AN ARCHITECTS INTRO TO **CONTAINERS**

Authored by

Matt Leib - *The CTO Advisor Contributing Analyst*

Larry Smith Jr. - *The CTO Advisor Contributing Analyst*

Contributor

Keith Townsend - *The CTO Advisor Principal*

APRIL 2021

THE.CTO.ADVISOR

THE CTO ADVISOR

A Whitepaper on

Containerization

INDEX

Introduction

What's the key technical capability of containers? To paraphrase Bryan Lile, the 2019 Cloud Native Foundation Conference co-chair, "Containers allow organizations to create applications that solve business problems without consideration of the technical underlay."

(https://twitter.com/CTOAdvisor/status/1333548839287091200?s=20)

Most organizations are in search of that very capability, although it is not yet entirely possible. The current maturity level of containers doesn't allow organizations to simply ignore the number of deployments, load balancers, or even which infrastructure on which an application will be deployed.

This research note talks to the current state of containers and a subset of the many considerations on achieving unencumbering application developers' business outcomes from the minutia of IT infrastructure implementation details.

There have been many significant shifts in IT paradigms over the years. The movement to centralized storage, the concept of virtualization (which, of course, was a port of a well-established mainframe concept), the cloud, and more recently, the move to microservices and containers. Containers are a significant change in approach to application deployment and are generally embraced along with DevOps methodologies. Along with all significant shifts in approach come considerations that need to be evaluated. This document will discuss a couple of differing approaches and the potential gotchas therein.

