DZone

DZONE'S 2019 GUIDE TO

Containers

ORCHESTRATION AND BEYOND

BROUGHT TO YOU IN PARTICIPATION WITH

ASPEN MESH

Couchbase

DIAMANTI

influxdata

NUODB

WhiteSource

Dear Reader,

Containers have come a very long way in the past couple of decades. Believe it or not, the concept of containers first emerged back in 1979 when the chroot system call was introduced to isolate processes and segregate files. It wasn't until 2000 that we saw the next major development — FreeBSD jails — and from there, container maturity has sped up drastically.

From the introduction of Docker to LXC to Kubernetes, the world of containers has been constantly evolving — and it is not going to slow down any time soon. This burgeoning interest, of course, has given way to myriad new developments, engineering hurdles, and security flaws.

Container security in particular has become important to a greater extent as container adoption has increased. The Docker Hub security breach of April 2019 revealed the usernames and passwords of 190,000 accounts using the repository to store database container images. The Dirty COW vulnerability was an 11-year-old bug that was reportedly resolved but got committed in October 2016 and resulted in attackers being able to possibly acquire remote root access to a computer. A runC vulnerability discovered in February 2019 would allow an attacker to gain root-level code execution.

As the adoption of containers and their counterparts increases drastically, engineers and enterprises have had to increasingly put work into ensuring not only container security but also general functionality and maintenance.

This means more concerted efforts toward the scalability, interoperability, and optimization of containers. As you'll learn in our key research findings, container adoption rates rose from 83% in 2018 to 89% in 2019, and container adoption rates within operation teams saw a huge increase from 38% to 60%. We've seen similar increases in terms of container usage throughout all stages of the SDLC as well as in the number of containers being run by organizations.

In addition to more key research on the growth of containers, DZone's third Guide to Containers dives into enhancing deployment containers on Kubernetes, explores how to use containers with serverless, and looks into major container orchestration trends. We'll also take a look at distributed microservices with containers, Kubernetes security, and key players in the realm of containers.

Thanks for reading and we hope you enjoy!

WRITTEN BY SARAH DAVIS
CONTENT MARKETING SPECIALIST AT ROI REVOLUTION

Table of Contents

DZone is...

Executive Summary

BY KARA PHELPS CONTENT & COMMUNITY MANAGER, DEVADA

Software developers use containers for almost everything. They've continued to find new ways to use containers ever since Docker began to build a wave of excitement around the technology in 2013. From Docker to Kubernetes to container-specific operating systems, container-related tools have become more or less indispensable to software development. We surveyed 464 tech professionals about their positive and negative experiences with containers, their expectations for the technology, and their choice of tools.

MUCH MORE WIDESPREAD THIS YEAR, ESPECIALLY IN QA/TESTING

DATA

When asked whether their organizations currently use container technologies, 70 percent of survey respondents said yes, a jump up from 45 percent last year. Among those who replied yes to this question, 60 percent said their organizations use container technologies in the QA/Testing department, increasing from 46 percent last year. 79 percent of these respondents also said their organizations use container technologies in their QA/testing environments, increasing from 63 percent last year.

IMPLICATIONS

Developers have shown their organizations the value of containers in their work. Containerized application platforms are designed to improve speed, reliability, and performance over virtual machines. As containers grow more widespread, that argument becomes easier and easier to prove with data.

Container technology is particularly crucial for quality assurance and testing. As teams work toward continuous testing and continuous deployment, they need to stay nimble as their workflow becomes more non-linear. Testers use containers to run multiple tests simultaneously, for example, as well as to reduce test times and improve consistency.

RECOMMENDATIONS

Container technology makes modern DevOps possible. The dramatic increase in adoption from 2018 to 2019 marks an important shift — a new majority of those surveyed are now using containers. Containers are no longer new, and their adoption has become a matter of keeping up with the competition.

GROWING ADOPTION, GROWING CHALLENGES

DATA

When asked what challenges containers present to their organization, 63 percent of survey respondents currently using containers said a major concern was refactoring or re-architecting legacy applications. That's a significant increase from last year, when 50 percent of those using containers reported this as an issue. The next top concern this year is a lack of developer experience with containers, a challenge shared by 58 percent of respondents using containers. In 2018, that percentage was negligibly smaller, at 56 percent.

IMPLICATIONS

Refactoring and re-architecting legacy applications has vaulted over a lack of developer experience to become the most widespread challenge among organizations using containers. As the adoption of container technology expands rapidly, so does the need to refactor and re-architect organizations' legacy systems — although many organizations choose to build new applications rather than refactor old ones. At the same time, a lack of developer experience is unavoidable for organizations just starting the transition to containers.

RECOMMENDATIONS

The challenge of refactoring and re-architecting is certainly a daunting one. With a focus on building new applications, however, and a commitment to move toward continuous integration and continuous deployment, your organization's efforts can be re-channeled. The issue of insufficient developer experience can be solved by taking some time each day to make sure your team is educated. The best education may be to simply dive in.

DOCKER AND KUBERNETES STILL DOMINATE

DATA

When asked which container technologies their organizations use, 94 percent of survey respondents using containers replied that they use Docker, followed (distantly) by LXC with just 4 percent. Docker use continues to increase — it's up from 91 percent last year.

When asked which container orchestration/management technologies their organizations use, 70 percent of respondents using containers said that they use Kubernetes, with Docker Swarm and Amazon ECS trailing at 33 percent and 31 percent, respectively. Kubernetes use is up substantially from 53 percent in 2018.

IMPLICATIONS

Docker and Kubernetes are closing in on total market domination of their own niches. Docker spearheaded the gradual move away from virtual machines to containers. The growth of Kubernetes (now one of the largest open source communities in the world) may be related to its ability to handle any cloud environment, as well as several well-publicized use cases. For example, Kubernetes helped the mobile app Pokemon Go scale extremely quickly when its popularity exploded over the course of a few days.

RECOMMENDATIONS

If you haven't yet learned your way around these technologies, now is the time. Docker and Kubernetes will grant you the most flexibility in terms of choosing a career path, considering the number of organizations that depend on both. There's an especially high probably that your future company will be using Docker. In terms of container orchestration, Docker Swarm and Amazon ECS are reasonable alternatives for the enterprise. Still, Kubernetes is the way to go if you're looking to upskill and teach yourself a tool on your own time.

Key Research Findings

BY JORDAN BAKER
CONTENT COORDINATOR, DEVADA

Demographics

For the 2019 DZone Guide to Containers, we surveyed 559 developers, with a 79% completion rating. Below, we've given a quick demographic breakdown of these respondents.

- Respondents live in three main geographical areas:
  - 35% reside in Europe
  - 26% live in the USA
  - 14% call South Central Asia home
- Respondents, on average, have 18 years of experience in the software/IT industry.
- Most respondents work for enterprise-level organizations:
  - 23% work for organizations with 100-999 employees
  - 22% work for organizations with 1,000-9,999 employees
  - 22% work for organizations with 10,000+ employees
- Respondents reported three main job roles:
  - 29% work as developers/engineers
  - 25% are architects
  - 20% make their living as developer team leads
- Survey-takers reported working on three main types of software development projects:
  - 84% are currently developing web applications/services
  - 52% are working on developing enterprise business applications
  - 30% are developing native mobile apps

How Orgs Use Containers

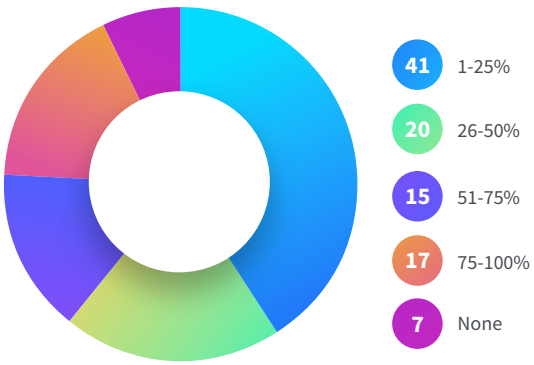
To set a baseline for the rest of the report, let's begin by examining the organizational use of containers, and how it has changed over time. When we asked respondents if their organization currently uses container technologies, 70% answered yes. This is a rather astounding number, as last year, in our 2018 Containers Guide Survey, only 45% of respondents reported that their organization used containers. Additionally, that percentage of respondents who told us that their organizations are not using containers fell from 25% in 2018 to 10% in this year's survey.

Taking a more granular look at these statistics, we find that every department within a typical software company's organizational structure witnessed large increases in container adoption. Among development teams, container adoption rates rose from 83% in 2018 to 89% in 2019; container use among DevOps teams grew from 77% to 82%; QA/Testing teams' adoption of containers went from 46% to 60%; and operations teams' container adoption rates climbed from 38% to 60%. These statistics represent a dramatic shift in the ways in which containers and container technologies are used throughout the SDLC.

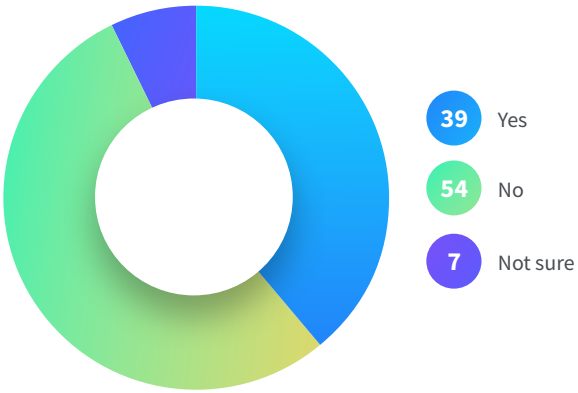
If we zoom in even farther, going from the departmental level to the environmental level, we see the same pattern occurring. In 2018, 87% of respondents reported using containers in development environments; this year, this number rose to 90%. Production/deployment environments saw a 7% percent increase

SURVEY RESPONSES

What percentage of your organization's workload is containerized?



Do you believe your organization has achieved Continuous Delivery?



in container use over the past year, with 80% of this year's survey-takers using containers in production/deployment. The containerization rates of QA/Testing environments jumped from 63% in 2018 to 79% in 2019; and staging environments witnessed a similar growth in containerization rates, going from 54% in 2018 to 71% this year.

All this is to say that container adoption has increased dramatically over the past year. This growth was not only reflected in the number of organizations, teams, and projects adopting container technologies but also the number of containers being run by those organizations. In last year's survey, 61% of respondents reported that their organizations ran 1-100 containers in production; this year, 46% of respondents reported thusly. Interestingly, the growth in larger containerized environments in production was spread out rather evenly among the different sizes. As this type of statistical breakdown does not lend itself to compelling prose, here are the numbers reflecting the year-over-year changes our respondents reported in terms of the number of containers their organization uses in production:

- 2018:
  - 1-100: 61%
  - 101-250: 13%
  - 251-500: 6%
  - 501-1,000: 2%
  - 1,001-5,000: 3%
  - 5,000+: 1%
- 2019:
  - 1-100: 46%
  - 101-250: 16%
  - 251-500: 9%
  - 501-1,000: 6%
  - 1,001-5,000: 4%
  - 5,000+: 4%

Containers as a DevOps Tool

Not much has changed over the past year in terms of the tools that developers use in their containerization efforts; the field is still dominated by Docker (and its other products like Docker Swarm, Docker Enterprise, and Docker Hub) and Kubernetes. Thus, rather than belabor the point of Docker's and Kubernetes's astronomical adoption rates, in this section, we'll evaluate the role that containers play as a means to achieve a more streamlined development process.

MICROSERVICES ADOPTION

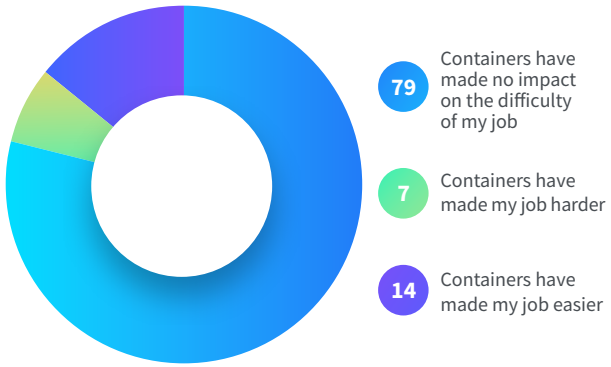
As noted in DZone's Guide to DevOps: Implementing Cultural Change, microservices have become an important facet of DevOps-based development — so much, in fact, that 58% of respondents to our 2019 DevOps survey reported using microservices in some capacity (either in production or development). In this year's containers survey, we also found a widespread adoption of microservices among respondents. When we asked survey-takers if their organizations have adopted microservices, 70% responded yes. This was a rather large year-over-year change from our 2018 Containers Guide survey, in which 57% of the respondents reported that their organization had adopted microservices. When we compare this year's data on microservices adoption to those organizations that are currently using containers, we find that the two are a popular pairing. Among those respondents who work for organizations that use container technology, 60% have adopted microservices. Additionally, only 10% of survey-takers who reported using microservices have not adopted container technologies.

CI/CD

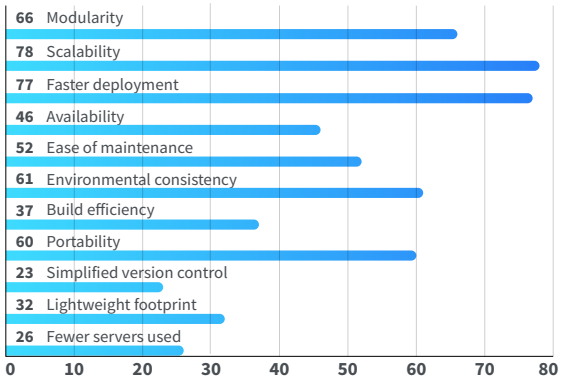
Continuous integration and continuous delivery constitute possibly the most well-known DevOps concepts, and thus seems like a good barometer with which to measure containers' effects on the DevOps process. When we asked survey takers whether

SURVEY RESPONSES

Have containers made your job easier or harder?



