CHAPTER 3

Overview of Benefits of Arc in the Enterprise

"Man is a tool-using animal. Without tools he is nothing, with tools he is all" was the opinion of Thomas Carlyle, a British author and philosopher who died 140 years ago. You might have thought he was a bit prescient about the evolution of technology when you consider another quote, "Go as far as you can see; when you get there you'll be able to see farther." In the enterprise, Arc is a tool to help you see farther.

What is "enterprise" computing? Gartner defines an enterprise as a business with more than a thousand employees or more than 50M in revenue.[1] Such organizations require specialized tools to scale, govern, and monitor large IT landscapes. Today, businesses of all sizes can take advantage of an enterprise computing paradigm simply by consuming

[1] www.gartner.com/en/information-technology/glossary/smbs-small-and-midsize-businesses

R. Maxwell, *Azure Arc Systems Management*,
https://doi.org/10.1007/978-1-4842-9480-2_3

SaaS [Software as a Service] or PaaS [Platform as a Service][2] offerings from major cloud vendors. However, solutions that might be required for a large enterprise could be burdensome to a smaller business to purchase and maintain. If you don't have acres of database servers, IT assets that may have a wide geographic distribution, edge computing installations, hybrid installations, or vast farms of VMs or containers, then your organization may realize enough benefit from using Azure's many monitoring tools without the Arc umbrella.

Let's take a closer look at specific use cases for Arc in some of the problem spaces that were highlighted in Chapter 1.

DevOps

Application Lifecycle Management (ALM) is management of an application during its span of usage in an organization from its design, implementation, and day-to-day usage to its eventual retirement. It is an area that has been codified into a discipline partially in response to what happens when there is no ALM. A premier example of what can go wrong is the case where a multi-tenant application begins to be deployed in a snowflake fashion, tenant by tenant, until it is no longer a single application but hundreds of applications with varying requirements and implementations. Maintenance and upgrades become progressively more difficult and expensive the longer this situation is allowed to continue. Conversely, a defined set of requirements for what the application does and proper source code management along with continuous integration and deployment can achieve the first of the 12 factor principles[3] of "One codebase tracked in revision control, many deploys" throughout the application's lifetime.

[2] www.ibm.com/topics/iaas-paas-saas

[3] https://12factor.net/

Arc is a particularly apt solution to the "many deploys" aspect of maintaining large distributed applications or services that may be hosted on-premise, in a customer's data center, at the edge, or upon a public cloud since it can manage Kubernetes installations in all of those places and assure that the desired consistency is achieved.

The impact on testing and quality assurance for an application is also profound, since automated deployments allow for environments to be strictly governed for consistency. Human-readable definitions allow disparities between environments to be quickly identified and remediated. Additionally, the disposable nature of containers means that unlike virtual machines, they are not intended to be customized individually in order to support a unicorn version of an application. Containers are often described as "cattle not pets" for this reason. When a test is run on development code, there is a reasonable assurance that if it passes in dev, it will also be runnable in QA and subsequently pre-prod or production.

Eliminating friction caused by manual deployment strategies and inconsistent environments greatly contributes to the ability to establish agile release cycles, meaning that code can be released frequently as needed without risk to the platform as opposed to large cumulative releases that are difficult to troubleshoot. This level of standardization and governance is truly new and doesn't rely on human intervention to achieve once configured. While still short of full Robotic Process Automation, the use of machine learning to mine logs and metrics[4] already allows for closing the loop on some types of issues with an automated response, thus moving steadily in that direction.

A Kubernetes control plane (such as Istio[5] service mesh) provides service discovery, configuration ingestion and validation, certificate management, runtime proxies, easy blue/green or canary releases, and

[4] https://docs.microsoft.com/en-us/azure/machine-learning/monitor-azure-machine-learning#analyzing-logs

[5] https://istio.io/

ease of operation compared to managing Kubernetes directly. What lifts Arc above using a service mesh alone is the ability to apply the same deployment configuration and security policies to clusters running in hybrid, edge, and multi-cloud environments and manage them all from a single perspective – a truly one-world approach to Kubernetes deployments.

