization.

Modernization approach 1: ***Refactoring***

What is it?

Refactoring refers to the process of restructuring code to improve performance without changing its intended function. Refactoring involves customizing an application so it can run on a cloud platform.

Why do it?

Companies will opt to refactor applications when, as in the case of rehosting, they are basically satisfied with the functionality but the code needs to run on a more modern foundation, such as a current operating system. With minimal changes to the underlying structure but no alteration of the software's functionality, these applications can run on modern flexible cloud systems. Refactoring applications reduces technical debt, improves performance and efficiency, cuts costs, and creates code portability where none existed before. As importantly, refactored applications can receive all vendor software updates, ensuring security standards and performance are maintained. Many organizations, for example, have machines running Windows Server 2008. Microsoft support for that operating system is about to expire. With minimal changes to the underlying structure but no alternation of the software's functionality, these applications can run on modern foundations and move to flexible cloud systems. Once refactored, applications can be containerized or leveraged with PaaS services.

How to do it

A partial refactoring modifies only specific portions of the application to enable the migration to the cloud platform, whereas a complete refactoring modifies applications so they can perform at a much higher level of efficiency and optimizes them so they can operate at much lower costs.

Modernization approach 2: ***Rearchitecting***

What is it?

Rearchitecting transforms single-tier architectures into distributed systems in which functions and processes are divided in a modern cloud implementation. When rearchitecting, applications transform from monolithic to microservices-based architectures.

Why do it?

Rearchitecting provides the maximum benefits of cloud transformation given that the new versions of applications are custom designed to align to business requirements and operate with maximum performance and efficiency. This is far more efficient and more reliable than refactoring.

How to do it

Rearchitecting an application involves the complete remodeling of the application across all layers to incorporate the latest technologies and architecture best practices. This may involve decomposing a monolithic application into a number of autonomous microservices or serverless functions and deploying them on private or public cloud platforms.

Modernization approach 3: ***Replacing***

What is it?

There are two choices when it comes to replacing aging, on-premises systems: redeveloping with new code built for modern cloud infrastructures using current development tools or shifting to SaaS. Replacing is the most labor-intensive option of the approaches and may involve some level of business disruption.

Why do it?

The replace option is for companies that are in need of a specific function, such as payroll or inventory, but know that the legacy system is so out of date that starting from scratch in the cloud or moving to SaaS would be the most efficient option. Companies that rebuild have an unprecedented opportunity to create functionality that exactly matches current needs and future requirements.

When choosing between rebuilding yourself or going with a SaaS solution, it makes sense to build functions that are highly proprietary, confidential, or critical to your organization from a competitive standpoint but go with SaaS solutions when there's a best-of-breed option available. For example, organizations likely wouldn't build their own CRM system, given the availability of powerful existing SaaS solutions. On the other hand, telecom providers may build back-end solutions for inventory management or digital billing, given the importance of both to retaining a competitive edge.

How to do it

Replacing an application involves analyzing a suitable SaaS or cloud-native platform that provides enhanced or like-to-like features from a functional perspective and yet offers all the benefits of cloud. A critical aspect of replacement is developing a phase-out strategy, migration plan, and reconciliation design and deployment architecture.

Part II: Transform
Modernizing across technology, processes, and people

With app modernization, an organization must transform across multiple pillars. To achieve maximum value, any application-modernization initiative should include a consideration of architecture, technology, and organizational structure.

Architecture transformation:

App modernization begins with a technical change, with organizations shifting from three-tier, tightly coupled, hard-to-change monoliths to a modern, agile, and loosely coupled microservices-enabled landscape that enables greater flexibility and agility. The journey starts with having the right architectural framework that incorporates industrialized design patterns and implements best practices focused on simplifying application development and maintenance cycles. It includes domain-driven design, the de facto architecture pattern for cloud-native architecture, that breaks up complex business domains and monolithic applications into smaller data-driven microservices and helps define clear boundaries for each context.

It takes an API-first design approach to ensure that the microservices conform to enterprise-wide consumption and comply with security and governance requirements. Asynchronous architecture allows reduced dependency on complex integrations and brings in autonomous behavior that results in parallel builds and faster deployments. Additionally, architecture is designed for network and system failures and leverages fifteen-factor design principles rather than the typical twelve-factor approach to achieve agility, scalability, and operational efficiency.

Technology transformation:

Adopting modern, cloud-native technologies such as containerization, serverless, microservices, and PaaS are key to achieving maximum benefit from app modernization. But the massive scale of modernization requires that the technology foundation go beyond updating functional capabilities to also focus on establishing DevOps and quality automation pipelines that can continuously bring the development and maintenance costs down and help improve operational excellence. Additionally, with app modernization, organizations shift from open-source technologies with limited re-use and on-premises infrastructure to leading-edge solutions on three PaaS patterns: traditional, custom, and public.

Fifteen-factor design

1. One codebase, one application
2. API-first
3. Dependency management
4. Design, build, release, and run
5. Configuration, credentials, and code
6. Logs
7. Disposability
8. Backing services
9. Environment parity
10. Administrative process
11. Port binding
12. Stateless processes
13. Concurrency
14. Telemetry
15. Authentication and authorization

Organizational structure transformation:

To support the modernized architecture, silos across the business, development, quality assurance, and testing need to be broken down and replaced with self-sustaining domain-driven teams. Whereas in traditional development approaches there are separate resources to support development, infrastructure, and quality assurance with large teams of 30 or 60 people per project, in a modern approach, the same resources need to be responsible for development, platform operations, and automation scripting.

In POD models, a single team of cross-skilled people works closely together across all aspects of maintenance and development to maximize speed and agility. If a team owns a feature, it does so in an end-to-end you-build-it-you-own-it model. Each POD is comprised of a team that includes a product director, scrum master, product lead, technical product analyst, full-stack developer, backend developer, SDET, and DevSecOps automation engineers. The POD team works together in an agile, DevOps model as they develop and maintain applications while making enhancements to incorporate emerging technologies and reduce technical debt.

In this chapter, we explore each of these areas in more detail to provide a comprehensive picture of how to successfully transform with application modernization.

Any application-modernization initiative should include a consideration of architecture, technology, and organizational structure.

Embracing a platform approach

One potential challenge that IT teams face when in a cloud-native environment is working within a highly distributed ecosystem with multiple moving parts, such as microservices and containers. If design patterns are not followed or implemented correctly, achieving maximum value can be a challenge. Because of this, we recommend a platform approach for any organization looking to embrace cloud-native development.

