Introduction to a software binary taxonomy

It’s Time to Evolve: End User Windows Application Delivery In the 2020s

It’s Time to Evolve: End User Windows Application Delivery In the 2020s: Introduction to a software binary taxonomy

Table of Contents

Abstract and introduction

Publication date: **August 3, 2022** (*Document revisions*)

Abstract

Today's business world relies heavily upon the heritage of business applications that are interacted with by users and that were created over the last 30+ years using the Microsoft Windows operating system (OS). Over time, these business applications are being replaced with alternatives. However, organizations still depend on the functionality provided by the Windows OS, and these applications are often challenging to efficiently deliver at scale using the Windows operating system. This paper outlines approaches to bring efficiency to the management of an end user Windows application portfolio delivered using Amazon Web Services' End User Computing (AWS EUC) services.

Are you Well-Architected?

The AWS Well-Architected Framework helps you understand the pros and cons of the decisions you make when building systems in the cloud. The six pillars of the Framework allow you to learn architectural best practices for designing and operating reliable, secure, efficient, cost-effective, and sustainable systems. Using the AWS Well-Architected Tool, available at no charge in the AWS Management Console, you can review your workloads against these best practices by answering a set of questions for each pillar.

For more expert guidance and best practices for your cloud architecture—reference architecture deployments, diagrams, and whitepapers—refer to the AWS Architecture Center.

Introduction

At the start of 2022, AWS offered two managed services that deliver Windows applications to end users: Amazon WorkSpaces and Amazon AppStream 2.0.

WorkSpaces is a Desktop-as-a-Service (DaaS) solution that helps customers build scalable and secure cloud-based desktops for any number of users. AppStream 2.0 is a fully-managed, secure application streaming service that allows customers to stream desktop applications from AWS to a web browser. AppStream 2.0 provides users with instant-on access to the applications they need with a responsive, fluid user experience on the device of their choice.

Both of these services share the same basic premise: they provide an environment upon which Windows applications are launched and interacted with by remote users. While there are differences in the underlying architectures of these services, the shared components create commonality when you consider the run time environments and software components that are required to support their delivery.

This document outlines challenges associated with Windows end user application delivery and the source of these problems. It provides a taxonomy and outlines factors aligned to the AWS Well-Architected Framework that affect application delivery. Finally, it defines a set of approaches that you can use to guide the choice of optimum application delivery. The provided taxonomy and approaches help to inform the decision-making process that needs to be considered when determining the approach to deliver different types of software binaries that need to be delivered from either the Amazon WorkSpaces or Amazon AppStream 2.0 service.

End user application delivery challenges

There are two main categories for capturing the challenges that organizations frequently encounter with Windows end user application delivery. The first category is technical and allows all the challenges associated with technology to be considered together. The second category is organizational and captures all of the non-technical challenges associated with application delivery, such as processes, internal capabilities, strategy, security, governance, and compliance. This document focuses mainly on the technical challenges, but does briefly explore some of the organizational challenges.

Technical challenges

One of the main challenges for Windows end user application delivery is the heritage of the OS and its ongoing support for applications that have been used by businesses for decades. The ongoing support presents a challenge in terms of introducing support for new features or standards, while continuing to support the old ones. The anatomy of Windows applications considering the heritage of Windows is explored in Challenge 1: Anatomy of a Windows Application in this document.

The packaging of Windows application binaries is an ongoing challenge with the Windows OS. Microsoft has introduced multiple standards over time (see Challenge 2: Packaging Formats for Windows Binaries in this document). In addition, the lack of standardization to align with a common packaging format amongst Independent Software Vendors (ISVs) has led to complexity and increased costs for organizations with a large portfolio of applications. Packaging costs in these organizations are incurred because the organizations frequently choose to re-package vendor-supplied packages. This is done to align with an internal packaging standard that enables conformity and compliance with the organization's packaging, security, and platform standards and policies.

Another challenge for Windows is its historical broad installation base. ISVs choose to author and release their software for Windows to target the broadest potential user base. The net result is a large number of vendors with a large number of applications, which in turn have huge differentiation in terms of design, packaging, and operation. The differentiation results in complexity as seen through each organization's application delivery lens, and this contributes to the complexity challenge associated with application delivery.

The Windows OS gives users the opportunity to achieve a specific technical outcome in multiple ways within the application delivery frameworks available on Windows. This ability highlights the flexibility of the platform, but the extensive choices that are available to organizations, administrators, and architects allow many different approaches for application delivery to be adopted. The breadth of these choices and the resultant complexity is unique to each organization and in time results in technical debt. This occurs because technological solutions are deprecated over time in favor of new approaches.

The introduction of a 64-bit Windows desktop OS by Microsoft in 2005 signaled that Windows end user computing was moving into the 64-bit era. Design decisions made during the creation of 64-bit Windows meant that application compatibility for legacy applications that relied on the NT Virtual DOS Machine (NTVDM) present in 32-bit Windows (i.e. DOS and 16-bit Windows applications) were no longer supported on the 64-bit OS. While this presented a challenge to organizations in 2005, over time the challenge has reduced. Many organizations still have legacy applications incompatible with 64-bit Windows. They have a dependency on 32-bit Windows, which, fortunately, will still be available and supported in Windows 10 until October 14th, 2025. However, running a mixed estate of 32-bit and 64-bit Windows 10 OS' within these organizations is a large overhead. It adds yet more complexity to the application delivery capabilities required by the organization. The introduction of Windows 11 in 2021 mandated the use of a 64-bit Windows desktop OS beyond 2025. Native legacy support for applications outside of participating in the Extended Security Updates program is time constrained to October 14th, 2025.

As outlined in Challenge 2: Packaging Formats for Windows Binaries in this document, Microsoft Windows Installer technology was introduced in 1999. This technology introduced the adoption of Windows Installer (MSI files) as the de-facto packaging format. Internal packaging standards were created by many organizations to satisfy their requirement for a widely-supported format and movement away from proprietary formats, such as WinInstall. Although MSIs became a commonly-used standard for packaging Windows application binaries, they also provided an infinitely extensible capability through the provision of custom actions. With this capability, packagers and ISVs can create their own proprietary code that can be launched during the installation or uninstallation of an application. Custom actions are evaluated in real time; the outcome of a custom action cannot be predicted because it depends on the state of the machine and the other applications deployed to it at the time of installation. The lack of determinism within Windows Installer packages across an enterprise in turn introduces a risk because of small variances across machines having the potential to result in applications being installed differently across a collection of what appear to be similar or identical machines.

Challenge 3: OS application delivery approaches and initiators in this document describes the extensive options that are available for enterprises to deliver end user applications to users. This section outlines the approaches available to install applications and the initiators that are available to determine when applications can be installed. Ultimately, the key goal for end user application delivery is to ensure that users can be productive and aren't hindered by applications being unavailable or not installed when they first access a new platform. To ensure this goal can be achieved, this section provides some insight into the options available to maximize user productivity.

Organizational challenges

With the advent of cloud and other new technologies and frameworks, there's a shrinking talent pool that has the knowledge and experience of both the Windows OS and the applications it delivers to address these application delivery challenges. Ultimately, the scarcity of skills will become a large challenge within organizations that choose to deliver applications using Windows. Therefore, if a set of guiding principles that are underpinned by best practices can simplify the decision-making process and be applied holistically, the organization can be empowered to address the talent pool challenge.

Some organizations give users an opportunity to develop their own applications within other applications. These types of applications have frequently been created using the built-in capabilities in some applications to extend the application's base functionalities. For example, the Microsoft Office suite has been used this way extensively to improve the efficiency of internal processes by tailoring an Office application for a specific use case. These End User Developed Applications (EUDAs) provide value to the business, but the lack of governance and adherence to any common standards presents a challenge to organizations because they can be complex to manage, deploy, and maintain within an organization. Most of these applications were created due to the ability to build them within an approved application and therefore they don't necessarily represent a failure of governance within the organization.

The heritage of Microsoft Windows has meant that software was historically installed and maintained manually on individual machines. This allowed administrators to tailor the installation of the applications and take into consideration other applications that are installed on the same machine. At first this involved trial and error on a per application and per machine basis. Over time, consistent patterns were observed, which allowed issues to be resolved quickly and efficiently through the creation of tacit knowledge amongst the administrators. Eventually, Windows management toolsets were created and evolved into today's endpoint system management tools

and Enterprise Mobility Management (EMM) tools. These tools automate the installation and therefore improve the scalability of the administrator teams. However, the tools don't have the same intimate knowledge of the application portfolios and interaction between applications. This can result in application conflicts and failed installations. It is important to establish an awareness of the application portfolio within an organization and where conflicts occur. These issues can then be proactively addressed before application deployment. There are tools available on the market from various vendors (such as Rimo3) that can assist with this application assessment, and this capability is useful not just for OS migrations but also for ongoing management of a Windows application portfolio.

The rate of change associated with applications being released on the Windows platform has accelerated significantly in the last 10 years. Traditional change and release management processes must evolve to satisfy the higher rate of change. Many organizations have embraced a DevOps or DevSecOps culture within their organization to provide the organizational capabilities required to effectively deal with increased release cadence.

From a historical perspective, it's important to understand why major organizations have created their own proprietary application packaging standards, and the impact that this has had on application delivery. The definition, development, and ongoing refinement of packaging standards that apply to Windows applications within organizations were initiated to ensure that applications adhered to the organization's security and compliance requirements. This helped to ensure that applications were not released that could compromise the integrity of the Windows devices the users were using. Compliance with the resultant proprietary standards was frequently achieved through the re-packaging of applications to ensure applications could be delivered to users, standards were maintained, and security was not compromised.

The net result of the need for compliance was a spawning of an entire application re-packaging industry and specialist skillset. Ultimately, the value to the business was that the risk associated with deploying non-compliant applications was eliminated or at least significantly reduced. However, well-architected applications that adhered to best practices and did not necessitate the weakening of any security would have been the better approach and could have allowed applications to be installed and deployed without the need for any re-packaging. The ongoing re-packaging of applications within major organizations using Windows could have been completely avoided through the definition and adherence to industry standards at the platform level from the outset. The situation cannot easily be addressed now, and this remains an ongoing organizational challenge for most large organizations, which continues to incur costs.

Challenge 1: Anatomy of a Windows application

When thinking about the optimum approach for deploying and maintaining an individual application (or a suite of applications) to a set of users, it is important to be aware of how Windows applications can be packaged, installed, and maintained during their lifecycle. By having this insight, it is possible to make informed decisions about the most optimum approach that can be used to deliver an application given the broad spectrum of options that are available on the market today.

At a very high level, a Windows application package, post-installation, is a collection of files and possibly metadata stored in the registry. The files associated with an application might be application binaries installed by the application package or binaries subsequently downloaded when it is launched. In addition, they can be a source of configuration settings for the application. Regardless of the type of file, when an OS is powered down, all that persists of an application are the files and any metadata stored in the Windows registry file.

The heritage of Microsoft Windows is deeply tied to the history of the personal computer (PC). When PCs were first launched, disk space was expensive. Methods to minimize the amount of disk space required for an application were important. The ability to share application binaries between applications such that the overall amount of required disk space reduced was considered a positive outcome during the initial design of Windows. This led to the use of dynamic linking with Windows applications, achieved by dynamic link libraries (DLLs).

In contrast, the use of static linking with Windows would have led to large application binaries that consumed larger amounts of disk space, and this was clearly misaligned with the objective to reduce the amount of disk space required by an application.

Because DLLs can be shared between Microsoft Windows applications, they aren't considered self-contained. This disadvantage leads to multiple complications when packaging and deploying applications and maintaining an end user Windows estate. The opportunity for a cross-dependency to exist between applications sharing a DLL, or even a chain of dependencies between applications, is very high on the Windows OS, and even more so in a large enterprise maintaining thousands of application packages.

The later sections of this document help define a taxonomy for Windows application binaries that can be used to provide a framework to help determine how to deploy and manage individual binary packages.

For completeness, Microsoft has documented a list of advantages for the use of dynamic link libraries here: Advantages of Dynamic Linking.

Portable Executable format

Microsoft Windows application binaries (EXE, DLL, OCX, SYS, MUI, and so on.) are stored using the Portable Executable (PE) format. This format has been used across other OS' in the past, but is predominantly used by Microsoft Windows. The PE file format is structured to allow each binary to be mapped into memory in an efficient way when it's required by the dynamic linker. This allows DLL, OCX, and other Windows binaries to be shared across multiple applications simultaneously when the dynamic linker loads them because of an application being launched by an end user.

Common shared runtimes

While the PE format provides an overall structure for individual applications, application runtimes are frequently made available as a package because they provide a common set of base functionalities used across many applications. This can prevent the need to package the runtime with each application, but it can also ensure that there exists the ability to consider a set of runtime binaries as a discrete version. The approach of bundling multiple runtime binaries into a single package allows updates to a runtime version and the initial deployment to be handled gracefully. Examples of common shared runtimes include Visual C++, Java Runtime, and the Microsoft .Net Framework.