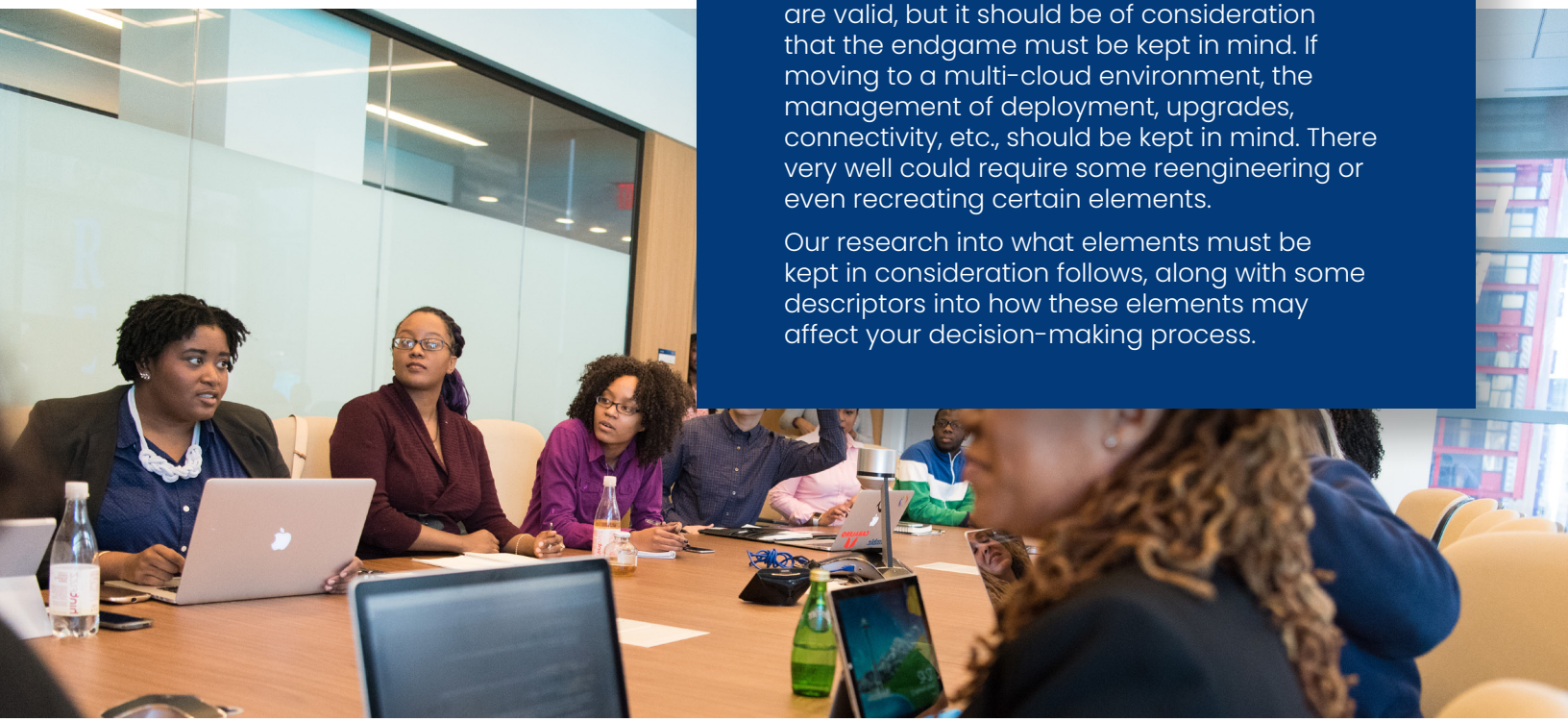
Our Research

Containers promise that the applications are far more dynamic than traditional, monolithic application rollout and upgrades. Micro-Services applications are designed to be updated in frequent small increments. As such, it can be fluidly updated or downgraded by the application team. Along with this is the potential promise of being able to stand up these applications using whichever on-premises platform or cloud provider suits your current needs.

And to be fair, in many cases, this has proven itself to be true, however not in every case.

The accepted approach to Containers has been to deploy using a cloud-based platform as a service. Many organizations are currently exploring the option of leveraging on-premises resources, with pre-packaged orchestration elements involved. Historically, the thought has been that the on-premises approach would be for sandboxing the approach, thinking that the micro-services dynamic to code generation, and ultimately approaching the orchestrational approach separately, either in tandem or afterward. Cloud-based platforms each have their unique orchestration, so that portion can be addressed independently. One approach that has become far more prevalent in current approaches has been that a centralized orchestration approach, regardless of where that platform will be located. All these options are valid, but it should be of consideration that the endgame must be kept in mind. If moving to a multi-cloud environment, the management of deployment, upgrades, connectivity, etc., should be kept in mind. There very well could require some reengineering or even recreating certain elements.

Our research into what elements must be kept in consideration follows, along with some descriptors into how these elements may affect your decision-making process.



Deployment Mechanisms

One aspect of deployment is building, testing, and updating the container images that make up your application. You must decide whether you will rely on traditional manual processes or go all-in with automated processes. The continuation of manual processes will remain highly prone to non-repeatable and inconsistent deployments, which in turn bring unacceptable risk to your application. Even if you have been relying on manual processes up to this point, container adoption is an ideal time to begin leveraging automated processes to ensure repeatability and consistency safely. As you embrace these automated processes, ensure installing all of your source files within source code repositories and leveraging automated CI/CD pipelines for deployments. These CI/CD pipelines provide the deployment mechanism in which your containers are managed while at the same time providing visibility into failures for everyone to clearly see and come together to solve.

Another aspect is the building, testing, and deployment of the host operating system on which the containers will run. These hosts should be managed in a similar, automated fashion, referred to as an Infrastructure as Code methodology. Initially, you may have separate pipelines for infrastructure and applications. The goal is to bring infrastructure deployments into a continuous workflow that will also deploy your containers once the infrastructure has been provisioned.

In a recent CTO Advisor report, we outlined moving to a hybrid infrastructure for VMware-based workloads where the shared datacenter being leveraged could be any of the four major providers: AWS (Amazon Web Services), Azure (Microsoft's cloud-based architecture), GCP (Google Cloud Provider), and the newcomer on the block OCVS (Oracle Cloud VMware Solution). That report can be found here. This report's relevance as it relates to today's conversation comes from the direction that VMware has outlined, involving project Tanzu, and the incorporation of a Kubernetes infrastructure housed within a VMware infrastructure, and the agility of leveraging this on any of the cloud providers. If you can take your VMware architecture and run it on a cloud provider's architecture, then you can do so with the Tanzu platform in a hybrid/ multi-cloud environment as well.

