No App Left Behind:
Tips & Considerations for Including Containers in Your Traditional Application Modernization Strategy
Table of Contents

Executive Summary .................................................................................................................. 3

Defining Containers and Understanding Their Benefits ....................................................... 4

Tips for Moving Your Organization Forward, The People Element

Tips for Analyzing Your Applications, The Strategy Element ............................................... 6

Tips for Embarking on a Containerization Journey, The Technology Element ....................... 7

Summary: Multiple Technologies Will Be Used in All Stages of an Application Modernization Journey .................................................................................................................. 12

Additional Resources and Citations ......................................................................................... 13
Executive Summary

There is no dispute that containers are a rising technology that is seeing wider adoption in the enterprise. At Skytap we recognize both the value and the complexity associated with this technology and have built capabilities into our cloud that accelerate the move to containers as part of the modernization journey. For traditional applications, that journey varies across enterprises based on their knowledge of cloud capabilities, their adaptability to change, the adoption of automation, and the requirements of the application.

This document will provide an overview of containers, how they can be applied when modernizing traditional applications and the considerations and benefits of using containers as part of your modernization strategy.

Skytap participates in each phase of the application’s modernization journey through our Infrastructure, Process, and Architecture Modernization (IPA) approach to successfully streamline and accelerate modernization alongside containers and other technologies as noted throughout this document. Our goal is to eliminate complexity where possible to enable teams to be more effective in developing and managing their applications with as much automation as possible—natively or by supporting integration with other tooling.

As in choosing a cloud provider, there was once a belief that a single provider could deliver on all of a business’s needs, but that has proven to be invalid with the rise of multi-cloud adoption. A similar pattern is proving true across the underlying application technologies as well. Virtual machines, various flavors of containers and their orchestration tooling, DevOps automation tools, native cloud services, and traditional applications will all need to work together, as no singular solution on its own can address the complete application lifecycle.

Skytap accelerates traditional application modernization by working in concert with these technologies and brings unique capabilities to help enterprises accelerate time to value, reduce errors in the SDLC, and deliver true-self service that provides teams with the access they need to on-demand resources so they can innovate faster.
Defining Containers and Understanding Their Benefits

Containers are the next wave of digital transformation for application development. With enterprise adoption of containers growing rapidly, Gartner predicts by 2020, 50% of global organizations will be running containerized applications in production.

The diagram demonstrates a visual difference between containers and VMs. Where VMs virtualize hardware to run multiple OS instances on a single physical server, containers provide a way to virtualize a single OS so that the multiple workloads stored in individual containers can share a single OS instance. Containerized applications are built on a microservices-based architecture.

An application using containers is composed of a suite of small services that run in individual containers. When combined with DevOps tooling, containers enable faster development, deployment, and the portability to run anywhere, on-premises or in the cloud. As enterprises look to bring containers into their application strategy, there are benefits that can be realized and considerations that must be must address.

Enterprises are moving to containers for following benefits:

1. **Developer agility:** Build with the language and tools which best align for the feature. Build individual components that are easier to, build, test, and operate.
2. **Fast provisioning:** Containers are provisioned via software into already-operating infrastructure. Start, stop, copy, and move in seconds.
3. **Cloud scale:** Containers allow a true scale-out, auto-scale up and down, fault-tolerant application. Unhealthy microservices can be automatically detected, terminated, and restarted.
4. **Consistency:** Every app container running is exactly the same; there are no gotchas where servers get out of sync.
5. **Isolation:** Containers virtualize CPU, memory, storage, and network resources at the OS-level, providing developers a sandboxed view of the OS logically isolated from other applications.
6. **Portability:** Containers are able to run on virtual machines, bare metal, a developer’s machine, on-premises, or in the cloud.
We'll discuss the benefits and considerations throughout this paper. We will also address how Skytap and containers when used together, can help enterprises achieve application modernization and realize faster time to value.

Tips for Moving Your Organization Forward, The People Element

Hyperscale cloud providers have made it easy for enterprises who are building greenfield, cloud-native applications to lead with a microservices and container strategy. Cloud-native developers leading these efforts are commonly proficient in agile processes, develop in modern languages, and bring their favorite flavor of CI/CD automation tooling.

However, to function at scale, a successful container implementation also requires a clear strategy, a container platform, orchestration tooling, and cross functional buy-in from application development, IT, networking, and security teams. Whether containers or fully cloud-native are the desired end state, there is a modernization journey that an enterprise must go through -- the business needs to buy in, existing skill sets need to be evaluated, and changes implemented.