Application dependencies

While common shared runtimes can be considered application dependencies, it is also important to consider that Windows applications can have dependencies between them. These dependencies must be tracked to ensure that users are able to access the full set of functionalities present within an application. For example, in a typical business, applications can create business reports. These business reports are frequently created in a widely-used file format so that they can be broadly accessed. In these instances, a widely-used file format is the Portable Document Format (PDF) and therefore, because the applications creating reports in PDF format frequently can't display PDF files, an alternative must be used. If a PDF file viewer application is installed, then users can use it to view the report. In this instance, the reporting application has an application dependency on the PDF viewer application so that the full functionality of the application can be used.

What happens when a Windows application is launched?

To understand how applications can coexist on an OS while also providing the possibility for an application conflict to arise, it is important to be aware of how non-UWP (Universal Windows Platform) applications are launched on Windows. UWP applications have a different lifecycle from non-UWP applications because non-UWP applications are either running or not running. In contrast, UWP applications can be in multiple states including Not Running, Suspended, Terminated Running, and Closed By User. These states and the differences between applications are explained in more detail here: Windows 10 universal Windows platform (UWP) app lifecycle.

This section provides a high-level technical view for how applications are launched when a user clicks on an icon to start them. For a Windows-based application to start, several events must occur first. When a user starts an application via an application shortcut, Run command prompt, or command line, the following events occur in sequence:

1. The OS creates a new process by calling the `CreateProcessA` or `CreateProcessW` Windows Win32 API, and an initial thread is created which includes the stack, runtime context, and thread object.
2. The application code is loaded into memory and a process object is created in the kernel.
3. Dynamic link libraries are loaded into memory if the application uses them.
4. Memory for items, such as data and stacks, is allocated from physical memory and mapped into the virtual address space.
5. The application begins by starting the initial thread.

> **Note**
>
> Note: For point 3 above, Windows follows a deterministic approach for loading DLLs into an application, which is documented here: Dynamic-Link Library Search Order. The deterministic approach heavily influences the way in which application delivery and packaging needs to be handled natively on Windows. This is because without the use of virtualization, redirection techniques, or application shims (Shim Database (SDB) Files) Windows behavior cannot be changed, and therefore application packaging standards and the associated packages must factor in the native DLL search order.

Issue:

When considering the use of DLLs, the PE format, common shared runtimes, application dependencies, and what happens when a Windows application is launched, there is a single major issue that arises from the dynamics across these areas, and this is the lack of determinism.

There is a lack of determinism around which DLL files an application will load when multiple applications have been installed that share the same DLLs. This is because when users have the opportunity to have a bespoke set of applications installed on their machines based on their role or job function, it is not possible to determine which version of a DLL might be present on a machine. It is important to be aware that the DLLs have no built-in mechanisms for backward compatibility, and therefore a newer or indeed an older version of a DLL might not implement the functionality expected by the application, resulting in a broken application. This issue is commonly known as "DLL Hell."

In practice, this situation may arise because different users installed applications in different orders, or perhaps the user changed their job role and now requires access to a different set of applications than they did previously while using the same machine. Therefore, applications have been installed in different orders than perhaps they have been tested and verified to work in for a specific department or job role.

Resolution

Given the issue with "DLL Hell," how can this be overcome?

- Firstly by being aware of the issue, how it manifests itself, and what the impact is within an organization's environment. The taxonomy outlined in the *A Taxonomy for Windows Software Binaries* section of this document can go a long way in raising awareness of this issue and helping identify software binaries that might exacerbate the situation.
- Secondly by taking steps to mitigate its impact. It's not possible to eliminate "DLL Hell" from a dynamic end user environment where users are free to install applications from a large corporate application portfolio, but the impact can be reduced. The steps outlined in the *Applying the Taxonomy to AWS EUC Services* section later in this document can reduce its impact.

Challenge 2: Packaging formats for binaries

The Windows OS, by virtue of its long heritage and multiple versions, has seen a wide range of software binary packaging formats employed to assist administrators and users with the

installation of applications by ensuring all components are available for an application to be successfully launched and interacted with by end users.

The following two lists capture the packaging formats that Microsoft has released over time, along with a list of some of the non-Microsoft packaging formats that are available for use on the Windows OS.

Current Microsoft packaging formats for the Windows OS include:

- **MSI (Windows Installer)** — Introduced in 1999 with Microsoft Office 2000.
- **MSU (Windows Update Standalone Installer)** — Introduced with Windows Vista in 2006/7 (see Description of the Windows Update Standalone Installer in Windows)
- **Universal Windows Platform (UWP)** — Utilizes the APPX file extension, and is underpinned by Open Packaging Conventions (OPC) and introduced with Windows 8 in 2012.
- **App-V 5** — Utilizes the APPV file extension, is also underpinned by OPC, and is introduced with version 5 of App-V in November, 2012.
- **MSIX** — Packaging format that was introduced at Build 2018 by Microsoft and is available in Windows 10 1809 and above (see What is MSIX?).

Some of the non-Microsoft application packaging options include:

- **Chocolatey** — Chocolately is a software management solution that uses a specific packaging format compiled into NuSpec files.
- **NuGet** — NuGet is an open-source package manager, actively developed by the .NET Foundation and supported by Microsoft for the deployment of .NET applications.
- **OpenWrap** — OpenWrap is a package manager for .Net applications recommended for the deployment of runtimes.
- **XCOPY deployment** — XCOPY deployment for .NET applications essentially involves copying files from a source location to the destination to install an application. This approach is still advocated by Microsoft today due to its simplicity (see Deploy an ASP.NET web application by using Xcopy deployment).
- **Self-extracting** — Various types of self-extracting installers have been created for applications on the Windows OS over the years and there is no single standard that these conform to.
- **Proprietary** — Outside of the self-extracting approach to packaging and installing applications, there is an indeterminate number of proprietary packaging formats for applications that have been created and used by software vendors for the Windows OS over many decades.

Issues

When considering application packaging formats, the issues that arise from the use of multiple formats, and the inherent limitations in some of these formats, there are multiple issues to consider:

- **Complexity** — The broad spectrum of different application packaging formats available from different vendors leads to complexity. While de-facto standards such as Windows Installer have emerged, these are not exclusively used by all vendors and suffer from complexity being introduced because of the broad use of custom actions within Windows Installer files by software vendors. Custom actions can be opaque in terms of their resultant impact to a Windows OS because they can be implemented within custom DLLs authored by the software vendor.
- **Skillsets** — Because so many software vendors supply their software in different formats, it can be difficult for an organization to manage an estate that leverages multiple formats because of multiple skillsets being required.
- **Visibility** — Multiple formats present the challenge that it is difficult to discover and therefore manage the files and versions of files across different formats within a single toolset.
- **Determinism** — Custom actions employed with Windows Installer files, and how these actions function across a group of machines, is non-deterministic because of the potential for changes to have been made to a collection of machines in different ways over time. The result of custom actions being evaluated through the installation logic employed within proprietary code cannot be predicted across an enterprise's machines. As outlined in Challenge 1: Anatomy of a Windows Application in this document, DLLs are loaded into memory based on a set of rules and while the rules are predictable, the state of a machine prior to installation is not. Therefore, different machines can result in different sets of software binaries being installed and available for the application to use. The net effect is increased complexity around the management and deployment of applications to Windows machines.

Some packaging formats define not only the way in which applications are released before installation but also the format in which they are stored on disk for consumption by users. A good example of a packaging format that defines both the way in which an application is packaged and also the way in which is it is stored on disk is Microsoft MSIX.

For Microsoft MSIX, this can be considered both an advantage and a disadvantage. From an advantage perspective, self-contained application packages have the option to either see the Windows file system as it is recorded in the Master File Table (MFT) or be provided with a

virtualized view specific to the application. This allows applications to be prevented from running into application conflicts because they have no visibility of the conflicting resources, such as conflicting registry keys or values and files on the file system. However, the disadvantage is that the application can be prevented from having visibility of a dependent application (for example, database middleware, PDF reader applications, and so on), thus a decision needs to be made: to provide visibility of the dependent application or not.

Resolution

The following points consider how to address this challenge:

- Reducing the complexity of application packages can be a large challenge, and within large enterprise environments, complexity is normally addressed through the implementation of standards that ensure there is strong governance in place around application packaging to enforce the standards. An enterprise might choose to create and maintain their own internal application packaging standards against which all applications need to be packaged to comply. Within the defined standards, there might also be a standard that prevents the re-packaging of vendor-supplied Windows Installer packages, and these packages might have badly-written custom actions embedded within them that break the standards. In this case, the enterprise needs to decide whether to re-package the application to remove the custom actions or grant an exception.
- The scarcity and cost of application packaging expertise means that an enterprise should attempt to minimize the amount of complexity introduced into their organization by supporting as few software packaging formats as possible. This helps with cost management and also ongoing operational efficiency because of common shared standards being in place to simplify software package maintenance and auditing of the application portfolio.
- Visibility of application binaries across multiple software packaging formats is a challenge when attempting to identify versions of shared files used by more than one application. As outlined previously, without understanding the files that are shared between applications, the opportunity for "DLL Hell" to arise and not be addressed increases significantly. Supporting as few packaging formats as possible allows an inventory of application binaries to be more easily created and maintained.
- Determinism, as it applies to being able to assess which runtime files are installed on a machine based on the applications installed, is difficult to address. It depends on the order in which applications have been installed and the deployment technology used to install the applications. Therefore, the insight that increased visibility of the application binaries and their respective

versions provides can considerably mitigate the impact that can arise due to unpredicted application incompatibilities.

Challenge 3: OS application delivery approaches and initiators

This challenge captures the details encountered when considering when to deploy applications and how to initiate the installation of the applications. With the Windows OS, there are many approaches available for both deploying the OS and the applications that will run on it. The flexibility is large, but by carefully considering the impact of the decisions prior to deployment, the user can take an optimized approach for deploying and maintaining applications. The following sections outline the high-level platform lifecycle for end-user-facing services such as WorkSpaces and AppStream 2.0, along with the initiators that can be used to start installing applications.

Platform lifecycle

When defining the platform lifecycle, the full range of deployment options available for use with the AWS EUC services were considered to ensure that all appropriate options were evaluated. The range of options for Windows platform deployment includes both license-included options (specifically, WorkSpaces and AppStream 2.0 instances underpinned by the Windows Server OS) and Bring Your Own License (BYOL) options (specifically, WorkSpaces underpinned by Windows 10). For Windows 10 specifically, the range of deployment options for initial OS deployment (specifically, not OS feature update releases) considered included:

- **Modern deployment** — Windows Autopilot and in-place upgrade.
- **Dynamic deployment** — Subscription activation, Azure Active Directory with Mobile Device Management (Azure AD/MDM), and provisioning packages.
- **Traditional deployment** — Bare metal (image-based and scripting-based), refresh, and replace.

All these options were discounted from consideration for the WorkSpaces and AppStream 2.0 services, except the bare metal traditional deployment approach, leveraging imaging and scripting because they're targeted at use cases that aren't relevant for these services or that have dependencies on other services that customers might not subscribe to.

The following figure is a high-level representation of the image-based bare metal deployment approach for the Windows OS. It is important to consider that other options exist to deploy and deliver applications to a Windows OS post-OS deployment, such as streaming, virtualization, and even the running of application binaries from a network share.

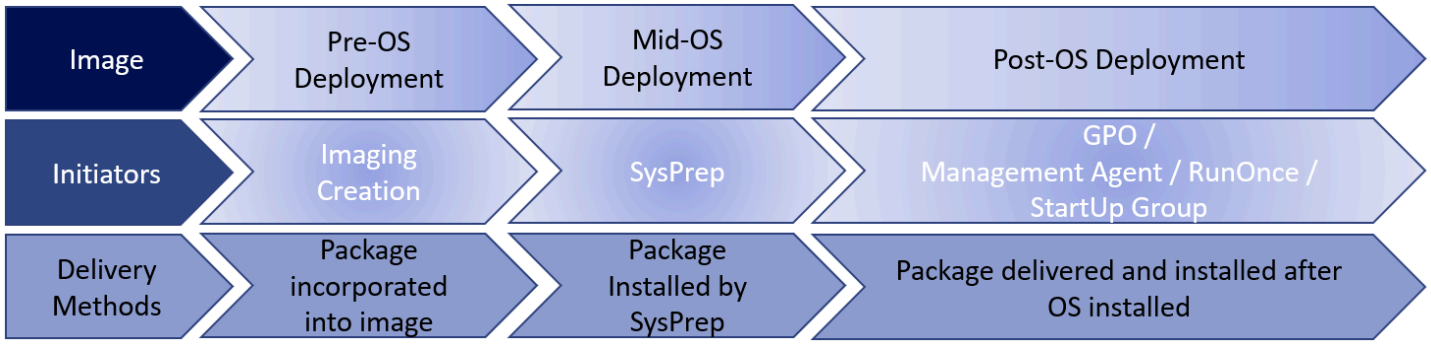
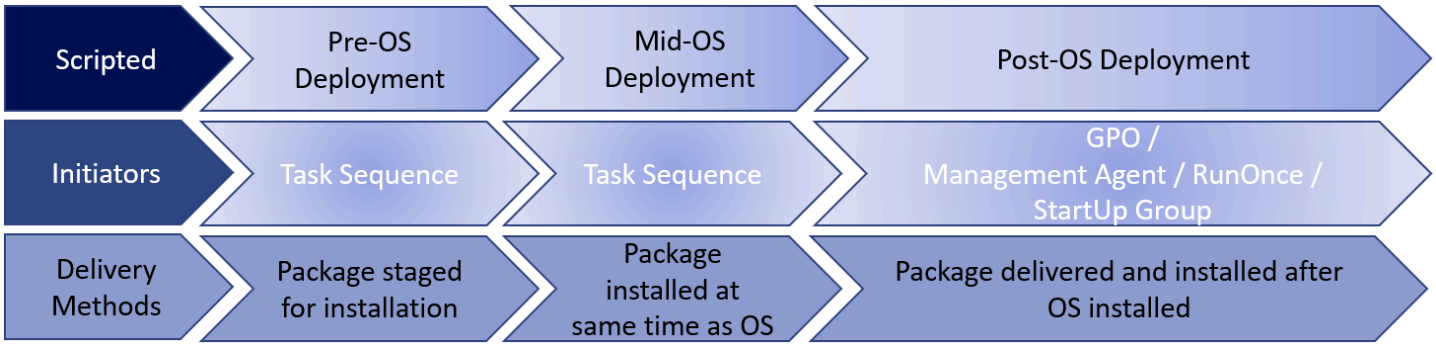


Image platform lifecycle application delivery — initiators and delivery methods

The preceding figure shows the platform lifecycle for a Windows OS being deployed via an image. As the platform transitions from pre-OS deployment through mid-OS deployment to finally reach post-OS deployment, initiators can be leveraged to deploy applications. In turn, these initiators align to different delivery methods for installing applications. The range of initiators, delivery methods, and differing suppliers that provide these capabilities provide a rich environment to choose deployment options from. However, this rich environment ultimately leads to complexity if there is insufficient governance in place that defines best practices and specific approaches that can and cannot be used within an organization.