An application's lifecycle may be short, such as those developed for seasonal campaigns, sign-ups, or other temporary service needs within an organization. Conversely, an application may live for decades with an indefinite endpoint. The Internal Revenue Service, many state governments, and industries like insurance and banking still rely on applications that were developed in COBOL for mainframes more than 40 years ago. In the case of financial institutions, what some might view as an antiquated language still has the benefit of being designed to handle fixed-point arithmetic calculations well. In this case, the central ALM stages of maintenance and feature enhancements will be managed not by a single group, but by a series of specialists passing the baton of responsibility for that application. Can Arc manage your more than 40-year-old mainframe running its 60-year-old programming language? That's unlikely; however, if you end up migrating off the mainframe and into the Azure cloud,[6] then those legacy applications can come under a very modern control plane with Arc.

During the maintenance phase, ALM core challenges include things like dealing with application failures, maintaining integration points, feature additions and enhancements, meeting evolving security threats, and more. How does Arc address some of these ALM challenges?

Arguably, the most critical issue faced in managing an application is an outright failure. Even if the length of the outage is brief, there may be substantial cost attached in terms of lost revenue and customer goodwill, the time or overtime of incident responders, and corporate trust in the

[6] https://docs.microsoft.com/en-us/azure/architecture/example-scenario/mainframe/ibm-system-i-azure-infinite-i

faulting systems. Downstream effects could include data corruption or loss, a scenario which carries its own costs (particularly if there is a breach of private information that results in legal action against the company). If the application is regulated, there may be consequences in terms of additional monitoring requirements or fines and reputational injury. If postmortem failure analysis shows security was breached, that requires a crisis response. If any of the preceding issues are encountered by a core business system that cannot be offline, then the impact is compounded.

There is confusion around the term "application" in that the singular term might make it seem there is one artifact that can be manipulated when new features or fixes are desired. In reality, what appears as a single application to a user is an orchestration of workloads that could be running on multiple systems of disparate types. The codebase itself may consist of separate repositories of application code, application variables, and environment configuration as well as deployment configurations. If the application is built using microservices, then each service will have all of these as a small stand-alone unit, and an application may have hundreds of such services. Then there is the data an application may create, consume, manipulate, or store. This could mean there are messaging systems, databases, data caches for quick retrieval, and flat file storage components. Connections between all these elements and users of the application must be made; thus, there are networking and sometimes Telco (in the case of dedicated pipes that do not traverse the public Internet) elements to also consider. Finally, all of this must have security as an integral component which can extend far beyond who or what can access a particular area and include validation of component identity with certificates, cryptographic stores for secret values like access keys, and more. Further, multiple applications might depend upon shared components or platform features compounding the risk of their failure.

This is why the objective of a product like Arc to have a very high-level view of all the elements of an application is so necessary for the maintenance phase of ALM and particularly proactively managing

potential points of failure. If, for example, a messaging queue fails, but the queue is monitored and has a failover strategy in place, then those sorts of micro failures can occur without the application as a whole suffering an outage. A modern application, especially one that is intended to run on a distributed computing platform, must have resiliency built into every application component to avoid being prone to failure.

The definition of an application as an amalgamation of a tremendous number of resources is also a good indicator of why Arc is an enterprise product, and it is at that level of complexity that its value will be realized against the effort of implementation.

In both existing and new enterprise applications, project failure is a constant risk to the extent that an entire industry has sprung up around the forensic analysis of failed enterprise applications. As a consultant, I encounter projects where we may be the second or third firm called in to restructure and deliver projects that prior consulting partners had failed to complete or that are installed but experiencing an unacceptable rate of failure. This experience has been so common that it led me to write a brief article on how to mitigate risk in consulting relationships.[7]

Over the past five years, estimates of software project failures have remained at the astonishing level of about three quarters of those initiated according to multiple sources, including Forbes contributor Steve Andriole who, in a scathing send-up of this being an accepted norm, cites management commitment to IT projects but also oversight and scope as critical issues.[8]

It's the oversight piece that could most benefit from tools like Arc. Like the 1980s PSA, "Do you know where your children are?" – a project with even a slim chance of success must think about incorporating

[7] www.linkedin.com/pulse/risk-mitigation-through-successful-consulting-ramona-maxwell/

[8] www.forbes.com/sites/steveandriole/2021/03/25/3-main-reasons-why-big-technology-projects-fail---why-many-companies-should-just-never-do-them

visibility and accountability long before the first server is provisioned. In order for projects to come under the umbrella of corporate governance, there first has to be a working governance model to adhere to, and then it must be applied from the moment an idea enters the design phase. Which governance policies apply to a particular piece of software as well as how they will be implemented and monitored should be a prebuilt decision based on the application requirements. Thus, a tool like Arc that automates the application and monitoring of policy across all registered resources shows its value from the very beginning of the ALM process when the first development environment is set up. The management of the application can be part of its development cycle so that if in some way it's not easily aligned with existing policies, adjustments can be made before it even reaches its first round of quality assurance testing.

