Zerto

The State of IT Resilience for Kubernetes

Version 1.0
June 2020

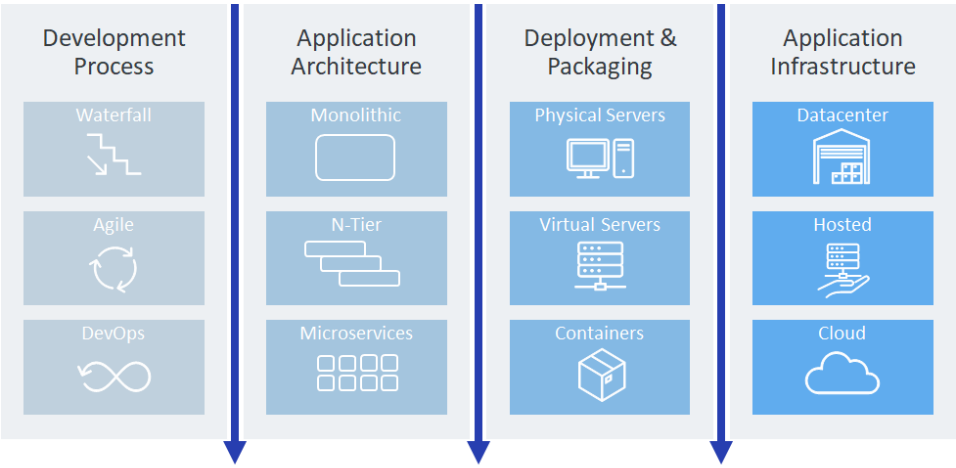
Zerto

Table of Contents

Container Adoption

Container technologies enable application developers to package small, focused code into independent, portable modules that include everything that's needed to run the code—and, critically, only what's needed to run the code. Containers are becoming increasingly popular as organizations discover the technical and organizational advantages of building applications with containers.

The major force driving containerization is the adoption of DevOps practices focused on accelerating the software development lifecycle, such as the adoption of cloud-native microservices.



Caption: Containers are one of the next-gen technologies and practices for developing and deploying applications

Container usage is expected to increase by 89% in the next 2 years[1], and 90% of all applications will feature microservices architecture by 2022, according to IDC.[2] And containers are replacing virtual machines (VMs) already, says 451 Research.[3]

Yet many IT organizations don't currently have container strategies, nor do they know how to manage containers at scale or how to protect and recover modern applications and persistent data.

No wonder that for many, bottom-up adoption means 'containers just happen.' Not unlike the cloud revolution that led to massive shadow IT, the container revolution could lead to uncontrolled, grassroots use of containers across the organization, with some undesired effects.

The question is, how do you protect persistent data of containerized applications while giving developers the freedom to use containers in their workflows?

The way we build and run applications has changed dramatically because of the lightweight, modular approach of containers. Let's take a brief look at why containers, and Kubernetes in particular, have amassed popularity among the developer and DevOps crowds from three different perspectives:

1. Developer Agility
2. Decoupling of Layers
3. Microservices

Container Advantages: Developer Agility

IT and DevOps organizations have vastly different needs and are often siloed departments. There's always room for more collaboration and alignment.

IT is judged on the quality of service: availability, performance, service level, response times, and more. IT is challenged by the complexity of their environments, sometimes with legacy applications and aging infrastructure. Add the high stakes of downtime or IT otherwise not working and it's no wonder IT wants to control change management processes tightly.

On the other hand, development teams embrace change constantly by creating new features or functionality, fixing bugs, and eliminating security issues.

Developers are judged on output: things like velocity, productivity, time-to-market, quality of the code. They care less about the infrastructure aspects of their application, or at least want to minimize the amount of time it takes for deploying and configuring the infrastructure required for their applications. Instead, they need those components to 'just work' and be there in order to successfully build and run applications. Development teams are playing a more prominent role in infrastructure, defining what the infrastructure needs to look like, and they often need to build infrastructure stacks for their applications specifically.

This speed puts pressure on traditional IT and its systems. They were never designed to cope with these requirements of agility, yet the capital investment and complexity make them inert and hard to change—at least until the infrastructure nears the end of the economical lifespan.

Public cloud computing has tipped the scales definitively, though. How can a small IT infrastructure team compete with the massive scale and budgets? The public cloud offerings excel in on-demand self-service with rapid elasticity and scalability, things where in-house teams traditionally struggle. No wonder developers have resorted to shadow IT, using cloud resources behind IT's back: they enabled developers to not have to care about infrastructure, automating the toil and hiding complexity behind a well thought out service offering.