What benefits do container technologies offer your organization?



they believe their organization has achieved continuous delivery, 54% said no and 39% responded yes. Interestingly, when we correspond this data to the data on organizational container adoption, we get a different story. Among those respondents whose organizations are using container technologies, 33% reported that their organization has achieved continuous delivery and another 33% reported that their organization has not. Thus, while containers are not the silver bullet for achieving CD, it seems they may alleviate some of the roadblocks along the way.

In terms of continuous integration, we see a similar story. 62% of respondents said that they believe their organization has achieved CI, while 32% told us they have not. Comparing these numbers to our data on organizations that currently use container technologies, we find that the rate of organizations both achieving and not achieving continuous integration fall. Among respondents who work for organizations that use containers, 50% believe their organization has achieved CI, while 16% say they have not achieved CI. Thus, much like in the case of continuous delivery, containers seem to act more as an impediment remover to the CI process than an active agent in CI's success.

EXPECTATIONS VS. REALITY

Containers have received a fair amount of hype over the past several years. In an attempt to peer through the buzz, in this section, we'll be using the data we collected from developers on what they expected from containers and what they've actually found working with containers to be like.

We began by simply asking respondents what benefits they were expecting container technologies to offer their organization. The top answers were as follows: scalability (72%), faster deployment (69%), ease of maintenance (54%), environmental consistency (51%), modularity (48%), and portability (42%). Out of all these potential benefits, ease of maintenance (52%) was the only one where expected and actual benefits (more or less) matched up.

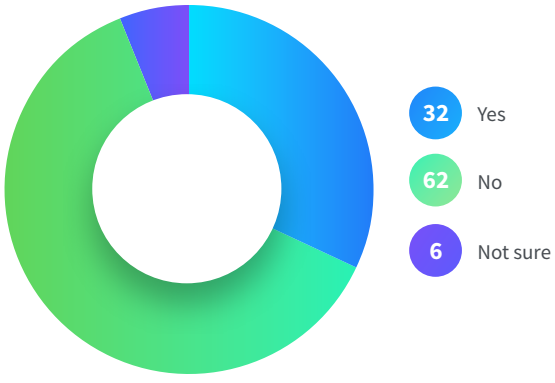
When we asked respondents what benefits containers actually brought to their organization, all the other factors enumerated above exceeded expectations (and, in some cases, by quite a wide margin). The benefits of containers that survey-takers reported actually witnessing in their organization were as follows: scalability (77%), faster deployment (77%), modularity (66%), environmental consistency (61%), and portability (60%). Modularity and portability seem to be the most unforeseen benefits offered by containers, with each garnering an 18% difference between expected and actual benefits.

When it comes to the expected versus actual challenges of containers, we saw equally interesting results. Unlike the potential benefits listed above, respondents only reported four main concerns regarding container adoption: lack of developer experience (80%, up from 71% in 2018), refactoring/re-architecting legacy applications (70%), ensuring application and network security (43%), and application performance monitoring (31%, down from 38% in 2018). Again, much like we saw with the benefits of containers, only one of these challenges actually matched respondents' expectations, with ensuring application and network security reported as a real challenge faced by 46% of respondents. The other three potential challenges proved less daunting than feared. 63% of respondents told us refactoring/ re-architecting legacy applications was an obstacle, 58% reported challenges with a lack of developer experience, and 41% said application performance monitoring proved a challenge (up from 33% in 2018). Out of these four main road blocks, however, only two were reported as a primary reason why organizations opted against container adoption: lack of developer (41%) experience and refactoring/re-architecting legacy applications (21%).

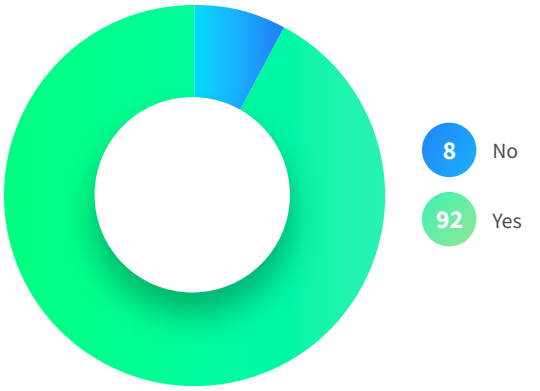
All in all, the benefits of container adoption seem to outweigh the negatives, as 79% of respondents told us they feel containers have made their jobs easier (up from 75% in 2018).

SURVEY RESPONSES

Do you believe your organization has achieved Continuous Integration?



Is your organization planning to increase the numer of containers they use?



Diving Deeper Into Containers

Twitter

 @kelseyhightower

 @lstoll

 @LibbyMClark

 @MayaKaczorowski

 @brendandburns

 @BrandonPhilips

 @sKriemhild

 @clare_liguori

 @jessfraz

 @abbyfuller

Books

The Docker Book: Containerization Is the New Virtualization
Dive into how to install, deploy, manage, and extend Docker, as well as how to use Docker to build containers and services for your specific needs.

Microservices and Containers
Get an analysis of how Docker containers plus microservices can aid in agile and scalable app development and deployment.

Kubernetes Cookbook: Building Cloud-Native Applications
Learn about successfully using Kubernetes to automate the deployment, scaling, and management of containerized applications.

Zones

Cloud dzone.com/cloud
The Cloud Zone covers the host of providers and utilities that make cloud computing possible and push the limits (and savings) with which we can deploy, store, and host applications in a flexible, elastic manner. The Cloud Zone focuses on PaaS, infrastructures, security, scalability, and hosting servers

Microservices dzone.com/microservices
The Microservices Zone will take you through breaking down the monolith step-by-step and designing microservices architecture from scratch. It covers everything from scalability to patterns and anti-patterns. It digs deeper than just containers to give you practical applications and business use cases.

DevOps dzone.com/devops
DevOps is a cultural movement, supported by exciting new tools, that is aimed at encouraging close cooperation within cross-disciplinary teams of developers and IT operations. The DevOps Zone is your hot spot for news and resources about Continuous Delivery, Puppet, Chef, Jenkins, and much more.

Refcardz

CI/CD for Containers
Containers and orchestration tools have often been cited as ways to facilitate continuous delivery and continuous integration. Download this Refcard to learn about the challenges and solutions to utilizing containers in your DevOps pipeline.

Java Containerization
This Refcard focuses on the design, deployment, service discovery, and management of Java applications on the open-source project called Docker so that you can get your Java application up and running inside a Docker-deployed Linux container.

Persistent Container Storage
Containers are great for building applications with ephemeral data. But what if you need your data to persist? Download this Refcard to learn what you need for container storage, discover the benefits of cloud-native storage.

Podcasts

Containerization and Why Everyone Loves Docker So Much
Learn about what containerization is, containerization software solutions, what the appeal of containers is, and how containerization is used.

Tom's Tech Notes: What You Need to Know About Containers
Get advice from nine industry experts on what you need to know about containers for modern application development.

Pod as a Service: Containerization & Container Orchestration
Learn about prominent container technologies and their benefits, considerations, orchestration, and management.

Easily Scale Your NoSQL Database Autonomously on Kubernetes With Couchbase

Flexible microservices are essential to today's enterprise architecture because they enable applications to evolve faster and more easily. However, their flexibility is greatly reduced when database clusters must be managed in silos because of a lack of standards among cloud providers.

Leverage Cloud Portability Across Platforms and Providers

The Couchbase Autonomous Operator for Kubernetes increases flexibility and decreases complexity and cost for the Couchbase NoSQL database in cloud environments. By enabling cloud portability and automating operational best practices, Couchbase makes it easy to take advantage of critical Kubernetes features like centralized management, persistent storage, auto-scaling, and auto-recovery.

Key Features of Couchbase

1. **Full-featured Database**
   Native integration with Kubernetes provides a comprehensive NoSQL database that supports critical applications with unparalleled performance. The Operator manages all the data services you expect from a NoSQL database, including rich SQL-based query and analytics, data synchronization with mobile, and full-text search.

2. **Deploy at Will**
   Couchbase doesn't force you to choose between on-premises, private cloud, or a specific public cloud deployment. You can easily deploy Couchbase within a managed private or public cloud to maximize flexibility, customizability, and performance.

3. **Use What You Know**
   Couchbase has developed strategic partnerships with the most popular enterprise providers, including Red Hat OpenShift Container Platform, and IBM Cloud Kubernetes Service. As cloud vendors build more ways to integrate with their container platforms, we make it easier to take advantage of their latest advancements.

WRITTEN BY ANIL KUMAR
DIRECTOR OF PRODUCT MANAGEMENT AT COUCHBASE

PARTNER SPOTLIGHT

Couchbase NoSQL Database

The only NoSQL database for business-critical applications

Category NoSQL Database

New Release Annual

Open Source? Yes

Case Study

Couchbase's mission is to provide the database that revolutionizes digital innovation. To make this possible, Couchbase created the world's first database specifically designed to help deliver ever-richer and ever-more-personalized customer and employee experiences. Built with the most powerful NoSQL technology, Couchbase was architected on top of an open-source foundation for the massively interactive enterprise. Our geo-distributed database provides unmatched developer agility and manageability, as well as unparalleled performance at any scale, from any cloud to the edge.

Couchbase has become pervasive in our everyday lives; our customers include industry leaders Amadeus, AT&T, BD (Becton, Dickinson and Company), Carrefour, Cisco, Comcast, Disney, DreamWorks Animation, eBay, Marriott, Neiman Marcus, Tesco, Tommy Hilfiger, United, Verizon, Wells Fargo, as well as hundreds of other household names. For more information, visit couchbase.com.

Strengths

- **Built for change at scale** – Support millions of interactions and easily respond to ever-changing business requirements
- **Memory-first speed** – Rich data access, in-memory replication, 99.999% availability
- **Security across the stack** – Encrypt, audit, protect, and secure your data anywhere
- **Cloud-native, available anywhere** – Cross datacenter replication (XDCR) in any cloud
- **Fully synced and offline-first** – Real-time data sync even when users are offline

Notable Customers

- AT&T
- PayPal
- eBay
- Tesco
- United

Website
couchbase.com

Twitter
@couchbase

Blog
blog.couchbase.com

The Essential Challenge of Kubernetes Security

QUICK VIEW

01. Security Kubernetes is maturing as the technology becomes more commonplace in enterprise IT.

02. Effective Kubenernetes Security is about what happens inside and outside of the cluster.

03. Tools and techniques are only as good as the policies and procedures they support.

BY BOB RESELMAN
INDEPENDENT DEVELOPER AND INDUSTRY ANALYST

Kubernetes is fast becoming **the** way to design and deploy applications intended to run at web scale. It's a big piece of software, filled with hundreds of details that need to be understood and accommodated. High on the list of details is security. However, unlike traditional distributed applications of the past, in which code and environment are somewhat static, the promise of Kubernetes is that it can support ephemeral systems. In Kubernetes, applications can be revised on demand and scaled up or down to meet the need of the moment. Thus, any approach to Kubernetes security must accommodate both the Kubernetes cluster and its host environment in a dynamic, comprehensive manner.

This is not an easy undertaking. Fortunately, creating a secure Kubernetes computing environment is quite achievable once an essential challenge is met. This challenge is to put tools, policies, and procedures in place that continuously monitor and secure both the Kubernetes cluster and the host environment in which it runs.

Allow me to elaborate.

Kubernetes Really is Very Cool

The beauty of Kubernetes is that it takes care of all the details that go with deploying, accessing, and upgrading the containers that make up a componentized distributed application. First, you submit declarative manifest files to Kubernetes, which indicate the repository where relevant container images are stored. In addition, there are manifest files that describe how to configure the services that will use those containers as well as the security infrastructure necessary to access the services and pods in which the containers reside. Once Kubernetes receives the manifests, it creates the required pods, including its containers. These pods are created in one or a mix of several virtual or physical machines that make up the nodes in a given Kubernetes cluster. Then, Kubernetes creates the services and security mechanisms that are defined in the relevant manifest file. Once the pods and services are up and running in the cluster, Kubernetes ensures that state of that deployment for as long as the application runs. Should a pod malfunction, Kubernetes will automatically replenish it. If a node fails, Kubernetes will replenish the failing node's pods in another node. All in all, it's pretty cool.

Security Inside and Outside the Cluster

Kubernetes takes a lot of human labor out of administering a multi-node distributed application. But, as powerful as Kubernetes's automation is, it creates problems in terms of security. These problems can be segmented into two types, those inside the cluster and those outside the cluster.

One example of an inside-the-cluster problem is the recent security vulnerability found in RunC, which is the container runtime used in many Kubernetes clusters. The security vulnerability made it possible for a malicious container to overwrite RunC in order to get root-level access to the host. Once RunC has root level access, it can wreak havoc on the hosting machine. The result can be catastrophic.

Another inside-the-cluster problem is based on the very nature of container architecture. A container is realized at runtime using a container image. Just about all Kubernetes applications use container images downloaded from a public or private repository on the internet. While most container images are safe to use, the fact is you can never really be sure. Container architecture makes it possible for one container image to reference another container image, which in turn can reference yet another container image, ad

infinitum in an image chain that can be very long. Trusting that all images in a chain are secure is a big assumption. Yes, there are tools out there that will help safeguard things. But, the important thing to understand is that the risk is there, and bad things have happened due to a malicious container making its way into a cluster.

