

Modern App Development and Enterprise DevOps series

Start in control and stay in control

Five Cloud Native adoption principles for enterprises



Table of Contents

Introduction	4
The innovation-ready enterprise	5
Achieving innovation with the right assets and mindset.....	6
Benefits of cloud native adoption.....	7
Principle 1: More business innovation with less IT	9
Modular business processes in a platform organization	10
Product-oriented development teams.....	11
Get compliance and security to onboard and understand cloud native	12
Business outcomes with objectives and key results (OKRs).....	14
The business case calculator: Evaluating the current application landscape	16
Readiness assessments	21
OKRs for Principle 1: More business innovation with less IT	23
Principle 2: Control with policy-governed practices.....	24
Define policies for all platforms.....	25
Automate policies.....	26
Make proactive and reactive policies	26
Policy compliance monitoring.....	29
Just-in-Time learning.....	29
OKRs for Principle 2: Control with policy-governed practices.....	32
Principle 3: Enable the organization with a cloud native foundation	33
Bring the Cloud Center of Enablement (CCoE) to teams	33
Set-up an Open Source Program Office	34
Landing zone(s).....	37
Hub and spoke pattern.....	40
Azure Developer CLI.....	41
Cloud Native Application Protection Platforms (CNAPP)	43
Principle 3 OKRs	45

Principle 4: Consistent, secure, and compliant from start to finish	46
Infrastructure as Code: Consistent and compliant	46
Context-based and secure cloud native building blocks	48
The Hero Pipeline	49
Scaling numerous teams and microservices	50
Responding to threats and vulnerabilities	54
Developer portals and scorecards	54
InnerSource	56
OKRs for Principle 4: Consistent, secure, and compliant from start to finish	58
Principle 5: Be adaptable and scalable	59
Agile Architecture: Just-in-Time and Just Enough Architecture	60
Beware of technical debt with the Single Responsibility Principle	62
Keep microservices adaptable	63
Leverage encapsulation to the max to stay platform agnostic	64
Build business-focused serverless functions and don't forget the defaults	65
The ability to change is closely related to the ability to recover	66
Designing scalable and containerized solutions	68
OKRs for Principle 5: Be adaptable and scalable	70
Sogeti Cloud Native adoption activities	71
How Microsoft and Sogeti can help	74
Resources	74
Authors	75
Contributors	75

Introduction

Modern businesses need to forget the idea of maturity models, which lead to a false sense of accomplishment and thereby lead to loss of vision and innovation. Transformation continues even after levels defined in maturity models have been achieved. The focus should instead be on the current and future capabilities that the business requires in order to stay competitive in an ever-changing marketplace. The capabilities should be based on measurable business targets and not merely technical competency.

Organizations realize new levels of innovation when they leverage the capabilities of cloud as a first choice for their business workloads. Cloud native applications enable the elasticity, performance, and security that modern enterprises demand by taking full advantage of containerized technology, serverless and low-code platforms implemented with either micro-service based or event-driven architectures. Rethinking team boundaries enable organizations to adapt to new technology shifts like the convergence between applications and data strategies. Data are now exposed on-the-fly with serverless while applications are built on data models. Generative AI capabilities are infused into the applications improving the natural way we interact with users and data while the software development lifecycle is dramatically augmented by AI-assisted programming.

While the benefits that cloud native technologies bring to the enterprise are enticing, there are common pitfalls, blockers, and concerns to overcome to ensure these benefits don't stall after the first application.

Security concerns will be an innovation blocker without alignment. Low visibility and fast-moving workloads can increase any potential or perceived threat. The modularity and independence of development teams together with the lack of skills can cause misconfigurations and insecure systems.

Speed of business innovation, agility, and efficiency enterprises achieve with cloud native adoption does not come for free. Developers need to learn how to adopt cloud native architecture principles. This means adopting a transformational approach towards a product-oriented and business value driven organization, where every member understands and leverages the full potential of cloud native technologies.

Automation and Generative AI continually impacts software development augmenting product owners, architects, developers, testers, reliability engineers in their activities of design, development, deployment and run. This enables new modern applications and the refactoring of existing applications with new AI-enabled features. The balance between innovation and control becomes even more one of the critical success factors for your organization and products.

This eBook sets out five key principles for successful cloud native adoption that address these issues and help enterprises leverage the full potential that cloud native technologies present:

1. More business innovation with less IT.
2. Control with policy governed practices.
3. Enable the organization with a cloud-native foundation.
4. Consistent, secure and compliant from start to finish.
5. Be adaptable and scalable.

The innovation-ready enterprise

Today, organizations face a volatile, uncertain, complex landscape where the predictive models of the past – budgets, forecasts, long-term strategies and information architecture – can become irrelevant overnight. One crisis follows another, demanding an immediate response and – often – a fundamental change in approach. Teams that are unable to respond rapidly and decisively miss valuable opportunities. Others face existential risks. We often refer to this new reality as '[Uncertainty squared](#)'.

When events are unfolding at such speed, it can be nearly impossible for leaders keep pace and respond to these challenges. So how do we build a resilient, innovation-ready enterprise that can overcome unexpected shocks, switch course overnight, and reinvent how it provides value?

Capgemini's [TechnoVision](#) introduces the mantra 'Being Like Water' to illustrate the capabilities that make an organization agile, responsive, adaptive, resilient, and creative – while continuously working towards achieving its objectives. Information Technology (IT) plays a crucial role in enabling this. Innovation-ready IT means platforms, practices, and partnerships that can be designed to not only enable fast, full-scale change, but to act as a catalyst for change.

To instill these capabilities across platforms, practices, and partnerships as an innovation-ready enterprise, organizations must focus on seven key imperatives:

Future fit technology strategy

Meet future customer, business and employee needs with a future fit technology strategy to increase adaptivity, creativity, and resilience

						
Demonstrate the business value of technology	Deliver a high-performing organization	Embed privacy and cybersecurity throughout the enterprise	Optimize your tech stack and services	Enable an insights-driven business	Evaluate and experiment with emerging tech	Inspire and innovate with technology
Implement techniques to measure and promote the business value of technology	Define and optimize an operating to deliver high performance	Increase business continuity mitigate cybersecurity and risks	Improve the speed, flexibility, resilience, and cost across hardware, software, and services	Coordinate strategy, roadmaps, operating systems	Cultivate a foundation to deliver value from emerging tech investments	Proactively seek out emerging technologies, novel solutions, and unconventional approaches

Achieving innovation with the right assets and mindset

To achieve these seven imperatives, most organizations must rethink how they view their assets.

Most of us understand 'business assets' as resources with economic value that an organization owns or controls, and they deliver benefits by generating cashflow, cutting expenses, improving sales or otherwise. Assets are usually reported on an organization's balance sheet and bought or created to increase a firm's value or benefit its operations. They can be tangible – manufacturing equipment, for example – or intangible – such as a patent.

Leadership teams often think differently about digital assets, which are much less tangible. Digital assets cannot be seen or felt and therefore often don't receive the same management attention and investment as physical assets. This is a mistake. Innovation-ready enterprises rely on 'digital' assets to provide the agility, creativity, and resilience needed to respond to sudden, systemic changes. Digital assets are the key to an organization's future-fit strategy: the means by which they instill adaptability, creativity, and resilience across their people, practices and partnerships, to respond to continuous change. A truly innovation-ready enterprise implements these [five key 'digital assets'](#) which can be accomplished by cloud native technologies:

- A unified digital platform
- Agile solution delivery
- Advanced analytics capabilities

- Hyper-automation and
- Business-IT fusion

With these digital assets in place, organizations can balance technological innovation with stakeholders' digital happiness and wellbeing by defining a 'future fit' roadmap that matches their innovation requirements with the latest tech trends. The result: organizational resilience, agility, and creativity – giving enterprises the confidence to lean into the future and make it work for them.

Implementing cloud native with the mindset that – **it must be easy to do it the right way** – enables teams to focus on the creation of business functionality where security and compliance is built-in to the 'unified digital platform'. A mantra that teams must be able to – **focus on business functionality before lunch** – makes the drive for hyper-automation and business-IT fusion real.

With these mindsets instilled in the cloud native adoption, the focus will be on the creation of business functionality. And with business functionality comes innovation.

Benefits of cloud native adoption

Cloud native adoptions require effort to properly establish, but once all the components are in place and personnel is trained, organizations can start reaping the benefits.

Accelerated reach and time-to-market

Cloud native applications also allow organizations to accelerate time to market by reducing wait times on infrastructure provisioning, with new resources available within minutes. It can also allow them to open up new markets around the world via the global span of cloud providers.

Enhanced customer experience

Composability of cloud native applications improves the customer experience by minimizing response times and downtime leveraging scalability, availability reliability capabilities.

The feedback loop is accelerated, from user comments to features - deployed in days. A/B testing is used to measure feature's effectiveness on targeted users while a progressive rollout maintains the service up and running during deployment.

Cost effectiveness

While increased cost is often the outcome with simple "Lift and Shift" scenarios, the adoption of cloud native solutions allows companies to use design patterns that can reduce costs while improving flexibility and availability. Elastic scaling and automation allow resources to be optimized to meet defined service level agreements, increasing cost efficiency. Asynchronous messaging solutions allow companies to flatten demand peaks, reducing the maximum size requirements for individual resources, thereby gaining operational efficiencies and improving the sustainability rating of the application.

Fortified security

Organizations thinking about adopting a cloud strategy often cite security as a principal concern, yet it could be argued that security is increased by moving to a cloud native solution. Often organizations maintain a security perimeter within which little to no checks are undertaken as to the provenance of API calls / service interactions. Cloud native applications combined with zero-trust patterns, ensure every call is validated to control its origin and the user's authorization and credentials to access the relevant resources. Furthermore, most cloud providers dedicate more resources to ensuring their systems remain secure, and together with active scanning and heuristics allows them to respond more quickly to threats.

Sustainability tracking

Many cloud providers provide calculators which enable organizations to calculate their total CO₂ footprint relating to their cloud infrastructure, helping to meet their sustainability targets. These can be used to identify how cloud native applications could reduce their overall footprint.

Exploring business growth with generative AI

Generative AI is disrupting and helping the reimagining the businesses through new possibilities and innovative solutions. Sogeti in alliance with Microsoft has a big focus on helping the customer relook at their portfolios and identifying the Generative AI use cases aligned with Microsoft's generative AI interactions models i.e. Beside, Inside and outside where in:

- **Beside** – Is about identifying the use cases around the cloud native developments, application modernization and other inflight projects where generative AI can help bringing the productivity boost to an end-to-end software development lifecycle.
- **Inside** – Are the AI powered application development scenarios, these applications are smart applications built on the data platforms running large language models, generative AI foundation model on top of it. These applications are built to bring the benefits around service innovations, faster time to market, better customer experience and revenue growth for business.
- **Outside** – Generative AI scenarios are helping to integrate, orchestrate different systems, end to end workflow automation through AI agents to bring operation excellence and cost reduction for business.

Sogeti uses these interactions models to bring business values for the customers in three broad scenarios: service innovation, productivity boost, and customer experience. We guide our customers towards the path to become an innovation ready organization by following the five cloud adoption principles.

Principle 1: More business innovation with less IT

Cloud native success is strongly connected to the innovation focus of the business and requires a strong integration between technology and business leaders. Business and IT units must grow even closer in the search for a productive balance. As a result, **fusion teams** are on the rise. [According to Gartner](#), "a fusion team is a multidisciplinary team that blends technology or analytics and business domain expertise and shares accountability for business and technology outcomes. Instead of organizing work by functions or technologies, fusion teams are typically organized by the cross-cutting business capabilities, business outcomes or customer outcomes they support."

"It's the speed with which a fusion team can conceive, design, build and deploy an app within the Microsoft ecosystem that makes this approach so valuable."

Alison Mulligan, Carl Cookson, Matt Beard | [How Fusion Development and advances in AI tools are helping developers get apps to market quicker](#)

When reading [Microsoft News](#), there's an overwhelming thread of business scenarios that emphasize how to cross-pollenate efforts between teams using resources that accelerate innovation:

- Mixed reality is transforming training
- Power Platform saves time and empowers every employee
- Microsoft Teams keeps everyone connected
- Dynamics 365 mixed reality boosts efficiency and scalability
- Employee engagement is being driven with Microsoft Teams

Moreover, teams need to be ready to apply these resources and join in efforts to innovate within the context of the age of AI. This reach of this evolution affects workflows across IT, development, and business team workflows alike. [As announced at Microsoft Build 2023](#), this includes the injection of AI in productivity software plugins, Teams message extensions, Power Platform Connectors, and [even security response from defenders](#). Successful fusion teams will be enabled not only by the organization's ability to organize its business and IT units, but by its effectiveness in weaving together the right toolchains and AI within its cloud-native architecture approach.

Take a look at these inspirational business cases as examples:

- [Ready for take-off: KLM and Sogeti test digital cockpit with AI](#)
- [Accelerating the UK Health Security Agency's COVID-19 mobile app testing](#)
- [Banner Health Transforms its Digital Patient Journey](#)
- [Rabobank takes the next step on its digital transformation journey](#)
- [Enxsis goes agile with public cloud migration](#)

Modular business processes in a platform organization

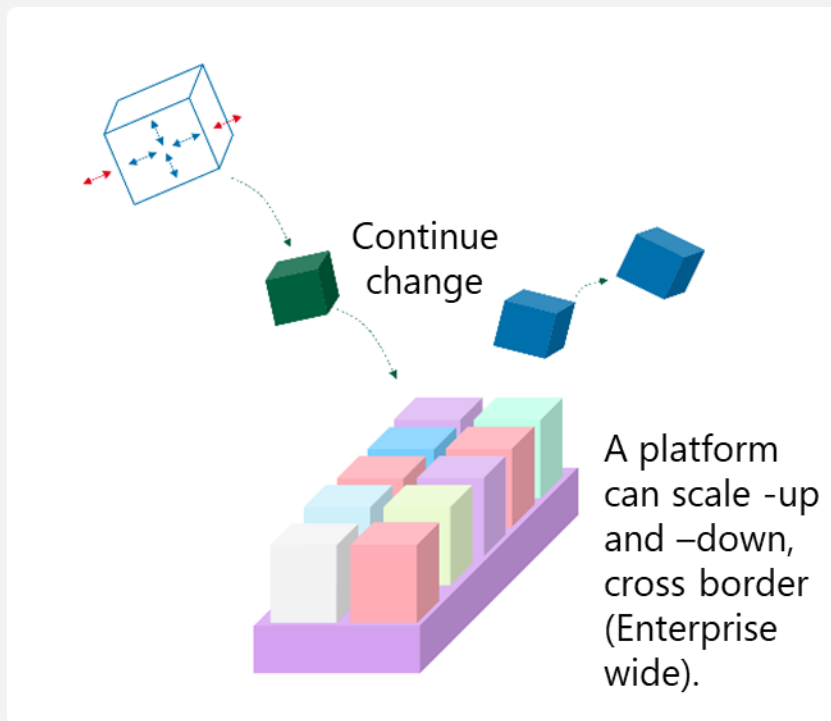
Business agility and resilience are usually empowered by modularity in software development. Complex systems are divided into modules which can be developed or maintained independently. In other words, they can be replaced or released in their own frequency (hourly, daily, or monthly) and evolve without impact on the business.

These independent modules expose functionality via external Application Program Interfaces (APIs) which are used to combine the modules together into a full business system. Examples of modularity in the industry are the [automotive and aircraft manufacturing sectors](#) where modularity has been widely accepted for many years, as well as in [the design industry](#). Module-based design in software development has been used since the 60s, and from there it has been the base of any software system with a result in several leading design methodologies as object orienting and domain driven design.

Many enterprise managers know that when this module-oriented approach is applied to the whole enterprise, it can bring the same benefits as it brings to software systems; it allows enterprises to change infinitely by adjusting 'business' modules as needed, reacting swiftly and smoothly to any new customer demand and market change. Organizations start to define services to become future-ready, changing the organization's team structure by applying Conway's Law.

'Any organization that designs a system (defined broadly) will produce a design whose structure is a copy of the organization's communication structure.'

How Do Committees Invent? by Melvin E. Conway, 1968



It is a challenge to transform an existing organization into a platform or module-based organization, that is more resilient, more flexible, and more competitive. This is also called a platform-based organization with full control to scale-up or scale-down all capabilities and to extend with new capabilities in a fast and flexible way.

Reconstructing the organization with independent, pluggable business services will bring with it innovation power. Teams can focus on the one, single business service they are responsible for, acting like true self-controlled product-oriented teams.

Product-oriented development teams

Reconstructing the organization with independent, pluggable business services will bring with it innovation power. Teams can focus on the one, single business service they are responsible for, acting like true self-controlled product-oriented teams.

Be familiar with the Day 0/Day 1/Day 2 approach within the software development lifecycle.

Day 0: Design and requirements

Day 1: Development and deployment

Day 2: Customer availability and maintenance

A traditional project closing on Day 1 with a software release - after the Day 0 design phase - is now shifting incorporating continuous operations and improvements on Day 2 in the product model.

At the age of cloud, Day 0 is no longer focusing on infrastructure provisioning while Day 2 pulls up the operations at the software layer.

Product oriented teams who have end-to-end ownership of a product or service with a strong product owner are one of the top common factors that correlate with realizing business functionality and the innovation power of the enterprise.

Previously, we discussed the motivation of moving to a “product-focused” delivery model, where DevOps product teams take full ownership of the end-to-end cycle of a product or service, while implementing iterative budgeting processes instead of large-scale annual budgets. But if software products are dispersed across

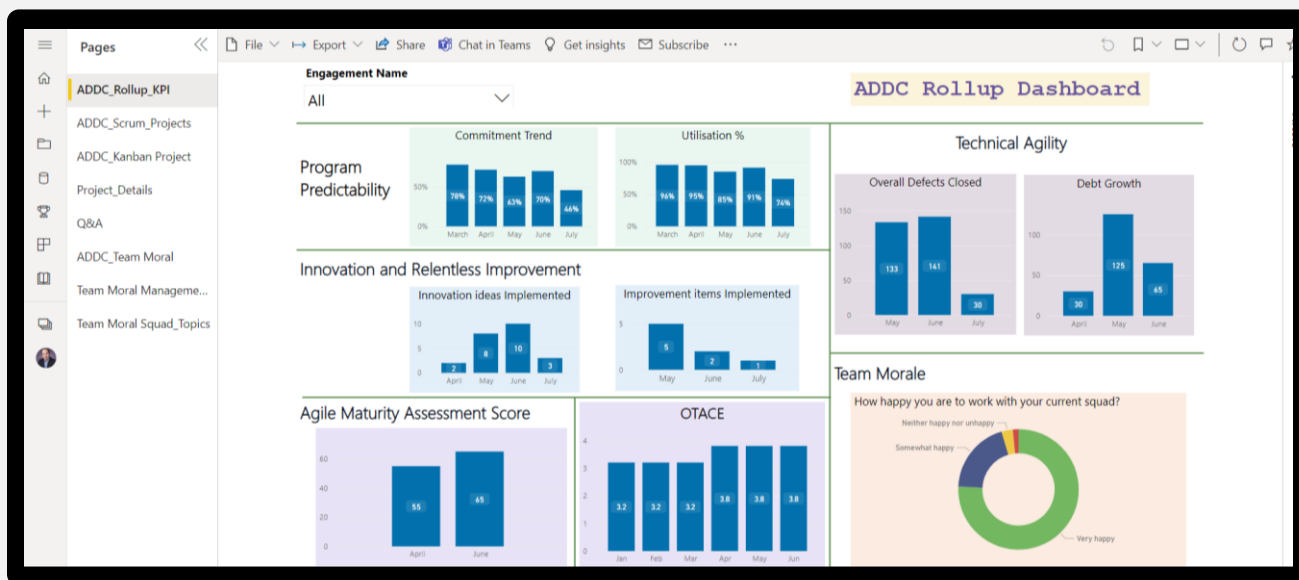
fragmented teams, business units and organizations, the lack of ownership will result in expensive duplication of systems and efforts, and disjointed customer support will decrease overall customer satisfaction.

Moving towards a platform organization with product-oriented development teams allows innovation to thrive within a modular approach, but it requires some effort as teams need to take ownership. Along with the business owner, they need to foster the freedom to experiment and innovate without sacrificing security and compliance agreed throughout the organization. The enterprise needs to be ready for these teams and embrace a unified approach to take care control is not lost.

This enterprise readiness starts at the highest level in the organization, as the whole budgeting process is impacted. The cross-functional, self-managed team focusing on one product goal has been part of the Scrum guide for years, however deviations from this reality often start with the way budgets are allocated.

Starting a product-oriented organization requires two core aspects:

1. **Build a training plan:** Including product management, agile methodologies like Scrum or Kanban, design thinking, market analysis, etc.
2. **Give ownership via InnerSource:** This helps control consistency and measure team readiness for delivering business services.



Sogeti ADDC (Agile DevOps Delivery Center) rollup dashboard

Get compliance and security to onboard and understand cloud native

Compliance and security teams don't want to be the blockers for business innovation. However, ensuring that the organization's operations, data, and systems are secure and compliant with relevant laws, regulations, and enterprise standards is not easy with the fast-moving targets of cloud native systems. With a clear

understanding of the benefits and risks associated with cloud native technologies, compliance and security teams alongside the business and DevOps teams can work together to ensure that enterprise data and systems are secure and compliant.

From a security and compliance perspective, cloud native systems implemented with either an event-driven or a microservices architecture are different from traditional implementations. Some key areas to focus on are:

- **Data flow:** In an event-driven architecture, data is processed in real-time as events occur. This requires the implementation of strong security controls to protect the data as it flows through the system.
- **Increased complexity:** Event-driven architecture can be more complex than traditional architectures due to asynchrony, loose coupling, event processing, and scalability.
- **Interservice security:** Which can increase the risk of misconfigurations or vulnerabilities in one service affecting the security of other services.
- **Decentralized processing:** Which can increase the risk of unauthorized access to sensitive data.
- **Decentralized data:** Data is often stored in separate, decentralized databases, which can increase the risk of unauthorized access or data breaches.
- **Dependence on APIs:** Which can increase the risk of API-related security threats, such as injection attacks and unauthorized access.
- **Increased communication:** Which can increase the risk of network-related security threats, such as man-in-the-middle attacks.

Readying security teams for cloud native

The goal should be to achieve an understanding of the security and compliance requirements that the enterprises must meet, how they are implemented and controlled to work together, and how cloud native systems take care of these requirements.

Once you have a good understanding of how the above focus areas pertain to your organization, the next step is to provide training and resources to help compliance and security teams understand cloud native technologies and best practices. As a baseline, supply understanding on how cloud service providers handle compliance and what measures are in place to ensure that data and systems are protected.

Then standardize how cloud native systems are implemented and use industry standards for it to support security officers and make their life less complex. Standardization increases the compliance and security of the systems, so risk managers don't have to reinvent the wheel.

Business alignment



Risk insights

Integrate security insights into risk management framework and digital initiatives



Security integration

Integrate security insights and practices into business and IT processes, integrate security disciplines together



Business resilience

Ensure organization can operate during attacks and rapidly regain full operation status

Security discipline



Access control

Establish Zero Trust access model to modern and legacy assets using identity and network controls



Security operations

Detect, respond, and recover from attacks; hunt for hidden threats; share threat intelligence broadly



Asset protection

Protect sensitive data and systems; continuously discover, classify, and secure assets



Security governance

Continuously identify, measure, and manage security posture to reduce risk and maintain compliance



Innovation security

Integrate security into DevSecOps processes; align security, development, and operations practices

For more guidance and best practices on secure cloud adoption, visit [Microsoft Learn](#) to read about:

- Mapping roles, responsibilities, and standards
- Guiding lasting transformation through culture and ownership
- Recruiting and growing diverse skillsets
- Cybersecurity architectures and frameworks for the enterprise

Business outcomes with objectives and key results (OKRs)

Business involvement is crucial to the success of cloud native. You increase this involvement by working together in one team (DevOps), and above all by speaking one common language. This common language must match the perception and ambition of the business. Therefore, it is clear that we define common objectives for business and IT and outline them distinctly. One approach is to structure the Objectives and Key Results (OKR) throughout the organization as a cascading model. This means that business objectives are traceable 1-to-1 from the IT objectives and vice versa, fully automated and without the intervention of Excel and manual processes.

This unified approach provides immediate transparency throughout the organization. The business can make quicker and clearer choices and IT can easily visualize the added value of IT systems, or visualize near-real-time performance in key results that are recognizable for the business.

