WHITE PAPER

MODERNIZING LEGACY APPLICATIONS FOR THE DIGITAL AGE

April 2019

EXECUTIVE SUMMARY

Technology today moves fast. Customers demand unique, high-quality digital experiences for interacting with your company, and they'll move on quickly to a competitor if they don't get what they want. Yet enterprises today struggle not only with new product development, but also with the ongoing maintenance and upkeep of a significant legacy application portfolio that has been a bedrock of their company's longevity. IT executives are quickly realizing that a bi-modal approach to their application portfolio, where some components can evolve quickly while others are left to stagnate, is now inhibiting growth, because innovative customer experiences are forcing fundamental changes in business processes. These modifications, which touch all applications, mean that legacy systems are now becoming a drag on innovation. A new approach is needed.

The two existing approaches to dealing with legacy applications — rewrite/replace and lift-and-shift — have been of limited benefit to enterprises seeking more agility. Instead, we have a third way, which we call ***application modernization***. Application modernization is a way to repackage legacy applications to imbue them with more agility, including some cloud-native features, and make them portable to a variety of environments such as the cloud or containers. We have developed a proven process for applying the Habitat technology to legacy applications that shows significant ROI and cost savings. For example, a Fortune 500 manufacturing company projects an estimated savings of at least $1.6M across a legacy application portfolio of two hundred applications using this approach, in contrast to their previous plan of lifting-and-shifting existing deployments and licenses to the new environment. If you are a CIO, IT director or manager, application portfolio owner, or anyone else struggling with how to balance the business's demands for increased product innovation with maintenance of an aging application portfolio and its underlying infrastructure, read on.

WHY MODERNIZE LEGACY APPLICATIONS?

Today's pace of technology innovation is staggering. It's hard to believe, but only ten years ago, in October 2008, did Amazon Web Services (AWS) make their Elastic Compute Cloud (EC2) generally available. AWS EC2 promised near-instant fulfillment & hourly pricing for server infrastructure that previously was only consumable via high-cost capital expenditures with three-to-five-year depreciation cycles. Other cloud providers, like Microsoft Azure, Google Cloud Platform, and DigitalOcean, soon followed. The dramatic reduction in time & effort needed to provision infrastructure gave rise not only to Chef, our original infrastructure automation technology, but came to define an level of customer expectation for provisioning speed that we now call *effortless infrastructure*.

Forward-thinking CIOs quickly saw the benefits of effortless infrastructure, opening up the ability to capture new markets with innovative product development and digital-first customer experiences for their companies. At the same time, however, IT departments were and continue to be hamstrung with a portfolio of legacy applications developed in a pre-cloud era. In the decade or so since the cloud first came of age, CIOs have largely tried to address this estate with two main approaches: rewrite/replace, or lift-and-shift to the cloud.

WHY NOT SIMPLY REWRITE OR REPLACE APPLICATIONS?

Rewriting or replacing applications is, of course, every CIO's dream. Gone will be applications written in old languages like Fortran, COBOL, Visual Basic, Microsoft .NET Framework 2.0, etc. as well as all the underlying infrastructure like servers running Windows 2003, middleware, monitoring and deployment tools, and more. In their place will be the most modern microservice architectures running in a Platform-as-a-Service (PaaS) like Pivotal CloudFoundry, or even on containers running under an orchestration platform like Kubernetes. Companies that set down this road quickly realized the daunting nature of the task before them. Any large enterprise has hundreds or thousands of legacy applications, and the speed at which application teams are able to rewrite applications is low. Worse still is the risk to the business of breaking existing workflows and business processes, particularly when those applications are regarded by the business as "working just fine." It's difficult to get executive buy-in to spend a huge amount of money for wholesale application replacement when, at the end of that painful process, they perceive the functionality to be the same. The CEO rightly demands to know why all that money was spent solely on a like-for-like substitution without improving the end customer experience.

