DZone

THE 2018 DZONE GUIDE TO

Containers

DEVELOPMENT AND MANAGEMENT

VOLUME II

BROUGHT TO YOU IN PARTNERSHIP WITH

aqua

CHEF

CLOUDFOUNDRY

DIAMANTI

ORACLE

PLATFORM9

Dear Reader,

Welcome to DZone's 2018 Guide to Containers — our second edition of this guide. Usage of containers has continued to climb, and Kubernetes has become the de-facto standard for orchestration, while Docker maintains its dominance as the containers of choice (although there are plenty of battles left to be fought in the container wars!). Today, containers are solidly in the heart of the developer experience. They allow developers to deliver more consistently and across a broader array of platforms — giving them control over everything from networking to security.

In this guide, we've focused on how containers enable a true multi-cloud experience, improve the developer experience, and what tools exist in the developer ecosystem to more easily manage, monitor, and secure them. Additionally, while containers are quickly becoming ubiquitous, there is still a way to go in how developers manage, maintain, and orchestrate an ever-growing number of container-based deploys. This guide attempts to explore the different options available to developers to help solve these problems.

One of the things I found most interesting about this guide, and its timing, is that it aligns particularly well with DZone's own journey to containers. Containers have been a useful tool for our development teams to easily install supporting tools locally during development, but only recently have we begun to fully embrace them. As we embed more and more containers into our everyday workflows, our developers' experience will improve — slowly at first, but then faster as we broaden our use cases and build the organizational experience.

Today, we use containers for reproducible builds of our core platform in staging environments, while leaving additional layers in place for customizations from customers. Over the course of the next month, we'll also begin to leverage containers in production — leading us directly into the challenges (and some of the solutions) outlined in this guide.

Thank you for taking the time to download this guide — we're excited by the broad spectrum of articles that are part of it and look forward to hearing your feedback. Don't forget — DZone's containers coverage doesn't stop here. Be sure to check out more great articles and opinions from the community in our Cloud Zone, Microservices Zone, and our newly launched Open Source Zone.

Happy reading.

By Matt Schmitt
PRESIDENT AND CO-FOUNDER, DZONE

DZONE IS...

BUSINESS AND PRODUCT
MATT TORMOLLEN, CEO
MATT SCHMIDT, PRESIDENT
JESSE DAVIS, EVP, TECHNOLOGY
KELLET ATKINSON, MEDIA PRODUCT MANAGER

PRODUCTION
CHRIS SMITH, DIR. OF PRODUCTION
ANDRE POWELL, SR. PRODUCTION COORD.
G. RYAN SPAIN, PRODUCTION COORD.
BILLY DAVIS, PRODUCTION COORD.
ASHLEY SLATE, DESIGN DIR.
NAOMI KROMER, SR. ACCOUNT MGR.
JASON BUDDAY, ACCOUNT MGR.
MICHAELA LICARI, ACCOUNT MGR.

EDITORIAL
CAITLIN CANDELMO, DIR. OF CONTENT & COMMUNITY
MATT WERNER, PUBLICATIONS COORD.
SARAH DAVIS, PUBLICATIONS ASSOCIATE
MICHAEL THARRINGTON, CONTENT & COMMUNITY MGR. II
KARA PHELPS, CONTENT & COMMUNITY MGR.
TOM SMITH, RESEARCH ANALYST
MIKE GATES, CONTENT TEAM LEAD
JORDAN BAKER, CONTENT COORD.
ANNE MARIE GLEN, CONTENT COORD.
ANDRE LEE-MOYE, CONTENT COORD.
LAUREN FERRELL, CONTENT COORD.
LINDSAY SMITH, CONTENT COORD.

SALES
CHRIS BRUMFIELD, SALES MANAGER
FERAS ABDEL, SALES MANAGER
ALEX CRAFTS, DIR. OF MAJOR ACCOUNTS
JIM HOWARD, SR. ACCOUNT EXECUTIVE
JIM DYER, SR. ACCOUNT EXECUTIVE
ANDREW BARKER, SR. ACCOUNT EXECUTIVE
BRIAN ANDERSON, ACCOUNT EXECUTIVE
SARA CORBIN, ACCOUNT EXECUTIVE
BRETT SAYRE, ACCOUNT EXECUTIVE

MARKETING
AARON TULL, DIRECTOR OF MARKETING
LAUREN CURATOLA, MARKETING SPECIALIST
KRISTEN PAGÀN, MARKETING SPECIALIST
JULIAN MORRIS, MARKETING ASSOCIATE

Executive Summary

BY MATT WERNER - PUBLICATIONS COORDINATOR, DZONE

Containers took the development world by storm and have seen major ecosystems develop around them. Developers who were once using Docker for their testing environments are now managing them with Kubernetes and running container-specific operating systems. To quantify and examine the change in the development world, we asked 508 tech professionals about their experiences using containers, including their choices of tools, what they expected before they used containers, and what pain points they experienced.

CONTAINERS LIVING UP TO THE HYPE

Data: Of those currently using containers, 82% expected them to make their jobs easier, 13% anticipated no change, and 5% expected their jobs to be more difficult. In practice, 75% found that their jobs were easier, 7% thought their jobs were more difficult, and 19% noticed no change in difficulty.

Implications: There was not a significant increase over last year in those who found that containers made their jobs more difficult. While 7% fewer respondents found that containers made their lives easier in practice, 4% saw no change to the difficulty of their jobs. Even so, a vast majority of respondents found that containers did make their jobs easier.

Recommendations: Reality is often very different from expectations, but even though there was a drop in those who thought their jobs would be easier with containers, there was not a similar increase in those who thought containers would make their lives harder. For the vast majority, containers are worth the investment, and it would more than likely behoove you to look at building and testing new applications using them.

CHALLENGES SEEM DAUNTING

Data: Of those who have yet to adopt containers, the biggest challenges on the horizon are the lack of developer experience (71%), refactoring legacy applications (70%), and ensuring security (43%). Those who are actively using containers have cited the major challenges as being a lack of developer experience (56%), refactoring legacy apps (50%), and application performance monitoring (33%).

Implications: Those who are actively using containers are finding most of the challenges that prospective users fear to be major concerns, but they do not experience as many issues with them. It makes sense that the lack of experience would be less of an issue in organizations using containers, and it's likely that rather than refactoring legacy apps, most organizations are simply building new applications with containers in mind. Security was seen as a much less frequent issue for those using containers, with only 28% citing it as such.

Recommendations: The best way to solve the experience issue is to start working with containers, and it may take too much time and effort to refactor legacy applications unless your organization is not focused on building brand new products. One issue that proved to be more major for container adopters was application performance monitoring, so figuring out what tools are on the market to handle that is a good first step to solving that logistical challenge.

DOCKER STILL AT THE TOP, KUBERNETES TAKES OVER

Data: Between 2017 and 2018, there has been little change in the number of organizations using containers (42% in 2017 and 45% in 2018), and an insignificant change in those not using containers (26% in 2017 to 25% in 2018). There was also an insignificant change in the containers used by organizations — Docker still reigns supreme with 91% of organizations using them in 2018. In 2017, 35% of organizations used Kubernetes, compared to 53% in 2018.

Implications: Unsurprisingly, Docker remains the most popular container technology used by organizations. While Kubernetes has been the most popular orchestration tool over the two surveys, the massive growth in Kubernetes adoption suggests that organizations are taking orchestration more seriously as a way to effectively manage containers.

Recommendations: The growth in organizations using containers has been slow, but those using containers are almost certainly going to be using Docker, and are likely to use Kubernetes. It's probably not worth it, unless there's a very specific use case, to look at other container technologies, as you'll need skills you can use between different positions and organizations. Orchestration tools are a little more flexible, as Amazon ECS and Docker Swarm are both viable alternatives to Kubernetes, and both Google and Microsoft have their own platforms for organizations using their cloud offerings — but if you're exploring on your own, Kubernetes should be your first choice.

Key Research Findings

BY JORDAN BAKER - CONTENT COORDINATOR, DZONE

DEMOGRAPHICS

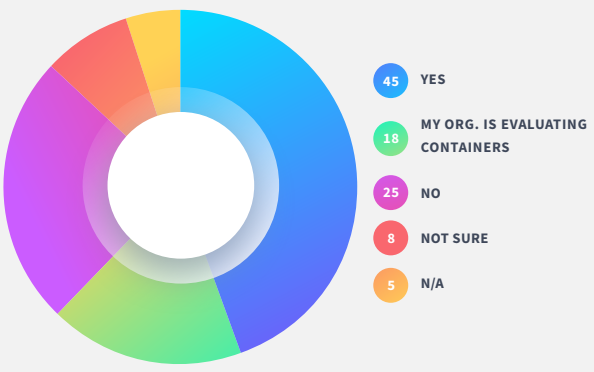
711 software professionals responded to this year's Containers Guide Survey, with 507 complete responses. The demographics for these respondents are as follows:

- 40% identify as software developers/engineers; 19% identify as developer team leads; 11% identify as architects.
- 30% work for organizations headquartered in the United States; 25% work for organizations headquartered in Europe; 14% work for organizations headquartered in South/Central Asia.
- 34% of respondents work on teams sized two to five members; 32% work on teams with six to 10 members; 11% are part of teams with 11-15 members.
- 27% of respondents work for organizations sized 100-999 employees; 19% for organizations that employ 1,000-9,999 people; 14% for organizations sized 10,000+ employees.
- In terms of industry, 27% work for software vendors; 14% work in finance/banking; 9% work in e-commerce; 9% work in consulting.
- The average experience among respondents is approximately 12 years.
- 59% primarily code in Java, 9% in JavaScript/Node.js, and 8% in Ruby.

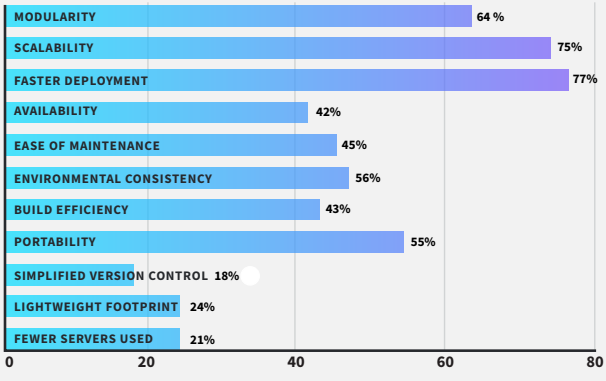
GENERAL USE OF CONTAINERS AND PREVALENCE OF DOCKER

45% of respondents told us that their organization is currently using container technology, while another 18% told us that their organization is considering adopting container technology. Of those who use containers, they appear to be rather heavily utilized throughout the development process. 87% of container users told us they

GRAPH 01. Does your organization currently use container technologies?



GRAPH 02. What benefits do container technologies offer your organization?



use the technology in development environments, 73% in a DevOps environment, 63% in QA/testing, and 54% in staging. The environments across which these containers are run is also rather evenly distributed. 51% of organizations house their containers on a hosted cloud, 48% on a private cloud, and 45% on local hardware.

A majority (61%) of respondents whose organizations use containers said they typically run one to 100 containers in production. Breaking this down, 44% of containerized respondents work for organizations that have containerized 1-25% of their workload, and 20% work in organizations with a containerized workload of 26-50%. Thus, 64% of respondents work in organizations that have containerized no more than half of their workload.

Among those who use containers, there is an overwhelming majority (91%) who use Docker, and 19% who use Docker Enterprise. The container technology LXC came in a distant third with 5% of responses. And of those organizations that are currently evaluating and considering container technology, 90% are looking at Docker, 18% at Docker Enterprise, and 6% at Solaris Zones.

METHODS AND TOOLS: ORCHESTRATION/ MANAGEMENT AND REGISTRIES

While Docker reigns supreme as a container technology, there was less of a consensus among respondents when it came to container-based methodologies and tools. In terms of orchestration/management technologies, 53% of respondents chose Kubernetes (which saw a massive jump from a 35% adoption rate in our 2017 Containers Survey), with Docker Swarm (29%) and Amazon ECS (28%) coming in second and third, respectively. Interestingly, of those developers who told us that they use Docker for their containers, 49% use Kubernetes as their orchestration/management tool and only 25% used Docker's own orchestration tool, Docker Swarm. For those organizations still evaluating orchestration/management technologies, 57% are considering Kubernetes, 33% Docker Swarm, and 30% Amazon ECS.

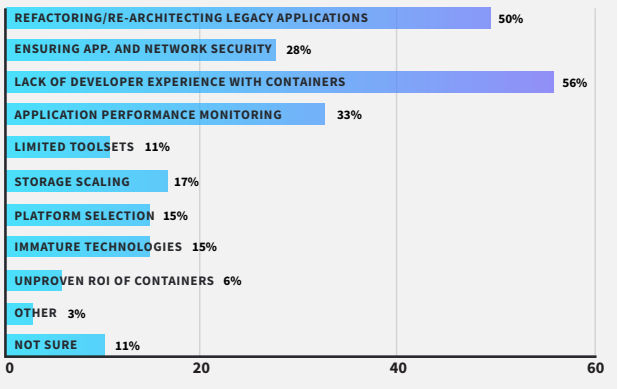
When it comes to container registries, Docker products fair better. Of those organizations using containers, 43% use Docker Hub for their registries, with 21% opting for Nexus and 20% choosing Artifactory. If we again compare these numbers to those organizations we know use Docker, we see that 53% of Docker users opt for Docker Hub for their container registries. Of those respondents who told us that their organizations are still evaluating their registry needs, Docker Hub is far and away the most popular choice for consideration, with a selection rate of 54%; the second most popular choice for this category was Nexus, chosen by 18% of respondents.

BENEFITS OF USING CONTAINERS

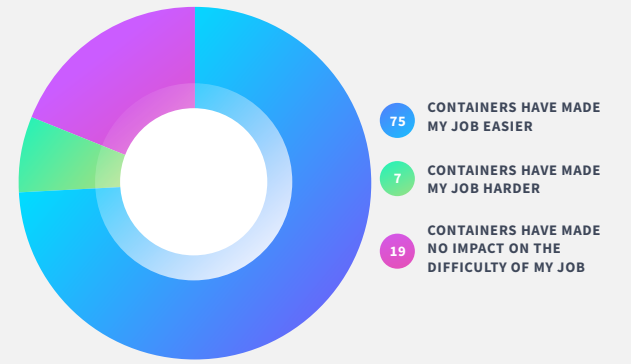
A major selling point of container technology is that it makes the lives of developers easier. When asked about their expectations prior to using containers, 82% of respondents told us that they expected containers to make their jobs easier. So, did containers live up to the hype? When we asked our audience, "Have containers made your job easier or harder?", a resounding 75% told us that containers have indeed made their jobs easier, with only 7% stating this technology made development tasks harder.

What is it about containers that makes them so widely popular? The answer boils down to three major benefits. Respondents stated that faster deployment time (77%) was the biggest benefit of containers, followed by enhanced scalability when building applications (75%) and greater modularity (64%). And, as developers gain more experience with containers,

GRAPH 03. What challenges do containers present to your organization?



GRAPH 04. Have containers made your job easier or harder?



these benefits only seem to increase. In our 2017 Containers survey, 56% of respondents reported faster deployment as a benefit (a year-over-year growth rate of 21%), and 53% reported scalability as a plus (which corresponds to a 22% growth in this year's survey). Interestingly, despite the prominence of Java developers among the DZone community (as noted in the Demographics section), the percentage of Java developers who claimed faster deployment (27% of Java devs), scalability (36% of Java devs), and modularity (31% of Java devs) were of less concern lagged behind coders from other language ecosystems. The language ecosystems that derive the most benefit from the use of containers, as stated by the responses, tended to be Python, Node.js, and Ruby.

CHALLENGES IN ADOPTING CONTAINERS

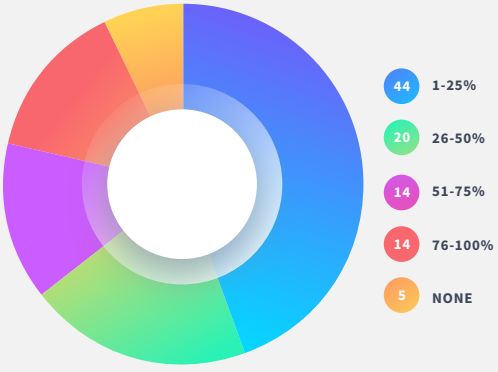
As stated earlier, despite the large-scale popularity of containers, and the benefits of faster deployment, scalability, and modularity for a multitude of language ecosystems, 25% of organizations are still not using containers and 18% are only considering adopting them. To understand why some organizations chose not to adopt container technology, let's look at the challenges containers presented to those who have adopted them, the apparent challenges present for those considering adopting containers, and the reasons why some organizations and developers have decided not to adopt container technology.

The biggest impediment reported by containers users (56%) is a lack of developer experience with this technology. Interestingly, when we compare this concern with organizational size, we see that the larger the organization, the bigger the concern over lack of developer experience with containers. 32% of respondents working for organizations sized 100-999 employees consider this a major obstacle and 25% of respondents working for organizations with 1,000-9,999 employees worry about developer experience with containers. Other oft reported challenges among container adopters were refactoring/re-architecting legacy applications (50%) and application performance monitoring (33%). Though, again, as development teams gain more experience with container technology, these problems are becoming less of an issue. In last year's survey, 68% reported developer experience as a major challenge, and 70% reported refactoring/re-architecting legacy applications as an issue. Thus, it appears that while enterprise-level software teams are still learning how to best use containers, they are quickly adapting.

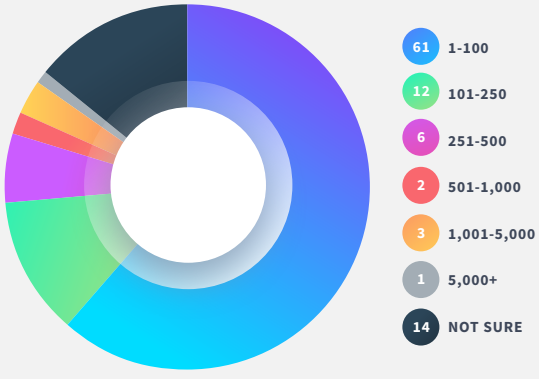
Of those respondents whose companies are still deliberating on the use of containers, a lack of developer experience with containers was the largest concern, chosen by 71%. The second largest challenge reported among this group was refactoring/re-architecting legacy applications (70%), followed by a distant third of application performance monitoring (38%). Here again, we see concern among enterprise-level developers about containers, as 23% of those reporting refactoring legacy applications work for organizations with 100-999 employees, 25% for organizations with 1,000-9,999 employees, and 17% for organizations with 10,000 or more employees.

Among those organizations that have chosen not to adopt containers, 48% report lack of developer experience as a primary concern and 24% stated that refactoring/re-architecting legacy applications was a concern. The only organizations for which ensuring application and network security was a deal breaker were those sized 1,000-9,999 employees, with 21% of respondents who work for such organization reporting thusly.

GRAPH 05. What percentage of your organization's workload is containerized?



GRAPH 06. How many containers is your organization running in production?