For a software project to have a shot at success, planning must include more than a vision of what the application will do, how it will be built, and how much it may cost. It should also assess all of the potential points of failure, their impact on both the component and the project as a whole, as well as whether or how to remediate them. A mantra of modern DevOps is to "fail fast" and thereby not have problems disappear into the deeper layers of the codebase where their origin will be much harder to diagnose. Seema Thapar explains how her team performs a "premortem"[9] in the PayPal blog in order to uncover potential points of failure, while "the team still has control to fix the problems" by inviting the execution team to create potential failure scenarios before even beginning the development of the first application components.

The God's eye view that a control plane like Arc provides is desirable in the management phase of ALM because often failures do not occur in a linear set of dependencies where a single weak link in a chain could be blamed for everything past that point. Enterprise systems far exceed the

[9] https://hbr.org/2007/09/performing-a-project-premortem

complexity of a vehicle, yet that might serve as a simple analogy. If you have a weak spark plug and the others still fire, you may be okay. If the weak plug is combined with a failing alternator, then you might soon be stranded. This is why there is some pushback to the idea of Root Cause Analysis being effective in a large enterprise environment. Failures are generally complex with multiple contributors and require a holistic approach to resolve. Enterprise infrastructure and software are both able to be designed with resistance to failure in mind. When individual elements are adequately monitored, and backups stand ready to take their place, it is unlikely that the system as a whole will fail.

According to the 2020 State of the Industry Report *Software Quality Analysis*[10] (SQA), a small report published by the Consortium for Information and Software Quality (CISQ), only 17% of organizations surveyed are requiring SQA tools in their development pipeline. One of the functions of such tools is to check for code compliance with known standards such as OWASP (security), HIPAA (healthcare privacy), and others. This dovetails well with Arc's policy features which will then monitor deployed applications if, for instance, you choose to extend Azure's App Service to your Kubernetes deployment running on non-Microsoft infrastructure (covered specifically in Chapter 6 of this book). Since the Health Insurance Portability and Accountability Act (HIPAA) is a standard governing how healthcare information should be managed, Azure complies with commonly recognized methods of applying those standards to IT infrastructure and provides predefined policy sets to enable adherence to this standard and many others.[11] Examples include specifying how specific application interactions must be constructed to avoid the likely compromise of private health information. Arc thus

[10] www.it-cisq.org/pdf/soti-report.pdf

[11] https://learn.microsoft.com/en-us/azure/governance/policy/samples/hipaa-hitrust-9-2

prevents another issue noted in CISQ's SQA report that software quality violations are often ignored by developers – a scenario that is unlikely to occur when it is known that substandard code will not be deployable.

This leads directly to another core DevOps objective, that of continuous integration of new features or modifications into the existing codebase. It is the methodology to achieve the 12-factor manifesto objective of a single codebase using frequent (sometimes multiple times per day) pushes of developer code into the main codebase for the application. In this way, new code is constantly verified to work within the context of the existing application code. When integration is not continuous, there is a significant risk that the feature branch may drift from the architectural patterns used in the application and also perhaps grow large enough that when it is finally integrated there are multiple incompatibilities to resolve. As the application continues to stray, the technical debt accrued in terms of the effort to fix or maintain it burgeons, a problem common enough that technical debt is one of the three main focuses of CISQ's 2022[12] report, where it was estimated to consume an astonishing 33% of an "average developer" workweek. Automating builds with GitOps is a primary way this tight feedback loop of continuous integration is accomplished.

GitOps

Once an organization moves toward automated deployments and configuration as code, then it becomes obvious that the configuration files are data, and the repository is roughly analogous to a database for that data and thus a candidate for applying well-known principles of data consistency. The configuration store (Git repository) becomes the single

[12] www.it-cisq.org/wp-content/uploads/sites/6/2022/11/CPSQ-Report-Nov-22-2.pdf

source of truth for environments and the applications deployed within them. The store also exposes the configuration to auditing and monitoring processes to assure configuration remains consistent.