Why do you need a platform approach?

A platform-centric approach provides a framework with best practices and proven patterns that have worked not just within a single organization but within others as well. A platform is a base set of capabilities and features that provide the foundation to optimize and innovate within a cloud-native environment. It can encompass technology elements as well as processes. For example, we worked with a global wireless carrier to establish two platforms: one was based on microservices with a standard technology set and the other a DevOps implementation containing standard tools, processes, and technologies. A platform can integrate with partners, suppliers, and consumers and drive your business capabilities.

From a modernization standpoint, a platform provides the infrastructure for transforming from haphazard communication between legacy systems to well-defined reusable services. This standardization allows a company to react to change faster. A platform is horizontal and sits outside any specific line of business, which facilitates standardization that simplifies the process of making technical changes while promoting efficiency and scalability. For example, Capgemini worked with an organization to build a NoSQL datastore when the traditional online database became a bottleneck. The underlying platform capabilities, such as the event bus, were essential for the technical team to make a decision on a solution, pilot it quickly, and launch to production without an impact on operations.

What should a platform consist of?

A platform is essentially a pre-defined set of APIs and services that enable and support applications, data, and business capabilities for the enterprise. It can include operational, orchestration, data, and network components as well as lead up to AI algorithms, DevOps tools, and security services. These pre-defined APIs and services can also be specific to your business and technology, as in the case of core business functionalities that are exposed by legacy wrappers for consumption by the business. A platform also includes a mechanism for standardizing across lines of business, allowing specific teams to request features via self-service provisioning. It also includes a sandbox for trying out new capabilities to be added.

Should you build or buy your platform?

A platform exists to facilitate the rebuilding of the portfolio and should improve processes. When thinking about how to construct a platform, you can opt to build these capabilities in-house or leverage third-party services like Kubernetes or PaaS features from a cloud provider.

Technical leaders should look at platform roadmaps to progressively migrate business operations from existing systems to platform capabilities. For example, re-architecting a legacy monolith as PaaS services requires the use of existing re-usable and production-proven capabilities that maximize efficiency and reusability. This is, in essence, a shift to a platform approach, facilitated by the process of application modernization. In the platform approach, all modernized capabilities are built to be reused internally and externally, enabling the move to API monetization.

It's important to remember that the transformation to a platform-centric approach will not happen overnight, and we don't recommend going for a big-bang migration approach to a new platform. Rather, we recommend first conducting a pilot project with a group that embraces change, and then building from there.

Taking a phased approach to modernization

Modernizing legacy applications can be complex, especially when multiple applications are involved. The most challenging aspect of modernization is the movement of users and consumers of legacy application services to the new modernized stack. This includes bringing in new systems during the interim phase, synchronizing data, providing business coverage with fully serviceable overlapping functionality, and, at times, developing temporary systems and processes for bridging legacy applications to the new system.

Due to the scope and size of a typical modernization initiative, organizations often must take a phased approach to introduce the updated platform. In a multi-phase strategy, organizations synchronize the new platform with the legacy applications until it's possible to switch over to the new platform and microservices. While enterprises are still in the process of modernizing, legacy and modernized services will coexist as organizations transition gradually.

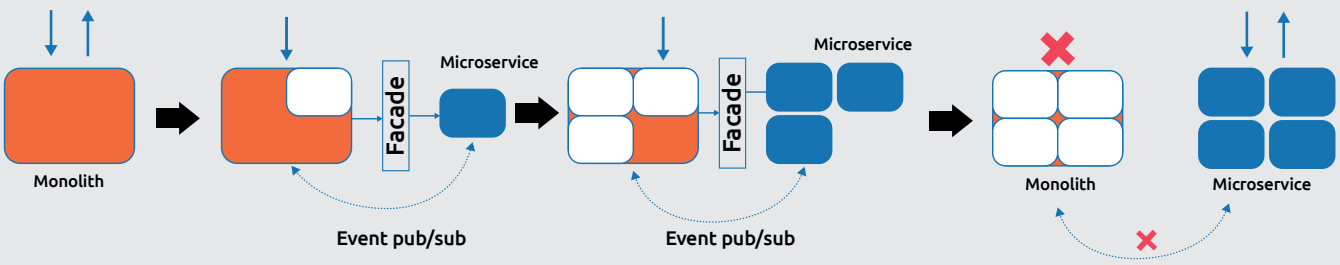
This type of phased approach should use a strangler pattern, which is a way of migrating a legacy application incrementally by replacing existing functionalities with new platform and services in a step-by-step manner. This strategy will require an initial load phase that ensures all operational and configuration data required for the new microservices to fully operate in production are available in the new platform. It also requires an overall state synchronization between the new and old systems to maintain data integrity and data-model dependencies between individual functions.

Initial load

In the initial load phase, a company migrates all available reference data, user information, and required history and transactional data from legacy applications to the new platform. Mapping is important in this phase. This includes mapping entities from the legacy application to the modernized equivalents on the new platform, transforming data from legacy applications to address the domain model of the new platform, and filling in the blanks or extrapolating where there are gaps given that the domain model of the new platform will be a superset of the legacy applications.

State synchronization with legacy systems

A big-bang approach, where you completely replace a complex system with cloud-native applications, can be a huge risk. The strangler pattern reduces the risk of a complete failure. However, running two separate versions of an application means that an organization needs to know where each feature is located. Every time a feature or service is migrated, users need to be updated with the new location. Strangler patterns facilitate this by creating a routing facade. There are two options for doing this: modifying monolithic functionality or using event-choreography patterns.



Phased introduction of modernized services

State synchronization option 1: Modifying monolithic functionality

The first option for the rollout involves routing current transactions on the legacy application to the newly developed microservices. All business logic and southbound communication are switched off in the legacy application as the transaction management responsibility shifts to the new services. This will require selectively switching off the functionality in the old applications while continuing to write database transactions to maintain data consistency for the operations of other functionalities that have not yet been transitioned to the new application. This approach simplifies the overall integration between the legacy system and new microservices platform as it does not require any external process to maintain data synchronization

State synchronization option 2: Utilizing event-choreography patterns

While the first option provides simplified integration of new microservices with the legacy applications, it may also require significant complex modifications to the existing application. To circumvent this complexity, organizations can use event-choreography patterns. In this approach, data is synchronized between the old and new applications as a non-intrusive, event-based architecture, which ensures that data consistency and integrity are maintained during the rollout. Ultimately, functionality in the old application is switched off once all services are transferred to the new system. This option will require the definition of events, publishers, and subscribers as well as a mechanism to exchange events between the old and new applications via a messaging bus.