Azure Container Hosting Demystified

BY SAM COGAN
SOLUTIONS ARCHITECT AND MICROSOFT AZURE MVP

QUICK VIEW

01. Clarify the different options for hosting containers on Microsoft's Azure platform.

02. Get a demonstration of which types of use cases are suitable for which container hosting solutions.

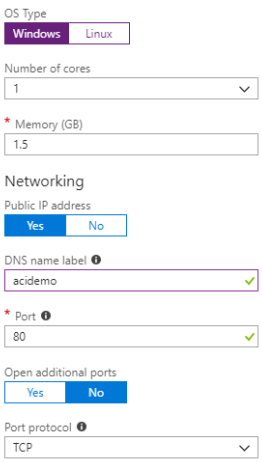
03. Map workloads to container hosting types in a simple to understand manner.

If you're just getting started with containers and want to run them on Azure, the number of options available can be overwhelming, especially with all the not-so-intuitive acronyms. There are many different options for hosting containers in Azure and all of them fill a specific need. Choosing the appropriate tool all comes down to understanding the benefits and use cases for each. In this article, we are going to take a look at the uses for each of the container hosting options in Azure.

STANDALONE CONTAINER HOSTING

If you're looking for somewhere to host a container (or containers) and don't need the services provided by an orchestrator, then the following Azure offerings may come in handy.

AZURE CONTAINER INSTANCES



Azure Container Instances (ACI) has recently gone GA. This service offers containers as a first-class instance in Azure. With ACI, you use the portal or CLI to create a container — no need to worry about provisioning VMs to host the container or installing software. ACI instances can be created with a single-line CLI or PowerShell command and will be up-and-running in minutes. Where you need multiple containers talking to each other, ACI supports container groups, which allows you to run numerous containers together on the same host and local network. ACI containers can be configured with CPU and RAM to meet your requirements and support Linux and Windows containers.

ACI is a relatively new service and does still have some limitations, e.g. it is not yet possible to attach them to an Azure virtual network. Note that running an ACI instance 24/7 will be slightly more expensive than running a VM of the same size, and you're paying a bit extra for ease of use. This means that ACI is excellent for bursty load types or tasks that run on a set schedule. We'll talk a bit more about some of its bursting capabilities when we discuss orchestrators.

AZURE WEB APP

It's now possible to host a container within an Azure Web App. Containers for Web Apps is designed for hosting web server-based containers, and it is currently limited to running in Web Apps for Linux (and so only supports Linux containers), so you're looking at containers running things like Apache, NodeJS, Nginx, etc. Setting this up is as simple as creating a new web app and then configuring it to point to a container registry. With containers for web apps, you pay the standard price for the size of web applications you select; there is no additional cost for using containers.

As these are Linux containers only, you're limited to languages supported on this OS in particular. If you want to use .NET, then it needs to be .NET Core.

AZURE FUNCTIONS

With the release of Azure functions 2.0 and Azure functions on Linux, we now have the option to use containers for our function application. Again, this is Linux-only containers, and you need to use microsoft/azure-functions-runtime as the base for your container image. This image contains the Azure Functions runtime that is required to allow this to work with the Azure Functions infrastructure.

When you move beyond running a couple of containers and need to be able to control multiple containers that are running in production with resilience and high availability, then you need to start looking at using an orchestrator.

Once you have deployed your image, you have access to all the functionality of Azure functions for your event- or schedule-driven workloads. You also have the same limitations such as the ten-minute maximum execution time for consumption-based functions.

AZURE BATCH

Azure Batch is Microsoft's solution for large-scale parallel batch processing. Classically, this involves spinning up virtual machines either from a gallery image and loading applications or a custom image. With containers for batch, we can now encapsulate all our application files inside of a container image rather than having to deploy them onto the gallery image or manage a full-blown VM image.

With container workloads for batch, you are still deploying the batch VMs, but these are running an OS version with container support enabled, onto which your container image is then loaded. Your container has access to the full resources of the VM it is running on (one container per VM). Containers for batch also supports the ability to pre-fetch your container image for your batch pool to reduce deployment time.

STANDALONE VM

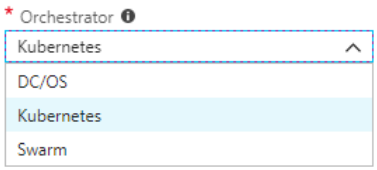
There is nothing stopping you from creating an Azure VM, installing Docker, and running your containers from there — and for dev and test workloads, this may be a good option. This approach, however, *does* mean you are on the hook for managing the underlying VM, updates, security, DR, etc. The Azure Docker Extension can be used to install Docker on the VM at deployment time.

ORCHESTRATORS

When you move beyond running a couple of containers and need to be able to control multiple containers that are running in production with resilience and high availability, then you need to start looking at using an orchestrator. Again, there are numerous options for hosting orchestration tools in Azure.

AZURE CONTAINER SERVICE (ACS)

ACS was the original container orchestration service in Azure. ACS allows you to create a container orchestration cluster running one of Docker Sware, Kubernetes, or Mesos. ACS is an interesting solution in that it is deployed like a PaaS service, but what gets implemented is a set of IaaS infrastructure (vNet, NSGs, virtual machines, etc.) that has been pre-configured to run your selected orchestrator. While the platform will set it all up for you because it is IaaS, you are still responsible for managing the VMs, updates, etc.



ACS is being replaced by AKS and will be removed around 12 months from the GA announcement of AKS. As such, I do not recommend making use of ACS for new workloads. Instead, use AKS or ACS engine (see below).

AZURE KUBERNETS SERVICE (AKS)

AKS is Microsoft's semi-managed Kubernetes service and is the replacement for ACS. Whereas ACS supported three orchestrators, AKS Microsoft made the (somewhat safe) bet on Kubernetes and this is the only orchestrator available. Where

ACS deployed all IaaS infrastructure, AKS does offer part of the service as a managed PaaS offering. When you use an AKS cluster the Kuberntes master nodes are hidden from you and are operated by Microsoft, the only VM resources you see are the worker nodes, which are still IaaS and need to be managed and updated as usual. In AKS, you are only paying for the worker nodes, the managed part is free.

NAME	PUBLISHER	CATEGORY
Kubernetes Service	Microsoft	

AKS is relatively new and is still in preview (with GA expected shortly). Because of this, it does not yet support all the features of ACS. In particular, Windows container support is missing. If you need Windows support, you would need to look at ACS engine.

I do not recommend making use of ACS for new workloads. Instead, use AKS or ACS engine.

AZURE CONTAINER SERVICE ENGINE (ACS ENGINE)

ACS Engine is a toolset that both ACS and AKS are built on top of, and which has been open-sourced for anyone to use. With ACS engine, you can use JSON files to detail the configuration of your orchestrator, then run ACS Engine to have it generate ARM templates that you can then deploy to Azure. The result of ACS engine is a container orchestrator running on IaaS infrastructure that is all customer-managed (no PaaS service here). The benefit of ACS engine is that you can use it to create clusters that aren't currently supported in AKS — things like Windows containers, mixed Windows and Linux clusters, later or beta versions of Orchestrators, support for Orchestrators other than Kubernetes, etc. If you need to use any of these options that are not supported in AKS or you want more control over the orchestrator cluster that gets created, then ACS engine would be the choice for you.

SERVICE FABRIC

Service Fabric is Microsoft's microservice platform, but it now also supports running and orchestrating containers. If you're purely looking at .NET applications, and you want to take advantage of some of the programming paradigms like reliable actors, then this is the platform to use.

OTHER SERVICES

While not running containers, there are also products that provide services that will be useful to you when deploying containers into Azure.

AZURE CONTAINER REGISTRY (ACR)

ACR is Microsoft's solution for private container registries. Using ACR, you can create a registry and then have your AKS cluster connect to it using Azure Role-Based Access Control. ACR also supports geo-replication, so if you are running multiple global clusters for DR or performance, then this provides an easy way to distribute your images.

AZURE FILES AND MANAGED DISKS

Azure Files and Managed Disks have been around for some time and are used heavily outside the container world, but for containers, they provide useful tools for persisting data. ACI can make use of Azure file shares for persisting data, whereas AKS and ACS can make use of both Azure files and managed disks, giving options for increased performance for container storage with premium disks if required.

SUMMARY

There is a good deal of options for deploying containers in Azure, and I suspect more will be coming in the future. It can seem a little overwhelming trying to decide which to use, but each does have its own niche, and with an understanding of this, you can determine which is best for your use case.

SAM COGAN is a Solutions Architect and Microsoft Azure MVP, focused on providing architecture and technical solutions for cloud-based SaaS projects using Azure. Sam is particularly focused on areas around cloud automation and deployment, DevOps, configuration management, and high-performance and grid computing in the cloud. Sam blogs about Azure and cloud related topics at samcogan.com.

Looking for the fastest path to container nirvana? Meet Habitat.

Habitat from Chef automates the building, packaging and deployment of your apps for containers, the cloud, legacy servers. All from a single artifact. See how Habitat makes your container dreams come true:

- Automate builds and application packaging integrated in your CI/CD pipeline
- Create packages that are portable across major container technologies, including Kubernetes, Docker, Mesos, Cloud Foundry and more!
- Modernize your existing custom or commercial applications with containers without fear.
- Build apps that you can deploy on legacy OSes, cloud platforms and more without starting over. Habitat packages can deploy where you want them on demand.

Ready to make the move? Learn more at http://chef.io/gethabitat

> "In today's world, supporting containers and microservices is a necessity. Habitat can help get us there with applications that run anywhere."
>
> **Steven Mullinax**, Dir. Infrastructure Engineering, Alaska Airlines

habitat

CHEF™
CHEF.IO

Accelerate Container Adoption With Habitat

Habitat by Chef accelerates your adoption of container technology by making sure your containers are well-built. Customers use Habitat for two main scenarios: first, lift-shift-and-modernize initiatives to extract legacy applications from underlying infrastructure and redeploy them onto newer, modern platforms like cloud VMs or containers; and second, to solve issues around packaging and deployment that the container ecosystem and even Kubernetes do not address. Applications packaged with Habitat look identical, creating portability and a unified management layer across old and new. In other words, there is no such thing as bimodal IT with a Habitat-led approach. Everything is fast IT.

Habitat packages apps in a unique two-stage process, first by creating a portable, platform-independent artifact that can then be exported to a platform-dependent artifact, such as a container, VM image, PaaS image, or tar file to run on bare metal. These artifacts are guaranteed to be not only immutable but isolated from the operating system. Strict dependency management means you know exactly what is inside the application package. Best of all, when you build a container using Habitat, it does not contain a full operating system image. This cuts down the container's size — now it only needs to contain your application and its dependencies, freeing you from traditional systems management problems like scanning containers for security vulnerabilities or patch management.

> There is no such thing as bimodal IT with a Habitat-led approach. Everything is fast IT.

Habitat has many other features, but the main benefit it gives you is one-click deployment orchestration for all of your applications, legacy or modern, whether you're deploying into containers or not. Try it today at habitat.sh.

WRITTEN BY JULIAN DUNN
DIRECTOR OF PRODUCT MARKETING - CHEF SOFTWARE

PARTNER SPOTLIGHT

Habitat by Chef

Automate the process of building, deploying, and managing any application in any environment — from traditional data-centers to containers

CHEF

CATEGORY
Automated Application Delivery, Packaging, and Management

OPEN SOURCE?
Yes

NEW RELEASE
Continuous: monthly major releases

CASE STUDY

Alaska Airlines is the fifth-largest U.S. airline, known for embracing innovative technology to improve customer experience. Calling themselves "a software company with wings," Alaska Airlines builds and operates mission-critical software to keep passengers moving safely and on time. Habitat by Chef helps Alaska Airlines move fast, improve efficiency, and manage risk.

Using Habitat:

- Alaska is able to build applications in one consistent way and deploy to multiple clouds as needed.
- Habitat helps Alaska maintain consistency in deployments and reduce downtime with non-disruptive updates.
- Habitat offers Alaska the flexibility to support hybrid environments and multiple application architectures as business needs change and new technologies emerge.

STRENGTHS

- Quickly build portable applications and deploy to servers or containers.
- Ensure your applications are well-managed and supervised.
- Automatically trigger dependent builds in CI/CD pipelines.
- Integrates with modern container controllers including Kubernetes, Docker, CloudFoundry, and AKS.

NOTABLE CUSTOMERS

- GE
- Ticketmaster
- Alaska Airlines
- NCR
- Rakuten

WEBSITE chef.io/habitat **TWITTER** @habitatsh **BLOG** habitat.sh/blog

Deploying Containerized Applications on Google App Engine

BY ALAN HOHN
LOCKHEED MARTIN FELLOW

QUICK VIEW

01. Deploying an application using Platform-as-a-Service (PaaS) is appealing because it simplifies development and deployment.

02. Once we get past the simple cases, we need to customize the environment where our application runs.

03. Using a containerized PaaS such as Google App Engine means we get our own Docker container, with all the isolation and customization that implies.

GOOGLE APP ENGINE

Platform-as-a-Service (PaaS) has become popular to build and deploy applications without having to worry about the underlying infrastructure that runs the application. When using PaaS, however, we need to make tradeoffs. Most importantly, when we give up control of the underlying infrastructure, we lose some ability to configure that infrastructure.

In this environment, a containerized PaaS such as Google App Engine (GAE) can provide a hybrid approach between letting the PaaS provider control the application runtime and having to build the runtime ourselves. Because applications in GAE run in Docker containers, we have the flexibility to customize the infrastructure while still being able to think in terms of deploying applications as opposed to deploying infrastructure. At the same time, there's a challenge in working with a PaaS like GAE because its application deployment behavior can be opaque. Sometimes, this means that you try a deployment before you know whether it will work.

To see how deployment to Google App Engine works and to understand some of the capabilities and challenges of customizing the infrastructure, let's look at a Node.js application deployed to GAE. This is very similar to an application that I built recently, but simplified for clarity. The example application provides a REST API that performs server-side rendering of Mermaid diagrams; this is challenging because it requires us to run a headless Chromium instance, which puts some unique demands on the application infrastructure.

To follow along, you can access the complete source code here. You'll need a GAE account and Google SDK installation. The installation instructions discuss running `gcloud init`, which will enable you to provide your account information and register your first application.

Note: While I'm describing customization in GAE, similar logic applies to customizing deployments in other common PaaS environments such as Heroku.

NODE.JS GAE APPLICATION STRUCTURE

For a standard Node.js application, there is very little we need to deploy it to GAE. Most importantly, we need an `app.yaml` file at the top level. This can be very simple:

```
runtime: nodejs
api_version: '1.0'
env: flexible
threadsafe: true
manual_scaling:
  instances: 1
```

The two important lines in this file are `env` and `runtime`. The

`env` line tells GAE to run this in the "flexible" environment, which means the runtime will use Docker. The `runtime` line specifies the Node.js runtime.

Other than `app.yaml`, the repository is a standard Node.js server using Express. The only concession we must make to GAE is to accept a `PORT` environment variable telling us what port to listen on; this allows GAE to put a load balancer in front of our application. We handle this inside the `server.js` startup script for the application, defaulting to 3000 if `PORT` is not set:

```
...
const port = process.env.PORT 3000;
...
```

The rest of `server.js` configures Express to route traffic to REST API endpoints. We start by configuring the `body_parser` middleware:

```
...
app.use(bodyParser.text())
...
```

This tells Express to pass the request body through the `text()` parser. This parser reads the request body in as plain text. By default, however, it only handles HTTP requests where `Content-Type` is set to `text/plain`, so our client requests will need to match that expectation.

We next create the API endpoint `/render`:

```
...
app.post('/render', function(req, res){
  render(req.body, function(content) {
    res.send(content);
  });
});
...
```

We provide a function that runs on a POST request to a specific URL path. By the time the function is called, the middleware has already read the request body, so we can access it as `req.body`.

Finally, we listen on the configured port:

```
...
app.listen(port, () => console.log(`Mermaid Server on
port
${port}`));
...
```

The provided lambda function is called once the server is established.

Now that we have our Node.js application and our `app.yaml` file, we run `gcloud app deploy` to deploy the application to GAE. GAE will build a Docker container with our application files inside, run `npm install`, and then run `npm start` to run our application. It provides a load balancer that accepts traffic at the hostname identified when we created the application and performs HTTP/S termination for us so that we don't have to configure our own SSL certificate.

For our example application, since it exports a REST API, we can use httpie to send it a Mermaid text and get a diagram back.

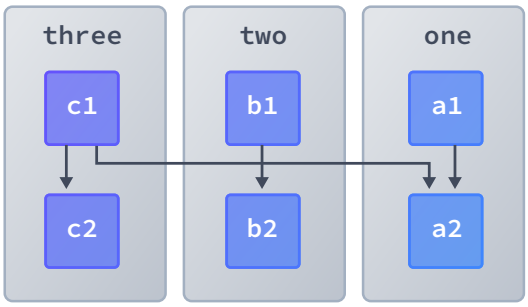
File `mermaid.txt`:

```
graph TB
    c1-->a2
    subgraph one
    a1-->a2
    end
    subgraph two
    b1-->b2
    end
    subgraph three
    c1-->c2
    end
```

Command:

```
cat mermaid.txt |
  http https://mermaid-server.appspot.com/render
  'Content-Type:text/plain' > render.png
```

This results in a PNG file with our rendered graphic:



CUSTOMIZING THE CONTAINER

As you can see, deploying a Node.js application to GAE is very simple. What makes this example a little more challenging

is that it uses Mermaid, a JavaScript library that renders diagrams from a text description. Mermaid relies on having a browser available with SVG support. Fortunately, we can provide this on the server side in our Node.js application using Puppeteer. Puppeteer downloads and runs Chromium and allows us to interact with a Chromium instance directly from Node.js. This includes opening tabs, loading pages, running JavaScript, and taking screenshots of the rendered content.

When using PaaS, however, we need to make tradeoffs. Most importantly, when we give up control of the underlying infrastructure, we lose some ability to configure that infrastructure.

To get Chromium to work, we need some libraries that aren't present in the default Node.js container that GAE uses. We therefore need some way to get these libraries installed in our Docker container before Puppeteer will work as expected. One option would be to use `runtime: custom` in our `app.yaml`. This would tell GAE to look for a `Dockerfile` in the same directory as `app.yaml`. You can see an example of this in the "try-puppeteer" repository.

There is an alternative, however, that avoids the need for a `Dockerfile`. We can take advantage of Node.js preinstall to run commands when our container is set up. We include this in our `package.json`:

```
"scripts": {
  "preinstall": "node preinstall.js"
},
```

This allows us to put any content we want into `preinstall.js`, and it will be executed prior to `npm install`.

In `preinstall.js`, we first check to see if we're running in production. If so, we use the `child_process` module to run the necessary `apt-get` commands to install Chromium dependencies.

```
const child_process = require('child_process');

const ne = process.env.NODE_ENV;
if (ne !== undefined && ne == 'production') {
  console.log("Production environment…");
  child_process.execSync('apt-get update',
stdio:'inherit'});
  child_process.execSync('apt-get -y install libxss1
ibgconf-2-4 libatk-bridge2.0-0 libgtk-3-0 libx11-xcb1
ibnss3 libasound2', {stdio:'inherit'});
} else {
  console.log("Production not detected, nothing to do");
}
```

Because we're in a Docker container and already running as root, `sudo` is not necessary to run `apt-get`. The exact list of dependencies is, of course based on what is needed for Chromium; I started with this post but tested the exact set of libraries needed so I could cut down the list somewhat.

