GETTING STARTED WITH CONTAINERS AND MICROSERVICES FOR ENTERPRISE LEADERS

In the world of software development, containers and microservices are getting a lot of attention. While traditional software architecture can feel clunky and slow-moving in a market that requires rapid response, agile DevOps teams are embracing containers and microservices to stay ahead of the competitive curve.

Let's take a look at what enterprise leaders should understand and consider when getting started with containers and microservices.

Solving the problems of monolith applications

For decades, software developers worked within large, monolith applications that contain all the code and libraries for every single activity or functionality the application could perform. It may have a few different components, but they generally are built and managed by one team.

When an app succeeds, which is the goal, more users download it, so you have significantly more traffic. Soon this traffic means a major increase in user-requested features and improvements. (The dedicated team will be busy trying to keep up!) These increases in fixes, improvements, and new features means that monolith application just got a lot more complex --- and probably a lot clunkier to boot, both in use and in management, with additional teams supporting it simultaneously.

Here's where real problems come in: if several dev teams are working on a single app, no matter

how big, each change in code can ripple across the app, creating unplanned or unforeseen ramifications. Code quality can go downhill quickly, resulting in reduced application and availability for end users – and a major dent to your bottom line.

Today, software dev teams are utilizing microservices and containers, usually together, to avoid this systemic problem and better incorporate DevOps principles.

Understanding containers and microservices

Unlike the monolith structure of traditional applications, mobile and desktop apps are frequently deployed using a combination of containers and microservices. So, what do these buzzwords means – and what do they promise?

Let's start with microservices, which is usually shorthand for 'microservice architecture'. In contrast to a traditional monolith environment, microservices encompass any way of designing applications as a suite of services, which can all be deployed independently.

Though these do play out in different technical ways, microservices tend to share certain characteristics, such as organization around business capability, automated deployment, decentralized language and data control, and endpoint intelligence. The apps, typically with a single function, rely on small, self-contained units that are connected via APIs, which means they aren't dependent on a specific language. Microservices rely on the least number of libraries and executables necessary and programmers can code them however it works easiest within fitting into a specific language and framework. The tiny size of microservices (hence the name) also ensures speed and agility.

Importantly, containers aren't the same as microservices. Instead, containers are isolated environments that exist in a virtualized operating system. Their virtualization is essential: because containers aren't tied to any software or physical machines, they solve many challenges around app portability. Containers are nimble, too, so they can be spun up in no time and speedily handles workload processes and application delivery. (A popular container system is Docker.)

Microservices and containers exist independently and function differently. Microservices can certainly work on their own, but containers can easily support and manage the deployment of microservices, especially when you have an app that relies on a variety of microservices, like delivering an app or migrating legacy systems to cloud services. Though different technical components, they offer similar benefits like increased efficiencies, speedy delivery, and ability to do more with existing resources – which are vital in an agile cloud environment. Scaling up becomes a lot more straightforward and less resource-heavy.

Of course, microservices and containers have their challenges. As new technologies, most developers are just learning how to utilize these tools. Especially during the learning curve, your enterprise must continue to maintain data security, system integrity, and normal service levels. If containers are the step above microservices in a software supply chain, programmers will also need to consider container orchestration – the higher-up coordination and execution of containers across nodes. (Kubernetes is the most well-known container orchestration tool.)

Longer-term, programmers warn of application sprawl (the more microservices and containers, the more complex your environment), problematic code that can snowball into technical debt, and other unforeseen complications if you don't establish proper monitoring and management.

Implementing microservices and containers

Success with microservices and containers hinges both on effective implementation as well as ongoing monitoring. Because microservices are so agile, you may continually opt for them, only to realize too late how much complexity they can add to your applications. To prepare for this, enterprise leadership generally have three areas of responsibility when it comes to successful software development:

1. **Managing resources.** This starts with the technical side, for instance: determining how much and which workloads to move to microservices, where to deploy containers, and how these changes will impact your current technology stack.
2. **Developing strategy.** Your strategy should offer a macro vision that considers whether all teams are using this technology to align with enterprise needs (not for the sake of new tech) and if team realignment can help achieve those business goals.
3. **Monitoring capabilities.** Rolling out microservices and containers isn't enough – monitoring is essential to ensuring success, and it never stops. You'll have to decide whether to build an internal management system or coordinate third-party vendor options. Either way, you'll want to be able to monitor code inside containers and microservices within your APM (application performance management) footprint.

One major issue with microservices is that traditional logging techniques and tools are ineffective because microservices are stateless, distributed, and independent. When establishing monitoring solutions, you'll need to implement new ways to log your microservices in a way that minimizes everything except high-level, highly unusual alerts. Otherwise, your developers will be overwhelmed with the related noise.

An APM solution may be just what you need to ensure meaningful monitoring. Best practices when considering an APM solution? Consider your goals, both short- and long-term, develop ways to train and share knowledge, define a measurement framework, prioritize automation, and, as always, secure your data.

Still, the largest change may be how microservices and containers effect your IT organizational structure. Teams of developers and operational IT folks alike will need the responsibility and the authority to act in the best way as quickly as possible, so the company can quickly respond to the market without disrupting or slowing business activities.