The following figure shows the platform lifecycle for a Windows OS deployed via a scripted build mechanism. As the OS is installed there are multiple opportunities to initiate the installation of applications. The goal is to have a full suite of applications installed as quickly as possible. Then users can be fully productive and not impacted by missing applications.



Scripted platform lifecycle application delivery: Initiators and delivery methods

Although the range of options for installing applications during the OS installation is broad, it is equally as important for a scripted build as it is for an image-based build to ensure that a consistent approach and governance is in place to determine which initiators will and will not be permitted for use within a specific enterprise.

The following sections describe each of the lifecycle states that an image or script-based OS installation transitions through (specifically Pre-OS, Mid-OS, and Post-OS) and the initiators that are available.

Pre-OS deployment

For image-based OS deployment, pre-OS deployment of applications essentially involves the installation of applications into the OS instance that is used to create the image. In the case of WorkSpaces, this would be the WorkSpace used to create the image that is used to create a custom bundle from which WorkSpaces are deployed. For AppStream 2.0, where either an Always-On or On-Demand fleet is used, applications are installed in an image builder, an image is created, and the image is then associated with the fleet. For Elastic fleets, application binaries are copied to a virtual hard disk (VHD) to create AppBlock, and the AppBlock is used to create applications that are associated with an Elastic fleet.

For script-based OS deployment, pre-OS deployment of applications involves the integration of automated installation within the task sequence or script being used to automate the installation of the OS. Because each installation is unique each time, they are more prone to failure when compared to imaging, where the application has been installed only once and has been tested as part of the overall testing of the complete image.

This approach is well suited to large or infrequently-changing applications, installing application dependencies, and common, widely-used applications. For image-based OS deployments, if any of these types of application binary change frequently, then without automation in place, it isn't appropriate to use this approach for application deployment. The image would need to be frequently modified, tested, and released. If the opportunity exists to automate image creation and maintenance, then these overheads can be mitigated to a degree. End user testing of the updated image prior to production deployment would still be required to confirm its validity.

Mid-OS deployment

During image-based OS deployment, there are opportunities to start the installation of applications as part of the end-to-end image deployment process. However, for the most part, the best practice is to include applications within the image to reduce the amount of time it takes to deploy a user's application environment with their application portfolio.

The exception to this approach is where applications do not work with the Microsoft System Preparation (Sysprep) process and therefore need to be installed individually and uniquely on every machine. One example of an application type that faces this challenge is applications that have

a custom approach to generating a unique machine identifier as part of a standard installation. By imaging these applications, the opportunity to generate a unique identifier does not exist, therefore the application must be installed using the vendor's supplied installation binaries on a per machine basis.

For scripted OS deployments, it is possible to use the "specialize" configuration pass of Microsoft Sysprep (see Windows Setup Configuration Passes) to run custom commands by using an answer file. This affords users an opportunity to install application binaries in the middle of the installation of the Windows OS. This can help to accelerate the readiness of a machine for a user by installing applications before their first login. However, not all applications successfully install using this approach. There is a degree of trial and error to determine if all the applications to be installed using this process will install. This approach is constrained by the security context that is available during this pass, which is typically only the *SYSTEM* security context. If an application needs to be installed using a user's domain credentials, then this process isn't suitable.

Mid-OS deployment is useful to ensure that a Windows OS is secure at the point of deployment and is not immediately vulnerable to attack before it is patched. Types of application software to consider installing during mid-OS deployment include security products such as antivirus and data leakage products and also the application of Windows security templates to ensure that the OS is locked down before a user first logs in. Other types of applications to consider installing during this phase include software that can facilitate the installation of other software such as runtimes, application delivery, and patching toolsets.

Post-OS deployment applies equally to both image and script-based OS deployments since the capability to deploy and deliver applications to the OS once it has been installed can use the same techniques regardless of how the OS was deployed.

Although the delivery of applications post-OS deployment can provide advantages by reducing the number of applications installed and maintained within the image or OS deployment task sequence, choosing to deliver applications post-OS deployment does has some disadvantages to consider.

For example, choosing to deliver applications post-OS deployment means that each user's full application portfolio is not immediately available on the machine they will access. This might impact the end user, and the scale of the impact depends on whether the user logs into the newly-provisioned machine shortly after it is provisioned or sometime after. For users to be fully productive, they need all their applications to be installed. The time it takes to deliver a user's full portfolio of applications must be as short as possible to reduce the impact to their productivity. If

the installation of the applications happens outside of business hours prior to first login, then the impact can be eliminated.

The use of post-OS deployment initiators, where applications can be installed at any time on demand by end users, leads to a situation where two users that are entitled to and have installed the same applications on two different machines can have different issues with applications. This situation can arise because the applications on the two machines might have been installed in different orders. Consequently, the files needed by the application to start might be different versions on the two machines, resulting in differing experiences and issues.

Issues

- **Complexity** — As outlined above, there are many options for choosing when and how to install application software as part of the overall OS platform lifecycle. There are many tools, techniques, and initiators available as options. Choosing the correct initiator and approach depends on the OS deployment type, platform, and business requirements. Ultimately, due to the vast array of options, governance is required within an enterprise to ensure that the smallest number of options are employed to satisfy the business requirements, thus fully reducing complexity to the largest extent possible.
- **Timing** — Poorly-timed application delivery can impact user productivity. For example, choosing to install most of a user's application portfolio post-OS deployment can impact a user's ability to work if they are granted access to their machine before their applications have been installed. Instead, consideration should be paid to either not grant the user access to their machine until it is ready or use a different OS deployment technique (for example, use imaging over scripted installation to ensure all applications are available when a user first logs into a machine) and initiators that would allow the applications to be installed earlier (for example, mid-OS deployment).
- **Differences in determinism** — Choosing to incorporate applications into an image rather than installing them individually on a per machine basis leads to a more deterministic and consistently reproducible application delivery approach since the applications can be installed and validated as part of a test image deployment prior to production deployment. The use of initiators (whether they be pre, mid, or post-OS deployment) result in a higher risk deployment (that is, less deterministic) because the installation of individual applications is more likely to fail compared to an image deployment.

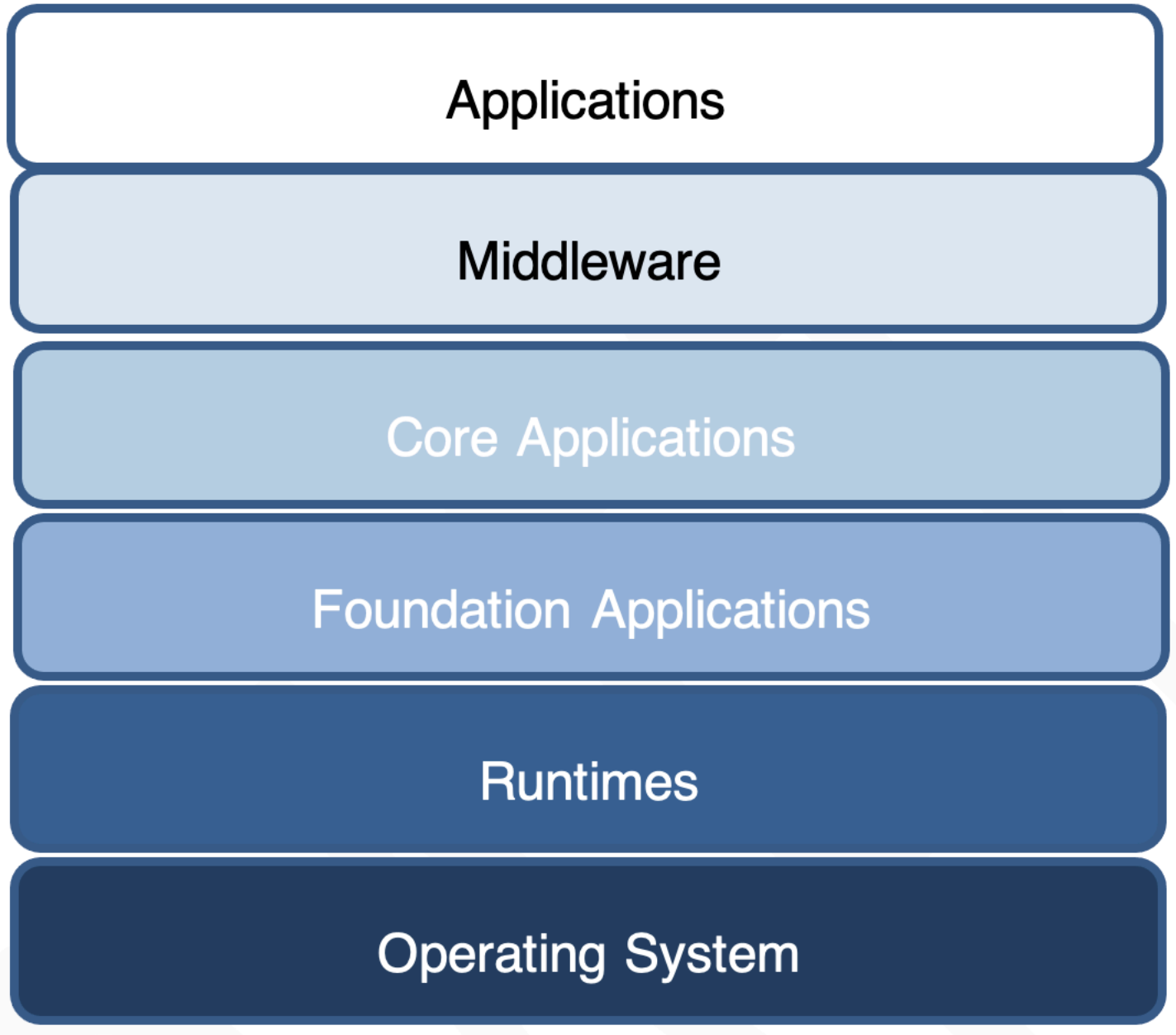
Resolution

Three issues were identified against challenge 3. How might these be resolved?

- As outlined in the resolution section for Challenge 2, complexity within an enterprise can be avoided or reduced through the introduction of standards and a strong governance process to avoid introducing unnecessary complexity while also recognizing that applications are required by the business to support its ongoing operation.
- The timing of the delivery of applications is essential within any EUC service to ensure that users can be productive as early as possible. However, choosing to deliver applications too early within the lifecycle of an OS can lead to integration challenges and potential unnecessary complexity because application installers are rarely written to support the installation of applications during the installation of an OS. Ultimately, a balance must be struck to ensure the correct ratio of complexity and user productivity.
- The differences in determinism associated with different application delivery approaches are influenced by balancing the complexity introduced by incorporating and maintaining an application in an image versus installing an application post-OS deployment, which leads to a less complex image. There are advantages and disadvantages for both approaches and these are explored later in this document in Applying the Taxonomy to AWS EUC Services.

A taxonomy for Windows software binaries

To facilitate the assessment of software binaries on the Windows platform and help inform the process used to determine the optimal application delivery approach, a taxonomy has been captured in the following figure. It is important to note that the following figure is not necessarily a representation of how applications are installed on an OS or how they interact with each other. Each "layer" of the taxonomy is discussed under the subsequent sections.



Windows software binary taxonomy

OS binaries

OS binaries are the compiled binary code provided by Microsoft for the Windows OS. To remain secure, stable, and reliable, Microsoft releases OS patches to address issues in the OS.

OS updates are binaries released by Microsoft to patch stability or functionality issues present within the Windows OS. These can be provided by Microsoft as private fixes as an ad-hoc release or incorporated into monthly quality rollups.