Outside-the-cluster problems take many forms. Of course, there are the security problems that go with any computer, virtual or real, that is connected to the internet, such as DDoS attacks, injected malware, or out-and-out invasion. However, a security issue that particular to Kubernetes has to do with the way pods (and the containers they host) are assigned to a node.

Let me explain.

Remember, unless precautions are taken otherwise, Kubernetes decides where to create a pod. Imagine that you have a cluster that's made up of two hundred nodes (virtual machines). These days no human is going to hand configure two hundred VMs. Instead, it's done automatically.

A script will create the two hundred VMs, which may be in one data center or in 15 in a variety of locations around the planet. The VMs are made part of the Kubernetes cluster, either by humans or by automation, as is typical when spinning up a Kubernetes cluster in a native cloud environment, such as Google Cloud.

After the cluster is created, either a sysadmin or script will run the Kubernetes manifests required to make the particular distributed application. Then, the application comes online. No big deal, right? It happens every day. Well, not so fast.

Let's say that the application of interest is a financial application subject to a regulation that requires the application to always run on hardware in the USA. Now, imagine that one of the machines in a data center in the USA fails. As described above, Kubernetes will replenish the pods that were running on the failed node elsewhere. Thus, Kubernetes function as designed: it will replenish the pods on another VM in the cluster. However, in this case, Kubernetes replenishes the pods on a virtual machine that is running in a data center in Ireland! The application is now is in violation of a regulatory requirement and the application's security is compromised. While this violation is not an overt, malicious assault, it is nonetheless a problem, potentially a serious one, particularly if the company that owns the financial application is subject to fines.

Take note, in the example described above, there was nothing wrong with the internal workings of the cluster. The issue at hand was outside the cluster. While automation makes large scale VM provision and Kubernetes deployment possible, it also obscures situations that are or might become hazardous.

So, what's to be done?

Security is Policy and Practice

Kubernetes has moved well beyond being an experiment in container management. What started out as an internal project at Google is now a mainstay technology for many companies large and small, and more are joining the fold every day. As a result, Kubernetes security is now a first-class concern. As a result, there's a lot of activity taking place in the Kubernetes Security space.

The Center for Internet Security keeps releasing updates to the Kubernetes Benchmarks, which is an exhaustive 253 page book, covering over 100 points of concern. The Benchmark describes exactly how to configure, deploy, and maintain a secure Kubernetes Cluster.

Also, the number of tools available to ensure a secure Kubernetes environment inside and outside the cluster keeps growing. These tools provide the continuous scanning and proactive prevention features that are necessary to ensure that everything is safe and secure, inside and outside the cluster.

For example, Docker Hub, the go-to repository for container image storage offers security scanning for its enterprise customers in its Trust Registry. Another example is Anchore, an open source tool you can use to scan OS packages, container images, and Dockerfile configuration. (For those you who want more information about container security solutions, Rancher Labs provides a useful list of tools that focus on inside and outside the cluster.)

Tools are necessary, no doubt. But, as history has proven more than once, all the tools in the world won't matter unless a company has policies and procedures in place that foster effective security practices and, most importantly, are easy to follow.

These kinds of policies and procedures lend themselves well to automation that increases consistency and effectiveness. (Automation never sleeps, humans do.) For example, DivvyCloud, a platform the specializes in hybrid and multi-cloud management, has automated policy definition and procedural support for the CIS Kubernetes Benchmarks. The platform has the ability to ensure that pods don't go astray. In addition, there are open-source tools such as, Kube-Bench and Kube-Hunter that put automation at the forefront.

But, regardless of whether your company takes an old school approach, manually inspecting and approving every cluster in force, or goes with state-of-the-art automation techniques, the important thing is to implement continuous, comprehensive security measures that focus inside and outside the cluster into the fabric of the software development process. Creating and maintaining a secure Kubernetes infrastructure must be as second nature to an IT Department as creating and maintaining a sterile operating room is to a hospital. What happens outside the patient is just as important as what happens on the inside. Whether it's an operating room in a hospital or the Kubernetes cluster in your company's data center, there's a lot on the line. Meeting the challenge of keeping things safe and secure inside and outside the cluster is essential for any company using Kubernetes in mission-critical, production environments.

BOB RESELMAN is a nationally-known software developer, system architect and technical writer/journalist. Bob has written four books on computer programming and dozens of articles about topics related to software development technologies and techniques, as well as the culture of software development. Bob lives in Los Angeles. In addition to his work on in a variety of aspects of software development and DevOps, Bob is working on a book about the impact of automation on human employment. LinkedIn Twitter

SERVICE MESH SIMPLIFIED

The secure way for enterprises to manage microservices

FOR DEVELOPERS

Developers can focus on business logic instead of managing infrastructure.

FOR OPERATORS

Platform Operators gain a highly secure and observable way to manage policy and configuration.

FOR BUSINESS

The Business can decouple deployment from service activation to gain control without slowing down.

ASPEN MESH

aspenmesh.io

TRY THE BETA

Securing Containerized Applications With Service Mesh

The self-contained, ephemeral nature of microservices comes with some serious upside, but keeping track of every single one is a challenge, especially when trying to figure out how the rest are affected when a single microservice goes down. If you're operating or developing containerized applications, there's a good chance that part of your days are spent wondering what your services are up to.

Problems like security, load balancing, monitoring, and rate limiting that had to be solved once for a monolith, now have to be handled separately for each service. The technology aimed at addressing these microservice challenges has been rapidly evolving:

1. Containers facilitate the shift from monolith to microservices by enabling independence between applications and infrastructure.
2. Container orchestration tools solve microservices build and deploy issues but leave many unsolved runtime challenges.
3. Service mesh addresses runtime issues including observability, traffic management, and security.

Improving Container Security with Service Mesh

A service mesh provides an advanced toolbox that lets users add observability, resiliency, and security to containers. One of the most valuable applications of a service mesh is bolstering cluster security. There are three distinct capabilities provided by the mesh that enable a more secure architecture.

TRAFFIC ENCRYPTION

Service mesh allows operators to leverage mTLS to encrypt traffic between services. The mesh automatically encrypts and decrypts requests and responses, removing the burden from application developers. A service mesh helps platform operators understand and enforce how services are communicating and prove it cryptographically.

SECURITY AT THE EDGE

A service mesh adds a layer of security at the perimeter Kubernetes clusters so any compromising traffic can be addressed as it enters the mesh. Service mesh route rules help manage compromising traffic at the edge. Egress capabilities allow you to dictate that network traffic does not go places it shouldn't.

ROLE BASED ACCESS CONTROL (RBAC)

In distributed organizations, an app should only have the minimum amount of permissions and privilege it needs to get its job done. A service mesh enables fine-grained RBAC so developers can continue to develop quickly within the security and compliance standards set by the platform.

Sidecar is a Great Place for Security

Microservices are an opportunity to improve your security posture but also present challenges around consistency. The best organizations manage this with the principle of least privilege. That's easier to apply when a small, single-purpose microservice has clear and narrowly-scoped API contracts. But there's a challenge as application count increases that this principle can be unevenly applied. Microservices, when managed properly, can increase feature velocity and enable security teams to fulfill their charter without becoming the Department of No.

A sidecar service mesh provides a great way to balance the need to move quickly with the need for an architecturally sound security posture. Sidecar-based service meshes like Istio put their datapath functionality into a separate container and then situate that container as close as possible to the application they are protecting. A sidecar service mesh provides the opportunity to get security right once in the sidecar, and then distribute the sidecar everywhere, and get back to adding business value instead of duplicating security efforts for every service. Check out this blog for more details on how a sidecar service mesh adds mTLS, perimeter security and fine-grained RBAC to Kubernetes clusters.

WRITTEN BY ANDREW JENKINS
CTO AT ASPEN MESH

WRITTEN BY ZACH JORY
HEAD OF MARKETING AT ASPEN MESH

Containers and Serverless: Powerful Abstractions in Exchange for Developer Velocity

QUICK VIEW

BY SIMONA COTIN
SENIOR CLOUD DEVELOPER ADVOCATE

Much like high-level programming languages are an abstraction of machine code, serverless is an abstraction for cloud infrastructure. And knowing when to choose high-level abstraction over low-level implementation (and vice versa) can be quite a challenge sometimes. In this article, we'll put serverless and containers side-by-side and work through the advantages and challenges for each. We'll also explore some of the most common use cases for both technologies and leave you with some tips that will help you choose more easily.

What Are Containers?

Containers are a form of virtualization that is more lightweight than the usual virtual machine. The basis of their implementation is LXC (LinuX Containers), which is a form of kernel-level virtualization available starting with the Linux Kernel v2.26.4. It is using a feature called `cgroups` that allows for better isolation levels and memory partitioning. In the current ecosystem, containers are the most basic element of building a cluster of services for your modern micro-service-based apps. There are multiple implementations of LXC; however, the brand that has become synonymous with containers is Docker, and we will focus on Docker containers because they are most often used in the wild.

Although Docker implements LXC concepts, it also extends the functionality and makes the containers and deployments more portable. Docker containers are application-centric; allow for automatic builds, Git-style versioning and operations, and incremental build sharing; and provide a rich tooling ecosystem for managing your containers. The tools available allow you to manage and orchestrate your services in multiple containers with `docker-compose`, and in more complex scenarios, you can use `kubernetes` for even more fine-grained control over the way your infrastructure behaves in scenarios where your services need to scale.

What Is Serverless?

You might already be using some serverless components. When your application relies on managed services like S3, Azure Storage, or Azure CosmosDB, you're already benefiting from using a serverless architecture. Fully managed and highly scalable services are core tenets of any serverless system. They clear the path for us to focus on features that are truly relevant to our business by removing the need for us to learn how to install, configure, and host them.

At the core of serverless computing are cloud functions. They enable us to run code in ephemeral containers in reaction to an event. The execution can be triggered by any managed services integrated with the platform or some custom sources you might define.

In serverless functions, there's a couple of things you need to be aware of. You'll end up writing code mostly in the same way you did before, using the programming languages that you normally use. But because serverless is abstracting away the server management for you, it also means that you don't have access to any of the low-level OS APIs. To

communicate with some of them, you'll need to use a platform API. Remember, in serverless, you share a physical machine with other users, so the platform needs to ensure perfect isolation for your environment. Because your code is running in ephemeral containers, and to allow your applications to scale out infinitely, you'll need to write stateless code. This means that you cannot rely on any state being preserved between function calls. If you do end up having to save state, then you'll have to use a data store like a message queue or a database.

As documented in the Serverless Working Group whitepaper, the anatomy of a serverless processing model is:

- **Event sources or triggers** cause the function to run.
- **Serverless controllers** deploy, control, and monitor function instances and their sources.
- **Function instances** are single functions/microservices that can be scaled with demand.
- **Managed services** include data stores, authentication providers, and events.

With serverless, the platform will automatically scale by dynamically adding and removing resources based on the number of incoming events. A scale controller will use heuristics for each event type. For example, when you're listening to queue events, it scales based on the queue length and the age of the oldest queue message. When no events are triggering your code to run, the scale controller will reduce the number of instances running to zero. Scalability is a hard problem to solve. By outsourcing the job of monitoring and spinning up new instances, we get to focus on understanding how components in our system communicate and optimize for that.

Infrastructure Abstraction

The goal of having containers as the building blocks for your infrastructure is to replace VMs. The advantages of using containers over VMs help you when prototyping by allowing you to quickly boot up your machines as opposed to the heavyweight VM. The resulting Docker containers can be easily redistributed as base images or packaged intermediate layers. You can build a library with these images and use them as required to construct your microservice-based application using a cookie-cutter approach for your service containers. Using orchestration tools like `docker-compose` and `kubernetes`, you can build and configure your microservices cluster locally along with the network configuration that allows you to specify the way your microservices communicate. This allows you to replicate your cloud architecture locally and version the building blocks. This helps with tracking down dependencies, facilitates debugging, and enables you to create a CD pipeline for complex microservice-based architectures. In a sense, it is very similar to having your own cloud available on your localhost. The orchestration tools allow you to have containers that fulfill the functionality of both the managed containers and serverless functions that you have available in your cloud provider of choice.

As your application architecture grows in complexity, so will the challenge of managing it using the Docker tools. Starting from the containers, as your microservices evolve, the images can start to become bulkier and may require trimming of the unnecessary cruft. Otherwise, it is likely to slow down your CI/CD pipeline speed. So, a periodic optimization of your Docker images could be a good practice. This would ensure that images are as lightweight as they need to be to keep your image storage requirements under control. Besides the containers that are part of the app, other challenges arise from the fact that the architecture is essentially that of a distributed system and as such, it requires mechanisms that enable communication between the services. Luckily, there are prebuilt Docker images for message queues like `kafka`, `rabbitmq`, and prepackaged database images for managing your cluster's persistent state. In normal conditions, the bulk of your services should be quite well-behaved. But more often than not, some parts of the system can run into unexpected errors. This is when having a centralized logging solution is very useful for diagnosing the issue. Again, there are various pre-built images for enterprise-grade centralized logging, like `logstash` and `fluentd`. These are valuable for tracking down issues that are causing failures; however, in more specific cases, you can run into performance issues like suspicious memory or CPU spikes. In these cases, a more specific type of tool can be used to troubleshoot.

Serverless is the latest step on the path of taking away the burden of infrastructure. It frees you up from spending time on planning, configuring, and managing servers. With serverless technologies, what we lose in controlling infrastructure we gain in developer velocity and fast, incremental iterations.