Both of these options have pros and cons. Determining the right approach requires having a deeper understanding of existing monolithic capabilities. In general, organizations should select the option that is simplest and leads to the least amount of technical debt.

Going cloud-native to get results

Adapted from the Capgemini report "Cloud-native comes of age"

A key objective of app modernization is to shift to cloud-native development, where applications are built directly in the cloud or on a PaaS platform rather than on-premises physical infrastructures. The shift to cloud-native development can help organizations realize significant gains in agility and quality. There are a few important things to keep in mind.

Assess the application portfolio and identify priorities for cloud-native development.

Because the cloud-native approach demands significant upfront investment in platforms, people, and skills, it is rarely the best way to achieve short-term, bottom-line cost reduction. CIOs need to evaluate which of their existing applications will benefit most from being rewritten as cloud native, and also which business initiatives and strategic priorities justify the investment of creating net-new cloud-native applications. Cloud native brings the greatest benefits when building new applications or services that drive competitive differentiation and top-line revenue growth. These will often be web, mobile, IoT, or big-data apps.

Start small, and then scale up to develop a skilled team.

Teaching thousands of developers new cloud-native skills at once introduces too much change and risk. A more feasible starting point is a single program involving one small team in a contained area. This allows the value of these new methods to be proven in a relatively low-risk manner. CIOs should select members who are change agents and future leaders to drive these early programs. Skills learned from these pilot projects can then be fed into further initiatives on a more ambitious scale. This delivers a gradual, sustainable increase in the in-house skills base.

Adapt the IT operating model to support both business agility and stability.

DevOps is the essential enabler of cloud-native development. DevOps is both a cultural shift and a technology movement. The cultural changes that DevOps brings include the removal of barriers between organizational units to enable collaborative discussions within teams. DevOps is underpinned by technology: specifically, the combination of agile tools, automated testing, and the automated provisioning of infrastructure and middleware, typically using a Cloud Management Platform (CMP) and Application Release Automation (ARA). The gradual shift from waterfall development to DevOps does not remove the need to document and test in a repeatable process with strong governance. Key safeguards must remain in place without slowing the process.

Be pragmatic in selecting technologies; lock-in risks and integration challenges are not insurmountable.

If an enterprise has the appetite to build engineering expertise on the scale of Netflix or Google and needs to use special features not available in an off-the-shelf PaaS offering, it should consider building a custom PaaS with open-source components. Otherwise, we recommend an off-the-shelf PaaS, either already pre-packaged (such as Cloud Foundry or a public-cloud PaaS) or a combination of containers and a container orchestrator (Containers-as-a-Service). Complete portability may not always be practical or justifiable in cost terms, but containerization and open source offer the flexibility to build applications in a hybrid model where they exist seamlessly in different environments.

By taking a thoughtful, comprehensive approach to cloud-native transformation, organizations can achieve velocity and flexibility that is simply impossible in monolithic systems.

Microservices: the foundation of cloud-native development

With microservices, you can make an update to just one element of an application, which accelerates release, updates, and enhancements. To get maximum value and business agility from microservices, there are a few important factors to keep in mind.

Build for the business

In order to provide the agility and value of cloud native, microservices design should focus on single business domains. Using an API-first design approach will ensure that the microservices conform to enterprise-wide consumption and comply with security and governance requirements. We recommend using fifteen-factor design principles. Domain-driven design, the de facto architecture pattern for microservices, breaks up complex business domains into smaller data-driven microservices and helps define clear boundaries for each context. Microservices should be as asynchronous as the individual business processes allow to reduce dependency on complex integrations.

Design for resilience

Microservices must be designed for network and system failures, such as delays, errors, or unavailability of another service or third-party system. Keeping communications asynchronous allows for resiliency when some services fail. Microservices should also provide a default functionality in case of failures from a downstream service. This could be an error message, or, if the business case permits, a default value that is acceptable until the external service is available. Even if microservices are built for a UI screen to consume, they must be responsible for all data input validation. There are common frameworks to do this using expression language and annotations, rather than code.

Ensure observability

Centralized logging and monitoring are a must for distributed microservices. This enables the quick, manageable tracking of failures when multiple APIs and services are involved. When services are built along with tracing and alerting mechanisms, the mean time to resolve the error is minimized. Log events for timeouts and shutdowns should include the level, hostname (instance name), and message. Log events can also be used for capacity planning and scaling, for example, indicating which services need higher instances. There are multiple frameworks for centralized logging which developers can use in a plug-and-play manner.

Enable automation

Testing tools should be used for the integration of services. Start with unit testing but don't overlook the importance of contract testing to ensure that APIs are functioning as per the agreed request and response. You should also use automation testing in the build pipeline to provide instant feedback on check-ins and failures. With these checkpoints in place, you can be sure that untested code doesn't make it to the development environment.

CASE STUDY

Building a foundation for digital manufacturing in the cloud

The challenge

A leading aerospace manufacturer was facing legacy-system challenges and wanted to modernize and improve predictive maintenance of its supply chain for better after-market service-parts management. The first step was to adopt a cloud-first approach to improve lead time, forecasting accuracy, inventory optimization, and, above all, planning.

Our approach

We worked with the organization to modernize its landscape. To enable portability, improve resource efficiency, and better utilize technology, we used containers as the primary deployment format for all cloud-native applications involved in the migration, with Capgemini managing the clustering, networking, and deployment automation in addition to creating an open-source container-orchestration system. We also worked to implement a plan for agile transformation for greater automation, discipline, and flexibility. We also took an incremental approach to DevOps with a strategy based on three-, six-, and 12-month plans.

Results

The process of app modernization built the foundation for change by freeing the organization from legacy platforms and creating the right cloud environment. In this way, the company was able to adopt AI/ML to improve its predictive asset maintenance for faster feedback and greater flexibility and scalability in response to change. Server provisioning now takes hours instead of days, and if the user community wants new predictive asset maintenance features like demand aggregation, teams can now develop and deploy them quickly. Additionally, the inclusion of security checks and reviews means that the organization has a better software pipeline.