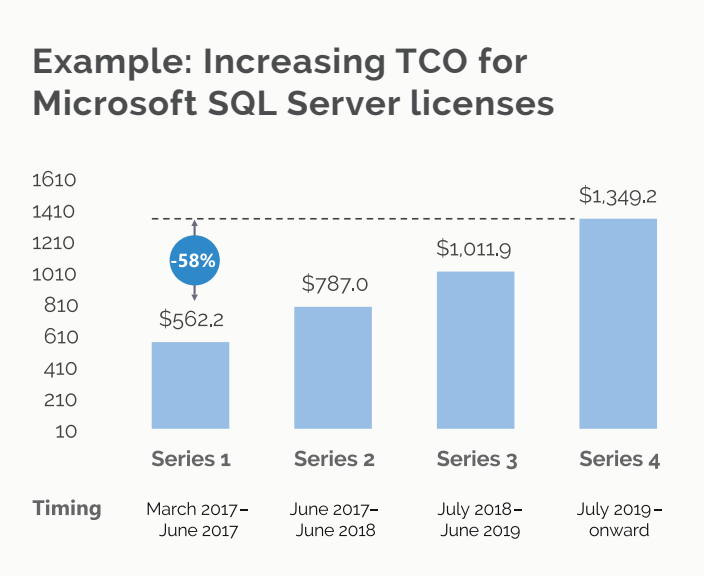
WHY NOT LIFT-AND-SHIFT LEGACY APPLICATIONS DIRECTLY TO THE CLOUD?

The second approach that IT departments take is to lift-and-shift legacy applications to the cloud or containers. This is similar to the Physical-to-Virtual (P2V) strategy organizations employed in the early 2000s to migrate from physical servers to virtual servers. But while this tactic does, indeed, realize the expected efficiencies of shifting capital expenditures to operational expenditures, it does not improve agility nor significantly reduce that operational expenditure. Some downsides:

- **Lack of cost reduction:** Operating system and middleware licenses don't go away, and in fact, both costs and risk continue to escalate the longer enterprises run end-of-life or near-end-of-life infrastructure.

- **Hiring and staff retention challenges:** Ongoing management of these applications, including having to hire and retain personnel who can deal with older technologies, is an enormous expenditure for companies — if those people can be found at all.

- **Poor performance of pre-cloud applications in cloud:** Applications designed for a pre-cloud data center environment – where the network has no packet loss, latency is predictable, there are no unpredictable "noisy neighbors" stealing CPU cycles from your workloads, and IP addresses are statically assigned – often function poorly when lifted-and-shifted to the cloud. Any consultant or vendor who waves a magic wand advocating lift-and-shift as a panacea hasn't been in the trenches actually dealing with such issues.



Windows Server Premium Assurance SQL Server Premium Assurance data sheet, March 2017

Where does this leave today's CIO, trapped between business demands for growth via new product development yet owning a large portfolio of legacy applications that they often weren't responsible for developing — and for which they often don't even own source code?

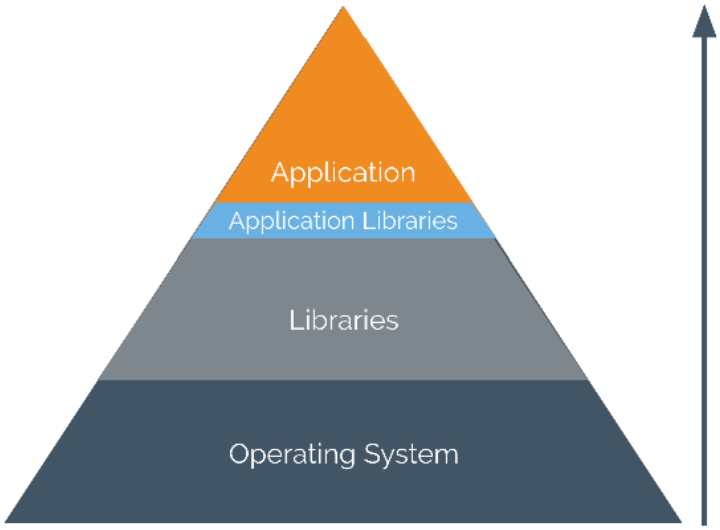
We believe there is a third way: *application modernization*.

APPLICATION MODERNIZATION WITH HABITAT: WHAT IS IT?

Application modernization refers to the process of intelligently repackaging legacy applications to increase their portability and manageability, including giving them cloud-native features like service discovery and dynamic configuration management. The term ***legacy application***, however, is often used negatively, because of the aforementioned management nightmare of dealing with software that was written years or even decades ago. But let's look at what legacy applications still do provide: business value. Clearly, your legacy applications are still critical to the business. Otherwise, IT operations would have decommissioned them long ago, and nobody would have noticed. The pain – and thus derision – arises from all that is needed to support this valuable business logic: ancient middleware frameworks and old operating systems, not to mention all of the management tooling, release pipelines, runbooks, manual processes, and everything else that goes along with it.