Both containers and serverless are key technologies in our cloud-native adoption journey. Containers will allow for full flexibility in migrating your existing workloads without needing to change a single line of code. Serverless, while requiring refactoring, will allow for the autoscaling of services that have spiky traffic and reduction of cost. We've walked through the fundamentals and given you the foundation you need to decide if, when, and how to use containers and serverless in your projects. To learn more, see "Create Serverless Applications" and "Deploy and Run a Containerized Web App with Azure App Service" for step-by-step interactive training and sandbox environments.

SIMONA COTIN is a web developer with a passion for teaching. She spends most of her time tinkering with JavaScript in the cloud and sharing her experience with other developers at community events like meetups and conferences or online. As a Cloud Developer Advocate, Simona engages with the web community to help create a great developer experience with Azure. She loves shipping code to production and has built network data analytics platforms using Angular, Typescript, React, and Node.js. LinkedIn Twitter

Container Infrastructure in 15 Minutes

Diamanti is an enterprise-class platform that integrates Kubernetes, Docker, compute, network, and persistent storage in one easy-to-deploy solution. With Diamanti, you get:

SPEED

- 15-minute container infrastructure deployment
- 2,400,000+ IOPS
- 100µs latency across cluster

SIMPLICITY

- Easy to buy
- Easy to deploy
- Easy to manage
- Easy to scale

EFFICIENCY

- 50% less infrastructure
- 70% lower capex
- Integrates with modern workflows

CONTROL

- Developer-managed infrastructure
- Predictable performance
- Container-granular QOS

DIAMANTI

www.diamanti.com

redhat. TECHNOLOGY PARTNER

CRN 10 Coolest Cloud Storage Startups

Gartner Cool Vendor 2017

database TREND-SETTING PRODUCT IN DATA MANAGEMENT

CRN EMERGING VENDORS 2018

Why Containers Are an Existential Threat to VMware

VMware has dominated the virtualization space since it introduced the technology 19 years ago. But experts now cite the rise of containers as an existential risk to VMware. The Docker container format, popularized by developers, is taking the datacenter by storm as enterprises move containers into production.

But what's driving this transformation? Pressure to modernize applications and accelerate time-to-market is greater than ever. That means more frequent releases, straining the old ways of deploying software and putting the monolithic application stack under siege. In contrast, many of today's most advanced distributed systems are built using frameworks that are deployed as containers and designed to run on clusters of bare-metal servers — without virtual machines.

However, this runs contrary to VMware CEO Pat Gelsinger's recent keynote speech at VMworld. In it he stated that virtual machines are still the best place to run Kubernetes and containers.

What should we make of this? In short, it's self-serving and customers know better. VMware's acquisition of Heptio for $550 million suggests VMware is eager to build IP around containers, and to figure out how to reduce the risk of virtual machine obsolescence.

Here's why VMware should be on the defensive: VMs require excessive overhead, because inside each VM you have to boot up an OS. So when you run containers on a VM, you're running a VM and an OS and a container. Running each container on a VM also introduces two layers of orchestration — one to manage the virtualization environment, and Kubernetes to manage the container environment. That means two layers of networking topology to architect, and two layers of storage to manage.

Many cloud-native apps are not meant to run on VMs, they're meant to run on bare metal. Most are I/O bound when it comes to performance. And with VMs, I/O performance — and more importantly TPS (transactions per second) — suffers.

If containers replace virtual machines and the enterprise datacenter standardizes on Kubernetes, what will the net impact be on the vendor ecosystem? We're on the cusp of a revolution much like VM revolution two decades ago. But at the time, no one knew how to operate VMs. No one knew how to optimize storage or manage networks so VMs would work efficiently. It was all new, and it was all far from optimal.

If containers replace virtual machines and the enterprise datacenter standardizes on Kubernetes, what will the net impact be on the vendor ecosystem?

Today, the first attempts at containerized infrastructure from the hyperconverged players have largely been about trying to squeeze containers into an obsolete virtualization model. But next-generation infrastructure vendors like Diamanti are focused on deploying Kubernetes and containers on bare metal without VMs — lessons learned from the most sophisticated container production environments in the world.

WRITTEN BY TOM BARTON
CEO AT DIAMANTI

Why Should Developers Care About a Service Mesh?

BY NEERAJ PODDAR
ENGINEERING LEAD AT ASPEN MESH

QUICK VIEW

01. Developers are becoming a key consideration and driver of the business strategy in a software-driven world.

02. Developers are tasked with helping the business find more efficient tools and ways of working.

03. One of the key technologies that can help developers enable organizations to gain a competitive advantage through software delivery is microservices.

04. Managing the infrastructure for microservice architectures can be a burden to developers, but service mesh removes infrastructure concerns from them, so they can spend more time on feature delivery that drives value for the business.

Developers are the New Kingmakers of the modern business world. The meteoric rise in developer status is largely due to the emergence of SaaS platforms and self-service sales motions. Today's products built by developers are sold directly to end users via platforms (also built by developers), thereby reducing the need to maintain a large and expensive salesforce. This change creates opportunities for businesses to reallocate funds to development teams that are the new drivers for growth. Modern businesses require that developers spend most (if not all) of their time adding business value to the organization.

A major innovation driver across industries is the creation of new applications built on microservices. Microservices solve many of the resiliency and scaling challenges of modern applications. However, as microservice architectures grow, organizations often struggle to manage the complexity involved with securing (e.g. mutual TLS) and understanding the communication (e.g. tracing, telemetry) between services. The natural inclination for solving these problems is to embed more complicated logic in the microservice itself, which increases complexity and also takes away precious developer time from adding business value. Additionally, the operations team in charge of the overall security and uptime of the solution now has to rely on the developers to enforce security policies or add observability, which creates a need for the development and security teams to become more aligned.

These are the problems that a service mesh is designed to solve. Service mesh is an infrastructure layer for managing communication between microservices in an application-agnostic way. It allows developers to move various communication, security, and visibility-related functionalities from application code to the infrastructure configuration managed by the operations team. In doing so, it decouples the two teams, reducing the burden on developers and shifting the entirety of infrastructure concerns to the operator. Developers should care about a service mesh as it reduces the set of things they have to maintain and build. So, let's explore the service mesh architecture to understand how it provides these advanced capabilities.

The Advantages of Service Mesh for Developers

Service meshes like Istio work by adding a proxy in the communication path, which can intercept all traffic (HTTP 1.1, 2.0, gRPC requests, etc.) going in and out of microservices. As the proxy observes the traffic it can take various actions like retries, timeouts, adding mutual TLS, and exporting telemetry on behalf of the application. There are various architectural options depending on where and how you deploy these proxies, each with benefits and drawbacks. Istio uses a sidecar deployment model where the proxies (Envoy) are deployed as a separate container which can be managed and controlled outside the application container ensuring the decoupling of operations and development teams. As the sidecar proxy augments the traffic flowing through your microservices, it is important for developers to know how to offload functionality to the proxy and update their application. Let's examine some of the capabilities provided by Istio and how developers can leverage these functionalities in their environments.

ADVANCED TELEMETRY

Gathering time series telemetry data is one of the best ways to understand how your application is performing and for getting alerts when user experience or performance degrades. Istio provides a rich set of metrics out-of-the-box, which can be easily used to infer the health and performance of your microservices. The sidecar proxy collects various request

and connection-level statistics and combines them with a user configuration, such as a Kubernetes service and deployment, to add context to the telemetry without changing the application code. This is beneficial for the operations team, as they can create advanced SLO dashboards and alerts in a consistent way without relying on application code. At the same time, developers are no longer required to generate service-level metrics if they have Istio in their environment and can focus on adding implementation-specific metrics which can aid in debugging and diagnosing application-level problems.

DISTRIBUTED TRACING

In a microservices architecture, it is often difficult to understand why a request was made from one microservice to another, as one incoming request from the end user can generate many internal requests between microservices. The volume of requests and the lack of context can make it difficult to pinpoint and debug failures in the system. Distributed tracing solves this problem by adding special HTTP tracing headers on every request which are propagated across microservices.

These headers contain the hierarchical context, which can be correlated to understand the dependencies between microservices. In Istio, the sidecar proxy when intercepting the traffic automatically adds these headers, creates the appropriate context (root/child spans) and sends the data asynchronously to the tracing backend system. This is great as the developers don't have to write additional logic to export this information, and the metadata associated with the spans is guaranteed to be consistent across microservices. However, to get the full view of the microservice dependencies you will need to propagate tracing headers from incoming requests to outgoing requests as described here in your applications. It is important to update your application to leverage the full benefit of distributed tracing in Istio.

REQUEST RETRIES

Failures in microservices are common, and sometimes retrying the request gets a successful response. Traditionally, retry logic has been embedded in the application code, which over time gets more complicated as routing logic is added to handle various scenarios. Istio allows you to configure advanced retry settings for a microservice, which can be customized for various paths and failures types. This enables operators to easily control this behavior independent of the application code.

MUTUAL TLS

Securing microservices is an ever-increasing challenge due to the distributed deployment model, complex communication patterns, and the difficulty in establishing trust among peers. To solve this issue, developers often end up implementing complicated security protocols like mutual TLS in their application, which makes the microservice code harder to maintain. This makes it difficult for operations to enforce uniformity across microservices and easily apply critical security patches, as it might require updating all applications. Istio provides mutual TLS out-of-the-box, which ensures all traffic between microservices is always encrypted and the identity of the peers are mutually verified without any modification of the application itself. The traffic between the application and the sidecar proxy continues to be in plaintext, the sidecar encrypts traffic before sending it to the destination, where the sidecar proxy on that side decrypts the traffic before forwarding it to the application. Additionally, the Istio control plane handles certificate rotations automatically, relieving some manual work from operations. This means if you're using Istio in your environment you can achieve your security goals without updating the applications!

SERVICE DECOMPOSITION

Most microservices often (ironically) start as macro services, and developers break them apart over time when a natural separation emerges either in APIs or policy requirements for data access. While you can always break them apart in code, service meshes like Istio enable a way to decompose these services without code changes, test the multiple services, and then split your code and build system to produce several smaller microservices. Istio also provides advanced pathbased routing, which can be leveraged to route traffic to different instances of your unified service based on API request paths. Using Istio telemetry you can verify the performance and behavior of API-separated instances before actually splitting them apart in code. This enables a more phased approach to decomposing your services into smaller, well-defined microservices.

Apart from the above-mentioned benefits, there are a few additional factors that developers should take into account before adding a service mesh like Istio in their environments. Adding a sidecar or any form of proxy outside your application introduces latency, which might not be acceptable for various latency-sensitive applications. In Istio, as the sidecar proxy is added on both the client and server, even a small amount of latency persidecar can add up if you have hard restrictions on the latency requirements for your service. Another tricky issue which I have often seen is that your application requires customized load balancing (i.e. some sort of stateful communication) or has a clustering protocol, and adding a sidecar (which by default handles load balancing and service discovery) breaks the existing behavior. There are ways to work around this and still get other benefits by adding a service mesh but understanding this caveat and configuring it correctly can ensure a smoother transition. In order to successfully adopt a service mesh, it is important for developers to have these conversations with their operations team so that they can minimize surprises and maximize benefits.

I hope this article simplifies the service mesh landscape for developers, informs them on why they should care about it, and how to easily embrace it to drive business value for their organization.=

NEERAJ PODDAR is the Engineering Lead at Aspen Mesh. He has worked on various aspects of operating systems, networking, and distributed systems over the span of his career. He is passionate about developing efficient and performant distributed applications. At Aspen Mesh, he is currently leading the efforts to build an enterprise service mesh and their hosted SaaS platform. In his free time, you can find him playing racquetball and gaining back the spent calories by trying out new restaurants.
LinkedIn

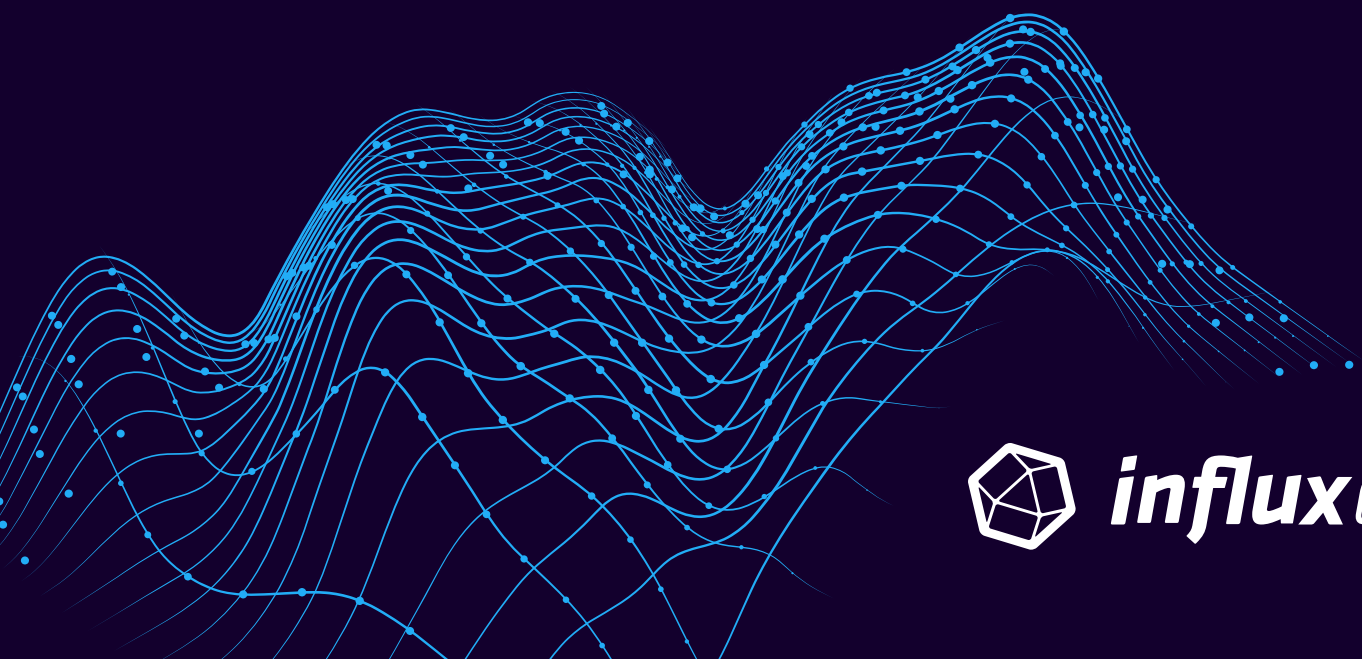
May 2019

Why Time Series matters for metrics, real-time, and sensor data

DOWNLOAD THE E-BOOK

"MySQL is not intended for time series data… I can testify it is like pounding nails with a screwdriver. It's definitely not what you want to do in any relational database."