Rewrite once, run anywhere: Choosing the right containerization approach

Key outcomes of cloud-native architecture, enabled by application modernization, include greater scalability, elasticity, and cost savings as compared to the traditional on-premises architecture. Indeed, the rapid uptake of cloud and microservices architectures is helping enterprises innovate and grow their businesses faster than ever. However, the modernization of the IT landscape can pose considerable challenges for IT organizations, from dealing with relatively new requirements associated with continuous delivery to managing distributed and granular application components and hybrid infrastructure across the build, deploy, and operate lifecycle.

Container platforms are helping companies address these challenges by enabling a vendor-agnostic ecosystem that operates seamlessly across hybrid and multi-cloud scenarios. Containers package application components and their dependencies so they can be easily version controlled and seamlessly ported and replicated across different data-center and cloud platforms. The process of containerization involves encapsulating an application and all of its dependencies so that it can run uniformly and consistently on any infrastructure. Containerization allows applications to be written once and run anywhere. This eases the development, testing, deployment, and overall management of applications.

Container platforms respond to modernization challenges

Although the benefits of containerization are clear, many enterprises struggle with how to containerize their workloads, how to accelerate and automate the migration to Kubernetes, and, finally, how to manage the overall run-time environment and operations including applications, containers, clusters, and PODS spread across on-premises and multi-cloud ecosystem. Choosing the right container orchestration layer for applications can be a challenge. Should you manage it yourself or rely on a solution offered by a PaaS provider or cloud-service provider?

Run-your-own Kubernetes versus managed Kubernetes

The Kubernetes open-source container- and process-orchestration system can deploy applications running either in containers or as processes. It is the standard container-orchestration tool and is available for enterprises to manage on their own or as a hosted service from cloud-service providers like AWS or Microsoft, or by managed Kubernetes platforms from companies such as RedHat or VMWare. Leveraging Kubernetes means processes can be handled automatically, allowing organizations to focus on functional applications without worrying about infrastructure components.

Many organizations decide to run orchestration on their own. This means manually handling processes, like lifecycle management, which includes storage, complex clustering, networking of nodes and PODs, scaling, load balancing, and scheduling, and that requires considerable time and investment.

In fact, except when an organization requires a high degree of customization because of versioning at the master-node level, managed container orchestration services have significant advantages over running your own Kubernetes, including speed, ease of operations, and costs.

Benefits of managed Kubernetes

With managed Kubernetes, cloud providers handle provisioning, security, load balancing, upgrading, and monitoring. Managed platforms also provide different application-deployment and orchestration scenarios, including cloud-native development, containerization with orchestration, simplified deployment and management of microservices, and a well-defined DevSecOps pipeline using different tool chains.

Parameters	Run-your-own Kubernetes	Managed Kubernetes with cloud-service providers
Ease of operations	Low fit	High fit
Speed	Low fit	High fit
Integration with providers	Low fit	High fit
Customization	High fit	Mid fit
DevOps, microservices	Mid fit	High fit
Multi-cloud hosting support	Mid fit	Mid fit
Cost structure	Mid fit	High fit

Low fit Mid fit High fit

Why embrace serverless?

With a modernized IT landscape, organizations have started thinking beyond virtual machines. Serverless technologies, in which application infrastructure is provisioned by the cloud platform, are taking center stage. Because it allows companies to shift away from the 24/7 run of servers and pay only for the compute cost of specific business functions through fully modularized applications, serverless is an increasingly popular option. But despite the value serverless can provide, adoption maturity is still in its early stages. Because of this, there are a number of myths about serverless circulating in the market, and it's important to separate fact from fiction and explore the real value of serverless.

Here are some of the misconceptions and reasons why serverless is increasingly the right architecture option for businesses:

Misconception #1:

Performance. Many think that serverless functions have longer warm-up and API response times, but this is a misperception related to the belief that one size fits all. In order to see benefits like cost reduction and faster execution speeds, you must configure the right memory size for the level of demand. This depends on function complexity related to the particular business process.

Misconception #2:

Expense. Serverless is only more expensive when less-than-optimal services or domains are run in a serverless environment. When executed properly and in the right domain, serverless costs less than traditional models thanks to pay-per-use and the elimination of the investment associated with infrastructure management and sysops. Business domains requiring higher levels of scalability and elasticity are ideal options for serverless, whereas when usage patterns are relatively flat there may not be any significant cost savings.

Misconception #3:

Vendor lock-in. Every cloud provider has its own version of serverless architecture and the concept is the same across all. This means platform-agnostic frameworks can be used to create a wrapper around serverless services and make the code independent of any specific vendor.

Misconception #4:

Security. Although the systems are shared, they are highly secured and PCI compliant. In fact, serverless is no less secure than other cloud-based environments.

Misconception #5:

Management. Serverless functions are widely supported by many coding languages (Java, Python, Node.js, .Net), popular CI/CD tools for deployment (Jenkins, VSTS, CodePipeline), and industrialized tools for monitoring (New Relic, Kibana, Digital Elk), making it not only easy but also feasible to manage and deploy enterprise-level serverless-based applications.

As serverless has proliferated, so have the troubleshooting tools that give companies the ability to manage their serverless architectures autonomously. For example, monitoring tools like New Relic and Kibana are easily integrated with serverless functions for application log management and monitoring. Additionally, some cloud providers enable remote function debugging.

Though serverless is not the answer to every business challenge, when used as intended it can provide significant advantages over widely used on-demand cloud services.

CASE STUDY

Improving crops with cloud-powered DNA analysis at Corteva

The challenge

Corteva Agriscience, once the agricultural division of DowDuPont, produces as much DNA sequence data every six hours as existed in the entire public sphere in 2008. The company wanted to shift away from on-premises processing and storage to ensure it could continue to scale to meet business demand.

Our approach

We worked with Corteva to replatform its existing Hadoop-based genome processing systems using a serverless, cloud-native architecture.

Results

Corteva is now able to rapidly scale and significantly reduce cost from its previous on-premises infrastructure. In addition, it gained tremendous speed and efficiency. By leveraging AWS cloud-native technology, genome processing has been reduced from 30 days to just one day.

How to become an integrator and join the API economy

Adapted from the Capgemini report *Unlocking the hybrid integration dividend*

Application modernization relies on whether the organization can access data at great speed and breadth. Modernized hybrid integration and microservices-based APIs are the means to making that happen.

APIs are key to unlocking the value of a modernized landscape and taking full advantage of cloud and other emerging technologies. Modern integration tools connect all these applications and their functionalities. There are different approaches to hybrid integration. Here are some of the possibilities and transformation action plans for each.

