DZone

THE DZONE GUIDE TO

Orchestrating and Deploying Containers

VOLUME I

BROUGHT TO YOU IN PARTNERSHIP WITH

Automic™ CoreOS Diamanti Dev Done Fast. Ops Done Right.™ INSTANA

JFrog PLATFORM9 Twistlock sysdig TIGERA

DEAR READER,

In 1955, a North Carolina businessman named Malcom P. McLean came up with a simple yet ingenious idea. He realized that it would be much easier to have a standardized container that could be moved directly from a trailer or train into a ship without unloading its content, thus improving transportation efficiency, reliability, and cost, and also reducing cargo theft. He bought a steamship company and started a revolution. 60 years later, most of the world's goods travel this way.

Today, a new type of container is revolutionizing the way we build, ship, and deploy applications, but unlike its physical namesake, these virtualized containers were not the invention of a single man. The ideas behind it started as early as 1979, with the introduction of the chroot system call in Unix V7, a primitive way of process isolation through changing its root directory to a new location in the filesystem. Two decades later, FreeBSD Jails was introduced, and it was probably the first truly isolated virtualization mechanism. Many others followed, such as Linux VServer, Solaris Containers, Open Virtuozzo, Process Containers, LXC, Warden, etc.

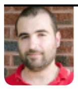
All this evolution in the virtualization space led to Docker taking the world by storm in 2013. It didn't just happen because Docker was an improved virtualization technology, but because it was an entire set of tools that allowed developers to create and run application containers very quickly, which was the big missing piece of the puzzle. Docker Hub was also a significant evolution, providing a centralized repository of thousands of images that everyone could use as either a base to develop their containers or as part of their application infrastructure.

However, Docker is not alone in this space. All the big names are in on it, as well as companies that own alternative solutions. The good news is that they are not trying to develop competing technologies, but instead, they are cooperating with the goal of developing industry standards for both the format and runtime software of containers on all platforms. This spirit of collaboration was spurred by the Open Container Initiative (OCI), established by Docker in 2015 who donated 5% of its code base to help jump start it. Even former competitors like CoreOS and VMware are participating.

This way, all those organizations can focus their time and resources on developing the tools required to support the additional challenges brought in by the containerized world, such as container orchestration, management, security, logging and monitoring, or routing and networking.

In this guide, we want to show you what the revolution is, what major problems the container technology has solved, and what new challenges it has produced. We'll show you the current solutions for those problems, as well as a glimpse of what the future has in store. We believe you'll enjoy it and find it useful, whether you are new to application containerization, or you're already a field expert.

Thank you for reading it, and thank you to everyone who contributed to it. Happy reading.



BY HERNANI CERQUEIRA
DISTINGUISHED ENGINEER, DZONE

TABLE OF CONTENTS

PRODUCTION
Chris Smith
DIRECTOR OF PRODUCTION
Andre Powell
SR. PRODUCTION COORDINATOR
G. Ryan Spain
PRODUCTION PUBLICATIONS EDITOR
Ashley Slate
DESIGN DIRECTOR
Billy Davis
PRODUCTION ASSISSTANT

MARKETING
Kellet Atkinson
DIRECTOR OF MARKETING
Lauren Curatola
MARKETING SPECIALIST
Kristen Pagàn
MARKETING SPECIALIST
Natalie Iannello
MARKETING SPECIALIST
Miranda Casey
MARKETING SPECIALIST
Julian Morris
MARKETING SPECIALIST

BUSINESS
Rick Ross
CEO
Matt Schmidt
PRESIDENT
Jesse Davis
EVP
Gordon Cervenka
COO

SALES
Matt O'Brian
DIRECTOR OF BUSINESS DEV.
Alex Crafts
DIRECTOR OF MAJOR ACCOUNTS
Jim Howard
SR ACCOUNT EXECUTIVE
Jim Dyer
ACCOUNT EXECUTIVE
Andrew Barker
ACCOUNT EXECUTIVE
Brian Anderson
ACCOUNT EXECUTIVE
Chris Brumfield
SALES MANAGER
Ana Jones
ACCOUNT MANAGER
Tom Martin
ACCOUNT MANAGER

EDITORIAL
Caitlin Candelmo
DIRECTOR OF CONTENT AND COMMUNITY
Matt Werner
PUBLICATIONS COORDINATOR
Michael Tharrington
CONTENT AND COMMUNITY MANAGER
Kara Phelps
CONTENT AND COMMUNITY MANAGER
Mike Gates
SR. CONTENT COORDINATOR
Sarah Davis
CONTENT COORDINATOR
Tom Smith
RESEARCH ANALYST
Jordan Baker
CONTENT COORDINATOR
Anne Marie Glen
CONTENT COORDINATOR

Want your solution to be featured in coming guides?
Please contact research@dzone.com for submission information.

Like to contribute content to coming guides?
Please contact research@dzone.com for consideration.

Interested in becoming a dzone research partner?
Please contact sales@dzone.com for information.

Special thanks to our topic experts, Zone Leaders, trusted DZone Most Valuable Bloggers, and dedicated users for all their help and feedback in making this guide a great success.

Executive Summary

BY MATT WERNER
PUBLICATIONS COORDINATOR, DZONE

In 2013, seemingly overnight, Docker caught the imagination and attention of the software development world. Since then, competitors have arisen, new tools to make the most of containers have been developed, and the open source community around the technology has been booming. Containers have been the subject of fascination at almost every tech conference since, particularly at DevOps-focused events, where their benefits for testing, environmental consistency, and faster time-to-market have been discussed by hundreds of different speakers from hundreds of different backgrounds. For a technology that's so recently entered the spotlight and has gotten so much attention, professionals have to ask: are they worth the endless hype and praise?

It's clear that containers are here to stay, at least for a while, and it's time to see how containers have been adopted by the software developers in DZone's audience. To learn about the benefits and challenges of using containers, along with the tools that they used to work with them, DZone asked over 800 software developers and IT professionals to share the details of their container journeys. In our largest guide yet, we've asked notable members of the DZone community to share their advice, stories, and opinions on the space as well.

A NEW CHALLENGE IS APPROACHING

DATA 70% of respondents found refactoring or re-architecting legacy apps to be a very or moderately challenging aspect of adopting containers. Lack of developer experience was cited as challenging by 68% of respondents, and 64% mentioned the difficulty of ensuring application and network security.

IMPLICATIONS Most of the challenges around adopting containers stem from how relatively new standalone container management technologies like Docker are. Implementing containers into existing systems and training developers on how to use them can take a significant amount of time and effort. Not only that, but teaching developers how to develop secure applications is already a challenge, and adding containers to the mix only complicates the process more.

RECOMMENDATIONS First, don't feel like you need to reinvent the wheel for all of your existing applications. Containerizing legacy apps may not be necessary right at this moment depending on the quality of the application, its necessity to the business and its customers, and other priorities on the team, so don't feel that adopting Docker for one project means you have to adopt it for every project. Spend time developing new applications for containers first. Second, security will improve the more developers adjust to planning their builds around security testing and writing secure code. Third, the best way to overcome a lack of experience is to practice in your spare time, as we'll elaborate below…

PRACTICE MAKES PERFECT

DATA 35% of developers who use containers in personal projects find refactoring challenging, compared to 43% who do not use containers in personal projects. 26% of those working on personal projects find developer experience a hindrance compared to 33% who don't, and 18% of those who use containers in personal projects find security to be difficult, compared to 22% who don't. A majority of respondents use containers in personal projects (55%).

IMPLICATIONS While all of these factors were considered some of the biggest challenges facing container adoption, those who use containers in their personal development do not consider them to be as difficult, suggesting that using these technologies outside of the office increases familiarity with the tools that can benefit the workplace.

RECOMMENDATIONS If you're finding it difficult to introduce or use containers in your work place due to any of the challenging factors previously mentioned, it may be worth it to install Docker on your home machine and try using it by yourself. The knowledge you can pick up from learning technologies on your own can cross over to your workplace and help teach and inspire your team to adopt containers.

WAS IT WORTH IT?

DATA Container users find that the greatest benefits containers provide are environmental consistency (63%), followed by faster deployment (56%), portability (54%), and scalability (53%). Those who are currently evaluating containers expect similar benefits (64%, 56%, 51%, and 48%, respectively).

IMPLICATIONS A majority of developers who use containers have found that they offer significant benefits for their application's health as well as their development process. It seems that those evaluating containers are more cynical about whether containers can truly improve portability and scalability. Several of these benefits, particularly environmental consistency and faster deployment, also improve DevOps processes. 40% of those whose organizations have adopted containers believe they have achieved Continuous Delivery.

RECOMMENDATIONS For those who believe adopting containers will benefit their organizations, applications, and processes, adoption should move forward, as the numbers are very consistent between those who are using vs. those who are evaluating containers. Those who are trying to achieve Continuous Delivery may also find that containers bring them closer to this ever-elusive goal.

Key Research Findings

BY G. RYAN SPAIN
PRODUCTION COORDINATOR, DZONE

DEMOGRAPHICS

811 software professionals completed DZone's 2017 Containers survey. Respondent demographics are as follows:

- 36% of respondents identify as developers or engineers; 20% identify as developer team leads; and 16% identify as architects.
- The average respondent has 14.5 years of experience as an IT professional. 60% of respondents have 10 years of experience or more; 20% have 20 years or more.
- 45% of respondents work at companies headquartered in Europe; 32% work in companies headquartered in North America.
- 17% of respondents work at organizations with more than 10,000 employees; 21% work at organizations between 1,000 and 10,000 employees; and 24% work at organizations between 100 and 1,000 employees.
- 83% develop web applications or services; 55% develop enterprise business apps; and 26% develop native mobile applications.