And to circle back to why containers make sense in this context: separation of concern. Containers allow IT to define the base images without needing to know the requirements of development teams. Development teams can take the container images and use them to build their own stacks, adding middleware and other dependencies to the base image without involving the infrastructure teams.

Container Advantages: Decoupling of Layers

Containers enable this separation of concern by using image layers. Layers can have different owners, and layers can be stacked on top of each other.

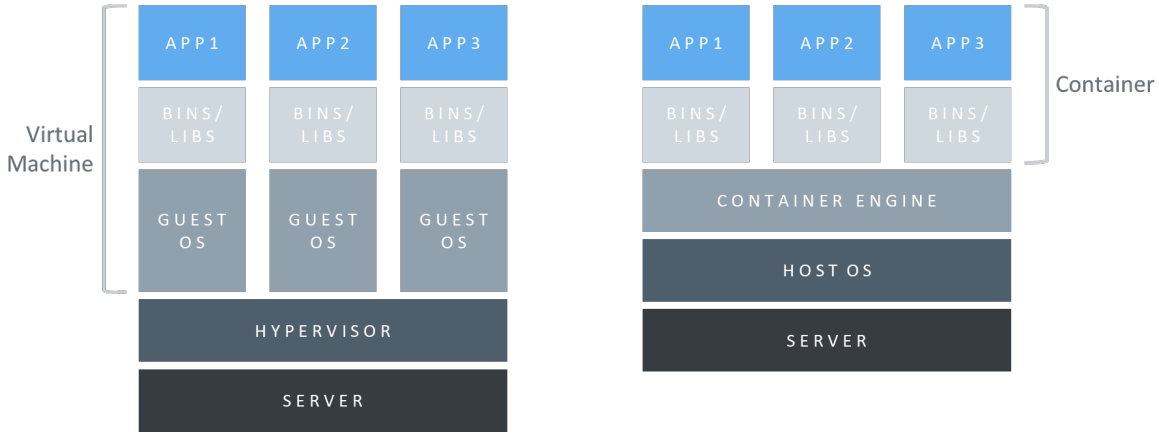
Compared to more traditional physical and VMs where storage layers are hard to separate, container images are a radical change in the way infrastructure is composed.

The tight coupling of operating system, middleware, application binaries, application configuration and application data means that building disk images is a complex, slow process. For this reason, once a VM is built, configured, and running, it usually lives on for months or years. Often, VMs aren't re-built for upgrades to the operating system, middleware, database services, or other major parts of a VM, but instead upgraded in place.

At scale, the perennial nature of VMs begins to show its shortcomings. Over time, small differences in scope and initial deployment tend to occur, drifting the configuration and building up technical debt, such as version conflicts, stability

problems, and unused files (posing security risks). These often result in major headaches and frustration for the development teams.

Without a good way to separate responsibilities, building VMs is mainly a task for IT operations, not developers. This technical complexity leads to an organizational disconnect, and developers often take matters into their own hands.



Caption: Unlike VMs, containers only include the minimum code and dependencies required in order to run, not an entire OS, and are thus more lightweight and portable.

With containers, the build process becomes much easier. Containers are made up of several immutable 'layers' of a disk image. Each layer is defined by a base image, plus any changes, such as additional software or configuration changes. These changes are codified in a text document, like a Dockerfile. And a little higher up the stack, Kubernetes defines entire applications, with potentially many different container images, using similar text files.

This high level of automation, and the ability to create images autonomously (based on IT-approved base container images), makes developers happy: they get what they need without toil. The immutable aspect of these containers turns what previously took weeks to request a VM with the right software into an off-the-shelf component with the container image, ready to be used for releases, deployments, testing, and other use cases.

Containers are ideal in this sense because the infrastructure team can deliver an artifact, a ready-to-go image for development teams to use and build on. These container image layers can be re-used across a wide number of applications; for example, a base Linux container that is used across all the enterprise Kubernetes estate.