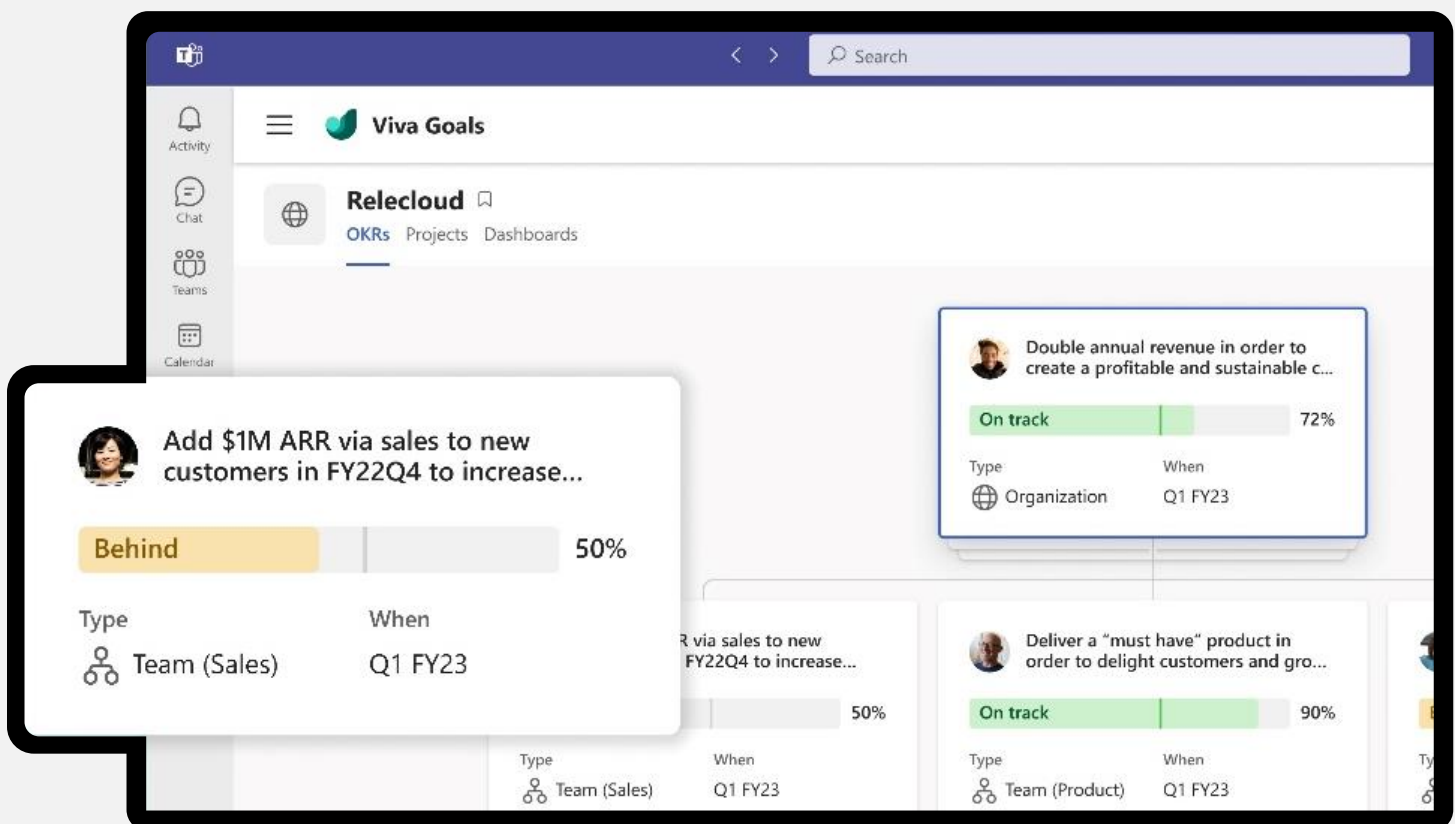
Five Cloud Native adoption principles for enterprises

The OKR framework consists of two parts:

1. **Objectives:** High-level, qualitative statements that describe what an organization wants to achieve. Objectives should be inspiring, meaningful, and specific to the organization mission. They should also be time-bound, so that progress can be tracked and evaluated.
2. **Key Results:** These are specific, measurable outcomes that demonstrate progress towards the objective. Key results should be quantifiable and time-bound, and they should be used to track progress and evaluate success. They should be specific enough to allow for clear and meaningful measurement, but also flexible enough to allow for adjustments if necessary.

Define OKR's at top level and cascade them down to teams using tools that are visible for everyone.

A great example is [Viva Goals](#), an integrated Microsoft solution to define and track goals. Capable to integrate with team dashboards and tracking systems like [GitHub](#).



Examples of OKRs within Viva Goals

Example

To illustrate the framework of an OKR, see a general example created below using generative AI technology from <https://chat.openai.com/chat>:

Objective: Increase our competitiveness in the market by modernizing our legacy applications and migrating to a cloud-native architecture by the end of the year.

Key results

-  Conduct market research to understand the current trends and best practices in cloud-native application development.
-  Complete a thorough assessment of our current applications and identify the top 5 priority applications for modernization based on potential impact on our competitiveness.
-  Develop a migration plan for each of the top 5 priority applications, including timelines, resources, and expected outcomes.
-  Train and upskill a team of developers on cloud-native development best practices and technologies.
-  Successfully migrate 2 of the top 5 priority applications to a cloud-native architecture within the first half of the year.
-  Achieve a 50% reduction in application downtime and a 30% increase in performance for the 2 migrated applications within the first quarter after migration.
-  Secure approval and budget for the remaining 3 applications to be migrated in the second half of the year.
-  Successfully migrate the remaining 3 applications to a cloud-native architecture by the end of the year.
-  Measure and report on the impact of the modernization project on our competitiveness in the market.

The business case calculator: Evaluating the current application landscape

Enterprises are always facing immense pressure to outperform their competitors and accelerate their growth. For any organization to remain relevant and effective, their existing software applications require flexibility for adoption of new business models that develop and maintain customer loyalty. Adopting new business models on an existing legacy application also requires changes in legacy business processes. But modifying existing business processes running for legacy may have blockers like:

- Risk of a small change forcing full application release and testing
- Long bug fixing cycles due to tight coupling of application modules
- Wide testing scope to locate any bugs in production
- Longer release cycles even for small changes
- High cost of change implementation
- Technical debt and End of Life support

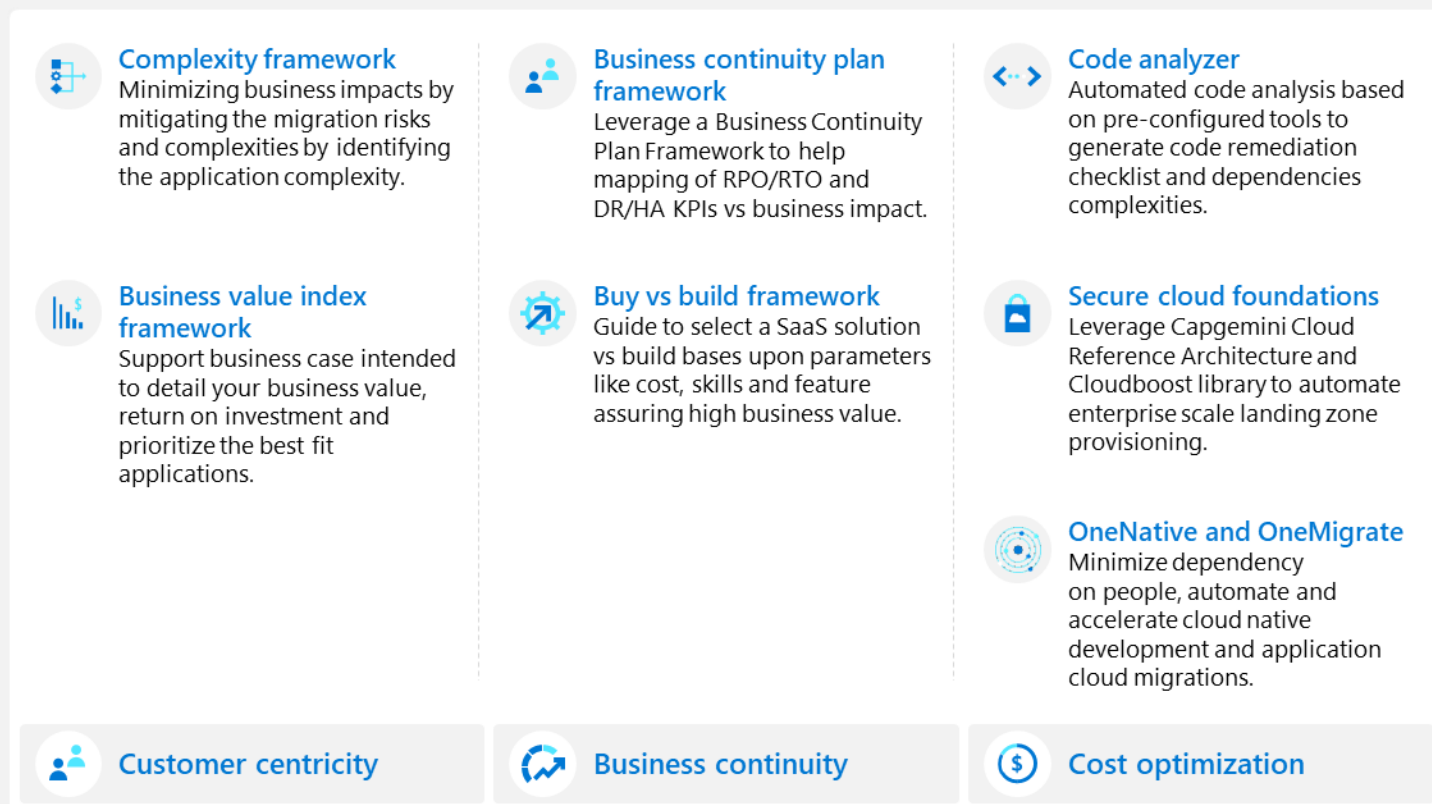
Five Cloud Native adoption principles for enterprises

Recognizing these challenges, customers need guidance on where to start on how to reduce technical debt and how it can be done with a cost-neutral approach that prevents impact to their business. The following business principles will help with this guidance:

 Customer centric	 Business continuity	 Cost optimization
✓ Transaction execution time reduction ✓ Faster releases of new features ✓ No learning curve post migration ✓ Application availability ✓ Customer data reliability ✓ Enhances user experience ✓ Business agility ✓ Scalable and resilient architecture	✓ Simplify complex migrations ✓ Load balanced ✓ DR/HA enablement ✓ RPO/RTO improvements ✓ Failback/failover automation ✓ Security compliant ✓ Zero/no trust access policies ✓ Always in control	✓ Technology upgrade ✓ Skills optimization ✓ Operations cost reduction ✓ Compute cost reduction ✓ Application tier standardization ✓ Minimal code changes ✓ Tools standardization ✓ Release automation ✓ Backup process automation ✓ Reduced manual errors

Our approach towards these principles is defined using a business calculator methodology built on key pillars (i.e. complexity framework, business continuity plan framework and business value index framework). Many of our customers struggle to achieve maximum return on investments, but our business value calculator methodology helps customers in such scenarios define strong business cases showing quantifiable return on investments.

These business value scores are calculated based upon criteria like application size and technical complexity, business criticality, scope of improvement for an application and quantifiable business benefits around operational cost reduction as well as faster time-to-market for new features and business models. This approach of using a business value calculator methodology also helps prioritize application modernization with a tradeoff between modernization cost versus achievable business values, which eventually fast tracks the digital readiness journey of any organization. Below is an example of the output from our business value calculator methodology, helping our customer's score a maximum business value Index.



As shown above, our business value calculator methodology is built on the principles of Customer Centricity, Business Continuity, and Cost Optimization, so let's see what this means for a typical organization:

Complexity Framework helps us collect data around pre-defined parameters about an application for its business functionality, technology, infrastructure, authentication / authorization approach, security compliances, database, release management process, operations management process, dependencies with other business critical applications, any existing pain points, and owners. All this information is analyzed through our application complexity framework to define any application as simple, medium, complex, and very complex.

Next comes the **Business Continuity Plan Framework** which is used to analyze the information about requirements like availability, backup strategy, RTO/RPO KPIs, dollar impact per second of downtime of a given application. This framework categorizes the various tiers of application where Tier 1 could be just backup requirements with no DR (Disaster Recovery), while Tier 5 would be for a mission critical application which requires an active DR implementation on all layers with frequent data backup strategy to meet the RTO/RPO KPIs in minutes.

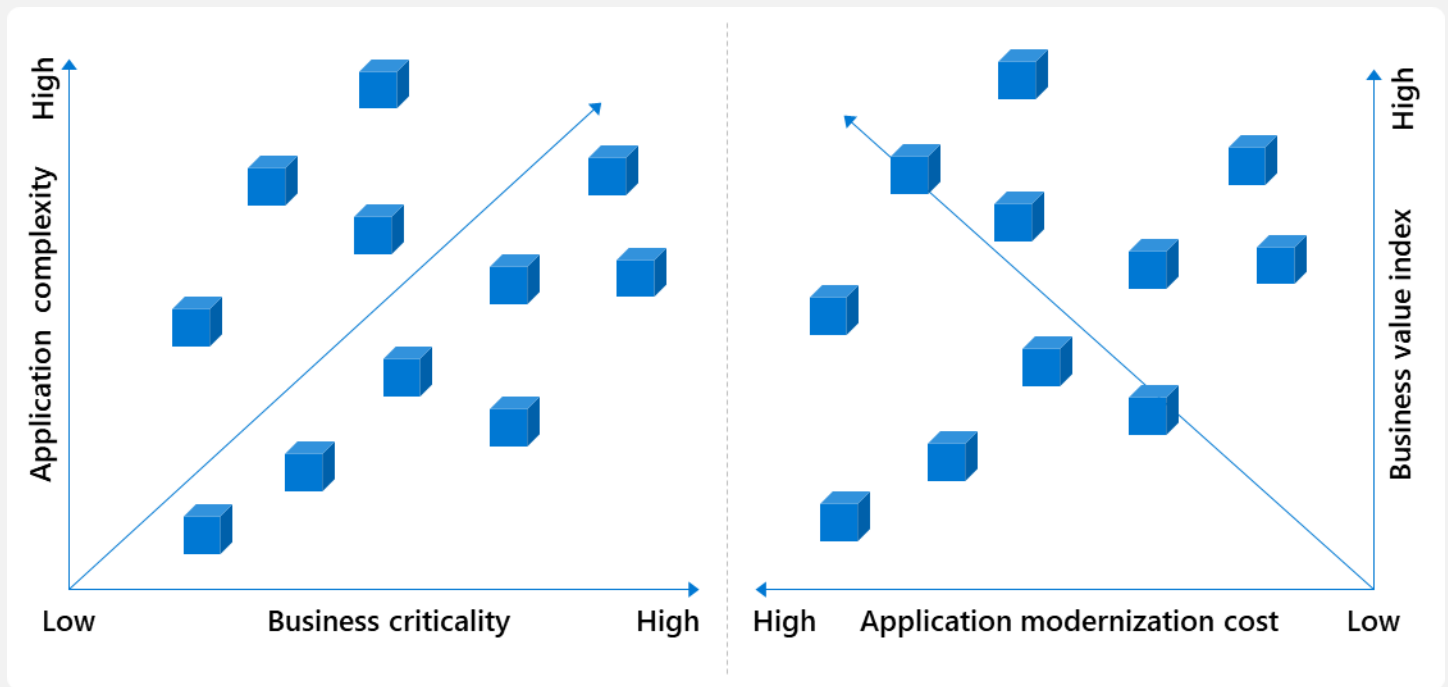
Once all the available information is analyzed through the Complexity Framework and Business Continuity Framework the Business Value Index Framework is then applied. This is mainly based upon how much remediation cost it will take to reach the target modernization state of an application, parameters of data collection and analysis around application current state, pain points, improvement areas and achievable target state.

 Application current state	 Pain points	 Improvement areas	 Modernized state
Application technology stack Application size Application complexity Application architecture monolithic vs multi tier Deployment process and tools ETL, batch jobs Dependencies Business criticality	Performance Availability/reliability Feature release cycle Operations Security Skills and expertise	SDK version upgrade OS, DB, middleware version upgrade Release process improvement Load balancer, DR & HA upgrade Database type conversion Legacy, ETL upgrades Security and compliance changes Dependencies, third party impacts Monolithic layers to tiers changes	Technology stack Layers and tiers architecture Release process BCP strategy and approach Backup strategy Deployment patterns – AVS, Openshift, Containerization, DR/HA

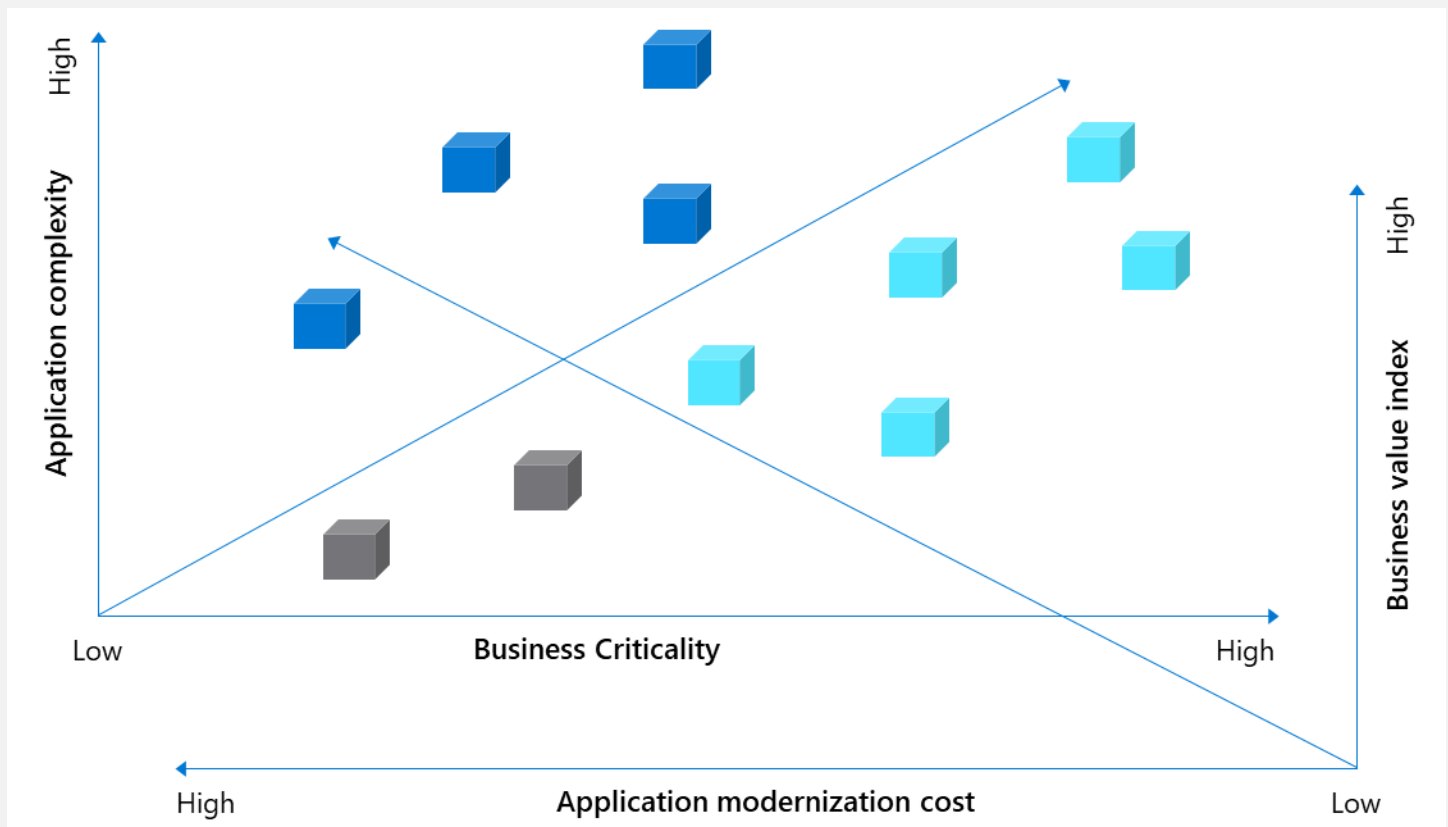
Each of the above criteria are scored from 0 – 5 as per their current level of implementation. The total score is used to determine what the business value modernized state will achieve vs. the current state and the score definition is:

- 0 – Nothing implemented
- 1 – In discussion
- 2 – Implementation in progress
- 3 – Basic Implementation available
- 4 – Implementation but needs upgrade
- 5 – Fully Implemented and fully functional

Using this scoring methodology in the existing framework helps to represent applications in different quadrants of four categories: application complexity, business criticality, modernization cost and business value it will bring when it reaches the defined target state through the framework. An example of this is shown below.



This analysis shows a set of applications; the first graph shows the mapping of application complexity vs. their business criticality; the second graph shows modernization cost and the business value it will achieve for the customer. By overlapping these graphs, we can achieve a map of which applications will generate the greatest business impact from modernization.



Notice how in the image above, the applications that have the highest business criticality and business value are highlighted are centralized toward the right. While applications that are most costly and least impactful are concentrated towards the lower left.

Applications shown in light blue are priority candidates for modernization programs. Although their complexity ranges from medium to high, the modernization cost remains low to medium. Therefore, the business value it will achieve is high and that's the key objective.

Applications in blue should be part of the modernization strategy next as the business value index is high but it will come on high cost of modernization.

Applications shown in grey indicate where customers may need to rethink their strategy as these applications would bring a low business value at a higher cost.

Using the calculator methodology in this way, we can help our customers in building their digital readiness and cloud native strategy based upon business value index instead of adopting a purely technology focus. Aligning IT and business becomes a seamless process, and the probability of success greatly increases for the digital readiness program, based upon principles of customer centricity, business continuity and cost optimization.

Readiness assessments

Many enterprises have already adopted cloud, most have a reference architecture and there are already several or many systems deployed. Still before adopting a cloud native approach, it is wise to challenge the current state and understand the completeness, readiness, and maturity.

This readiness assessment covers the technology capabilities of the organization and other topics discussed in this chapter, such as how the business is connected to the IT department and how teams deliver software systems.

In the first stage of the cloud native adoption, it's important to understand how well the enterprise is enabled and if there are any organizational blockers that need to be resolved first. Later, organizations can focus on the readiness of the landing zone and the capabilities of teams.

Microsoft offers a wide range of online [assessments](#) which offer key insights in organizations' state of readiness and focus areas.

- [Cloud Adoption Strategy Evaluator](#)
- [Cloud Adoption Security Review](#)
- [Developer Velocity Assessment](#)
- [DevOps Capability Assessment](#)
- [App and Data Modernization Readiness Tool](#)

Improve your results

Our recommendations for improving your results are organized by category below.

Recommendations Unanswered

Identifying your motivations MEDIUM



6 recommended actions

Show less ^

When responding to critical business event that drives cloud adoption, review the get started guide for Accelerating Migration.

Explore the Migrate methodology in the Cloud Adoption Framework which outlines the strategy for performing a cloud migration.

Explore the Migrate methodology in the Cloud Adoption Framework which outlines the strategy for performing a cloud migration.

Explore the Innovate methodology of the Cloud Adoption Framework that guides the development of new products and services.

Review the Cloud Center of Excellence guidance.

Review examples of cloud adoption plan with four horizons.

Documenting business outcomes you expect LOW



3 recommended actions

Show less ^

Basic concepts of fiscal outcome conversations are around revenue, cost and profit. Read more here

The ability to respond to and drive market change quickly is the fundamental measure of business agility. View Samples here

Learn more about business outcomes and business outcome template

Financial considerations for you to evaluate HIGH



1 recommended action

Show less ^

Learn ways to achieve more with your cloud investments

Technical Considerations in creating your business case HIGH



6 recommended actions

Show less ^

Achieve more with your investment in the cloud

Achieve more with your investment in the cloud.

Estimate how much you can save through rightsizing with the Azure Virtual Machines cost estimator.

First project criteria and expectations.

Azure Hybrid Benefit for Windows Server

Estimate how much you can save through reserved instances with the Azure Virtual Machines cost estimator

Organizational alignment and Skills Readiness Plans MEDIUM



3 recommended actions

Show less ^

Explore organizational and technical readiness learning paths.

Learning module - Review the Azure compute services and explore how they solve common business needs.