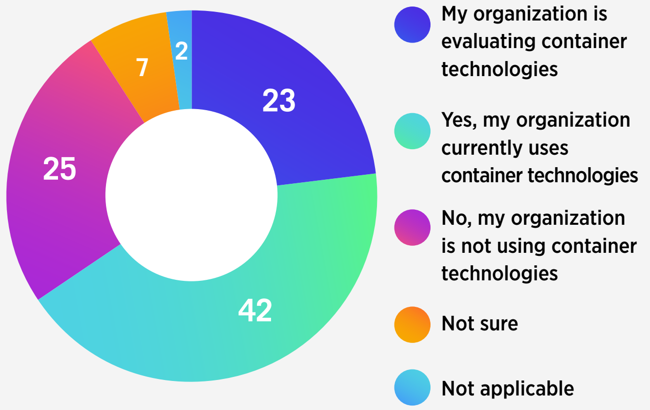
TOOLS OF THE TRADE

It should come as no surprise at this point that Docker is the most dominant single tool in the containers ecosystem at the moment. 92% of survey respondents who work in organizations that use container technologies said their organization uses Docker, followed far behind by Docker Enterprise at 12%, and LXC at 4%. However, there are plenty of other technologies in the ecosystem to discuss. For container orchestration/management, the spread between tools is much more even. 35% of respondents working at organizations that use containers said their organization uses Kubernetes, while 32% said they use Docker Swarm. In addition, 24% of the respondents who said their organization uses one of these said they use both. Amazon ECS was also a contender for orchestration/management tools, with 26% of respondents reporting that their organizations use it. For container operating systems, CoreOS Container Linux was most popular choice at 26%, with boot2docker close behind at 20%, though 26% of respondents weren't sure what OS their organization is using for their containers. OpenShift was the most popular PaaS for containers at 18%, and Cloud Foundry was next at 12%, but 47% of all respondents at container-using organizations said they don't use a PaaS at all.

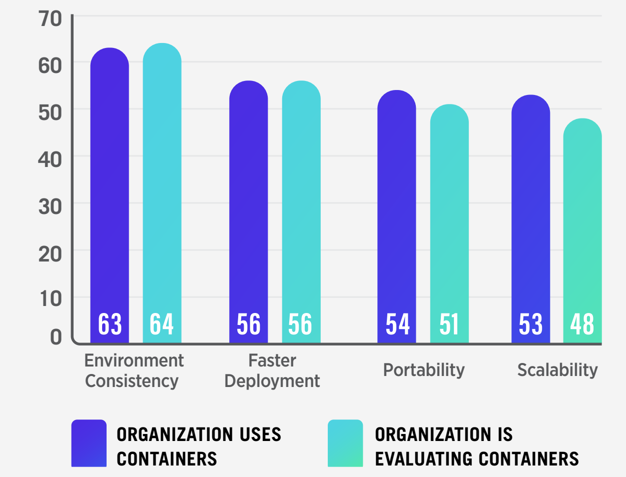
ALL ABOUT THE BENEFITS

Using containers can have many benefits depending on how they are being used and what they are being used for. We asked respondents what benefits they saw from containers in their organization: 63% said that containers greatly benefited environment consistency, 56% said that they greatly benefit faster deployment, 54% said that they greatly benefit portability, and 53% said they greatly benefit scalability. Those

▸ Does your organization currently use container technologies?



▸ Benefits to the organization expected/received from containers



working at organizations still evaluating containers said they expected these great benefits in these areas at very similar rates: 64% for environmental consistency, 56% for faster deployment, 51% for portability, and 48% for scalability.

CONTAINER COMPLICATIONS AND CONCERNS

The benefits of containers, however, can be offset by challenges that they invoke. Respondents from organizations that use containers found that refactoring or rearchitecting legacy applications was a major challenge with containers — 70% of these respondents said this was either "very challenging" or "moderately challenging." 68% of these respondents found lack of developer experience with container technologies challenging; and 64% said ensuring application and network security was either very challenging or moderately challenging. However, while those working at organizations evaluating container technologies said they expect the same top three challenges, their percentages were significantly higher. 86% expect that refactoring or rearchitecting will be a challenge, 80% expect lack of development experience will be a challenge, and 75% expect that ensuring security will be a challenge.

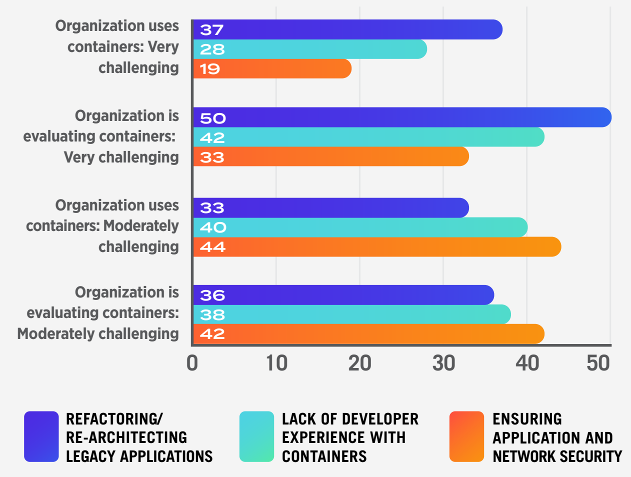
A PIECE OF THE PUZZLE

The container ecosystem is just one moving part of the IT landscape, and it impacts and is impacted by other moving parts. For example, containers and microservices architectures are two such pieces that affect one another. We found that 50% of total respondents said their organizations had adopted microservices, while 37% said no (11% were not sure and 2% said they were not applicable). Of the respondents who said their organization uses containers, 68% said their organization has adopted microservices; of those working for container-evaluating organizations, 52% said their organization had adopted microservices; and for the respondents who said their org doesn't use containers, 26% said they had adopted microservices. The belief that an organization had achieved Continuous Delivery also varied between respondents based on container usage. 40% of respondents at container-using organizations said they believed they had achieved CD, versus 28% for container-evaluators and 23% for non-container-users. Responses on deployment speeds were affected as well; overall, 25% of respondents said their organization performed on-demand deployments (multiple times a day), but 36% of those at organizations using containers said they deploy on-demand, compared to 15% of those evaluating containers and 19% of those not using containers.

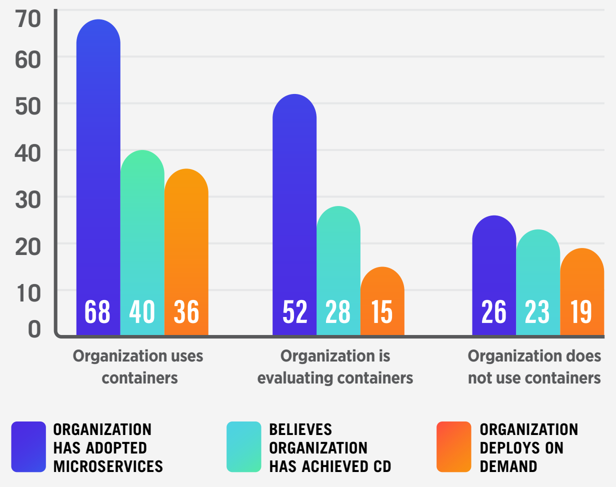
OFF THE CLOCK

We found that whether or not a developer said they use containers for personal development projects affected their answers on how they felt about containers, as well as how they developed overall. First, those working at organizations that use containers were much more likely to say they use containers for their own projects: 69% of these respondents use containers for personal dev projects, compared to 56% who work at companies evaluating containers and 37% who work at companies that do not use containers. Those respondents who do use containers for personal projects were less likely to find the top container challenges discussed earlier "very challenging." For example, 35% of personal container users thought refactoring/rearchitecting legacy apps was very challenging, while 43% of those who don't use containers in their own projects thought the same. Respondents who said they are learning and keeping their skills up to date by participating in developer community activities were overall more likely to use containers in their own projects: 75% of respondents who contribute to open source projects (14% of overall responses), 65% of respondents who attend meetups (29% overall), and 63% of those who attend conferences and workshops (40% overall) said they use containers in their own projects.

Challenges to the organization expected/presented from containers

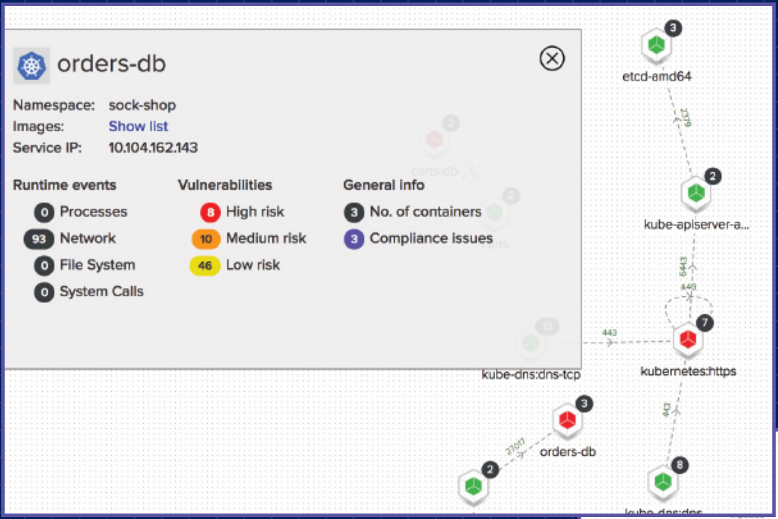


Impacts on microservices, CD, and deployment



Twistlock

THE MOST COMPREHENSIVE CONTAINER SECURITY SOLUTION



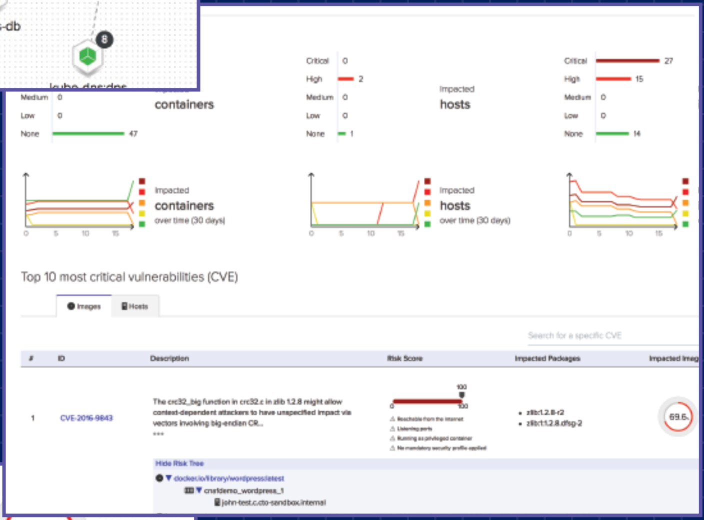
orders-db

Namespace: sock-shop
Images: Show list
Service IP: 10.104.162.143

Runtime events	Vulnerabilities	General info
0 Processes	8 High risk	3 No. of containers
93 Network	10 Medium risk	3 Compliance issues
0 File System	46 Low risk	
0 System Calls		

RUNTIME DEFENSE

Automated, scalable active threat protection and cloud native application firewall



Top 10 most critical vulnerabilities (CVE)