Microsoft OS updates are generally packaged as an update package in an .msu file. The contents of an .msu file are documented by Microsoft in KB934307 (see Description of the Windows Update Standalone Installer in Windows), but essentially can contain one or more .cab files that can be used to store application binaries, that in turn will be used to patch the OS.

OS security updates

OS security updates are binaries released by Microsoft to patch security and reliability issues in the Windows OS. These are released on a frequent basis by Microsoft and can align to a regular monthly or ad-hoc release cadence, depending on the severity of the risk associated with the issue to be addressed.

OS feature updates

OS feature updates for Windows 10 are typically released twice a year by Microsoft as part of the Windows-as-a-Service (WAAS) service. For Windows 11, an annual cycle has been adopted for feature updates. These are large updates and frequently replace a large portion of the OS binaries with new files.

Runtimes

Runtimes are files required for the day-to-day operation of the OS and applications that reside upon it. They might be required to support applications (for example, software distribution, anti-virus) or other runtimes (for example, visual C++ DLLs, .Net framework, Java runtime environment, and so on).

Shared runtimes afford users an opportunity to save large amounts of disk space on a Windows OS with many applications installed on it that can benefit from a common runtime. However, where applications are only compatible with a specific version of a runtime, the updating of the shared

runtime can result in the application failing to launch or operate as expected. In these instances, the Windows OS has natively adopted multiple approaches to resolve this specific issue, and these are discussed in the Microsoft Windows 10 Assessment and Deployment Kit (ADK) (see Download: Windows 10 Assessment and Deployment Kit (ADK)) and its associated documentation.

Foundation applications

Foundation applications are not a native Windows concept, but are a useful concept to leverage within a managed Windows environment to group similar types of applications. Foundation applications are defined as applications that are included in a base OS image. They are typically used by everyone within an enterprise (for example, a web browser, anti-virus, software installation agent, and monitoring agent). They provide a baseline set of functionalities specific to the enterprise over and above that offered natively by the Windows OS, and must be installed to comply with an enterprise's specific policies. These applications are separate and distinct from core applications, which are defined in the next section.

Helping to group common applications that provide management, security, baseline functionality, or application delivery capabilities to the enterprise's base image is important. Understanding the baseline can be extremely useful for identifying and listing the applications required to be installed in all OS images if imaging has been employed.

Core applications

Core applications are defined as those applications that are used by a high percentage of users within a specific enterprise (for example, productivity suite, PDF viewer, file compression tool, and so on).

These applications may be installed on to the same set of machines that foundation applications are installed on. However, there are instances when this may not be preferable. For example, productivity suites are expensive to license on an ongoing basis. If a proportion of the userbase using Windows do not need these applications, then license costs can be reduced by separating the two sets of applications into Foundation and Core.

Middleware

Middleware is defined as computer software that connects software components or applications. Middleware sits "in the middle" between application software that may be working on different

OS'. A common example is database middleware. For the purposes of application delivery, middleware is generally defined as a dependency of an application (for example, an application has a dependency on database middleware to be able to interact with a specific database server).

Applications

Applications are defined as computer software designed to help users perform a particular task. These include the binaries and settings needed for the operation of the application. Applications can have dependencies on other applications, middleware, runtimes, and the OS.

Applications can be considered the most frequently used unit when deploying application binaries. This is because applications are the most frequently-used units interacted with by end users and they are widely understood by both end users and the organization that helps to deliver them to the end users. In contrast, non-application dependencies of an application, such as runtimes and middleware, frequently remain unseen by end users as they have no need to directly interact with these.

Examples of applications include a productivity suite such as LibreOffice and complex graphical applications used for Computer Aided Design (CAD), but also include what might otherwise be considered a "plugin", such as Microsoft Office Add-ins.

Web applications

Web applications are a sub-type of application accessed using a web browser, and are hosted on a server. Web applications can have a dependency on installed software in some cases, and therefore it is important that the OS running the web browser is able to satisfy these dependencies for a user to successfully use the web application. Such dependencies may be plug-ins, runtimes, and multimedia player software, and so on.

Applying the AWS Well-Architected Framework to Windows-based end user application delivery

The AWS Well-Architected Framework helps guide the decisions that are made while building systems on AWS. Using the Framework helps ensure that architectural best practices for designing and operating secure, reliable, efficient, cost-effective, and sustainable workloads are applied in the AWS Cloud. It provides a way to consistently measure architectures against best practices and identify areas for improvement. In the context of Windows-based end user application delivery, aligning considerations and guidance around factors that can influence the choice of application delivery approach for an application or collection of applications to the AWS Well-Architected Framework ensures that design principles associated with each pillar have been considered.

The AWS Well-Architected Framework has six pillars that need to be considered when choosing to deliver Windows applications to end users. The following six sections and their associated sub-sections explore these pillars in detail but do not represent an exhaustive list.

Operational excellence pillar

The sub-sections that follow outline some operational factors that can influence the choice of application delivery approach and the optional application of a specific toolset to solve common application delivery challenges.

End user support

The likelihood of an issue affecting users, as well as the impact on both end users and the support organization, will need to be assessed before releasing software updates. This might involve a risk assessment to determine the size of the risk associated with the software release. Two risk-sizing dimensions should be applied in terms of the likelihood that risk might occur, and the size of the impact on the business that the risk would have if it does occur.

While risk management is important, other organizational constraints, such as the capacity of the organization's incident management process as well as the Mean Time to Response (MTTR) metric, need to be considered. If the organization's incident management and service desk become overwhelmed by a large influx of incidents because of a failed software release, it could have a large impact on the organization's ability to continue trading. If the MTTR within an organization is currently high, then one should consider whether it is currently an optimal time to release new software.

To help mitigate the identified risks and constraints outlined previously, consider software deployment strategies to minimize the impact on the organization. Phased deployment is frequently used to slowly introduce new software into large organizations with discrete groups of users receiving the update at different times. The ability to roll back software changes is frequently sought as part of a change management or release management process, although this is not possible with every change. The AppStream 2.0 service provides a very strong capability in this area through the ability to maintain multiple image versions and VHDs and provide a quick rollback to an earlier version of the software if required. Managing the rollout of new images in WorkSpaces can be more complex due to the opportunity for individual users to install their own applications in their workspace, thus diverging away from the application set included in the baseline WorkSpaces image that would be re-applied if the user were to rebuild their workspace.

End user training

Training may also be a consideration when determining an organization's readiness to use the software. If the update to software involves a new product or a new version of a product that requires training, then end user training is a dependency for the release of the software.

How users are trained depends on the skillset of the users receiving the software and the capabilities available within the organization to deliver training. At the very least, it's important that the operational teams within the organization are informed, that the internal knowledge management system is updated, and that, if appropriate, the operational staff are trained on the software close in time to the introduction and deployment of the software. This training may also need to be extended to the software users. How the training is delivered also depends on the resources available within the organization to facilitate the introduction of the new software. Resources such as personnel to run interactive training classes and provide training videos or documentation, will need to be assessed.

To facilitate the introduction of new software, it is important to consider the identification of champions across an organization. Champions can reduce the resistance to change that some organizations may experience when introducing new software, and can also act as a "voice of the customer" throughout the duration of the project or program running to introduce the software. In this role, the champions can ensure that every use case for the new software being introduced has been considered, and that the likelihood of issues following the introduction is reduced.

Service operating model

The AppStream 2.0 and WorkSpaces services have different operating models, where the AppStream 2.0 service is non-persistent, and the WorkSpaces service is persistent. With these

differences in mind, the choice of software deployment mechanism to be used with each service is heavily influenced by the underlying operating model.

AppStream 2.0 On-Demand and Always-On fleets use the concept of an image, a copy of which is represented each time an AppStream 2.0 instance is launched. Therefore, application updates can be incorporated into the image and delivered to the entire user base simply by the users logging out and logging back in. Elastic fleets use a VHD file with the application binaries included; therefore, application updates can be delivered in the same way with users logging out and back in to get updated applications. This process can work efficiently with a small number of applications; however, for more complex scenarios, separate fleets or additional delivery methods may need to be considered due to application dependencies and potential requirements for different conflicting versions of runtimes.

WorkSpaces uses a persistent operating model and therefore does not benefit from the single point of update that AppStream 2.0 has. The choice of deployment mechanism is therefore dependent on the type of software binary, its impact to the integrity of the WorkSpaces instance, and the associated user's productivity. Software deployment and installation tools can be used to push updates on a frequent basis to WorkSpaces instances. Alternatively, there are many AWS partner solutions that can be used to deploy software updates to WorkSpaces instances, and can minimize the impact of updates by using application isolation or virtualization.

Security pillar

Security is a very broad area, and so this section is limited to considering how the criticality of security vulnerabilities that exist within software binaries in the taxonomy can influence the choice of application delivery approach.

Updates to software binaries that have been initiated due to a security vulnerability being present within existing software need to be assessed using a standard threat management and response approach. This triages the threat to the organization to determine the size of the potential impact, determine whether there are any mitigating controls, and ultimately formulate the response to the threat, including its priority.

Depending on the risk rating of the vulnerability associated with the security update, the organization needs to determine a suitable approach to deliver the update. This may require urgent delivery in the case of addressing Zero-Day vulnerabilities, or a slower response where an organization has determined that they are not vulnerable to a threat due to mitigating or compensating controls.

The choice of application delivery approach is also heavily influenced by the type of mechanism that was originally employed to deliver the software. If the software was deployed as part of the base image, then an OS or application patch management toolset will need to be leveraged. If software was deployed post-OS deployment using an application delivery or virtualization toolset, then the tools can be used to deliver incremental security updates to address security vulnerabilities that are restricted in terms of their impact to a single application.

Reliability pillar

The reliability pillar of the AWS Well-Architected Framework defines five design principles. This section focuses on two of these: "Scale horizontally to increase aggregate workload availability" and "Automatically recover from failure."

Scalability must be considered in end user application delivery. Fortunately, the AWS-managed EUC Services (that is, AppStream 2.0 and WorkSpaces) provide an underpinning capability that ensures that contention for compute resource is minimized, which means that users get the highest chance to have a good user experience. Clearly, different applications have differing compute requirements in terms of processor, and memory or disk I/O, and instances must be sized correctly for the workload. However, the avoidance of shared compute resources encountered with multi-session services and platforms provides a strong foundation from which to provide a high-quality, end user experience.

The reliability of a workload depends on several factors, the primary of which is resiliency. Resiliency is defined as "The ability of a workload to recover from infrastructure or service disruptions, dynamically acquire computing resources to meet demand, and mitigate disruptions, such as misconfigurations or transient network issues." When considering resiliency in the context of end user application delivery, the underpinning AWS services employed to deliver the applications should be considered to determine whether the services and the architecture employed to deliver the services satisfy the business requirements for resiliency. The time taken to recover a workload (in this case an application) is defined as the Recovery Time Objective (RTO).

Business criticality

If an application is critical to the operation of a business, then the choice of how and when to update it will be affected by its criticality relative to other applications within an organization's portfolio.

Applications that are integral to the daily operation of a business are likely to benefit from being made available using a mechanism or platform that is highly available and resilient.

Other applications that are not critical to the daily operation of a business may co-exist on less-available platforms where the business is able and willing to tolerate an outage of the application for an extended amount of time.

The RTO, Recovery Point Objective (RPO), and business criticality of the application should be factored into the decision process when determining the optimum platform and application delivery approach to use in order to deliver a specific application and ensure the objectives can be met and the business isn't unduly impacted by an application outage.

RPO

Fortunately, the RPO for most end user applications is not dependent on recovering the state of the machine on which the end user facing component of the application runs. This is because the state of the applications, in terms of backend databases or central file stores, is separate from the machine running the application user interface.

However, there are exceptions where the application state and data are held locally on the machine executing the application. These applications introduce challenges that need to be considered within the context of application delivery to ensure that businesses can remain operational. The degree of impact of losing a machine executing the application, and the associated state and data, must align with the RPO defined for the application being delivered with the business consulted to determine the RPO requirements for the specific application within the context within which it is being used.

RTO

The RTO for end user applications will determine how quickly applications need to be available to a business when a disaster occurs that prevents access to applications using the standard access mechanism. If applications are not immediately available on the base platform used to access an application (specifically, applications that are installed or delivered subsequently to the user login using a software distribution tool or third-party delivery toolset), then this can incur a delay that may prevent the required RTO from being met. Therefore, it is essential to determine whether the RTO can be satisfied when considering the aggregate length of time it takes to deliver a specific application (specifically, time to provision or restore the instance that will run the application, as well as the time it takes to deliver the application if it is not included in the base image).

Availability

Availability is defined in different ways as it applies to architecture; from an application delivery perspective, availability can be defined as "the ability of an application to perform its agreed

function when required to do so". Thus, for applications used by different businesses and in different contexts, varying views for the desired availability of an application can emerge, guided by the criticality of the application to the business within the context that it is used. It is certainly possible for a single application to have different availability requirements within a single business, based on the use of the application in a specific department, function, or job role.