The `{stdio:'inherit'}` configuration for `execSync` was essential in developing this; it causes Node.js to send the output of the `apt-get` commands to `stdout`, which means it shows up in the GAE logs for the application. This got me past a couple cases where package names had changed. The logs are available in the GAE console or using the `gcloud app logs read` command.

CONCLUSION

Platform-as-a-Service has made application deployment fast and easy by hiding the work of configuring the lower-level infrastructure. Sometimes, however, we need access to that infrastructure to configure it the way we want. For containerized PaaS environments such as Google App Engine, we can leverage the capabilities of the underlying Docker container to customize our application's infrastructure as needed without losing our simplified development and deployment model.

ALAN HOHN is a Lockheed Martin Fellow with a background as a software architect, lead, and manager. He has a lot of experience re-architecting embedded systems, mostly using distributed Java, and in combining embedded and enterprise approaches. Lately he's been doing quite a bit of work with virtualization, DevOps, and cloud technologies.

CHECKLIST

Should You Use Containers?

BY SHRAVANI CHAKRABORTY, INTELLIPAAT

Potential Benefits of Containerization

The technology of containerization gained momentum after Docker launched its open-source container technology and AWS started providing container services supported by Docker. The benefits harnessed by this technology are:

- **Platform independence:** An application that is encapsulated with all of its dependencies into a container is able to run on any machine, making it a highly portable technology.
- **Rewriting code is not required:** One of the prime benefits a company can harvest is that developers need not write code all over again for every platform the container is going to run on. This saves time, money, and effort all at the same time.
- **Best-suited for in-house application development:** Containerization of applications is best-suited for companies that develop in-house applications. This helps in porting the applications and makes sure that these will function in production as it used to during development.
- **Dividing any application:** Containerization not only helps divide in-house applications — any kind of application can be successfully divided into microservices, making the entire system lot faster than before.
- **Lightweight in nature:** Containers are quite lightweight, as in order to port an application, the entire operating system needs to not be encapsulated along with it. It is just the application that goes inside the container, which makes it less complicated and more light-weight.

Containerization vs. Virtualization

The description above about containerization sounds pretty similar to virtualization, right? It is true that this concept has emerged from virtualization, but there is a huge difference between these two. Let's compare them:

APPLICATION ABSTRACTION	HARDWARE ABSTRACTION
Virtualizes the OS	Virtualizes hardware
Every container shares the same base underlying OS	Every VM has its own hardware, application, and OS
Runs of top of the host of the OS	Runs on top of hypervisor installed on a bare-metal system
Lighter overhead	Heavier overhead

While virtualization has provided the abstraction of a machine, you can call containers a moderately upgraded version of VMs.

Container Implementation Challenges

With the latest infusion of an open-source platform by Docker, this technology has grabbed a lot of attention because of the benefits achieved by its implementation. However, there are certain challenges that are faced during deployment.

- **One OS crash fails the entire system:** Containers are considered to be resource-efficient, as all the containers access the same OS. However, this provision makes the entire system prone to failure — a single crash in the OS will cause all the containers to crash.
- **Compatibility issues:** Migrating containers can be a tedious task, as you face a challenge if the server OS is not compatible, which limits its migrations ability.
- **Not suitable for all tasks:** Containerization is better-suited for applications that are made of small chunks of applications and can be divided into microservices. However, some applications are not formed in this pattern, so it is not possible to subdivide. Still, these applications can be successfully run on virtual machines.

How to Decide If You Should Implement Containers

Your decision depends purely on your requirements. If you need to run several instances of a single application, then implementing containers can be useful. But if you want to run multiple applications, then you should go for VMs.

If you have to run just one operating system, then containers can be beneficial. At the same time, if you want the freedom of running any operating system, then VMs may be the better option for you. Though these differences may make you consider implementing both technologies, these can co-exist on a single system serving different purposes inside a firm.

However, this technology is still in its nascent stage, setting space for more cutting-edge technologies to revolutionize the digital economy altogether.

CLOUD FOUNDRY

SAVES YOU

10 WEEKS

$100,000

ON AVERAGE PER APP DEVELOPMENT CYCLE

GET THE FULL REPORT

cloudfoundry.org/users2018

2018 CLOUD FOUNDRY USER SURVEY

Running Containers at Enterprise Scale

The popularity of containers is on the rise as more companies shift to a cloud-native application architecture and a continuous delivery model. Containers are one of the popular options for deploying and managing applications in a cloud-native way. They are being used for new applications, refactored applications, distribution of packaged software, and even to replatform legacy applications.

Running containers at enterprise scale is complex. As environments grow, that complexity increases. In order to run applications at scale, today's enterprises require adaptability in their tech stack. Cross-platform flexibility provides the ability to withstand shifts in technology so you can adopt new tools, languages, or platforms down the road. Cloud Foundry Container Runtime (CFCR) gives you more granular control and management of containers with Kubernetes. CFCR delivers a uniform way to instantiate, deploy, and manage highly available Kubernetes clusters using BOSH.

Different types of applications call for different approaches. Cloud Foundry gives you the choice. If you need more flexibility and developer-built containers for apps, or are using pre-packaged apps delivered in a container, using CFCR will give you control over application delivery. Cloud Foundry Application Runtime (CFAR) utilizes containers as part of its DNA, and has since before Docker popularized containers. CFAR will likely be the best deployment method for many applications – not just 12-factor microservices.

Enterprises should be able to ship software whenever, wherever, and however they want, and trust that their applications are safe. Cloud Foundry technologies enable you to do just this, regardless of which Runtime you choose.

Learn the concepts.

WRITTEN BY CHIP CHILDERS
CTO, CLOUD FOUNDRY FOUNDATION

PARTNER SPOTLIGHT

Cloud Foundry

Cloud Foundry gives you the right tool for the right job with two complementary open source technologies for app developers and operators.

CLOUD FOUNDRY

CATEGORY
Platform-as-a-Service (PaaS)

RELEASE SCHEDULE
Continuous

OPEN SOURCE?
Yes

CASE STUDY

Chinese telecommunications giant Huawei has annual revenues of $46 billion and 170,000 employees — they also have a strong presence in the United States market. They deploy 4,500 applications, but in their effort to evolve faster and disrupt their highly competitive market, their Chief Architect Dr. Ying Xiong asked: "Can we deploy 300-400 containers per minute?"

Huawei aimed to create a self-monitoring and self-healing solution, with fast recovery and auto-scaling, with continuous deployment and application upgrades in a multi-language, multi-framework development environment.

The solution? A Communications-as-a-Service initiative built on Cloud Foundry PaaS-- and a Cloud Foundry-certified FusionStage platform which supports the CF CLI, using Diego as its runtime engine and Kubernetes and Docker as tools for container orchestration. Today, Huawei can:

- Deploy up to 400 containers per minute
- Innovate faster on key products, such as enabling video monitoring from a phone
- Empower developers to focus on writing code

STRENGTHS

- Polyglot and multi-cloud
- Run apps at scale
- Simplify the development lifecycle and container management

NOTABLE USERS

- Allstate
- American Airlines
- Cloud.gov
- Ford
- The Home Depot

WEBSITE cloudfoundry.org **TWITTER** @cloudfoundry **BLOG** cloudfoundry.org/blog

Istio Service Mesh for Containers

QUICK VIEW

01. Get an understanding of what a service mesh is.

02. Learn when to use a service mesh.

03. Learn what Istio is.

04. See how Istio provides service mesh capabilities.

BY CHRISTIAN POSTA
CHIEF ARCHITECT, CLOUD APPLICATION DEVELOPMENT AT RED HAT

Containers have simplified how we deploy and manage applications by abstracting the underlying infrastructure and focusing on the app and its dependencies. This allows us to run our applications on any kind of deployment platform (e.g. Kubernetes, Nomad, Mesos, etc.) whether self-managed on-premise, self-managed in the public cloud, or as a service in the public cloud. As we continue building our applications and services in the cloud and also integrate with our existing, non-cloud-native workloads, we'll end up trying to solve a lot of layer-seven networking issues like service discovery, load balancing, resiliency, and others. We want to focus on building differentiating applications and services, not on getting distracted with these new infrastructure problems. Istio is an open-source project from the folks at Google, IBM, Lyft, Red Hat, Tigera, and others that aims to solve these application networking issues irrespective of application architecture (microservices, functions, monoliths), language (Java, Go, Python), deployment platform (Kubernetes, Mesos, etc.), or even location (private datacenter, public cloud, hybrid, etc.). Containers give a unifying way to package and deploy applications across infrastructure. Istio gives application owners and operators a way to unify how those applications talk with each other.

ISTIO SERVICE MESH

Istio is an implementation of a service mesh. A service mesh is decentralized application networking infrastructure that transparently enables applications to communicate securely and reliably, and also adds observability and traffic/policy control. A service mesh does this by using application proxies through which all of the traffic passes. These "service proxies" are deployed one-to-one with application instances and all network traffic for that application instance passes through its own proxy. So, in other words, for each instance of an application, it has its own proxy. Whenever an application needs to communicate over the network, e.g. communicate with another service or make calls to a database, its calls transparently flow through the proxy so the proxy can add additional networking functionality. This means any of our services can be enhanced to use a service mesh without even knowing it's there.

In Istio, the default proxy is a powerful layer-seven proxy named Envoy Proxy. Envoy is deployed as a "side car" next to our application instances and provides the bulk of the networking functionality. A "side car" is a helper process that gets deployed alongside our main application's process to enhance its capabilities. With containers, these side-car deployments are packaged up just like any of our applications and are highly reusable. Some container deployment platforms, like Kubernetes, make it easy to deploy both a side-car container and our application container as an atomic unit. This way, the application seems to have additional capabilities but with a much more composable and reusable architecture.

RESILIENCY AND OBSERVABILITY FROM OUR DATA PLANE

The application proxies in our architecture through which all application traffic flows make up the "data plane" in our service mesh. The data plane is where things like reliability, observability, and control can be implemented. For example, if a "shopping cart service" needs to interact with a "product catalog service" to look up details about certain items in a customer's virtual shopping cart, it may initiate this with a request/reply protocol like HTTP or gRPC. When making requests over the network, we must be weary of latency, network hops, and network failures. The product catalog service could be overloaded and return errors, or worse, introduce significant latency while processing a request. With the data plane, we can implement resiliency outside of the applications by adding request timeouts, retries, and circuit breaking to the request path via the data plane proxies without expecting the application to properly code this into its implementation.

> Containers give a unifying way to package and deploy applications across infrastructure. Istio gives application owners and operators a way to unify how those applications talk with each other.

Since all traffic would be flowing over the data plane proxies, we have a great opportunity to collect metrics and other signals about what's happening when services communicate with each other. With Istio, Envoy collects very detailed telemetry about the connections, requests, failures, and protocols that can aid debugging and overall understanding of the system at run time. Using our previous example, the application proxy for the shopping cart service can use this telemetry to report back to its service operators that we've had a drop in the throughput of the service because calls to the product catalog are taking a lot longer to succeed.

A DATA PLANE NEEDS A CONTROL PLANE

Requests and messages flow through the data plane, which is made up of application proxies. As operators of our individual services, we need a way to set rules about how the data plane proxies should behave. We need both an API as well as higher-level constructs through which to reason and establish these rules. Istio's control plane provides three areas that operators can leverage to manage their service mesh:

1. **Istio Pilot**: Establish configuration of traffic routing and resilience policies.
2. **Istio Mixer**: Handles usage policies (quota, ACL, rate limiting, etc.) and provides a central point for collecting telemetry.
3. **Istio Auth**: Manages certificate issuance, revocation, and rotation that can be used to prove identity and establish mTLS.

With the Istio control plane, we can control how traffic gets routed between our services and any resilience policies we expect callers of our service to honor. In Istio, all of the configuration is treated as YAML resource files that get sent to an API. The set of resource files you send to the API get synthesized to proxy-specific configuration that gets sent to the data plane, which then enforces the rules. For example, with Istio, we can define a `VirtualService` rule to specify traffic behaviors and qualities. A `VirtualService` rule looks like this:

`catalog-route.yaml`:

```
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: catalog-route
spec:
  hosts:
  - catalog.prod.svc.cluster.local
  http:
  - match:
    - uri:
        prefix: “/newcatalog"
    route:
    - destination:
        host: catalog.prod.svc.cluster.local
        subset: v2
  - route:
    - destination:
        host: catalog.prod.svc.cluster.local
        subset: v1
```

With this simple rule, we're specifying any service that tries to talk to the `catalog.prod.svc.cluster.local` service that it

should be routed to the `v1` of that service. If we send requests to the catalog service at `catalog.prod.svc.cluster.local/newcatalog`, then our request should be routed to `v2` of the catalog service. With Istio, we can set request-specific rules for how a request gets routed within the cluster, which gives us fine-grained control over traffic flow.

> With the Istio control plane, we can control how traffic gets routed between our services and any resilience policies we expect callers of our service to honor.

With `DestinationRule`, we can define how load balancing and circuit-breaking work. For example, if we're trying to communicate with the `catalog.prod.svc.cluster.local` service, we can specify a rule like the following, which defines how consumers of the service interact with it and maintain resilience during network issues:

`catalog-destination-rule.yaml`:

```
apiVersion: networking.istio.io/v1alpha3
kind: DestinationRule
metadata:
  name: catalog-destination-rule
spec:
  host: catalog.prod.svc.cluster.local
  trafficPolicy:
    loadBalancer:
      simple: LEAST_CONN
    outlierDetection:
      http:
        consecutiveErrors: 5
        interval: 5m
        baseEjectionTime: 15m
```

In this rule, we specify to use the least-connection load balancing rule and set the "outlier detection" rules to trip after five consecutive errors. The "outlier detection" functionality behaves very closely to how a circuit breaker behaves. If it sees five errors to the same endpoint within 5m, it will remove the endpoint from the load balancing pool for a minimum period of 15m.

Creating these rules can be done by posting to the Istio API server (Istio Pilot); however, a convenience CLI is provided with the Istio distribution. For example, you can use the *istioctl* CLI tool to create the previous destination rule:

```
$ istioctl create -f catalog-destination-rule.yaml
```

SECURITY AS A FIRST-CLASS CONCERN

A service mesh is intended to provide a uniform communication control and observation infrastructure regardless of how you deploy your applications. With Istio, security is a first-class concern. As systems become more distributed and deployed on untrusted cloud networks, we need a way to secure that communication. Istio leverages a new spec called SPIFEE (secure production identity for everyone) to assign and validate identity of workloads. For example, Istio on Kubernetes leverages the "service-account" construct from Kubernetes and encodes that into x.509 certificates that get assigned when your application gets deployed. Istio Auth then manages the issuance of these certificates, which the data plane then uses to secure the communication transport without the application having to configure anything. We can get strong client and server verification (mutual TLS) between the services that add a layer of security that would otherwise be very difficult to set up and maintain.

Service mesh is a hot topic right now in the container space for good reason. Getting applications built and deployed with containers is a great abstraction to enable organizations to take advantage of cloud infrastructure; however, those workloads will need to communicate with each other in non-trivial ways. Istio provides an open-source implementation of a service mesh that aims to tackle the hard problems of service communication like traffic control, reliability, observability, and security. Check out istio.io to get started and keep up with the latest advancements.

CHRISTIAN POSTA is a Chief Architect of cloud applications at Red Hat and well known in the community for being an author (*Introducing Istio Service Mesh*, O'Reilly 2018, *Microservices for Java Developers*, O'Reilly 2016), frequent blogger, speaker, open-source enthusiast and committer on various open-source projects including Istio, Apache ActiveMQ, Fabric8, etc. Christian has spent time at web-scale companies and now helps companies create and deploy large-scale, resilient, distributed architectures - many of what we now call microservices. He enjoys mentoring, training and leading teams to be successful with distributed systems concepts, microservices, DevOps, and cloud-native application design.

Creating a Positive Developer Experience for Container-Based Applications: From Soup to Nuts

BY DANIEL BRYANT
TECHNICAL CONSULTANT AND PRODUCT ARCHITECT, DATAWIRE

QUICK VIEW

01. CI/CD often needs to be adapted or enhanced when embracing a technology as disruptive as containers, and so does the entire developer experience.

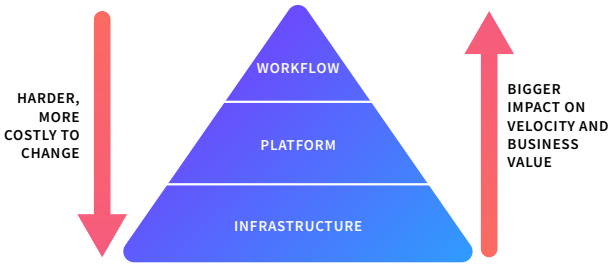
02. This article examines the key points within a typical workflow, and suggests how practices need to change when working with containers.

03. We will also explore the need for new testing techniques and see how these must be integrated with observability and monitoring platforms.

Nearly every engineering team that is working on a web-based application realizes that they would benefit from a deployment and runtime platform — and ideally, some kind of self-service platform (much like a PaaS) — but as the joke goes, the only requirement is that "it has to be built by them." Open-source and commercial PaaS-es are now coming of age, particularly with the emergence of Kubernetes as the *de facto* abstraction layer over low-level compute and network fabric. Commercially packaged versions of the Cloud Foundry Foundation's Cloud Foundry distribution (such as that by Pivotal), alongside Red Hat's OpenShift, are growing in popularity across the industry. However, not every team wants to work with such fully featured PaaSes, and now, you can also find a variety of pluggable components that remove much of the pain of assembling your own bespoke platform. However, one topic that often gets overlooked when assembling your own platform is the associated developer workflow and developer experience (DevEx) that should drive the selection of tooling. This article explores this topic in more detail.

INFRASTRUCTURE, PLATFORM AND WORKFLOW

In a previous TheNewStack article, *Kubernetes and PaaS: The Force of Developer Experience and Workflow*, I introduced how the Datawire team often talks with customers about three high-level foundational concepts of modern software delivery — infrastructure, platform, and workflow — and how this impacts both technical platform decisions and the delivery of value to stakeholders.



IF KUBERNETES IS THE PLATFORM, WHAT'S THE WORKFLOW?

One of the many advantages with deploying systems using Kubernetes is that it provides just enough platform features to abstract away most of the infrastructure — at least to the development team. Ideally, you will have a dedicated platform team to manage the infrastructure and Kubernetes cluster, or perhaps use a fully managed offering like Google Container Engine (GKE), Azure Container Service (AKS), or the soon-to-be-released Amazon Elastic Container Service (EKS). Developers will still benefit from learning about the underlying infrastructure and how the platform interfaces with this (cultivating "mechanical sympathy"), but fundamentally, their interaction with the platform should largely be driven by self-service dashboards, tooling, and SDKs.

Self-service is not only about reducing the development friction between an idea to delivered (deployed and observed) value. It's also about allowing different parts of the organization to pick and choose their workflow and tooling, and ultimately

make an informed trade-off against velocity and stability (or increase both velocity and stability). There are several key areas of the software delivery life cycle where this applies:

- Structuring code and automating (container) build and deployment.
- Local development, potentially against a remote cluster, due to local resource constraints or the need to interface with remote services.
- Post-production testing and validation, such as shadowing traffic and canary testing (and the associated creation of metrics to support hypothesis testing).

Several tools and frameworks are emerging within this space, and they each offer various opinions and present trade-offs that you must consider.