John Burk, Senior Software Developer



influxdata®

Monitoring in the Era of Containers

Containers are a game-changer for everyone. Another abstraction between the infrastructure and the application layers, containers are a group play and concern IT System Ops, NetOps, and DevOps. However, professionals in different roles approach container monitoring from different perspectives, all valid, important, and necessary to build a complete monitoring strategy.

With the fragmentation of applications into containerized microservices running in clusters, each container packages the necessary resources to execute its part of the deal. However, unless its workload, counterparts, and the network putting the pieces together are equally performant, the sum of the parts will fall short from delivering a satisfactory whole.

New paradigm requires new perspectives

Container monitoring from the infrastructure perspective went a long way with platforms like Kubernetes, where an orchestration logical layer was added to the mix to commoditize infrastructure layer on which microservices run, while automating deployment and optimizing resource assignment. However, a misbehaving or underperforming microservice, even if running on a compliant declared state cluster, can still create a dismantling effect on network and overall application performance.

Therefore, unless the health of the microservices and their inter-container activities in a proliferous meshed network is also kept close to heart, containerized application monitoring will suffer from short-sightedness. It is clear that monitoring only from the outside will not fit the bill.

InfluxData and its partner ntop are taking the next step in monitoring containerized application environments with extended Berkeley Packet Filter (eBPF): shedding some more light from the inside to guide IT to find out where things are broken or breaking, and who is causing it.

Bringing all together

eBPF opens one more channel to reach deeper in the inner observation required to make the link between anomalous behavior and performance variations to what is causing them in a distributed containerized environment.

However, not only all data, but all eyes and hands should come to the same place and reduce the burden from setting up, learning ramp up, managing, and gathering information pieces from multiple monitoring sources. The complexity of the container era demands one platform for all types of data, one integrated data source, and one UI for all visualizations. Bringing these all together will compound insights and perspectives, leading to a monitoring solution that enables more intelligent alerts and actionable information.

InfluxData is the leading time series platform and comes out of the box ready to use. Learn more.

WRITTEN BY DANIELLA PONTES
SR. PRODUCT MARKETING MANAGER AT INFLUXDATA

PARTNER SPOTLIGHT

InfluxData

Act in Time

Category Time Series Data Platform

New Release Quarterly release cycles

Open Source? Yes

Why Use InfluxData to Gain More Visibility Into Network Monitoring

Learn how to use InfluxData for your network monitoring to gain the necessary visibility in the status, performance and responsiveness of your enterprise, cloud or hybrid application environments. Get a faster and easier tool to start collecting data from multiple sources and quickly perform root-cause analysis reducing your MTTR.

Strengths

- Developer happiness
- Faster Time to Awesome
- Ease of scaleout and deployment
- Centralized and integrated full stack monitoring

Notable Customers

- Coupa
- PayPal
- Wayfair

Website
influxdata.com

Twitter
@InfluxDB

Blog
influxdata.com/blog

Case Studies for Deploying Containers in Your Organization

QUICK VIEW

01. Get an in-depth use case looking at Cherre's continuous integration (CI) and continuous delivery (CD) pipelines.

02. Explore a discussion of the combination of technologies used within the pipeline and supporting arguments for their usage.

03. Take a look at the benefits that Cherre enjoys by iterating this pipeline repeatedly.

BY STEFAN THORPE
HEAD OF DEVOPS AND SECURITY AT CHERRE AND CTO AT CAYLENT

At Cherre, we use a combination of Helm Charts and Helmfile to enhance container deployments on Kubernetes according to development best practices. This methodology, in union with high internal DevOps standards, allows our team to:

- Deploy containers at scale
- Use the same continuous integration and delivery pipeline for every application
- Keep secrets and passwords within app code (through Helm Secrets and Kubernetes' Key Management System)
- Roll out complete versioned replicas of all our apps for backup and during disaster recovery

This article will outline and examine the benefits of this methodology through examples — beginning by highlighting the combination of technologies used, then examining how we link them, and, finally, underscoring the advantageous results we realize through implementing these processes to make our pipeline more efficient.

The continuous integration (CI) and continuous delivery (CD) pipeline we've generated at Cherre is designed so that it can be replicated over and over. Process iteration through the technologies and tools we employ means that we can deploy containers at scale using a uniform method without reinventing the wheel every time. The automated process, once designed and set up, requires little in the way of modifications each time, even if alternate services are employed. The tools we employ in our container deployment pipeline are Helm Charts, future simple/helm-secrets, roboll/helmfile, and CircleCI. The final process template we have created is a collection of templates that will then work for any application.

Helm

Helm is the base technology of the process which helps us render and lint template files using configurations from values.yaml. Then it's about simply applying the rendered Kuberentes manifest together with corresponding metadata. Helm is the `apt-get`/`yum` of Kubernetes. Created by the folks from Deis, it's a package manager for deploying applications to Kubernetes. As well as making it super easy to version, package, and release deployments, Helm does so in a manner that allows users to deploy, delete, upgrade, and even rollback those deployments as flexible Helm Charts — Charts being the terminology that Helm uses to describe a package of configured Kubernetes resources.

Within the Cherre deployment pipeline, Helm is leveraged specifically for its Charts feature. Building a Chart starts with creating a skeleton structure chart which establishes a directory. Within this directory are the three files we are most interested in: Chart.yaml, values.yaml, and NOTES.txt.

- The Chart.yaml file outlines the chart's purpose through its name, description, and version.
- Values.yaml describes the variables necessary for the template files directory. For complex deployments that sit outside the default templates capability, it's just a case of editing the Go template files in this directory.
- Finally, NOTES.txt provides information post-deployment to whoever deployed the chart. For example, it could outline how to use the chart, or deliver default settings, etc.

For multiple, reiterated deployments, it's just a case of rewriting the chart involved as well as writing a helmfile. The Helm aspect of our pipeline pretty much follows the tool's standard best practices as outlined here. We don't do anything particularly special otherwise. Here is our app example chart.

Helm secrets allow us to combine the simplicity of Google Cloud's KMS system and the straightforward functionality offered by Helm deployments.

HELM-SECRETS

When faced with secrets management in Kubernetes, there aren't a lot of good built-in options, so we turn to helm-secrets for leveraging secret keys. What we've traditionally seen is that developers create Kubernetes secrets manually. This is because encrypting and versioning configuration within code has previously been difficult, time-consuming, and, at times, unachievable. Helm secrets allows us to combine the simplicity of Google Cloud's KMS system and the straightforward functionality offered by Helm deployments. To make this even easier, we organize our values and secrets file into named production-type environments. See the outlined configuration example below from the GitHub repo for this case study.

- values.yaml
- secrets.yaml
- sandbox/values.yaml
- sandbox/secrets.yaml
- production/values.yaml
- production/secrets.yaml

One common mistake we see here, though, is that people often try to duplicate an entire values file. This does not conform to DRY best practices (Don't Repeat Yourself), and leads to an inability to test and safely deploy into different environments. That's why we enforce the principle of only adding a value to an environment — such as production — if it overrides a default set in the pre-configured default values file. What this means is that when you are reviewing a secret or an environmental secret file, you're only seeing default overrides. This makes it clearer to view any differentials for that environment.

Secondly, KMS also means that we can use our cloud Identity Access Management (IAM) to configure encryption — and, more specifically, decryption.

HELMFILE

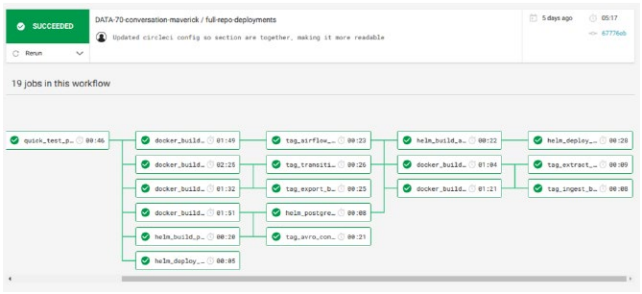
Another integral part of the Cherre pipeline is the tool helmfile, which encompasses secrets and helm differently. The helmfile.yaml is a declarative configuration file that makes it simpler to deploy and manage a large number of different Helm Charts. The GitHub repo page provides many examples of how best to utilize the tool for different Charts. The benefits of optimizing this tool are:

- Multi ad-hoc dev branch deployments from the developer's local machine
- Unique sandbox branch deployments for full integrated testing.
- Staging deployments
- Automated production deployments

In order to achieve a standard pipeline and execution with CI and CD, we standardized our Helm file. An example of that can be found here.

CircleCI

CircleCI is renowned as the second largest continuous integration system in use at GitHub. All our workflows at Cherre run through the tool. We leverage .yaml anchors to template out the main flows, i.e. `docker_build` to `docker_deploy` to `helm_chart_build` and `helm_chart_deploy`. It is possible to use CircleCI Orbs (which optimizes pre-written scripts), but we had already created templates before its release and have no need to revisit them just yet. CircleCI triggers a helmfile command which in turn launches the Helm Chart deployment. The flow of the pipeline in its entirety looks like the following image:



Within the workflow, as you can see, we do quick unit tests followed by Docker image builds. The images are then retagged before being pushed to the correct location in the cloud. Finally, we implement our Helm Chart builds, followed by the Helm Chart deployment.

The pipeline can then be adjusted according to necessary dependencies. For example, with `helm_build` for PostgreSQL, there is no Docker dependency, as it's part of another service. This means it doesn't have to wait until a particular point in the process, so we can deploy that early on. GraphQL is another example of an external Helm Chart, so once again, we don't build it out — we just deploy it early.

An advantage of establishing a CI/CD pipeline as outlined above is the ability to release a new version alongside the old version, then switch traffic on the application. By adopting the blue/green deployment procedure here, we are able to roll out complete versioned replicas of all our apps for backup. After ensuring that the new version meets requirements through testing, we update the Kubernetes Service object that is playing the role of the load balancer to reroute traffic to the new version by replacing the version label in the selector field. We can also roll back to an older — safer — version for disaster recovery quickly and with minimal risk when necessary.

Wrapped around the whole process is our Cherre library, which describes additional wrappers according to each function. These functions help to outline and set up additional environments and variables as required. Building a source library of template Helm Charts only improves the efficiency and productivity of the process. Furthermore, it becomes increasingly easier to save time within the launch process by being able to refer to previously used configurations.

> CI and CD improve the velocity, productivity, and sustainability of software development teams and help organizations respond to market changes better.

Good CI and CD is not about a set of tools or a single process, though. They are ongoing development practices that involve following the key principles and ensuring team responsibility for building quality into every application that you deploy. CI and CD improve the velocity, productivity, and sustainability of software development teams and help organizations like us — of all sizes — respond to market changes better.

STEFAN THORPE Head of DevOps and Security at Cherre, CTO at Caylent, Cloud Solutions Architect DevOps Evangelist. Stefan is an IT professional with 20+ years management and hands-on experience providing technical and DevOps solutions to support strategic business objectives. LinkedIn Twitter

Are Containers Still Relevant in 2019?

QUICK VIEW

01. Containers continue to be very popular among developers — even with those who have yet to use containers.

02. For orchestration, Kubernetes shows continued dominance overall whereas Docker Swarm seems to be the winner for smaller companies.

03. Serverless proves to be very popular with developers who have used it, but greater exposure will likely influence trends towards the technology.

BY LOU BICHARD
JAVASCRIPT FULL STACK ENGINEER AT DAZN

That's the question we're answering today.

The cloud market continues to expand. AWS (the dominant cloud solution) has viable competition in big players like Google with GCP and Microsoft with Azure.

The market for packaging and deploying compute services is no different. Container-based tools such as Docker have matured beyond "new technology" and are often considered a "safe bet."

But how are the market trends looking? Will containers continue to dominate? And do innovations in cloud offerings like AWS Fargate (the hostless container platform) make container orchestration tools like the successful Kubernetes redundant?

Today, we'll go through the data on how containers are currently used and what our predictions are for how we're likely to use them in the future.

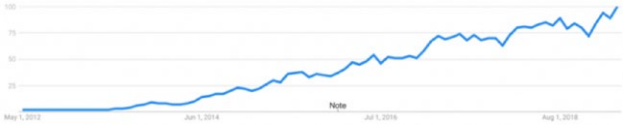
By the end of this article, you should know for sure which compute services you should be betting on in 2019 and beyond.

Container Trends: What the Data Tells Us

To better understand the trends with containers, let's dive into the data.