Images | Hosts

Search for a specific CVE

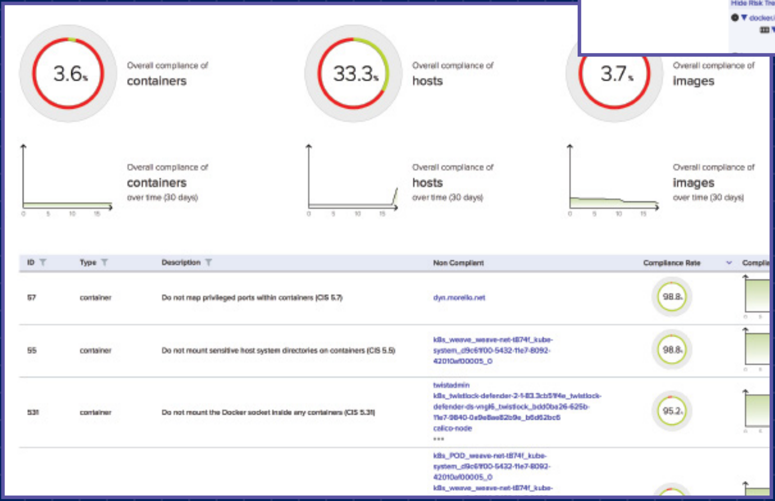
#	ID	Description	Risk Score	Impacted Packages	Impacted Images
1	CVE-2016-9843	The crc32_big function in crc32.c in zlib 1.2.8 might allow context-dependent attackers to have unspecified impact via vectors involving big-endian CR...	100	• zlib:1.2.8-r2 • zlib:1:1.2.8.dfsg-2	69.6

Hide Risk Tree

docker.io/library/wordpress:latest
cnsfdemo_wordpress_1
john-test.c.cto-sandbox.internal

VULNERABILITY MANAGEMENT

Precise controls to detect and prevent vulnerabilities before they reach production



3.6%
Overall compliance of containers
33.3%
Overall compliance of hosts
3.7%
Overall compliance of images
Overall compliance of containers over time (30 days)
Overall compliance of hosts over time (30 days)
Overall compliance of images over time (30 days)

ID	Type	Description	Non Compliant	Compliance Rate
57	container	Do not map privileged ports within containers (CIS 5.7)	dyn.morello.net	98.8%
55	container	Do not mount sensitive host system directories on containers (CIS 5.5)	k8s_weave_weave-net-t874f_kube-system_d9c6f100-5432-11e7-8092-42010af00005_0	98.8%
531	container	Do not mount the Docker socket inside any containers (CIS 5.31)	twistadmin k8s_twistlock-defender-2-1.83.3cb5114e_twistlock-defender-ds-vngl6_twistlock_bdd0ba26-625b-11e7-9840-0a9e8ee82b9e_b6d52bc6 calico-node ...	95.2%
			k8s_POD_weave-net-t874f_kube-system_d9c6f100-5432-11e7-8092-42010af00005_0 k8s_weave_weave-net-t874f_kube-	

COMPLIANCE

Extend and enforce your corporate compliance across your container environment

TO LEARN MORE, VISIT...

Twistlock.com

How Containers Can Revamp Your Approach to Security

Containers make it possible to adopt a fundamentally new approach to security. Rather than relying solely on perimeter-level defenses and waiting for intrusion attempts to start before reacting, you can minimize security vulnerabilities within your production environment and prevent intrusions before they even begin.

The small surface area of containers minimizes the opportunity for attackers to find a vulnerability to exploit. The transparency of containerized applications, combined with environment parity, makes it much easier for your security experts to become part and parcel of the software delivery process and identify potential security gaps at any layer (within application code itself, in environment parameters, or in the host server) before software is released to production. Environment parity also helps to ensure that software that is deemed secure during pre-production testing can be trusted to remain secure once it is in production.

That's not all. In addition to helping to prevent opportunities for attack, microservice architectures and containers also enable organizations to be proactive about responding to attacks once they are underway. In the old, reactive world of security, a response to an intrusion would involve assessing how the attack occurred, then trying to figure out what hole facilitated it. Under the new, proactive paradigm, organizations can mitigate attacks instantly. Using immutable infrastructure and machine learning, security policies can be updated constantly so that attacks are detected in real time. Responses can also be automated in order to stop an intruder in their tracks, rather than waiting until the damage is done to shut them out.

WRITTEN BY JOHN MORELLO
CTO, **TWISTLOCK**

PARTNER SPOTLIGHT

Twistlock Enterprise Edition

Twistlock protects today's applications from tomorrow's threats via advanced intelligence, machine learning, and automated policy enforcement.

CATEGORY
Container and Cloud Native application security

NEW RELEASES
6x year

OPEN SOURCE
No

CASE STUDY

ClearDATA provides secure, managed services for healthcare and life sciences on AWS. ClearDATA customers must comply with significant regulatory requirements, have huge amounts of sensitive data to manage, and customers demanding better data collaboration.

In order to help their clients deliver solutions faster, ClearDATA wanted to deliver a new set of product and service offerings to allow health organizations to run Docker containers using AWS' EC2 Container Services (ECS).

Using Twistlock in their environment has enabled ClearDATA to monitor and enforce compliance requirements, check for vulnerabilities from development through production, and automate runtime defense that scales within the ECS environment.

STRENGTHS

- **Runtime Defense**: Automatically prevent next gen attacks against containers and cloud native apps
- **Vulnerability Management**: Detect and prevent vulnerabilities before they make it to production
- **Compliance**: Extend regulatory and corporate compliance into your container environment
- **Cloud Native Application Firewall**: Automatically protect your apps in a 'software defined' manner
- **Access Control**: Define new or extend existing policies and enforce them across your container stack

NOTABLE CUSTOMERS

- ClearDATA
- Aetna
- Booz Allen Hamilton
- AppsFlyer

WEBSITE twistlock.com **TWITTER** @twistlockteam **BLOG** twistlock.com/blog

Introduction to Highly Available Container Applications

QUICK VIEW

01 Running a highly available containerized application with failover and load balancing requires different tools from Docker at development time.

02 It takes a lot of work to make containerized networking easy.

03 Container orchestration is essential for scaling and networking issues that arise with highly available applications in containers.

04 The two main frameworks for container orchestration have similar approaches to provide an environment for containers.

BY **ALAN HOHN**
LM FELLOW, **LOCKHEED MARTIN ROTARY AND MISSION SYSTEMS**

CONTAINER DEVELOPMENT VERSUS PRODUCTION

In *The Mythical Man Month*, Fred Brooks estimates that it takes nine times the effort to create a complete software system as it does to create just the core software functionality. This rule of thumb certainly applies to a containerized system architecture, which is much more complex than just a simple application running in a container.

To design a container architecture, we need to gain a deeper understanding of how container infrastructure works, especially in three main areas:

- **Multiple containers**: How does our approach change when our system consists of multiple containers?
- **Multiple hosts**: How can we distribute our containers across multiple hosts without having to explicitly manage individual container or host instances?
- **Networking**: How can we connect a set of containers while also providing the same isolation we expect when we run containers for separate applications?

It might be tempting to avoid this complexity by just bundling our system components into a single uber-container. Unfortunately, this solution eventually hits limits. First, in order to scale a system efficiently, we usually need to scale different parts in different amounts. For example, we generally need many more web server instances to handle serving content to users than we need database instances for those web server instances to use. Second, we still have to contend with sharing state across those multiple uber-containers. Third, we miss out on some of the isolation advantages with an uber-container, such as avoiding a failure in one component rippling into other components.

So it appears that we are stuck running multiple containers. Indeed, the usual approach is the *microservice architecture*, where each discrete piece of our system gets its own container so that it can be scaled independently, upgraded independently, and developed independently.

And we also need multiple hosts in order to create a highly available system because servers are fleeting. So we need to run lots of instances of lots of container images, spread across multiple hosts but networked together. Here's how that is accomplished. I'll be using the Docker and Kubernetes ecosystems for examples, but the core issues arise no matter what technology is used.

CONTAINER ORCHESTRATION

Running multiple linked containers in Docker can be done in a shell script. Here's the one I use for running a development instance of Atlassian JIRA:

```
#!/bin/bash
docker run --name postgres -e POSTGRES_PASSWORD=password \
  -v $(pwd)/pgdata:/var/lib/postgresql/data -d postgres:9.4
docker run -it --name jira --link postgres:postgres \
  -p 8080:8080 -v $(pwd):/mnt jira
```

But this approach is missing essential features. First, there is no built-in mechanism to restart any failed containers. Second, this solution is tied to a specific host and can't easily be distributed to multiple hosts or scaled to multiple instances. Third, the script requires close reading to figure out what is going on.

This is why container orchestration exists; it addresses all of these issues and more. A simplified Docker Compose file for the same purpose looks like this:

```
version: '3'
services:
  postgres:
    image: postgres:9.4
  jira:
    image: jira:latest
    links:
      - postgres:postgres
    ports:
      - "8080:8080"
```

A Kubernetes Replication Controller definition would look much the same. Both are much more readable than our Bash script. Also, by allowing an orchestration engine to run the containers, we can restart failed containers, scale to multiple instances, and distribute our application across multiple hosts.

CONTAINER NETWORKING

Scaling to multiple instances and distributing across multiple hosts raises important issues with networking. On a single host, it is easy to understand how links work between containers. Docker gives each container its own set of virtual network devices. These devices are all connected to some software-defined network, and Docker determines the IP address a container gets. So a link between containers is ultimately just an entry in /etc/hosts that ties a name to the correct IP address.

However, once we start using a container orchestration engine, the situation gets more complicated. First, there might be multiple instances of each container, so the orchestration engine must apply a unique name to each. Second, these instances may come and go. To allow for a more dynamic way for containers to find each other, both Docker and Kubernetes provide a Domain Name Service (DNS) server that is automatically updated as instances come and go. Where there are multiple instances, the DNS server either provides the IP address for one, or sends the whole list.