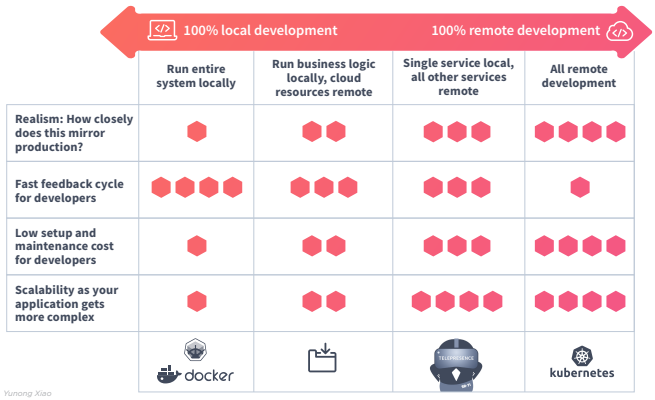
EMERGING DEVELOPER WORKFLOW FRAMEWORKS

There is a lot of activity in the space of Kubernetes developer workflow tooling. Shahidh K. Muhammed recently wrote an excellent Medium post, *Draft vs. Gitkube vs. Helm vs. Ksonnet vs. Metaparticle vs. Skaffold*, which offered a comparison of tools that help developers build and deploy their apps on Kubernetes (although he did miss Forge!). Matt Farina has also written a very useful blog post for engineers looking to understand application artifacts, package management, and deployment options within this space: *and Kubernetes: Where Helm Related Tools Sit*.

Learning from these sources is essential, but it is also often worth looking a little bit deeper into your development process itself, and then selecting appropriate tooling. Some questions to explore include:

- Do you want to develop and test services locally, or within the cluster?
  - Working locally has many advantages. I'm definitely an advocate of techniques such as BDD and TDD that promote rapid code and test cycles, and these approaches do require a workflow and platform that enable fast development iterations. There is also no denying that there are many benefits — such as iteration speed and the limited reliance on third-party services — that come from being able to work on a standalone local instance of a service during development. Finally, there is an argument that your system may not be adhering to the architectural best practices of high cohesion and loose coupling if you find that you cannot work on a service locally in isolation.
  - Some teams want to maintain minimal (thin client) development environments locally and manage all of their tooling in the cloud. The rise in popularity of cloud-based IDEs, such as AWS Cloud9 and Google CloudShell Editor, can be seen as a validation of this. Although this option typically increases your reliance on a third-party cloud vendor, the near-unlimited resources of the cloud mean that you can spin up a lot more hardware and test against real cloud services and data stores.
  - Using local/remote container development and debugging tools like Telepresence and Squash allows you to implement a hybrid approach where the majority of your system (or perhaps specific environments, like staging and pre-live) can be hosted remotely in a Kubernetes cluster, but you can code and diagnose using all of your locally installed tooling.

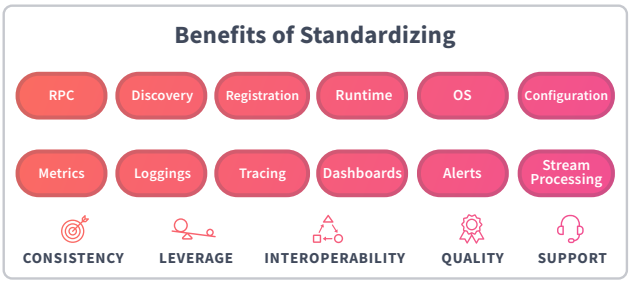
Development Environments for Kubernetes



- Do you have an opinion on code repository structure?
  - Using a monorepo can bring many benefits and challenges. Coordination of integration and testing across services is generally easier, and so is service dependency management. For example, developer workflow tooling such as Forge can automatically re-deploy dependent services when a code change is made to another related service. However, one of the challenges associated with using a monorepo is developing the workflow discipline to avoid code "merge hell" across service boundaries.
  - The multi-repo VCS option also has pros and cons. There can be clearer ownership, and it is often easier to initialize and orchestrate services for local running and

debugging. However, ensuring code-level standardization (and understandability) across repos can be challenging, as can managing integration and coordinating deployment to environments. Consumer-driven contract tooling such as Pact and Spring Cloud Contract provide options for testing integration at the interface level, and frameworks like Helm (and Draft for a slicker developer experience) and Ksonnet can be used to manage service dependencies across your system.

- Do you want to implement "guide rails" for your development teams?
  - Larger teams and enterprises often want to provide comprehensive guide rails for development teams; these constrain the workflow and toolset being used. Doing this has many advantages, such as the reduction of friction when moving engineers across projects, and the creation of integrated debug tooling and auditing is easier. The key trade-off is the limited flexibility associated with the establishment of workflows required for exceptional circumstances, such as when a project requires a custom build and deployment or differing test tooling. Red Hat's OpenShift and Pivotal Cloud Foundry offer PaaS-es that are popular within many enterprise organizations.
  - Startups and small/medium enterprises (SMEs) may instead value team independence, where each team chooses the most appropriate workflow and developer tooling for them. My colleague, Rafael Schloming, has spoken about the associated benefits and challenges at QCon San Francisco: *Patterns for Microservice Developer Workflows and Deployment*. Teams embracing this approach often operate a Kubernetes cluster via a cloud vendor, such as Google's GKE or Azure's AKS, and utilize a combination of vendor services and open-source tooling.
- A hybrid approach, such as that espoused by Netflix, is to provide a centralized platform team and approved/managed tooling, but allow any service team the freedom to implement their own workflow and associated tooling that they will also have the responsibility for managing. My summary of Yunong Xiao's QCon New York talk provides more insight to the ideas: *The "Paved Road" PaaS for Microservices at Netflix*. This hybrid approach is the style we favor at Datawire, and we are building open-source tooling to support this.



- How comfortable (and capable) is the team in regard to automation?
  - One of the many benefits of platforms like Kubernetes is the API-driven approach to deploying and operating applications. Teams that are comfortable with automating workflows with tooling and APIs can take advantage of increased release velocity and stability via automated testing in production through the use of traffic shadowing and canary testing (and the associated, observability, monitoring, and logging tooling).
  - Using container technology offers many other benefits in addition to the provision of management APIs. However, if existing workflows, practices, and tooling are kept while embracing this new technology, then limited benefits may be seen.

CONCLUSIONS

This article has provided several questions that you and your team must ask when adopting Kubernetes as your platform of choice. Kubernetes and container technology offer fantastic opportunity, but to fully take advantage of this, you will most likely need to change your workflow. Every software development organization needs a platform and associated workflow and developer experience, but the key question is: *How much of this do you want to build yourself?* Any technical leader will benefit from understanding the value proposition of PaaS and the trade-offs and benefits of assembling key components of a platform yourself.

DANIEL BRYANT works as an Independent Technical Consultant and Product Architect at Datawire. He currently specializes in enabling Continuous Delivery within organizations through the identification of value streams, creation of build pipelines, and implementation of effective testing strategies. Daniel's technical expertise focuses on DevOps tooling, cloud/container platforms, and microservice implementations. He is also a Java Champion, contributes to several open-source projects, writes for InfoQ, O'Reilly, and Voxxed, and regularly presents at international conferences such as OSCON, QCon, and JavaOne.

Oracle's Commitment to Containers and Open Source

Sure, you have heard about Amazon Web Services, Microsoft Azure and Google Cloud, but have you considered Oracle Cloud yet? Oracle's IaaS, PaaS, and SaaS portfolio includes offerings in ERP, HCM, IoT, Blockchain, AI Platform, and of course, Containers and Container Pipelines.

Oracle Cloud is pleased to launch Oracle Container Engine for Kubernetes (OKE) and Registry, which allow developers to deploy and run Docker containers on Kubernetes at scale, and leverage enterprise application management capabilities like automated release pipelines, rolling releases and rollbacks, elastic scaling, and self-healing. Developers can get started quickly with containers, and DevOps teams can gain visibility and control for Kubernetes environments.

To support DevOps and application development, Oracle Container Engine and Registry can interoperate with any Continuous Integration/Continuous Deployment (CI/CD) toolchain. In addition, Oracle offers Container Pipelines (Wercker), which allows you to create multi-stage multi-branch CI/CD workflows with ease.

Oracle is a platinum member of the Cloud Native Computing Foundation (CNCF) and contributes to community projects that include Docker, Kubernetes, and Terraform. Containers are so important that Oracle's flagship products — Oracle Java, MySQL, Oracle Linux, and others — which are available via the Docker store.

To take a free trial of Oracle Container Engine for Kubernetes and Registry, please reach us at kubernetes-trial_us@oracle.com.

WRITTEN BY AKSHAI PARTHASARATHY
PRINCIPAL DEVELOPER MARKETING DIRECTOR - ORACLE CLOUD

PARTNER SPOTLIGHT

Oracle Container Engine (OKE) and Registry

Container Native, Developer Friendly, and Enterprise Ready Kubernetes

ORACLE

CATEGORY

End-To-End Container Lifecycle Management

PRODUCT PROFILE

Oracle makes Kubernetes easy. With Oracle Cloud, you leverage Kubernetes-certified management, which is container native, developer friendly, and enterprise ready. We can take care of your container infrastructure, so you can work on more important tasks, like building and operating your container-based applications.

Oracle Container Engine and Registry are built with open source technologies and put developers first. We use the Docker-based runtime, and standard, upstream Kubernetes code with no proprietary addons. The Registry is fully Docker v2 compatible and allows you to store and manage private images.

Oracle Container Engine includes built-in cluster addons, such as Kubernetes Dashboard, DNS, and Helm. You get SSH access to your worker nodes, while leveraging a fully-managed and highly-available IaaS which can be provisioned on bare metal or virtual machines.

OPEN SOURCE?

Yes

FEATURES

- Easily deploy and manage container-based apps
- Store and share container images in highly-available private and public repositories
- Use bare metal servers, VMs, persistent storage, load balancing, multiple availability domains, and much more
- Exercise full REST API and CLI, along with Kubernetes Dashboard, DNS, and Helm
- Leverage Container Pipelines (Wercker) for complex CI/CD workflows and specific automation tasks
- You can find out more at the Oracle Developers Portal

WEBSITE developer.oracle.com **TWITTER** @oracledevs **BLOG** blogs.oracle.com/developers

Best Practices for Multi-Cloud Kubernetes

BY JIM BUGWADIA
FOUNDER AND CEO, NIRMATA

QUICK VIEW

01. Containers and Kubernetes are building blocks that enable application portability across public and private cloud providers.

02. Kubernetes has become the new cloud operating system that allows enterprises to leverage multiple cloud providers and cloud-native best practices.

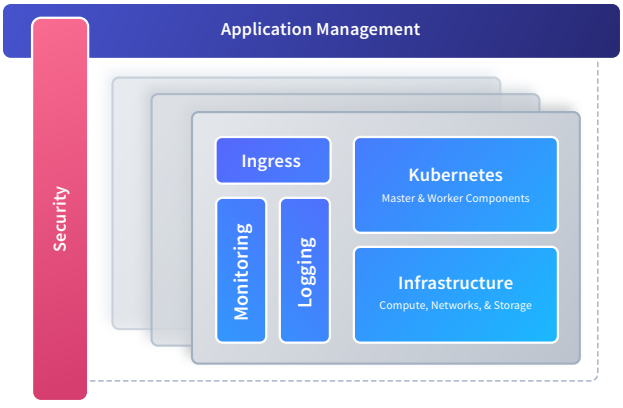
03. Enterprises should develop a multi-cloud strategy for managing infrastructure services, Kubernetes components and clusters, and define common layers for security and application management.

The 2018 State of the Cloud Survey shows that 81% of enterprises use multiple clouds. Public cloud computing services, and modern infrastructure platforms, enable agility at scale. As businesses seek to deliver value faster to their customers, it's no surprise that both public and private cloud adoption continue to grow at a healthy pace. In fact, according to the latest figures from IDC, worldwide server shipments increased 20.7% year-over-year to 2.7 million units in Q1 of 2018, and revenue rose 38.6%, the third consecutive quarter of double-digit growth!

Another exciting mega-trend is the emergence of containers as the best way to package and manage application components. Kubernetes has been widely accepted as the best way to deploy and operate containerized applications. And, one of the key value propositions of Kubernetes is that it can help normalize capabilities across cloud providers.

But with these advances also come new complexities. Containers address several DevOps challenges, but also introduce a new layer of abstraction that needs to be managed. Kubernetes addresses some of the operational challenges, but not all. And, Kubernetes is a distributed application that itself needs to be managed.

In this article, we will discuss the best practices and guidelines to address the key operations challenges for the successful deployment and operations of Kubernetes clusters across different cloud providers. The perspective we will take is that of an IT operations team building an enterprise Kubernetes strategy for multiple internal teams.



1. LEVERAGE BEST-OF-BREED INFRASTRUCTURE

All cloud providers offer storage and networking services, as do on-premises infrastructure vendors. A question that arises when considering multi-cloud strategies is whether to use each providers' capabilities, or an abstraction layer. While both approaches can work, it's always prudent to try and minimize abstraction layers and utilize the vendor-native approach. For example, rather than run an overlay network in AWS, it may be best to use the Kubernetes CNI (Container Network Interface) plugin from AWS that offers native

networking capabilities to Kubernetes. This approach also enables use of other services like security groups and IAM.

2. MANAGE YOUR OWN (UPSTREAM) KUBERNETES VERSIONS

Kubernetes is a fast-moving project, with new releases available every three months. A key decision to make is whether you want a vendor to test and curate Kubernetes releases for you or whether you want to allow your teams to directly use upstream releases.

As always, there are pros and cons to consider. Using a vendor-managed Kubernetes provides benefits of additional testing and validation. However, the Cloud Native Computing Foundation (CNCF) Kubernetes community itself has a mature development, test, and release process. The Kubernetes project is organized as a set of Special Interest Groups (SIGs), and the Release SIG is responsible for processes to ensure the quality and stability of each new release. The CNCF also provides a Kubernetes Software Conformance program for vendors to prove that their software is 100% compatible with the Kubernetes APIs.

Within an enterprise, it's best to use stable releases for production. However, some teams may want clusters with pre-GA features. The best bet is to provide teams with the flexibility of choosing multiple validated upstream releases, or trying newer versions as needed at their own risk.

3. STANDARDIZE CLUSTER DEPLOYMENTS VIA POLICIES

There are a several important decisions to make when installing a Kubernetes cluster. These include:

1. **Version**: the version of Kubernetes components to use.
2. **Networking**: the networking technology to use, configured via a CNI (Container Networking Interface) plugin.
3. **Storage**: the storage technology to use, configured via a CSI (Container Storage Interface) plugin.
4. **Ingress**: the Ingress Controller to use for load-balancer and reverse proxy of external requests to your application services.
5. **Monitoring**: an add-on for monitoring Kubernetes components and workloads in the cluster.
6. **Logging**: a centralized logging solution to collect, aggregate, and forward logs from Kubernetes components as well as application workloads in the cluster to a centralized logging system.
7. **Other Add-Ons**: other services that need to run as part of a cluster, like DNS and security components.

While it's possible to go through these decisions for each cluster install, it is more efficient to capture the cluster installation as a template or policy, which can be easily reused. Some examples of this could be a Terraform script or a Nirmata Cluster Policy. Once the cluster installation is automated, it can also be invoked as part of higher level workflows, like fulfilling self-service provisioning request from a service catalog.

4. PROVIDE END-TO-END SECURITY

There are a several items to consider for container and Kubernetes security, such as:

- **Image Scanning**: container images need to be scanned for vulnerabilities before they are run. This step can be implemented as part of the Continuous Delivery pipeline before images are allowed into an enterprise's private registry.
- **Image Provenance**: while image scanning checks vulnerabilities, image provenance ensures that only "trusted" images are allowed into a running cluster or environment.
- **Host & Cluster Scanning**: in addition to securing images, cluster nodes and also need to be scanned. Additionally, routinely running the Center for Internet Security (CIS) Benchmarks for Securing Kubernetes is a best practice.
- **Segmentation & Isolation**: even when multi-tenancy may not be a hard requirement, it's best to plan to share clusters across several heterogeneous workloads for increased efficiencies and greater cost savings. Kubernetes provides constructs for isolation (e.g. Namespaces and Network Policies) and for managing resource consumption (Resource Quotas).
- **Identity Management**: in a typical enterprise deployment, user identity is provided by a central directory. Regardless of where clusters are deployed, access user identity must be federated so that it can be easily controlled and applied in a consistent manner.

- **Access Controls**: while Kubernetes does not have the concept of a user, it provides rich controls for specifying roles and permissions. Clusters can leverage default roles or use custom role definitions that specify sets of permissions. It's important that all clusters within an enterprise have common definitions for these roles and a way to manage them across clusters.

While each of security practices can be applied separately, it makes sense to view these holistically and plan for a security strategy that works across multiple cloud providers. This can be achieved using security solutions like AquaSec, Twistlock, and others in conjunction with platforms like Nirmata, OpenShift, etc.

5. CENTRALIZE APPLICATION MANAGEMENT

As with security, managing applications on Kubernetes clusters requires a centralized and consistent approach. While Kubernetes offers a comprehensive set of constructs that can be used to define and operate applications, it does have a built-in concept of an application. This is actually a good thing as it enables flexibility in supporting different application types and allows different ways of building more opinionated application platforms on Kubernetes.

However, there are several common attributes and features that any Kubernetes application management platform must provide. The top concerns for centralized application management for Kubernetes workloads are discussed below.

APPLICATION MODELING & DEFINITION

Users need to define their application components and also compose applications from existing components. A core design philosophy in Kubernetes is its declarative nature, where users can define the desired state of the system. The Kubernetes workloads API offers several constructs to define the desired state of resources. For example, deployments can be used to model stateless workload components. These definitions are typically written as a set of YAML or JSON manifests. However, developers need to organize and manage these manifests, typically in a Version Control System (VCS) like Git.

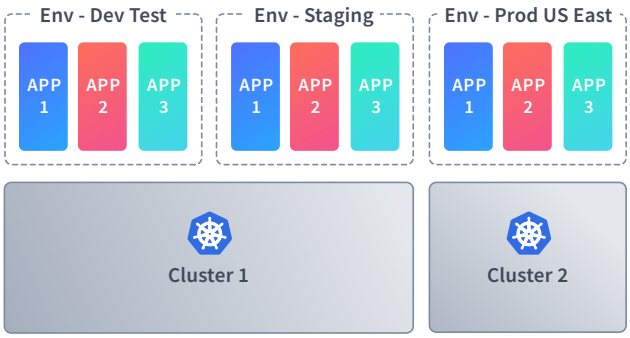
While developers will want to define and manage portions of the application manifests, other portions of these manifests specify operational policies and may be specific to runtime environments. These portions are best managed by operations teams. Hence, the right way to think of how an application manifests itself is as a pipeline that is composed dynamically before deployment and updates.

A Kubernetes project that helps with some of these challenges is Helm, a package manager for Kubernetes. It makes it easy to group, version, deploy, and update applications as Helm Charts.

Kubernetes application platforms must provide easy ways to model, organize, and construct applications manifests and Helm Charts, with proper separation of concerns between development and operational resources. The platform must also provide validation of the definitions to catch common errors as early as possible, along with easy ways to reuse application definitions.

ENVIRONMENTS — APPLICATION RUNTIME MANAGEMENT

Once applications are modeled and validated, they need to be deployed to clusters. However, the end goal is to reuse clusters across different workloads for greater efficiencies and increased cost savings. Hence, it's best to decouple application runtime environments from clusters and to apply common policies and controls to these environments.