From an AWS perspective, both the WorkSpaces and AppStream 2.0 services can be highly available, if required by a business to ensure the ongoing operation of a business. However, high availability for an application can often attract additional costs, and so the level of availability must be appropriate for the application being delivered and the context within which the application is being used (specifically, the business criticality, RTO, and RPO must be considered).

Change management

Change management within end user application delivery applied to the Windows OS is a frequently-occurring process where a mature and robust capability is required to ensure that end users are not impacted by changes and the business continues to operate through the avoidance of any impact that changes could incur.

Change management applied to the updating of components that are shared between applications can be impactful to the business since updating an application dependency shared between many applications can affect more than one application. Examples of application dependencies that can be very impactful include runtimes, middleware, and other applications frequently accessed from inside other applications such as PDF viewer applications or printer drivers.

To mitigate the risks associated with updating shared components, it is important to adopt robust processes that ensure the test coverage for the components is very broad and may require approval from many business lines that need the dependency.

Software release cadence

The cadence of the release of software is a major factor that influences the choice of the software deployment mechanism, and it should not be considered in isolation, but rather in conjunction with where within the software binary taxonomy the binaries for the software component reside. If an application or associated update is released frequently, then overheads can be incurred in terms of application packaging and testing if the application integrates with or is a dependency of other applications. If the application is stand-alone, its impact is reduced.

Changes to runtimes

Different runtimes can have different usage patterns within an enterprise. For example, the standard Microsoft C++ runtimes can be widely used within an organization, whereas for an organization that has very few Java-based applications, the Java Runtime Environment (JRE) can be infrequently used or not required. With the disparity of use across different runtimes, it is important to take into consideration how widely-used a specific runtime is to determine the size of the impact updating it may have within an organization.

Because runtimes sit at the second lowest layer in the taxonomy, they can be widely used by all the layers above, and therefore extensive testing and change management will be required for commonly-used runtimes specifically.

Changes to foundational applications

The approach to follow for both change management and testing will depend on the functionality provided by the foundation application. For example, a foundation application providing a set of security capabilities, such as anti-virus and anti-malware, will require different testing and consideration compared to a foundation application providing application delivery or software distribution capabilities.

Changes to core applications

Core applications are fundamentally very broadly adopted within an organization, and therefore must be tested extensively across an enterprise to ensure that updates do not adversely impact a business. It is frequently the case that core applications are incorporated into a base image to ensure core applications are available to users at first login. Users testing the release of updates to core applications must take into consideration the release of updates to existing machines (for example, WorkSpaces), as well as updates to a base image.

Changes to middleware

Middleware can be frequently shared between multiple Lines of Business (LoBs), with some application vendors stipulating that a specific version be installed in Windows. At times, if an LoB or user needs to use two different applications that have a dependency on the same middleware, but one application mandates that a newer version be used and the other application mandates that a different version be used, a conflict arises that is difficult to address natively within Windows. AppStream 2.0 can help in these instances because it can be used to deliver a single application that uses a different version of middleware than that installed on the user's main

machine. Another approach is to leverage virtualization to overcome this issue, and this can be achieved using a third-party application delivery toolset.

Another issue can arise where an organization itself stipulates that only a single production version of middleware is allowed within the organization. If the middleware is used by two different LoBs that use different applications that require different versions of the middleware, then a challenge exists to either support two versions or work with the vendors of the applications to ensure support for the single selected version. For an application owner within an organization, it can become a significant challenge to manage multiple versions of middleware across multiple LoBs and multiple software vendors.

Image maintenance

Image maintenance is the process undertaken whenever a change is required to a base image due to a change needing to be made in response to an update to software binaries within the software taxonomy. The change may be initiated in response to software binary updates required in any layer of the taxonomy.

The frequency of image maintenance will be determined based on the cadence of the release of new software binaries that need to be applied to an image. At the lowest level, the monthly cadence for the release of updates to the Windows OS dictates that, at the very least, a monthly cadence should be considered.

If updates are restricted to the OS only, then the Managed Image Updates (see Amazon AppStream 2.0 adds support for fully managed image updates) capability of the AppStream 2.0 service can patch the OS and base image associated with an AppStream 2.0 fleet. The process for patching the OS can be automated and scheduled as outlined in this blog post: Scheduling managed image updates for AppStream 2.0.

For the WorkSpaces service, two options are available for ensuring individual WorkSpaces are patched with OS patches. The first approach is to leverage a Windows patching toolset such as AWS Systems Manager or Windows Software Update Services (WSUS) to patch individual machines. The second approach is to create a new custom bundle every month that includes the latest patches and then rebuild all WorkSpaces using the new custom bundle. The second approach may not be appropriate for all use cases since the uniqueness of individual WorkSpaces is lost through the updating of the custom bundle and the rebuilding of a large estate of WorkSpaces could take a considerable amount of time.

Outside of OS patches, application updates may be required more or less frequently than the monthly cadence. Therefore, the location of software binaries within the taxonomy needs to

be considered since updates to runtimes, foundational, or core applications will likely be more impactful across an application portfolio than an update to a single application. The guidance outlined in the Applying the Taxonomy to AWS EUC Services section of this document outlines how to approach updates across the WorkSpaces and AppStream 2.0 services.

Performance efficiency pillar

The performance efficiency pillar, as it applies to end user Windows application delivery, aligns to the capabilities of both the WorkSpaces and AppStream 2.0 services very well to ensure the structured and streamlined allocation of IT and compute resources. Both services are globally available in multiple Regions and allow customers to go global in minutes, and the simplicity of both managed services democratize advanced technologies to make advanced technology implementation easier for organizations.

For the WorkSpaces service, multiple bundle sizes are offered within the service, which allows users' instances to be rightsized and determined on the application portfolio, and the resource requirements of those applications. It is important to consider monitoring the performance of the instances to ensure the ongoing correct allocation of resources based on an individual user's consumption of resources within their own unique application portfolio.

While a WorkSpaces instance may be rightsized initially, a user's resource requirements can vary over time, and the ongoing monitoring and rightsizing of instances can ensure that users are receiving a good user experience and that the organization is not over-provisioning resources for individual users and incurring unnecessary costs. The user self-service capabilities with the WorkSpaces service allow a user to both scale up and scale down their own compute resources at their own discretion, should an organization choose to enable this facility. While this capability can reduce the operational burden on the service desk, it does have cost implications, and therefore needs to be considered carefully, and ideally enabled within an automated service request process.

AppStream 2.0 leverages application auto-scaling to ensure that fleets are rightsized to accommodate the number of concurrent users that need access to a desktop or application using AppStream 2.0 across a seven-day week. Multiple approaches are available to scale a fleet up and down to correlate with the number of business users who need access, and discussion of these is too much to consider here. However, the existence of the scaling policies allows customers to maintain the efficiency of operation of their fleets and across the broad range of AppStream 2.0 instance families, customers can select rightsized instances within specific instance families to optimize both user experience and cost.

As outlined previously for WorkSpaces, it is important to consider monitoring individual AppStream 2.0 instances for compute resources to ensure that the correct amount of memory, vCPU, and network bandwidth, among other metrics, is available to support the business applications that the instances deliver.

Cost optimization pillar

The cost optimization pillar includes the ability to run applications to deliver business value at the lowest price point. In the context of Windows end user application delivery, let's consider some of the areas of application delivery that can have a financial impact to an organization.

Application portfolio

An application portfolio within an enterprise environment is typically complex and multi-dimensional in terms of the aspects of the portfolio that need to be considered from an application delivery standpoint. As this document focuses specifically on end user application delivery using the Microsoft Windows OS, the application portfolio can be influenced by:

- **The number of software packages across the taxonomy** — Having a large application portfolio used by a diverse userbase adds complexity to application delivery due to the requirement to deliver discrete sets of applications to differing userbases within an enterprise. Both the WorkSpaces and AppStream 2.0 services would need to be supplemented with an additional capability to deliver sets of applications if a common baseline image were in use.
- **The type of applications within the portfolio** — Simple applications (for example, a website delivered to a browser) are much easier to deliver compared to applications that need to be installed. If only a web browser is required, then WorkSpaces Web may be a better end user service choice than AppStream 2.0 or WorkSpaces.
- **The complexity of the applications being delivered** — Self-contained applications with no dependencies are much easier to deliver compared to complex application suites that might have multiple dependencies such as runtimes, middleware, and shared plug-ins used between applications.
- **Cross-dependencies between applications** — Having multiple shared dependencies between applications can lead to challenges associated with releasing new versions of the dependency and parent applications since each application that depends on the dependent application needs to be tested to ensure it continues to function correctly with an updated version of the dependency. The more applications that share the dependency, the larger the effort is to test and validate the updated dependency. Third-party application delivery toolsets that employ

virtualization can be used to reduce the need for extensive testing, and mitigate the risk that an updated dependency may result in dependent applications becoming inoperable.

- **The age of the applications being delivered** — Old applications built for previous versions of Windows, or for OS' no longer supported, may not install or run on the latest version of Windows. They may also fail to gracefully co-exist with other newer applications due to resource requirements or incompatibilities.

Timeliness of application delivery

The timeliness of application delivery is directly influenced by the business criticality associated with the application, as discussed above in the Reliability Pillar, but it is also indirectly influenced by the release cadence for the software being delivered. The cadence of software release can impact the requirement for timeliness of application delivery when the ability to deliver releases becomes challenging. This is typically encountered when there are large or frequent (that is, daily, or more often) updates that need to be delivered to end users.

The AWS EUC services can help to address some of these challenges. For example, AppStream 2.0 can help to deliver updates for frequently-updated applications through the refreshing of the image associated with an AppStream 2.0 On-Demand or Always-On fleet, or the VHD associated with an Elastic fleet, with users receiving the update whenever they start a new session. Similarly, AppStream 2.0 can help deliver very large application binary updates in a timely manner through the updating of an image or VHD file and the association of the new image or VHD with an AppStream 2.0 fleet or AppBlock.

WorkSpaces can also help with deploying large application updates frequently through the creation of a custom bundle that includes the updated application binaries within it.

Deploying large application updates to existing provisioned, persistent machines can be challenging, however, the approaches above provide mechanisms to achieve this. Alternatively, third-party application delivery toolsets can be leveraged to optimize application delivery and ensure applications are delivered within the time frames required by the business.

Application packaging

As outlined in the Challenge 2: Packaging formats for binaries section earlier in this document, there are no consistencies in the packaging format that software vendors choose to use to deliver their binaries to the Windows OS. Consequently, many enterprises choose to re-package the

application binaries they receive from vendors in a consistent format to ensure that their own internal standards are complied with for all software distributed to users. While this is beneficial to the enterprise from a governance and security perspective, there are consequences to consider.

Time to market

Because all applications need to traverse the application packaging process within an enterprise to ensure compliance with internal standards, there is an opportunity for the process to become a bottleneck and prevent timely packaging and therefore delivery of applications to users. It is therefore paramount to ensure that the application packaging process can scale to cater to the peaks in demand that may arise for application updates. Without this, the time to market for application updates will become unpredictable and may impact an enterprise's ability to respond to security vulnerabilities if updates cannot be packaged, tested, and released quickly.

Cost to deliver and budgeting

The operation of an application packaging process in support of ensuring that corporate governance and the associated standards are complied with can be costly from a holistic organizational perspective, but it can also be hindered by individual departments or LoBs as well. If departments or LoBs fail to have sufficient budget to ensure their applications are packaged, then the safe delivery of updated applications cannot be guaranteed. If an LoB owns an application that is widely deployed in an organization as a dependency used by many other applications, then the impact can be significant.

Skillset availability

As outlined earlier in the document, the availability of personnel with the skillset required to successfully package applications in a reliable, fast, and compliant manner is likely to prove challenging over time as the skillset required to do so becomes scarcer and more expensive.

Sustainability pillar

The sustainability pillar focuses on environmental impacts (especially energy consumption and efficiency since they are important levers for architects to inform direct action to reduce resource usage). In the context of Windows end user application delivery, within the pillar it is important to consider efficient resource usage for both application testing and delivery on a day-to-day basis. By deploying AppStream 2.0 or WorkSpaces into a separate AWS account, it is possible to review the carbon footprint of these services through the use of the Customer Carbon Footprint Tool.

Image testing

Image testing can require significant effort as well as resources within an organization. Therefore, resource constraints (specifically, people and compute resources) may impose a limitation on the frequency of image updates that can be delivered. To help address this constraint, automation can be leveraged to test components of the image, to minimize the amount of manual effort required and elapsed time using a compute resource to test an update to an image.

The size of the update can also impact the timeline for testing if a monolithic image update with multiple updates to the OS, runtimes, foundational, and core applications have been made within a single update.

Application package testing

In a similar way to image testing, application package testing can require significant effort and resources within an organization. This can also be addressed to an extent using automated testing applied to application packages. However, while automated test coverage can be useful for the main functionality within an application, there are frequent use cases for applications that are not covered and that are unique to LoBs within an organization. To ensure full coverage, it is frequently necessary to ensure the availability of LoB stakeholders to undertake this testing and ensure the testing of LoB specific use cases are included as part of the application release management process.