Business App 1	Business App 2	Business App 3	Business App 4
MSVC, COM+, etc.	MS .NET 2.0	IBM WebSphere	Tomcat 6 / Java 7
Windows 2003	Windows 2008 R2	Red Hat Linux 5	Red Hat Linux 6

One of the main sources of this pain is due to the way in which we have been building and packaging applications up until this point. Traditionally, we build applications starting from the infrastructure (servers, operating systems) up towards the application, as shown in this diagram:



This leads to a high degree of coupling between all of these elements. In fact, this entire stack is what *becomes* the application, which IT departments carry around until that application is replaced. This is extremely frustrating because everything under the application is just a supporting element. The business value sits at the top, but we must drag along everything else needed to support it.

What if there was a technology we could apply to applications to extract them from the underlying support structure, make them portable, and thus allow them to be redeployed on anything from bare metal through virtual machines to containers and beyond? That is the approach we have taken with Habitat. Visually, that looks like this:

Keep this:

Business App 1	Business App 2	Business App 3	Business App 4

Bundle as much of this as needed with the app:

MSVC, COM+, etc.	MS .NET 2.0	IBM WebSphere	Tomcat 6 / Java 7

Eliminate or reduce dependency on this:

Windows 2003	Windows 2008 R2	Red Hat Linux 5	Red Hat Linux 6

The result is a platform-independent artifact that can then be deployed onto whatever target platform you select. This strategy can not only help you migrate applications that are, for example, deployed on Windows Server 2008 onto Windows Server 2012 or 2016, but also migrate these applications into containers, if they are the right profile for your workload.

BENEFITS OF APPLICATION PORTABILITY

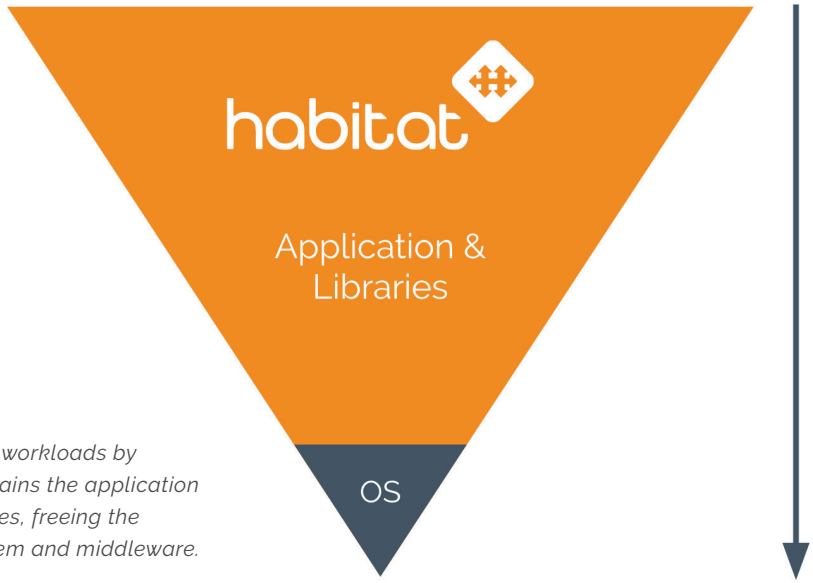
Making legacy applications portable in this way not only helps you reduce costs – allowing you to reduce or eliminate your dependency on older operating system licenses, for example – but has many additional benefits as well, namely:

- **The ability to easily right-size workloads to infrastructure**: With a portable application package, you can now choose your application's deployment target independent of how it was previously deployed. Bare metal, VM, containers (with or without Kubernetes) are all options now.

- **One way to production & management for all applications**: Packaging all applications in a portable format allows for a single continuous delivery workflow to align with all other Habitat-built applications, either legacy applications or modern microservice applications. Habitat creates a single view across your estate, with one way to manage everything.

- **Future-proofing your application portfolio**: VMs were state of the art in early 2000s. Containers are state of the art today. Migrating between states of the art has been painful due to lack of portability. Packaging applications in an infrastructure-independent way allows you to immediately take advantage of future innovations in infrastructure, be that serverless, unikernels, or whatever becomes the "Kubernetes of the future" that hasn't yet been invented.