Kubernetes allows creating virtual clusters using Namespaces and Network Policies. Kubernetes application platforms should make it easy to leverage these constructs and create environments with logical segmentation, isolation, and resource controls.

CHANGE MANAGEMENT

In many cases, runtime environments will live long lives and changes will need to be applied to these environments in a controlled manner. The changes may originate from a build system or from an upstream environment in the delivery pipeline.

Kubernetes application platforms need to offer integrations into CI/CD tools and monitor external repositories for chang-

es. Once changes are detected, they should be validated and then handled based on each environment's change management policies. Users should be review and accept changes or fully automate the update process.

APPLICATION MONITORING

Applications may be running in several environments and in different clusters. With regards to monitoring, it's important to have the means to separate the signal from the noise and focus on application instances. Hence, metrics, states, and events need to be correlated with application and runtime constructs. Kubernetes application platforms must offer integrated monitoring with automated granular tagging so that it's easy for users to drill-down and focus on application instances in any environment.

APPLICATION LOGGING

Similar to monitoring, logging data needs to be correlated with application definitions and runtime information and should be accessible for any application components. Kubernetes application platforms must be able to stream and aggregate logs from different running components. If centralized logging system are used, it's important to apply the necessary tags to be able to separate logs from different applications and environments and also manage access across teams and users.

ALERTING & NOTIFICATIONS

To manage service levels, it's essential to be able to define custom alerts based on any metric, state change, or condition. Once again, proper correlation is required to separate alerts that require immediate action from the rest. For example, if the same application deployment is running in several environments like dev-test, staging, and production, it is important to be able to define alerting rules that only trigger for production workloads. Kubernetes application platforms must be able to provide the ability to define and manage granular alerting rules that are environment and application aware.

REMOTE ACCESS

Cloud environments tend to be dynamic, and containers elevate the dynamic nature to a new level. Once problems are detected and reported, its essential to have a quick way to access the impacted components in the system. Kubernetes application platforms must provide a way to launch a shell into running containers, and to access container runtime details, without having to access cloud instances via VPN and SSH.

INCIDENT MANAGEMENT

In a Kubernetes application, it's possible that a container exits and is quickly restarted. The exit may be part of a normal workflow, like an upgrade, or may be due to an error like an out-of-memory condition. Kubernetes application platforms must be able to recognize failures and capture all details of the failure for offline troubleshooting and analysis.

SUMMARY

Containers and Kubernetes allow enterprises to leverage a common set of industry best practices for application operations and management across cloud providers. All major cloud providers, and all major application platforms, have committed support of Kubernetes. This includes Platform-as-a-Service (PaaS) solutions where developers provide code artifacts and the platforms does the rest, Container-as-a-Service (CaaS) solutions where developers provide container images and the platform does the rest, and Functions-as-a-Service (FaaS) solutions where developers simply provide functions and the platform does the rest. Kubernetes has become the new cloud-native operating system.

When developing a multi-cloud Kubernetes strategy, enterprises must consider how they wish to consume infrastructure services, manage Kubernetes component versions, design and manage Kubernetes clusters, define common layers of security, and application management.

JIM BUGWADIA is a founder at Nirmata, a SF Bay area startup that provides multi-cloud container services. Jim has been building large-scale distributed software systems for the last two decades: he started his career developing C++ software at Motorola for cellular network infrastructure, where his team launched the world's first CDMA network. Since then Jim has held several developer, architect, and leadership roles at companies like Cisco, Bell Labs, Trapeze Networks, and Pano Logic. In 2013 Jim founded Nirmata, a startup aiming to address Enterprise DevOps pain points by providing fully-automated multi-cloud management for microservices style applications deployed in containers.

HAVE CONTAINERS PEAKED YOUR INTEREST?

To say containers have made an impact in software development would be an understatement. More and more developers and organizations are looking to adopt containers to make development and testing easier. Containers are also a natural fit with microservices architectures that can improve deployment speed and software quality. Since the hype and excitement is showing no sign of slowing down, is it worth climbing the mountain? Let's look at what over 500 DZone survey respondents think.

Organizations have seen enormous benefits from adopting containers. **77% have reported faster deployments, and 75% noticed they could better scale applications**. Other benefits include modularity (64%), portability (55%), and environmental consistency (47%).

Of those who have adopted containers, 81% of respondents expected their jobs would be easier as a result, and 75% actually noticed their jobs were easier. Only 7% found that their jobs were more difficult, and 18% saw no change. It seems containers are indeed worth the hype, or **at the very least don't make things more difficult**.

As expected, **Kubernetes is the most considered orchestration tool among those who are considering containers** (57%), followed by Docker Swarm (34%), Amazon ECS (31%), and Google Kubernetes Engine (20%). Only 7% of organizations are not considering orchestration tools.

Several challenges are making organizations nervous about adopting containers. The two biggest by far are a lack of developer experience (71%) and re-factoring legacy applications (70%). Other concerns include application and network security (42%) and performance monitoring (40%).

Organizations and teams are moving fast on containers. **63% of organizations who have adopted containers did so in less than six months**, and of organizations who have not yet adopted containers, only 30% have spent over six months evaluating them.

A Deep Dive Into Cloud-Agnostic Container Deployments

BY STEFAN THORPE
CTO AT CAYLENT

QUICK VIEW

01. Learn about the changing landscape of container technology and its increasing usage.

02. Get a comparison between the two major players in container orchestration: Kubernetes and Docker Swarm.

03. Dive into getting the most out of Kubernetes through the use of Helm and charts for deployment.

Container technology has been evolving from its originally primitive perspective in the late 1970s to the Docker era which debuted in 2013 and now, it's safe to say we're now firmly ensconced in the age of Kubernetes. Since the deployment method's inception — and Docker popularizing the practice — containers have grown in renown and have dramatically enhanced the development landscape. Their usage has drastically improved the manners in which developers can implement distributed applications.

The focus on consistent container deployments across various platforms has made them increasingly embraced by enterprises of all shapes and sizes — especially as the latest in container orchestration's support efforts assist even the greenest developers through development to production and have led to more containers being deployed and managed than ever before. The urge for improved control has attracted various software options as answers to proper container orchestration.

Kubernetes and Docker Swarm remain the two major tools on the market and are used by prominent internet companies for container orchestration. Other players on the scene are Amazon Elastic Container Service (Amazon ECS), Shippable, Apache Mesos, Marathon, and Azure Container Service (AKS), to name a few.

KUBERNETES VS. DOCKER SWARM: MOST EFFECTIVE DEPLOYMENT METHOD

Container orchestration refers to the automated organization, linkage, and management of software containers. These concepts are conventional for most of the tools mentioned above. This article aims to deep dive into a comparison between the two dominating players. Below are the features availed by both Kubernetes and Docker Swarm.

- **Clustering**: For synchronized computing ability across multiple machines.
- **High availability**: Run code all at once in various locations.
- **Fault tolerance**: If a container fails, it can be relaunched automatically.
- **Secret management**: Share secrets safely between different hosts.

ADVANTAGES OF DOCKER SWARM

There are fundamental differences in the way Kubernetes and Docker Swarm operate, though, which pose advantages for one platform over the other for different end users. Here are some pros of Docker Swarm vs. Kubernetes.

EASY, FAST SETUP

It is easy to install and also configure Docker Swarm orchestration. You only need to deploy a node and request it to join a cluster. A swarm allows a node to join clusters either as a manager or a worker, thus providing more flexibility. For an idea of how simple this is, check out Creating a High-Availability Docker Swarm on AWS.

WORKS WITH EXISTING DOCKER APIS

The Swarm API derives most of its functionality from Docker itself. Kubernetes needs its own API, client, and YAML, which are all different from Docker's standards.

LOAD BALANCING

Docker Swarm provides automated load balancing via any node. In any network that permits connection of any container through any node, all containers within a cluster can join.

SHARING OF DATA VOLUMES

Docker Swarm simplifies local data volume sharing. Volumes (directories) can be generated individually or together with the containers, then divided among multiple containers.

DISADVANTAGES OF DOCKER SWARM

LIMITED FUNCTIONALITY

Docker Swarm performance revolves around the provisions of the Docker API. If the Docker API lacks a specific operation, it can't use Swarm.

LIMITED FAULT TOLERANCE

Docker Swarm has finite fault resistance.

NO BUILT-IN SCALABILITY

At the time of writing, scaling containers and the support infrastructure is achievable but very tricky to do.

UNSTABLE NETWORKING

Docker's underlying raft network has shown many signs of instability and the result is that many production environments are having issues. There's flaky networking underpinning everything.

ADVANTAGES OF KUBERNETES

SIMPLE SERVICE ASSEMBLY WITH PODS

Kubernetes designates container pods as services to permit the concept of load balancing. You can achieve each function by using a set of pods and policies to setup load balancing. This configuration does not use IP addresses.

HIGH SERVICE AVAILABILITY RETENTION

Kubernetes monitors the system and clusters progressively to upkeep service health and maintain its availability.

HIGH SCALABILITY

Gain the ability to build clusters across different location and providers, with built-in auto-scaling at the cloud and container levels.

HIGHLY EXTENSIBLE

Kubernetes was designed to be extremely flexible and support many plugins. This has allowed for very large community growth, and as such, a massive selection of tools and plugins. Here's a small curated collection of some of the more popular tools for 2018.

- **Elevated data sharing**: Kubernetes allows containers to share data within pods. It also allows external data volume managers to exchange data between pods.
- **Cloud integration**: Kubernetes is highly extensible and its cloud integrations are awesome. A load balancer service will deploy the matching managed service in GCP, AWS, or Azure. Many other platforms — Docker included — have also added Kubernetes support.

DISADVANTAGES OF KUBERNETES

POTENTIALLY OVERWHELMING TO INSTALL AND CONFIGURE MANUALLY

Kubernetes necessitates a set of manual configurations to tie its components to the Docker engine. It comes with unique installations for every operating system. Before installation, Kubernetes requires information like node IP addresses, their roles, and numbers. There are many tools available to simplify the install and config process, though.

A NEW LEVEL OF COMPLEX

Kubernetes is considered relatively white-box, i.e. you can get a lot more out of it, but you really need to have a deep understanding of what makes Kubernetes tick to achieve this. The platform is not designed for novices and the faint of heart to navigate.

Throughout the pros and cons of Docker Swarm, you can note that Swarm's focus is on the ease of adoption and integration with Docker. Kubernetes, on the other hand, stands open and flexible. The K8s platform delivers top support for highly complex demands. This versatility is behind the preference for Kubernetes by many high-profile internet companies.

THREE MAJOR KUBERNETES OBJECT MANAGEMENT TECHNIQUES

There are three Kubernetes deployment techniques to make your work plan faster and smoother. Among the three, you should choose only one method at a time to manage a Kubernetes object. Use of multiple techniques can cause unstable performance. Here are the three types of Kubernetes object management techniques that you can use.

IMPERATIVE COMMANDS

These are "easier-to-use" commands, and they're simple to recall over and over again. The commands deliver a one-step change to a cluster, and you work directly on live objects. Type operations into the `kubectl` command line as flags or arguments. This

technique does not provide any history of earlier configuration, though. Hence, previous commands can't be used in change review processes. Imperative commands are ideally applicable in development projects.

IMPERATIVE OBJECT CONFIGURATION

With imperative object configuration, the `kubectl` commands mainly focus on an operation like `create`, `replace`, and others. It also defines an optional flag and at least one file name. Alongside the file name, it gives a complete definition of an object in YAML or JSON format. This technique is preferred by many developers, as it allows commands storage in a source control system.

It is also useful in change review processes, as commands can be integrated. Furthermore, configurations can serve as templates for new objects.

Just as a captain steers a ship, Helm offers greater control for Kubernetes clusters. It can be thought of as a package manager that offers greater flexibility in the creation of Kubernetes definition YAMLs through a templating language and structure.

DECLARATIVE OBJECTS CONFIGURATION

Declarative objects configuration allows you to operate on object setup files stored locally. However, you can't specify the operations to carry on the files. This mandate rests with `kubectl`, and it can automatically create, update, or delete operations per object. The `kubectl` functionality enables working directories where you will need different processes for different objects.

LEVERAGE KUBERNETES PERFORMANCE WITH HELM CHARTS

WHAT ARE HELM CHARTS?

Just as a captain steers a ship, Helm offers greater control for Kubernetes clusters. It can be thought of as a package manager that offers greater flexibility in the creation of Kubernetes definition YAMLs through a templating language and structure. There are both client-side and the server-side segments of Helm for Kubernetes (Helm is the package manager and the Tiller is the in-cluster component that works with the K8s API server). The client collaborates with the server to effect changes to the Kubernetes cluster.

In a standard Helm sequence, the user initiates the Helm `install` command. The Tiller server responds by setting up the relevant install package into the Kubernetes cluster. Such packages are known as charts, and they offer a convenient approach to distribute and install packages.

Helm fulfills the need to efficiently and reliably provision Kubernetes orchestration without all the individual configuration involved. Charts are the software equivalent of development templates. Therefore, you can quickly achieve installation, updates, and removal without any fuss. Helm charts will even help you override the Kubernetes disadvantages as addressed earlier. Your team can concentrate its focus on developing applications and improving productivity instead of deploying dev-test environments. Helm takes care of all of that for you.

In addition to these benefits, your team doesn't need to preserve service tickets during Kubernetes deployments and you also eliminate the complexity of maintaining an App Catalogue too. GitHub/Kubernetes/Helm has a huge repository of communal charts in storage to be used in a matter of clicks. Among the most reliable Helm charts you could consider are ones for MySQL and MariaDB, MongoDB, and WordPress.

CONCLUSION

In conclusion, if containers are discouraging you at all, use Kubernetes to scale them. Kubernetes is highly extensible and comes with a lot of plugins to support your productivity. Furthermore, deploy all your containers and clusters using Helm and Helm charts to further boost efficiency.

STEFAN THORPE is CTO for the DevOps-as-a-Service consultancy management platform Caylent. He is an IT professional and DevOps evangelist with 20+ years of management and hands-on experience providing technical solutions to support strategic business objectives. Stefan leads and fine-tunes development teams to optimize for innovation and growth.

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'll 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

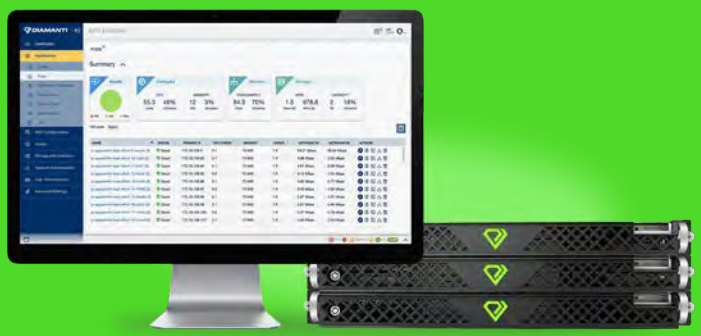
CONTROL

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

DIAMANTI

www.diamanti.com

certified kubernetes 1.8 · docker · redhat TECHNOLOGY PARTNER · CRN 10 Coolest Cloud Storage Startups · Gartner Cool Vendor 2017 · database TREND-SETTING PRODUCT IN DATA MANAGEMENT 2018



Docker on Windows: Powering the 5 Biggest Changes in IT

BY ELTON STONEMAN
DEVELOPER ADVOCATE AT DOCKER

QUICK VIEW

01. Containers are helping Windows organizations meet the biggest challenges in the IT industry — from migrating to the cloud to modernizing traditional applications.

02. Container platforms should bring consistency to all containerized applications — old and new.

03. Docker on Windows provides high agility, low cost, and the flexibility to run the same apps in a hybrid cloud or multi-cloud scenario.

Docker runs on Windows Server 2016 and Windows 10, powering Windows apps in lightweight, portable containers. You can take existing applications and run them in Windows Docker containers with no code changes, and you can take new applications and simplify the whole CI/CD process using Docker. Windows containers come with production support from Microsoft and Docker, so you can confidently make the move away from VMs to containers.

This article explains how Docker works on Windows, and it looks at how containers are helping Windows organizations meet the five biggest challenges in the IT industry — from migrating to the cloud to modernizing traditional applications and driving new innovation.

DOCKER ON WINDOWS

Docker started in Linux, using core features in the kernel for isolating workloads, and making them easy to use. Microsoft added container support to Windows in 2016, partnering with Docker engineers to bring the same user experience to Windows.

All the Docker concepts work in the same in Windows and Linux: you package your app into a Docker image using a Dockerfile, distribute the app by pushing the image to a registry, and run your app by pulling the image and starting a container. The container uses the underlying operating system to run processes so containers are fast and efficient, as well as being portable and easy to secure.

Windows containers are based on images that Microsoft provides and maintains, releasing new versions each month with all the latest Windows updates. Nano Server is a lightweight server option for new applications using technologies like .NET Core, NodeJS, and Go. Windows Server Core is a full server runtime with support for existing workloads, including .NET Framework and Java apps.

The Docker experience on Windows is the same as on Linux: you install Docker on your Windows server, and you don't need to install any other software. Any apps you run are by Docker, and the application containers have all the dependencies they need. Containers use resources from the host server, but they're isolated from each other. You can run different versions of the .NET Framework in different containers on the same server with no interference.

You can learn how to Dockerize Windows applications in two video series on YouTube. Modernizing .NET Apps With Docker for IT Pros is aimed at operations teams. It shows you how to deploy existing apps to containers using artifacts like MSIs, and how to modernize the delivery and management of apps by integrating with the Docker platform without changing code.

Modernizing .NET Apps With Docker for Developers shows how to modernize the architecture of an existing application, break down a monolithic design into smaller services, run them in containers, and plug them together using Docker.

The support for new and old Windows applications in containers is an enabler for meeting the major challenges facing enterprise IT.

1. CLOUD MIGRATION

Running in the cloud should bring agility, flexibility, and cost savings. To get those benefits for existing Windows apps, you typically had to choose between two approaches: Infrastructure-as-a-Service (IaaS), and Platform-as-a-Service (PaaS).

IaaS means renting Windows VMs and deploying your apps in the same way as the datacenter. It means you can re-use your existing scripts and processes, but you take the inefficiencies of running virtual machines into the cloud, which means you have a whole suite of VMs to monitor, manage, and update. You can't scale up quickly because VMs take minutes to start, and you can't run apps with higher density so you're unlikely to see significant cost benefits.

PaaS means using the full product suite of your cloud provider and matching the products to the features your app needs. In Azure that could mean using App Services, API Management, SQL Azure, and Service Bus queues. It's likely to give you high agility, and using shared services means you should save on cost - but it's going to take a project for every app you want to migrate. For each app, you'll need to design a new architecture, and in many cases, you'll need to change code.

Docker on Windows gives you a new option that combines the best of IaaS and PaaS: move your apps to containers first, and then run your containers in the cloud. It's a much simpler option that uses your existing deployment artifacts without changing code, and it gives you high agility, low cost, and the flexibility to run the same apps in a hybrid cloud or multi-cloud scenario.

2. CLOUD-NATIVE APPS

Cloud-native applications are container-packaged, are dynamically managed, and use microservice architectures. Docker brings cloud-native approaches to building new Windows apps. You can deliver a project using a modern technology stack like .NET Core and NodeJS.