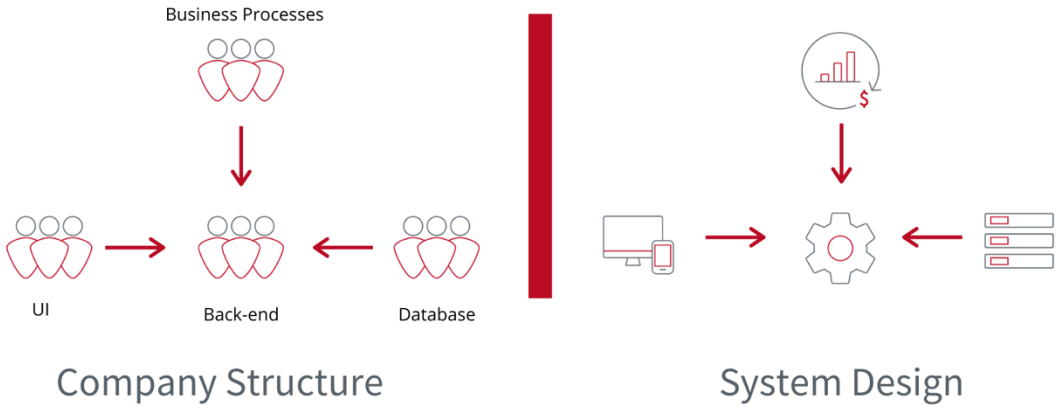
Updates to the images from the infrastructure teams happen at their own pace and according to their own processes, but are picked up automatically in the development pipeline so developers are always using a secure, up-to-date version without any effort on their part, while maintaining any changes made in the additional layers.

It also allows developers to slim down the scope of their builds. Instead of massive images that contain everything from bare operating system to their latest application build, developers only need to package up their application into a container image. This reduces complexity, speeds up deployments, and reduces security risks.

Operational friction between development and infrastructure teams, an unnecessarily large and perennial infrastructure environment, and lack of decoupling between layers are all in sharp contrast with how lean and agile software development works. It's no surprise, then, that the traditional approach no longer works for modern software development.

Container Advantages: Cloud-Native Microservices

Many developers are choosing a more proactive approach to removing organizational complexity by breaking down their work into smaller chunks, creating more flow in the software development lifecycle, removing manual processes, and taking ownership of all dependencies in the pipeline to bring code from local development all the way to production. Containers, microservices, and cloud-native application design are facilitating this.



Caption: container adoption presents the opportunity for system design to drive organizational structure rather than vice versa, as in Conway's Law.

A microservices architecture breaks down application components into its smallest parts and assigns ownership of each microservice to a multidisciplinary team. These microservices are loosely coupled, small, and independent pieces of a larger network of services that make up an application. This allows each team to operate independently from other teams in developing and releasing software, reducing the time-to-market for new code. No wonder that for many developers, Kubernetes means less friction than they were accustomed to in a VM world.

In essence, coupling of microservices to multidisciplinary teams is a sort of reverse Conway's Law, which states that 'any organization that designs a system will inevitably produce a design whose structure is a copy of the organization's communication structure.' Here the opportunity lies in structuring the communication around the emerging developer practices.

And containerizing these microservices has become standard practice, leading to an explosion of the number of containers. Not unlike the VM sprawl that happened ten years ago, container sprawl forces organizations to re-think their container management strategy.

Many inevitably land on using Kubernetes, the de facto standard for container orchestration: 71% of more than 200 enterprise decision makers surveyed by 451 Research indicated they were using Kubernetes to manage their container infrastructure.[4]

Kubernetes is an open source platform for managing containerized applications and relies heavily on declarative configuration and automation to deploy and manage containers. Kubernetes has a vibrant ecosystem, much of which is open source as well. Most notable in this context are the Container Storage Interface (CSI) for providing storage services to Linux containers, and Helm, a package manager to find, share, and use software built for Kubernetes.

Kubernetes is the orchestration layer that manages containers across a group of physical or virtual servers. Kubernetes is specifically designed to manage the ephemeral nature of thousands of containers spinning up, scaling up, and winding down. Kubernetes manages versioning of containers, figures out how containers should talk to each other over a network, exposes services running in the containers, and handles storage considerations. It also handles hardware failures and maintains container resilience.

But as with every technical platform, organizational challenges run the risk of remaining unsolved. With microservices, Kubernetes, and multidisciplinary teams, teams run the risk of not implementing all the best practices from an IT operations or security perspective.

The Data Protection Challenges of Containers

One of the major areas where development and DevOps teams may fall short is data storage and data protection. Although Kubernetes includes some limited options for these, enterprises are finding gaps when it comes to true end-to-end protection and resilience.