Arc makes use of a CNCF project called Flux[13] to achieve what is known as desired state configuration in Kubernetes clusters.[14] Flux monitors the Git repository (polling for changes every five minutes by default) containing an application's configuration for changes in order to assure that the state you have defined for your application via a Kubernetes manifest is a match to what is running in your cluster. Flux's state management is not only additive that is pushing new configuration, but unlike many current deployment pipeline tools, it can also remove values that are no longer part of the manifest. It can be used to automatically update your configuration to the latest version of a container, thus assuring its deployment configuration will automatically remain current.

The creator and CNCF contributor of Flux, Weaveworks (who are also reputed to have coined the term GitOps), proudly noted at their 2022 GitOps conference[15] that Flux is integrated into the GitOps pipeline of major platforms, including AWS, VMware, Azure, Red Hat, and more. As we will discover in the upcoming chapter on Arc-enabled Kubernetes, Microsoft has been using Flux since version 1.0 and with the integration of version 2.0 notes that GitOps using Flux is now a "first-class citizen"[16] with Flux available as a managed service so that both Azure Kubernetes Service and Arc-enabled Kubernetes can be deployed using GitOps pipelines. In addition to facilitating swift deployments, Arc-enabled Kubernetes clusters also benefit from continuous monitoring and self-healing of deployed

[13] www.cncf.io/projects/flux/

[14] https://learn.microsoft.com/en-us/azure/azure-arc/kubernetes/conceptual-gitops-flux2

[15] www.weave.works/blog/gitops-days-2022-recap-major-clouds-vendors-offering-gitops-with-flux

[16] www.youtube.com/watch?v=6OjhYD2wLkY

resources, thereby creating a resilient platform for enterprise workloads. GitOps integration means all the typical deployment tools from the CLI and PowerShell to portal commands are available, and you can monitor the status of your deployment as it runs from within the Azure portal. This expeditious approach allows deployments across Azure, on-premise, and competitor's clouds to run uniformly and at scale while reaping significant cost savings by eliminating tedious and error-prone manual deployments.

Automating deployments using GitOps greatly accelerates the pace of deployments and provides operators with fast feedback. Integrations become simpler when both the infrastructure and code are predictable and visible to development and operations teams. Test and QA cycles for developers also benefit since deployment may go from hours and days to minutes and seconds.

Overall, GitOps contributes greatly to implementing continuous integration and deployment as part of an agile software development approach to ALM. Key concerns around the safety of deployments are minimized or eliminated, and the entire process comes under an organization's governance umbrella, no longer subject to the vagaries of individual approaches and preferences that are not testable, repeatable, or safe.

Governance and Policy

Enterprise-level governance is a process with known standards and accepted methodologies. Typically, it may involve an executive steering committee reporting to the highest levels of the organization, senior IT leaders such as CTO or CIO, serving as a consultative guide to the committee as well as representation from legal and compliance departments. A comprehensive governance plan will specify standards across broad categories that can have a direct impact on regulatory compliance, security, performance, profitability, and more. Almost always

implementation of these standards will have some impact on IT, or in larger organizations there will be subsidiary governance manifests around technology standards.

IT implementation of these standards requires considerable planning and systems architecture since the application of a governance mandate may be extremely complex. Ideally, both the governance document and its implementation specification will be living documents versioned in lockstep with any update to policy triggering a review of the implementation plan to assure it remains compliant. Conversely, when CTOs become aware of offerings in the constantly evolving technology landscape that will more comprehensively fulfill governance objectives, the implementation specification may update without any change to the governance plan. In real-world scenarios, synchronicity between governance and operations is extremely difficult to accomplish, let alone maintain. That is why yoking them together through Arc's ability to extend the reach of Azure's policy engine to the far netherlands of large IT estates is a powerful leverage in the effort to apply governance.

Scenarios that tend to precipitate changes to governance include business growth and acquisition, evolution of threats both commercial and operational, profit opportunities requiring compliance upgrades, and the rapid changes to technology itself.

Arc lives up to its definition as an enterprise-wide control plane by facilitating monitoring and collection of data that can then be fed back into threat assessment tools. It's important to remember that Arc is not the tool performing the monitoring; rather, as a control plane, it facilitates access via an agent where that is required. That agent is performing on your behalf the activities you want applied to an asset, from monitoring to updates and application of policies.

A March 2021 Cloud Native Computing Foundation outlines an expectation that DevOps best practice will move beyond the implementation of policy defined in document repositories to Policy as Code, which the author defines as "the process of managing and

provisioning policy enforcement tooling through machine-readable definition files."[17] This is not so far-fetched since in coding a template for a Kubernetes deployment things like ingress and egress rules or applying security standards are in effect implementing policy.