Compute Resources

How will you run your container platform? Will you deploy onto bare-metal servers or VMs? This question is sometimes highly controversial because many will argue for VMs or bare metal, but the reality is that it depends.

Do performance or scale requirements dictate that bare metal should be used? There are often no requirements that would justify bare metal, so it becomes a political requirement. The thing here to remember is, BARE METAL is HARD! Especially if you are not leveraging automated processes to provision the bare-metal servers. VMs, on the other hand, are, in most cases, much simpler to provision from a template. Granted, VMs require a hypervisor deployed onto bare metal. However, this is a problem solved long ago for most IT organizations.

Compute resources, whether bare metal, VMs or public cloud, need to provide a similar container application platform. This is extremely important to ensure flexibility between cloud providers and on-premises. Differences in storage and networking capabilities between your container platforms will impede application mobility or limit the locations a containerized application can be deployed.

Backup, DR and Replication (Stateful versus Stateless)

Once you have applications that use containers, you must protect the application and its data.

The container instances are usually recreated from an image registry or source code, either to their original location or to a new location for DR. Any persistent data will require protection too, either data volumes attached to containers or data persistence in databases or file servers.

Embracing Infrastructure as Code, source control, and CI/CD pipelines enables container-based applications to be rebuilt from source code, provided the source repositories are available after a disaster.

Always factor data protection and disaster recovery into your source management and container platform plans. Stateless container applications can be simply redeployed and connected to external data sources. However, many applications have persistent data attached to containers which makes the containers stateful. Protecting the container state data can be complicated, particularly if recovery will be to a different container platform, i.e., from on-premises to the public cloud.

Protecting external data persistence is no different for container-based applications than traditional applications.

File servers, database servers, and object stores should be protected with backups and replication based on the application's DR policy. The possible complication is that a micro-services architecture may lead to a greater variety of data persistence services, possibly adding NOSQL databases or object storage that the organization did not previously use. These newer technologies usually have a different approach to data protection and availability that must be understood before they are adopted.

Protecting storage that is attached directly to containers is a little more complex. While most containers do not have persistent storage attached, stateful containers have their own data that must be protected. It is essential to identify the uses where the data attached to the container is a cache that can be recreated or is the primary copy of valuable data. Usually, only the primary data requires protection unless recreating the cache causes a performance problem at failover.

Should that container-based application be housed solely within one datacenter, the dataset can be stored in a single location. Any further deployment only needs to point to the existing dataset rather than involving the replication and the inherent challenges with cache coherency between physical locations.

It should be noted that there are some prerequisites and potential benefits that can be leveraged if that infrastructure is based on specific hardware platforms. Both HPE and NetApp, for example, require the implementation of what's known as a "Fog Layer," which is designed to ensure that the replication of data from site to site and the goal of keeping the data consistent across all target datacenter homes. As solutions like these mature, some of the orchestration layers may prove to be designed to work best with one or all of the solutions being used by these architectures. These are yet to solidify.

Networking

Networking

Containers add a further level of virtual networking within each container host which must be configured to control access to individual containers. This is in addition to any hypervisor-based network and the physical network. Ideally, the container networking should be integrated with the other layers through networking plugins to the container platform. In cloud services, the provider usually manages this integration; on-premises, it is another design element.

Applications running in containers are accessed over the network in the same way that traditional applications are accessed. Generally, there is a network load balancer as the first point of access. The significant difference comes when micro-services are deployed, each micro-service communicates with other microservices over the network. Having a consistent strategy for enabling, securing, and controlling communication between microservices will enable portability between container platforms.

You will likely control the firewall rules via the container host or an external control mechanism. On-premises you might still manage firewall rules via traditional physical firewalls, but you could still bring those rules all the way down into the host, depending on the solution. If the solution you are using provides visibility into the containers at the host level, your rules can become extremely granular. These solutions can sometimes also apply to the public cloud, but you might also need to leverage cloud-native features such as security groups, etc.