Scenario 1: API-first strategy

There are several different reasons for and approaches to adopting an API-first strategy. Some organizations are creating a portfolio of APIs from a consumption standpoint, such as for mobile apps and self-service portals. Others are integrating cloud-based and cloud-native apps, along with existing legacy enterprise apps for seamless connectivity between back-end and front-end systems.

Transformation action plan:

1. Identify and define your organization's goals. For example, incremental revenue streams, opening new business channels, and improving customer or employee experience and satisfaction levels.
2. Leverage a comprehensive API assessment and adoption framework to assess how an API-led integration approach can address integration flows, unlocking of siloed data and services, and reusability, security, and governance requirements.

Scenario 2: Legacy modernization

To modernize legacy IT estates, many organizations are refactoring their legacy monolithic applications using a microservices and API-based architecture. Others are adapting legacy systems to support modern business and technology needs by implementing an integration layer with services and APIs to intermediate their legacy systems and various consumption channels. This allows legacy systems and applications to coexist with modern technologies, such as cloud-native apps, SaaS, IoT, and mobile.

Transformation action plan:

1. Identify the monolithic legacy systems that form the backbone of your organization – applications that are technologically incompatible with newer ones.
2. Create a business case for modernizing your legacy IT estate.
3. Leverage available capabilities to move your legacy transformation forward at pace, including a modernization framework, core integration framework, and a robust architectural approach.

Scenario 3: Legacy middleware/ ESB modernization

A mature hybrid integration strategy is essential to achieve sustainable business growth. Organizations are looking at born-in-the-cloud iPaaS or hybrid integration platforms to support their digital-transformation initiatives and to cut the high operational costs associated with legacy middleware platforms.

Transformation action plan:

1. Make a sound business case for modernizing your integration platform
2. Use a comprehensive cloud-integration assessment framework to clarify the advantages of adopting a modern, iPaaS-based integration platform.

Whatever journey you take, one thing's for certain: leveraging API-led integration is key to succeeding in the digital era.

CASE STUDY

Reinventing a wireless carrier with a platform for the future

The challenge

Focused on breaking industry dogma with amazing customer service, this carrier sought to deliver products to the market at fantastic speed through a digital company strategy and organization.

Our approach

Capgemini delivered an end-to-end digital transformation, including strategy, agile implementation, managed services, leveraging microservices, Apigee Management, continuous integration, delivery, testing, platform, and infrastructure management. We reconstructed the legacy billing systems using microservices running on a Pivotal Cloud Foundry platform. The company now has an agile digital architecture and platform for the billing system of record that keeps up with its dynamic business environment.

Results

The business team now uses web, retail, and care channels to flexibly and rapidly adapt and scale services, with continuously improving customer experience. The cloud-native architecture reduced manual steps by 50 percent and reduced end-to-end billing domain delivery timeframes by 25 percent. Now ideation to execution takes days, not months.

Shifting from project-based to product-based development

Application modernization fundamentally changes an organization's approach to application development, enabling unprecedented levels of business agility and creating the need for a shift to a DevOps way of working.

With the increasing uptake of DevOps, the traditional waterfall model has been turned completely on its head, and what used to be project-driven development is now product-driven development.

A single team of cross-skilled people – a product team – works closely together across all aspects of development to ensure optimal outcomes, from provisioning to running the SCRUM team to product development, CI/CD pipeline, and product release. If a team owns a feature, it owns it end-to-end, so there is no longer a single team responsible for testing and nothing else. It's a you-build-it-you-own-it model in which everyone shares responsibility to maximize speed and agility.

Companies which have thus modernized applications then need to also transform people and processes to enable the shift from project-based development to product-driven development. There are a few requirements.

Ensure it's a joint initiative between business and IT.

The reason for shifting to a DevOps way of working is to align business and IT teams and create a continuous feedback loop from both sides that allows for greater agility. Because of this, ensuring a DevOps initiative is fully owned by both the business and IT is critical to success. But achieving this can be a challenge given that development has historically been an IT-led and IT-specific endeavor. To be successful, the business needs to be a core part of the DevOps team. A POD-as-a-Service structure should be composed of cross-functional and multi-disciplinary teams where the business (the product owner) provides continuous feedback, governance, and guard-rails to drive technology evolution, and IT brings in the right toolsets and controls to drive feature richness. But it's not just about having new processes. This alignment between business and IT also represents a major cultural shift that leaders need to address.

Make operations an integral part of the process.

One of the most important components of the continuous feedback loop inherent to a product-driven development approach is ensuring operations is woven into the overall DevOps process. Traditionally, operations has been an afterthought and a separate phase that was overshadowed by other phases of development and testing. In a product approach, the development team and operations team are one and the same.

In the POD- based model, developers have to think more holistically about factors like the automation of potential error or defect scenarios from the beginning of the development process, because they are ultimately responsible for fixing defects when they arise. If they don't consider them at the start of the process, development will not be optimized or efficient. Because this emphasis on operations calls for a significantly different set of skills than a traditional developer would have, a strong training and recruitment plan is needed.

Ensure you have the right tooling.

We often see organizations struggling to fully realize the benefits of DevOps because the tools they use haven't shifted alongside the significant process changes made. For example, while Excel and Jenkins are perfectly appropriate in traditional development models, Jira and Slack are more appropriate for the DevOps way of working.