Enterprises will need to consider the following items related to containers:

1. A container must have a config file, an image file, a registry, and a host to run at scale.
2. Security principles change, and there are more parts to secure.
3. SDLC, deployment tools, and approaches are different than non-containerized applications.
4. Networking is handled in conjunction with a container networking layer rather than using durable or hard-coded addresses.
5. Management of the application and OS are different due to orchestration, and tools are new.
6. Containers may not be supported on older OSes used by traditional applications.

Before an enterprise adopts a container strategy, Gartner provides the following guidance:

1. Allocate appropriate resources to teams spearheading the effort. Containerization is time-consuming and can be expensive.
2. Obtain training for your personnel to ensure a quick onramp and long-term consistency in managing these new and complex environments.
3. Adopt infrastructure automation best practices before attempting to deploy containers. This can include scripting, server automation, and continuous configuration automation tools.
**Tips for Analyzing Your Applications, The Strategy Element**

Before companies start to decompose their application into containerized microservices, it's important to develop an application modernization strategy to identify which components of the application will move and when. As part of this strategy, it's imperative that enterprises have a plan and a safety net for components that cannot be containerized; you don't want your whole containerization effort to fail after you encounter one component that can't or shouldn't be forced into a container.

A general principle for containerizing an application is to start from the outside and over time, move inward. The best candidates for early containerization have one or more of these characteristics:

- Statelessness: not storing data persistently
- Few dependent services: components that have few or no services that depend on them, “leaf nodes” in the application
- Simplicity of purpose: doing one thing well rather than many things
- Simple deployment requirements: programmatic startup with no manual steps

Some examples include web servers that host static content, web-tier application components, and components that support and surround the core application such as test rigs, hosts that run scripts and diagnostic tools. More specific guidance on application candidates can be found here from Docker.

As noted above, enterprises should plan for the possibility that the entire application may not be containerized, resulting in a hybrid app. Hybrid apps aren't failures or even somehow incomplete. Often, technical constraints and cost-benefit analysis naturally cause companies to conclude that some applications are best modernized into a combination of containerized microservices that communicate with some core non-container legacy components. With a hybrid app, communication between the legacy, VM-based components and containerized microservices can be achieved with the introduction of an intermediate message broker such as RabbitMQ⁴.

When containerizing a traditional application, enterprises will find that the operational monitoring and management tools are very different than current tooling. The introduction of containerized microservices requires tooling changes for how the health, performance, and status of an application are determined. Not only is the tooling new, but education is required for developers and operations engineers so they can...
continue to be effective managing the application once deployed. For instance, containerized microservices are deployed as fleets of anonymous instances; engineers will no longer easily be able to remotely log in (SSH) to named servers to troubleshoot the application. Troubleshooting will still be required, but the mechanics are different in the containerized world. If engineers aren’t actively included and trained, they will be less effective at diagnosing critical issues, resulting in greater mean-time-to-repair (MTTR) and ultimately a worse end-user experience.

Similarly, it’s important to evangelize the benefits of containerized microservices to engineers from the beginning—engineers should want to use them. It’s important to focus on good developer tooling and the user experience of engineers themselves. Otherwise, they will view the whole effort as cumbersome and disruptive, resisting rather than embracing change.

We’ll discuss many of these items throughout the next sections.

Tips for Embarking on a Containerization Journey, The Technology Element

Modernizing a traditional application occurs in phases. It’s worth noting that there is a difference between “move an application” and “modernize an application.” In the former, you may be able to solely lift and shift an application’s VMs or containers to the cloud, leaving them as-is. In the latter, you first lift and shift, but your intention is to change and improve the application over time by leveraging new cloud paradigms with the goal of delivering faster innovation for a more competitive edge.

When choosing to modernize the application, it’s important to look at its structure and lifecycle holistically. As most enterprises are starting their modernization from a virtualized application, they typically hold off on moving to containers until they are ready to rearchitect the application.

Several of Skytap’s large enterprise customers have used Skytap Cloud to successfully containerize as part of the journey to modernize their traditional applications, including a large retail customer who used Skytap to accelerate this journey. Further, our own engineering team is currently using Skytap Cloud to incrementally containerize our technical underpinnings.

From our customers’ and our own success in modernizing traditional applications, a repeatable process has emerged over time:

1. Start with your traditional, monolithic application
2. Move application components to heavyweight, monolithic containers (optional)
3. Incrementally decompose heavyweight containers/VMs into microservices

For each step of the process, you’ll learn which benefits you can realistically expect and what challenges you’ll likely face.
Step 1 (Optional): Move Traditional Applications to Heavyweight, Monolithic Containers

As a first step, some companies start by moving their monolithic, VM-based application to heavyweight containers with the same exact application architecture that is modernized or changed. This is done using Docker Enterprise Edition or an open-source tool called Image2Docker.