CROSSING HOSTS

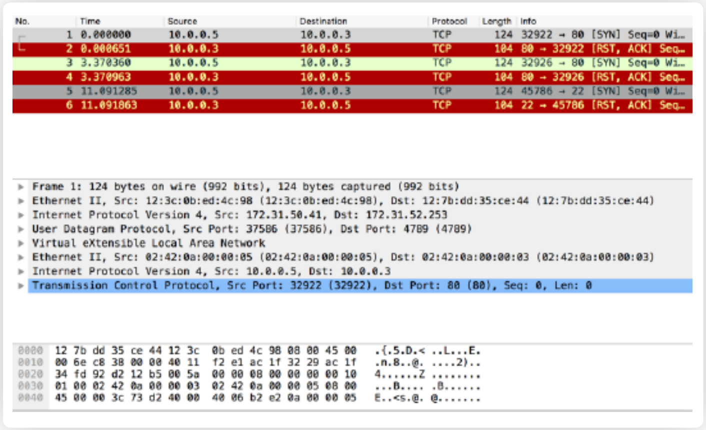
Multiple software-defined networks allow for isolation if we run multiple sets of containers. DNS provides discovery of container instances on those software-defined networks. But in order to spread our containers across hosts, we need one more feature, which is connecting a software-defined network on one host with the right software-defined network on the other host.

To allow for a more dynamic way for containers to find each other, both Docker and Kubernetes provide a Domain Name Service (DNS) server that is automatically updated as instances come and go.

Of course, the container orchestration engine places the software-defined networks on the same subnet and avoids duplicate IP addresses. But any switches and routers between the hosts are going to get confused by container IP addresses. So the host encapsulates the traffic inside messages that look like normal host-to-host communication. There are multiple ways to do this, but the most popular is Virtual Extensible Local Area Network (VXLAN).

VXLAN works by sending User Datagram Protocol (UDP) packets on port 4789 between hosts. The receiving host unpacks and sends the contents to the correct container. As a result, the container appears to be directly connected to another container, even when those containers are running on separate hosts.

Here's an example Wireshark capture from a Docker Swarm environment. One container is connecting to another on port 80.



```
No.        Time        Source        Destination    Protocol  Length  Info
      1 0.000000    10.0.0.5      10.0.0.3       TCP         124  32922 → 80 [SYN] Seq=0 Wi…
      2 0.000651    10.0.0.3      10.0.0.5       TCP         104  80 → 32922 [RST, ACK] Seq…
      3 3.370360    10.0.0.5      10.0.0.3       TCP         124  32926 → 80 [SYN] Seq=0 Wi…
      4 3.370963    10.0.0.3      10.0.0.5       TCP         104  80 → 32926 [RST, ACK] Seq…
      5 11.091285   10.0.0.5      10.0.0.3       TCP         124  45786 → 22 [SYN] Seq=0 Wi…
      6 11.091863   10.0.0.3      10.0.0.5       TCP         104  22 → 45786 [RST, ACK] Seq…

▶ Frame 1: 124 bytes on wire (992 bits), 124 bytes captured (992 bits)
▶ Ethernet II, Src: 12:3c:0b:ed:4c:98 (12:3c:0b:ed:4c:98), Dst: 12:7b:dd:35:ce:44 (12:7b:dd:35:ce:44)
▶ Internet Protocol Version 4, Src: 172.31.50.41, Dst: 172.31.52.253
▶ User Datagram Protocol, Src Port: 37586 (37586), Dst Port: 4789 (4789)
▶ Virtual eXtensible Local Area Network
▶ Ethernet II, Src: 02:42:0a:00:00:05 (02:42:0a:00:00:05), Dst: 02:42:0a:00:00:03 (02:42:0a:00:00:03)
▶ Internet Protocol Version 4, Src: 10.0.0.5, Dst: 10.0.0.3
▶ Transmission Control Protocol, Src Port: 32922 (32922), Dst Port: 80 (80), Seq: 0, Len: 0

0000  12 7b dd 35 ce 44 12 3c  0b ed 4c 98 08 00 45 00   .{.5.D.< ..L...E.
0010  00 6e c8 38 00 00 40 11  f2 e1 ac 1f 32 29 ac 1f   .n.8..@. ....2)..
0020  34 fd 92 d2 12 b5 00 5a  00 00 08 00 00 00 00 10   4......Z ........
0030  01 00 02 42 0a 00 00 03  02 42 0a 00 00 05 08 00   ...B.... .B......
0040  45 00 00 3c 73 d2 40 00  40 06 b2 e2 0a 00 00 05   E..<s.@. @.......
```

You can see the TCP port 80 connection attempt (blue bar). This is inside a regular IP packet inside an Ethernet message. This whole Ethernet message is then hidden inside a VXLAN message that travels over UDP on the host network.

So now our container orchestration environment can deploy a set of containers spread across many hosts and give the software in these containers access to each other over what looks like a completely independent private network.

EXPOSING SERVICES

We're still missing one piece in order to construct a typical system. We need at least one thing to be accessible from outside the container network. With "docker run", this just meant exposing a port. Docker opens this port on the host, and all traffic to it goes to a port on the container.

Both Docker Swarm and Kubernetes support a similar approach in an orchestration environment. However, there is extra complexity because there may be multiple instances across multiple hosts. So the orchestration agent on every host must listen on all exposed ports used by any container, then route the traffic to a host running an instance of that container.

This way of doing things has the limitation that we must either communicate the port to clients (which is complex) or find a free port (which removes one of the advantages of containers: lack of resource conflicts with other applications). One alternative is to allocate an IP address to a service that is accessible from outside the orchestration environment. This is currently possible with Kubernetes, though it requires some integration with the external IP provider. Docker Enterprise Edition has an alternate approach specific to HTTP that adds an entry in DNS that routes to an HTTP load balancer. The HTTP load balancer selects the server based on the host identified in the HTTP request.

ARCHITECTURAL IMPLICATIONS

A typical containerized system has lots of instances of services. These instances start and stop at any time. Some services are exposed to the outside world. So how do we locate a service? From the client side, there are two solutions. First, we can get the address of one instance and start our conversation with that one. Second, we can get a list of instances and either pick one or load balance across multiple instances. The first approach might not be as efficient, but it does not require any special logic in the client other than the robustness to handle losing its server connection.

The orchestration agent on every host must listen on all exposed ports used by any container, then route the traffic to a host running an instance of that container.

In the service, we also have two main approaches. First, we can have one instance handle all of the client traffic itself, with other instances just acting as backup. Second, we can have all of the instances sharing the work as much as possible. The second approach scales better, but is of course much more complex, especially if the service is storing data that needs to be synchronized.

For a containerized microservice architecture, the main appeal is scaling across many instances. So we would like to have clients and services that understand load balancing. However, in real systems, clients such as browsers don't know about our highly distributed, load balanced approach. So at some level, we have to use both these approaches: a single front-end (with backup) that dispatches work, and lots of load balanced services that do the heavy lifting. Not surprisingly, both Docker Swarm and Kubernetes are designed around this pattern.

Of course, this article is an introduction to a very complex topic, but hopefully it is a useful foundation that makes it clear why container orchestration is important and why the two main implementations offer so many of the same features.

Alan Hohn is a Fellow with Lockheed Martin Rotary and Mission Systems. While most of his background is in Java, especially Java EE and OSGi, lately he's been doing a great deal of work in DevOps and with containerized architectures and applications, especially the applicability of containers to dynamic, complex military applications. This has also meant learning Go, which is his new favorite language.

Secure Networking for Kubernetes

BY MIKE STOWE

DIRECTOR OF COMMUNITY AT TIGERA

While Kubernetes has quickly become the leading container orchestrator by greatly reducing many of the challenges of deploying, scaling, and managing containers, it still leaves developers with a basic challenge: building a secure, scalable, and resilient network for containers to be deployed on.

Not only is it vital that your network be scalable, but it's just as important that it is built to be secure to prevent outside breaches and prevent internal breaches by building on isolation policies (or limit the attack surface once a malicious user is inside your network).

This approach means focusing on several different layers of the network (from L3 to L7) to ensure that a breach on any microservice, container, node, or host is quickly isolated and limited.

One approach to provide multi-layered networking and security for Kubernetes is to use Canal (Flannel and Project Calico) with Istio for a microservices mesh.

STEP 1: INSTALLING CANAL

The first open source project we'll install is Canal, an installer script for two different open source projects: Flannel, an open source networking solution to build in scalable networking utilizing VXLAN-based overlays; and Project Calico, an open source networking and policy isolation project from Tigera. Using Calico, we'll be able to set up policies to create dynamic firewalls, which will enable us to greatly reduce our infrastructure's attack surface (at least on layers 3 and 4, i.e. filtering based on IP address and port).

Canal will install both of these projects and configure them to use Flannel's networking and Project Calico's isolation policies.

CANAL REQUIREMENTS:

- Kubernetes command-line tool has been installed (kubectl)
- Kubernetes cluster is configured to provide serviceaccount tokens to pods
- Kublets have been started with `--network-plugin=cni` and have `--cni-conf-dir` and `--cni-bin-dir` properly set
- Controller manager has been started with `--cluster-cidr=10.244.0.0/16 and --allocate-node-cidrs=true`

INSTALLING CANAL:

To install Canal for Kubernetes 1.6, run the following commands via kubectl:

```
$ kubectl apply -f https://
raw.githubusercontent.com/
projectcalico/canal/master/k8s-
install/1.6/rbac.yaml

$ kubectl apply -f https://
raw.githubusercontent.com/
projectcalico/canal/master/k8s-
install/1.6/canal.yaml
```

For additional installation guidelines, visit github.com/projectcalico/canal.

STEP 2: INSTALLING ISTIO

With Istio, we'll be able to add in numerous additional features (such as fault-injection, in-memory rate limiting, and log collection), while providing security and load balancing on Layers 5 and 7.

ISTIO REQUIREMENTS:

- Kubernetes command-line tool has been installed (kubectl)
- Access to a Kubernetes cluster

INSTALLING ISTIO:

With Kubernetes installed, installing Istio is as easy as downloading and extracting the file, ensuring you have the correct RBAC configuration for Istio, and installing via kubectl.

1. Download and extract the installation file: `curl -L https://git.io/getIstio | sh -`
2. Add the istioctl client to your PATH: `export PATH=$PWD/bin:$PATH`
3. Check Role-Based Access Control (RBAC) Settings (if error, continue to step 4, otherwise visit istio.io/docs/tasks/installing-istio.html for steps to configure RBAC)
4. Install Istio
   a. To install Istio with Auth module (additional security layers): `kubectl apply -f install/kubernetes/istio-auth.yaml`
   b. To install Istio without the Auth module: kubectl apply `-f install/kubernetes/istio.yaml`

STEP 3: CUSTOMIZING AND GOING FURTHER

To learn more about each of these projects and how to further customize and setup your secure network, visit the appropriate project's documentation:

Flannel: coreos.com/flannel

Calico: projectcalico.org

Istio: istio.io

Once you have Canal and Istio deployed, you will have the tools available to build a secure Kubernetes environment for your applications.