CCoE chart and aligning owners that will drive it in the org

Creating your business case HIGH



0 recommended actions

Show less ^

OKRs for Principle 1: More business innovation with less IT

Objectives	Key results
Adopt a product-oriented approach	- Skill up your team using product-oriented trainings with related product-centric culture - Incorporate user feedback in your product backlogs <i>Tip: Link customer feedback to your product's backlog.</i> - Increase 3 to 5 key business metrics by a specific percentage (product usage, customer engagement, funnel and conversion, revenues) - Improve 3 to 5 key technical metrics by a specific percentage (defects and outages, performance, time to market to release a new feature)
Implement an OKRs strategy across the organization	- Implement a dashboard to align business goals with objectives and periodically measure the key results - Make the dashboard transparent for all involved stakeholders and easily break down into details
Setup a Value realization practice to evaluate business outcomes	- Define business and IT drivers with expected benefits (quantitative and qualitative) on 80% of your initiatives. - Increase cloud native service utilization ratio and evaluate correlation with time to market acceleration, quality improvement and overall cost optimization. - Add a value realization approach within your FinOps practice
Evaluate the business value and actual costs of the application landscape	- Perform a business value assessment on your applications to prioritize business outcomes of application modernization - Calculate your readiness score on multiple dimensions like cloud strategy, DevOps capabilities, security, and developer velocity to define an action plan and measure progress
Enable fast and lightweight innovation	- Define lightweight innovation environments to onboard any new experiments based on cloud native / serverless services - Measure cost and footprints of innovation, sandbox, and non-production environments - Define 3 to 5 fast prototyping capabilities supported by generative AI to provide requirement and gap analysis, code generation, user interface, data sampling and testing

Principle 2: Control with policy-governed practices

Speed and independence are a must for teams to be agile and innovative. But it's critical that security and compliance are not compromised in any way in this drive to innovate. Cloud native technologies help bring control with the flexibility needed for an innovation-ready enterprise.



Achieve security compliance by working with policy governed practices that do not dictate but that empower and monitor the state.

In prior years, policy enforcement roles were established to work centrally; the security and compliance role was a check that could bottleneck the organization. But to become an innovation-ready enterprise, organizations need to evolve from a restrictive and centralized governance model to a model of secure enablement.

Policies are not meant to be merely restrictive, but instead achieve two main objectives:

- To maintain a secure organization based by prohibiting actions that lead to threats
- To guide an organization on executable actions with unified objectives in mind

By building these policies collaboratively between IT and business, this set of rules and procedures will guide and regulate teams. These policies serve as a framework for decision-making, providing clear guidance on how teams work to ensure consistency in organizational operations. In practice, policy-governed practices are well documented, maintained and reviewed on a regular basis. To ensure that the policies are consistently applied without sacrificing team efficiency, automation is key. Although DevOps teams are cross-functional by nature, they simply can't cover every piece of knowledge and take responsibility of all aspects of the enterprise. Policies need to be designed, agreed upon, clearly articulated, and automated to help teams to conform to the needs of the enterprise.

In the Microsoft Cloud Adoption Framework, the governance section references five disciplines for cloud governance based on business risks and opportunities for controlled cloud adoption.

Define Corporate Policy



Business risks

Document evolving business risks and the business' tolerance for risk, based on data classification and application criticality



Policy and compliance

Convert risk decisions into policy statements to establish cloud adoption boundaries



Process

Establish processes to monitor violations and adherence to corporate policies

Five Disciplines of Cloud Governance



Cost management

Evaluate and monitor costs, limit IT spend, scale to meet need, create cost accountability



Security baseline

Ensure compliance with IT Security requirements by applying a security baseline to all adoption efforts



Resource consistency

Ensure consistency in resource configuration and enforce practices for onboarding, recovery, and discoverability



Identity baseline

Ensure the baseline for identity and access are enforced by consistently applying role definitions and assignments



Deployment acceleration

Accelerate deployment through centralization, consistency, and standardization across deployment templates

Learn more at [Microsoft Learn](#)

Define policies for all platforms

Beyond just the cloud platform, there are multiple other aspects to consider from a security and compliance perspective: the DevOps platform, the Low Code platform, how teams deliver software, what modular building blocks they use, and what compliance policies must be met. Moreover, policies need to be tuned based on both business risks and opportunities. It's highly necessary to prepare these security and compliance policies for this platform to stay within proper guardrails, but they need to secure as much as they need to realize business functionality and enable business opportunities.

Policies

Cloud	DevOps	Low code
Cost management	Security baseline	Security baseline
Security baseline	Intellectual property	Scaling boundaries
Identity baseline	Open- and InnerSource	Data privacy
Resource consistency	Release strategy	App consistency
Deployment acceleration	Responsibility	Cost management

There are different policy categories for Cloud, DevOps, and Low code platforms based on the capabilities and usages of these platforms:

- **Cloud:** Policies will cover identity baseline, resource consistency, and cost management.
- **DevOps:** Policies will cover intellectual property with the usage of open and InnerSource within the enterprise. This is also required for securing the software supply chain.
- **Low code:** Policies should remove the risk of out-of-control usages.
- All platforms (Cloud, DevOps and Low code) have the same secure baseline policy category

As a result, these policies enable an applied governance letting federated organizations making the required choices for their projects and apps without compromising neither the whole organization neither lowering velocity of these teams.

Automate policies

A design consideration when defining policies is that the policy guardrail, validation and even the countermeasure must be capable of being automated. When policies are automated, the enforcement is frictionless, and there is no need for manual intervention which takes time and inhibits the speed of establishing business functionality.

For instance, a policy definition can be used to [automatically create a DNS record](#) in an Azure Private DNS zone in hub and spoke network architecture when an Azure Private Link for a PaaS service is configured.

As a default, this automation support must be shifted left as much as possible, making sure that teams can act on it as early as possible in the early stages of development. For example, several tools exist to make the life of DevOps teams easier when it comes to complying with cloud policies. Using PSRule from Microsoft, we can create rules that test the compliance of our infrastructure as code scripts prior to deploying, removing the frustration of having pipelines fail due to policy denials. Furthermore, it also allows us to scan our IaC scripts to ensure compliance with the Microsoft Cloud Reference Architecture, ensuring the implementation of best practices.

Make proactive and reactive policies

From a central and more traditional view, the thinking is often that these policies should be enforced via strict guardrails, making sure teams can never cross the line. But in innovative organizations, these policies are not always set as hard lines so organizations and teams receive the flexibility to innovate they sometimes need to cross the finish line.

To enable teams without losing control, monitoring is key. It helps understand what teams are doing, supplies alerts when boundaries are crossed, and helps support teams with experts when they want to innovate. We call this “reactive policy enforcement” with just-in-time learning.

Proactive	Reactive
Leverage platform capabilities	Automated policy validation with Just-in-Time-learning
- Configured features and capabilities - Reusable templates and components - Self-service with guidance - InnerSource	Implementation of discovery tools and review cadences to bring in control and support in phases while driving towards a central and fully recognized source for information, guidance, direction and acceleration.

Proactive control provides the security and compliance boundaries by leveraging all platform capabilities like identity and access management. **Reactive control** gives teams the freedom to work in an innovative way, while keeping insights and providing guidance when needed.

Several policy examples are outlined below:

Proactive identity policy automation

Microservice systems serving a wide variety of businesses need proper security management. Identity policies are common in authentication management, who can login. For authorization management identity policies are not that common. This is a challenge given the growing amount of collaboration between business applications, since any action done by a user is fully contextual: what the user is allowed to do depends on the context, such as a workflow state, an ad-hoc role dynamically added, or a delegation given by a colleague or partner.

Security leaks come from authorization, especially in multi-roles apps with fully dynamic roles that evolve with the process (for example: corporate eBanking with maker-checker workflows). Such "dynamic" roles are usually managed everywhere in services. Managing them in a purely ad-hoc way (and everywhere in the code) is a recipe for a guaranteed security breach.

Standardization is key to avoid such issues. Defining a policy that dictates a single source of "access truth" with calls to the security engine, immediately at entry into the service, so that the security management is easily located by auditors is mandatory.

There are also positive only security policies, where no access to anything at the lowest level without any specific right (which can be given just for one call). Security by removal of rights is risky as a miscommunication within the team may lead to excessive access granted, whereas positive granting of rights is safe.

Reactive identity policy automation

A common security practice for applications is to "Use cloud vaults for secret management in all stages". To manage modern applications and our microservices, Kubernetes has become a good solution to orchestrate container-based applications. However, it is necessary to store the secrets! Kubernetes secrets bring a first

answer to have a safe to store sensitive data. However, secrets are described in YAML files encoded in base64, which are easy to decode.

To remove this security limitation, while managing the secrets from the Kubernetes cluster, we can propose a solution that allows us to manage the secrets and provide a security policy for our keys.

For this, there are many solutions provided natively by cloud providers and also in the open source world, such as HashiCorp Vault. Vault is one of the most commonly used solutions when you want to secure access to your secrets. It's also possible to outsource the management of your Kubernetes secrets in VAULT and guarantee their security.

Passwordless deployments, with the CI/CD automatically generating and storing passwords in vault, increases security while also ensuring regular recycling of passwords.

Proactive containers policy enforcement

Containers should only come from approved internal sources, all base containers should be scanned and maintained to reduce attack vectors. Base containers should be available for all major development languages used within the enterprise and pre-configured with standard functionality. Tools like Microsoft's SBOM (Software bill of material) generation tool should be used to document the detailed contents of the docker image that can be analyzed to assist in vulnerability detection in the deployed container. The published images should be signed using Docker's content trust mechanisms together with [the Notary project](#) to ensure that the contents have not been manipulated / altered.

Data Loss Prevention policies (Power Platform)

One of the most important assets which any organization possess is its data, whether it's a product, IP information, employees PI data, or otherwise. Most applications work on this data to create information, interfaces and knowledge which can be used in various activities.

Power Platform is no different. Hence while configuring the Power Platform environment, care needs to be taken to ensure that only required data is allowed to flow through various apps and that this data is gated. This helps to ensure data does not end up in systems and platforms that are not privy to the information.

One way of ensuring these policies are applied and followed in Power Platform is by applying data loss prevention (DLP) policies. Generally, DLP are restrictive, but Power Platform is directed towards not only IT-oriented users but also non-IT decision makers and power users. The DLP policy implementation should then strive for a balance between protectiveness and openness as too many restrictions may end up discouraging the end users from the platform.

In Power Apps, DLP policies enforce rules for which connectors can be used together by classifying connectors as either Business or Non-Business. These are also referred to as Data groups. If we put a connector in the Business group, it can only be used with other connectors from that group in any given app or flow. Sometimes we might want to block the usage of certain connectors altogether by classifying them as Blocked.

Five Cloud Native adoption principles for enterprises

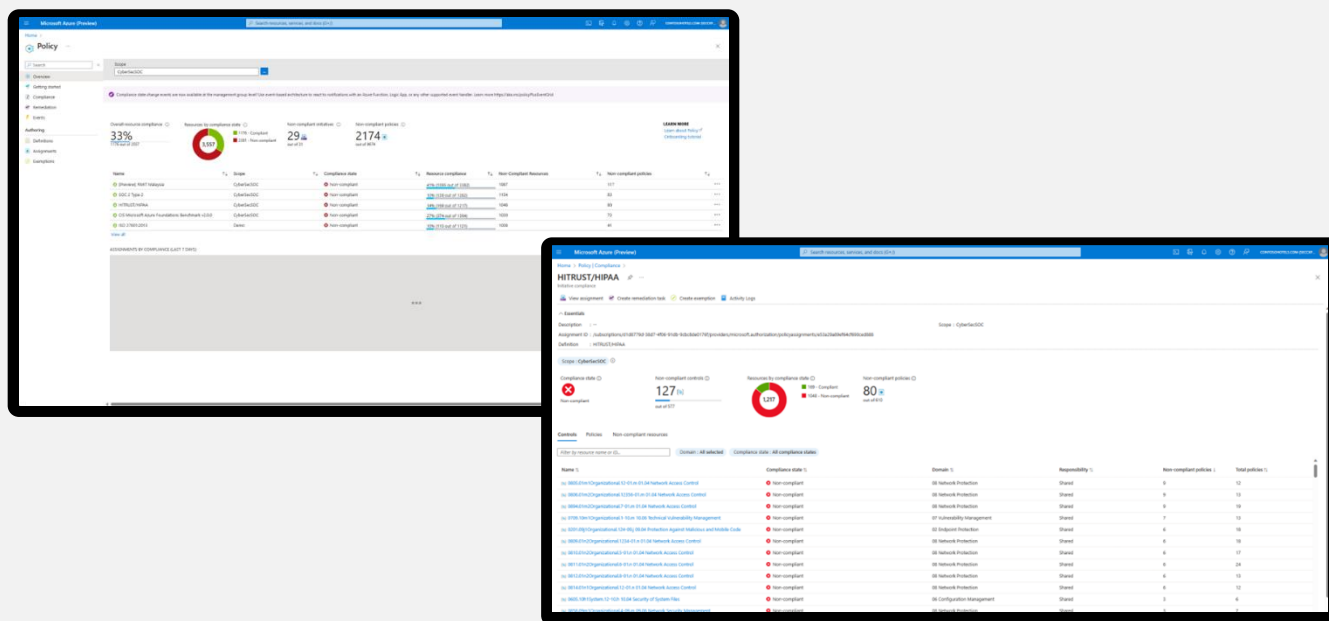
For example, if SharePoint and Teams are placed in a Business group and Twitter is placed in a non-business group and Twilio is added in a Blocked group, then any flow which combines SharePoint or Teams with Twitter will not function. Similarly, any flow which combines SharePoint with Twilio or Twitter with Twilio will not function. But a flow between SharePoint and Teams will function properly.

Multiple DLPs can be defined targeting different environments that give the flexibility to manage the level of restrictions according to environments purposes.

Policy compliance monitoring

It is important to have a robust policy compliance monitoring practice in place as part of the enterprise risk management strategy. Policies should be effectively followed so organizations know that the cloud native adoption is running in a manner that supports business goals. It's considered best practice to automate not just policy implementation, but policy review and evaluation as well. The goal is to ensure that policies are adhered to continuously without manual intervention. Using a continuous compliance dashboard helps to more confidently review that cloud native solutions are adopted successfully and that businesses are reaping the benefits.

An example of a continuous compliance dashboard is this landing zone design for healthcare where cloud workloads are continuously validated on HIPAA compliance.



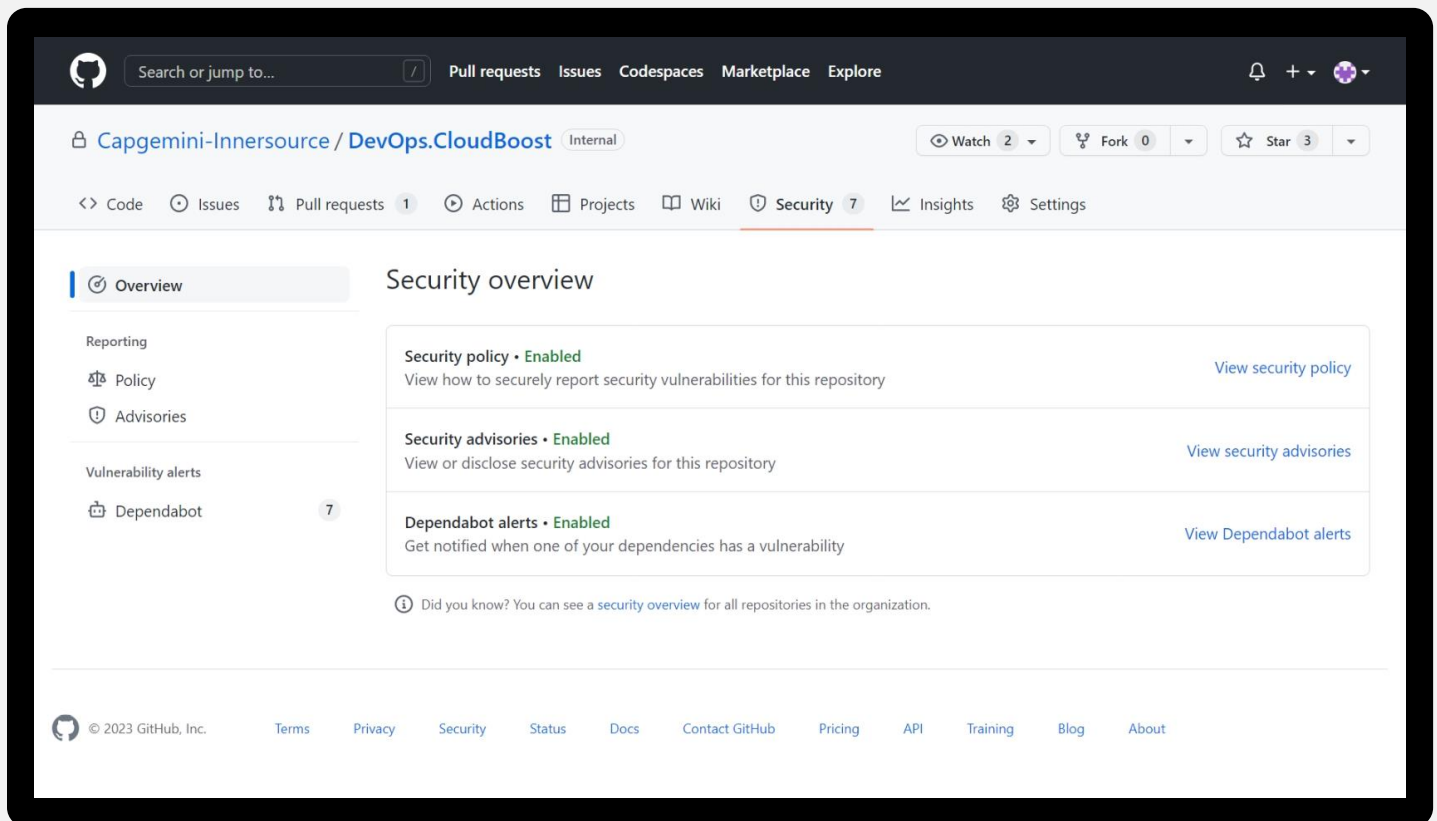
Just-in-Time learning

With the fast-moving nature of cloud native technologies and the evolving developer workforce, it can be difficult to keep all teams up to speed with the current policies and practices. Just-in-Time learning is a flexible and efficient practice which helps teams to learn and acquire the skills needed to stay compliant. It can happen in a variety of ways, but let's start with security checks in the developer workflow as an example.

Five Cloud Native adoption principles for enterprises

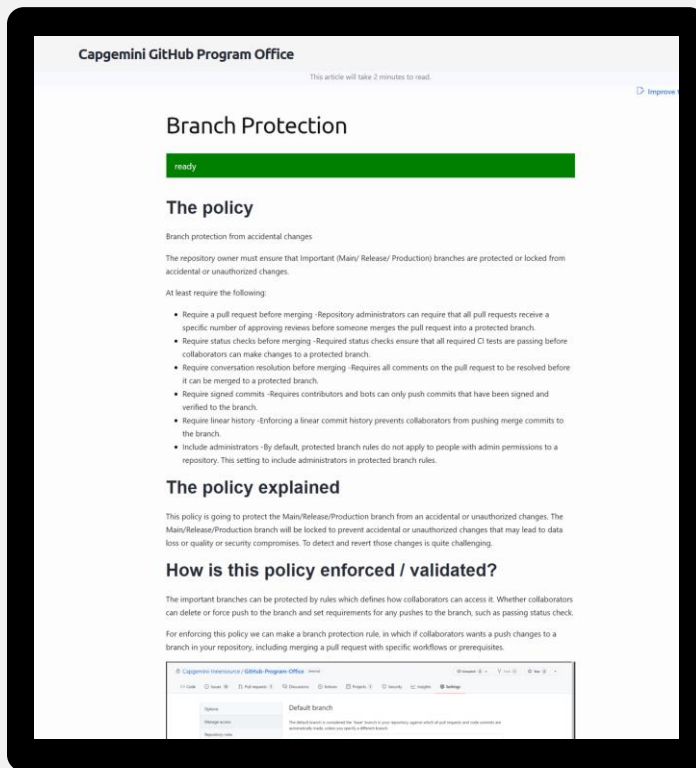
As a user, when you are assigned a pull request for the first time, you will get an e-mail addressing the importance of each review for the quality of the software solution, the best practices for execution of a code review, and how GitHub checks can help you automatically verify the pull request.

Imagine later that Dependabot finds a new vulnerability in your code. When this happens, it would be really helpful to have just-in-time-learning that points you to your security policy on certain levels of security incidents. This would not be something a developer would immediately think of when approving the pull request of Dependabot.

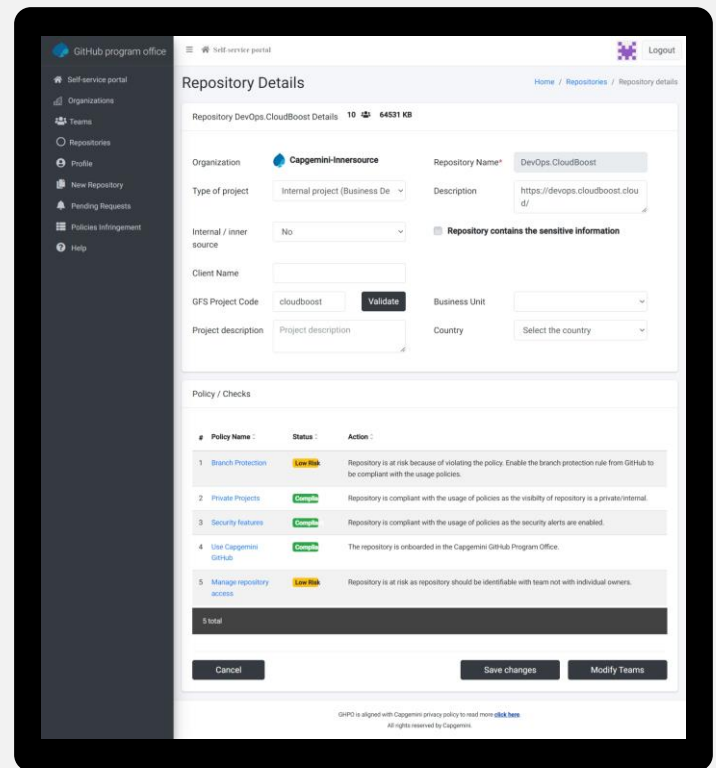


GitHub Security Overview dashboard linking to both security advisories and security policies

For policy guidance it can be implemented as a micro-learning practice where team members receive short, targeted learning experiences that are relevant to the policy that applies at that moment. This may be a video tutorial, a quick reference guide, or interactive simulations directly on the screen. When fully automated, new policies can be added instantly and made available to employees seamlessly.



An explanation of a policy from the Program Office



Repository details from Program Office

An extended implementation of just-in-time learning is to give an action next to the learning resource. For example, when a policy is not met, the team gets a notification with the explanation of why this is important, what the details are, and what steps are needed to become compliant again. To make it even easier for the team, this action can be automated. With a push of a button the team knows why, and what and is compliant once again.

OKRs for Principle 2: Control with policy-governed practices

Objectives	Key results
Start in control governing cloud resources with policies	- Define 1 to 3 levels to assign policies in the hierarchical structure of your organization using management group levels, subscriptions, and resource groups corresponding to entity/product/project/environment levels - Define and Assign policies as a baseline of your organization and levels. Use policy initiatives to group policies - Use tags with policy inheritances to locate and manage resources without complexifying resource naming - Extend policies for all platforms using DLP policies for Power Platform, APIM policies for APIs definition consistency and resources outside of Azure with Azure Arc <i>Tips: Use hierarchical structure or policy scope exclusion to avoid policy effect when needed</i> <i>Set policy enforcement mode to Disabled to test policies on existing resources without getting the policy effect and logs</i> <i>Use Azure Arc to manage supported resources outside of Azure</i> <i>Show the compliance with the policies via a (near) real time dashboard (green-orange-red), at cascaded levels e.g. value-stream – Application – Configuration Item</i>
Stay in control with policy compliance	- Monitor 100% overall resource compliance - Perform periodic reviews of the remediation policy by collaborating during the remediation state Implement compliance state change events on key critical resources to react on these notifications - Monitor and manage policy creation to avoid unnecessary custom definitions and policy duplication
Automate your configuration with policies	Define and implement reactive policies to propagate configuration <i>Tips: Use "Deny" policies to avoid default deployment configuration and use "DeployIfNotExists" to automatically create the desired configuration</i>

Principle 3: Enable the organization with a cloud native foundation

Implementing and deploying cloud native technologies presents challenges to many organizations. Besides the transformation of legacy applications to cloud native technology, this also concerns the integration of various architectures and the need to use new innovative technologies as efficiently and compliantly as possible. Enterprise DevOps teams need to be enabled for this adoption.



