Modern Applications: From Monolith to Microservices

Enterprise IT is changing. It's a continuous cycle and has happened many times over in the last decade. First, organizations virtualized data centers. Then, they moved some (or even most) workloads to the cloud. And now it's happening again as organizations re-architect those applications—no matter where they run—from monolithic systems to microservices managed by small DevOps teams driving their own technology buying decisions.

It's all part of a broader trend of digital transformation that often involves technologies like containers and Kubernetes. If you're not yet excited about this change, you should be. Applications that run in a *cloud-native* manner are more resilient, more automated, more secure, and generally more fun to build and run once you've mastered the basics.

At Pure, we're so excited about the trend toward cloud-native applications that we acquired Portworx®, the industry's number one Kubernetes data-services platform.

We have lots to share about why you should add Portworx into your mix of critical application-infrastructure technologies. But we realize that some background might be helpful first. Over the next few weeks, we'll provide all the background you need so you're as excited about this latest shift in enterprise IT—and Portworx—as we are. Here's what the series will cover:

- Part 1: The shift from monolithic applications to microservices
- Part 2: How containers and Kubernetes accelerate microservice applications
- Part 3: Storage and data requirements for Kubernetes apps

- Part 4: Portworx, the Data Services Platform for Kubernetes

Let's get started.

The Problem with Monoliths

A monolithic application is an app where all the code elements (user interface, business logic, and data access) are combined into a tightly coupled system. Sometimes it's all in a single program. Other times it's a set of highly interconnected services with many mutual dependencies, such as a single Oracle database.

If that seems abstract, there's an easy way to know if an application is monolithic: *maintenance notifications.* Because the entire application is a single program or tightly coupled, updating even a single line of code requires redeploying the entire application. That makes updates complex and time-consuming, and IT teams will usually warn users with frustrating messages like, "This application may be unavailable from Friday at 5 pm to Monday at 8 am due to an upgrade." Because of the disruption this unavailability causes, these updates are rare—often quarterly or annually—and new features must wait until the next release. As a result, monolithic applications are often seen as slow to respond to user and business needs.

Enter Microservices

When was the last time you or a family member got a maintenance notification from Netflix, Disney+, or Roblox? Exactly. There's never a good time to update these services because someone is always binge-watching a new show or in the middle of an epic game. If there's never a good time to do an update, it's actually always the best time to do an update. It sounds counterintuitive. But to solve the update problem, the principle of microservices states that you should break an application into smaller pieces that communicate via APIs, where each part can be updated independently from other parts. As a result, if Roblox, a gaming platform with more than 150 million monthly active users (and a Portworx customer), needs to update its password-reset functionality, it doesn't need to kick millions of kids offline. This feature is a different microservice that Roblox can update independently of the rest of the gaming platform. And that means happy devs and happy kids.

To generalize beyond simply avoiding maintenance windows, the benefits of microservices come in two big categories: agility and resilience.

Agility

Adrian Cockcroft was one of the biggest proponents of microservices during his time as the chief cloud architect at Netflix. Cockroft has said that to achieve speed and outpace the competition, "everything basically is subservient to the need to be able to make decisions, and build things, faster than anyone else." You don't win by out-planning the competition; you win by out-executing them.

This agility is the first benefit of microservice architectures. Martin Fowler, another well-known proponent of microservices, advocates decoupling a running system into "a suite of small services" that operate independently and communicate via APIs. By limiting the dependencies of any service on other parts of the system, you can change microservice architectures quickly in response to a feature request or a newly discovered bug. With monolith applications, "a change made to a small part of the application, requires the entire monolith to be rebuilt and deployed." Ugh.

Resilience

Microservice architectures are not only more agile but also more robust and resilient.

Because microservices are expected to fail, they "need to be designed so that they can tolerate the failure of [other] services," says Fowler. IT teams ensured the uptime of monolithic applications by purchasing bigger and bigger servers, using clustering across data centers, and making many, many copies of the data for each environment. Cloud-native applications, however, increasingly run on unreliable virtual machine instances or "bare metal" servers that were not architected for resiliency. Netflix famously embraced this concept with its Chaos Monkey toolkit that "randomly terminates virtual machine instances and containers that run inside of your production environment. Exposing engineers to failures more frequently incentivizes them to build resilient services."

Microservices enable IT teams to more easily build and run the applications their users want and need to stay ahead of competitors. Many of the largest consumer and enterprise applications today run in microservices, proving that it's not just a trend for small organizations but also for the largest and most complex. Indeed, the larger the organization is, the more benefits there are to gain from adopting microservices because teams are often spread out with limited direct communication. Using microservices is a great way for these teams to communicate via naturally occurring organizational boundaries. This is sometimes referred to as the "two-pizza team" approach to management: If you need to order more than two pizzas to feed the team building a service, then the service is too big. When it comes to microservices, let your stomach be your guide.

In Part 2 of this series, we'll look at how containers and Kubernetes evolved along with microservices to provide the perfect building blocks and management tools for modern applications.