You can run those apps on Nano Server containers — which are smaller and faster than full Windows Server Core containers — but they run on Windows Server, so you don't need your team to become Linux experts to start building cloud-native apps.

You can also integrate your Windows microservices with fantastic open-source projects which already run on Docker to add features and extend capabilities — software like NATS, which is the message queue project from the Cloud Native Computing Foundation (CNCF). It's enterprise-grade software which comes packaged in a Windows Docker image, so you can drop it right into your solution with no complex configuration.

Cloud-native applications are container-packaged, are dynamically managed, and use microservice architectures. Docker brings cloud-native approaches to building new Windows apps. You can deliver a project using a modern technology stack like .NET Core and NodeJS.

3. MODERNIZING TRADITIONAL APPS

New cloud-native apps are an important part of innovating for the future, but most enterprises already have a much larger landscape of traditional applications. These are apps which typically have a monolithic architecture and manual deployment steps, which are complex and time-consuming to develop and test, and fragile to deploy.

Many Windows organizations are also managing apps across a range of operating systems - including Windows Server 2003 and 2008. It's hard to maintain an application landscape that is running on diverse operating systems, which each have different toolsets and different capacities for automation.

The Docker container platform brings consistency to all containerized applications, old and new. The Windows

Server Core Docker image — which Microsoft maintains — has support for older application platforms, including .NET 2.0 and 32-bit apps. You can take a ten-year-old application and run it in a Windows container, without even having the original source code.

This is letting organizations migrate off older operating systems and move to a model where every app runs as a container. You can even run a hybrid Docker swarm cluster, using a mixture of Linux and Windows servers. Then you can run brand-new microservices apps in Linux containers alongside traditional .NET Framework apps on Windows containers on the same cluster, and use one set of tools to package, deploy, and manage all those apps.

4. INNOVATION

Technical innovation doesn't end with cloud-native apps. Trends like IoT, machine learning, and serverless functions are all coming closer to mainstream, and they're all made easier and more manageable by Docker.

One of the biggest concerns in IoT projects is how to manage the software running on the devices, and how to distribute and safely deploy updates. When you run your device software in containers, you can use the distribution mechanism built into Docker to deploy new updates without writing custom code.

Machine learning frameworks like TensorFlow tend to have a large list of dependencies, but running them in Docker makes it trivial to get started. You can even package your own Docker container image with your trained models and make them available publicly on Docker Hub or privately inside your organization. Then, anyone can start taking advantage of your trained models just by running containers.

Serverless is all about containers. Developers write code and the serverless framework takes care of packaging the code into a Docker image, and then it runs containers to execute the code when a trigger comes in, like an HTTP request or a message on a queue. Serverless isn't just for cloud deployments, and there are great open-source projects like OpenWhisk, Nuclio, Fn, and OpenFaas that are powered by containers. You can run serverless functions on the same Docker cluster as your microservices and your traditional apps.

Gloo is a recently released project that lets you tie together portable serverless functions with proprietary serverless functions, microservices, and traditional apps. Gloo is a function gateway that can be extended with plugins and runs on Kubernetes. It is designed for microservice, monolithic, and serverless applications. For example, an enterprise developer could modernize a traditional application by containerizing it with Docker Enterprise Edition and then, using Gloo, start to add functionalities to it using microservices, portable functions using Fn, and proprietary functions using AWS Lambda.

5. DEVOPS

The last big challenge facing enterprise IT is about cultural change and the move to DevOps, which should bring faster deployments and higher quality software. DevOps is usually positioned as people and process change, but it's hard to make big changes unless you underpin them with technology change.

Moving to Docker helps drive the change to DevOps, even for teams currently building and deploying Windows applications using older technologies. Everything in the Docker container platform is automated, which gets you fast delivery and reliable deployments and rollbacks. And the key artifacts — Dockerfiles and Docker Compose files — become the joint responsibility of developers and IT Pros.

Having teams working on the same technology and speaking the same language is a great way to break down barriers. People are excited by Docker, too. It's an interesting, powerful new technology which is easy to get started with and can improve practices from development to production. Teams adopting the Docker container platform are enthusiastic and that helps drive big changes like the move to DevOps.

SUMMING UP

Docker on Windows is today's technology, and it's an enabler for meeting the real challenges that enterprises face. You can get started with Docker very easily by packaging up your existing applications or adding Docker support to new projects. Then you can run your apps in containers which are the same in every environment, right up to production, where you have support from Microsoft and Docker.

ELTON STONEMAN is a Developer Advocate at Docker, focused on spreading the word to the Microsoft community. He's a Microsoft MVP and has been building software on Windows for a very long time.

DIVING DEEPER

INTO CONTAINERS

#CONTAINERS TWITTER ACCOUNTS

 @proudboffin

 @nathankpeck

 @danielbryantuk

 @christianposta

 @kelseyhightower

 @ThoHeller

 @nigelpoulton

 @spboyer

 @ericschabell

 @anders_wallgren

CONTAINERS 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.

CONTAINERS REFCARDZ

Blockchain & Distributed Ledger Technology for Documents

Learn how to reduce the complexity of your code when needing digital signing and verification functionality, assure the integrity and authenticity of a document, and automate your document-oriented workflows.

Getting Started With Docker

In what seems like an instant, Docker has become the benchmark for organizations to automate infrastructure, isolate applications, maintain consistency, and improve resource utilizations. In this Refcard, learn how to run a container, explore several useful commands, and discover how to create local Docker machines.

Understanding Cloud Computing

This Refcard will walk you through the basics, from where exactly clouds are located, to deciphering the differences between cloud service types (IaaS, PaaS, SaaS), and the key benefits you can expect to receive.

CONTAINERS PODCASTS

Containerization (And Why Everyone Loves Docker So Much)

Learn about containerization software, what the appeal of containers is, who uses containerization and at what scale, and more.

Kubernetes, Docker, and the Distributed Operating System

Kelsey Hightower discusses how Kubernetes works, why it is important to an average developer, and how software engineering is going to look in the next 5-10 years.

Containerization and Container Orchestration

In episode 3 of the Pod as a Service podcast, learn about all things containers, including prominent technologies and their benefits, considerations, orchestration, and management.

CONTAINERS RESOURCES

30 Essential Container Technology Tools & Resources

Gauge your containerization options with these resources about container architecture, cluster management, storage, and more.

The Ultimate List of Containerization Tools

Learn about various containerization tools, from container management platforms to container automation tools to system-level virtualization technologies.

Manging Compute Resources for Containers

In this Kubernetes documentation, learn about resource types, how pods with resource limits are on, how to monitor compute resource usage, and more.

Implementing Cloud-Native Enterprise Applications With Open-Source Software

BY IMESH GUNARATNE
FORMER SOLUTIONS ARCHITECT AT WSO2

QUICK VIEW

01. The most important elements of orchestrating and deploying containers are security and the ability to maintain hybrid environments.

02. The biggest change in containers has been the growth and adoption of Kubernetes and companies' desire to move out of dev/test directly into production.

03. Serverless and function-as-a-service (FaaS) are the future of containers.

Linux container technologies such as kernel namespaces, cgroups, chroot, AppArmor, and SELinux policies have been in development since 1979. In 2013, an organization called dotCloud built a complete ecosystem for making Linux containers extremely usable by introducing a better interface, a REST API, a CLI, and a layered container image format — and called it Docker. This exploded the interest in Linux containers and began to revolutionize the way software is being designed and deployed to achieve optimal infrastructure resource usage, scalability, and maintenance. Google, who was contributing to cgroups, LMCTFY, and other related Linux kernel features, initiated the Kubernetes project in 2014. With their experience of running containers at scale over a decade, Google was well-positioned to introduce this open-source container cluster manager. This made the next major milestone of container technologies, which lead to the inception of newer architectural patterns, distributed service management frameworks, serverless technologies, observability tools, and, most importantly, the Cloud Native Computing Foundation (CNCF).

Today, enterprises are rapidly adopting these technologies for implementing production systems using containers at different scales. CNCF is now taking the lead on standardizing the cloud-native technology stack by categorizing the spectrum, defining specifications, improving interoperability, allowing technology leaders to collaborate, and building an open-source, vendor-neutral ecosystem that is portable to public, private, and hybrid clouds.

WHAT IS CLOUD-NATIVE?

"Cloud-native" is nothing new, but it's a new term to define the concepts used for building and running applications on any cloud platform without having to change the application code. This approach may involve adopting microservices architecture, containerizing application components, and dynamically orchestrating containers using a cloud-agnostic container cluster manager — including tools for managing services and observing the deployments.

A REFERENCE ARCHITECTURE FOR IMPLEMENTING CLOUD-NATIVE ENTERPRISE SYSTEMS

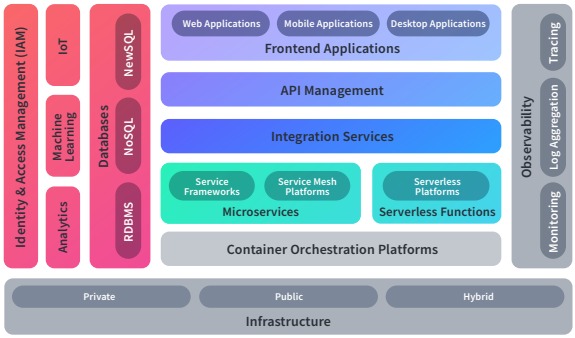


Figure 1: A reference architecture for cloud-native enterprise systems.

The above diagram illustrates a reference architecture for implementing cloud-native enterprise systems using container-based technologies. According to the current state of the ecosystem, microservices, serverless functions, integration services, and managed APIs are ready to be deployed on containers handling production workloads. Those components can be deployed on private, public, and hybrid cloud environments using a cloud-agnostic container orchestrator. Nevertheless, stateful, complex distributed systems such

as database management systems, analytics platforms, message brokers, and business process servers may need more maturity at the container cluster manager and at the application level for natively supporting completely automated deployments.

CONTAINER ORCHESTRATION

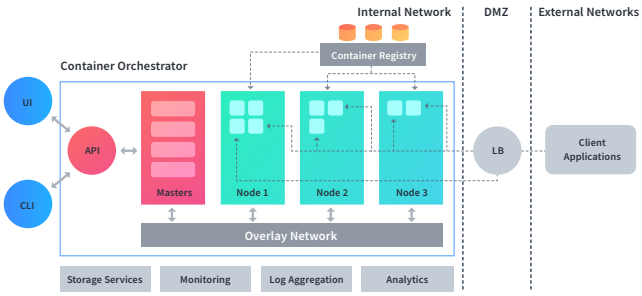


Figure 2: A reference architecture for a container cluster manager.

Today, Kubernetes is considered to be the most compelling open-source platform for orchestrating containers. It is now at Version 1.10 and currently being used in production to manage hundreds (if not thousands) of container hosts running millions of containers. At its core, it provides features for container grouping, self-healing, service discovery, load balancing, autoscaling, running daemons, managing stateful components, managing configurations, credentials, and persistent volumes. Moreover, it provides extension points to implement custom resources and controllers for advanced orchestration requirements needed by complex systems such as big data analytics, databases, and message brokers. Kubernetes can be installed on any virtualization platform without requiring any special tools, and can be spawned on AWS, Google Cloud, Azure, IBM Cloud, and Oracle Cloud as managed services by only paying for the virtual machines required for running the workloads.

Alternatively, organizations can also use RedHat OpenShift, Mesosphere DC/OS, Hashicorp Nomad, and Docker Swarm for container orchestration. OpenShift is a Kubernetes distribution which provides additional application lifecycle management and security features. It is available as CentOS-based open-source distribution and RHEL-based enterprise distribution. DC/OS has been implemented using Apache Mesos, Marathon, and Metronome by Mesosphere, and it's specifically optimized for running big data analytics systems such as Apache Spark, Cassandra, Kafka, HDFS, etc. It also has an open-source distribution and an enterprise edition. Some key features such as user management and credential management are missing in the open-source version. Docker Swarm is another container cluster manager implemented by Docker which is bundled into the Docker runtime. By design, it integrates well with Docker and provides a simpler deployment model compared to other systems. Nevertheless, Swarm has not been adopted much in the industry in comparison to other cluster managers.

MICROSERVICES FOR BETTER AGILITY, SPEED, AND COST

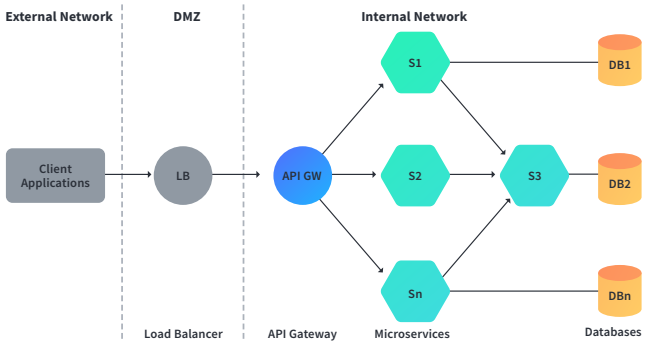


Figure 3: A reference implementation of microservices architecture.

Microservices architectural style proposes that an application should be implemented as a collection of independently manageable, lightweight services — in contrast to its opposite, monolith architecture, in which an application is implemented as a single unit. The microservices approach allows each service to have a single focus, loosely coupled, lightweight, highly scalable modular architecture to achieve better resource usages, optimized deployment models, fewer maintenance costs, and faster delivery times. Such services can be implemented in any programming language that supports REST- and RPC-based services.

Spring Boot, Dropwizard, and Spark are the most widely used open-source microservices frameworks for Java. Out of these, Spring Boot provides the advantages of Spring's dependency injection, data access, batch processing, security, and integration inclinations. Services that are mission-critical and require the merest latencies can be implemented with Golang using Echo, Iris, or Go kit. Otherwise, if the developers' preference is more toward JavaScript Express, Feathers and LoopBack would be striking options. LoopBack stands out for exposing CRUD APIs with OAuth2 security with a few lines of code. Besides the above, Flask, Sanic, and Tornado would be attractive alternatives for Python developers.

OPTIMAL GOVERNANCE FOR MICROSERVICES WITH SERVICE MESH ARCHITECTURE

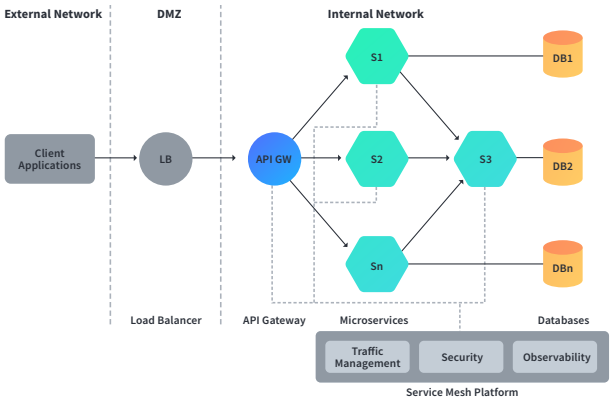


Figure 4: Usage of a service mesh in microservices architecture.

Once a large system is decomposed into hundreds and thousands of smaller services according to microservices architecture (MSA), managing inter-service communications, service identity, authorization, monitoring, logging, and obtaining telemetry data might become challenging tasks. Last year, IBM, Google, and Lyft joined together to implement a solution for this problem with the Istio project, by combining IBM's Amalgam8 project, Google's Service Control implementation, and Lyft's Envoy proxy. Istio might be today's most comprehensive service mesh platform that can provide traffic management, security, policy enforcement, and telemetry data extraction at the application deployment time without having to implement the code in the services. Istio does this by injecting Envoy proxy into the service pods and dynamically intercepting the communication between services for controlling traffic using a central management layer. Service security can be managed with Istio using Mutual TLS. Role-Based Access Control (RBAC) monitoring is provided with Prometheus, Grafana, Heapster, and native GCP and AWS monitoring tools, and distributed tracing is provided with Zipkin and Jaeger. Due to the popularity of Istio, NGINX implemented another service mesh based on Istio called nginMesh by using NGINX as the sidecar proxy.

Linkerd is another popular open-source service mesh platform implemented using Finagle and Netty (by Buoyant and later donated to CNCF). Linkerd uses proxy daemons on each container host for intercepting inter-service communication unlike proxy sidecars in Istio. This model requires services to route requests specifically to the proxies using additional configurations. Buoyant has improved this architecture and produced Conduit, targeting Kubernetes by incorporating a Kubernetes object injection model similar to Istio.

USING SERVERLESS FUNCTIONS FOR EVENT-DRIVEN EXECUTIONS

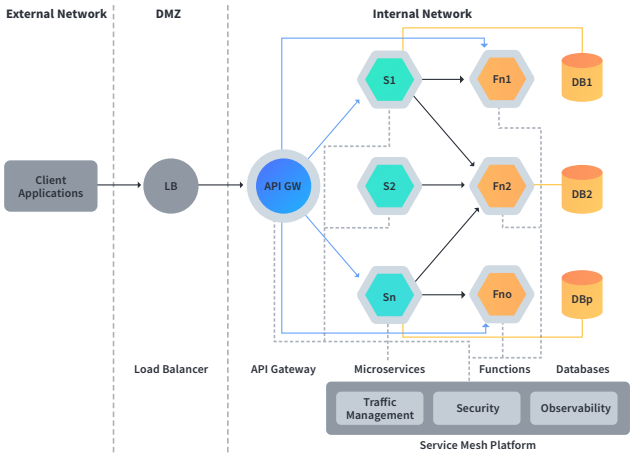


Figure 5: Usage of serverless functions in microservices architecture.

One of the key aspects of MSA is its ability to reduce the infrastructure resource usage by allocating resources at a granular level according to the actual service resource requirements. Nevertheless, at any given time, it would need to run at least one container per service. The serverless architecture attempts to further optimize this by decomposing the deployable unit up to functions and running functions only when needed. Serverless functions became popular when AWS introduced the AWS Lambda platform. Today, almost all public cloud vendors provide a similar offering, such as Google Cloud Functions, Azure Functions, and IBM Cloud Functions. Most of these platforms support programming languages such as Node.js, Java, and Python — except for Google Functions, which only supports Node.js. On the above public cloud offerings, users only get billed for the number of function invocations, considering the amount of infrastructure resources and time required for executing each.

Modern enterprises are now adopting microservices architecture for implementing highly scalable, cloud-agnostic applications that achieve better agility, speed, and lower cost.

Today, Apache OpenWhisk is one of the most widely used serverless frameworks for implementing on-premise serverless systems. It was initially developed by IBM and later donated to ASF for wider community adoption. OpenWhisk was designed using a highly extensible architecture to enable adding new languages and event triggers without much effort. Moreover, it supports creating a chain of functions for implementing a sequence of business operations. One of the key design decisions OpenWhisk has made for optimizing resource usage is to create containers on demand and preserve them for a given period of time.

Fission is another popular serverless platform specifically designed for Kubernetes. In contrast to OpenWhisk, Fission uses a configurable pool of containers for reducing the cold start time of functions and provides function composition capabilities. In terms of deployment, Fission can be integrated with Istio for incorporating service mesh features and function autoscaling based on Kubernetes Horizontal Pod Autoscalers. Kubeless is a similar platform developed by Bitnami for hosting serverless functions on Kubernetes. It uses a custom Kubernetes resource for deploying code, and as a result, functions can be managed using the standard Kubernetes CLI.