Make it easy for teams to do it the right way with self-service and automation.

Putting up an ivory tower enterprise office will often not bring the enablement teams require. Enablement must come in a “fit for need strategy” with automation and self-service capabilities, with all the enablement baked in. A good example is the just-in-time learning discussed earlier, bringing knowledge and automation as smoothly as possible to the teams.

There are multiple areas where teams need enablement. The technical areas on cloud and DevOps and the organizational enablement on working model and responsibilities. The enablement for cloud comes in the form of platform capabilities with landing zones, together with an organizational capability that ensures knowledge is shared. The same for DevOps, enablement is in the form of DevOps platform, developer environment and an organizational capability to support teams on DevOps practices, reusable assets and way of working.

Bring the Cloud Center of Enablement (CCoE) to teams

Often the approach of a central team, under the name Cloud Competence Center (CCC) or a Cloud Center of Excellence (CCoE) is used. But all too often, this central team dictates ways of working 'as the experts' and becomes the single point of congestion. A central team that facilitates the other teams to build up knowledge and skills themselves as soon as possible is often much more successful in building knowledge and experience in the teams. The term “Cloud Center of Enablement” (CCoE) is preferable to the other examples; “enablement” expresses that the central team facilitates the other teams to build up knowledge and skills.

Accelerating the development and adoption of new knowledge and expertise while promoting standardization are common reasons for establishing a CCoE (or CCC). In practice, there appear to be many more underlying reasons. Some examples are:

- The interaction between the (DevOps) teams and the support roles from architecture, risk management, and compliance management does not run optimally.
- There is no or insufficient interaction aimed at recognizing and realizing reusable components, code and templates.
- Re-use of components, code and templates is difficult to start-up.

- There is much uncertainty about the application of guiding principles (reference architecture).
- There are complaints about guiding principles, but hardly any proposals for improvement.
- The recognition and adoption of new innovations is slow.
- The efficiency and quality of the team that manages the cloud foundation, respectively the cloud infrastructure, is disrupted because many ad hoc knowledge and expertise questions are also fired at them.
- There is a (still growing) diversity of redundant tooling.

The CCoE helps the organization to innovate, accelerate, maintain compliance and security, and focus on everyone who is—or will be—involved in further digitalization where cloud technology plays a role. This may also include, the managers, lawyers, buyers, risk managers, architects and IT service managers.

Based on the considered challenges and the defined “fit for need strategy”, the Enablement Center initiates the following activities:

- **InnerSource Library:** Initiators of InnerSource knowledge sharing on all aspects with everything as code conform the jointly created reference architecture.
- **Relevant new technology:** Actively follow innovations and needs of teams, and work with teams to get it embedded.
- **Training, coaching and mentoring:** Ensure that there are enough learning resources and onboarding programs.
- **Innovation sparring partner and hands-on assistance:** Partner externally with experts to get knowledge inside.
- **Quality and compliance sparring partner:** Bring policies to life and support teams to get compliant.
- **Reduce bottlenecks in guiding principles:** Take away the disruptive effect on compliance and productivity of teams of uncertainty about principles.
- **Enable community of practices:** Events, return of experiences, and practice sharing.

Set-up an Open Source Program Office

While a Cloud Center of Enablement supports teams on their journey to the cloud, an [Open Source Program Office](#) does the same for DevOps. Both are often combined, and it's wise to combine them.

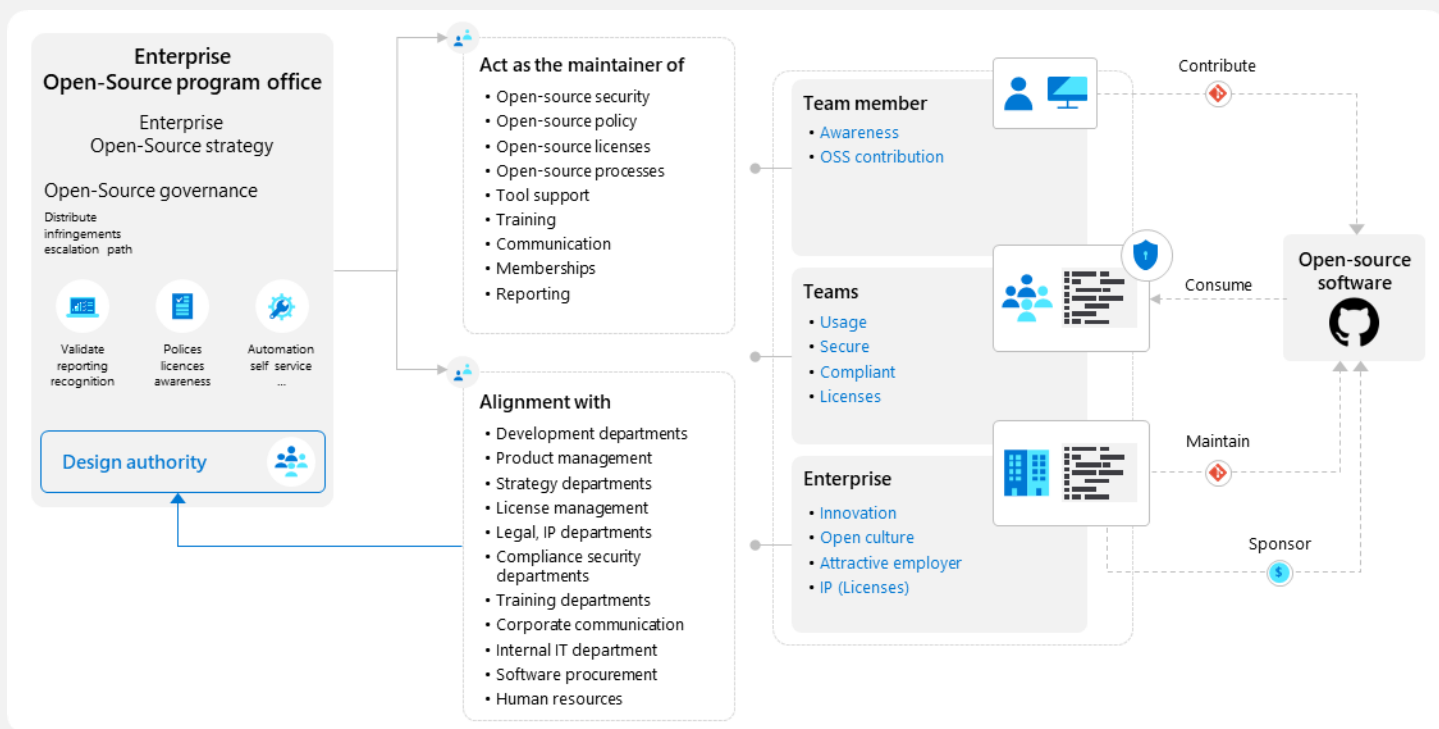
For large enterprises, their open source strategy must be supported by an open source program office. An open source program office supports, controls and executes the open source strategy of the enterprise. Due to the size and structure of large enterprises the set-up and governance of an enterprise open source program office has some specific focus areas to bring the value of open source to the whole organization without losing control and introducing risk.

The term open source refers to something people can modify and share because its design is publicly accessible. ... Open source projects, products, or initiatives embrace and celebrate principles of open exchange, collaborative participation, rapid prototyping, transparency, meritocracy, and community-oriented development.

[What is open source? | Opensource.com](https://opensource.com)

Open source software and open source practices bring a wealth of benefits to enterprises on many different levels. These benefits include:

- For **teams** the benefit will mainly be in the consumption of open source software, as the team doesn't have to build it from scratch and can use what has been made before. Open source has other specific advantages like, trust, openness, flexibility and many more.
- For **enterprises** the benefit is in the improved reputation as an employer and the sharing of knowledge an open culture brings to organizations. Enterprises can either maintain or sponsor open source communities with knowledge, time and funding.
- For **team members**, the benefit will be in the created value and the recognition they gain of the wider developer community. Above all they will be seen as a valuable member or even as a technology leader of the community.



However, along with these benefits, open source also presents many challenges enterprises need to handle and solve to gain the benefits whilst also mitigating risk. The scale, variety and dispersed knowledge across the organization are all causing specific enterprise challenges for the adoption of open source.

Therefore, enterprises must set up an Open Source Program Office to promote and bring awareness to all aspects of open source to the whole organization and take away the constraints and guides for the use of—and contribution to—open source software.

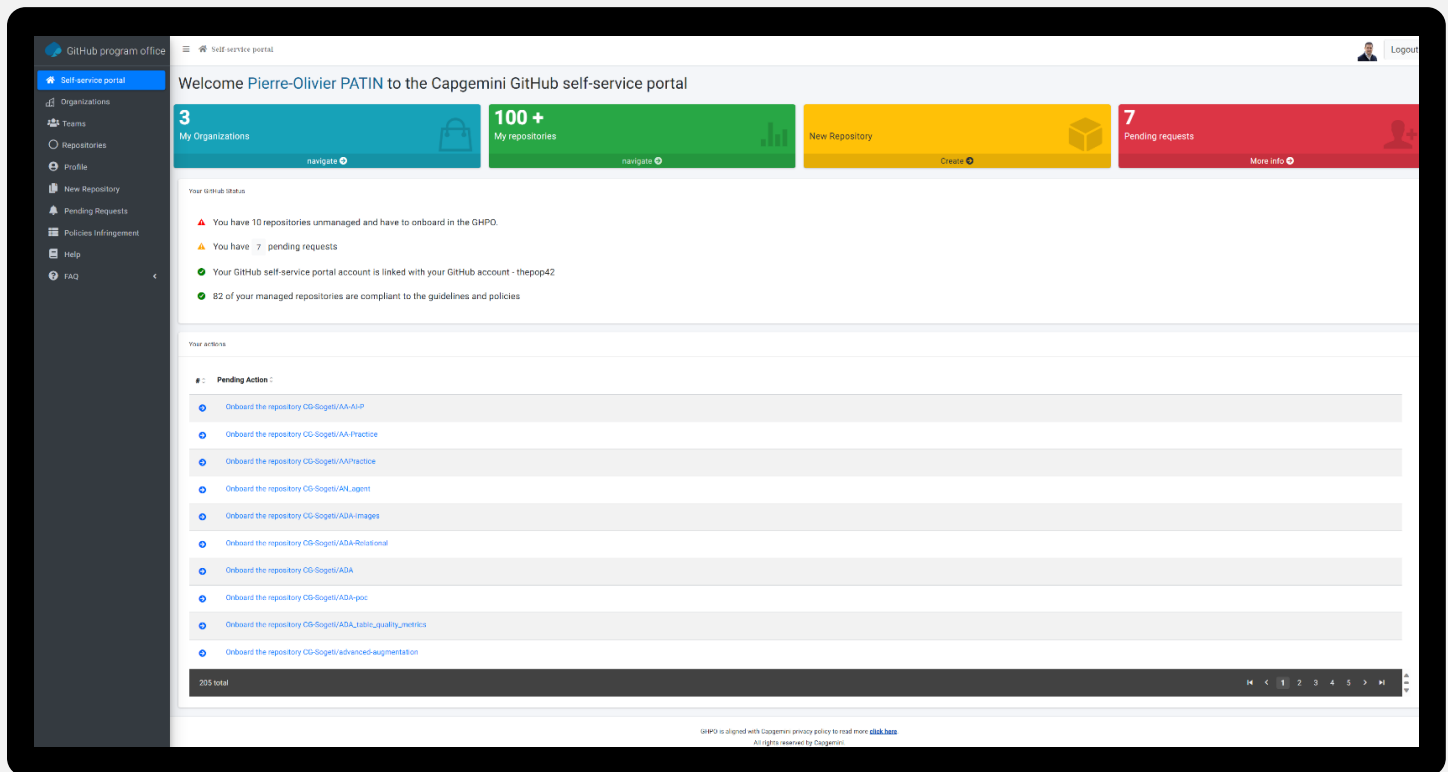
Recognition and motivation to open source contributions by employees need to be embedded in the organization to gain innovation power and to be an attractive employer in the industry.

Moreover, policies and validations need to be implemented to ensure that advantages of open source are fully realized and take care that all open source usages and contributions are secure and compliant with enterprise needs.

Automation, self-service and the principle of “it must be easy to do the right thing”, is mandatory for enterprises in the set-up and creation of an Open Source Program Office. Lacking automated validation of policies, recognition of contributions, support and awareness will discredit the adoption and makes running the Open Source Program Office inefficient and counterintuitive.

The governance on open source is driven by the users and contributors—the developers—alongside experts from every stakeholder team as a design authority for changes on open source policies.

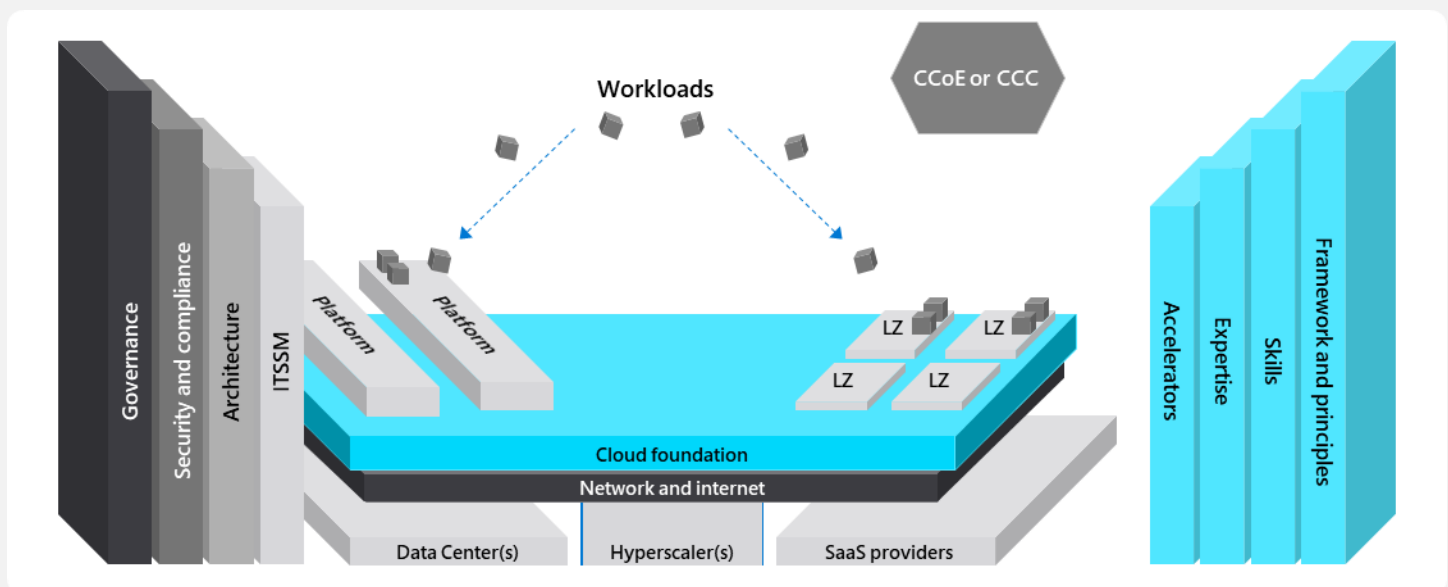
An Open Source Program Office drives the open source strategy of the enterprise. The Open Source Program Office acts as the maintainers of the strategy, policies, recognition, and automation. No single body, group or experts define the open source strategy on its own – it’s a collaborative process.



The Capgemini GitHub Program Office self-service portal

Landing zone(s)

In addition to knowledge and automation support via either a center or office, with cloud native the enablement support also comes from the platform itself.



Defining multiple platforms is growing in popularity. Platforms enable the organization to centralize capabilities and support (DevOps) teams to grow their self-service maturity. Using platforms strategically, all recurring processes are provided by the platform, the user, or the DevOps team and can fully focus on adding value.

We distinguish between technology platforms and business platforms. Business platforms are platforms established and managed by DevOps teams that perform business functionalities, such as managing communications with customers or handling transactions. Examples of business platforms are CRM-systems or e-commerce shops.

Technology platforms are driven by purpose on their technical capabilities like IoT, data, AI or integration platforms, defining a set of services composing this platform, with aligned roadmap and transversal features across these services like identity, monitoring, security, and administration.

With the explosive growth of self-service capabilities and Infrastructure-as-Code, we now have multiple technology platforms. It doesn't make sense to categorize them as a single, specific type of technology. Roughly, we recognize Technology Platforms as some of the following:

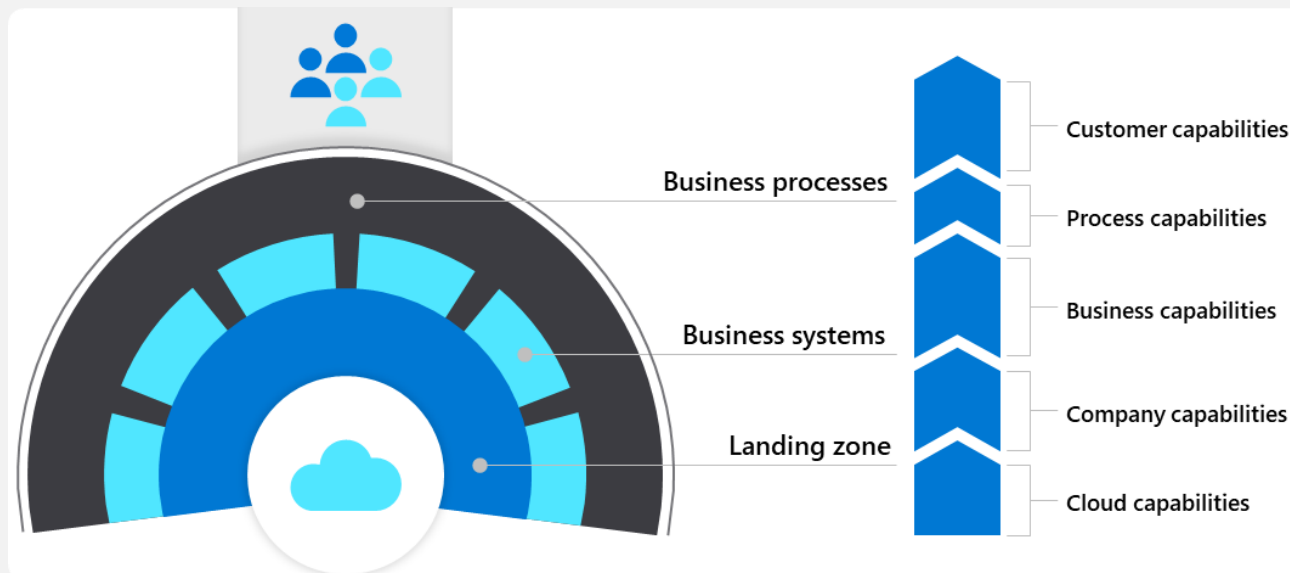
- **Cloud Foundation:** The centralized infrastructure layer in the organization where on-premises, hyperscalers and SaaS are connected and compliant with enterprise policies. All mandatory generic services are organized in this layer.
- **"Specific" Technology Platform:** A platform with a single functional focus. Like event streaming or data-storage, container-services, etc. The platform team will take care of the recurring tasks and will unburden the DevOps-teams so they can focus on adding value.
- **Low Code Platform:** A LowCode platform like PowerApps provides an organized set of services that will help the users to accelerate their innovation and agility.
- **Landing Zone:** A specific configured environment that easily scales via images or spokes. Depending the requirements and focus, a landing zone can be configured on the Cloud Foundation or at any specific technology Platform.

As a result, the key benefit is to not "reinvent the wheel" when a new need, project or product are required, but to leverage the business or technical platform to "stack" this new initiative in the condition of platform onboarding.

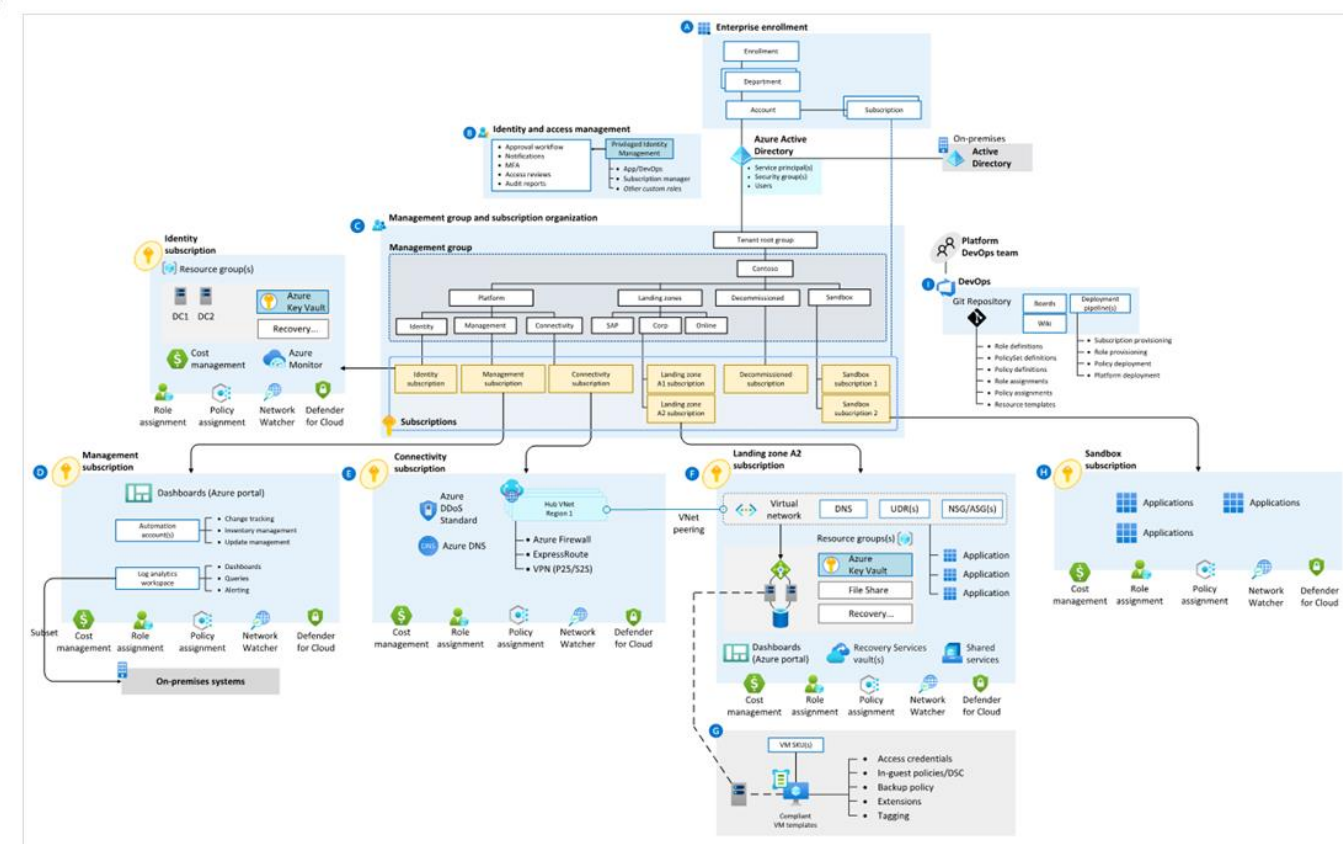
As an example, a new AI use case on an AI platform triggers the end-to-end configuration and deployment implementing one of the predefined patterns available for this platform. The team can focus on the specific activities related to its use cases without handling all prerequisites and implementation details.

Landing zones are a particularly well-known concept, widely used by organizations building their cloud capabilities. A landing zone is a company platform configured to conform to company standards and industry regulatory requirements. It is a base which enables business projects to follow security, compliance, and all other non-functional requirements.

Five Cloud Native adoption principles for enterprises



Since the very beginning of large-scale cloud adoptions more than 10 years ago, the notion of landing zones has evolved towards full enterprise scale landing zone designs like the ones that can be found on GitHub from Microsoft.



Learn more [about this diagram](#)

Today, the default is that these foundations are fully automated, like the Sogeti AKS landing zone. A new evolution is the enrichment of landing zones with DevOps capabilities and embedded security capabilities. Teams start next to the landing zone to automate the provisioning and configuration of DevOps tools and developer environments in the Spoke.

Hub and spoke pattern

While the term "hub and spoke" originally comes from a model of establishing network topology, a similar framework can be applied to landing zones. The hub is the central transit location with shared capabilities for the spokes, which can run different kinds of workloads. This framework can be applied to landing zones where a central hub has the capability to provide - preferably via self-service by Teams - multiple spokes for teams to work in.