There are a number of reasons why data storage and data protection in container environments is challenging, starting with the fact it's very different than how storage in VM environments work. And unlike mature virtual environments, Kubernetes has fewer guardrails to make sure that new workloads are configured correctly for data protection.

Let's look at how persistent storage differs between Kubernetes and virtualization platforms such as VMware vSphere or Microsoft Hyper-V.

Containers & pipelines

A major difference stems from a container's immutability. Container images are immutable layers of a fully documented process of installation and configuration. So instead of merely capturing the end result, the container image, it makes more sense to protect the factory that produces the images, including all the configuration scripts (such as Dockerfiles and Kubernetes YAML files) and documentation, also known as a pipeline.

Secondly, the oft-forgotten aspects of data protection for containers are the systems that create the containers as part of the CI/CD pipeline. These are tools like code repositories, build servers (such as Jenkins), and artifact repositories that store containers and application releases. By protecting these workloads, most of the 'factory' that produces container images is protected.

Lastly, it also makes sense to capture the current state of running containers to avoid having to rebuild the entire deployment of many applications and even more containers after a catastrophic failure.

Stateless vs. Stateful

However, after protecting the factory and the running state, we're not there yet. There's still a major piece of the puzzle missing: protecting persistent application data.

In the earlier phases of container adoption, it was often believed that containers were only suitable for stateless workloads, and that storing any data in a container was impossible. We now know this is wrong: both the underlying container runtime and Kubernetes itself fully support a diverse variety of workloads, including stateful applications.

While container images themselves are ephemeral—meaning that any file system changes are lost after the running container is deleted—there are plenty of options for adding stateful, persistent storage to a container. Even enterprise storage arrays already in use in on-premises datacenters can often provide stateful storage to Kubernetes clusters.

In Kubernetes, persistent data is presented using the concept of volumes. A volume, at its core, is just a directory on the container host, with some data in it, accessible to a container. How that directory comes to be, the medium that backs it, and its contents are determined by the particular volume type used. PersistentVolumes (PVs) tie into an existing storage resource. They're cluster-wide objects linked to the backing storage provider that make these resources available for consumption. A PersistentVolumeClaim (PVC) is a storage consumption request from an application.

Organizational Alignment of Cloud Services

It's tempting to use a cloud storage service for object or file storage. It's quick, it's easy. It's cheap (at least when you start out). However, a potential challenge arises when that cloud storage service remains outside of the control of those responsible for data protection. Invisible persistent storage resources lead to a risk of unprotected and insecure data, without backup, disaster recovery, application mobility, and more.

Managing cloud storage is just as hard to do right as enterprise, on-premises storage. Organizations must ensure a consistent approach to accessing and managing cloud storage so developers can use the services they need while IT at-large can maintain oversight, security, and overall responsibility.

Data Protection and Disaster Recovery

Looking back at how different container storage is compared to more traditional virtualization platforms, we see that container storage is no less simple, and perhaps even more complex, because the container data protection ecosystem is much less mature than their virtualization counterparts.

There are many solutions in the virtualization space that tie into the hypervisor and the storage platform to optimize data protection and disaster recovery for virtualized workloads.

Simply porting over those solutions does not work for containerized applications given the major technical differences between VMs and containers. Container-native data protection requires a re-write to accommodate for those changes.

For example, one of the most powerful characteristics of Kubernetes is that everything is configured using declarative configuration code—i.e. infrastructure as code. By merely retrofitting a data protection solution and forcing developers to use a separate interface to define data protection, you're not taking advantage of the new paradigm Kubernetes has to offer.

And this works the other way around, too. Why re-invent the wheel for containers where the wheel does not need reinventing? A good example is a journaling engine for granular point-in-time recovery.

Challenges Summary

We've learned that containers are becoming immensely popular because they enable developers to create the required infrastructure for their applications quickly and easily. Containers are easy to build automatically and repeatedly, and their combination of both ephemeral and immutable characteristics make them a breeze to work with in production, minimizing technical debt and configuration drift over time.

Yet many IT organizations don't have any idea how to protect, migrate, and recover these modern applications and their persistent data. And without a good data protection strategy for containerized applications, organizations risk their data and they risk jeopardizing all the benefits they've gained by adopting containers in the first place.