This obvious need to automate infrastructure controls has led to projects like Crossplane,[18] which, similarly to Arc, are attempting to extend infrastructure management, but in this case using a Kubernetes control plane in a cloud-agnostic fashion suitable for multi-cloud deployments.

Determining whether Arc or an alternate approach would be better for automated policy implementation and monitoring depends on several factors. Is the enterprise historically a consumer of Microsoft products and therefore its operations team is well equipped to manage non-Microsoft assets using familiar Azure paradigms? Organizations already invested in the Azure Stack would also find Arc to be an obvious choice. Since Crossplane is basically a Kubernetes operator, if an organization isn't running Kubernetes clusters it's an unlikely choice. Crossplane also lacks Arc's tight integration with Microsoft SQL Server, so that is likely to be a key decision factor in organizations where that database is widely used.

For companies with a long history of Linux Servers and open source software though, the CNCF Crossplane project would likely integrate much more smoothly with existing workflows. Interestingly, Crossplane claims both Red Hat, a publisher of a popular Linux distro, and Microsoft as among its supporters. While the two tools aren't mutually exclusive, for instance, you could use Arc's monitoring capabilities in conjunction with Crossplane's infrastructure; deployment chaos can sometimes result in environments overloaded with too many tools and not enough integration. Since the infrastructure landscape serves as the foundation

[17] www.cncf.io/blog/2021/03/29/what-is-kubernetes-policy-as-code/
[18] https://crossplane.io/

of critical line-of-business applications, its architecture must be holistic, avoiding redundant or shadow IT solutions that create more problems than they solve.

Modernization

Digital transformation, shift left, modernization, and cloud migration are just a few of the phrases reflecting global efforts to transform enterprise IT to a distributed computing model. First, there were microservices, then a stampede to the cloud, and now a more realistic hybrid approach. What causes an organization to embark upon this digital journey?

Insecure, nonperformant, and nonscalable applications often incite modernization efforts, as highlighted in Table 3-1, discussing the notorious Equifax breach of consumer credit data.

Table 3-1. *Data Breach Factors*

Risk Management Java Example	The famous Equifax breach was attributed to an unpatched Struts vulnerability. Security Boulevard logged 20 years of Struts vulnerabilities and the packages they affect	Lacks built-in protection against many common security risks. Amid legacy libraries are known vulnerabilities. Weak or ineffective “security” strategies such as obscuring URLs
Scalability	Not cloud-ready OOTB. Model/View/Controller with tightly coupled layers. Relies on a plug-in system	Accomplished by clustering and load-balancing technologies
Operability	Security options require construction of filters, expert knowledge of vulnerabilities, and remediation	Security can be misconfigured, heavily manual. Scaling approaches require ongoing configuration and administration

Many organizations have thousands of such legacy applications that are difficult to maintain and protect. Some of these may not be identified or considered in an overall risk profile due to the challenges of cataloging large infrastructures and their dependencies. Remediating security vulnerabilities is sometimes not possible on legacy platforms, and performance improvements may also face obstacles. Thus, it becomes apparent that *modernization has value independent of cloud adoption*.

Modernization has value independent of cloud adoption![19]

A principal benefit of Arc to enterprise modernization efforts is the capability it provides to catalog large infrastructures and monitor hosted application stacks running on them. Modernization projects start with an inventory of current applications and their business value. Beginning left with the value to the business is key to not recycling old tech for its own sake, but instead serving the reasons for which a particular business or nonprofit organization exists. Sometimes, initial assessments may reveal that much of the functionality of an old application is being served elsewhere, and the app itself is still living simply because it performs one calculation or is familiar to certain individuals who prefer it to adoption of newer methods and different applications. Some of the effort that was initially expected for the transformation may fall away quickly when business value is the first criteria for retention of an application.

It is extremely challenging to enumerate large corpus of applications and their supporting infrastructure, so much so that many organizations never complete a thorough audit of their IT landscape. It is key to a successful effort to consider the time required to accomplish the initial assessment. If large efforts are not undertaken concurrently with sufficient

[19] 2020 Magenic Masters Course on Practical Application Modernization Strategies, authored and delivered by Ramona Maxwell

team support and a good dose of automation, the risk becomes that the intended modernization strategies themselves may become outdated before they are implemented, creating a vicious cycle of always remaining in an outmoded state. The adoption of Arc can provide a dose of prevention if all new assets are required to be exposed to its control plane, but application modernization projects are often a juncture at which technical debt must be paid if success is to be achieved. Some systems aren't worth the debt acquired, so the question becomes how to replace them with whatever product is the modern standard for the company's industry in order to assure it remains competitive with its peers. Often, that product will be a SaaS solution.