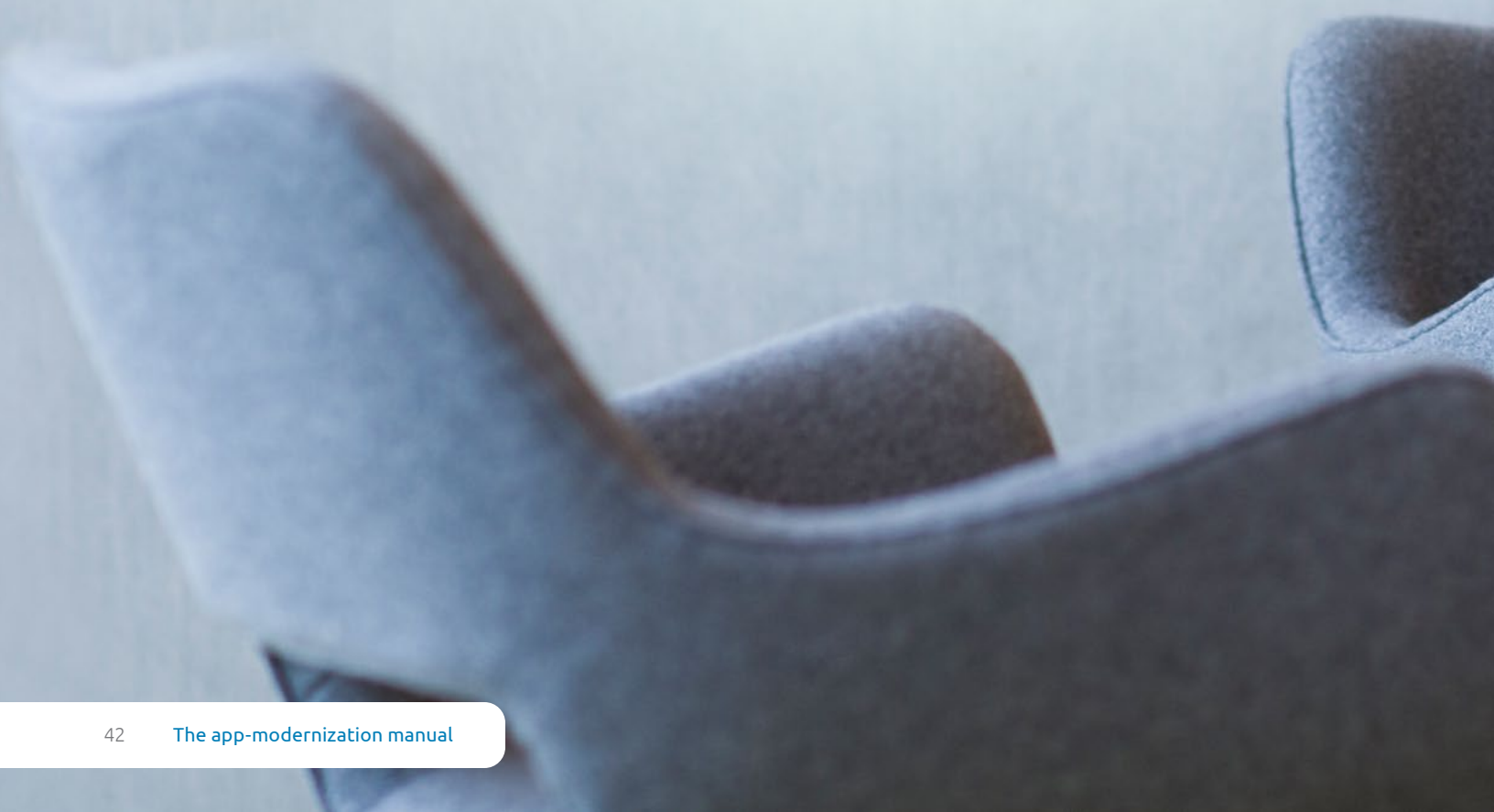
Also, in an optimized POD-based shared-responsibility model, it is important to automate the maximum number of tasks that require multiple teams to collaborate for a single outcome. Test automations should include not just functional tests but also security, infrastructure, and network tests; otherwise, teams will find themselves falling into the traditional model where multiple teams perform siloed tasks. Organizations need to rapidly shift from DevOps to DevSecOps, where security is no longer thought of as an external component or team.

Remember that velocity takes time.

Product visioning and grooming and release plans take time for the team to achieve the level of collaboration and efficacy required to see results. We often encounter teams with unrealistic expectations about how soon they will see gains in agility and speed; in our experience, it takes at least four to five sprints along with a robust change-management and governance plan. Being realistic is important for managing expectations, and while it may take time to see high velocity, remember that once you do, it will continuously increase at a much faster pace than in the traditional model.

Part III: Operate
Driving value from a modernized environment

Once you've updated your architecture, technology, and organizational structure and are working within a modernized environment, the next step is optimizing operations. Although modernized landscapes can deliver business value much faster, they also require management of a highly distributed ecosystem from application features down to containers, platform, and infrastructure layers. Because of this, managing modernized environments using traditional application-development and support models does not provide the level of scalability that is required by modernized applications. In this section, we'll discuss how to optimize operations by leveraging managed services and POD-as-a-Service (POD) models while also scaling with the right blend of design-time and run-time controls.



Leveraging cloud-native managed services for long-term benefits

Traditionally, managed-services providers have focused on running and operating on-premises infrastructures and, more recently, handling IaaS for cloud providers like AWS and Microsoft Azure. By leaving management of their IT ecosystem to expert partners that have delivered similar projects at scale, organization are leveraging best practices and templates to get the most out of cloud and enable greater efficiency, agility, and innovation. But because of the traditional separation between infrastructure and application groups, these teams have operated in a siloed fashion.

But in the cloud-native world, where everything – even infra – is delivered as code, DevOps is the default for application delivery and IT and business KPIs are one and the same, because the traditional barrier between application and infrastructure teams needs to be broken down. Infrastructure and application-development teams need to work together, and managed services must encompass both.

A cloud-native ecosystem is generally complex and highly distributed, with a sophisticated quilt of interwoven technologies and tools that most organizations – and many legacy managed-services providers – are not equipped to run and maintain. A cloud-native managed-service provider will draw on extensive experience and expertise to support the end-to-end lifecycle management of this ecosystem, which includes code pipeline management, container registries, container platform maintenance, and orchestration. In this complex landscape, managed services provide a high-degree of observability, automations that can detect and self-heal, and site reliability engineering for high availability and performance, all of which are critical in a cloud-native environment.

Managing environments using traditional application-development and support models does not provide the level of scalability that is required by modernized applications.

Cloud-native managed services done right

Cloud-native managed services ensure you realize the full potential of developing and running in the cloud. When done right, cloud-native managed services deliver the following core benefits.

The right mix of opacity and transparency: A good managed-services provider will hide complexity for developers, ensuring that they are able to deploy to PaaS platforms without needing to think through the details of the running environment. While development should be a simplified, "black-box" process during the run phase, managed services should enable a high degree of visibility. A managed services provider will build observability into the front-end of the code at both the infrastructure and applications levels with clear dashboards for monitoring and automations that enable self-healing based on KPIs.

Seamless management: Cloud providers have emerged as essential extensions to enterprise IT infrastructure, providing elasticity on demand and unparalleled presence across geographies. As organizations take advantage of a range of cloud-native services from a variety of cloud providers, it's important that they're able to leverage multiple clouds at the same time. Managed-services providers can operate these disparate environments to ensure that functionality remains seamless from one cloud to another.