The question is, how do you protect persistent data of containerized applications while giving developers the freedom to use containers in their workflows? How do you make sure developers have no way to opt out of using data protection by making it a seamless, automatic, and transparent part of their workflows?

Key Considerations for Kubernetes data protection

It's clear that containerized applications require disaster recovery, data protection, and mobility. Simply opting for non-native solutions from legacy backup and disaster recovery providers will only add time, resources, and barriers to application development and delivery. Selecting the right data protection solution makes a substantial difference in an organization's agility.

Infrastructure and operations (IO) organizations must look for a platform that delivers the necessary availability and resilience without sacrificing the development speed of enterprise applications and services. This means being able to protect, recover, and move their containers without adding more steps, tools, and policies to the DevOps organizations. The following key considerations provide solution requirements that will provide both IT/IO and DevOps teams with the capabilities and outcomes they're looking for.

Data Protection as Code

Reducing application downtime and data loss is a priority for any application, especially containerized ones. Containers are not immune to the usual challenges of cyber-attacks, accidental deletions, infrastructure failures, etc. It's crucial that organizations utilize a platform that delivers the ability to simply resume operations.

Using a native solution will drive a "data protection as code" strategy. This means data protection and disaster recovery operations are integrated into the application development lifecycle from day one—applications are born protected. Organizations using this approach will be able to ensure the resilience of their applications without sacrificing the agility, speed, and scale of containerized applications.

Replication and Journaling Technology

Utilizing replication technology will give you the freedom to simply rewind to a previous checkpoint, delivering a low recovery point objective (RPO). This approach is non-disruptive and gives you more flexibility and availability than a traditional backup approach using snapshots that can be potentially hours behind production systems, leaving gaps in data protection. Continuous data protection (CDP) has long been the gold standard in the VM world, and it should be no surprise it's emerging as the best option for containers as well.

Application-centric Protection and Mobility

Utilizing application-centric protection is key. True protection shouldn't just include persistent data, it must also provide the ability to protect, move, and recover containerized applications as one consistent entity, including all associated Kubernetes objects and metadata.

Avoid Vendor Lock-in

When evaluating a platform, ensure that it supports all enterprise Kubernetes platforms and allows data to move to where the application needs to run, without any lock-in to a specific storage platform or cloud vendor so the persistent data is as mobile as the containers themselves. This means protection and support across leading platforms, such as:

- Microsoft Azure Kubernetes Service (AKS)
- Amazon Elastic Kubernetes Service (EKS)
- Google Kubernetes Engine (GKE)
- VMware Tanzu
- IBM Cloud Kubernetes Service
- Red Hat OpenShift

Regardless of the platform, being enterprise-class means being technology-agnostic. DevOps needs flexibility and agility, and data protection should follow suit.

Conclusion

Containers and Kubernetes are here to stay. Developers love the platform because it gives them the freedom they need to create, build, and run applications quickly. Developers share custody of the Kubernetes platform though, because IT admins are responsible for staying in control and taking care of business continuity.

In the future, Zerto will help solve the complex enterprise challenges of containerized applications to ensure developers can continue to use Kubernetes and containers without the need to change their workflows, while guaranteeing data protection and compliance are covered.

Zerto announced plans to deliver resilience for next-gen applications with its first release to deliver data protection, disaster recovery, and mobility for Kubernetes applications, whether on-premises or in the cloud. Easily protect, recover, and move any Kubernetes application and its persistent data for accelerated delivery within your development lifecycle.

For more information, visit Zerto's website and follow the blog series on data protection and disaster recovery for Kubernetes applications.

[1] Red Hat Global Customer Tech Outlook 2019

[2] IDC FutureScape: Worldwide IT Industry 2019 Predictions

[3] https://451research.com/images/Marketing/press_releases/Application-container-market-will-reach-2-7bn-in-2020_final_graphic.pdf

[4] 451 Research. "Hybrid Cloud Drives Growing Container Production Use and Disruption, May 20, 2020

About Zerto

Zerto helps customers accelerate IT transformation by eliminating the risk and complexity of modernization and cloud adoption. By replacing multiple legacy solutions with a single IT Resilience Platform, Zerto is changing the way disaster recovery, data protection and cloud are managed. With enterprise scale, Zerto's software platform delivers continuous availability for an always-on customer experience while simplifying workload mobility to protect, recover and move applications freely across hybrid and multi-clouds. **www.zerto.com**

IS-14266