As existing candidates for modernization are identified, a common approach is to perform a capability assessment[20] of inventoried items against a matrix of industry norms. This can help segregate applications by their importance, known deficits, and other key attributes before deciding how to treat each similar set. A popular industry paradigm of the "Five Rs"[21] can help to differentiate between core strategies for rationalizing a large application portfolio. These five are

- Rehost
  - A modern application will be platform agnostic
- Refactor
  - Look for opportunities to reorganize the application so that part or all of it can be migrated to microservices or even PaaS offerings
- Revise

[20] https://cio-wiki.org/wiki/IT_Capability

[21] https://docs.microsoft.com/en-us/azure/cloud-adoption-framework/digital-estate/5-rs-of-rationalization

    - Revise the current application only to the degree that allows it to be hosted on a cloud platform
- Rebuild/Retain or Retire
    - Rebuild on-prem, retain existing, or retire
- Replace
    - The beauty of greenfield development with the constraint of assuring uninterrupted service

The Five Rs are shown in Figure 3-1.

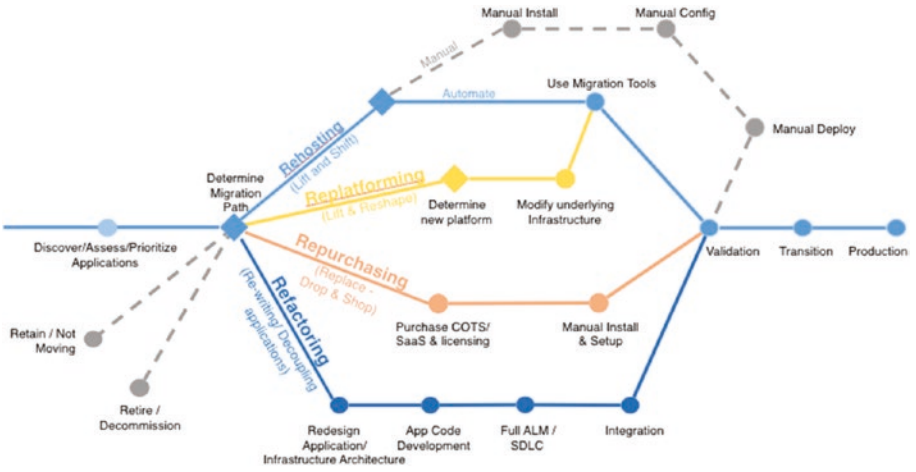

Figure 3-1. *The Five Rs – Credit: Stephen Orban, AWS*

Performing adequate capability assessments will assist in determining which approach to take with a particular application or group of similar applications. For instance, applications that are already platform agnostic or containerized may simply be *rehosted*. An important goal in rehosting is to remove manual configurations so that even if the applications themselves are not optimized to be cloud native, their management is much less burdensome. Sometimes, this path is taken as part of a longer-term strategy where individual apps will be assessed for further modernization after a

cloud migration is complete, and sometimes it is the end goal. This may be the case when the reduction in operational costs significantly reduces the total cost of ownership for those applications, while modernization may have much lower returns (particularly for applications that may be near the end of their lifecycle). The operational benefits of moving select application types to a cloud provider's platform in a "lift and shift" were present before Arc, which only sweetens the deal with dramatically improved administrator overview and control plane capabilities.

Refactoring the application is often preferred for large monolithic applications that are reaching their capacity limits as to cost-effective scaling, have become entangled with internal and external dependencies that were not well managed, or may be difficult to integrate with modern software applications and effectively secure. This process is rarely quick and is sometimes fraught with unexpected pitfalls in terms of dependencies. Often, a Strangler Fig[22] pattern (originally conceived by Martin Fowler[23]) is used to peel off portions of functionality into individual microservices that will over time form the building blocks of the now modernized application. This extended timeline previously also applied to the benefit of managing the new microservices in the cloud as they gradually replaced old functionality from the original application. However, Arc's ability to manage on-premise assets as natively belonging to Azure accelerates governance, server management, and security benefits of the cloud from the moment the Arc agent is installed and management controls applied. Microsoft's implementation of the Strangler Fig pattern particularly recommends a façade over the application as a whole, both the legacy monolith and the modern services, in order to avoid dependent users or applications having to make adjustments as

[22] https://docs.microsoft.com/en-us/azure/architecture/patterns/strangler-fig

[23] https://martinfowler.com/bliki/StranglerFigApplication.html

to the location of app components (when that applies). In a sense, Arc accomplishes the same abstraction for the configuration and management tasks involved in administering the application.