You can also learn about other tools and additional resources for Kubernetes networking and network policy here.

```
$ dctl cluster create  my-cluster [args]

$ dctl network create my-network [args]

$ dctl volume create my-volume [args]

$ kubectl create -f my-deployment.yaml

//done. taking rest of week off.
```

ENTERPRISE CONTAINER STRATEGY
Start to finish in 15 minutes

Why take months when you can get it done in 15 minutes?

Learn more at diamanti.com

Diamanti
Dev Done Fast. Ops Done Right.™

The Challenges of Moving Containers From Developer Laptops Into Production

Containers are taking off with developers, but moving the application from development on a laptop to production in the datacenter is a huge IT challenge. The challenges include planning, tool selection, integration into the network, persistent storage, and everything else it takes to run the infrastructure after day one.

Why? Containers break legacy networking and storage infrastructure and create time and labor-intensive considerations for operators:

Which Software stack? Choosing a reliable, efficient, interoperable set of tools is daunting. There are dozens of container technologies with more emerging by the day.

What is my network model? Most enterprises going the DIY route immediately struggle with container networking's concepts of port mappings, overlays, and requirements of L3 all the way to the endpoint, with a host of interoperability challenges.

How do I provide persistent storage? Legacy scale-up storage arrays don't fit modern scale-out containers. Delivering performance of databases and key value stores at scale to containers has ops teams scrambling to deliver persistent storage.

"Diamanti lets us focus on building and deploying unique media offerings faster than ever." -NBCUNIVERSAL

WRITTEN BY JEFF CHOU
CEO, **DIAMANTI**

PARTNER SPOTLIGHT

Diamanti Container Platform

Diamanti
Dev Done Fast. Ops Done Right.™

Purpose-built bare metal container platform for moving your applications from development to production.

CATEGORY
Container Platform

NEW RELEASES
Quarterly

OPEN SOURCE
Yes

CASE STUDY
What if you could reduce your container production deployment cycles from a dedicated team of specialized engineers and a 9-12 month schedule to just 15 minutes and a single administrator? With Diamanti, you can. We give you a reliable, performant, simple container strategy.

Diamanti is the first container platform with plug and play network and persistent storage that seamlessly integrates the most widely adopted software stack - standard open source Kubernetes and Docker - so there is no vendor lock-in. QoS on network and storage maximizes container density. 24x7 support also allows you to focus on building applications instead of building and supporting the infrastructure.

STRENGTHS

- No vendor lock-in, seamless integration with Docker and Kubernetes
- Easy plug-and-play with your existing network
- Bare metal containers... no hypervisor tax!
- 24x7 full stack support
- Up to 8x infrastructure consolidation reported by Diamanti users
- 3-5x performance gain

WEBSITE diamanti.com **TWITTER** @diamanticom **BLOG** diamanti.com/blog

Containers are the New Build Artifact

QUICK VIEW

01 Containers are a convenient mechanism for deploying software artifacts with all of their necessary dependencies bundled into them.

02 You can use your build system to create container images and publish them to a private container registry. Once centralized, these images can be used by any environment to run the container.

03 Images can be versioned by using image tags — this can include both the artifact version and other base image attributes, like the Java version, if you need to deploy in various permutations.

BY **TODD FASULLO**

SOFTWARE DEVELOPMENT PROFESSIONAL, **SMARTSHEET.COM**

We can all acknowledge that containers are gaining traction and rapidly becoming a common development tool. Containers make it easy to download and run pre-built images of the software components we use every day in our development and test environments (and maybe even in production environments if we are really cutting edge). But how well-integrated are containers in our own build and deployment systems? It is challenging to ensure consistency between our development and test environments. And the local development environments our developers are running can be even harder to verify to ensure the correct versions of our applications and services. And most importantly, we have to ensure that the testing is being performed against the exact same code and environment that will eventually be deployed to production.

If we think back to when we were first developing centralized build systems, we dealt with many of these same issues. We wanted to ensure everyone was using the same X.Y.Z version of an artifact, and that what was compiled into the artifact was exactly what was checked into our source control repository. Centralized build systems solved many of these problems and made sure that only code from source control repositories was compiled into artifacts. The artifacts were then published to the central artifact repositories everyone pulled from. This solution removed the need for individual developers to compile all artifacts on their local machines, share them via a central file share, or email them to their co-workers. In addition, our developers only needed to build and test the components of the system they were directly working on.

The DevOps movement of the past decade then started to address this for the rest of the environments our artifacts were running in. These systems would be responsible for everything else running in those environments beyond the artifacts we are compiling. This controlled the version of Java running on our systems, the Java Cryptography Extension (JCE) policy files, the proper configuration files, etc. It helped make sure our artifacts had the correct version of all the other dependencies installed and set up correctly in their runtime environment.

Maybe you are already using a configuration management tool like Puppet, Chef, Ansible, etc. to set up and maintain these dependencies on long-running hosts. These tools are terrific and have greatly improved our ability to set up and maintain long-running environments consistently. They have also accomplished a major goal of the Infrastructure as Code movement — ensuring that all of our configuration is checked into source control and managed by a release process just like our build artifacts (it is all checked in, right?). But now we have a configuration tool that is likely separate from our deployment process. Which means, now we have to manage both a configuration management system and the deployment process.

What if we evolved our thinking of what the build artifact is? How about if we bundle the build artifact with the runtime environment and everything it needs to run as a stand-alone deployment artifact? Container images are a perfect solution for this — we can bundle our artifacts and all of the other dependencies into a single image. Our build systems can create container images and push them to a central container registry. Docker has their own open-source private registry implementation, and many of the existing artifact repositories like Artifactory and Nexus have added the ability to manage Docker images. Once an image is published to a container registry, we can then share and deploy the exact same image to every environment — local dev, QA, and production. Need to support multiple versions of a runtime component like the Java Runtime Environment (JRE)? We can build different container images for each of our supported versions of Java. In Smartsheet's case, we have set up multiple base images for the different versions of Java we currently run. To update from Java 8.0.121 to Java 8.0.131, we switch our default image and generate new container images from that specific version.

But hold on a second, you say. You are telling me to build a single container image that can be deployed to any environment? How do I set up environment-specific attributes like database connection settings, secrets, etc? Externalizing your configuration is already an important aspect of building a proper Twelve-Factor App. Containers provide a number of options for managing the external configuration of the container image — we can mount files or directories from the host, use environment variables, access external configuration stores (database, Consul/Vault), use the new Docker secrets feature, etc.

Storing data (log files, database data, etc.) is an obvious challenge many quickly encounter when they first start running containers. I'm sure you can find many articles focused entirely on this topic. Simple solutions include mounting directories from the host file system into the container to store persistent data. As an example, a MySQL server could mount the /var/lib/mysql directory (where the data files are stored — NOT the executable files) from a host folder. This still allows you to change the version of MySQL you are running by stopping the container with the old version and then starting the new version with the same data directory.

Services with stateful data like databases are often a lower priority for migrating to containers. Eventually, the tools and ecosystem will better evolve to support them. We are starting to see this with tools like Kubernetes that expose persistent data primitives across clusters in a cloud environment like Google Container Engine. Our technologies will also evolve to better support containers with cloud-native distributed data stores like CockroachDB.

If you agree that containers could be a useful deployment artifact, you can start by generating them as part of your build pipeline. Containers can easily be built with command line tools that are easy to integrate into existing build systems. At Smartsheet, we use a Jenkins build environment that compiles artifacts and then executes downstream jobs, which generate container images with the build artifact. Each artifact is compiled using an array of base container images. This generates a container image for each supported version of Java, Tomcat, and other services we want to be able to deploy it with.

These containers are then tagged with the version of the build artifact, plus information about the base image, to allow deployment of each container image. We also tag images with easy defaults, so if a user wants to run the current version of a container, they can use the image `<private registry>/servicefoo`. For a specific version of a build artifact with the default version of Java, one would specify the image `<private registry>/servicefoo:1.5.1`. For a specific version with a non-default version of Java, they would use the image `<private registry>/servicefoo:1.5.1-java8u121`.

In the words of a co-worker, containers are just like a Go application — a statically linked image containing everything we need to execute at runtime. Our container registry provides a library of images that we can deploy to any local dev, QA, or production environment. Just provide a Linux kernel with a container runtime, and our application will have everything it needs to run. To walk through a very simple example, take a look at the Docker Container Java Hello World GitHub example. After all, containers are the new build artifact.

Todd Fasullo is a long time member of the Smartsheet Engineering team, having joined in 2006 prior to the launch of their very first offering. Most of Todd's time is dedicated to designing and scaling Smartsheet's application to ensure a seamless experience for each of Smartsheet's 100,000 paid accounts. When not improving the Smartsheet back-end infrastructure you can find Todd improving development/build/testing infrastructure which has been running on Docker since early 2014 (originally on version 0.9). In his spare time you can find Todd tooling around the Pacific Northwest mountains with his family by bike, ski, or on foot.

WHAT ARE YOUR DOCKER CONTAINERS DOING?

“Sysdig gives my team unprecedented visibility into our applications. We're using it extensively to help us solve complex issues in our most innovative products” – Adam Hertz, VP of Engineering at Comcast

We’re about to make your life easier.

Enterprises are moving to Docker containers for faster software development that promises developer agility, software portability, and scalable microservices. That’s all good news.

Here’s the bad news: Docker environments get harder to operate, because microservices and containers break legacy monitoring tools. Containers are great for developers, but they will wreak havoc on your DevOps team.

Worried? Relax, we got your back.

Sysdig is the first and only solution that can natively monitor Docker environments. Sysdig ContainerVision™ provides request-level visibility inside containers without invasive instrumentation. This approach overcomes the limitations of your old monitoring tools, and at the same time makes monitoring Docker simpler and more robust. All that means you can sleep at night when Kubernetes is auto-scaling your apps. Go on, press the snooze button.

Visit sysdig.com and find out!

sysdig

What's Inside Your Containers? Sysdig, Purpose-built for Containers

More and more, companies are depending on their software for revenue generation and differentiation. And in that case, software running in production requires deep visibility for monitoring, alerting, and troubleshooting to ensure a great customer experience, high performance, and uptime.

Enterprises are moving to containers for the promise of faster, more effective development that quickly creates more value in software. The container movement promises developer agility, software portability, and dynamic scaling of resources. They also represent the building block for microservices, which in turn promise more of these benefits.