Day-to-day application delivery

In addition to resource consumption during image and application package testing, the rightsizing of computer resources to deliver a portfolio of applications to an LoB is important to ensure a good user experience, and also to ensure that too much compute resource is not allocated, resulting in an unnecessary sustainability impact. Employing a user experience monitoring toolset to help inform the rightsizing of AppStream 2.0 or WorkSpaces instances is important since not every user of a specific application or indeed every user within the same LoB needs or uses the same amount of compute resource. The employment of these toolsets can help to tailor the size of compute resource based on individual users' requirements.

In addition to the coarse-grained rightsizing of instances for users, where a user needs access to applications within their portfolio that cannot be hosted or perhaps are best not hosted on the same compute resource, it is important to ensure the optimal choice of application delivery approach.

For example, where a user needs to use a 3D CAD application infrequently, rather than provisioning a graphics processing unit (GPU)-accelerated workspace where the user can use the CAD application alongside the rest of the application portfolio, a better choice (from a sustainability perspective) would be to choose to deliver the CAD application using a GPU-accelerated AppStream 2.0 instance for infrequent use and a non-GPU accelerated workspace for daily use. This will ensure that the additional power usage and cooling requirements associated with a GPU-accelerated workspace instance are not incurred daily.

Applying the taxonomy to AWS EUC services

Now that we have defined a taxonomy and outlined how the AWS Well-Architected Framework should be considered in the context of end user application delivery, what value does the taxonomy provide?

As outlined earlier, the taxonomy helps to inform decision making when determining how to deliver application binaries. Specifically, it can be applied when using AWS EUC services, as it allows similar types of software binaries to be grouped together to simplify the management and maintenance of applications, accelerate application delivery, and ensure that users can be productive as early as possible through the timely delivery of applications.

The AppStream 2.0 and WorkSpaces services are the two AWS managed services that can deliver Windows applications to end users. The two services do have some fundamental differences between them that are too large to discuss in this document. However, how applications can be delivered from the services by leveraging the software binary taxonomy defined earlier is considered within this section. Fundamentally, it helps to guide the selection of application delivery type by helping to:

- Decide which applications to include in a base image.
- Choose an optimal application deployment mechanism.
- Understand the interaction between applications and how to deploy them even with application compatibility challenges.

WorkSpaces

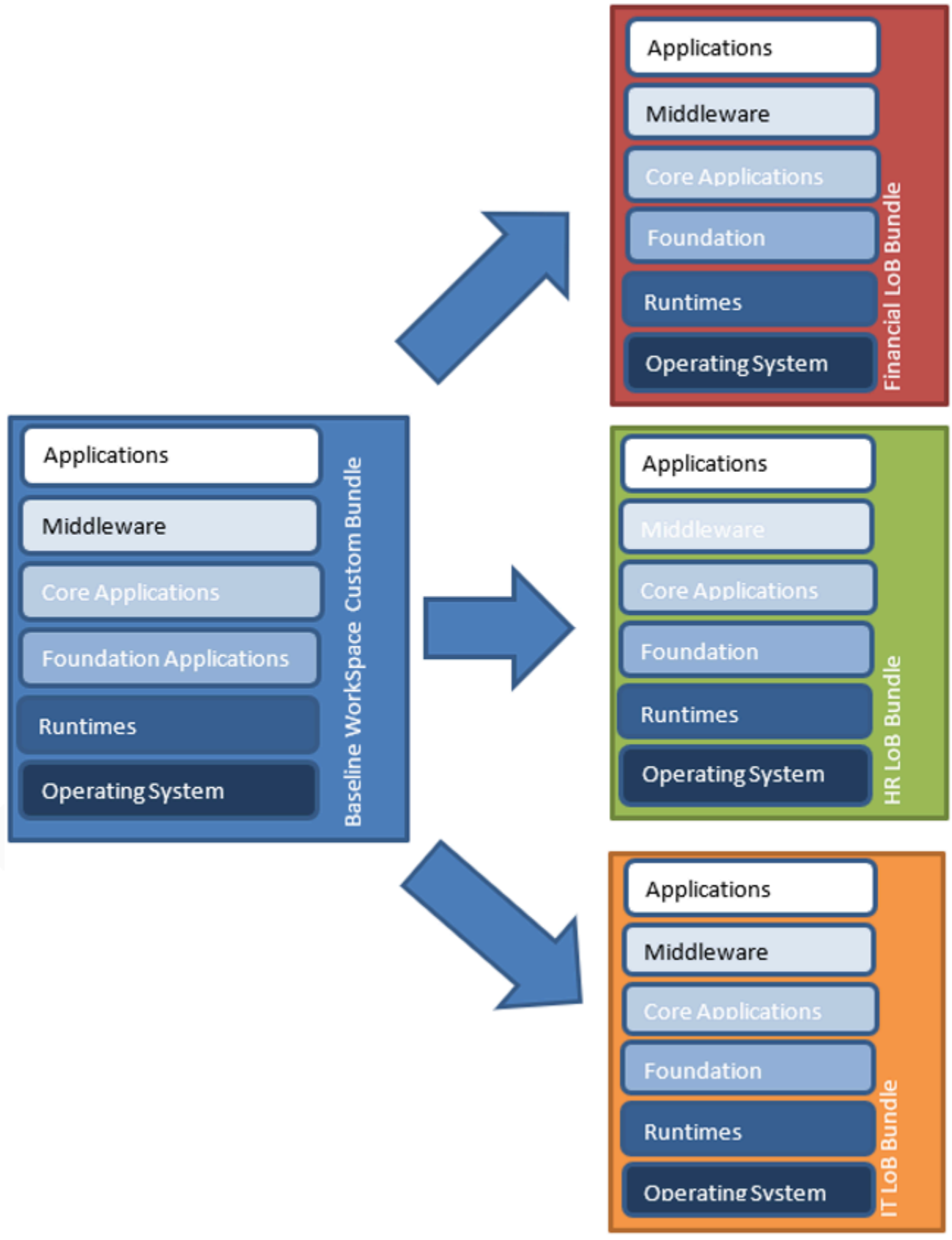
Consider an initial example of a corporate desktop with multiple applications that are to be used on WorkSpaces. In addition to the software included in the corporate desktop, multiple LoBs have specific applications that also need to be used on WorkSpaces.

From a WorkSpaces standpoint, the best approach to create a custom bundle to satisfy this requirement is to install the runtimes, foundation, and core applications into a single WorkSpace, and then use this to create a custom bundle. This then becomes the baseline image upon which other options for application delivery can be considered.

Once a custom bundle has been created, there are three options that can be used to deliver applications specific to an LoB to users.

Option 1

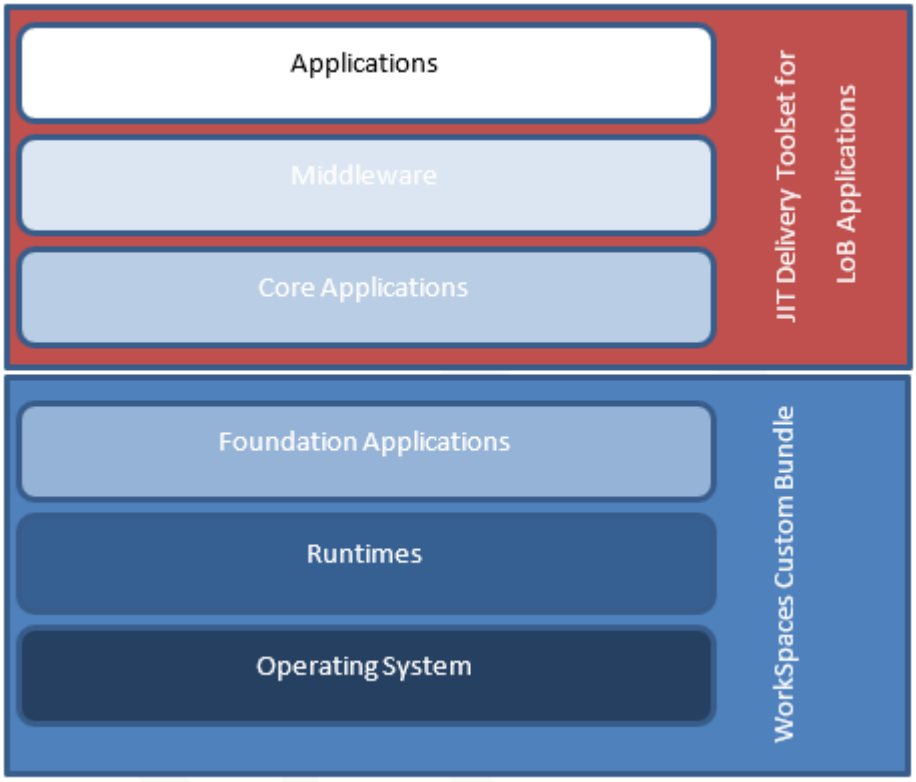
The first option (shown in the following figure) is to create LoB-specific bundles from the original custom bundle. Each LoB bundle would include the additional applications specific to that LoB.


Applications
Middleware
Core Applications
Foundation
Runtimes
Operating System
Financial LoB Bundle
Applications
Middleware
Core Applications
Foundation Applications
Runtimes
Operating System
Baseline WorkSpace Custom Bundle
Applications
Middleware
Core Applications
Foundation
Runtimes
Operating System
HR LoB Bundle
Applications
Middleware
Core Applications
Foundation
Runtimes
Operating System
IT LoB Bundle

Option 1: Create LoB bundles from common custom bundle

Option 2

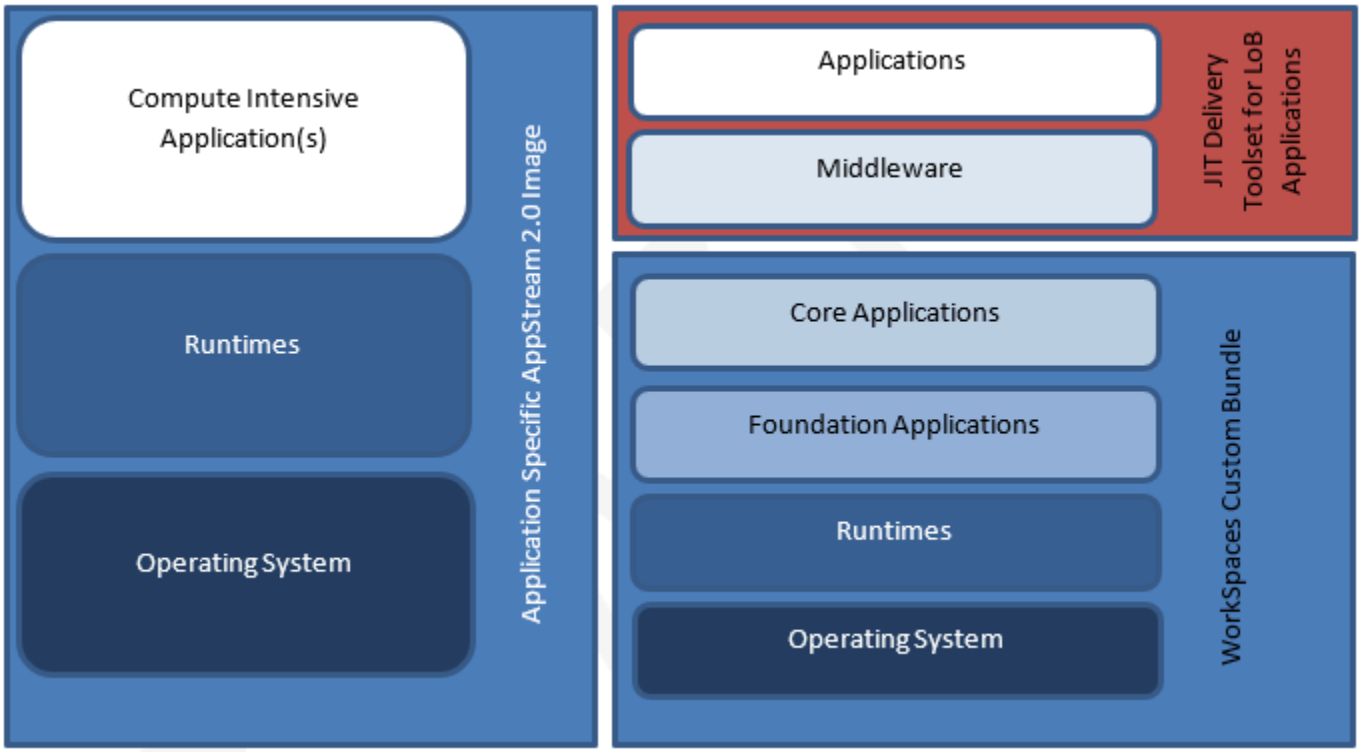
The second option (shown in the following figure) is to use the custom bundle as the base OS, with foundation and core applications included within it, and then use a third-party application delivery toolset to deliver the LoB-specific applications to each LoB user's WorkSpace.



Option 2: Create shared custom bundle and deliver LoB applications to it

Option 3

The third option (shown in the following figure) is to use a custom bundle as the base OS, with foundation and core applications included within it, then use a third-party application delivery toolset to deliver the LoB-specific applications to each LoB user's WorkSpace, and use AppStream 2.0 to deliver discrete applications that need additional compute capabilities and are used infrequently.



Using Amazon WorkSpaces alongside Amazon AppStream 2.0 to deliver a compute intensive application.

Some of the pros and cons for each of the three approaches are considered in the following table.