Azure Deployment Environments

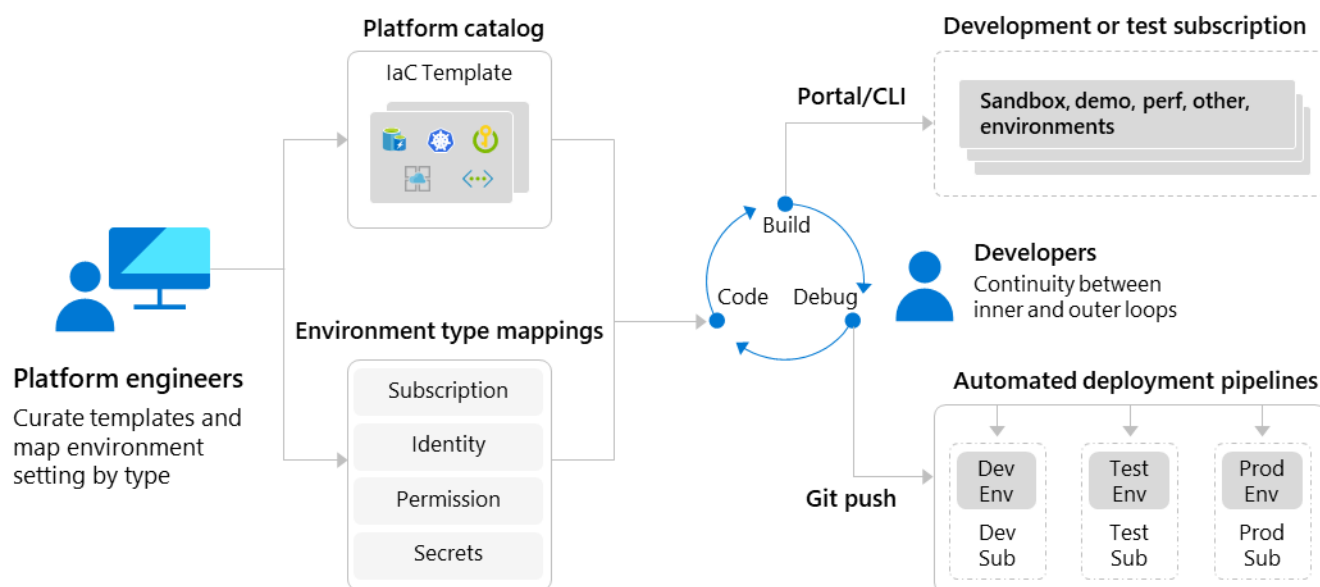
A great example is the provisioning of Spoke landing zones in Azure Deployment Environments (ADE). This service provides developers with self-service, project-based templates to deploy environments for any stage of development. It allows developers to self-serve app infrastructure within enterprise guardrails by providing just-in-time elevation or privilege using automated identity management.

With ADE, developers don't need to be granted blanket access to the subscription or resource group, but instead will be able to create environments by providing basic information that they are already familiar with – such as the project they're working on, type of environment they would like to create, and the template (SPOKE) in which they would like to create the environment. Behind the scenes ADE uses secured and isolated managed identities to perform the deployment on behalf of the user and within the enterprise policies configured for that type of environment.

With Azure Deployment Environments, Dev Infra Teams create Spokes called "environment templates" that developers can use to create the infrastructure needed to run their applications.

Azure deployment environments

Empowering developers through platform engineering



Learn more about Azure Deployment Environments at [Microsoft Learn](#)

Azure Developer CLI

Azure Developer CLI is another example of the automatic provisioning of environments (spokes) with additional configuration of CI/CD release pipelines, project configuration and more.

Azure Developer CLI (azd)

Building the fastest "code to cloud" path for developers

1

Select a template

2

Deploy and set up developer workflow

3

Commit code change and automatically deploy to app running on Azure



ToDo app with
Node.js API +
MongoDB

\$ azd up —t todo-nodejs-mongo
\$ azd pipeline config

\$ git push

\$ azd monitor --logs

update automatically

monitor application

Azure Developer CLI (azd) is an open source tool that accelerates the process of building cloud apps on Azure. It supports developers from project initialization to development with extensions in editors and IDEs, to CI/CD and production deployment. The CLI provides best practice, developer-friendly commands that map to key stages in your workflow, whether you're working in the terminal, your editor, integrated development environment (IDE), or DevOps.

```

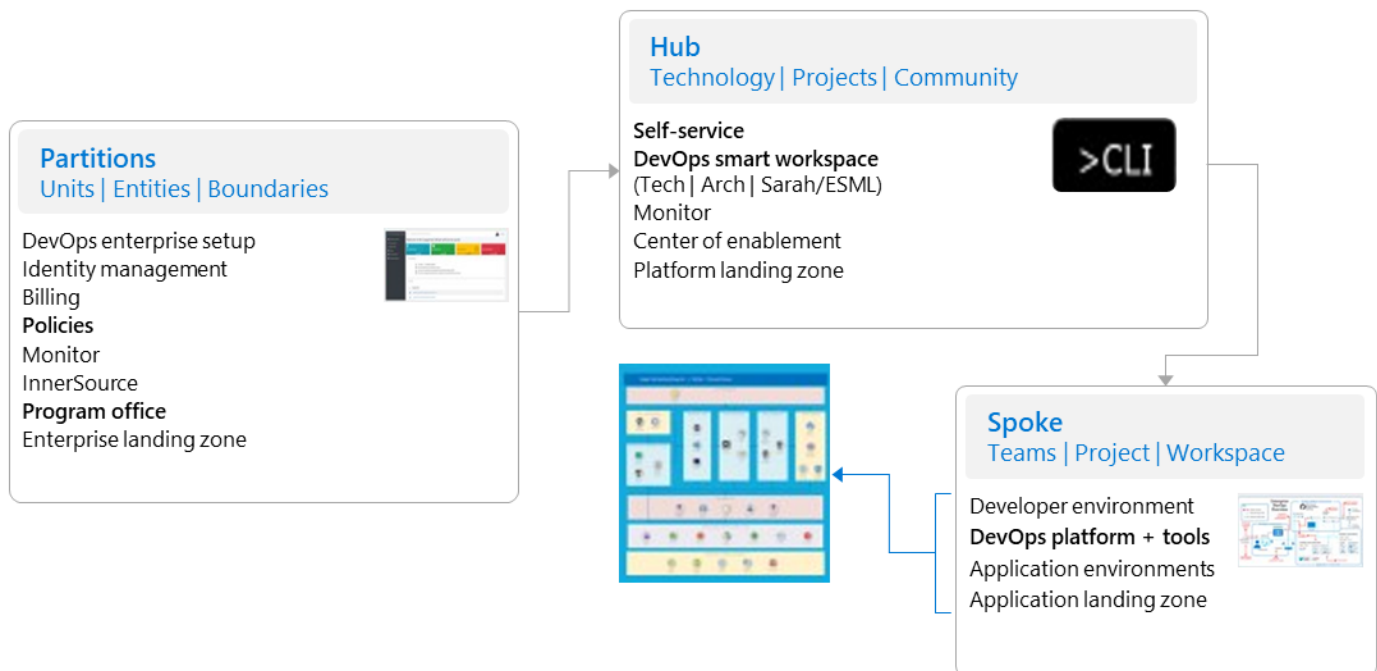
├── .devcontainer      [ For DevContainer ]
├── .github            [ Configure GitHub workflow ]
├── .vscode           [ VS Code workspace ]
├── assets            [ Assets used by README.MD ]
├── infra             [ Creates and configures Azure resources ]
│   ├── main.bicep    [ Main infrastructure file ]
│   ├── main.parameters.json [ Parameters file ]
│   ├── app           [ Recommended resources directory organized by functionality ]
│   └── core          [ Contains all of the Bicep modules used by the azd templates ]
├── src              [ Contains directories for the app code ]
└── azure.yaml        [ Describes the app and type of Azure resources ]
    
```

Learn more about Azure Developer CLI at [Microsoft Learn](https://learn.microsoft.com/en-us/azure/developer/azure-developer-cli/)

It is also fully customizable, allowing teams to create and share templates that adhere to the company's required standards and policies for integration with the landing zone spokes. The template can also contain customized CI/CD pipelines that can ensure that all required SAST / IaC and other scanning are included automatically.

This greatly reduces the work required for development teams to get up and running in creating value for the company, while reducing the DevOps team's cognitive load in relation to cloud infrastructure compliance.

Enterprise DevOps foundation



Cloud Native Application Protection Platforms (CNAPP)

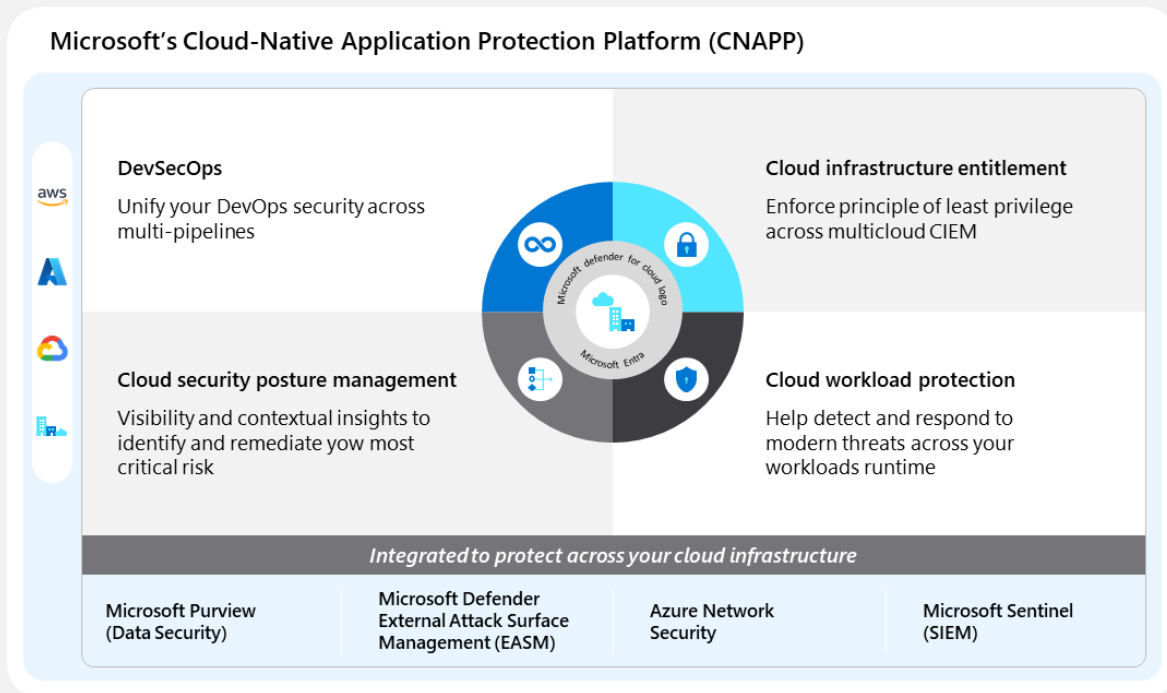
In the dynamic landscape of cloud native applications, where infrastructure is ephemeral, auto-scaling is the norm, and microservices architecture prevails, a specialized protection platform becomes crucial. Gartner brings an additional security focus to the landing zones and foundations described previously. This focus is defined in the term *cloud native application protection platform* (CNAPP). CNAPP helps with the security of cloud native systems to ensure visibility. Visibility is a major obstacle in cloud native security, particularly in large, fragmented environments where teams struggle to track data flow across multiple applications. As cloud native systems expand, there is a growing need for a comprehensive and transparent solution to address this challenge. A Cloud Native Application Protection Platform serves as a dedicated defense line, addressing the unique security challenges posed by these applications. As traditional security tools often fall short in these dynamic environments, a tailored protection platform offers container and orchestration-aware security mechanisms. It ensures the integrity of application images, monitors runtime behavior, manages identity and access, and facilitates compliance adherence. With the increasing adoption of cloud native practices, safeguarding these applications with a purpose-built platform is not only a security imperative but also a strategic move to enable innovation while minimizing risks.

CNAPP represents an approach to safeguard cloud environments by integrating several risk protection measures, including DevSecOps, Cloud Security Posture Management (CSPM), Cloud Workload Protection Platform (CWPP), and Cloud Infrastructure Entitlement Management (CIEM), into a single comprehensive platform. It supports not only the operational run of cloud workloads with these integrations, but also the enterprise teams with security testing tools and practices, all integrated.

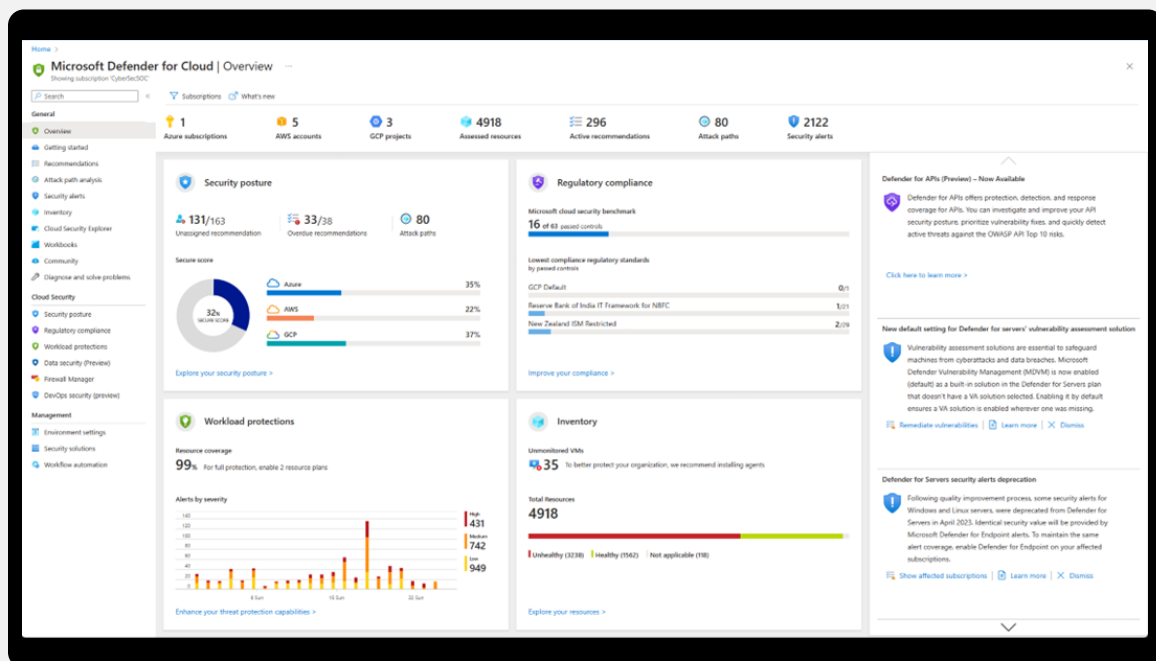
CNAPPs are the leading edge of cloud security. A CNAPP unifies security and compliance capabilities to prevent, detect, and respond to modern cloud security threats from development to runtime.

A CNAPP delivers a unified experience for organizations that synthesizes insights and drives effective collaboration among developers, DevOps teams, security teams, and security operations center (SOC) analysts to reduce excessive risks for cloud-native applications and to embed security across the continuous integration and continuous delivery (CI/CD) lifecycle.

[Microsoft Defender For Cloud Provides CNAPP Security | Microsoft Security blog](#)



Microsoft provides a single and unified platform to protect cloud native applications, called Microsoft Defender for Cloud. Microsoft Defender for Cloud is a comprehensive cloud security solution designed to safeguard cloud native applications against evolving cyber threats. Built upon cutting-edge technologies and years of cybersecurity expertise, it offers a holistic approach to threat detection, prevention, and response within cloud environments. By leveraging advanced AI and machine learning capabilities, Defender for Cloud continuously monitors activities, detects anomalies, and identifies potential security breaches across cloud native applications, containers, and serverless architectures. It provides real-time visibility into the security posture of these applications to proactively identify vulnerabilities and mitigate risks.



Principle 3 OKRs

Objectives	Key results
Federate—through self-service—an extended organization and teams around practices, knowledge sharing and delivery excellence	• Define a federate organization between the Center of Enablement and the different entities and teams • Measure cloud adoption as a percentage ratio of resources over time • Measure the organization's cloud service catalog adoption over time • Measure onboarding time of new cloud services in the organization • Measure the time to market to deploy new products involving cloud services • Measure people skills development following the number of trainings, certifications, and technical talks over time
Setup an InnerSource program	• Measure 3 to 5 key engagement metrics (number of products or initiatives onboarded, issues, pull/merge requests, users involved) • Measure 3 to 5 key community metrics (internal meetups, dedicated workshops, public exposition through events and knowledge sharing, etc.) • Evaluate the qualitative impact on productivity, time to market and innovation
Automate cloud platform deployments and CI/CD pipelines to start in control	• Landing zones are deployed as code and consistent across multiple regions and entities. • Landing zones comply - at deployment and upgrade - with the set of good practices defined according to your context <i>Tips:</i> <i>Leverage the Hub and Spoke pattern applied to landing zones</i> <i>Leverage policies to ensure perpetual compliance checks after deployment</i> <i>Understand limits of current infrastructure configurations</i> <i>Leverage Azure Advisor to explore contextual recommendations</i>
Enhance Cloud Security and Compliance	• Implement a cloud-native security framework, conducting a thorough security assessment and remediation for all cloud-native applications • Automate security monitoring and incident response for cloud-native environments • Achieve compliance with relevant industry standards (e.g., CIS benchmarks) for all cloud-native workloads
Promote modern developer environments	• Define the developer environment as part of your product architecture including developer capability requirements • Measure 3 to 5 key metrics on developer experience (developer onboarding time, ease of connectivity/integration, developer satisfaction, developer environment security and compliance)

Principle 4: Consistent, secure, and compliant from start to finish

When policies are defined and foundations are built, the next challenge is how to make sure all DevOps teams adopt them and that the outcome of their work—the resulting business functionality—starts in compliance and stays in compliance.

From a business and velocity standpoint, teams need to have sufficient autonomy to be healthy and productive. Forcefully strongarming tools and solutions upon teams can end up creating a negative impact on team performance.

However, it's challenging to manage all dependencies while tracking and responding to reported vulnerabilities without impacting business. Moreover, it requires active DevOps teams and close collaboration between DevOps, security, and enablement teams.

With many 'self-organizing' DevSecOps teams all working on business functionality, teams also need to have a consistent way of working, a common way of implementing and running all of these hundreds of pieces of business functionality. For any cloud native implementation to be successful from a compliance and security perspective, a consistent way of working – from start to finish – is critical.

Having clear, standardized reporting OKRs and a paved supported path with tool support to make doing the right thing easy will lead to a consistent and auditable way of working. This approach makes proving compliance and diminishing exceptions easier, which also results in a smaller attack surface.

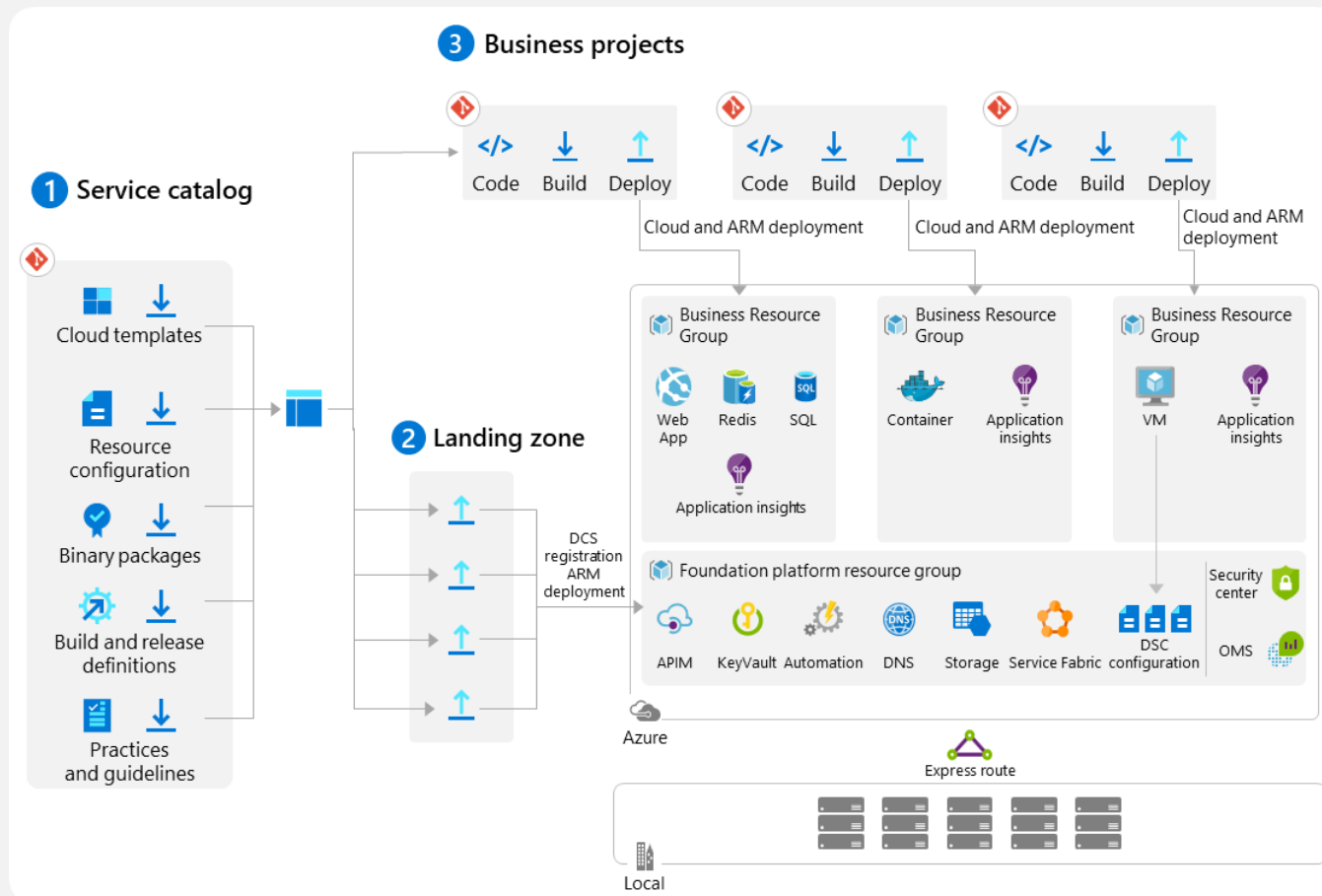


Start small, involve, and enable all and learn fast with continuous monitoring to start efficient and stay efficient.

Infrastructure as Code: Consistent and compliant

With an InnerSource library of validated and standardized infrastructure building blocks, teams can accelerate and focus on the business functionality while knowing they are secure and compliant.

In GitHub, Sogeti maintains a library of templates that conforms to the standards and is validated against the [Well Architected Framework](#).



S04 Connectivity

Resource	Description	Compliant
Azure LoadBalancer	Loadbalancer	
Network Security Group	A network security group (NSG) includes rules that allow or deny traffic to a virtual network subnet, network interface, or both.	
OneMigrate Vnet	Virtual Network configuration for the OneMigrate deployment	
Virtual Network with 7 subnets	Virtual Network with 7 subnets.	