Security at scale and by design: Though some aspects remain relatively similar, the security and governance required for cloud-native workloads generally differ quite a bit from what is required in a traditional environment. Given the dynamic nature of cloud-native workloads, security needs to be delivered at scale with built-in automation. A managed-services provider will ensure that security is tightly woven into platforms at the architecture level, from service mesh and secret management to fine-grained logging and encryption.

Availability via application-level logic: In traditional IT environments, availability is deployed at the network level via highly scripted load balancers and global DNS solutions, while in a cloud-native environment workloads are configured with service mesh technology that auto-discovers microservices and automatically reroutes traffic. An experienced service provider will be able to configure and maintain a service mesh.

In short, when implemented correctly, cloud-native managed services help organizations reap the full benefits of developing and running in the cloud: simplicity, observability, scalability, and automation. They allow IT teams to focus on driving business value instead of managing the details of their cloud-native environment. For example, with Capgemini teams managing its environment, a large furniture retailer leveraged DevOps and cloud native to completely transform the e-commerce experience. Additionally, we worked with a hospitality company to manage the DevOps CI/CD frameworks and pipelines across vendors, ultimately helping the organization realize a 10 percent increase in sales, a 20 percent reduction in operational costs, and a 10 percent reduction in development costs.

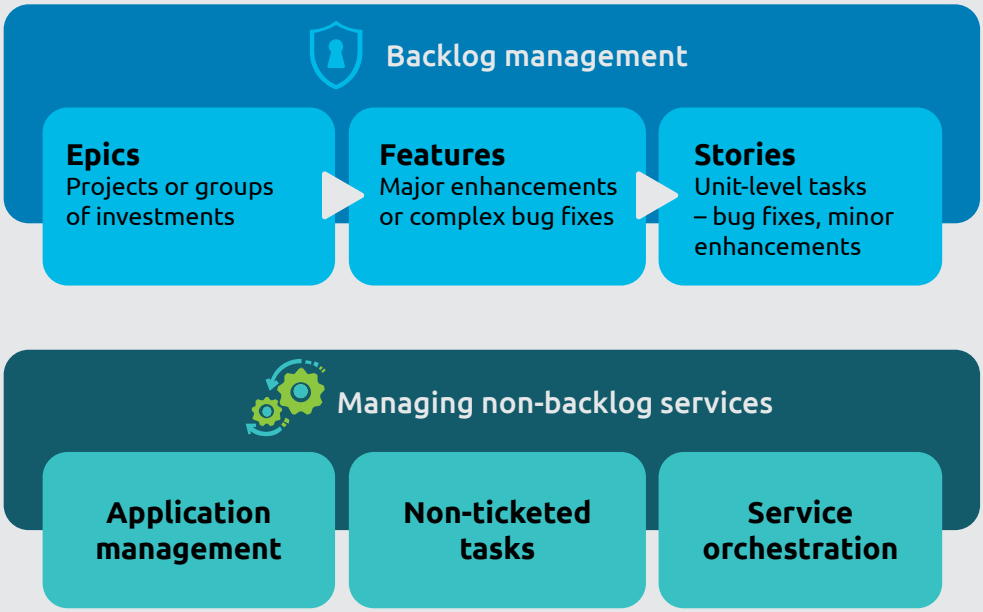
The potential of cloud-native is clear. With the right managed services in place, organizations can ensure they're taking full advantage.

Embracing the POD model for ADM

In a traditional model, an IT organization can meet all of its service-level agreements and KPIs while still not meeting business needs. The product-centric model is more outcomes-focused, with KPIs tied to business results.

One of the main benefits of PODs is their ability to drive efficiency, and, as a result, savings. This is thanks to DevOps modes of development and automation, the elimination of time-consuming handoffs between AD and AM teams, and the presence of a designated product owner who manages backlogs and prioritizes critical work. Because of this, with POD teams, it's possible to achieve savings of 30 to 35 percent. Additionally, it cuts the costs associated with shadow IT, a problem many organizations face when IT isn't able to deliver the level of speed and agility required.

But savings and efficiency represent only a fraction of the benefit of the POD model. If a company looks to a POD model to simply save money, it is not going to be successful. The true value of a POD lies in its ability to enable IT to move at the pace of business today. Instead of it taking three months for the development team to get a new feature or product to market, it takes three weeks with the POD model.



Scaling beyond modernization

Following modernization, organizations may face challenges achieving maximum agility from their applications for many reasons. For example, the scalability and elasticity of the application portfolio may remain dependent on IaaS and PaaS layers. Additionally, the application layer may still operate in a traditional static manner, meaning that DevOps teams face the same challenges experienced in a monolithic architecture.

To truly achieve agility at scale with application modernization, enterprises need to think beyond the adoption of microservices, containerization, and the DevOps approach in the development process. Accelerated development velocity is of no relevance if there are challenges further along in the workstream. For example, if there is a bottleneck in operations, improvements in time-to-market from having an updated architecture will never yield the anticipated benefits.

The path beyond adoption of modern architectures includes establishing the right blend of intelligent design-time and run-time controls to continuously monitor, govern, predict, and self-heal a cloud-native ecosystem. Design-time controls focus on preventing failures by injecting design patterns on service discovery, event sourcing, rules management, and data-model designs. Run-time controls help detect, predict, and auto-resolve failures by implementing mechanisms such as continuous logging, monitoring, and proactively self-healing for faulty or low-performing microservices. This means operations has the same level of elasticity and scalability as the application layer.