Another networking concept that you'll likely need to explore when using containers is a service mesh. Service meshes provide functionality such as load balancing and service discovery to dynamically provide services externally to your container platform without the need to configure load balancing rules, etc. A service mesh can also provide DNS resolution for locating resources externally, which is also dynamically managed. Service meshes ultimately allow your application services to be discovered and consumed independent of the underlying infrastructure.

Each cloud provider has unique capabilities regarding networking, bandwidth requirements, firewall/ fault tolerance, etc. Suppose specific networking functionality is required to make your application function properly. In that case, the administrator will need to understand the specific requirements and account for those changes in the deployment on different platforms.

Should that application be deployed to a second locale within the same infrastructure, all security and networking parameters would be already established, and thus, consistency can be maintained.

Security

As with deployment, security must separately address the container hosts and the container images. Container hosts do not require any application component installs, so only the base operating system, container runtime, and infrastructure management tools will require configuration and security patching. The container hosts should be managed similarly to any other operating system, a well-understood enterprise operations activity. Using clouds with container-as-a-service offerings makes this patching the cloud provider's responsibility.

Container images cannot effectively be patched in place; the image is treated as an immutable object. To remediate an image, a new version of the image must be generated and distributed. This is an apparent reason to adopt automated lifecycle management for container images, with the work done by CI/CD pipelines.

Security vulnerabilities may enter your container images from your application code and its dependencies or from public base images. Vulnerability scanning should be part of the CI/CD pipeline and should be applied to container image registries to check for freshly discovered vulnerabilities.

Another area of concern when dealing with security within your container images is to ensure that you never hard code any credentials, keys, tokens, certs, etc. These elements should be injected into the container at runtime from an external system or another applicable method.

Stor
age

Storage

Container hosts require storage for container images and any transient files created by your application. Typically, this is not a large amount of storage, but if it is exhausted, all containers on the host will fail. For on-premises deployment, plan to regularly clean up images and leftover containers on each host or redeploy container hosts.

Containers that have persistent storage attached present more of a challenge, particularly for high-availability and DR requirements. Persistent storage is presented to containers through a storage plugin from your storage provider. Consider whether you have compatibility between different platforms, on-premises, and public cloud.

Monitoring/Telemetry/ Logging (Observability)

Once again, normal host and infrastructure monitoring are required, while a container-based application will need a new approach to ongoing management. The distributed nature of container and microservice applications adds complexity to issue resolution, bringing the observability approach, which adds request tracing to the usual logging and metric collection as data sources.

On-premises you may choose to use a container management platform that includes tracing, such as RedHat OpenShift or a dedicated observability platform like honeycomb.io. Some public cloud providers have frameworks for observability; however, these seem to be tied to the specific cloud provider, preventing a unified view across multiple providers. Observability is an immature market, over time major vendors will undoubtedly add to their monitoring tools.

Summary

Building applications in containers, particularly in a multi-cloud environment, requires significant changes from monolithic applications. The CTO Advisor will always recommend doing your research and evaluating the ramifications of adopting a new platform. Containers and microservices are not a one-size-fits-all technology and may require a significant change to deliver value.

Moving to DevOps methodologies with Infrastructure and Code and automated software development lifecycle requires significant people and process change to get the greatest business benefit. Security and data protection must be built into your container workflow from the beginning, and the public cloud may be the best platform for your containerized application.

Findings

The purpose of this document has been to ensure our audience understands not just their motivation for moving toward a container-based micro-services model but as well some of the more significant requirements for providing a platform for these applications.

An executive commitment to container-based applications is essential to the organizational changes required. Changes will be required in the way that operations and infrastructure teams work, as well as the software development and architecture teams. Remember, the agility of the micro-services approach can be hugely beneficial, but it isn't necessarily the only approach. It could be that the shift in approach may be wrong at this point for many organizations. As in many emerging technologies, we do believe that maybe the organization should take a wait and see where maturation takes place.

As we know, the agile approach with micro-services can significantly decrease the frequency and impact for both the deployment of fixes and the roll-back of faulty code. Reducing remedial work and focusing on feature development can genuinely aid in the innovative and dynamic nature of many organizations. All of the in-house developers' efforts will be stymied if the container platform is unable to accommodate the rate of change or exposes the business to excessive risk.