But what data, exactly? We're going to look at the high-level trends from Google to give us the broad picture. After that, we'll drill into what percentage of companies as a whole are using containers. And finally, we'll look at what technologies these container-adopting companies are running and what their languages of choice are.

Let's start with the best high-level indicator for market changes: Google Trends. Since the key term "container" can be ambiguous and skew our data, we can instead search data related to the biggest container platform, Docker, for a reasonable look at the data.



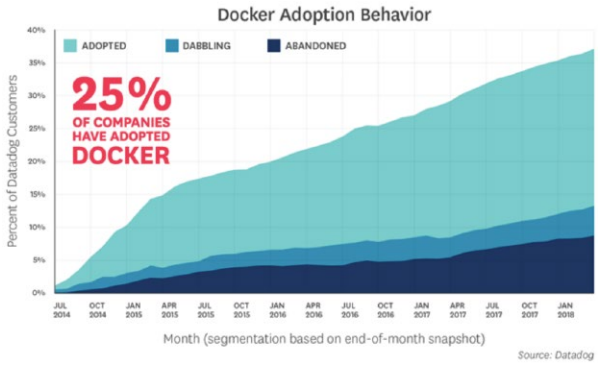
Google searches for "Docker" since 2012

As expected, search interest in containers has been growing since Docker's inception in 2014, with no signs of slowing into 2019 and no hallmark fad signs of rises and crashes in popularity.

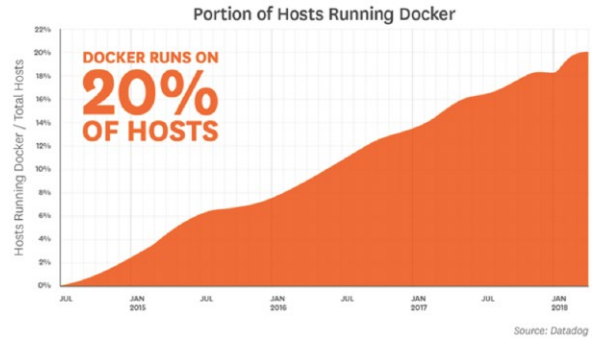
But Google only gives a basic indication. To get more understanding of how many technology companies are using containers, let's cast an eye over DataDog's report on the emerging trends in container orchestration from December 2018. DataDog is a leading APM platform, with a wide range of integrated companies using its product to monitor their technology. DataDog's report therefore gives us some interesting stats about how containers are currently used in the market. Let's take a peek.

Currently, 25% of the DataDog userbase is reported to be using Docker containers somewhere within their technology stack. Additionally, 20% of overall hosts that are monitored are also running Docker technologies. The data is also supported by research conducted by Digital Ocean that shows 49% of developers are now using contain-

ers — a staggering statistic. 78% of those who aren't using containers say they "plan to adopt them in the future." And lastly, the IDC's 2018 report on the rise ofenterprise container platforms found that 85% of container adopters are using containers for production apps.



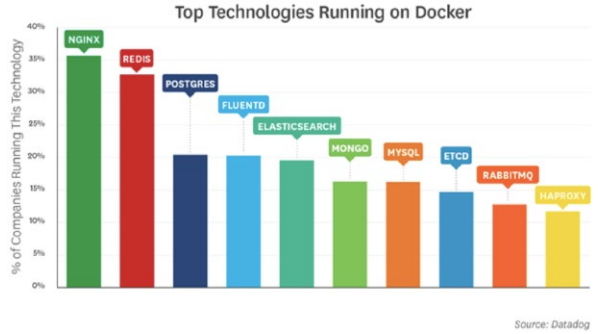
DataDog Docker adoption



Number of hosts running Docker (DataDog)

From the data, we can see that containerization still proves very popular, even in 2019. But the next natural question is: If we're using containers so much, what are we using them for? What types of technology are we running on them?

Nicely, DataDog reports on this data, too. The DataDog report shows NGINX, a load balancer/web server, as the most popular containerized technology, followed by Redis and Postgres. Interestingly, the top technologies for container technology are a mixed bag of web servers, databases, and messaging brokers. From the data, we can really see just how versatile containers are and the breadth of their usage.



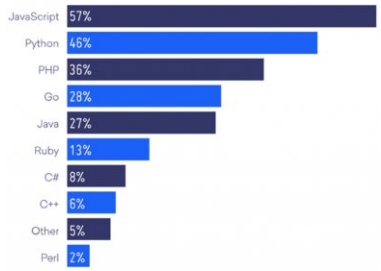
DataDog reveals half of Docker environments are orchestrated

From the above chart, we can see what out-of-the-box technologies are being run on containers. Yet, we know that not all companies are running these types of technologies, and some are running their own applications, written in their language of choice. So that leaves us with the question: What language is used with container technologies? And luckily, we have that data, too!

Digital Ocean's cloud trends report showed the most frequently used languages for those using containers. They report JavaScript to be topping the chart followed by Python, PHP, and then Go.

What the Digital Ocean data tells us is that lighter-weight scripting languages seem to be favorable alongside container usage. Which raises the question: Why?

One reason for the correlation (more on this topic later) could have to do with the increased adoption of microservices and microservice-type architecture. These architectures encourage small, simple, testable, and independently deployable software components that can be achieved with these lightweight scripting languages.

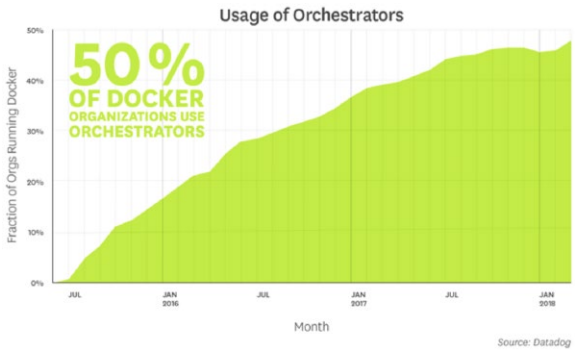


Language popularity according to Digital Ocean cloud trends report.

And that concludes our look at the container market. It seems on the face of it that Docker continues to be a very popular choice for packaging and deploying code in 2019. As you may already know, it's very common for container runners to use an orchestration tool to manage their running containers. And if containers are still going strong, what does that mean for their orchestration counterparts?

Container Orchestration Trends

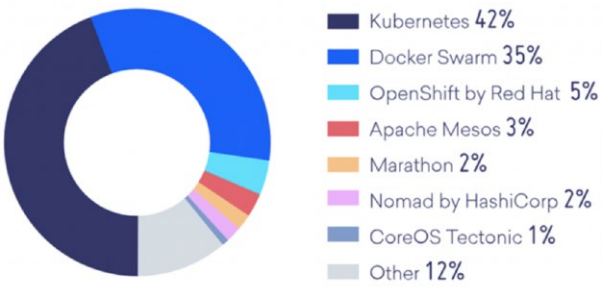
So, we know that container usage is strong, and growing. But what about their orchestrators? Well, DataDog's survey shows that half of the companies running Docker use some form of orchestration tool.



DataDog reveals half of Docker environments arerchestrated

Seeing over half of container users also using orchestration tooling is a lot. And I know what question you're going to ask now: What orchestration tool are they running?

The cloud trends survey by Digital Ocean reveals that nearly half of container users use Kubernetes as their container solution, with Docker Swarm coming in second and the competitors paling in significance. Interestingly, most Docker Swarm users were smaller companies, possibly due to the simpler nature of Docker Swarm setup.



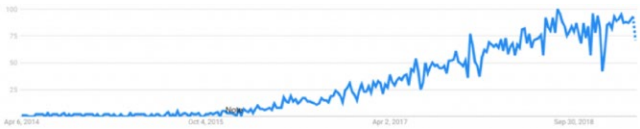
Popular orchestrators

We can clearly see that containers (in particular, Docker) prove to be very popular, and if you're looking for an orchestration tool, it seems like either Kubernetes or Docker Swarm are the key technologies to consider.

However, when discussing trends for containers, there is an elephant in the room: serverless. Serverless is a newer technology than containers, which allows compute resources to be provisioned in the cloud on an on-demand basis. While not *strictly* a direct competitor to containers, greater adoption in the serverless tech would (*ceteris paribus*) see a decline in numbers for container usage. Therefore, our research would be incomplete if we didn't also assess the trends surrounding serverless — so let's do that now.

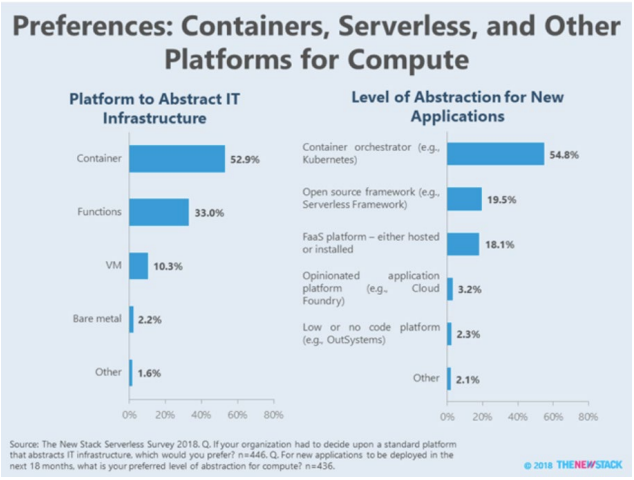
Will Serverless Kill Containers?

To be uniform with our review of containers, let's start by looking at the Google Trends. We can see that searches regarding serverless, like containers, have grown over time. Rather surprisingly, the current Google search volume for the keyword "serverless" matches the term "Docker."



Google searches for "serverless" in the past five years

But does that mean that developers are starting to use serverless as much as they are containers? Let's find out. A survey completed by *The New Stack* asked developers how they would prefer to standardize their infrastructure and 53% voted for containers, compared with 33% for serverless (with the rest of the respondents opting for regular VMs or even bare metal).



So, what does the data show us? One thing is for sure: It doesn't show containers as the outright winner, when compared with serverless for the standardization of infrastructure, which is interesting given containers' aforementioned adoption rates.

However, it should be noted that *The New Stack* found that most developers are still fairly naive in their understanding of serverless. This means that over time, as more developers are exposed to serverless, their preferences might lean toward the technology as their standardization platform of choice.

Are Containers Still a Good Bet?

That concludes our look at the container market for 2019.

As we go through 2019, I predict we're going to continue to see many more companies tending towards cloud-native. When fears are allayed over vendor lock-in, technologies like serverless will rise in adoption and popularity. However, since containers and serverless aren't direct competitors *per se*, technology choice will likely not be the differentiating factor for success. Rather, success will be determined by a company's curiosity to explore and embrace new technologies.

LOU BICHARD is a JavaScript full stack engineer with a passion for culture, approach, and delivery. He believes the best products emerge from high-performing teams and practices. Lou is a fan and advocate of old-school lean and systems thinking, XP, continuous delivery, and DevOps. LinkedIn

NuoDB | The SQL Database Designed for Distributed Deployments

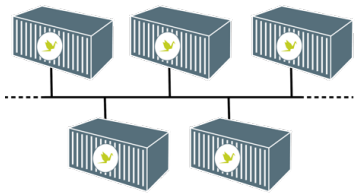
In 2010 we founded NuoDB to build a revolutionary database formed by two foundational principles:

1. Applications that run using Structured Query Language, or SQL, to store, manipulate, and retrieve data in databases will continue to do so because they require the data management criteria synonymous with SQL.
2. Cloud technology drives the need for organizations to be "distributed" at all layers of the technology stack, including the database.

With those two beliefs, we set out to build a SQL database designed for distributed deployment; a database that delivers on the promise of cloud innovation.

DATABASE MUST-HAVES	CUSTOMERS GAIN
• **Active-Active** across multiple geo configurations • **Container-native** to deliver the agility businesses and applications need • **In-memory transaction processing** boosts performance and scalability • **On-demand scale** that matches other cloud technologies • **SQL / ACID-compliance** that databases of record demand	• **Accuracy**: rely on the accuracy of your data • **Control**: manage database costs against usage needs • **Flexibility**: migrate from on-prem to ANY cloud • **Performance**: improve speed and availability • **Redundancy & Reliability**: eliminate disaster recovery woes

Why NuoDB?



NuoDB running in container

NuoDB empowers businesses to become agile and always on, to quickly respond to changing customer, competitive, and business demands by providing a container-native SQL database for enterprise critical transactional applications in hybrid cloud, distributed, or multi-data center environments. NuoDB's distributed SQL database offers the benefits of traditional databases, including ACID transactions, SQL support, and enterprise capabilities, plus the core strengths of NoSQL: it's always on, cloud-native, and provides on-demand scalability.

About NuoDB

NuoDB's distributed SQL database helps enterprise organizations overcome the complex challenges faced when trying to move enterprise-grade, transactional applications to the cloud.

For technical organizations adopting a distributed, cloud-first strategy through hybrid-cloud, container-native, microservices, and other modern architectures, NuoDB is the only solution offering flexibility without sacrificing performance, SQL dependency, and availability.

NuoDB is backed by three former CEOs of the four original relational database companies. Our senior management team includes former executives from such organizations as ExaGrid, Hewlett-Packard Enterprise, IONA Technologies, Iron Mountain, Microsoft, Object Design, Oracle, Teradata, and Veracode.

NuoDB is headquartered in Cambridge, MA, USA, with offices in Belfast, Bulgaria, Dublin, and London.

WRITTEN BY ARIFF KASSAM
VP OF PRODUCTS NUODB

Executive Insights on the State of Containers

QUICK VIEW

01. The most important elements of orchestrating and deploying containers are security and configuration.

02. Containers have accelerated application development and have made the ability to scale a non-issue.

03. Developers need to keep security, process engineering, and 12-factor application methodology in mind to work effectively with containers.