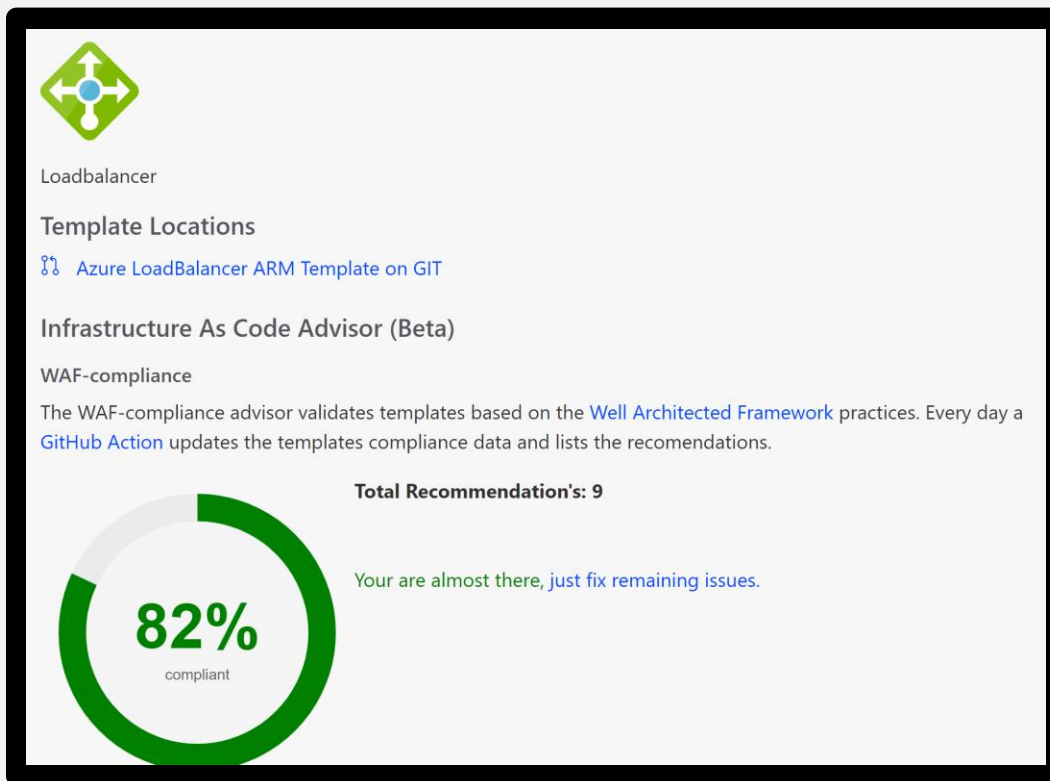
S05 Security

Resource	Description	Compliant
Azure Application Gateway	Azure Application Gateway	
KeyVault	Creates a key vault for the storage of secrets, keys and certificates	
KeyVault update secret	Update or add a secret to an existing Azure KeyVault. Used during release when a resource is created which exposes a secret.	

S06 Integration

Resource	Description	Compliant
API App Service	Azure API App deployment in an existing Azure App Service Plan.	
Azure API Management	This set of templates helps automating the provisioning of Azure API Management.	
Azure Container Registry	Azure Container Registry, a private registry for hosting container images. Store Docker-formatted images for all types of container deployments.	
Azure Front Door	Azure Front Door is a global, scalable entry-point that uses the Microsoft global edge network to create fast, secure, and widely scalable web applications	

The results are visual in the documentation pages of the corresponding resources. This takes care of configuration drift and evolving cloud platform (new API versions) and evolving practices are always visible for the teams.



Context-based and secure cloud native building blocks

The concept of building blocks is quite popular and in most companies informs the core activities of architecture and design: instilling component selection, reusability, integration, and consistency.

In the age of cloud, the same principles apply and implementing these building blocks break down to the following steps:

1. **Select:** With the significant list of cloud services and flavors, organizations need to define the main services they would like to use. It's an iterative approach with new needs or concerns bringing new services over time.
2. **Integrate and Secure:** Predefined according to security requirements
 - a. How will the service be deployed from a network standpoint?
For example, PaaS deployment with public access, in a virtual network, using private link and private endpoints and/or using service endpoints.
 - b. How will the service be integrated from identity standpoint?
For example, Managed identities, Service principals, keys and certs
3. **Customize:** Define the service tiers, Azure region, tags, monitoring integration, etc.

These building blocks are validated at the enterprise level and new applications will be designed using these building blocks. This ensures they meet the nonfunctional requirements and avoid repetitive specifications of these services in the design of the application itself.

Regarding Zero Trust principles, most customers are still looking for a “Two-factor security approach”, requiring two different security layers to ensure authentication/authorization with identities and access management with network isolation.

Once a solid foundation of building block components has been established, organizations tend to define a concept of platform, gathering services and capabilities around the same purpose. Examples include compute platforms using VMs and containers or integration platforms using messaging and APIs. Ultimately this helps to foster expertise, evolution, and support on these fields.

The Hero Pipeline

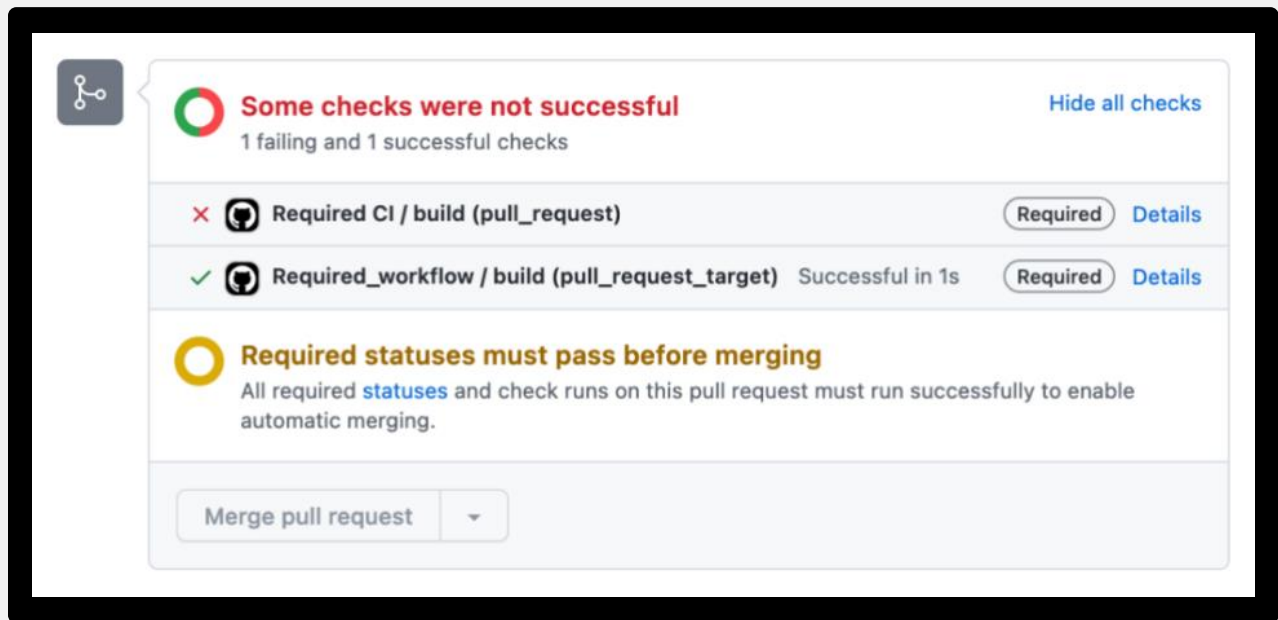
Adopting an everything-as-code approach helps your enterprise’s deployment reliability, version control, and testing and security effectiveness. An essential advantage of pipelines-as-code is the reduced risk of human error, as workflows are defined as code. This eliminates the possibility of an engineer forgetting a step or clicking the wrong button by mistake when following a manual checklist.

In a pipeline-as-code approach, you store all the CI/CD pipelines inside the version control system as files—enabling tighter version control for security reviews. Successful adoption of pipeline-as-code in your application pipeline hardens security at every stage of deployment. For example, now you can scan your holistic application code using Static (SAST), Dynamic (DAST) and Interactive (IAST) Application Security Testing methods. You can also integrate Runtime Application Self-Protection (RASP) inside servers to further protect applications.

When multiple teams work together on your cloud native platform, automated CI/CD pipelines are already an important practice. Not only for the standard checks on integration, but for the compliance of your policy too.

With reusable workflows you can provide a lot of value to your teams in providing them the approved building blocks for these pipelines. By standardizing your pipeline, it will already be easier for your teams to improve their quality and policy compliance.

While this is great, this might not ensure all these workflows are used by your teams. In enterprise scenarios, where teams could sprawl the repositories, you might want to add a layer of control on your pipelines. With GitHub Actions, workflows can be flagged as “required” in a GitHub organization. This will automatically enforce the execution of your workflow on all repositories in that organization.



[Required workflows – GitHub Docs: Currently in public beta](#)

Now we can take the standardization one step further and define a hero pipeline. By combining all the security, compliance and deployment best-practices into a single required pipeline, enterprises can reduce duplication of CI/CD configuration code.

Currently, there are some prerequisites and restrictions to consider:

- For required workflows to run, the pull requests source repository must be in the same organization as the target repository.
- Secrets used in a required workflow should be created at either the organization level or in the target repositories.
- CodeQL is not supported in required workflow.

Secrets to be used in your hero pipeline can be stored per organization or per (target) repository. To allow certain customizations, the new feature for "Configuration variables" (in Beta) can be powerful for enabling some control to the teams in execution of your pipeline.

Next to that, with pipelines as code teams can monitor and report on the consistency of their deployment process. GitHub has the capability to report on the pipelines that are being used via the GitHub REST API, while the capability to share pipeline building blocks supports teams to provision services in a consistent manner.

Scaling numerous teams and microservices

To get the agility and flexibility benefits of microservices and serverless architecture it is important to pay attention to the way they are built, deployed, and integrated.

Five Cloud Native adoption principles for enterprises

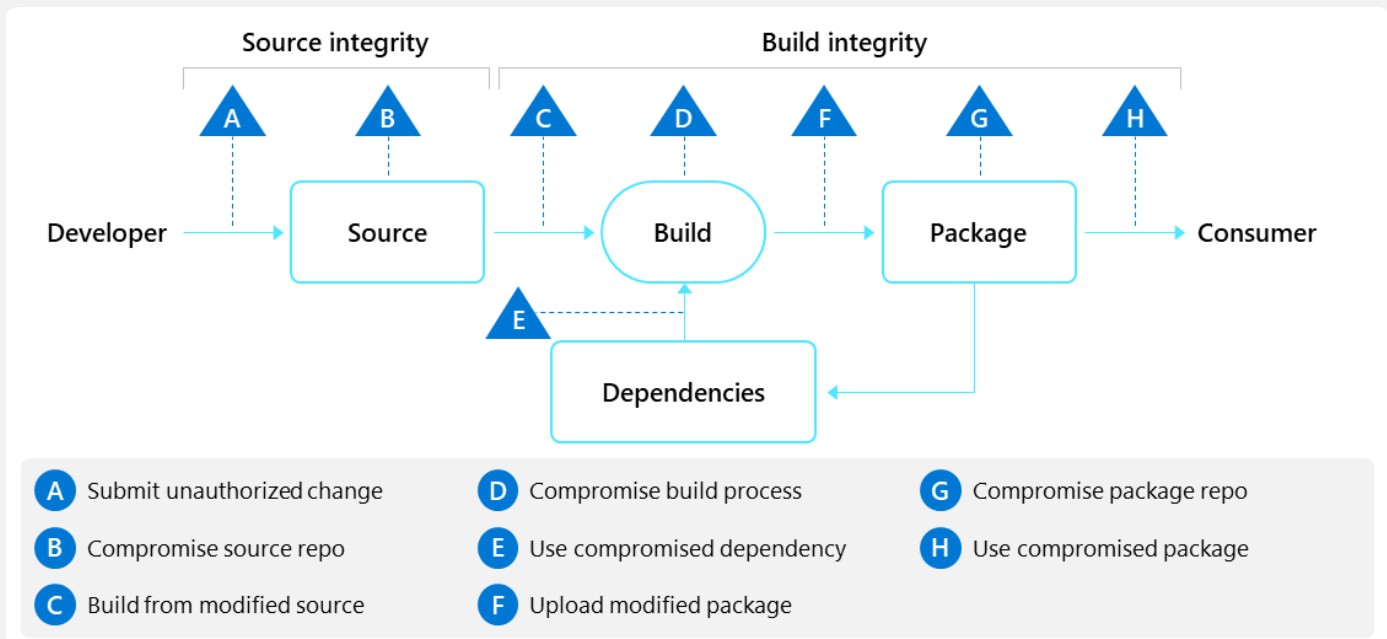
The first area to evaluate is scale. There will be many CI/CD pipelines for one application. Consistency, reusability, and versioning of pipelines with as-code principles is important.

Secondly, microservices and serverless components will evolve at a different pace, so it is required to have insights in the execution state of the pipelines. Dashboards, triggers, and notifications are all required to gain an understanding on which version is deployed, when, and where.

A third focus area is responsibility. Feature teams are responsible for one product or service. When a team is dismantled, a service can become an orphan. In today's ever-evolving landscape, an orphan service quickly falls behind and becomes a security liability. This underscores the importance of developing towards DevSecOps models to bake security into processes across teams. Some may anticipate evolving to DevSecOps to be tumultuous in its process changes. But when compared to the technical debt of later refactoring an older DevOps model versus a DevSecOps model (especially in the aspects of microservices), it's clearly more beneficial to begin this evolution as soon as possible.

The process to deliver software and infrastructure must be challenged to enforce security on all stages of the process. SLSA, as part of the CNCF, is a framework to assist in strengthening security to prevent the addition of malicious code (e.g. man in the middle). This framework accompanies teams for the deployment of the software factory and validates everything during build and delivery processes.

SLSA stands for Supply Chain Levels for Software Artifacts. SLSA is a security framework with standards and controls for assessing the security and integrity of software. Software development organizations can use the SLSA best practices and standards to implement and ensure that the software they deliver is secure and trustworthy.

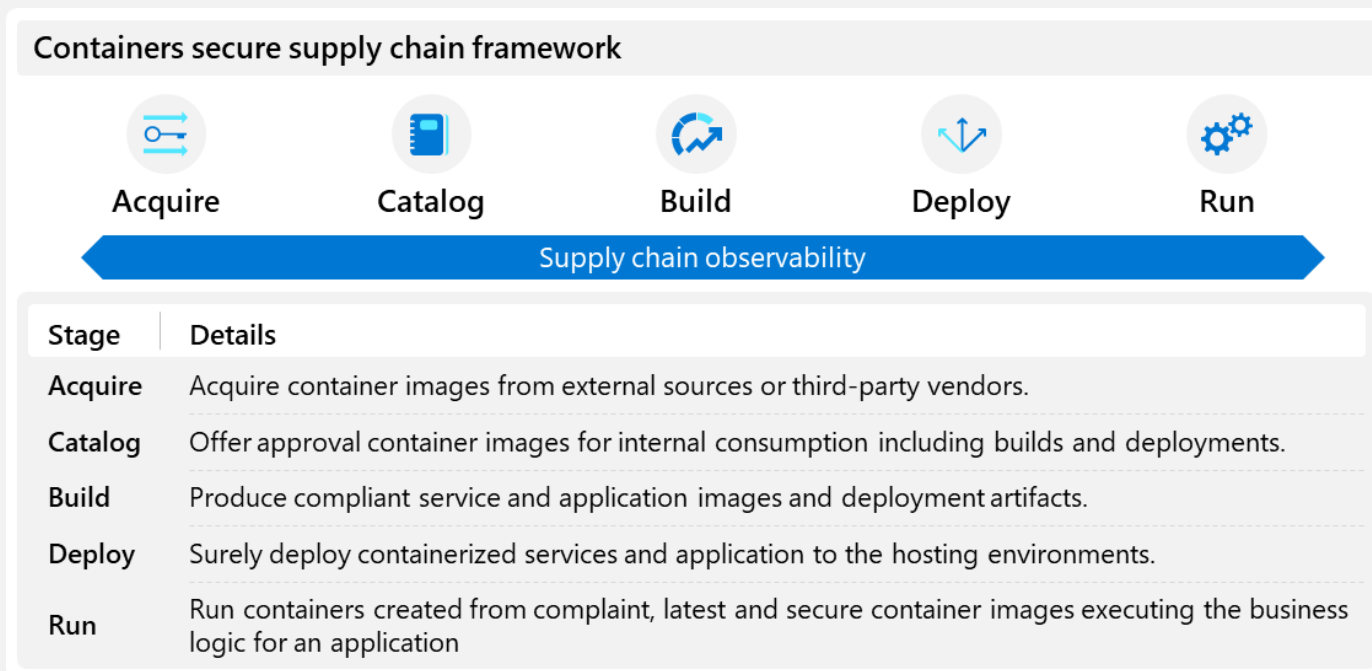


The supply chain problem from <https://slsa.dev/>

Five Cloud Native adoption principles for enterprises

This is one of the practices which ensures teams are building reliable, secure microservices, each service must be continuously tested and checked with high standards that conform with the SLSA. Automation and continuous validation of quality and compliance is mandatory for microservices. Each service, the platform it runs on, and what it is built with must have the zero trust security principle implemented and validated.

When following a service build, deployment, and run flow, the following practices must be followed:



Learn more about the Containers Secure Supply Chain Framework at [Microsoft Learn](#)

The base container is an important focus for teams. The base container images and their contents (vendor or OSS) must come from a trusted source. Especially when building hundreds of services by different teams, automation is key in this process. [SLSA provides GitHub Actions](#) that provide the proof or origin of artifacts used during the build.

Teams often utilize ACR Image Management, which includes integrated security and network integration, as a secure foundation for their base images and other OCI artifacts such as Helm charts. The origin of these artifacts is protected through signing and verifying using tools like Notary. Additionally, Microsoft Defender for Containers ensures that malware and vulnerabilities are scanned and reported to Microsoft Defender for Cloud. This ensures that all parties involved have access to a centralized source of information about the state of the base images.

Container registry images should have vulnerability findings resolved

Exempt Disable rule View policy definition Open query

Unhealthy registries 20 / 24 Severity High Total vulnerabilities 360 Vulnerabilities by severity High 48 Medium 311 Low 1 Registries with most vulnerabilities dmt 232 ima 120 ascd 93 Total vulnerable image: 88 Out of 318 scanned

Description Remediation steps Affected resources Security Checks Findings Disabled findings

Search to filter items...

ID	Security Check	Category	Applies To	Severity	Patch Available
372268	GNU Bash Privilege Escalation Vulnerability for Debian	Local	29 of 318 Scanned Images	High	No
178391	Debian Security Update Multiple Vulnerabilities for perl	Debian	13 of 318 Scanned Images	High	Yes
178369	Debian Security Update for tzdata (DLA 2424-1)	Debian	12 of 318 Scanned Images	High	Yes
176875	Debian Security Update for systemd	Debian	7 of 318 Scanned Images	High	Yes
177442	Debian Security Update for file (DSA 4550-1)	Debian	6 of 318 Scanned Images	High	Yes
176750	Debian Security Update for apache2 (DSA 4422-1)	Debian	6 of 318 Scanned Images	High	Yes
178486	Debian Security Update for Open Secure Sockets Layer (OpenSSL) (...)	Debian	5 of 318 Scanned Images	High	Yes
374644	Go XML attribute instability Vulnerability	Local	5 of 318 Scanned Images	High	No
178701	Debian Security Update for apache2 (DLA 2706-1)	Debian	4 of 318 Scanned Images	High	Yes
176853	Debian Security Update for libssh2 (DSA 4431-1)	Debian	4 of 318 Scanned Images	High	Yes

Trigger logic app Exempt

78391-Debian Security Update M...

Description

Perl is a family of two high-level, general-purpose, interpreted, dynamic programming languages. Perl is found to be affected by Heap based buffer overflow and integer overflow vulnerability.

Affected OS: Debian 9 Debian 10

General information

ID 178391 Severity High Type Vulnerability Published 2/2/2021, 3:47 PM GMT+2 Patchable Yes Cvs3.0 base score 8.6 CVEs CVE-2020-10543 CVE-2020-10878 CVE-2020-12723

Remediation

The Customers are advised to update Perl here

Patch: Following are links for downloading patches to fix the vulnerabilities: Debian 10 Debian 9

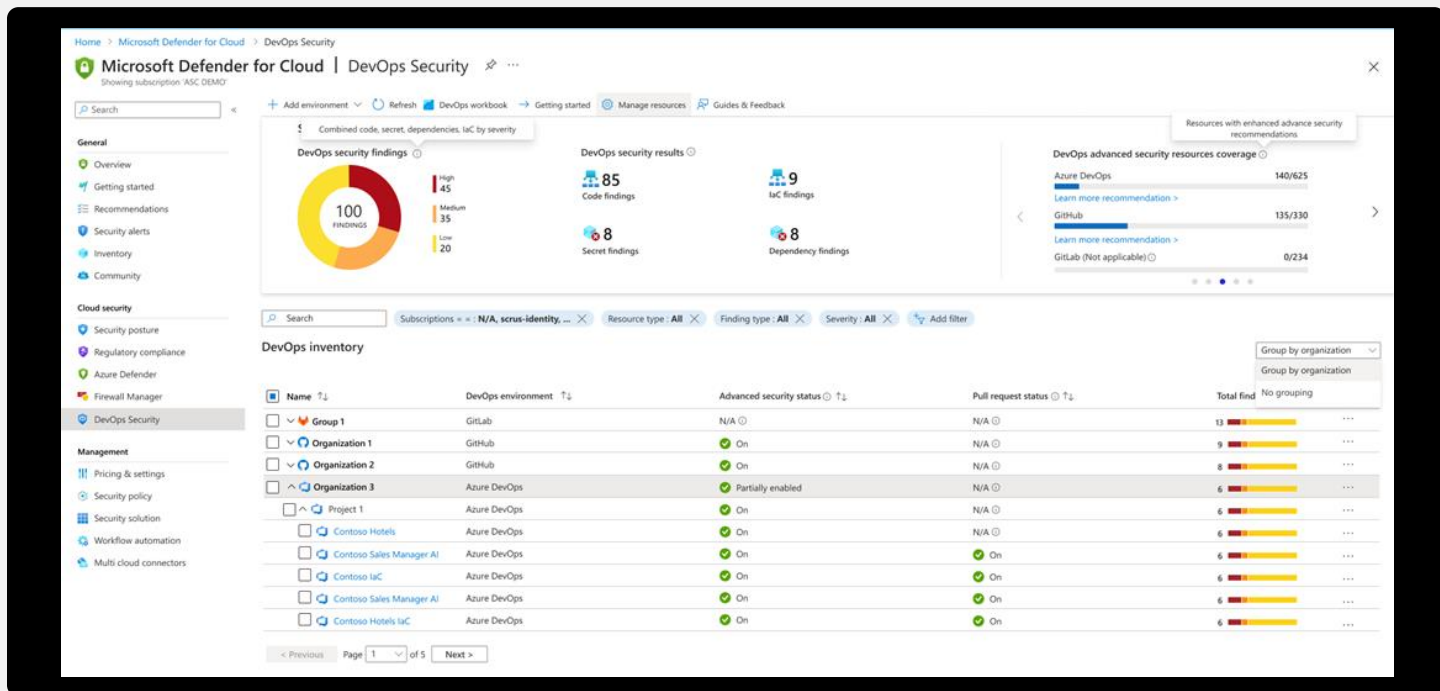
Additional information

Vendor references CVE-2020-10543 CVE-2020-10878 CVE-2020-12723

Affected resources

Digest	Repository	Registry
53e2	netcore	ch1gea
7f29	node	ch1gea

Teams build on top of the secure verified base image and apply their secure development practices. Capabilities of the platforms must be used in the optimal way, like GitHub Advanced Security, and GitHub Advanced Security for Azure DevOps, to check on either supply chain vulnerabilities or code issues. Validations can be set to check if teams enable and use these capabilities. Reporting out to Microsoft Defender for Cloud gives insights to the security teams on the state of the pipelines and any potential vulnerabilities. This ensures that teams don't operate in isolation and that all run confirmed policies that are consistent.



The supply chain practices from SLSA flows through this stage, and teams need to continuously validate and prove that the components where the software is made is compliant. Tools like [Ratify](#) supports validating this chain by acting as an admission controller for containers being deployed to a cluster.

Responding to threats and vulnerabilities

During the run phase, the platform must continuously scan for vulnerabilities in deployed images or software. If a vulnerability is reported, the platform needs to match that vulnerability to the affected system and a security alert needs to be created and handled by the security team for initial triaging, prioritization, and mitigation. Once the initial triaging has been done, the security team needs to collaborate with the owning DevOps team to ascertain the right action and plan to handle the reported vulnerability. Principle 3 discusses CNAPP capabilities which can handle these kinds of actions.

The only way to stay both secure and operational is to have an active DevOps team maintaining the system and keeping all components updated according to the lifecycle policy. This minimizes both the risk of security vulnerabilities and the time to remediate.

An active DevOps team with modern tooling like Dependabot can easily keep abreast of updates and stay secure and compliant with little effort and complete peace of mind knowing all is secure.

Developer portals and scorecards

An enterprise likely has many DevOps teams which are working on multiple cloud native services. The likelihood is that these teams rely on each other in multiple aspects. From a knowledge and experience

Five Cloud Native adoption principles for enterprises

perspective, it is important that they work in the same way so they share with fellow developers and build on their combined, collaborative knowledge and experience.

Efficiency is optimized when every DevOps team follows the same practices, and uses the same tools to make sure they start and stay compliant with organizational requirements.



A good example of automated validations of DevOps teams practices is the OpenSSF Scorecard implementation for open source projects. Based on a set of [checks](#), it validates the public GitHub repository of a team and scores them if they follow the [security practices](#).