Revision is a less common approach that touches the application only to the minimal degree needed to move it to a different host. It might apply to singular use cases where a prerequisite functionality needs to move quickly before the components that depend upon it or for a group of applications that need only one small adjustment to move.

Choosing to *rebuild* or *retain or retire* an existing application is often prudent choices. As part of the original assessment process, there are often good candidates for retirement due duplication of functionality, low usage, and other criteria. Removing these, along with any associated cost allocation, contributes to the speed and lowers the cost of the modernization project as a whole. As will be discussed in the chapter on data migration, it may be helpful to bring host servers under management as early as possible in the project so that usage patterns may be tracked. Rebuilding an application that gains little from transitioning to microservices but will still benefit from cleaning up outdated technical stacks and other typical problems can be worthwhile. The rebuild effort can include a modern deployment pipeline to reduce risk and reliance on the old-school practice of using feature flags (switching the release on if it appears to work or off if it creates issues) to add new capabilities to live systems. When the rebuilt application is brought under Arc's control plane, then its management and security as part of the enterprise application portfolio will also be assured. Even applications that are selected for retention only (e.g., do nothing) should be evaluated as candidates for consolidated management under Arc since the payback from an Arc deployment is partially tied to gathering up all stragglers. Not only the application but its operational ecosystem stand to benefit from the centralized management that Arc provides, for example, Microsoft

Defender for Cloud[24] can protect servers outside of Azure when they are Arc enabled, potentially replacing several on-premise or competitor cloud antivirus [AV] solutions, thereby reducing both complexity and cost.

The *replace* option is guaranteed to generate enthusiasm in business stakeholders, architects, and developers alike due to the opportunity to use cutting-edge technology and innovate solutions that if executed correctly may propel a business into new profit opportunities. It is also, as previously discussed, fraught with the danger of project failure or cost escalation. Even if costs are contained, it is generally an expensive option that may require extensive validation to gain approval. If designed correctly, it will be resilient (self-healing), rapidly scalable, secure, and potentially have many other enhancements, such as artificial intelligence, machine learning, and more. It provides a golden opportunity to apply governance and management strategies from the very beginning of a new application's lifecycle.

Upgrades

Since many of the options discussed earlier refer to classic hosting of applications in data centers (whether on-premise or on a cloud provider's infrastructure), the issue of how to update running systems remains critical.

Common types of updates include patches to fix security vulnerabilities or bugs in a current version of software, updates which are improvements to software that are not a new installation but usually increment the version, and lastly upgrades which usually completely uninstall an older version of software in order to reinstall a completely new version but retain the user data and preferences. The necessity of applying any of these may occur all the way through the technical stack

[24] https://learn.microsoft.com/en-us/azure/defender-for-cloud/quickstart-onboard-machines?pivots=azure-arc

from a server's firmware and operating system to end-user software and everything else in between, including things like network protocols, firewalls, etc. Microsoft has its tradition of "Patch Tuesday"[25] releases, and other companies also have distinct routines with an out-of-band release signaling a significant security vulnerability that must be dealt with immediately. The incredibly complex job of managing a living IT landscape has led to Herculean efforts by administrators with entire teams devoting 48 hours of their weekend to installing and validating an endless stream of items which are continuously being stapled onto software packages. Colloquial wisdom held that responsible administrators would stay six months behind Microsoft's release date before installing update packages, since the updates themselves sometimes created issues and were often impossible to roll back. An early incentive toward cloud-hosted PaaS[26] (when reluctance to abandon private data centers was rampant) was the Nirvana of living in a world relieved of the need to manage patching, and this is still a primary driver toward rehosting applications on PaaS since not only OS components are updated but also SDKs for application code as well as database and security updates.

Arc's approach to updating servers is world's away from all-night sessions of humans watching for maintenance side effects. It was designed to enable uniform patching and upgrades across hybrid and multi-cloud environments in service of a single control plane mandate for enterprise infrastructure administration.