Table 1 — Advantages and disadvantages associated with WorkSpaces application delivery options

Option	Advantages
1	Common shared runtimes can be tested with applications and a high degree of certainty gained to ensure applications will function. **Result** — Faster and more reliable application delivery and launching.
	Applications are available immediately after login. **Result** — User productivity improved.

Option	Advantages
	Runtimes, foundation, and core applications can be tested together, and confidence is gained that there are no application compatibility issues between applications. **Result** — Faster and more reliable application delivery and implementation.

Option	Disadvantages
1	Image maintenance is an overhead due to an image being used for each LoB. **Result** — Increased administration and testing overhead associated across multiple LoB images.
	Image will go stale and require updating, and the release cadence of the applications may result in this happening quickly. **Result** — Increased administration and testing overhead associated across multiple LoB images.
	Delivery of applications unique to a group of users cannot be achieved without an application delivery tool, and allowing bespoke application deployment on a per-user basis is more likely to give rise to application compatibility issues. **Result** — User productivity may be impaired.

Option	Advantages
2	One common image is used across multiple LoBs, therefore less image maintenance is required. **Result** — Less administration and testing, which frees up time to focus on value-adding activities instead.
	Confidence can be gained over time in the compatibility between the included applicati ons through the deployment of frequent updates to LoB applications. **Result** — Faster and more reliable application delivery.

Option	Disadvantages
2	Applications may not be available immediate ly after first login due to the use of a common base image with no LoB applications incorpora ted into it. **Result** — User productivity impaired.
	Dependency on third parties to deliver applications, leading to incremental cost increases. **Result** — Increased operational expenditure.

Option	Advantages
3	Existing WorkSpaces bundle used for delivery of the majority of users' application portfolios. **Result** — Complexity for delivering the majority of users' application portfolios is minimized through re-use of a common bundle.
	WorkSpaces and AppStream 2.0 instances can be rightsized based on application requireme nts. **Result** — Workspaces are not over specified to accommodate the compute requireme nts of the most demanding application, thus reducing costs. AppStream 2.0 instances are only used when the application is required, thus not incurring any unnecessary costs when the application is not in use.

Option	Disadvantages
3	Two services with different application delivery approaches and instance sizing are required to deliver the entirety of the users' application portfolio. **Result** — Two services must be operated and supported with images created and maintained for both, adding to the complexity of application delivery.

Recommendation

It is not possible to make a single approach recommendation that is the most suitable choice for all organizations. The preceding table shows that there are both advantages and disadvantages associated with the three options explored, and these should be evaluated from an organization's unique perspective with the target application portfolio in mind, to determine the optimal approach to follow. If an organization determines that there are more pros than cons associated with one option over the other options, they should choose that option.

AppStream 2.0

Challenge 3: OS application delivery approaches and initiators in this document lists the initiators that are available natively on the Windows OS to install applications. As a managed service, not all these options are available with AppStream 2.0 as it is not possible for customers to use the pre-OS and mid-OS deployment initiators to install applications. Therefore, AppStream 2.0 provides the opportunity to either incorporate applications into an image or VHD, or leverage post-OS deployment initiators to deliver applications.

Stream view options

AppStream 2.0 provides two different streaming views that can be used to deliver applications to end users. These are the application stream view and desktop stream view.

Application stream view

The AppStream 2.0 application stream view can be used to present users with access to one or more applications that have been installed into an AppStream 2.0 image. Rather than presenting a full Windows desktop as "desktop stream view" does, application stream view presents users with one or more icons to enable them to launch individual applications from their AppStream 2.0 fleet instances. The presentation of the applications depends on whether applications are being accessed using a web browser or the AppStream 2.0 native client, and whether "Native application mode" is enabled within the native client. Essentially, there is extensive choice for how applications are displayed to end users, and the choice for which option to utilize is dependent on the use case, business requirements, and in some cases, user preference. The process to determine the optimal approach is beyond the scope of this document; however, the ability for AppStream 2.0 to deliver an individual application or a set of applications without a Windows desktop is important to be aware of.

Desktop stream view

Using AppStream 2.0 with desktop stream view enabled means that a full Windows desktop experience is available for end users to interact with. Since a full desktop is available, there is a higher likelihood that the AppStream 2.0 image is going to be used by more than one group of users, each requiring a different portfolio of applications. In this scenario, the optimal application delivery approach would be to establish a shared baseline image that includes the applications in common between the groups of users, and then use an additional toolset or additional AppStream 2.0 fleet to deliver the applications specific to the groups of users independently on demand. This would minimize the amount of maintenance associated with the image since only a single image is required for the main fleet and it would allow future use cases to be satisfied easily through the on-demand delivery of applications.

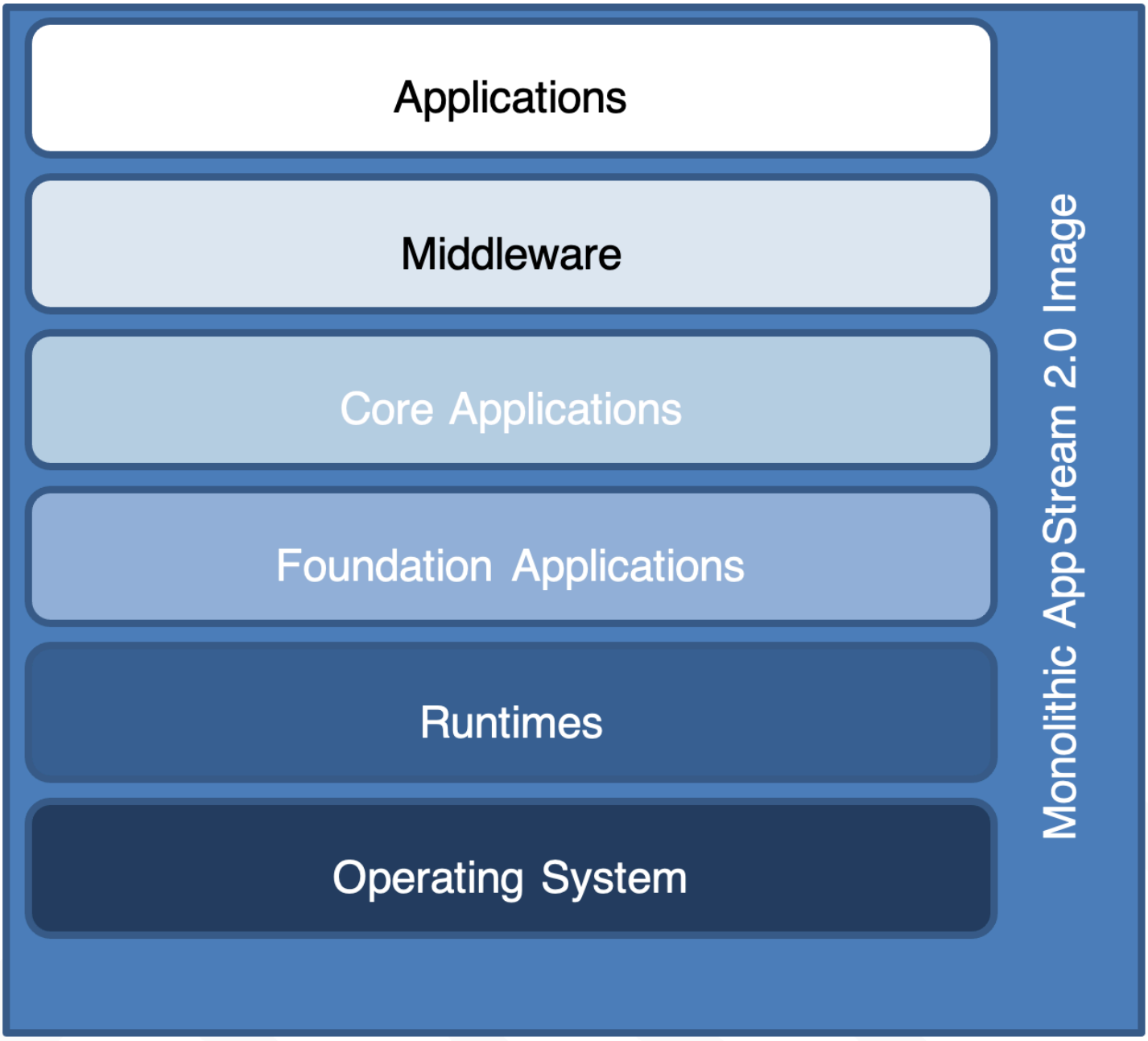
However, there may be challenges that prevent the use of this approach, such as the size of the application binaries, licensing terms for the applications, cost of leveraging an additional toolset, time it takes to deliver the application binaries on demand, and desired end user experience. Therefore, there are trade-offs to be considered in the choice of approach and the associated tools. For completeness, we consider the different approaches that are available for application delivery with AppStream 2.0 in the following sections.

Application delivery options

With the two streaming view modes in mind, there are typically four approaches to consider for delivering applications to end users that are provided from an AppStream 2.0 fleet.

Option 1: Monolithic image

Where single or multiple applications are being used by a large user base and need to be delivered using AppStream 2.0, the best approach is to incorporate the applications, any dependencies, and required runtimes within the AppStream 2.0 image.



Using a monolithic AppStream 2.0 image to deliver a complete set of applications.

Some of the advantages and disadvantages of this approach are outlined in the following table:

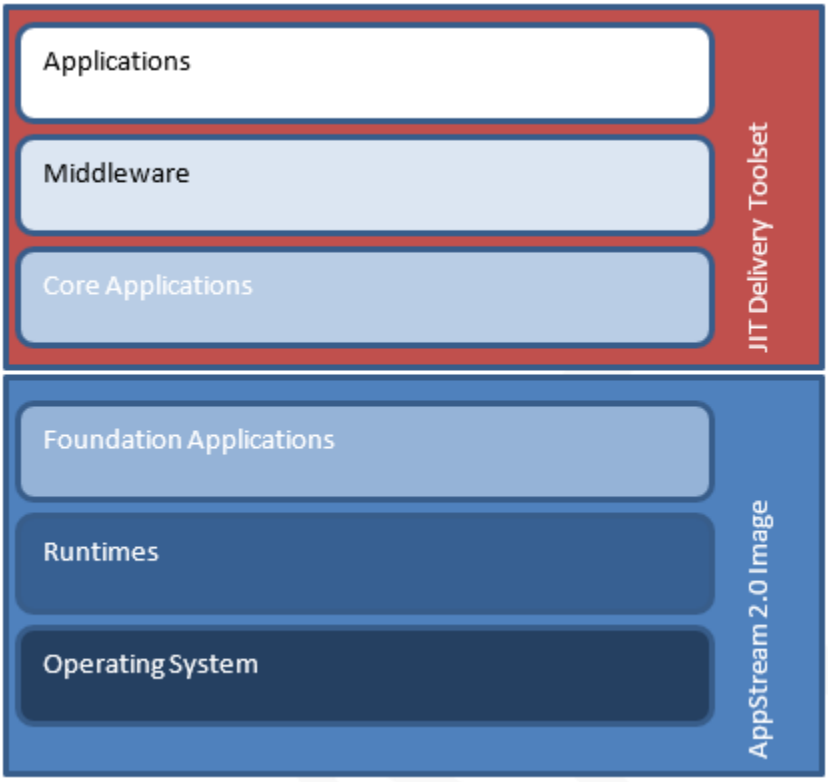
Table 2 — Advantages and disadvantages of using a monolithic AppStream 2.0 image.

Advantages
Simple Approach: All application binaries (specifically runtimes, foundational, core, middleware, and application) are installed in the same image.
Applying the software binary taxonomy is simple since all binaries are installed into the same image. The taxonomy can be utilized to inform the cadence of image updates based on the layer of the taxonomy that the different binaries are in.
Does not need an application entitlement mechanism to authorize users' access to the applicati ons within an AppStream 2.0 instance.
Managed image update mechanisms can be used to keep the AppStream 2.0 image current with Windows OS security patching baseline.
No external application delivery infrastructure dependencies are required to deliver and start the application.
Timely delivery of applications is guaranteed since the image contains all the applications required by the target userbase.

Disadvantages
Image maintenance: Any change required to an application or runtime will require a new image to be created due to its monolithic nature. If the image creation and therefore maintenance process is automated, then the impact of changes can be minimized.
Conflicts can arise between applications, adding complexity. These can be addressed prior to any user encountering them by using native OS capabilities or using a third-party toolset.
Testing of the image can be complex, depending on the number of applications installed in the image, and the interaction between these applications in support of the business processes that use them.
Change management around the image can be complex depending on the number of components included in the image.

Option 2: Baseline Image with "Just in Time" delivery toolset

A baseline AppStream 2.0 image can have runtimes, foundation and possibly core applications installed within it to deliver the baseline set of applications. Where users require additional applications to be delivered for a LoB, then additional third-party toolsets can be utilized to deliver these applications "Just in Time" or JIT after an AppStream 2.0 session has been started and a user is logged in and active on an instance.