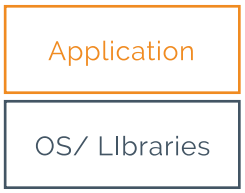
These benefits are available to you for all applications packaged with Habitat, not just legacy applications.



Habitat creates portability for all workloads by making an artifact that only contains the application and a manifest of its dependencies, freeing the software from the operating system and middleware.

HOW DO I GET STARTED WITH APPLICATION MODERNIZATION?

In order to modernize applications by packaging them with Habitat, extracting them from their underlying middleware and operating system, it's important to identify what, exactly, those applications depend upon in those layers. This can often be a non-trivial exercise particularly for older applications that are tightly coupled to those layers. We recommend classifying legacy applications into two types: those that are tightly coupled to the operating system, and those that are loosely coupled to the operating system.



- **Loosely-coupled applications** require fewer operating system provided libraries and services. Those libraries they do depend on are easily relocatable (can be moved around the filesystem) with no impact. Examples of applications that are loosely coupled are ones written in Java (J2SE/J2EE), C/C++ without heavy use of operating system version-specific APIs, .NET Core (not to be confused with .NET Classic), or interpreted languages like Python, Ruby or Perl.

- **Tightly-coupled applications** often depend on operating system-provided features like Internet Information Services (IIS). Examples of such applications are ones written in .NET Classic or Microsoft Visual C/C++, or ones that load custom drivers into the operating system kernel.

Habitat can package both types of applications and deliver portability benefits regardless of the level of coupling. We recommend you prioritize your focus based on return versus level of effort. As such, start with loosely-coupled applications so as to familiarize yourself with Habitat's workflow and approach and be able to demonstrate tangible business benefits before tackling more complex applications.

One leading cause of past migration failure is often due to the lack of understanding of underlying dependencies, something that has plagued previous industry attempts to get off older operating systems. This is particularly true when trying to migrate COTS (commercial off-the-shelf) applications or even in-house software where the source code is no longer available. Fortunately, the Habitat Studio development environment provides a fast, isolated *clean room* to allow engineers to iterate and test the packaging process (described in the *Habitat plan*) without cumbersome virtual machine workflows of the past.

STEPS TO APPLICATION MODERNIZATION AND PLATFORM-INDEPENDENT PACKAGING

We have found that following this checklist for analyzing legacy applications – how they work and what they depend on – invaluable for our customers to begin planning their legacy application migrations. Gathering this information can be done in parallel with learning Habitat. This way, engineering both understands the Habitat workflow and enough of the target application & environment to be productive right away.

- **Understand system architecture.** What application components depend on what, both at an operating system level, but also within that application? For example, a content management system might have a front-end component for displaying published content that also depends on a back-end database/content server to be available. What is the communication protocol between these components? How does access control work? Draw a block diagram that maps all network-connected components, their communication protocols, any orchestration flows currently needed to make dependent components work, and so on.

- **Identify any existing packaging format.** How is the application currently packaged? Is it an RPM, .deb, zip file, MSI, .exe, other? Determine what tools are needed to extract the files from the existing packaging format. Many extraction tools for formats like Inno or InstallShield packages are available in the Habitat core plans to help you get started faster.

- **Is the source code available?** Building applications from source is easier and often preferable, because you can then specify where application objects live after compilation. However, with a bit more effort, Habitat can also package binary-only applications such as COTS applications or ones for which the original source code has been lost.

- **If compiling from source, what build tools are required?** Determine what compilers, pre-processors, or build systems (e.g. MSBuild, make, Maven, Apache Ant, etc.) are needed to build from source. These dependencies will not be packaged with the application but are needed by the clean room environment (Habitat Studio) to successfully compile the application.

- **What does the application depend on at runtime?** Having a good hypothesis about what libraries or other dependencies are needed by the application prior to starting the packaging process will save time and minimize the need to use low-level tools within the Habitat Studio to examine the application.

- **What lifecycle hooks are needed by the application?** At minimum, determine how the application should be properly started and stopped. It's also helpful to decide how the application should respond to other lifecycle events when run under the Habitat Supervisor. How should it react to configuration state changes across the supervisor network, for example? What command can be run to determine if the application is healthy? Is there anything that needs to be done when gracefully shutting down the application?