INTEGRATION SERVICES FOR API COMPOSITION

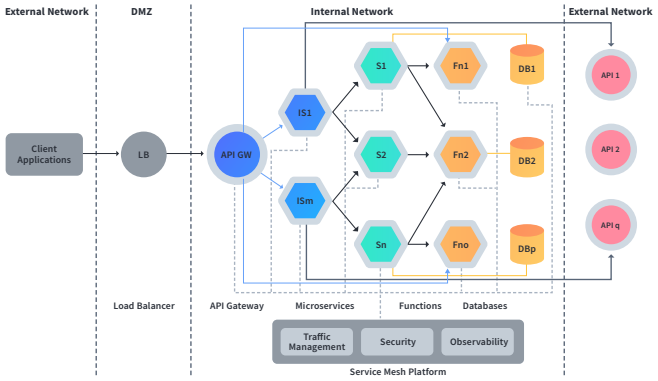


Figure 6: Usage of integration services in microservices architecture.

Implementing integrations can be achieved through microservices using standard programming constructs. If the system grows over time, it would require a considerable amount of effort and repetitive work by introducing a considerable amount of integrations. Ballerina is a new programming language purposely built by WSO2 to fill this gap in the container native ecosystem. It provides integration constructs and connectors for implementing distributed system integrations with distributed transactions, reliable messaging, stream processing, and workflows. It provides native support for Kubernetes, Prometheus, and Jaeger.

DISTRIBUTED OBSERVABILITY TOOLS FOR IMMEASURABLE INSIGHTS

Observability mainly divides into three categories: monitoring, logging, and tracing. Monitoring involves observing the health of the applications, including socket status, resource usage, request counts, latencies, etc., and generating alerts for the operations teams to take actions on actual system failures (excluding false positives). Prometheus is one of the most widely used tools available today for monitoring distributed systems. It was initially developed as an open-source project by ex-Googlers working at SoundCloud and later donated to CNCF. Prometheus provides features for active scraping, storing, querying, graphing, and alerting based on time series data.

Centralized logging is the second crucial aspect of distributed systems for investigating issues in production environments. Fluentd is one of the main open-source projects of this segment. It provides a unified logging system for connecting various sources of log data to various destination systems. Fluentd was initially developed at Treasure Data and later donated to CNCF. It can be integrated with other open-source monitoring tools, such as Elasticsearch and Kibana, to implement a complete solution for monitoring service logs. Moreover, it can be used for collecting data from a wide variety of systems (including lightweight IoT devices) and building data analytics systems.

Distributed tracing is the third key aspect. Distributed tracing helps provide better insights on analyzing latency bottlenecks, root-cause analysis of errors, resource utilization issues, etc., for applications that are built using a composition of services. Jaeger, Zipkin, and AppDash are three popular open-source projects inspired by Google's distributed tracing platform Dapper. Out of these three, Jaeger and Zipkin are more popular, and Jaeger has better support for OpenTracing-compatible clients.

CONCLUSION

Modern enterprises are now adopting microservices architecture for implementing highly scalable, cloud-agnostic applications that achieve better agility, speed, and lower cost. At a high level, designing such systems may require technologies for container orchestration, implementing microservices, serverless functions, integration services, APIs, service management, and observability. Today, CNCF is taking the lead in providing a vendor-neutral ecosystem for implementing such cloud-native applications using open-source technologies that empower state-of-the-art patterns and practices. Over the last few years, container orchestration features required for hosting stateless applications have matured and are now used in production by many organizations. The mechanics required to run complex stateful applications on containers, such as distributed databases and big data analytics systems, are now being supported. Over time, almost all software applications may run on container platforms incorporating the above technologies. Therefore, organizations should plan for the future by considering the reference architecture explained in this article.

REFERENCES

- Cloud Native Computing Foundation (CNCF) Landscape
- A curated list of microservices architecture-related principles and technologies, Marc Fornos
- Best Practices for Microservice Performance, Google Cloud
- Evolving Distributed Tracing at Uber Engineering

IMESH GUNARATNE is a former Solutions Architect at WSO2 and an Apache committer. WSO2 is an open-source middleware company that delivers solutions for API management, integration, identity and access management, IoT, and analytics. Imesh has consulted WSO2 clients on implementing middleware solutions while specializing in API management and container-related technologies. He contributed to Apache Stratos and BallerinaLang and led the WSO2 PaaS team on delivering WSO2 Private PaaS, Docker, Kubernetes, DC/OS, PCF, and AWS ECD-based solutions.

Full Lifecycle Security for Containers and Cloud-Native Applications

Aqua Security helps you navigate the complex world of containers with a platform that automates and simplifies application security in containerized environments, whether on-premises or in the cloud.

kubernetes MESOSPHERE docker RED HAT OPENSHIFT RANCHER

Shift Left Security

Integrate security into the CI/CD pipeline to provide image risk analysis and rapid remediation early during the build, fixing issues early and avoiding security roadblocks.

Deploy Only Approved Images

Create and enforce an image assurance policy that only allows images that adhere to security and compliance guidelines to be deployed - including vulnerabilities, embedded secrets, malware, secure configuration and more.

Streamline Runtime Protection

Zero-touch runtime controls ensure container integrity and immutability, host and orchestrator hardening, network protection and specific container behaviors, without sacrificing application performance and availability.

Secure Once, Run Anywhere

Apply automated, consistent controls to images, containers, nodes and clusters across any orchestration platform, on Linux and Windows, and across cloud providers

Ensure that your images are free of vulnerabilities

aqua MicroScanner

Free-to-use vulnerability scanner for Docker images - lets you check your container images for vulnerabilities. If your image has any known high-severity issue, MicroScanner can fail the image build, making it easy to include as a step in your CI/CD pipeline.

Try it now ›

To learn more, visit www.aquasec.com

contact@aquasec.com

aqua

Crossing teams: Bridging the gap between DevOps and Security

Many organizations are gradually adopting DevOps methodologies to deploy services to the cloud with limited success. Although an increased number of containers are launched into production, only 19% of all production workloads are containers[1]. The reason is security. Security is a major roadblock, preventing this evolution from accelerating. Here's why:

1. Inadequate capabilities of legacy network/host-based security solution (e.g. legacy firewalls lack visibility into same host inner-container communication)
2. Lack of collaboration. DevSecOps means introducing changes through speed and scale while factoring-in security. Adding security to the mix, for DevOps teams, means 'approval gates', restricted access, etc. Here are some examples of common DevOps practices which are considered security no-no's:
   - Storing secrets in images and forgetting to remove them before deployment
   - Deploying images as root into production

So, how do we add security at scale and speed without impeding development?

We do this by shifting left and enabling security checks and remediation early on in the build stage by integrating scanning capabilities with CI tools (e.g. Jenkins). Developers can scan images, view added/inherited vulnerabilities and take immediate action. In addition, we automate the security 'approval gates' by enabling security admins to create and enforce image assurance policies to only allow images that adhere to security and compliance guidelines to be deployed. We enforce this using our zero-touch runtime protection.

[1] 2018 ESG Research: Trends in Hybrid Cloud Security: Minding the Gap, Key Research Study Findings.

WRITTEN BY SIGALIT KAIDAR
DIRECTOR OF PRODUCT MARKETING - AQUA SECURITY

PARTNER SPOTLIGHT

Aqua Security

Full Lifecycle Security for Containers and Cloud-Native Applications

CATEGORY
Container & workload security solution

RELEASE SCHEDULE
Several times per year

OPEN SOURCE?
No

CASE STUDY

A global company with 30,000+ VMs was looking to move its apps to the cloud while making containers their default app delivery model. They needed a solution to embed automated end-to-end security controls, with vulnerability scanning, runtime controls for their production environments, and secrets delivery to runtime containers. Maintaining business continuity was a key factor, as application downtime would cause the company significant financial and reputational damage.

They turned to Aqua to secure the entire lifecycle of containerized applications while maintaining app high availability and performance so that the runtime environment has no single point of failure due to security. They also use Aqua's Tenant Manager to empower different application teams to create their own security policies, managing them centrally as tenants.

Aqua's container security platform can scan images in the CI/CD pipeline, control which images can run, securely deliver and monitor secrets use, and enforce runtime and network controls on running containers.

STRENGTHS

- "Shift left" security, enabling DevSecOps to accelerate application delivery with full automation and no compromise on security , reducing the application attack surface.
- Protect cloud-native workloads (host-base/CaaS/serverless) in runtime against known vulnerabilities, zero-day exploits, malware, and insider threats, with more automated response and fewer false positives.
- Limit breach impact with a microservices firewall.
- Secure applications once and deploy them anywhere without the need to re-configure security policies and controls, supporting cloud migration and hybrid-cloud strategies.

NOTABLE USERS

- EllieMae
- Telstra
- UNC Chapel Hill Cancer Research

WEBSITE aquasec.com **TWITTER** @AquaSecTeam **BLOG** blog.aquasec.com

Container Wars: Kubernetes vs. Docker Swarm vs. Amazon ECS

BY JULIA PEARSON
FULL STACK SOFTWARE ENGINEER AT NEWSCRED

QUICK VIEW

01. This article compares container orchestration technologies and details the reasons why an engineer might choose one over the others.

02. While Docker Swarm, Kubernetes, and ECS might sound like they have the same offering, they each provide on-boarding ease, configurability, and simple cloud integration respectively.

03. By breaking down the terms each service uses and finding the similarities, we answer the question: "Which container orchestration service is best for me?"

Primitive container technology has been around since the late 1970s but didn't become prominent until Docker debuted in 2013. From that point, containers have exploded in popularity; they are dramatically changing the DevOps landscape and the way we build, ship, and run distributed applications. It's no coincidence, then, that Docker's growth and container usage are developing in synchrony.

However, running a single container is like a lone musical instrument playing its symphonic score in isolation. Container orchestration allows you, the conductor, to unify the orchestra by managing and shaping the sound of the whole ensemble.

Container orchestration tools offer useful and powerful solutions for creating, managing, and updating multiple containers in coordination across multiple hosts. On top of that, orchestration allows you to share data between your services and process tasks asynchronously. Within your production environment, you can run multiple instances of each service over multiple servers to make an application highly available. The more we simplify orchestration, the deeper we can dive into the application and break down smaller microservices --- which raises the question, *Which tool do you pick to conduct your symphony?*

In this article, I'll be comparing the three major players in container orchestration to help you choose the right one for you.

SYNOPSIS

Container orchestration relies heavily on your infrastructure, meaning it is important to look at how these solutions integrate with your current cloud provider/on-premise solution. Are you willing to buy into one cloud provider's entire toolchain, or do you need something more diverse?

Kubernetes is emerging as the current leader in the container orchestration space, surpassing Docker Swarm thanks to its configurability, reliability, and large community. Created by Google as an open-source project, Kubernetes works harmoniously with the whole Google Cloud Platform. Plus, it runs well on almost any infrastructure.

Swarm is Docker's own orchestration tool which is now fully integrated with Docker Engine and makes use of the standard API and networking. Built into the Docker CLI, Swarm Mode requires no additional installation, and new Swarm commands are easy to pick up. Deploying a service can be as simple as using the `docker service create` command. Docker Swarm is fighting back against Kubernetes for the lead in the popularity race by making strides in performance, flexibility, and simplicity to gain renewed adoption.

Amazon Elastic Container Service (ECS) is Amazon's proprietary container scheduler and designed to work in harmony with other AWS services. This means that AWS-centric solutions like monitoring, load balancing, and storage integrate easily into your service. If you are using an alternative cloud provider to Amazon — or if you are running your workload on-premise — then ECS is probably not a fit.

KUBERNETES

Developed from 15 years of working with Linux containers, Kubernetes (abbreviated to K8s or Kube) is Google's open-source answer to container management. It works in multiple production environments including bare metal, on-premise VMs, most cloud providers, plus combinations/hybrids of all three.

Clusters include several major components:

- **Pods**: A group of one or more containers that are created, scheduled, and deployed together on the same node.
- **Labels**: The key-value tags (i.e. the names) assigned to identify pods, services, and replication controllers.
- **Services**: Services give a name to a group of pods, acting as a load balancer to direct traffic to running containers.
- **Replication controllers**: A framework responsible for ensuring that a specific number of pod replicas are scheduled and running at any given time.

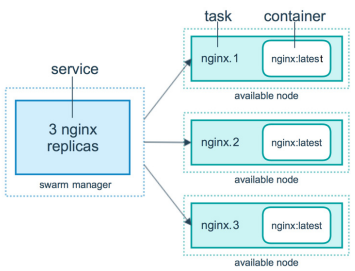
Kubernetes installation is the most complex out of all three, but the process is simpler with the right tools. For integration with existing orchestration systems or bare metal environments, kubeadm works well. Helm is a popular tool that streamlines installing and managing Kubernetes applications. One of Kube's major advantages is that you have ultimate control over its configuration, and more common platforms have plenty of documentation to support your tailored setup. On top of that, Kubernetes has a vast community of users and resources to tap into for support on Slack, StackOverflow, and GitHub if you encounter any problems.

DOCKER SWARM

Docker Swarm extends single-host Docker, allowing developers to quickly and easily deploy multiple containers and microservices. Of the three tools, it's the most lightweight and easiest to transition to, as it's already built into Docker Engine.

Swarms are a cluster of nodes that are comprised of:

- **Manager nodes**: Control orchestration, cluster management, and the distribution of tasks.
- **Worker nodes**: The sole purpose of workers is to run containers and services as assigned by a manager node.
- **Services**: A service describes how you'd like an individual container to distribute itself across your nodes. To create a service, specify the exact information as in an ordinary `docker run`, plus new parameters (e.g. number of container replicas).
- **Tasks**: Single containers place work within these "slots" according to the Swarm manager.



Swarm is great for people just starting with containers or who don't need to configure every little detail. Most importantly, Swarm allows you to scale your containers to dramatic numbers with ease.

Swarm Mode installation requires no special effort, as it's already a part of Docker Engine. The feature comes as standard starting with version Docker 1.12 and above.

ELASTIC CONTAINER SERVICE (AMAZON ECS)

AWS's own container management service, Amazon ECS, is a Docker-compatible service that allows you to run containerized applications on EC2 instances and is an alternative to both Kube and Swarm.

While Docker has won everyone over with its simplicity, Amazon ECS is a comparatively complex tool, as you have to learn a whole new platform. Components within ECS consist of:

- **ECS clusters**: Groups of EC2 instances that run tasks.
- **Task definition**: A text file, in JSON format, which includes much the same information as a `docker run` command, plus details including which containers should run on on one host.
- **Service**: Your tool for running and maintaining specified numbers of task definition instances across your cluster.
- **Service scheduler**: Keeps watch over running tasks and makes sure that the correct number is up, plus the feature rescheduled tasks if they have failed.
- **Container agents**: This feature allows you to connect your cluster instances to your container.

Amazon ECS provides maximum value for those looking for seamless integration between containers and other AWS services. It's a fully managed service that offers high availability, scalability, and security. On top of that, your AWS support plan includes it as standard.

OVERVIEW

Ultimately, choosing the right container orchestration tool comes down to what your priorities are. Taking into account what technology you need it to work alongside.

Are you tied to AWS? It might make sense to use ECS over Kubernetes, and vice versa if Google Cloud is your provider. Are you most keen on configurability and are you willing to labor over complexities for your perfect service? Kubernetes should be your platform of choice.

JULIA PEARSON is a full stack software engineer working at Newscred, a content marketing software company in New York. After graduating from Amherst College, she started her career at Caylent, building a DevOps-as-a-Service platform where she developed an interest in container technology and automating scalable, efficient infrastructure.

Executive Insights on the Current and Future State of Containers

BY TOM SMITH
RESEARCH ANALYST AT **DZONE**

QUICK VIEW

01. The most important elements of orchestrating and deploying containers are security and the ability to maintain hybrid environments.

02. The biggest change in containers has been the growth and adoption of Kubernetes and companies' desire to move out of dev/test directly into production.

03. Serverless and function-as-a-service (FaaS) are the future of containers.

To gather insights on the current and future state of containers, we talked to executives from 26 companies. Here's who we spoke to:

- **Matt Chotin**, Sr. Director of Technical Evangelism, **AppDynamics**
- **Jeff Jensen**, CTO, **Arundo Analytics**
- **Jaime Ryan**, Senior Director, Project Management and Strategy, **CA Technologies**
- **B.G. Goyal**, V.P. of Engineering, **Cavirin Systems**
- **Tasha Drew**, Product Manager, **Chef**
- **James Strachan**, Senior Architect, **CloudBees**
- **Jenks Gibbons**, Enterprise Sales Engineer, **CloudPassage**
- **Oj Ngo**, CTO and Co-founder, **DH2i**
- **Anders Wallgren**, CTO, **Electric Cloud**
- **Navin Ganeshan**, Chief Product Officer, **Gemini Data**
- **Carsten Jacobsen**, Developer Evangelist, **Hyperwallet**
- **Daniel Berg**, Distinguished Engineer Cloud Foundation Services, **IBM**
- **Jack Norris**, S.V.P. Data and Applications, **MapR**
- **Fei Huang**, CEO, **NeuVector**
- **Ariff Kassam**, V.P. Product, **NuoDB**
- **Bob Quillan**, V.P. Container Group, **Oracle**
- **Sirish Raghuram**, CEO and Co-founder, **Platform9**
- **Neil Cresswell**, CEO/CTO, **Portainer.io**
- **Sheng Liang**, Co-founder and CEO and **Shannon Williams**, Co-founder and VP of Sales, **Rancher Labs**
- **Bill Mulligan**, Container Success Orchestrator, **RiseML**
- **Martin Loewinger**, Director of SaaS Operations and **Jonathan Parrilla**, DevOps Engineer, **SmartBear**
- **Antony Edwards**, CTO, **Eggplant**
- **Ady Degany**, CTO, **Velostrata**
- **Paul Dul**, V.P. Product Marketing Cloud Native Applications, **VMware**
- **Mattius McLaughlin**, Engineering Manager & Containers SME, **xMatters**
- **Roman Shoposhnik**, Co-founder, Product & Strategy, **Zededa**

1. The two most important elements of orchestrating and deploying containers **are security and the ability to maintain hybrid environments**. On a container platform, there are four major elements that orchestration must address: networking, storage, security, and management. While you need Kubernetes (K8) to take advantage of Docker, you still need a compliance and security platform. **Follow the CNCF pathway to containerization.**

You must be able to support containers and non-containers since it will take longer than you think to migrate from VMs to containers and you will need observation into and monitoring of both. You will also need to have a holistic view into the hybrid cloud landscape.

2. The languages, frameworks, and tools mentioned most frequently to orchestrate and deploy containers are **Java, Docker, and Kubernetes** with Go and Jenkins also mentioned with great frequency.

3. By far the most dramatic change in the orchestration and deployment of containers in the past year has been the **growth and adoption of Kubernetes** and companies' desire to move out of dev/

test directly into production. K8 came out and emerged with broad adoption as the standard for facilitated ecosystems that everyone is supporting. Now that Docker has begun introducing K8 support into Docker Enterprise Edition, there is a lot more clarity on when and where to use each orchestrator for maximum efficiency. K8 is easier and has more documentation, more maturity, and better tooling. We're now able to set up K8 in five or 10 minutes versus days or weeks.