BY TOM SMITH
RESEARCH ANALYST AT DEVADA

To understand the current and future state of containers, we gathered insights from 33 IT executives who are actively using containers. Here's who we spoke to:

- Tim Curless, Solutions Principal, AHEAD
- Gadi Naor, CTO and Co-founder, Alcide
- Carmine Rimi, Product Manager, Canonical
- Sanjay Challa, Director of Product Management, Datical
- OJ Ngo, CTO, DH2i
- Shiv Ramji, V.P. Product, DigitalOcean
- Antony Edwards, COO Eggplant
- Anders Wallgren, CTO, Electric Cloud
- Armon Dadgar, Founder and CTO, HashiCorp
- Gaurav Yadav, Founding Engineer Product Manager, Hedvig
- Ben Bromhead, Chief Technology Officer, Instaclustr
- Jim Scott, Director, Enterprise Architecture, MapR
- Vesna Soraic, Senior Product Marketing Manager, ITOM, Micro Focus
- Fei Huang, CEO, NeuVector
- Ryan Duguid, Chief Evangelist, Nintex
- Ariff Kassam, VP of Products and Joe Leslie, Senior Product Manager, NuoDB
- Bich Le, Chief Architect, Platform9
- Anand Shah, Software Development Manager, Provenir
- Sheng Liang, Co-founder and CEO, and Shannon Williams, Co-founder, Rancher Labs
- Scott McCarty, Principal Product Manager - Containers, Red Hat
- Dave Blakey, CEO, Snapt
- Keith Kuchler, V.P. Engineering, SolarWinds
- Edmond Cullen, Practice Principal Architect, SPR
- Ali Golshan, CTO, StackRox
- Karthik Ramasamy, Co-Founder, Streamlio
- Loris Degioanni, CTO, Sysdig
- Todd Morneau, Director of Product Management, Threat Stack
- Rob Lalonde, VP and GM of Cloud, Univa
- Vincent Lussenburg, Director of DevOps Strategy; Andreas Prins, Vice President of Product Development; and Vincent Partington, Vice President Cloud Native Technology, XebiaLabs

And, here's what they told us:

Key Findings

1. The most important elements of orchestrating and deploying containers are security and configuration. Think about security from the beginning by analyzing the deployment files for your container and ensure they are optimized for best security practices, analyzing runtime behavior, and building segmentation policies. Follow security best practices with end-to-end control of risk and change. Have control over what ends up in different environments. Make sure the containers you are deploying are always scanned for vulnerabilities.

Have visibility of deployment to improve speed, deliverability, and scalability. This leads to improved isolation of failures and better development hygiene. The more deployable units you have, the more you have to keep track of. As such, centralize and automate configuration management. Develop, test, and deploy container-based applications quickly and seamlessly using automated CI/CD pipelines.

2. Containers have accelerated application development and made scalability a non-issue. Containers open DevOps teams to feature and product velocity, control, and security. There's greater access to testing and report-

ing. Resource isolation reduces mean-time-to-resolution (MTTR). Containers can be spun up and torn down in seconds. According to research conducted by Sysdig, 95% of containers live less than one week.

Scaling is no longer a concern for deployment, it's something runtime takes care of. The lack of environmental restrictions makes scaling easier as parameters change dynamically.

3. A variety of methods are used to secure containers during orchestration, deployment, and ongoing operation. Make sure Kubernetes (K8s) is configured correctly. Think through the lifecycle of containers and the container platform – especially how they are configured and how they meet the Federal Information Processing Standards (FIPS).

Automation is crucial – secure SDLC practices as well as CI/CD and fail code before build time. Force vulnerability scanning of container images. Use security benchmarks to facilitate profiling and reduce exposure.

Best practices are to protect environments across development, build, and runtime to ensure the full lifecycle is taken care of, including runtime post-breech detection. TLS is recommended for communication between containers using a vault to store secrets.

4. The most frequent use cases for containers are focused on scale, with multiple industries mentioned. This is a function of the ease of spinning up new environments in minutes and the ability to automatically scale up a workload or app in response to an increase in demand and additional load. Automatic scale-in and -out is used to adjust to application user demand and build-in continuous availability to meet consumer demands.

5. The most frequently mentioned failures revolve around a lack of skills and security. There is a lack of highly-skilled talent and production-level knowledge for running containers, as well as a lack of understanding of application factors, components, and architectural dependencies. People think getting into containers will be easy without realizing they need to plan for platform upgrades and the need to build automation to handle upgrades. When you start to use microservices, you need to think about how you are going to manage, deploy, and monitor. People need to understand containers are immutable components. Deep expertise in K8s, DevOps, and creating the tools needed to implement application workflows in a containerized environment can be hard to come by.

Be mindful of security best practices such as encrypting sensitive data. Security configuration for workloads needs to be hardened. Avoid using network isolation tools since these can leave huge security gaps that can lead to the exfiltration of data due to misconfiguration, application vulnerabilities, or even an open source library coming from a corrupted source. Be aware that security breaches align with potential application security vulnerabilities.

6. Concerns about containers revolve around security, complexity, and skills. Understand how to how to handle security and failover within containers. Neither containers, nor containerized orchestration, deliver an enterprise-grade secured workload environment. Have proper security visibility in place. Security measures put in place by newcomers will be inadequate given the rise in the number and sophistication of attacks. People are beginning to realize the broad vulnerability due to images on the public registry. More education, and learning, needs to take place.

Complexity is a tremendous concern. There continues to be a gap in ease of use. There are a lot of solutions to add on to a basic K8s cluster with varied degrees of supportability and incubation. It takes six months to get your head around what's going on. It can be an extremely difficult task to pick the right technology and tools to use. Additional complexity accrues when deciding on an upgrade strategy for the system and tool set.

There's a significant skills shortage. Companies are getting on the container bandwagon; however, their people do not have the skills to help them get the most out of them. There is a dearth of talent with hands-on experience. Companies are relying on consultancies, third-party vendors, or platform solutions.

7. The future of containers is greater maturity, less complexity, and serverless. Containers will disappear into the background like all other good technology. There is solid proof of value with regards to developer experience and velocity. Use of containers will go mainstream as companies move from talking about cloud and containers to using them in production. The technology will only become more stable, standardized, and portable as it matures. Containers will replace virtual machines and will grow well past the use of Docker containers as the primary container type. Other container products will grow in popularity and use as well.

Containers will be made less complicated making K8s easy for a developer to get up and running quickly. We're seeing a trend of abstractions built on top of K8s. Containers are also making it easy for everyone to go serverless. There's no need to rely on a machine with a VM, it's easier to go serverless. Serverless and FaaS are on the way, and service meshes like Istio will help to track myriad smaller components.

8. Developers need to keep security, process engineering, and the 12-factor application methodology in mind. Be cognizant of the security of containers from the beginning. Embrace security tools that perform continuous analysis on all activities within the container environment from a granular level. Understand the security and privacy implications of the data you are manipulating. The more developers can adopt frameworks and guardrails to help the security teams manage the security of containers.

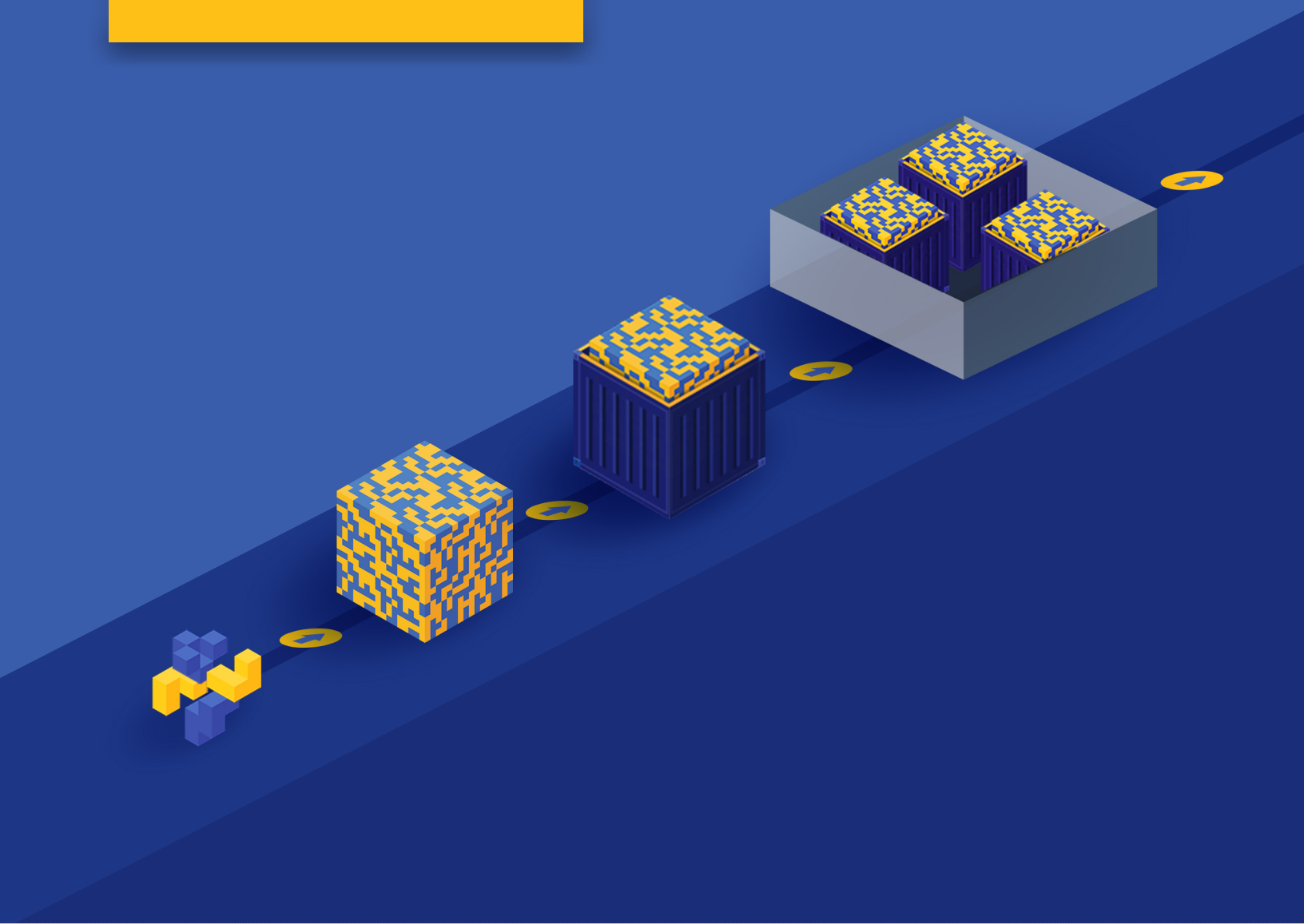
Follow process engineering best practices and look for "HelloWorld" tutorials for K8s, Docker, and Docker Swarm. Learn how to organize an application to optimize for containers and orchestration guidelines like the 12-factor application methodology. The entire container architecture needs to be 12-factor.

TOM SMITH is a Research Analyst at Devada who excels at gathering insights from analytics—both quantitative and qualitative—to drive business results. His passion is sharing information of value to help people succeed. In his spare time, you can find him either eating at Chipotle or working out at the gym. LinkedIn - Twitter

WhiteSource

Secure Your Open Source Components in Your Containers

FIND OUT HOW



Want to try us out? **Sign up** for a free trial here and start securing your Containers.

Secure the Open Source in Your Containers From Start to Finish

Containers rely heavily on open source components, comprising the vast majority of code in most modern applications today. While providing a powerful building block for our software, these open source components can be used by attackers to exploit your product if they are not protected.

As developers play a greater role in vulnerability management for their applications, we have a responsibility to make sure the open source components in our containers are secure and compliant.

The challenge that many developers face in working securely is achieving sufficient visibility into the multiple layers of their containers, lacking the control to guard against risky components. It is hard to stay protected if you can't see what you're working with.

Responding to the need to ensure security coverage, WhiteSource for Containers offers teams a holistic solution for the entire container development lifecycle from building your container registries and even all the way to Kubernetes with real-time alerts and policy enforcement so developers can focus on building great products.

Providing support for over 200 programming languages, WhiteSource for Containers offers native integration to all environments along the way, including all CI servers, container registries, and container orchestration tools like Kubernetes.

View all the relevant information pertaining to containers with easy to navigate dashboards to produce container vulnerability reports, manage policies, and track the security status of deployed containers.

Containers are a powerful tool that help teams reach deployment faster. However, in order to reap their benefits, you need to take control of your vulnerability management. WhiteSource for Containers allows developers to code with confidence without compromising on security.

WRITTEN BY RAMI SASS
CEO & CO-FOUNDER, WHITESOURCE

PARTNER SPOTLIGHT

WhiteSource for Containers

WhiteSource for Containers provides continuous and automated open source security throughout the container lifecycle.

Category Container Security | Application Security | Software Composition Analysis | DevOps | Secure Coding

New Release Continuous

Open Source? Yes

Case Study

Due to the sensitive regulatory environment Siemens Healthineers works in, Siemens H. needed assurance that they are remaining compliant and secure in their use of open source components.

"With open source software, usually the source code is available for all to see, including hackers," explained Code Clinic Lead, Neil Langmead. In order to avoid costly mistakes that can result from vulnerable or risky open source components being added to their products, Siemens Healthineers turned to WhiteSource.