**Benefits**

There are a handful of benefits to this approach:

- Portability between environments is increased as the underlying container image binary remains the same.
- The SDLC receives some gains due to consistency across different stages since the original container image is preserved across multiple internal staging and pre-production environments.
- Reduction in Storage - As the underlying operating system of a container image can be shared, some enterprise may see a reduction in storage utilization. Note: that storage savings are significant only if the the bulk of the application’s storage is the operating system. Many applications’ storage utilization is instead dominated by application data; in this case, the storage savings will likely be marginal.

**Considerations**

To use the well-known analogy, in this step heavyweight containers are still “pets” not “cattle.” That is, they are singleton instances needing to be managed manually and idiosyncratically—“hand-fed.” This is no different from the management of the preceding VMs, where there are distinct and durable “hostnames” (really, the container analogue) and where the failure of a container is likely to bring down the application.

Even after Step 1 is completed, several desirable qualities of a fully modernized app have yet to be achieved.
In practice, the benefits of portability at this stage aren’t as great as they may appear at first sight. This is because moving the binary is only a small part of moving workloads. Often, the processes and workflow, costs, dependencies, and security concerns that surround the binary aren’t themselves portable across environments.

Moving to monolithic containers is optional and makes tactical sense for some applications but not others. If an application is simple and the primary interest in containerizing it is for portability and OS-level independence, then undertaking this step makes sense.

If the application is complex and the primary interest in containerizing it is for fault tolerance, horizontal scalability, or improved reliability, then skipping this step may make more sense. In this latter case, time investment is relatively low (usually months) but the overall value added is also low.

<table>
<thead>
<tr>
<th>Area</th>
<th>Consideration</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hard-Coding</td>
<td>When taking the “heavyweight container” approach, hardcoded configuration files are still embedded creating dependencies that the container construct might not be able to functional with properly.</td>
</tr>
<tr>
<td>Start-Up Sequencing</td>
<td>Heavyweight containers must be sequenced during start-up, and manual actions to start the application are still likely needed. While some of this may be automated using tools like Chef, not all items might be feasible. Start-up Sequencing and Hard-Coding will make it as difficult as before to spin up new instances of the application so gains may be minimal.</td>
</tr>
<tr>
<td>Manual Build Still Required</td>
<td>Heavyweight container images are still likely to be built manually, so continuous integration and continuous delivery (CI/CD) are likely also manual, not yet fully automated. As containers are “pets,” multiple instances of each cannot be created on-demand, therefore the application cannot yet support fault tolerance, high availability (HA), and horizontal scalability. Similarly, updating the application code via new container images is still a manual, careful undertaking. Fundamentally, the application is still monolithic.</td>
</tr>
</tbody>
</table>
Step 2: Incrementally Decompose Heavyweight
Containers/VMs into Microservices

The true value of containerization is realized in Step 2, when the application is decomposed into microservices. The specifics steps to split apart a monolithic application are highly dependent on the app itself, but there are some general guiding strategies; see the “Guidelines and Tips” section above. The value added in step 2 is high, but the process is incremental and can take years not months.

Benefits
When a monolithic application has been decomposed into containerized microservices, the upshot is that the updated application is truly robust, scalable, and accommodates easy, fast upgrades thus enabling the organization to shift to more agile workflows. However, getting to this state is a progression that requires consideration for the underlying infrastructure and SDLC processes to support this effectively.

There are numerous benefits for the modernized application as containers are now managed as “cattle” and orchestrated by a technology such as Kubernetes:

- **Robustness:** Unhealthy containers can be automatically detected, terminated, and restarted. The application becomes self-healing, increasing uptime and improving the user experience.
- **Elastic scalability:** The application’s microservices can auto-scale up and down to meet variability in demand. End-user demand is automatically determined; when usage and latency thresholds are exceeded, the container orchestration layer spins up new instances of overloaded application components to smoothly handle load. This also means that end-users get consistently good performance when using the application—no more slow page-load times when demand is high. Further, the container orchestration layer can use computing resources more efficiently so there are fewer idle servers and ultimately less waste of resources.
- **Safe upgrades:** Upgrading the application is easier and safer, as containerized microservices lend themselves well to upgrade strategies like canary deployments, where new code is provisionally staged and tested end to end before being fully deployed—and where code that fails acceptance tests can be transparently rolled back with minimal user impact.

In short, while decomposing and refactoring the application into microservices is time-consuming, the resulting application is no longer monolithic and easier to build, test, and operate.

Considerations
On the next page are some areas where an organization will need to place careful consideration as these are often the areas that provide the largest challenge when incorporating containers within their modernization strategy.
In Step 2, enterprise need to understand the multitude of challenges that are likely to arise. Treating container instances as cattle requires the introduction of a container orchestration layer such as Kubernetes or Docker Swarm as well as a container networking layer such as Calico. These new, complex technologies necessitate large operational changes in how the application is managed. Even the decision to choose the specific orchestration and networking technologies requires significant due diligence, testing and experimentation before

Changes to the application code itself will be required. Containers impose their own security model, which is likely to violate assumptions in the application code. This manifests as a variety of permission errors. Additionally, container orchestration technologies manage the distribution and consumption of configuration information differently - how a service is registered and discovered - from traditional applications, and application code will need to change in response.