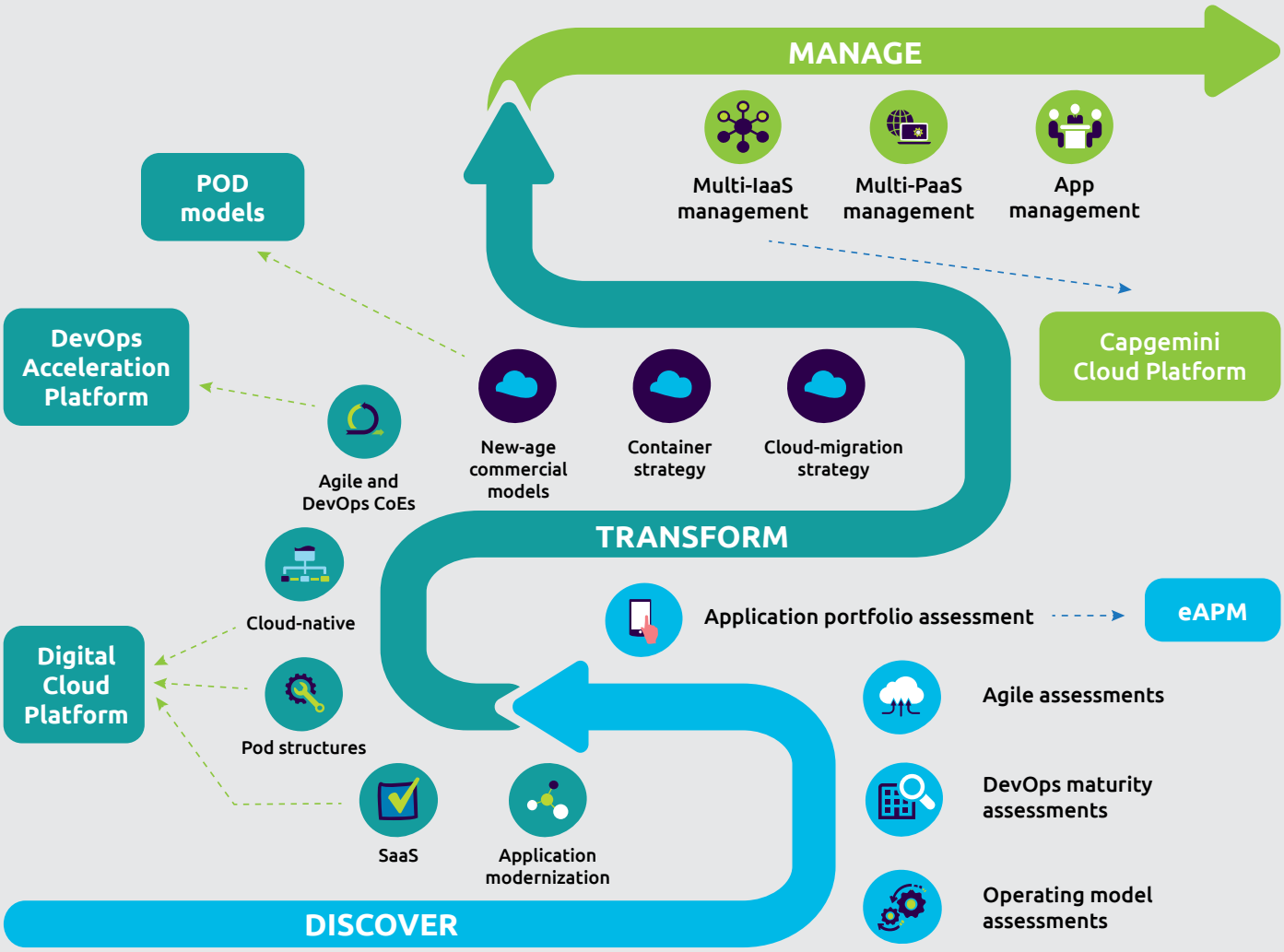
A DevSecOps toolchain includes an end-to-end run-time controls approach with intelligent logging, monitoring, and auto-healing solutions that can proactively detect or predict application errors and security vulnerabilities, correlate and perform root-cause-analyses across app code, container, PaaS, and IaaS layers using policy and AI/Ml-based models, and trigger auto-resolution/business transaction reconciliation using purpose-built RPA and orchestration solutions.

In conclusion

Starting on your application-modernization journey, it's important to remember that every organization's trajectory is different, guided by different objectives and business needs. This is why it's so important to start with an assessment. It's also why, when building out your strategy, a one-size-fits all approach won't do it. Certain applications will be simply moved to the cloud while others will need to be entirely rearchitected. Specific tools and processes will require updates, but you'll likely need to take a phased approach before you reap the full benefits. Finally, once you've modernized, your journey still won't stop there – you need to scale and further optimize costs to benefit over the long term.

Capgemini brings deep expertise and experience honed over thousands of implementations to every application-modernization project. We've worked with organizations of all types to help them enable unprecedented levels of agility with a comprehensive approach that includes the following pillars:

- **Discovering the right approach with eAPM**, a best-in-class portfolio-management analysis tool with a graphical visualization layer to help build the business case and roadmap for transformation
- **Enabling the adoption of microservices-based architectures with Digital Cloud Platform**, which includes industry blueprints, transformation roadmaps, and a library of pre-built software components to help accelerate cloud-native development by 30 to 50 percent
- **Enabling an end-to-end DevOps pipeline** with our DevOps Acceleration Platform, Capgemini's web-based platform that brings together all critical DevSecOps tooling in one simple, easy-to-use interface with built-in compliance, monitoring, and reporting
- **Migrating workloads to containers** with replatforming-in-a-box that makes it easy to begin the cloud-native journey quickly and simply
- **Shifting to product-centric delivery and leaner operations** with POD-as-a-Service models that bring together AD and AM in one business-outcomes focused team for unprecedented agility
- **Managing modernized workloads in a standard and consistent manner** across multi- and hybrid-cloud ecosystems using the Capgemini Cloud Platform and a multi-cluster managed services model.



We are here to assist with any part of your modernization journey and look forward to joining you on the path to embracing Monday's idea as Friday's reality.

About Capgemini

Capgemini is a global leader in consulting, digital transformation, technology and engineering services. The Group is at the forefront of innovation to address the entire breadth of clients' opportunities in the evolving world of cloud, digital and platforms. Building on its strong 50-year+ heritage and deep industry-specific expertise, Capgemini enables organizations to realize their business ambitions through an array of services from strategy to operations. Capgemini is driven by the conviction that the business value of technology comes from and through people. Today, it is a multicultural company of 270,000 team members in almost 50 countries. With Altran, the Group reported 2019 combined revenues of $18.5 billion.

Learn more about us at

www.capgemini.com

Note: current conversion is €1 to $1.18 (8/15/20)

For more information, please contact:

Rishi Kulkarni
Enterprise Architect Director
Cloud Native Practice Lead
rishi.kulkarni@capgemini.com

Kaushik De
Principal, NA Application and Cloud
Technologies Go-to-Market Lead
kaushik.de@capgemini.com

You may also visit:

https://www.capgemini.com/us-en/service/cloud-services-2