Yet the operations of these environments get harder and more complex, because microservices and containers break legacy monitoring and analytics tools. These dynamic environments, often built on black-box containers, need a different approach to understanding how your application is doing.

How do you see not just containers, but the services they comprise? How do you see inside your containers, to see how your applications are performing? Can you actually determine what your containers are doing?

"Sysdig is the first and only solution that is purpose-built for container native monitoring and troubleshooting."

OUR SOLUTION:
Sysdig is the first and only solution that can natively monitor any infrastructure and app, including container-based ones. We are creating solutions delivering monitoring, alerting, and troubleshooting in a microservices-friendly architecture. Our open source Sysdig technology has attracted a community of over a million developers, administrators, and other IT professionals looking for deep visibility into systems and containers. Learn more at sysdig.com

WRITTEN BY APURVA DAVÉ
VP MARKETING, **SYSDIG**

PARTNER SPOTLIGHT

Sysdig Monitor

sysdig

"The kernel-level approach that Sysdig uses is pretty powerful. We see everything, without sidecar containers, even when Kubernetes moves things." – BRIAN AZNER, DIR. ENG., MAJOR LEAGUE SOCCER

CATEGORY
Container Monitoring and Troubleshooting

NEW RELEASES
Continuous

OPEN SOURCE
Open source and commercial solutions.

CASE STUDY
WayBlazer brings artificial intelligence to the travel industry. They analyze travelers' searches, yielding highly personalized travel recommendations for customers.

Wayblazer runs AWS with EC2, Docker, and Kubernetes. They have different Kubernetes prod, test, and project-specific namespaces that they need to monitor, at a high level, then by service, infrastructure, and application.

Sysdig has helped reduce costs and increase efficiency. Sysdig is the out-of-the-box solution they wanted, built for container environments. Once Sysdig was deployed and instrumented, it immediately started bringing in tagging from Kubernetes and gathering metrics. Sysdig provides the service, infrastructure, and application metrics they need in one place and enabled them to visualize anything they want.

STRENGTHS
- ContainerVision: Deep, request-level visibility inside containers without invasive instrumentation.
- Service-Oriented Performance Management: Measures services, infra, & app performance
- Application-Intelligent Monitoring: Auto-discover apps: no plug-ins, no config
- Trace-driven Troubleshooting: Dashboarding and alerts integrated with deep troubleshooting.
- Automatic leveraging of orchestration metadata

NOTABLE CUSTOMERS
- Comcast
- McKinsey
- UK Government
- TD Bank
- Major League Soccer
- Cisco

WEBSITE sysdig.com **TWITTER** @sysdig **BLOG** sysdig.com/blog

DZone

QUICK VIEW

01 Understanding the challenges and complexity when building services architectures.

02 How a service mesh fits with containers.

03 How a service mesh can help you manage and operate your services architecture.

Application Routing With Containers

BY **CHRISTIAN POSTA**
CHIEF ARCHITECT CLOUD APPLICATIONS, **RED HAT**

Containers have changed how we think about building, packaging, and deploying our applications. From a developer's perspective, they make it easier to package an application with its full set of dependencies and reliably recreate that application on another developer's workstation. It also allows us the more reliably deliver applications from dev to test to production (possibly in a CI/CD pipeline). Lastly, in this world of microservices, containers help deliver microservices to production at scale. With the move to cloud-native applications and services architectures like microservices, we gain some advantages in our deployment and management infrastructure, but our applications need to be designed with different principles in mind compared to traditional design: design for failure, horizontal scaling, dynamically changing environments, etc. Interestingly enough, services implemented with these considerations in mind invariably find themselves dealing with more complexity in the interactions between services than the services themselves. Can containers help here?

COMPLEXITY HAS MOVED TO SERVICE INTERACTION

First, we should decide what the problem is and how this complexity in the interactions between services manifests itself. Services will need to work with each other in a cooperative way to provide business value and thus will need to communicate. In these cloud architectures, this communication will happen over the network. This is the first source of complexity that traditional applications with colocated components don't usually have to confront. Any time a service has to make a call over the network to interact with its collaborators, things can go wrong. In our asynchronous, packet-switched networks, there are no guarantees about what can and will happen. When we put data out onto the network, that data goes through many hops and queues to get to its intended destination. Along the way, data can be dropped completely, duplicated, or slowed down. Moreover, these behaviors make it difficult to determine whether our communication with our collaborators is failing/slow because of the network or because the service on the other end has failed/is slow. This can lead to unsafe consequences, like services unable to deliver service to their customers, collaborations partially succeeding, data inconsistencies between services, and more. Related to problems that occur because of network failure/degradation (or perceived failure/degradation) are things like, how does a service find and talk to its collaborators? How does it load balance across multiple instances of its collaborators?

When we build these cloud-native services with containers, we now need to account for the complexity introduced by communication over the network. We need to implement things like service discovery, load balancing, circuit breakers, timeouts, and retries so that our services stay resilient in the face of this uncertain network behavior. This sounds like a lot of responsibility for our applications. We could create reusable libraries to help with this. Indeed, that's the approach many of the big internet companies took. Google invested massive engineering work to implement an RPC library that helps with these things (Stubby, now gRPC). Twitter did as well with their Finagle framework. Netflix was even nice enough to open source their efforts with their Netflix OSS libraries like Ribbon, Hystrix, and others. To make this work, we need to restrict our frameworks and languages to only those for which we can implement and maintain these cross-cutting concerns. We'd need to re-implement these patterns for each language and framework we'd like to support. Additionally, every developer would need the discipline to apply these libraries and idioms consistently across all the code they wrote. In many ways, folks like Netflix had to write these tools because they had no other choice; they were trying to build resilient services on top of IaaS cloud infrastructure. What choices do we have today?

CONTAINER PLATFORMS

For basic service discovery and load balancing, we should be able to leverage our container platform. For example, if you're packaging your application as Docker containers and you're using Kubernetes, things like load balancing and basic service discovery are baked in. In Kubernetes, we can use the "Kubernetes service" concept to define application clusters (each instance running in a container or Kubernetes "pod") and assign networking (like virtual IPs) to these clusters. Then we can use basic DNS to discover and interact with the cluster of containers even if the cluster evolves over time (addition of containers, etc).

SERVICE MESH FOR CONTAINERIZED SERVICES

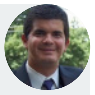
What if we could implement these resilience concerns and more across our services architectures without requiring language and framework-specific implementations? That's where a "service mesh" fits into the picture. A service mesh sits between our services and solves these issues without having to use frameworks or libraries inside the application. With a service mesh, we introduce application proxies that handle communicating with other services on behalf of our application. The application or service talks directly to the proxy and is configured with appropriate timeouts, retries, budgets, circuit breaking, etc. for communicating with upstream services. These proxies can either be implemented as shared proxies (multiple services use a single proxy) or application-specific "sidecar" proxies. With a sidecar proxy, the proxy is deployed alongside each instance of the service and is responsible for these horizontal concerns; that is, the application gains this functionality without having to instrument their code directly.

Linkerd and Lyft Envoy are two popular examples of proxies that can be used to build a service mesh. Linkerd is an open-source project from startup Buoyant.io, while Envoy is an open-source project from ride-hailing company Lyft. In a container environment, we can implement sidecars by either deploying the proxy in the same container as your application or as a sidecar container if you can specify container-affinity rules like with Kubernetes pods. In Kubernetes, a pod is a logical construct that considers an "instance" to be one or more containers deployed together. Implementing sidecar proxies in Kubernetes becomes straightforward.

With these sidecar (or shared) proxies in place, we can reliably and consistently implement service discovery, load balancing, circuit breaking, retries, and timeouts regardless of what's running in the container. With containers, we abstract away the details of the container for the purposes of uniform deployment and management, and with a service mesh, we can safely introduce reliability between the containers in a uniform way. Since these application proxies are proxying traffic, doing load balancing, retries, etc, we can also collect insight about what happens at the network level between our services. We can expose these metrics to a central monitoring solution (like InfluxDB or Prometheus) and have a consistent way to track metrics. We can also use these proxies to report other metadata about the runtime behavior of our services, including things like propagating distributed tracing to observability tools like Zipkin.

Lastly, we can introduce a control plane to help manage these application proxies across the service mesh. For example, a newly announced project, Istio.io, provides just that. With the control plane, not only are we able to understand and report what's happening between our services, we can control the flow of traffic as well. This becomes useful when we want to deploy new versions of our application and we want to implement A/B style testing or canary releases. With a control plane, we can configure fine-grained interservice routing rules to accomplish more advanced deployments.

Containers enable a new paradigm of cloud-native applications and container platforms help with the management and deployment of those containers. From a services architecture point of view, however, we need to solve some of the complexity that has now been moved between our services. Service meshes aim to help with this and application proxies help remove horizontal, cross-cutting code (and their dependencies) from our application code so that we can focus on business-differentiating services. Containers and container environments help us naturally implement this service-mesh pattern.



Christian Posta (@christianposta) is a Principal Architect at Red Hat and well known for being an author (Microservices for Java Developers, O'Reilly 2016), frequent blogger, speaker, open-source enthusiast, and committer on Apache ActiveMQ, Apache Camel, Fabric8, and others. Christian has spent time at web-scale companies and now helps companies creating and deploying large-scale distributed architectures - many of which are now called microservices-based. He enjoys mentoring, training, and leading teams to be successful with distributed systems concepts, microservices, DevOps, and cloud-native application design.

DO YOU REALLY KNOW WHAT'S INSIDE YOUR CONTAINERS?

START MANAGING YOUR CONTAINERS WITH THE ONLY UNIVERSAL, SECURE, ENTERPRISE-READY SOLUTION

DOWNLOAD YOUR FREE TRIAL

THESE COMPANIES ALREADY RELY ON JFROG TO MANAGE THEIR CONTAINERS

amazon Google UBER CISCO ORACLE UBS slack

adidas Microsoft NETFLIX facebook vmware Walgreens BOEING

BEST BUY FUJITSU citi VISA SAP Morgan Stanley HALLIBURTON airbnb EA

JFrog

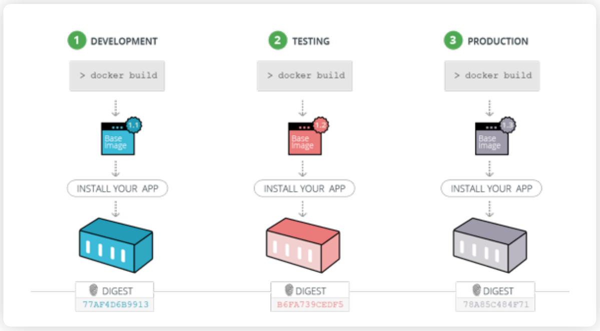
Taking Docker to Production is a Matter of Promotion, Not Rebuilding

Docker has enjoyed meteoric growth over the last few years and is used by many software development organizations. However, while this technology is widely used in early phases of the development pipeline, it is rarely used in production. The question is: "why?"