The mechanism of communication between containerized microservices—introduces its own set of challenges. The container orchestration layer will routinely spin up and down new instances of a microservice in order to satisfy load, execute application upgrades, or evacuate a container host for maintenance. Instances of any given microservice are basically anonymous and must be treated as cattle, just one member of a herd. This requires major changes in how microservices communicate with one another. There is no longer a single hostname or IP address that can be hard-coded into the application; the application instead needs to interface with the container networking and clustering layer to work properly in container constructs.

Because the orchestration layer manages containers as cattle, companies must change how they gather logs, collect and analyze performance data, monitor and alert on the health of the application. Further, decomposing the monolith into microservices reduces observability: when a problem with the application is observed, one must determine which microservice is at fault. Thus, there is a need for new tooling to troubleshoot the distributed system. Google built its own tool called Dapper\(^2\) to solve this problem; there are also open-source technologies such as Twitter's OpenZipkin\(^3\) that aim to improve observability and reduce time-to-detection (TTD).

It's very important to understand that some pieces of the application are hard to containerize, and in fact may never actually transition. Stateful services such as databases are difficult, since container orchestration layers were built with the assumption that containers are stateless (write no persistent data) and transient (can be terminated and recreated at any time). Currently, the industry doesn't have a robust solution to this problem, especially at enterprise scale. At the very least, extensive and thorough testing of containerized stateful services is critical due to the risk of data loss. Further, even putatively stateless services can cause data inconsistency in other parts of the application, absent code changes. For instance, even if a service keeps no state of its own but executes a sequence of remote state-changing operations, it may not be resilient to being terminated in the middle of the sequence causing data to become inconsistent. It must therefore

If an application has POWER components that run on AIX, these cannot be containerized. Similarly, components that depend on older operating systems or complex middleware and frameworks may be stubbornly resistant. Thus, it is crucial that companies plan for the eventuality of a hybrid application, composed mostly of containerized microservices but with some key pieces resident on traditional VMs or bare metal. In short, companies must have a safety net for the older pieces of the application. More on compatibility from Docker [here](#).
Summary: Multiple Technologies Will Be Used in All Stages of an Application Modernization Journey

There are different approaches available for modernizing traditional applications that do not require an application rewrite. Throughout this discussion, we've focused on containers and virtual machines which are the primary constructs often leveraged. It's often thought that it must be either VMs or Containers or Cloud Native Services when in reality it's a combination of these technologies that brings the application through modernization.

Skytap adds value at all phases of the containerization journey, from the very beginning where nothing is yet containerized, through intermediate phases where only some components of the application are containerized, up through full containerization of the application.

If an application is hosted within a Skytap environment, at the beginning and in the intermediate containerization phases, isolation and safety are provided. Since Skytap environments are fully self-contained, the containerization effort doesn’t halt or interfere with development against the traditional application hosted somewhere else (either in a separate Skytap environment or on-premises). A company doesn’t fall into a dichotomy of adding business value via their traditional, monolithic application vs. decomposing, modernizing, and containerizing it—they can occur in parallel.

It may well be the case that the final application is hybrid, where most of it has been decomposed into containerized microservices, a modern edge has been added using cloud native services but where there are one or more legacy pieces that should or must remain in VMs. Skytap environments are specially designed to support complex hybrid applications. Companies can rest assured that when using Skytap, their efforts to containerize their applications won’t be wasted or failures if there are pieces that resist containerization.

Dive deeper to gain additional knowledge on how to implement Skytap and Containers together by reading “A Practical Guide to Modernizing Traditional Applications Using Skytap Cloud, Containers, and CI/CD Tools”
Additional Resources

The State of Containers

Container Strategy for Executives

Continuous Delivery of Fully Functional Environments: How Skytap Uses Skytap Cloud to Achieve DevOps

How Skytap Complements, Not Competes with, Puppet and Chef

Knocking Down Our Own Skytap Stack with "Jenga"

How to Prepare Your Enterprise for a Docker Containers Initiative

Citations

1. Gartner - How to Prepare Your Enterprise for a Docker Containers Initiative, November 2017
3. How to Prepare Your Enterprise for a Docker Containers Initiative, November 2017
5. https://github.com/docker/communitytools-image2docker-win
   http://cloudscaling.com/blog/cloud-computing/the-history-of-pets-vs-cattle
   https://research.google.com/pubs/pub36356.htm