Imagine you have Ubuntu Server running on an EC2 in AWS as part of your enterprise fleet. About as un-Microsoft as you can get, right? By installing the Arc agent on this server, it "projects" its runtime information into your Azure Arc dashboard. As with servers you may have running in Azure, it can then live up to its control plane moniker by applying updates

[25] www.microsoft.com/en-us/msrc/faqs-security-update-guide

[26] https://searchcloudcomputing.techtarget.com/definition/Platform-as-a-Service-PaaS

and policies, running analytics, and more. Arc can then apply policy to assure the Azure Monitor agent (required as of mid-2024, formerly the Log Analytics agent was used[27]) is installed on your Ubuntu Server. Once configured, this opens up the plethora of monitoring options that will be discussed in Chapter 8. This is one of many examples of how Arc allows administrators to benefit from their existing Azure skills to manage a diverse IT landscape.

Arc utilizes the information provided by the agent to determine whether the patch level of your Ubuntu Server matches the desired state you have configured for servers of that type. If it does not, then updates can automatically be applied to make the server compliant as shown in the diagram in Figure 3-2 from Microsoft's documentation.

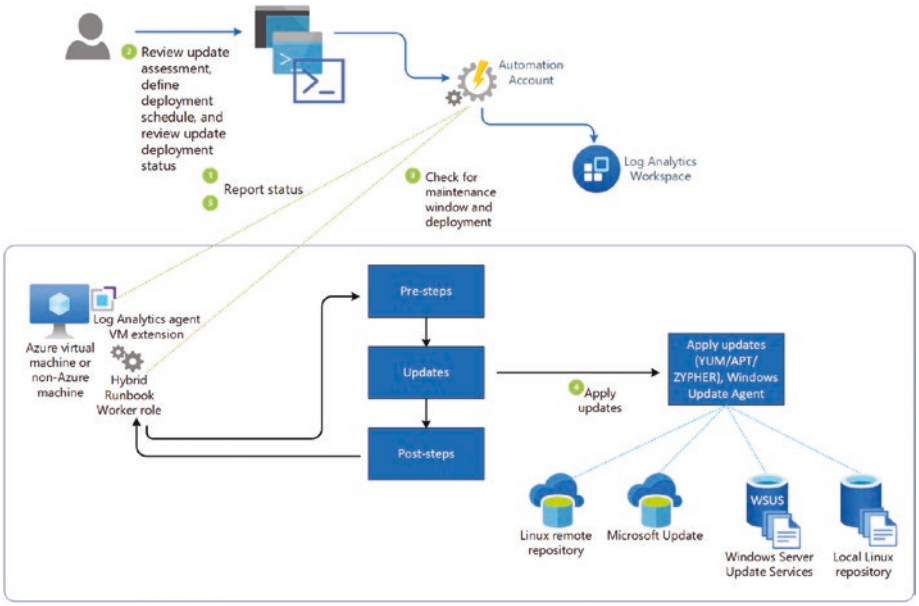

Figure 3-2. *Credit: Microsoft Azure Update Automation Documentation*

[27] https://learn.microsoft.com/en-us/azure/azure-monitor/agents/azure-monitor-agent-migration

This does not mean abandoning appropriate cautions including patch testing environments, proper change management practices to document each update, and subsequent thorough smoke testing to assure applications running on the newly patched server still work properly. What it does accomplish is automation to assure critical patches aren't delayed, monitoring that will raise alerts before failures can cascade into adjacent systems, flagging of troubled installations as not suitable for production, as well as discovery of what the challenges will be in applying a particular update to production systems (for instance, if a reboot is required, then a planned outage may also be necessary). An organization's security profile will be raised proportionate to the assets under active management, as today's threats aim to probe for any outlier that can provide a foothold for further intrusion. Today, systems management without automation is nearly equivalent to no management at all.

Systems management via policy is also advantageous in that effective patch and update management depends on prioritization of both the systems being updated and the changes being applied to them. Servers hosting core line-of-business applications require more vigorous oversight, detailed analytics, and prompt attention to risk than those with less critical workloads. Likewise, updates themselves have varying levels of criticality and value that must be considered in the constant rebalancing of resources and requirements that is the Wallenda walk of enterprise systems management.

In each of the aforementioned topics of this chapter, there is another layer that will ultimately determine the success or failure of the application, and that is end-to-end security. In Chapter 4, we'll examine the practical application of a zero trust security model in the enterprise and how Arc may contribute to its implementation.