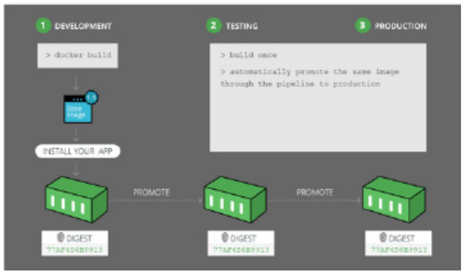
Consider a typical software delivery pipeline. A build from the development CI server is promoted through the pipeline, provided it passes the respective quality gates at each phase before it gets to production. This guarantees that the same binary originally built is the one deployed to production. So why odesn't it work like this for Docker?

It's because building a Docker image is so easy: you just run `$ docker build`. Consequently, the image I rebuilt at each phase of the pipeline. Why is this a problem? The answer is dependencies. Because each phase of the delivery pipeline is executed at a different time, you can't be sure that the same version of each dependency used for the build in the development version was also used when building the production version, and (almost) every line in your Dockerfile is resolving a dependency! From the `FROM` line, via `apt-get` or `yum install` and all the way to `COPY` or `ADD` your application files – we get some versions of some files, hoping they are the right versions for us. It's fragile and the chances we get the exact same set of files every build, let's be frank, are slim.

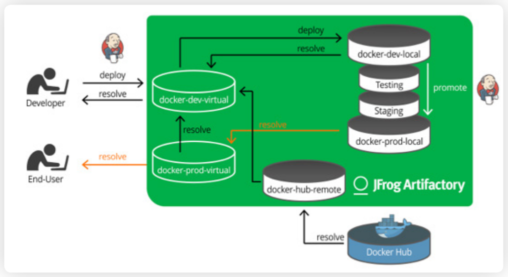
So you end up with this:



While what you should be doing is this:



You should be promoting your development build as an immutable and stable binary, through the quality gates to production. The problem is that a Docker tag limits you to using one registry per host. How can you build a promotion pipeline if you can only work with one registry? The answer is by using Artifactory virtual repositories.



An Artifactory virtual repository aggregates any number of repositories into a single entry point for both upload and download of Docker images. This allows us to:

- Deploy our build to a virtual repository which functions as our development Docker registry
- Promote the build within Artifactory through the pipeline using testing and staging repositories
- Resolve production ready images from the same (or even a different) virtual repository now functioning as our production Docker registry
- You can even take it a step further and expose your production Docker image to customers through another virtual repository.

That's it. You build a Docker image, promoted it through all phases of testing using virtual repositories, and once it passed all those quality gates, the exact same images you created in development is now available for download in your production servers.

WRITTEN BY BARUCH SADOGURKY - DEVELOPER ADVOCATE, **JFROG**

Is Docker the End of Traditional Application Release Management?

QUICK VIEW

01 Application release is more than packaging the deliverables into an easy-to-manage format.

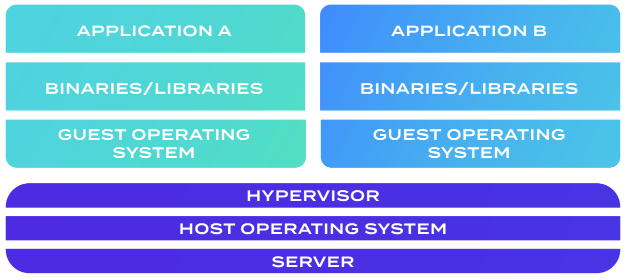
02 Integration with external solutions is an essential component of application release.

03 Automating application release provides long-standing benefits, such as accelerated releases, that are compliant with internal standards and lower operational risk.

BY **LARRY SALOMON JR.**
TECHNICAL EVANGELIST

Ever since its release in 2013, Docker has quickly catapulted into the enviable position of being the darling of every operations manager's eye. If you've been vacationing on Mars since then, here is what you've missed.

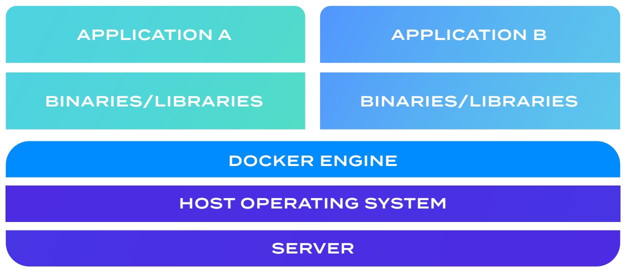
Docker is a partitioning capability within the address space of an operating environment. By allowing the partition to use the host OS directly, even though that OS resides outside of the partition (known as a container), the start-up time is substantially reduced, as is the resource requirements for the management of the container (those of you who are familiar with z/OS will find this concept to be "somewhat familiar").



Financial people love this because the cost of acquiring licenses for the operating environment can be substantially reduced since, theoretically, every component that is not part of the application itself can reside outside of the container. This means only one Windows license needs to be procured versus one per VM (which is the required process if Docker is not used).

THE CONCEPT IS SIMPLE, BUT HOW DOES IT WORK?

Essentially, a special file (called a **dockerfile**) contains one or more instructions on how a container is to be created. The dockerfile is used as part of a process to generate the container on the file system, which can contain as little as a single application and its associated binaries. This container (a subdirectory in the file system) is then transferred to the target environment as any set of files would be and is started there using the Docker run time, which can be invoked via the command line interface or an API (typically REST based, but there are other implementations).



System Administrators love this because containers are easy to deploy (XCOPY anyone?) and maintain (REST interfaces can be easily integrated into any modern Infrastructure Management platform).

THE REQUIREMENTS OF AN ENTERPRISE-CLASS RELEASE AUTOMATION SOLUTION

Unfortunately, this concept falls down when people try to use it as a substitute for true application release management. More specifically, we can describe application release management using five of the six question words that everyone learned in high school English:

Who. Not just anyone in an organization should be able to deploy an application to an environment. In fact, even for those allowed to do so, there should frequently be others who have to approve the deployment decision.

What. For organizations that truly embrace the concept of business agility, deploying a complete application every time is unacceptable. Artifacts deemed as low risk (e.g. content updates) may be deployed immediately while higher risk artifacts will be queued up to be released after a lot of testing and other validations. Docker falls into this category but has limitations, which will be touched on below.

Where. The target environment of a deployment is frequently different from every other possible target environment that an application will touch during its journey from development to production. These differences are typically addressed by making changes to the configuration of the application after it has been deployed.

When. Release windows are not a new concept. Even in non-production environments, a case for establishing a release window could be made since environments are often shared among multiple teams within the same function or even across functions (i.e. testing and development may use the same environment).

How. Probably the most problematic process to fully integrate into an organization's operational capabilities, the process of deploying an application is far more than simply understanding how to install and configure it. For example, integration with an ITSM application to ensure that change requests have been entered and are in the correct state has to be incorporated into the process of deployment so that the state of the operating environment is always well understood. This is discussed in more detail below.

Of the five question words above, Docker only addresses one of them, and not in the most effective manner possible. Consider the scenario of a well-known bank based in Europe. They currently have in excess of a thousand production releases every month. This was accomplished by recognizing that not all production releases are high risk. In the example under **What**, it was noted that certain types of artifacts had minimal impact. As a result, the release of those artifact types could be expedited, which helped ensure that this bank's customer facing assets were always meeting the needs of their clientele.

If they were using Docker, however, the entire application would need to be rebuilt regardless of the types of artifacts that were actually approved for production release. The risk that unapproved binaries could be released into production is simply unacceptable for most companies. And this is only for one of the five items above - Docker does nothing to address the other four.

APPLICATION RELEASE MANAGEMENT IS MORE THAN THE APPLICATION

It is tempting to think of application release management in terms of the application only, while forgetting that the application, from the business's perspective, is part of the bigger picture. In the **How** section above, ITSM was mentioned, but this is not the only technology with which the release process must integrate. In fact, the SDLC toolchain is littered with a whole host of solutions that fit specific needs: Hudson and Jenkins for Continuous Integration; Git and Subversion for Source Code Management; Nexus and Artifactory for Artifact Management; Chef and Puppet for Configuration Management; etc.

Additionally, the process of releasing an application during its entire lifetime often includes governance that is specific to the process but isn't part of the process, *per se*. However, these stages through which the build must traverse are essential to ensuring minimal risk while releasing with a high cadence. They include approvals, validations, and other types of activities.

AUTOMATION IS THE KEY TO EVERYTHING

Everything we've spoken about is critical to an application release, but, in the end, results are what matter. End users need new functionality, and the speed at which the application development team can both produce the new functionality and deliver it to the end users determines how quickly that new functionality will translate to additional revenue.

Furthermore, repeatability in the process ensures a much higher rate of application deployment success. Conversely, failed deployments cost your company money while production instances of your application are down during triage and remediation. Two studies by major analyst firms in the past 3 years determined that the cost amongst Fortune 1000 companies for application outages that were due to change, configuration, or other handoff related issues was in the range of $200k-400k per hour.

Each of the tools in the previous section has relevance within only a small portion of the application build and release process. Similarly, Docker addresses the management of the artifacts associated with application release in such a way that it eases the deployment of those artifacts, but that's it. The coordination of these and other solutions' capabilities is something that must be managed by an orchestration solution, specifically one that was purpose built for application release automation.