Scorecard is an automated tool that assesses a number of important heuristics ("checks") associated with software security and assigns each check a score of 0-10. You can use these scores to understand specific areas to improve, in order to strengthen the security posture of your project. <https://github.com/ossf/scorecard>

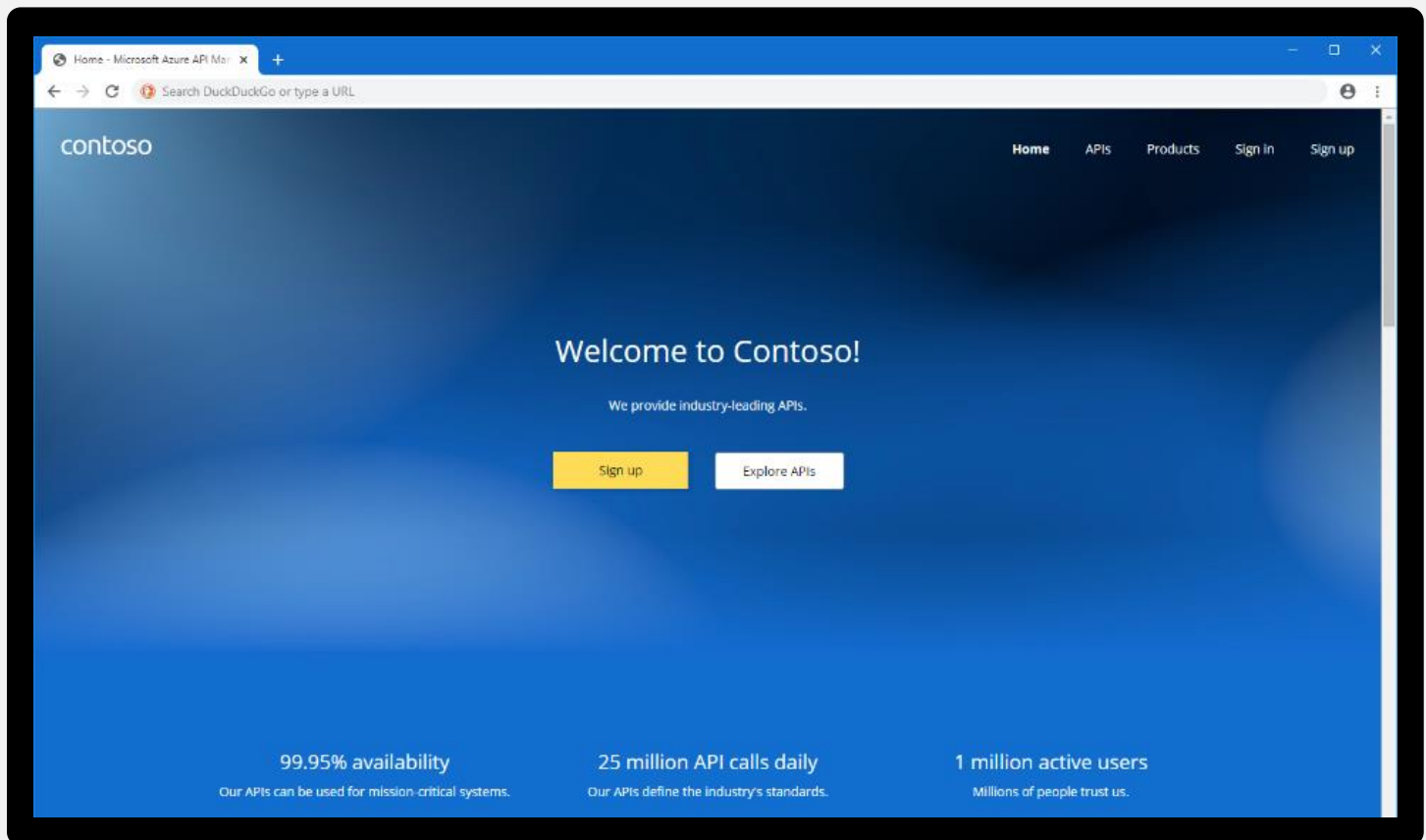
Here's an example scorecard output with remediation advice:

RESULTS			

Aggregate score: 7.9 / 10			
Check scores:			
SCORE	NAME	REASON	DOCUMENTATION/REMEDIATION
10 / 10	Binary-Artifacts	no binaries found in the repo	github.com/ossf/scorecard/blob/main/docs/checks.md#binary-artifacts
9 / 10	Branch-Protection	branch protection is not maximal on development and all release branches	github.com/ossf/scorecard/blob/main/docs/checks.md#branch-protection
?	CI-Tests	no pull request found	github.com/ossf/scorecard/blob/main/docs/checks.md#ci-tests
0 / 10	CII-Best-Practices	no badge found	github.com/ossf/scorecard/blob/main/docs/checks.md#cii-best-practices
10 / 10	Code-Review	branch protection for default branch is enabled	github.com/ossf/scorecard/blob/main/docs/checks.md#code-review
0 / 10	Contributors	0 different companies found -- score normalized to 0	github.com/ossf/scorecard/blob/main/docs/checks.md#contributors
0 / 10	Dependency-Update-Tool	no update tool detected	github.com/ossf/scorecard/blob/main/docs/checks.md#dependency-update-tool
0 / 10	Fuzzing	project is not fuzzed in OSS-Fuzz	github.com/ossf/scorecard/blob/main/docs/checks.md#fuzzing
1 / 10	Maintained	2 commit(s) found in the last 90 days -- score normalized to 1	github.com/ossf/scorecard/blob/main/docs/checks.md#maintained
?	Packaging	no published package detected	github.com/ossf/scorecard/blob/main/docs/checks.md#packaging
8 / 10	Pinned-Dependencies	unpinned dependencies detected -- score normalized to 8	github.com/ossf/scorecard/blob/main/docs/checks.md#pinned-dependencies
0 / 10	SAST	no SAST tool detected	github.com/ossf/scorecard/blob/main/docs/checks.md#sast
0 / 10	Security-Policy	security policy file not detected	github.com/ossf/scorecard/blob/main/docs/checks.md#security-policy
?	Signed-Releases	no releases found	github.com/ossf/scorecard/blob/main/docs/checks.md#signed-releases
10 / 10	Token-Permissions	tokens are read-only in GitHub workflows	github.com/ossf/scorecard/blob/main/docs/checks.md#token-permissions
10 / 10	Vulnerabilities	no vulnerabilities detected	github.com/ossf/scorecard/blob/main/docs/checks.md#vulnerabilities

The developer portal Backstage.io is another example of a tool that supports DevOps teams to work in a consistent way. Backstage.io centralizes and standardizes the developer tooling to streamline the environments for teams. Via a self-service portal, it gives an overview of the services available for teams.

Regarding microservices and APIs of applications, one component of Azure API Management that enables organizations to make their APIs easily discoverable and consumable by developers is the developer portal. APIs are exposed as products with sample code. APIs definitions in the developer portal are automatically sync from the Azure API Management services and the related documentation automatically generated.



Learn more about the developer portal at [Microsoft Learn](#)

InnerSource

Organizations that use InnerSource experience benefits that are typical of open source development, such as:

- **High-quality code:** With unit tests, code coverage, and continuous integration, teams improve code earlier in the lifecycle.
- **Comprehensive documentation:** Code is better documented, both in comments and less formally in discussions, leading to a single source of truth.
- **Effective code reuse:** Code and architecture are discoverable and available across teams and the organization.
- **Strong collaboration:** Code reviews have less friction, team communication becomes stronger, and contributions increase in number.

- **Healthy culture:** Silos are broken down, so developer happiness improves, leading to better retainment and recruitment.

InnerSource works to grow awareness and bring knowledge to communities.

When starting an InnerSource community it is recommended to follow the same practices as open source communities, see <https://opensource.guide/> for guidance. It's also good to set up the Open Source Program Office as discussed in Principle 3.

Practices learned from helping large organizations to leverage knowledge sharing and standardization drive to be more efficient are:

- There must be management support - it doesn't come for free.
- Workgroups and communities can be used to give specific organizational focus.
- Use training and skilling workshops for individual learning
- Measure and act on what is effective (and not effective) and report on it.
- Have a dedicated group of maintainers and use the champion model for security.
- Stay as close to the teams with the technology.
- Have the best developer documentation.
- Recognize and reward contributions.
- Ensure consistency and compliance with policies.
- Track and measure policy compliance.

The products that are maintained in InnerSource can be many things but should aim to practice staying close to the team. The focus must be on team assets like:

- **Infrastructure as code:** The smallest cloud building blocks.
- **Pipelines as code:** Reusable pipeline templates.
- **Policies as code:** For the platform, automation, and method of working, documentation is key for this.
- **Projects as code:** Templates for default tasks.
- **Language frameworks and other components:** Be careful not to reinvent the wheel.
- **Landing zone automation with DevOps tool configuration:** For self-service as discussed in the previous principle.

One guiding principle on InnerSource is not to reinvent the wheel, use what is exists from vendors or open source communities and build on top of it. When an InnerSource product requires too much effort, it's normally quick to stall.

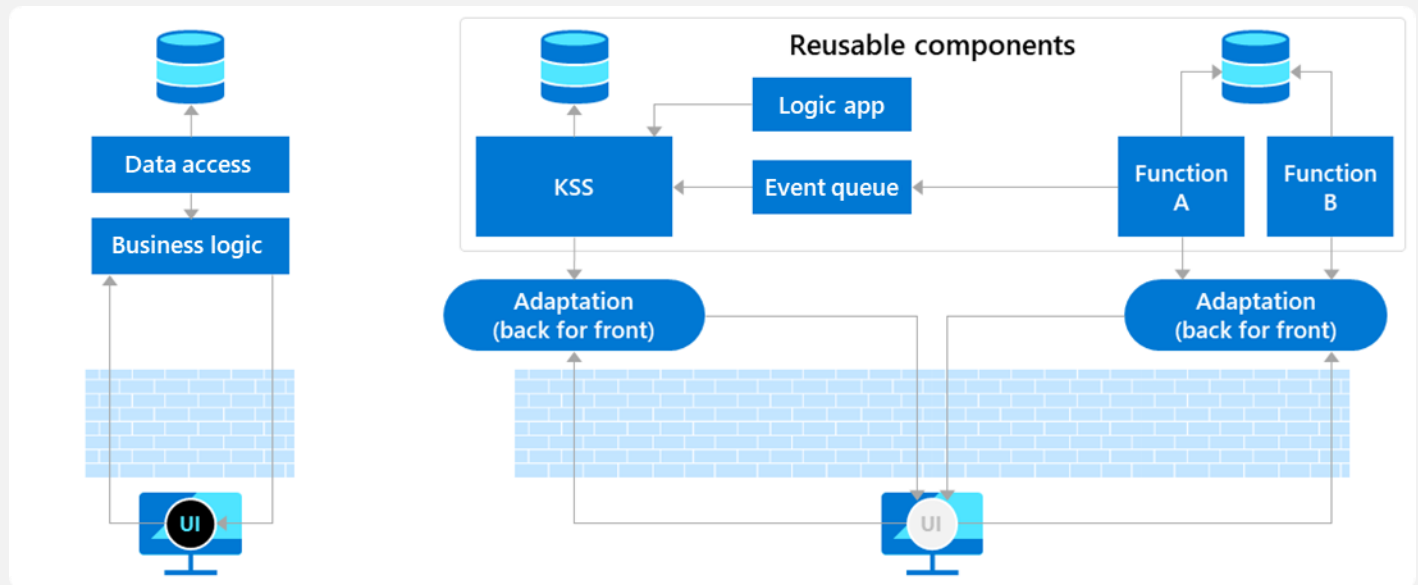
OKRs for Principle 4: Consistent, secure, and compliant from start to finish

Objectives	Key results
Maximize infrastructure consistency and increasing deployment speed	Ensure that 95% of infrastructure resources are deployed consistently across environments using IaC <i>Tip:</i> Use tag to define deployment method on resource
Define cloud building blocks according to security rules and sourcing strategy	Ensure that 95% of cloud services in the production environment are defined outside the design of the application to reusability <i>Tips:</i> Leverage a two-factor security approach using identities and network isolation Use policy to enforce pricing tiers or region; use IaC for template deployment
Maintain enterprise compliance with policies along the lifecycle of the application and platform	Ensure that 100% of resources are in compliance with core mandatory policies <i>Tips:</i> Consider a sandbox for your teams with lower policy enforcement for testing purpose Consider the audit policy effect for first stage adoption before deny or append effects
Use pipeline for deployment at Day 0	Create a discipline of pipeline utilization from Day 0 <i>Tip: Propose default CI/CD pipeline capabilities to onboard new products / teams</i>
Enable a culture of sharing across the organization and new responsibility model for Product / DevOps teams	- InnerSource is a nonfunctional requirement for any product or initiative - Products / APIs publication at organization top level is a nonfunctional requirement for any product or initiative - Product / DevOps teams are responsible for requirement discovery, through continuous architecture, component selection and implementation, to operability

Principle 5: Be adaptable and scalable

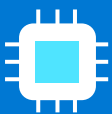
Early cloud applications used to be applications that were the center of their world, simply leveraging cloud APIs. Today, with the cloud services offer growing, an application exists more by the composition of cloud services than by its code which is not always the “center of the universe”.

Most of the time business features are delivered by a core package such as an ERP and the service layer. Serving the front Single Page Application is just an adaptative layer. Nowadays named “back for front”, the sole responsibility of this layer is adapting the outcome of existing business services so that they match the needs of the web/mobile application. The “back for front” approach aggregates data from several business services, filters unnecessary data points that would damage performance, adds some security filtering depending on user role and sends back the whole data package to the web/mobile application in one single request response and payload to reduce latency. This is especially important for mobile apps when the user is on the go with limited connectivity. By mixing serverless with product services, we can optimize development and operation costs.



Application Monolith architecture vs Composable architecture mixing ad-hoc and cloud native services

Composition allows reuse of services common to many business areas, such as file upload/download, directory access, emailing or access to common business entities.



Make your systems ready for change and solve technical debt and pave the way towards a component-based architecture.

Making systems that are adaptable for change is a challenge. Due to business pressures, tradeoffs are quickly made introducing new technical depth. After the failure of object-oriented in business applications, where the promise was to modify applications with just a single line, composition of services is the first successful technical solution to move towards a component approach for building applications.

Agile Architecture: Just-in-Time and Just Enough Architecture

While many companies have adapted to DevOps development, many are still being held back by outdated and onerous architecture processes which impact the ability of the company to quickly respond to ever-changing business needs. The emerging Just-in-Time (JIT) with Just Enough Architecture (JEA) philosophies attempt to address this issue by adapting architecture processes to align to DevOps principles more closely.

JIT/JEA is an architectural approach that seeks to optimize the delivery of technology solutions to meet specific business needs, maximizing value to the business.

[Agile & IT Architecture: The JIT-JEA way of working | Capgemini](#)



Just good enough
architecture



Just good enough
documentation



Just good enough
governance



Just in time and,



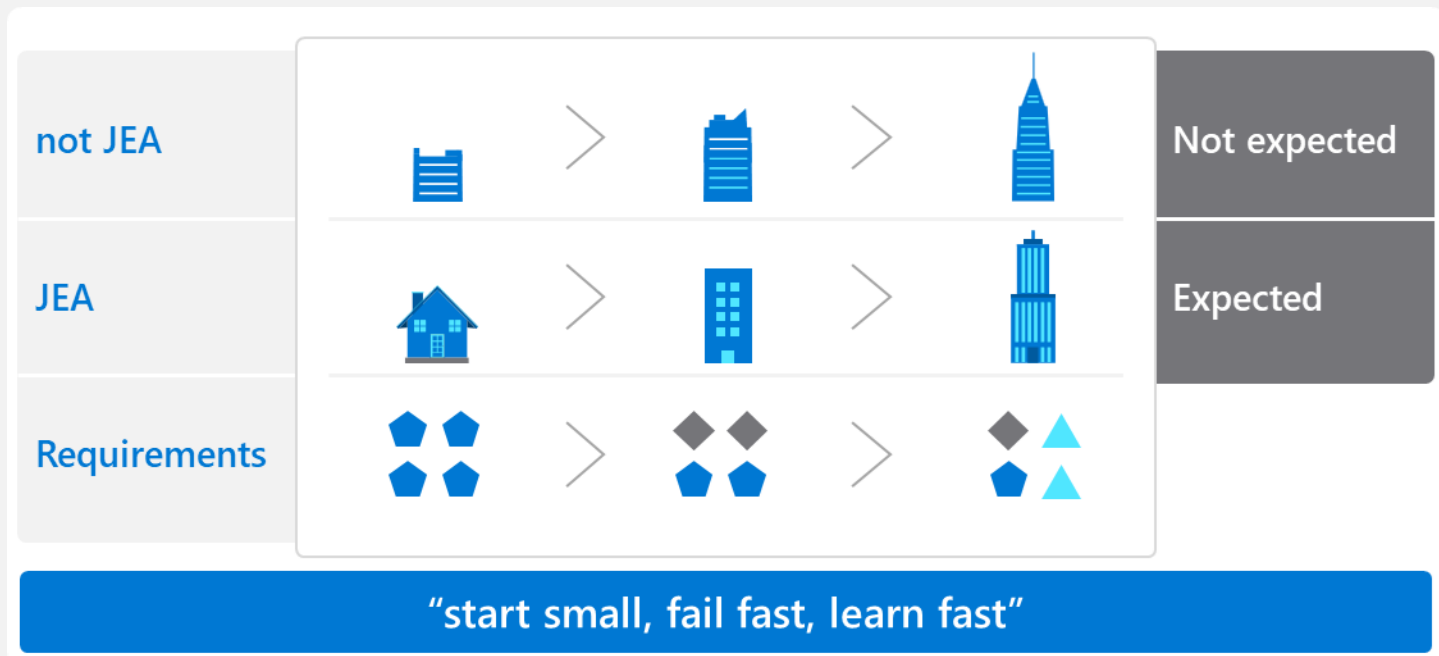
In iteration-size
chunks

[Continuous architecture](#) enables an agile operating model, starting from the problem you would like to solve, enabling product-centric approach, elevating architecture validation through its implementation, and promoting consistent responsibility model from product design to product operability.

As illustrated in the schema below, we design and build “just what it takes” to meet the current requirements - avoiding big design up-front (BDUF).

In a product-centric approach, with an iterative way of release in conjunction to continuous user feedback loop, functional requirements are evolving implying technical requirement updates.

This approach is only possible because the architecture is elevating abstractions between layers and loosely decoupled integration between components supported by scalable infrastructure and services.



In a JIT/JEA IT architecture, solutions are delivered quickly and incrementally, allowing for early testing and feedback from the business. This approach enables IT to respond quickly to changing business requirements, and to avoid unnecessary development of unused or redundant components. The JIT/JEA approach also focuses on delivering only what is needed to meet specific business needs, reducing development time, effort, and cost.

For architecture practice effectiveness, let's also adopt a structured approach shared at enterprise level. For instance, the IT architecture pillars, Availability, Reliability, Security, Maintainability, Operability, on which we can now add Sustainability, help in grouping and articulating the nonfunctional requirements.

JIT/JEA has been defined by Capgemini via 10 key principles which detail the definition of what should be considered as "just enough" in relation to architecture assignments. These key principles are:

1. **Understand the As-Is:** Ensuring there is a holistic view across business, data, applications, and infrastructure of what is currently installed and what will most likely change (not needed for complete green field).
2. **Understand the context:** Discerning the context across both business and IT, including any external factors (regulation, etc.) that may affect the results.
3. **Define the principles:** Formalizing traceable business objectives and principles, driven by the business mission and vision.
4. **Know the requirements:** Understanding and/or delivering the functional and non-functional requirements.
5. **Record decisions:** Documenting the rationale for all architectural and design decisions, ideally reflecting the principles and business needs.
6. **Ensure traceability:** Providing clear traceability back to the business objectives within the architecture.

7. **Develop the solution:** Implementing the solution(s) including investigating alternatives to ensure that decisions are not made in isolation.
8. **Assumptions and constraints:** Capturing, validating, and managing any assumptions and constraints that affect the architecture.
9. **Risks and issues:** Proactively documenting and managing risks and issues - both processes as well as results.
10. **Plan:** Creating a clear and sensible plan/roadmap to achieve the desired business outcome(s).

This type of IT architecture provides significant value to the business by enabling organizations to quickly respond to market demands while leveraging new technologies and adapting to new business opportunities. Additionally, the JIT/JEA approach allows organizations to minimize technology risk by delivering solutions incrementally within boxed budgets, thereby reducing the risk of large-scale project failures.

JIT/JEA architecture also facilitates better alignment between IT and business goals, as IT solutions are aligned to specific business requirements and not architectural ivory towers. The adoption of JIT/JEA relies heavily on the governance principles defined for the business, as these need to be sufficiently detailed to allow architects to choose the relevant architecture without needing to go through complex validation hierarchies.

Beware of technical debt with the Single Responsibility Principle

When software systems are developed with suboptimal practices, they may accumulate technical debt, which refers to the eventual cost of maintaining and evolving these systems. Enterprises want to be adaptable, however, it can be enticing to resort to adapting “fast and dirty” practices that create more harm than good. Microservices and service-oriented architecture require serious planning and governance. It’s very easy to deviate from a clean architecture, and excuses to do so are numerous.

A good practice is the single responsibility principle, coming from the SOLID principles, from which states that single responsibilities shouldn’t be spread over multiple implementations - in this case, services.

“A module should be responsible to one, and only one, actor.”

Robert Martin – Clean architecture: a craftsman’s guide to software structure and design (2018)

The problem is that cloud native architecture is hybrid in its implementation by nature: a service delivered to the user is composed from several parts that leverage native PaaS features. A similar problem shouldn't be solved in a different way! Rather, construction from smaller elements like serverless functions, events queues, and storage accounts should be packaged into reusable components.

This isn't a new architecture principle. It's simply far easier to deviate from this approach when building cloud native solutions, due to the many solutions available.

Keep microservices adaptable

With a microservices architecture becoming the "go to" choice for organizations integrating cloud native capabilities, it is important to address certain topics up front to prepare the architecture for success: meeting functional and nonfunctional requirements while ensuring simplicity, evolution and cost-effectiveness.

First, prior to commencing the microservice architecture, ensure that the OKRs have been defined. These should allow the definition of acceptable terms of service which need to be supplied by the microservice in terms of availability, response times, data loss, etc. It will also help to ensure that the design system meets the principles of JIT/JEA and does not overcomplicate the solution. The defined OKRs may also impact decisions relating to data processing. For example, eventual consistency when used correctly can reduce resource performance requirements while meeting data availability criteria compared to traditional transactional consistency.

Second, a review of the [Microsoft Architecture design patterns](#) will help to identify the major patterns and their usage in relation to microservice architectures. These patterns cover many of the typical issues faced by cloud native applications and provide a good basis on which microservices should be implemented. Many of these patterns can be packaged in reusable components and shared amongst development teams, facilitating their usage.

The combination of these two steps should allow for a better understanding of the target architecture and the patterns required to be implemented to achieve the microservice minimum viable product.

Teams responsible for developing microservices should consider supplying API clients to subscribers, like we see with the hyper-scalers providing client APIs for their services. This allows subscriber teams to more easily use the APIs without having to delve into the complexities of authentication and authorization processes, and other API related problems. It also ensures that the team responsible for the development of the microservice is also responsible for its usage, helping to abstract away communication issues from the subscriber team. A well written client API can allow the microservice team to implement many changes to back-end microservices without significantly impacting the subscriber's code.

In order to reduce the effort associated with microservices development and ensure coherence, the sidecar pattern should be considered for providing standard functionality across microservices. Implemented as a container sharing resources with the application, the sidecar container extends features of the application

container without modifying the logic of the application container. Typically logging, monitoring, tracing, and messaging are functionalities that are required across microservices, which are a viable candidate for the sidecar pattern. (e.g. adding https to an http app container, adding inbound/outbound request tracing, etc.). Each of these operations can be managed via plugin on top of the app logic without modifying the application container.

Instead of creating specific functionality per application, generic modules should provide common functionality which developers can then use easily within the main code. It's generally a better practice to use multiple sidecars, rather than having a single sidecar containing all the generic modules. This allows developers to pick and choose the functionality they require. Using the sidecar pattern reduces code duplication across microservices allowing developers to focus on the key business requirements of the microservices rather than boilerplate code that has little to no real business value.

The [Dapr framework](#) (Distributed Application Runtime) relies heavily on the sidecar pattern to allow developers to be more productive. It facilitates distributed microservice connectivity implementing as a runtime common design patterns like service invocation, publish and subscribe, binding, state management, secret management, etc.