- **Identify service discovery requirements.** One of Habitat's benefits when applied to legacy applications is the ability to provide dynamic service discovery to them, even though they were architected for a pre-cloud era where IP addresses and hostnames were hardcoded. It's important to understand where such static information lives and how the Habitat plan should be engineered to provide this level of dynamism to applications that did not have it before.

- **Determine the role of infrastructure or compliance automation.** When dealing with legacy applications, it's often necessary to rely on infrastructure automation technology like Chef to configure the underlying operating system to support the application. Examples of this are: allocating enough disk space and creating logical volumes for the application, tuning the system kernel, configuring and assigning network interface aliases, and other low-level tasks. Finally, determine how you plan to make sure the deployment environment is hardened and correct. You can use InSpec to not only validate the environment for security before starting the application, but it can also be used for simple correctness checks for application pre-requisites, e.g. is enough disk space available? are the ports needed by the application available or is something else holding onto them? and so on.

- **Train staff on scripting basics.** Habitat is designed to be easy to use, and the primary packaging language is via Habitat-provided Bash or PowerShell functions (for Linux and Windows, respectively). Ensure that your engineering team has basic training in these common scripting languages before learning to use the specific Habitat packaging functions.

CUSTOMER SUCCESS STORY: MOVING A LEGACY APPLICATION FROM WINDOWS SERVER 2008 TO WINDOWS SERVER 2016

A large manufacturing company was in the middle of an initiative to migrate all legacy applications into a new private cloud built on top of next-generation data center technology. But they were struggling to determine how to move hundreds of applications into this new environment without incurring a significant one-time expense just to lift-and-shift without any realized ongoing savings. Many of their existing deployment processes were manual, and they wanted a way to use automation to dramatically reduce operational expenditures for their new data center environment.

As a proof-of-concept, the customer decided to use Habitat to modernize a legacy three-tier Embarcadero (formerly Borland) Delphi application from 2002. This application has several components, including a Windows GUI, a Windows service, some COM+ deployable components, all deployed to a Windows 2008 platform and connecting to an Oracle database.

Prior to automation with Habitat, the customer was performing manual runbook-based installation procedures for every change. Each deployment took two engineers at least two days, with an additional five days to test and troubleshoot issues. By packaging all components with Habitat, the customer realized the following benefits:

- All of the Habitat packages can be deployed to any server with a 100% automated process and a 100% success rate.
- Several of the components (besides the desktop GUI) can now run in containers.
- Deployment time has been reduced to under ten seconds in comparison to the seven days as mentioned previously.

The customer estimates they are saving approximately $19,200 annually on manual release management costs for a single application. Based on the cost savings realized here, they plan to target over two hundred additional legacy applications for packaging in Habitat. This will not only eliminate the costs for release management (which conservatively will save more than $1M/year), but also the cost for IT operations to "shadow" the installation of the most complex applications, which is projected to save an additional $600,000/year.

Beyond cost reduction, the customer also described the following additional benefits:

- Risk reduction, as they were able to integrate InSpec and Chef for automated drift detection and remediation
- Improved compliance management, as audit data is automatically collected on every deployment, gathered regularly in production, with a historical record that can be presented to auditors on-demand, rather than scrambling at audit time
- Ability to respond extremely quickly to zero-day vulnerabilities
- All deployments are run through a pipeline so that errors are found early in the process rather than in a production environment
- Increased employee job satisfaction by eliminating boring, manual tasks and moving them to more interesting engineering work.

SUMMARY

The two main strategies for dealing with legacy applications today: rewrite and replace, and lift-and-shift, provide only incremental increases in agility. They are also either of limited practical scope (as is the case with rewrite/replace) or don't realize the necessary operational benefits at scale for all of IT to become fast IT (lift-and-shift). Instead, a third approach, which we call application modernization, repackages and modernizes legacy workloads in-place to increase their manageability, makes them portable, and allows for them to be easily migrated to modern operating systems or even cloud-native infrastructure like containers. We've taken a deep dive into the different types of legacy applications and the general approach that Habitat takes to make this a successful process. Finally, we demonstrate the tangible benefits of a Habitat-led approach to reducing costs and accelerating cloud migration for a Fortune 500 manufacturing company's portfolio of legacy applications.

To learn more about how Habitat can help you modernize your legacy application portfolio and make it just as fast as your most modern applications, please visit www.chef.io/solutions/modernizing-apps or contact us today at awesome@chef.io.