4. Having a **well-defined security policy and following best practices** is most effective for securing containers. There are multiple layers to secure: 1) repository of container images; 2) cluster of nodes; 3) the container layer; 4) the deployment layer; and 5) container hosts.

A security solution must be able to integrate seamlessly, be lightweight, run distributed, be accurate, respond in real-time, and operate at cloud scale. It must be automated since the orchestration model for application containers is highly automated.

Knowing what's happening in your environment is paramount. Knowing what containers are running versus what containers you expect to be running is key to ensure you are not exposed to any crypto-jacking exploits where hackers gain access to an insecure Docker daemon and start bitcoin miners on your Docker hosts.

5. **There are multiple verticals and two dozen applications of real-world problems being solved using containers.** Ad media, financial services, gaming, healthcare, insurance, oil and gas, retail, and transportation are all using containers to manage AI/ML workloads, reduce infrastructure costs, improve scalability, responsiveness, increase security, and handle big data.

Philips is able to spin up containers to read MRIs for anywhere from 30 seconds to five minutes based on what's needed versus "always on." The GE Predix Platform uses containers to extend services offered to customers and call back to the platform. American Airlines used our application design process to go from design to production in a month to access on-prem services via a containerized cloud-native app. European financial services companies are moving to open banking whereby they expose consumer-level banking information to other consumers to create an ecosystem of applications. A retailer with multiple outlets and slightly different infrastructure make application installation simpler, easier, and more reliable by packaging the app in a container.

6. The most common issue affecting the orchestration and deployment of containers is **lack of knowledge and experience**. People don't know what containers are and the advantages of using them. There's a learning curve with many instructions of how to use Docker and K8. People will find examples of K8 implementations but fail when they move to production because they have not used all of the features and lack the proper configuration for deployment.

There's a shortage of experienced K8, Docker, other tool engineers, as well as DevOps professionals. You still need a developer team with container experience. K8 is architected as a modular set. It takes developers of solution architecture to design a system to manage containers. You need to define personnel needs, timelines, and technology for container management in the cloud.

Containers are not a panacea for bad code and a lack of an agile or DevOps methodology. You need to understand how the culture, processes, and tools will change and realize this will take years, not months.

7. The three most frequently expressed concerns over the current state of containers are: **1) security; 2) lack of education; and 3) expectations**. People think containers are inherently secure when they are not. You must be proactive in following best practices to secure them. People need to be concerned that the frequency of attacks and exploits will continue to increase.

There is a lack of knowledge about container technology and a lack of education for end users, developers, and security professionals. Everyone needs to become more educated on the different aspects of security.

Understand what containers can and cannot do and how long it will take to see the benefits. Realize you will have mixed workloads between containers and VMs for a long time. Do not silo the two or it will make integration even more difficult. People see containers as being easy and straightforward. Things are moving much faster when you are deploying 10,000 containers versus five or 10 VMs.

8. **Serverless and function-as-a-service (FaaS) are the future of containers**. There will be the flexibility to run anywhere to go serverless with great tools that manage abstraction and the portability layer. There will be more innovation around serverless consumption models on top of K8 making K8 easier to use and hiding details from developers when they are not interested. Ultimately, organizations will skip containers and go directly to FaaS like Lambda.

9. **Security and continuous and automated delivery** are most frequently mentioned as things developers needs to keep in mind regarding containers. Think about how your own apps are going to be secured and how the data is going to be secured in motion and at rest. Ensure your application is running in a secure, stable, and repeatable manner. Use the same logic in safely using resources as you did when security and efficiency were primary goals. Figure out how to use continuous delivery and automate the process to improve the quality and security of your applications and containers.

TOM SMITH is a Research Analyst at DZone 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.

PLATFORM9

Achieve Production-Grade Kubernetes In One Hour.

Platform9's SaaS-Managed Kubernetes platform lets you go live quickly. With a few simple clicks, your Platform9 dashboard starts offering visibility and management for your Kubernetes clusters across compute, storage, and network — and your clusters can go live on public clouds or on-premises.

Try Our Sandbox

The Next Generation [Open] Cloud

Platform9 Managed Kubernetes

Kubernetes has become the leading platform for powering modern cloud-native microservices. Its popularity is driven by the many benefits it provides, including cloud-native design, portability across execution environments, and an open-source codebase.

It's easy enough to deploy Kubernetes in a small test bed. You can download it from upstream repositories on one or more virtual machines or physical servers. However, running Kubernetes at scale with production workloads requires much more thought and effort. Here are some criteria to consider while evaluating a Kubernetes solution for your enterprise workloads:

- **High Availability** - Do you have replication for recovery against failures?
- **Hybrid Cloud and Federation Support** - Does your Kubernetes solution support growth across private and public clouds?
- **Enterprise-ready features** - What about single sign-on, role-based access control, isolated networking, and persistent storage?

Platform9 SaaS-Managed Kubernetes works across any underlying infrastructure: physical servers, virtualized environments, or public clouds, including AWS, Azure, and GCP.

Apart from the service being fully managed and working across any server or cloud infrastructure, several common enterprise integrations are built-in:

1. Single pane view of multiple clusters
2. Highly available, multi-master Kubernetes clusters that can be automatically scaled-up and scaled-down
3. Common Enterprise integrations such as single sign-on, isolated namespaces, and the ability to reply applications via Helm charts
4. Cluster federation that provides a truly seamless hybrid environment

Take the first step towards SaaS-Managed Kubernetes. Sign up for a sandbox!

WRITTEN BY BICH LE
CO-FOUNDER AND CHIEF ARCHITECT, PLATFORM9 SYSTEMS

PARTNER SPOTLIGHT

Platform9 Managed Kubernetes

Platform9 SaaS-Managed Kubernetes is the industry's only solution that is infrastructure agnostic, working across public clouds and on-premises infrastructure.

PLATFORM9

CATEGORY
Production-Grade Containers

RELEASE SCHEDULE
Every 6 weeks

OPEN SOURCE?
Yes

CASE STUDY

Containers make it easy to deploy and run modern applications. They are lighter-weight compared to VMs and make more efficient use of the underlying infrastructure. Kubernetes, backed by Google, is the leading container orchestration solution.

Platform9 Managed Kubernetes was launched earlier this year and has seen rapid adoption. It provides customers, large and small, with seamless management of containers across on-premises infrastructure and public clouds, thereby reducing vendor lock-in. Coupled with our integration with IaaS solutions such as OpenStack and VMware, Platform9 has already on-boarded companies such as CSPi, BitSight, Crunch, Cadence, and Amobee. CSPi has used PMK as an underpinning for their next generation solution. BitSight chose PMK for its SLA guarantees, range of integrations, and ability to run on any infrastructure.

STRENGTHS

- Kubernetes anywhere
- Manage both containers and IaaS
- Guaranteed SLAs
- Awards from Gartner, MIT

NOTABLE USERS

- CSPi
- Bitly
- Veritas
- Providence Health
- EBSCO

WEBSITE platform9.com **TWITTER** @platform9sys **BLOG** platform9.com/blog

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	FREE TRIAL	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	Open source	anchore.com
Anchore	Anchore Navigator	Container monitoring	Free tier available	anchore.io
Apache Software Foundation	Mesos	Cluster management software	Open source	mesos.apache.org
Apprenda	Apprenda Cloud Platform	PaaS, container platform, Kubernetes-as-a-Service	Demo available by request	apprenda.com/platform
Aqua	Aqua Container Security Platform	Container image security	Demo available by request	aquasec.com/products/aqua-container-security-platform
Bitnami	Stacksmith	Containers-as-a-Service	30 days	stacksmith.bitnami.com
CA Technologies	Automic Continuous Service	Service orchestration	30 days	automic.com/products/automic-service-orchestration
Canonical	Ubuntu Core	Container OS	Open source	developer.ubuntu.com/core
Cisco	Contiv	Container-defined networking	Open source	contiv.github.io
Codefresh	Codefresh	CI/CD platform for containers	Free tier available	codefresh.io
CoreOS (acquired by Red Hat)	Quay Enterprise	Private container image registry	30 days	coreos.com/quay-enterprise

COMPANY	PRODUCT	PRODUCT TYPE	FREE TRIAL	WEBSITE
CoreOS (acquired by Red Hat)	Clair 2.0.1	Container image security	Open source	coreos.com/clair/docs/latest
CoreOS (acquired by Red Hat)	Tectonic	Containers-as-a-Service	Free up to ten nodes	coreos.com/tectonic
CoreOS (acquired by Red Hat)	Flannel	Container-defined networking	Open source	coreos.com/flannel/docs/latest/
Datadog	Datadog	Server monitoring, container monitoring	14 days	datadoghq.com
Datadog	Diamanti	Container-defined storage hardware	Demo available by request	diamanti.com/products
Docker	Docker	Container platform	Free tier available	docker.com/get-docker
Docker	Docker Swarm	Container orchestration & clustering	Open source	github.com/docker/swarm
Docker	Docker Compose	Multi-container application tool	Open source	docs.docker.com/compose/install
Docker	Docker Hub	Container image registry	Free for one private repo	hub.docker.com
Docker	Docker Trusted Registry	Private container image registry	Open source	docs.docker.com/ee/dtr
Docker	Docker Cloud	Containers-as-a-Service	Available by request	cloud.docker.com
Docker	Docker EE	Containers-as-a-Service, container security	Available by request	docker.com/enterprise-edition
Google	Kubernetes	Container orchestration	Open source	kubernetes.io
Google	Google Cloud Container Registry	Container orchestration	$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

COMPANY	PRODUCT	PRODUCT TYPE	FREE TRIAL	WEBSITE
Instana	Infrastructure Monitoring	Infrastructure monitoring	Available by request	instana.com/infrastructure-management
JFrog	Artifactory	Artifact repository manager	Available by request	jfrog.com/artifactory
Joyent	Triton	Container networking & management platform	$250 credit	joyent.com/triton/compute
Kontena	Kontena Platform	Container platform	Open-source version available	kontena.io/platform
Loggly	Loggly	Log management & analytics	Free tier available	loggly.com
Mesosphere	DC/OS	Container orchestration	Open source	dcos.io
Mesosphere	Enterprise DC/OS	Container orchestration & monitoring	Available by request	mesosphere.com/product
Microsoft	Windows Nano Server	Container OS	180 days	docs.microsoft.com/en-us/windows-server/get-started/getting-started-with-nano-server
Microsoft	Azure Container Service	Containers-as-a-Service, Orchestration-as-a-Service	Available by request	azure.microsoft.com/en-us/services/container-service
Nirmata	Nirmata	Container management platform	Free tier available	nirmata.com/product
Oracle	Smith	Microcontainer builder	Open source	github.com/oracle/smith
Oracle	CrashCart	Microcontainer debugging tool	Open source	github.com/oracle/crashcart
Oracle	RailCar	Rust-based container runtime	Open source	github.com/oracle/railcar
Oracle	Wercker	CI/CD platform for containers	Free tier available	wercker.com
Packet	Packet	Infrastructure-as-a-Service	Demo available by request	packet.net/features
Pivotal	Cloud Foundry	PaaS, container platform	Open source	pivotal.io/platform
Platform9	Platform9 Managed Kubernetes	Kubernetes-as-a-service	Sandbox available	platform9.com/managed-kubernetes

COMPANY	PRODUCT	PRODUCT TYPE	FREE TRIAL	WEBSITE
Portworx	Portworx	Container data services & networking	Demo available by request	portworx.com
Prometheus	Prometheus	Container monitoring	Open source	prometheus.io
Rancher Labs	Rancher 2.0	Container management platform	Open source	rancher.com/rancher
Rancher Labs	RancherOS	Container OS	Open source	rancher.com/rancher-os
Rapid7	Logentries	Log management & analytics	Available by request	logentries.com
Red Hat	Atomic Host	Container OS	Open source	projectatomic.io/download
Red Hat	OpenShift Container Platform	PaaS, container platform	Free tier available	openshift.com/products/container-platform
Sematext	Docker Agent	Container monitoring & log management	Free tier available	sematext.com/docker
Sematext	Kubernetes Agent	Kubernetes monitoring & log management	30 days	sematext.com/kubernetes
Sensu	Sensu Core	Container monitoring	Open source version available	sensuapp.org
Splunk	Splunk Cloud	Log management & analytics	15 days	splunk.com/en_us/products/splunk-cloud.html
Sumo Logic	Sumo Logic	Log management & analytics	Free tier available	sumologic.com
Sysdig	Sysdig Cloud	Container monitoring	Open source	sysdig.com/opensource
Tigera	Canal	Container-defined networking	Open source	github.com/projectcalico/canal
Twistlock	Twistlock Trust	Container image security	Available by request	twistlock.com
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

COMPANY	PRODUCT	PRODUCT TYPE	FREE TRIAL	WEBSITE
WSO2	WSO2 Carbon	Integration platform/PaaS	Free solution	wso2.com/products/carbon
WSO2	WSO2 Enterprise Integrator	ESB	Free solution	wso2.com/products/enterprise-service-bus
WSO2	WSO2 Message Brokers	API management	Free solution	wso2.com/products/message-broker
Chef	Chef Automate	Continuous deployment platform	Demo available by request	chef.io/automate
Chef	Chef Automate	Application and infrastructure release automation	Open source version available	chef.io/chef
Cloud Foundry	Cloud Foundry Container Runtime	Container management platform	Open source	cloudfoundry.org/container-runtime
CA Technologies	CA Application Performance Management	Container monitoring	Free trial	ca.com/us/products/docker-monitoring.html
InfluxData	InfluxData Platform	Database-as-a-Service	14 days	influxdata.com/products
CloudHealth	CloudHealth	Container optimization	14 days	cloudhealthtech.com/solutions/containers
Nutanix	Acropolis Container Storage	Container storage	Free tier available	nutanix.com/products/acropolis/container-storage
StorageOS	StorageOS	Container storage	Free tier available	storageos.com/product
Datera	Datera	Container data services	Demo available by request	datera.io/solutions/#containers
DigitalOcean	DigitalOcean	Container management	Request early access	digitalocean.com/products/kubernetes
Redis Labs	Redis Enterprise	Database as a service	30 days	redislabs.com/redis-enterprise
Scalyr	Scalyr	Log management & analytics	30 days	scalyr.com/product
NeuVector	NeuVector	Multi-vector container firewall	Free trial	neuvector.com/run-time-container-security
CyberArk	CyberArk Conjur	Container security	Open source	cyberark.com/products/privileged-account-security-solution/cyberark-conjur

GLOSSARY

AMAZON WEB SERVICES: A secure cloud services platform with compute power, database storage, and more to help scale businesses.

AZURE CONTAINER INSTANCE: A cloud platform that allows for the rapid creation and launching of containerized applications.

BUILD ARTIFACT: The resulting application or object created by a build process. Typically involves source code being compiled into a runtime artifact. In the Java ecosystem, this involves Java source code being compiled into a JAR or WAR file.

CANARYING/TRAFFIC SHADOWING: A form of testing supported by modern API gateways, edge services, and service meshes in which a fraction of production traffic is routed to a new service and tests conducted with real data.

CONTAINER: Resource isolation at the OS (rather than machine) level, usually (in UNIX-based systems) in user space. Isolated elements vary by containerization strategy and often include file system, disk quota, CPU and memory, I/O rate, root privileges, and network access. Much lighter-weight than machine-level virtualization and sufficient for many isolation requirement sets.

CONTAINER IMAGE: Essentially a snapshot of a container. They are created with a build command and produce a container that you can later run.

CONTAINER-NATIVE: The idea of running containers on bare metal (instead of VMs) that is purposely designed for containers.

CONTAINER ORCHESTRATION: Platforms for developing, deploying, and maintaining container software.

CONTROL PLANE: A set of components responsible for managing and configure the data plane (made up of service proxies).

DATA VOLUME: A marked directory inside of a container that exists to hold persistent or commonly shared data.

DEPLOYMENT: The action of making containers and software available for use in production.

DEVELOPER EXPERIENCE (DEVEX/DX): The collective of methodologies, tooling, and practices that developers use when attempting to convert a business hypothesis or idea into code and ultimately through to functionality running within an observable production environment.

DEVOPS: An IT organizational methodology where all teams in an organization, especially development teams and operations teams, collaborate on both development and deployment of software to increase software production agility and achieve business goals.

DOCKERFILE: A file that contains one or more instructions that dictate how a container is to be created.

DOMAIN NAME SERVICE (DNS): A hierarchical naming system for computers and systems

EVENT-DRIVEN ARCHITECTURE: A software architecture pattern where events or messages are produced by the system and the system is built to react, consume, and detect other events.

GAE FLEXIBLE RUNTIME: The specific framework on which to run an application in the GAE flexible environment; examples include Node.js, Java, and Python.

GCLOUD: A core command in the Google SDK that allows interaction with GAE to initialize and deploy applications.

GOOGLE APPLICATION ENGINE (GAE) FLEXIBLE ENVIRONMENT: A Google-provided PaaS that builds and deploys applications as Docker containers.

INFRASTRUCTURE AS CODE: Process for managing and deploying hardware and software infrastructure through machine automation rather than manual configuration.

INTEGRATION SERVICES (INTSERV): Specifications that help guarantee QoS on networks.

KUBERNETES: An open-source container orchestration system to automate deployment, scaling, and management of containerized applications.

LOG FILE: A document that records events that happen in a piece of software or messages between different pieces of software.

METADATA: A set of structured logic that provides context and information to analyze data.

MICROSERVICES ARCHITECTURE: A development method of designing your applications as modular services that seamlessly adapt to a highly scalable and dynamic environment.

ORCHESTRATION: The method to automate the management and deployment of your applications and containers.

PACKAGE MANAGER: Collection of tools that automate software development processes.

PAVED ROAD PAAS/WORKFLOW: A set of best practice frameworks, tools, and methodologies, typically supplied and managed by a centralized team; this term was popularized by Netflix.

PLATFORM-AS-A-SERVICE: An environment that provides a standard approach to deploying applications, with no effort or minimal effort required to configure the underlying infrastructure.

PRIVATE CONTAINER REGISTRY: A private and secure location to publish, store, and retrieve container images for software you use in your infrastructure.

PROXY: An agent that intercepts and forwards traffic.

RESILIENCE: The ability for a system or server to recover from equipment failures, timeouts, and power outages.

SERVERLESS: A platform providing computing, networking, and storage without the need of managing (virtual) machines.

SERVICE MESH: Decentralized, transparent application networking infrastructure helping applications and services communicate with each other reliably regardless of what programming language or framework they use.

SERVICE PROXY: A lightweight proxy that understands application semantics (requests, retries, timeouts, etc.) that is capable of transparently augmenting an application's capabilities to add resilience, observability, security, and control. Service proxies make up the "data plane" in a service mesh.

WHITE BOX: Technology that is open and transparent — the internal structure/design is a known quality.

INTRODUCING THE

Open Source Zone

Start Contributing to OSS Communities and Discover Practical Use Cases for Open-Source Software

Whether you are transitioning from a closed to an open community or building an OSS project from the ground up, this Zone will help you solve real-world problems with open-source software.

Learn how to make your first OSS contribution, discover best practices for maintaining and securing your community, and explore the nitty-gritty licensing and legal aspects of OSS projects.

COMMITTERS & MAINTAINERS

COMMUNITY GUIDELINES

LICENSES & GOVERNANCE

TECHNICAL DEBT

Visit the Zone

BROUGHT TO YOU IN PARTNERSHIP WITH Flexera