Large microservices clusters also get huge benefits from loosely decoupled architecture based mostly on queuing services. Queues are widely used in an operating system's design as they offer a simple solution to complex synchronization problems. The same pattern applies for large microservices-based applications: designing with queuing in mind from the ground up helps build scalable solutions (symmetric pods), splitting complex tasks into a pipeline of queued items. Queuing also improves sustainability, scaling should be aligned with acceptable delivery delays as defined via OKRs, in order to flatten peaks and distribute the load across the available time window.

Leverage encapsulation to the max to stay platform agnostic

Good architecture principles allow building a great solution by using encapsulation. Encapsulating calls to services and calls from services to the Kubernetes platform keep the system agnostic, and allow the use of native cloud services like Azure Kubernetes as a Service. This immediately begins taking advantage of a much better integrated administration with impacts in terms of lower operating cost, and easier learning curve, stronger security, and lower deployment costs. These effects are amplified when using accelerators such as InnerSource libraries like the Sogeti CloudBoost library.

The Kubernetes example has its counterpart in the queuing worlds, which is also a key area of modern, resilient, and scalable applications. Such service can be easily encapsulated, allowing to deploy on several cloud vendor platforms and even on on-premise platforms for hybrid workflows. Publishing a message on an event bus can be done with PaaS services or open source products such as Kafka. The real business value comes from the decoupling brought by this publish/subscribe approach (or sometimes simple queuing). The technology behind the paradigm should just work.

Examples are endless, with the possibility to encapsulate access to any data, whether a simple key-value, a structured hierarchy represented in JSON format or a binary file containing a video. For just a relatively small expense, encapsulation brings the power and time to market of PaaS. And even without going multi-cloud, your encapsulated blob storage can easily be later replaced by a multi-tier strategy leveraging hot and cold storage, without having to change a single line of code to your app.

Encapsulation allows companies to choose the right service for the job. This simplifies hybrid set-ups and for companies needing multi-cloud capability, this allows them to benefit from fully managed PaaS services instead of having to downgrade to self-managed processing clusters.

Build business-focused serverless functions and don't forget the defaults

Together with the performance benefits and enhanced speed of development, serverless functions become a major asset in scalability and resilience. It meets application execution needs without having to worry about the sizing underlying the infrastructure because they are executed on the cloud-provider.

Notable benefits of adopting serverless include:

- **Reduced operating costs:** Rather than having to buy server farms in case of traffic surges that will remain mostly unused, this solution allows you to pay only for what is used.
- **Adaptability:** It is no longer necessary to plan traffic peaks and it evolves automatically to adopt the needs of applications using billing per second.
- **No server maintenance:** Precious time is saved by the operational teams.
- **Productivity:** Developers can easily deploy their code without managing servers.

But to be successful in adopting and scaling serverless technologies, it is necessary to understand the business needs, and define an architecture based on business concerns and capabilities.

Enterprises must adopt a business-driven architect model allowing them to define microservices based on architecture components (e.g. shopping cart management, user management, etc.). When teams fail to define the business components, the problem can be that the different functions get sub-functions and functions are chained to each other. This will create complex workflows in an application and complexity on your architecture.

Serverless functions allow companies to design asynchronous architectures to avoid any dependencies with other components or applications. Separating orchestration of events and management of events, the developer in charge of the serverless function focuses on a simple processing triggered by an event. Such an input can be easily mocked for testing by manually generating the event. It helps development teams to be autonomous in their development, knowing that they know how to reproduce the expected events without

needing to know the logic of the whole application. This is probably one of the easiest implementations of loosely coupling.

Even if we can develop the functions quickly, the product owner or business analyst should not forget to deal with the non-functional requirements:

- **Security:** Authentication and authorization management
- **Testing:** Unit tests
- **Load:** Performance test
- **Traceability:** Log management, necessary for debugging or investigations
- **Monitoring:** Alerting rules management

Each function should be designed to perform a specific task focusing on event-driver triggers: HTTP requests, IOT events, messaging, etc. Serverless architecture is used to run tasks during a short period. The main use cases for serverless are as follows:

- Event-driven architecture, in the case of an event (e.g. file event, invoice payment) that will trigger a process.
- HTTP event, to execute a HTTP request (e.g. new user) exposed through an API Gateway to realize synchronous or asynchronous treatment. The usage of IOT to trigger a serverless function for receiving or sending messages.
- Scheduled tasks, where a serverless function is triggered from a cron job or better from an Azure Logic App, and executes a regular task. (e.g. this can be used to generate a daily report send notifications).
- Under workflow, more and more serverless functions can be included in complex workflows allowing to orchestrate a complete process.

It is important when designing a serverless function, to think of simple and fast actions. If not, we will have to move to other architectures such as microservice.

The ability to change is closely related to the ability to recover

In the modern world, service-interrupting events can happen at any time. These can range from a network outage, to a cyber-attack, or even a natural disaster. When such unexpected events occur, it is important to have a robust, targeted, and well-tested DR (Disaster Recovery) plan. But it's also important to be adaptable, ready for change, and ready to accommodate new innovations.

With a well-designed, well-tested DR plan in place, the impact on the business's bottom line should be minimal. No matter how advanced the DR solution is, there is likely to be some level of impact, depending upon the type of outage. The solution should be built in such a way to minimize the RTO (Recovery Time

Objective) and RPO (Recovery Point Objective), but it is important that the solution is seamless and does not require overhead to failover or fallback the application in the event of DR.

Any cloud native service can be categorized into 3 major groups, with each group requiring a different DR strategy:

- **Data Persistent Services:** Services with stored data within. This data must be retained during a failover to secondary region in case of disaster and fallback to primary region once all services in the primary region are back online. Some of the ways to retain the data and planning DR could be taking regular snapshots of the data and copying them to the secondary region where the DR site is configured and/or setting-up data replication between the data stores and services between two regions. Some of these services include Global Distribution of Azure Cosmos DB, Replication in Azure Blob storage and several more.
- **Code Based Services:** Services that consume data from any cloud data source, process it and store it a different data store, but do not store any relevant data. These services are mostly implemented in a container, PaaS services or serverless function. A way to plan DR strategy for these types of services is to deploy them in multiple regions simultaneously. Some of these services include Azure App service, and Azure Functions.
- **Global Services:** The above services are regional services, which means they must be deployed in specific geographical regions. Services like Azure DNS, CDN services like Azure Content Delivery Network, etc. are termed as global services. These services are deployed across the cloud provider's distributed global platform. Users consuming these services may sometimes experience degraded performance or site slowness but chances of complete disruption are extremely low considering their global footprint.

Following above DR service and deployment principles let understand some of the planning principles. Any organization that plans to use public cloud should have a cloud-based disaster recovery plan following three key stages:

1. Analysis

The analysis phase of a disaster recovery plan should include a comprehensive risk assessment, as well as impact analysis of your existing IT infrastructure and workloads. Post collection of all the relevant technical information one can evaluate how current infrastructure stands against these challenges and determine the Recovery Point Objective (RPO) and Recovery Time Objective (RTO) of the workloads.

2. Implementation

The implementation phase of a DR plan helps to outline the steps and technologies needed to address disasters as they occur. The goal is to lay out a plan that helps you implement all necessary measures and respond in a timely manner. This can be documented in a simpler method using a document called 'DR Runbook'. Below are four key steps of a DR implementation:

- a. Preparation: a detailed plan explaining how to respond during events, including clear roles and responsibilities.

- b. Prevention: detailed plan to reduce potential threats and vulnerabilities. Typically includes regular updates and employee training.
- c. Response: detailed plan with automated measures implemented to ensure quick response during disasters.
- d. Recovery: detailed plan with automated measures that quickly recover the data needed for normal operations.

3. Testing

To ensure the viability of your plan, you need to test and update it on a regular basis. This can help ensure that staff remain properly trained and that the plan remains relevant to your needs. It is also advisable to perform DR drills at timely duration (once or twice a year) to ensure DR preparedness and effectiveness.

Teams are responsible for the availability of their applications in the cloud. It is important to define what a disaster is and to have a disaster recovery plan that reflects this definition and the impact that it may have on business outcomes. Organizations should ensure they evaluate Recovery Time Objective (RTO) and Recovery Point Objective (RPO) based on impact analysis and risk assessments, then choose the appropriate architecture to mitigate against disasters.

Cloud paradigm shifts organizations to consider more and more MTTR (Mean Time To Restore) and MTBF (Mean Time Before Failure), assuming in a non-zero probability to fail, the criticality to return as a normal state as soon as possible while minimizing the occurrence of failure observed by the users.

Ensure that detection of disasters is realistic and timely — it is vital to know when objectives are at risk. Ensure you have a plan and validate the plan with testing. Disaster recovery plans that have not been validated also risk not being implemented due to a lack of confidence or failure to meet disaster recovery objectives.

Designing scalable and containerized solutions

In a microservice architecture, each service faces a certain load. The difference in load requires a different number of instances to be created for each service and due to the dynamic nature of the requests, a real-time load distribution method needs to be deployed to allocate and use the resources optimally. This method is the essence of auto-scaling.

As cloud applications grow in complexity, sometimes teams reorganize into a distributed microservice architecture and software delivery cycles get faster. Throughout all of this, DevOps engineers keep looking for ways to streamline and automate the continuous deployment of code.

Once you get to a point where you're running multiple containers across multiple machines, spinning up more instances of your components doesn't scale linearly and dealing with this growing complexity gets a little fraught. Many teams will address this complexity by also using an orchestration engine, like Kubernetes.

To ensure stability and smooth functioning of processes, interactions at the producer-consumer level need to be controlled along with systematic addition of new instances, which can be done using backpressure management mechanisms. Typically, they choose these strategies below to handle the excessive demand faced by systems in a backpressure condition:

Throttling

To reduce the bandwidth of any system, one of the effective solutions is to reduce the rate of incoming flow of requests from producers. This is usually done to allow time for consumers to process pending requests before restoring its normal situation. Once this is achieved, producers can produce messages at their normal rates.

Horizontal Pod Autoscaler (HPA)

If the workload can be scaled, HPA responds to the resource requirements by increasing or decreasing the number of Pods. It ensures there is consistent performance irrespective of the situation, leading to cost-effective qualitative results. Few instances when HPA adds more Pods are when the memory threshold is exceeded, an increased rate of client requests per second is recorded, or while servicing external requests. Each workflow has a different HPA object, which regularly checks the pre-determined threshold of the metrics to accommodate changes at the earliest. While adding new consumers in the consumer groups is easy, it cannot cross the number of partitions. This increased number of consumers improves the throughput of the system and reduces the workload. A balance is created on all individual VM nodes, slowly returning to a stable condition. This is the recommended method to increase the number of instances for effectively scaling a cluster.

Vertical Pod Autoscaler (VPA)

Unlike horizontal scaling, vertical scaling is done to increase the power capabilities of the system. For example, ramping up the storage capabilities of the CPU, RAM, etc. VPA automatically recommends the values for CPU limits, based on the dynamic incoming requests. Hence, this is usually implemented when the messages are processed in batches. Because when the demand increases, the size of the batch increases, pressuring resources like the CPU. In such a situation, VPA scales or enhances resources like the CPU and memory and slowly brings the system back to its stable condition. Since microservices architecture normally deals with stateless processing and not with internal states, vertical scaling might not be the ideal choice for them.

Cluster Autoscaler (CA)

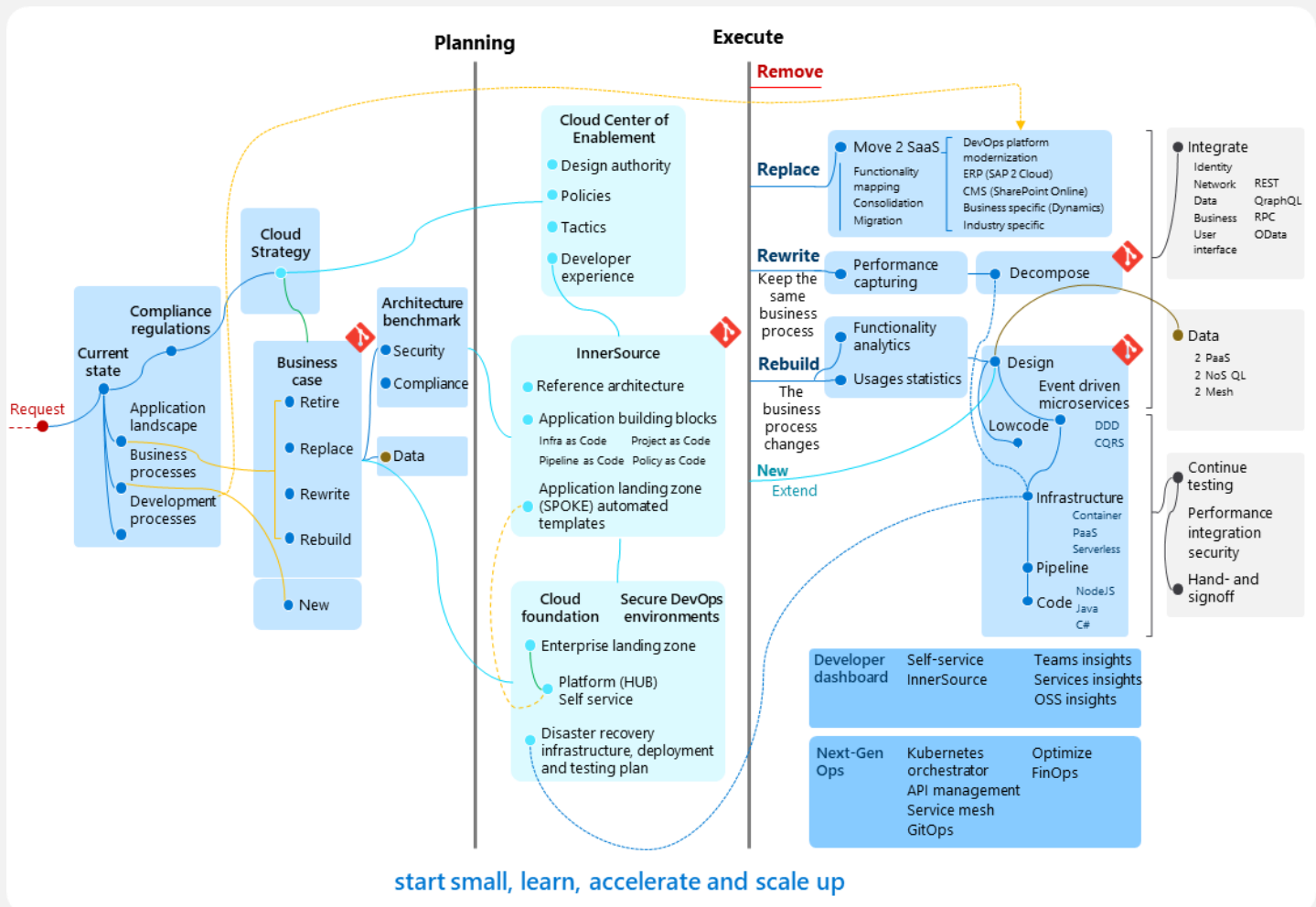
While horizontal scaling scales the number of consumers, there is a threshold for each cluster which ideally should not be crossed. Once it is exceeded, efforts to mitigate backpressure will not result in a stable system as it indicates an exhaustion of resources on that cluster. During such situations, a number of VMs are added to the cluster. The number of nodes is adjusted, as per the capabilities of the pods to share the workload.

OKRs for Principle 5: Be adaptable and scalable

Objectives	Key results
Adopt an agile and continuous architecture approach for CI/CD continuity	• Validate proof of architecture through demonstrator instead of architecture documentation • Define sets of nonfunctional requirements including velocity, scalability, agility, and sustainability used as a baseline for new architecture • Establish a target reduction in architecture cycle time (by percentage) and measure the results achieved in a new architecture or updated architecture • Assess and adapt on a quarterly basis the alignment between architecture orientations and decisions with business goals
Keep moving with adaptable microservices	• Expose - as products – your APIs leveraging composable architecture to enable business capabilities • Manage and secure 100% of your APIs in production with an API management solution using API policies • Extend adaptable capabilities to your APIs with GraphQL • Monitor 3 to 5 key metrics on APIs (usage, callers, latency, retries, etc.)
Continuously reduce technical debt	• Maintain and update the assessment and mapping of technical debt • Allocate at least 10% of your resource to reduce the technical debt in refactoring applications – as a continuous practice into your product developments and operations • Define modernization and retirement targets and measure progress quarterly
Maximize recoverability with MTTR (Mean Time To Restore) and MTBF (Mean Time Before Failure)	• Calculate MTTR and MTBF for all critical applications • Conduct Failure Mode Analysis for all critical applications with related mitigation and remediation plan • Conduct chaos testing to validate application components behavior on failure and on restoration <i>Tips:</i> <i>Aside MTTR/MTBF, calculate application Composite SLA multiplying the nominal SLA of each service involved – not considering the lowest SLA of the services</i> <i>Use Azure Chaos Studio to increase your test coverage with cloud infrastructure failures</i>
Enhance application scalability	• Implement auto-scaling for resource increases and adjustments on user driven metrics (usage in users, orders, etc.) • Optimize cost effectiveness of your application with a refactor modernization scenario leveraging serverless capabilities, asynchronous communication through events, and business logic extraction • Conduct performance testing to validate application components behavior on different loads

Sogeti Cloud Native adoption activities

Starting on the left with a request which can span the modernization of the whole application landscape, one business process or a new innovative idea.



Request

To get the right request is important, get the business informed and trained on the capabilities of cloud native is mandatory. Cloud native is not a technology play only.

Read the first principle "Principle 1: More business innovation with less IT" and practices on sections "Modular business processes in a platform organization", "Product Oriented Development teams" and "Business outcomes with objectives and key results – OKRs"

Current state

Is the organization ready for cloud native? Are there any big blockers which need to be resolved first? To get some good insights, check the current application landscape on type of applications and usages. Also, investigate the business processes - are they highly entangled and complex and are there clear customer

journeys? Next to the business and application landscape it is important to understand the way of working and the operating model of teams.

Read the practices on readiness in the first principle "Principle 1: More business innovation with less IT" on section "Readiness assessments"

Compliance and security

The current state of compliance and security is important to get an understanding of the current regulations and compliance requirements. How is compliance and security implemented, enforced, monitored and what is the knowledge of cloud native within the security and compliance teams?

Read the practices on readiness in the first principle "Principle 1: More business innovation with less IT" on section "Get compliance and security onboard and understand cloud native"

Cloud Strategy

If there is already a cloud strategy in place, is this one ready for cloud native or should it get a tactical update?

Read the practices on readiness in the first principle "Principle 1: More business innovation with less IT" on section "Readiness assessments"

Business case

When the application landscape and the capabilities of the teams are understood, it's time to assess the applications. A business case needs to be created for each application. Don't forget to look at the data, and because you also know the compliance and security requirements, immediately benchmark the current set of applications. One activity that can be done during the analytics of applications is to capture standard components and investigate default architecture building blocks. From here, we can commence planning.

Read the practices on readiness in the first principle "Principle 1: More business innovation with less IT" on section "Business case calculator, evaluate the current application landscape"

Cloud Center of Enablement

In addition to governing the design, overall development lifecycle and target operating model setup are also important. Innovation Ready organizations requires these talent pools to make sure that the cloud native approach is covering all aspects of aligning business with IT, adherence to best practices and development standards, how the nonfunctional requirements are covered (and more importantly standardized design with maximum automation), adherence to security and compliances, understanding short term vs long term design aspects, and futuristic architecture to support long term business goals.

Read the practices on readiness in the third principle "Principle 3: Enable the organization with a cloud-native foundation" on section "Bring the Cloud Center of Enablement (CCoE) to teams"

CCOE would need to have full controls throughout all the 1 to 5 principles mentioned in this paper. Microsoft Learn provides [learning resources on Cloud Center of Excellence](#) to help organizations achieve their core objectives around business and technical agility.

InnerSource

A Cloud Native approach brings speed and velocity to enable organizations to release features and products faster to their end users. Standardized design and development approach enables organizations to start small and scale to wider set of applications. Organizations can leverage a well-defined enterprise reference architecture governed by Cloud Center of Enablement (CCoE), consideration around type of application architecture patterns, application of non-functional building blocks, level of automation, and use of readily available components as key components to faster pace developments.

Read the practices on readiness in the fifth principle "Principle 5: Be adaptable and scalable" on section "Start small, learn, accelerate and scale"

Cloud Foundation with DevOps Environments:

Cloud Foundation is the starting point for any type of application modernization or cloud native development on a hyperscaler platform. Sogeti promotes the principle of *Everything as a Code* for cloud foundation provisioning which includes subscription hierarchy, Role-Based Access Control (RBAC) policies, security and compliance policies, cloud services provisioning scripts and templates, and an automated pipeline as a code approach with all the checks and gates implemented. A cloud foundation provisioning approach should be built on considering standardization and maximum reusability to enable the organization to have an enterprise scale cloud foundation.

Read the practices on readiness in the fifth principle "Principle 5: Be adaptable and scalable" on section "Start small, learn, accelerate and scale"

How Microsoft and Sogeti can help

Secure DevOps or DevSecOps for enterprises requires secure DevOps environments powered by cross-team collaboration, a focus on developer velocity, and cutting-edge tooling. Microsoft offers learning resources, products, and services to position all DevSecOps teams for innovation, regardless of language, framework, or cloud.

Azure expert managed service provider, Sogeti, continuously improves business-focused digital delivery for DevSecOps teams utilizing cloud and DevOps Centers around the globe. Sogeti uses its DevSecOps Adoption Framework and CloudBoost library to drive continuous improvement and InnerSource adoption within DevSecOps teams—leveraging the full platform capabilities of Azure for governance, security, and compliance as a cloud-native foundation.

Inspire other technical leaders to integrate security into their DevOps practices by sharing this whitepaper on social media or email.



Resources



[Microsoft DevSecOps solutions](#): Integrate security into every aspect of the software delivery lifecycle. Learn how Microsoft offers a complete solution to enable DevSecOps, or secure DevOps, for apps on the cloud (and anywhere) with Azure and GitHub.



[DevOps Velocity Assessment](#): Discover the Developer Velocity Index (DVI) score for your organization and get guidance to boost performance.



[DevOps Workflow Generator](#): Visualize your DevOps toolchain to better understand how tools relate to each other and learn how to boost your DevOps processes and workflow.



[GitHub Advanced Security for Azure DevOps](#): Learn how you can integrate developer-focused security into your DevSecOps model, with the same great code, secret, and dependency scanning, now in Azure DevOps.



[Documentation](#): Read how Azure Security helps to protect your applications and data, support your compliance efforts, and provide cost-effective security for organizations of all sizes.



[Talk to our sales team](#): Talk to our specialists to see how Microsoft can help you develop your DevSecOps team.



Sogeti works with Microsoft and GitHub to ensure its clients create value from their DevOps platform, approaches, and cloud capabilities.

[Sogeti DevSecOps Adoption Framework](#): Release faster with a DevSecOps Enterprise Reference Architecture, product descriptions, and DevSecOps Blueprints.

[Sogeti CloudBoost Library](#): Fully automated Cloud Landing zones, a Cloud foundation for Enterprise DevOps teams to land applications on a secure base.

[Sogeti OneNative Services](#): Support for the cloud native journey by providing Sogeti product and Feature teams which create business value with experienced DevOps teams on a secure and compliant foundation.

Authors



Samit Jhaveri is the **Director of Product Marketing** with **Microsoft Azure** focused on AI for Developers GTM, DevSecOps, and Developer experience. He serves as the business leader working across product management, sales leadership and finance with responsibility for defining and executing the e2e go-to-market strategy including pricing and offers and execution plans such as campaigns and field and partner motions for growing the business. Prior to his current role, Samit led an engineering team at Microsoft's Server and Tool Division and was responsible for shipping several B2B solutions for different vertical industries. Samit earned his MBA from the University of Washington and a Masters in Management Information Systems from the University of Arizona.



Pierre-Olivier Patin is the **Global CTO of Applications and Cloud Technologies** at **Sogeti, part of Capgemini**. He has been in the cloud industry for 10 years with broad expertise on infrastructure, data, and applications. As a global cloud technologist, he speaks frequently about architecture, DevOps practices and design, cloud platform utilization, and software engineering. He works closely with Sogeti's large enterprise customers to ensure their cloud adoption and software engineering capabilities maximize business outcomes.

Contributors

Clemens Reijnen

Amarjeet Singh

Francois Vaille

Krunal Shah

Mahesh Jadhav

Tony Jarriault

Johan Flikweert

Martijn Mulder

Frederic Cruchet

John Dennison

Sandip Rane

Prasanta Mohanty

Loïc Cimon

Matt Braafhart

David Trigano

Henrik Rendahl

Erik Hjalmarsson