Using a baseline AppStream 2.0 image with a JIT toolset to deliver a complete set of applications

Some of the advantages and disadvantages of this approach are outlined in the following table:

Table 3 — Advantages and disadvantages of using a baseline AppStream 2.0 image with JIT toolset

Advantages
The base image is as lightweight as possible while incorporating common runtimes, foundatio n, and possibly core applications. Therefore, the image needs a lower frequency of maintenance compared to an image containing all the applications the users require.
A trusted baseline image that has been thoroughly tested can be utilized to underpin other sets of applications that are delivered using a JIT approach (for example, multiple LoBs sharing

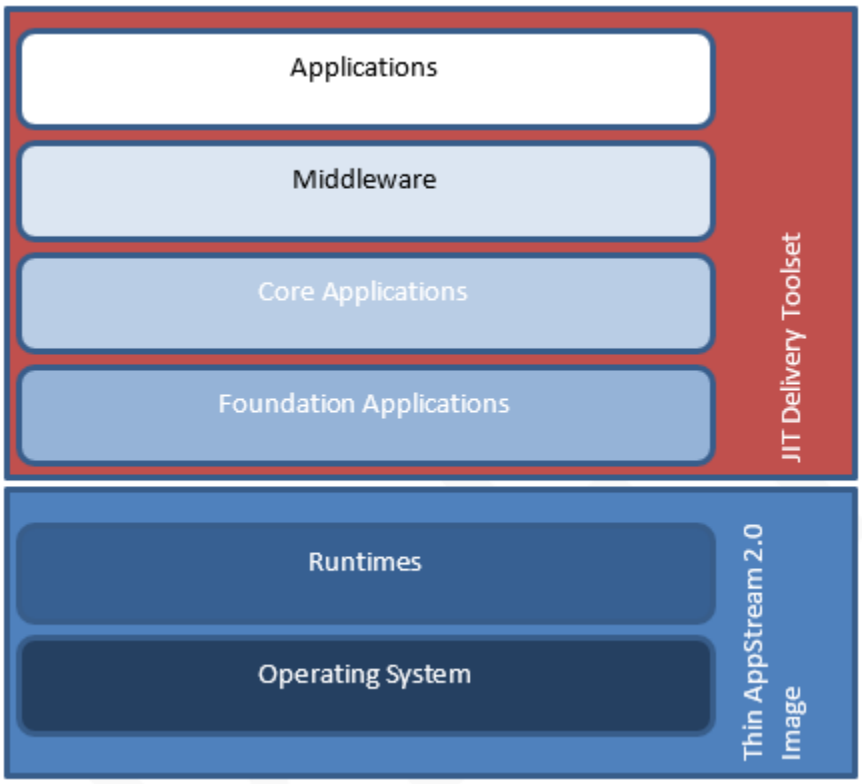
Advantages
a common baseline image but using an application delivery toolset to deliver the remaining applications in their portfolio).
The opportunity for application conflicts to arise can be reduced compared to option 1, due to fewer applications co-existing on the same instance.
A managed image update mechanism can be used to keep the AppStream 2.0 image current with the Windows OS security patching baseline.
Testing of the base image is lighter weight than option 1 since there are likely to be fewer components in the base image to test.

Disadvantages
It is more complex to operate than option 1 due to a dependency on additional infrastructure and the potential to use application entitlements to restrict the additional applications that users need access to.
It is more expensive to implement than option 1 due to a dependency on a third-party solution and additional infrastructure.
The timely delivery of applications may not be possible depending on the size of the application binaries and capabilities of the third-party JIT delivery toolset.
It relies on an application entitlement mechanism to authorize users' access to the applications within an AppStream 2.0 instance.
A managed image update mechanism may break applications incorporated into the image due to security patches being applied, and therefore all image updates need to be tested.

Option 3: Thin Image with "Just in Time" delivery toolset

In contrast to options 1 and 2, if an enterprise is using a broad application portfolio across a large userbase, the third option to consider is to have a thin image that only includes the third-party delivery toolset with all applications delivered on-demand to the base thin image using a third-party application delivery toolset.

The Dynamic Application Framework (see Use the AppStream 2.0 Dynamic Application Framework to Build a Dynamic App Provider) capability within AppStream 2.0 allows a dynamic application catalog to be presented to an end user. The dynamic catalog may be determined based on a user entitlement (for example, Active Directory group membership), and in this scenario presents an opportunity to only entitle users access to the applications that they require and initiate the delivery of these applications to the AppStream 2.0 instance.



Using a thin AppStream 2.0 image and JIT delivery toolset to deliver a complete set of applications

Some of the advantages and disadvantages of this approach are outlined in Table 4.

Table 4 — Advantages and disadvantages of using a thin AppStream 2.0 image with JIT toolset

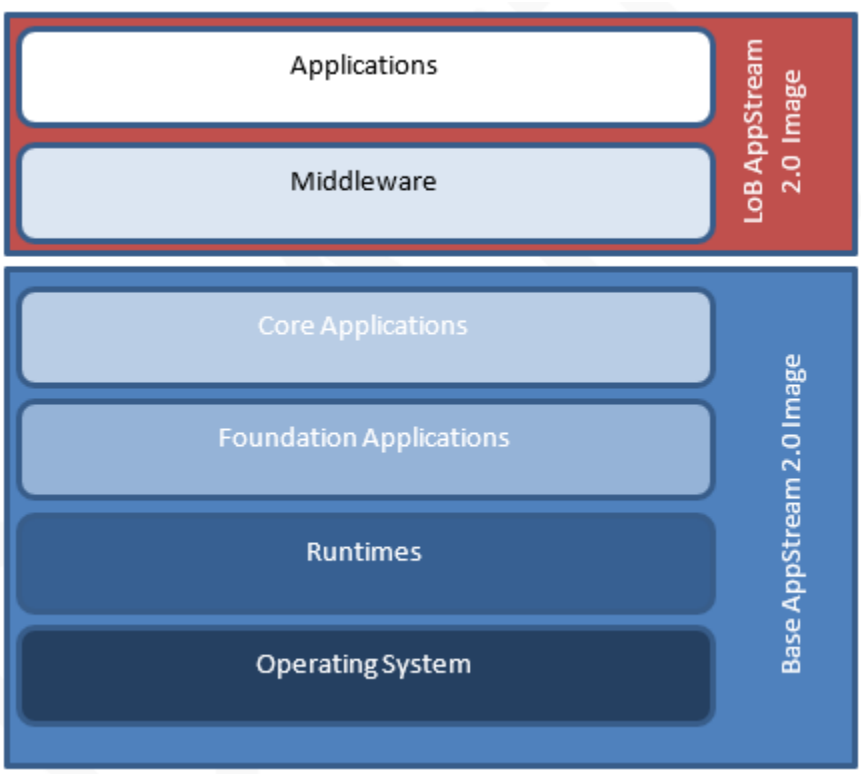
Advantages
Thin image can be maintained easily using the Managed Image Update mechanism. Only the JIT delivery toolset needs to be maintained over above the base image.
Flexible approach that is agile enough to deliver any applications that the 3rd party JIT delivery toolset can deliver and that are compatible with the AppStream 2.0 service.

Advantages
Consistent approach used to entitle users to all applications, simplifying software license management and user operations.
Testing the thin base image is quick and simple since only the OS and the third-party delivery toolset need to be tested.

Disadvantages
More complex to operate than option 1 due to dependency on additional infrastructure and the potential to use application entitlements to restrict the additional applications that users need access to.
Timely delivery of applications may not be possible depending on the size of the application binaries and capabilities of the 3rd party JIT delivery toolset.
More expensive to implement than option 1 due to dependency on third party solution and additional infrastructure.
3rd party delivery toolsets typically require applications to be re-packaged in a proprietary format adding complexity to the delivery of applications and cost to the overall solution.
Opportunity for application conflicts to arise is increased compared to option 1 and 2 unless consideration is paid toward careful management of the application portfolio to avoid the issue. This is because blind entitlement of users to applications may introduce an application conflict (e.g. different incompatible runtime versions required between applications, single file extension used between applications). Understanding the software binary taxonomy, applying it to the application portfolio and understanding the operation of individual applications can mitigate this risk.
Relies on application entitlement mechanism to authorise users' access to the applications within an AppStream 2.0 instance.
Testing the delivery of individual applications is simple, however, the interaction between them can give rise to a complicated and in-depth testing requirement.

Option 4: Nested AppStream 2.0 sessions

In a similar way to the third option outlined above for WorkSpaces, it is possible to use nested AppStream 2.0 sessions to deliver a user's complete application portfolio. With this option, the user's main desktop would be delivered using Desktop Stream View with the organization's foundation and core applications included in the image. A user's or LoB's specific application portfolio would then be delivered using a separate AppStream 2.0 fleet that provides only these specific applications and uses Application Stream View to deliver these into the user's AppStream 2.0 Desktop View delivered Windows desktop.



Using a base AppStream 2.0 image in Desktop View (Foundation and Core applications) with an additional AppStream 2.0 image (LoB applications) to deliver a complete set of applications

Some of the advantages and disadvantages of this approach are outlined in the following table.

Table 5 —Advantages and disadvantages of using nested AppStream 2.0 images.

Advantages
The base image is as lightweight as possible while incorporating common runtimes, foundatio n, and possibly core applications. Therefore, the image needs a lower frequency of maintenance compared to an image containing all the applications the users require.
A trusted baseline image that has been thoroughly tested can be utilized to underpin other sets of applications that are delivered using a JIT approach (for example, multiple LoBs sharing a common baseline image but using different fleets to deliver their entire application portfolio).
The opportunity for application conflicts to arise can be reduced compared to option 1, due to fewer applications co-existing on the same instance.
A managed image update mechanism can be used to keep the AppStream 2.0 image current with the Windows OS security patching baseline.
Testing of the base image is lighter weight than option 1 since there are likely to be fewer components in the base image to test.
A LoB AppStream 2.0 image can be tested and released in isolation from the foundation and core applications, and on a different cadence.
The timely delivery of applications is always possible since they are only delivered using a fleet and image.

Disadvantages
It is more complex to operate than option 1 due to a dependency on an additional AppStream 2.0 image and the potential to use application entitlements to restrict the additional applicati ons that users need access to.
It is more expensive to implement than option 1 due to use of at least one additional AppStream 2.0 fleet (but possibly multiple fleets).
A managed image update mechanism may break applications incorporated into the image due to security patches being applied, and therefore all image updates need to be tested.

Recommendation

The options outlined above cater to differing scenarios frequently seen within enterprise environments. There is not a single recommendation that can be made to determine the optimal option to use, but rather the options should be considered alongside the software binary taxonomy to determine the best approach to satisfy the application delivery requirements of a given scenario when using the AppStream 2.0 service.

Given the options outlined above, where the advantages for one option outweigh the disadvantages for that option as well as the advantages for the other options being considered, it is recommended that this option is chosen in preference to the others. Ultimately, a compromise between choosing one option over another will need to be made as each option has its own merits and limitations.

Conclusion

Challenges associated with end user Windows application delivery are technical challenges such as a legacy environment and a broad installation base, and organizational challenges such as user autonomy and rate of change. These were explored at the start of the document and then supplemented by three challenges that are inherent and unavoidable in Windows application delivery. These challenges were the Anatomy of a Windows application, Packaging formats for binaries, and OS application delivery approaches and initiators. The impacts of these three challenges can be mitigated through the resolutions considered alongside each challenge, and also through the application of the software binary taxonomy included in this whitepaper.

The taxonomy included in this document can be applied to Windows end user application delivery in any context. It can be used outside of the AWS end user computing services such as AppStream 2.0 and WorkSpaces. It can be valuable to organizations of all sizes to help guide their decision-making around how and when to combine or separate application binaries to simplify and maintain a consistent approach to application binary delivery.

The factors described within each Well-Architected Framework pillar are important considerations for any organization's end user application delivery strategy, and are expected to provide additional perspectives to ensure a well-rounded and considered strategy can be formulated.

End user Windows application delivery is a complex subject due to the heritage of the Windows OS. This whitepaper provides you with a taxonomy for classifying Windows software binaries. Classifying software according to the taxonomy and understanding the impact of the dimensions on the specific software being assessed helps to decide the approach for distributing software to the target service. Understanding the positives and negatives of the different approaches, along with the potential impact to either the integrity of the platform or the software being delivered, will lead to an optimized decision-making process. Having a traceable decision-making process that can record the decisions made and the justification behind the decisions provides significant benefits to any organization from an audit and end user productivity perspective.

Contributors

Contributors to this document include:

- Author: Mark Homer, Senior Architect, Amazon Web Services
- Andrew Wood, Principal Architect, Amazon Web Services
- James Scanlon, Senior Architect, Amazon Web Services
- Matt Taylor, Senior Manager, Amazon Web Services

Document revisions

Date	Description
August 3, 2022	First publication

AWS Glossary

For the latest AWS terminology, see the AWS glossary in the *AWS Glossary Reference*.

Notices

Customers are responsible for making their own independent assessment of the information in this document. This document: (a) is for informational purposes only, (b) represents current AWS product offerings and practices, which are subject to change without notice, and (c) does not create any commitments or assurances from AWS and its affiliates, suppliers or licensors. AWS products or services are provided "as is" without warranties, representations, or conditions of any kind, whether express or implied. The responsibilities and liabilities of AWS to its customers are controlled by AWS agreements, and this document is not part of, nor does it modify, any agreement between AWS and its customers.