"We chose WhiteSource because of its ease of use, its excellent data, and for the in-depth security vulnerability information that comes with the reporting engine." With WhiteSource, Siemens was offered the widest coverage of plugins and languages that they needed, as well as the continuous monitoring and policy enforcement safeguards they required in order to allow their team to code with confidence. Read more here

Strengths

- **Automated workflow**: Enforce policies automatically at all stages of the container lifecycle
- **Native integrations to container registries**: Track vulnerabilities in images automatically
- **Kubernetes deployment**: Automatically scans any image deployed to production
- **Comprehensive coverage**: Supports 200+ programming languages and many Linux distributions
- **Pinpoint accuracy**: Proprietary algorithms guarantee no false positives

Notable Customers

- Microsoft
- Comcast
- EllieMae
- IGT
- Spotify

Website
whitesourcesoftware.com

Twitter
@WhiteSourceSof t

Blog
resources.whitesourcesoftware.com

Containers Solutions Directory

This directory of container platforms, orchestration tools, cloud platforms, and registries provides comprehensive, factual comparisons of data gathered from third-party sources and the tool creators' organizations. Solutions in the directory are selected based on several impartial criteria, including solution maturity, technical innovativeness, relevance, and data availability.

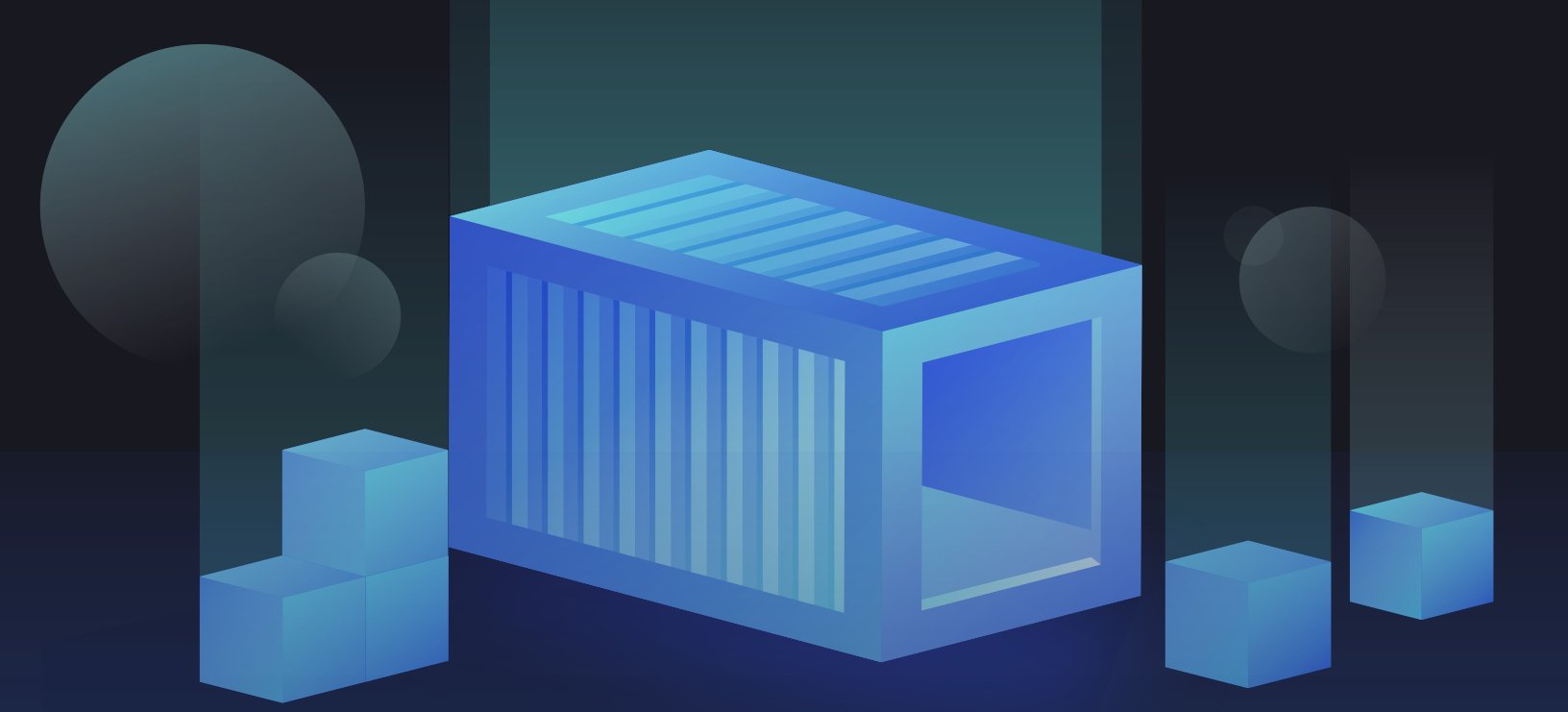
Company	Product	Product Type	Open Source?	Website
Amazon	EC2 Container Registry	Container image registry	Free tier available	aws.amazon.com/ecr
Amazon	EC2 Container Service	Containers-as-a-Service	Free tier available	aws.amazon.com/ecs
Anchore	Anchore	Container image security & compliance	Open-source version available	anchore.com
Apache Software Foundation	Mesos	Cluster mgmt software	Open source	mesos.apache.org
Aqua	Aqua Container Security Platform	Container image security	Demo available by request	aquasec.com/products/aqua-container-security-platform
Aspen Mesh	Aspen Mesh	Enterprise service mesh	Open source	github.com/aspenmesh
Atos SE	Apprenda Cloud Platform	PaaS, container platform, Kubernetes-as-a-Service	Demo available by request	apprenda.com/platform
Bitnami	Stacksmith	Containers-as-a-Service	Open-source version available	stacksmith.bitnami.com
CA Technologies	Automic Continuous Service	Service orchestration	30 days	ca.com/us/products/ca-automic-service-orchestration.html
CA Technologies	CA Application Performance Management	Container monitoring	Available by request	ca.com/us/products/docker-monitoring.html
Canonical	Ubuntu Core	Container OS	Open source	developer.ubuntu.com/core

Company	Product	Product Type	Open Source?	Website
Chef	Chef Automate	Continuous deployment platform	Demo available by request	chef.io/products/automate
Cisco	Contiv	Container-defined networking	Open source	contiv.io
Cloudera	Cloudera Enterprise 6	Analytics, search, data pipelines	Open-source components	cloudera.com/products/cloudera-enterprise-6.html
Cloud Foundry	Cloud Foundry Container Runtime	Container mgmt platform	Open-source version available	cloudfoundry.org/container-runtime
Codefresh	Codefresh	CI/CD platform for containers	Free tier available	codefresh.io
Couchbase	Couchbase	NoSQL database	Open-source versions available	couchbase.com
CyberArk	CyberArk Conjur	Container security	Open source	conjur.org
Datadog	Datadog	Server & container monitoring	14 days	datadoghq.com
Datera	Datera	Container data services	Available by request	datera.io/solutions/#containers
Diamanti	Diamanti	Container-defined storage hardware	Demo available by request	diamanti.com/products
DigitalOcean	DigitalOcean	Container mgmt platform	Open-source versions available	digitalocean.com/products/kubernetes
Docker	Docker	Container platform	Free tier available	docker.com/get-docker
Docker	Docker Cloud	Containers-as-a-Service	Free tier available	cloud.docker.com
Docker	Docker Compose	Multi-container app tool	Open source	docs.docker.com/compose/install
Docker	Docker EE	Containers-as-a-Service, container security	Available by request	docker.com/enterprise-edition
Docker	Docker Hub	Container image registry	Free tier available	hub.docker.com
Docker	Docker Swarm	Container orchestration & clustering	Open source	github.com/docker/swarm
Docker	Docker Trusted Registry	Private container image registry	Open source	docs.docker.com/ee/dtr

Company	Product	Product Type	Open Source?	Website
Google	Kubernetes	Container orchestration	Open source	kubernetes.io
Google	Google Cloud Container Registry	Container image registry	$300 free credit	cloud.google.com/container-registry
Google	Google Kubernetes Engine	Containers-as-a-Service, container orchestration	$300 free credit	cloud.google.com/kubernetes-engine
Hedvig	Hedvig Distributed Storage Platform	Container-defined storage	Demo available by request	hedviginc.com/product#hedvig-distributed
IBM	Cloud Kubernetes Service	Containers-as-a-Service, Kubernetes-as-a-Service	Free tier available	ibm.com/cloud/container-service
InfluxData	InfluxData Platform	Database-as-a-Service	Available by request	influxdata.com/products
Instana	Infrastructure Monitoring	Infrastructure monitoring	14 days	instana.com/infrastructure-management
JFrog	Artifactory	Artifact repository manager	14 days on cloud, 30 days on prem	jfrog.com/artifactory
Joyent	Triton	Container networking & mgmt platform	Open-source components	joyent.com/triton/compute
Kontena	Kontena Platform	Container platform	Open-source version available	kontena.io/platform
Loggly	Solarwinds Loggly	Log mgmt & analytics	Free tier available	loggly.com
Mesosphere	DC/OS	Container orchestration	Open source	dcos.io
Mesosphere	Enterprise DC/OS	Container orchestration & monitoring	Demo available by request	mesosphere.com/product
Microsoft	Azure Container Service	Containers-as-a-Service, Orchestration-as-a-Service	12 months	azure.microsoft.com/en-us/services/container-service
NeuVector	NeuVector	Multi-vector container firewall	Available by request	neuvector.com/run-time-container-security
Nirmata	Nirmata	Container mgmt platform	Free tier available	nirmata.com/product
Nutanix	Acropolis Container Storage	Container storage	Free tier available	nutanix.com/products/acropolis

Company	Product	Product Type	Open Source?	Website
NuoDB	NuoDB	Distributed SQL database	Demo available by request	nuodb.com
Oracle	CrashCart	Microcontainer debugging tool	Open source	github.com/oracle/crashcart
Oracle	RailCar	Rust-based container runtime	Open source	github.com/oracle/railcar
Oracle	Smith	Microcontainer builder	Open source	github.com/oracle/smith
Oracle	Wercker	CI/CD platform for containers	Open source	devcenter.wercker.com
Packet	Packet	Infrastructure-as-a-Service	Open-source components	packet.net/features
Pivotal	Cloud Foundry	PaaS, container platform	Available by request	pivotal.io/platform
Platform9	Platform9 Managed Kubernetes	Kubernetes-as-a-service	Sandbox available	platform9.com/managed-kubernetes
Portworx	Portworx	Container data services & networking	30 days	portworx.com
Prometheus	Prometheus	Container monitoring	Open source	prometheus.io
Rancher Labs	Rancher 2.0	Container mgmt platform	Open source	rancher.com/rancher
Rancher Labs	RancherOS	Container OS	Open source	rancher.com/rancher-os
Rapid7	Logentries	Log mgmt & analytics	Available by request	logentries.com
Red Hat	Atomic Host	Container OS	Open source	projectatomic.io
Red Hat	Clair 2.0.1	Container image security	Open source	github.com/coreos/clair
Red Hat	Flannel	Container-defined networking	Open source	github.com/coreos/flannel
Red Hat	OpenShift Container Platform	Container-defined networking	30 days	openshift.com/products/container-platform
Red Hat	Quay Enterprise	PaaS, container platform	Available by request	coreos.com/quay-enterprise

Company	Product	Product Type	Open Source?	Website
Redis Labs	Redis Enterprise	Private container image registry	Free tier available	redislabs.com/redis-enterprise
Scalyr	Scalyr	Database-as-a-Service	30 days	scalyr.com/product
Sematext	Docker Agent	Log mgmt & analytics	30 days	sematext.com/docker
Sematext	Kubernetes Agent	Container monitoring & log mgmt	30 days	sematext.com/kubernetes
Sensu	Sensu Core	Kubernetes monitoring & log mgmt	Free tier available	sensuapp.org
StorageOS	StorageOS	Container monitoring	Free tier available	storageos.com/product
Splunk	Splunk Cloud	Container storage	15 days	splunk.com/en_us/products/splunk-cloud.html
Sumo Logic	Sumo Logic	Log mgmt & analytics	Free tier available	sumologic.com
Sysdig	Sysdig Cloud	Log mgmt & analytics	Open source	sysdig.com/opensource
Tigera	Canal	Container monitoring	Open source	github.com/projectcalico/canal
Twistlock	Twistlock Trust	Container image security	Available by request	twistlock.com
VMWware	CloudHealth	Container optimization	14 days	cloudhealthtech.com/solutions/containers
VMware	Photon OS	Linux container host	Open source	vmware.github.io/photon
Wavefront	Wavefront	Cloud monitoring & analytics	30 days	wavefront.com/product
Weaveworks	Weave Net	Container-defined networking	Open source	weave.works/oss/net
WhiteSource	WhiteSource	OSS detection, selection, alerting, & reporting	Open-source version available	whitesourcesoftware.com
WSO2	WSO2 Carbon	Integration platform/PaaS	Open source	wso2.com/products/carbon
WSO2	WSO2 Enterprise Integrator	ESB	Open source	wso2.com/products/enterprise-service-bus
WSO2	WSO2 Message Brokers	API mgmt	Open source	wso2.com/products/message-broker



VISIT THE

DevOps Zone

Container technologies have exploded in popularity, leading to diverse use cases and new and unexpected challenges. Developers are seeking best practices for container performance monitoring, data security, and more.

Keep a pulse on the industry with topics such as:

- Testing with containers
- Container performance monitoring
- Keeping containers simple
- Deploying containers in your organization

Visit the Zone

TUTORIALS

CASE STUDIES

BEST PRACTICES

CODE SNIPPETS