SUMMARY

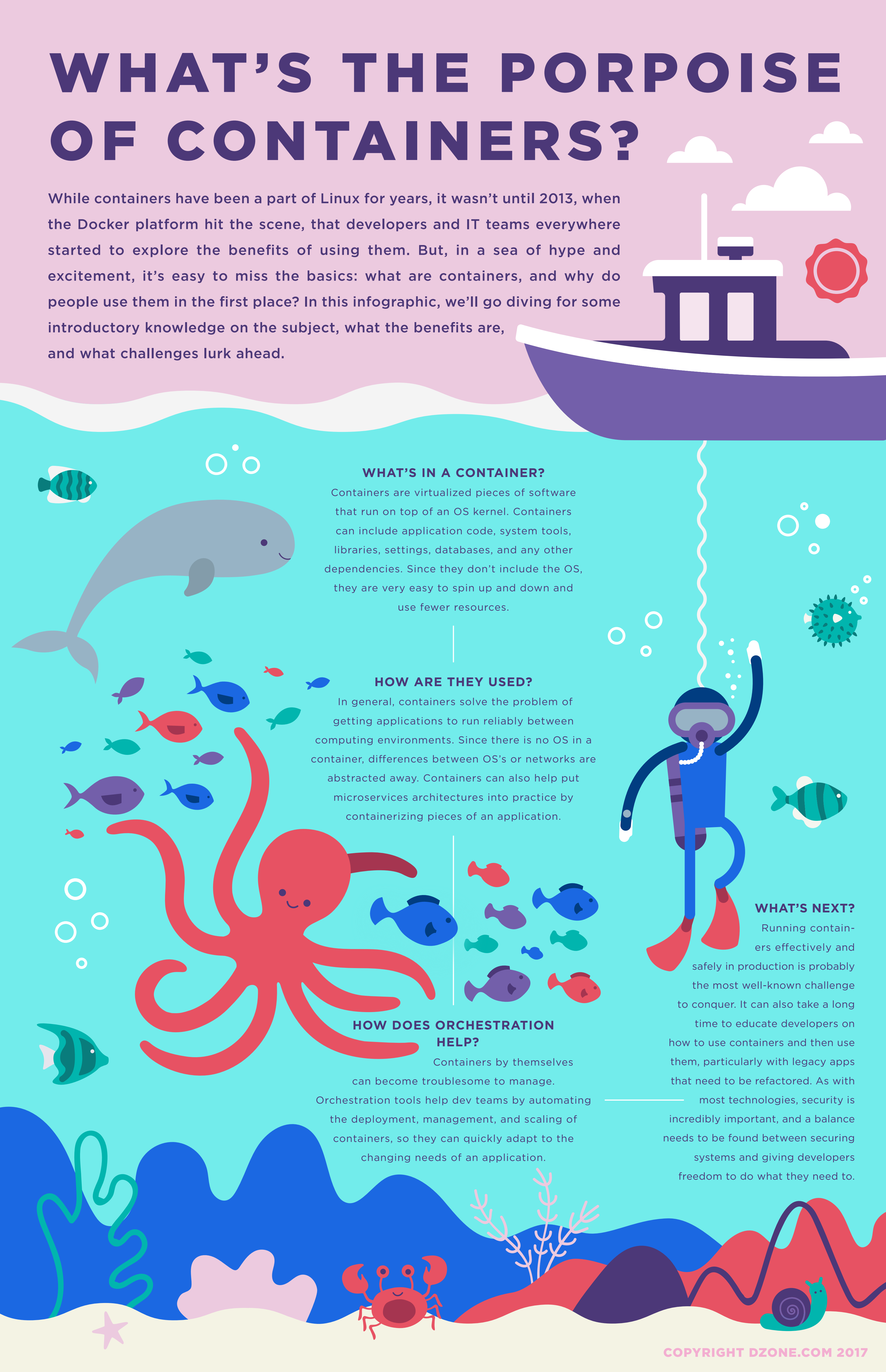
To summarize, Docker is an exciting technology that should be viewed as simply another mechanism that exists within the greater whole of the application release cycle. But it should not be viewed as a replacement for a well-defined methodology that not only includes the "what," but also includes "who," "where," "when," and "how."

Investing in an enterprise-class automation solution built specifically to automate the release of your mission-critical applications will not only increase the speed with which you deploy your applications but will also increase the rate of your company's digital transformation as more applications are deployed using the solution, providing dividends for years to come.

Larry Salomon Jr brings 18 years of IT experience focused on application development and delivery. Coupled with his strong focus on the business relevance of technology, Mr. Salomon is a recognized thought leader in the application delivery and IT automation spaces. You will find him both in several LinkedIn discussion groups and in his blog on business related topics (larrysalomon.blogspot.com). Mr. Salomon holds a Bachelor of Computer Science degree from Clemson University and holds certifications in DevOps v2, ITIL v3, and Six Sigma.

WHAT'S THE PORPOISE OF CONTAINERS?

While containers have been a part of Linux for years, it wasn't until 2013, when the Docker platform hit the scene, that developers and IT teams everywhere started to explore the benefits of using them. But, in a sea of hype and excitement, it's easy to miss the basics: what are containers, and why do people use them in the first place? In this infographic, we'll go diving for some introductory knowledge on the subject, what the benefits are, and what challenges lurk ahead.



WHAT'S IN A CONTAINER?

Containers are virtualized pieces of software that run on top of an OS kernel. Containers can include application code, system tools, libraries, settings, databases, and any other dependencies. Since they don't include the OS, they are very easy to spin up and down and use fewer resources.

HOW ARE THEY USED?

In general, containers solve the problem of getting applications to run reliably between computing environments. Since there is no OS in a container, differences between OS's or networks are abstracted away. Containers can also help put microservices architectures into practice by containerizing pieces of an application.

HOW DOES ORCHESTRATION HELP?

Containers by themselves can become troublesome to manage. Orchestration tools help dev teams by automating the deployment, management, and scaling of containers, so they can quickly adapt to the changing needs of an application.

WHAT'S NEXT?

Running containers effectively and safely in production is probably the most well-known challenge to conquer. It can also take a long time to educate developers on how to use containers and then use them, particularly with legacy apps that need to be refactored. As with most technologies, security is incredibly important, and a balance needs to be found between securing systems and giving developers freedom to do what they need to.

Drive Agility, Empower DevOps with Automic V12

Automic™
RELEASE AUTOMATION

Automic is a leader in Automation and DevOps, empowering digital transformation at large enterprise IT organizations across all industry verticals. Customers rely on our Application Release Automation product to drive the agility, speed and reliability required for businesses to stay competitive in the digital age. The goal is to provide consistent, repeatable and auditable deployments across the Continuous Delivery pipeline.

Find out more at automic.com

Automic, the leader in business automation software owned by CA Technologies, helps enterprises drive competitive advantage by automating their IT and business systems - from on-premise to the Cloud, Big Data and the Internet of Things. With offices worldwide, Automic powers 2,700 customers across all industry verticals including Financial Services, Manufacturing, Retail, Automotive and Telecommunications.

Automic™.com | **@automic**

3 Ways To Bolster Your DevOps and Continuous Delivery Initiatives Using Release Automation

1. VISUALIZE YOUR DEPLOYMENT PROCESSES AND MONITOR THEM LIVE

A powerful workflow engine lies at the core of the Automic platform, so the first benefit new users get is a visual workflow creator. Whether you start by simply wrapping Automic around your existing scripts, or you go in and create your first workflows using our guided wizards, you'll immediately start to (literally) see the execution and any rules in progress; what happens in parallel and what doesn't; and the output of each step, on any number of targets.

2. USE BUILT-IN ROLLBACK

Ensure that you employ a rollback framework that enables package developers to handle rollback rules as part of the action. This way rollback does not have to be designed by you as part of a deployment. Most Automic-provided actions are auto-rollback enabled and users developing their own actions can leverage the rollback framework to make their own actions equally robust.

3. ORCHESTRATE CONTAINER SERVICES LIKE DOCKER AND AUGMENT THEM WITH METADATA DRIVEN RULES

Containers are on the rise, and Automic supports Docker-based container service orchestration. A key use case for Automic customers who run container services is the augmenting of Docker files with metadata. For example, data related to the capacity required for each service, which is used to determine the underlying VM capacity and orchestrate deployments of hybrid applications and services that utilize container and non-container based services.

You can read the full article here.

WRITTEN BY RON GIDRON

PRODUCT MARKETING DIRECTOR OF RELEASE AUTOMATION, **AUTOMIC SOFTWARE**

PARTNER SPOTLIGHT

Automic Release Automation v12

Automic™

Automic V12 has been designed for the hybrid enterprise to drive agility across operations, while empowering new DevOps initiatives.

CATEGORY	NEW RELEASES	OPEN SOURCE
Release Automation/ Service Orchestration	Bi-annually	Yes

STRENGTHS

- V12 allows agents to auto-update with zero business impact
- V12 removes maintenance windows
- V12 makes managing core business applications agile
- V12 drives intelligent insights across automation silos

CASE STUDY

TASC uses Automic Release Automation to simplify administration and gain extra flexibility by abstracting its process flows. The company relies on Automic to orchestrate their deployment pipeline, integrating with Sonatype Nexus, Maven, and Subversion (SVN) as they promote their apps through their JBoss-based development, QA, staging, and production environments.

"Automic has helped us grow and become agile for our customers, both internally and externally. It's also helped us grow into areas that, quite frankly, other companies and other organizations haven't been able to move into."

- TOM FLITTER, DIRECTOR OF APPLICATIONS AND INTEGRATION AT TASC

NOTABLE CUSTOMERS

- Bosch
- Netflix
- AMC Theatres
- ExxonMobil
- Vodafone
- NHS SBS
- General Electric

WEBSITE Automic.com **TWITTER** @automic **BLOG** automic.com/blog

5 Docker Logging Best Practices

BY **JEFFREY WALKER**
FOUNDER AND OWNER, **STARTUPLABS**

QUICK VIEW

01 Microservices have gotten a boost through Docker, which is the most popular "containerization" model for building independently deployable services.

02 Unlike traditional application logging, Docker logging effectively means not only logging the application and the host OS, but also the Docker service.

03 The major approaches to Docker logging are: app-based logging, using data volumes, the Docker logging driver, dedicated logging container, and sidecars.

Containers have become a huge topic in IT, and especially in DevOps, over the past several years. Simply stated, containers offer an easy and scalable way to run software reliably when moving from one environment to another. Containers do this by providing an entire runtime environment in one package, which includes the application, plus all dependencies, libraries and other binaries, and configuration files needed to run it.

Closely aligned with containers are microservices, which represent a more agile way of developing applications. A microservices architecture structures an application as a set of loosely coupled services connected via functional APIs that handle discrete business functions. Instead of a large monolithic code base, microservices primarily offer a "divide and conquer" approach to application development.

Leading the charge in the world of container infrastructures is Docker, a platform for deploying containerized software applications. The real value of containers is that they allow teams to spin up a full runtime environment on the fly. Docker is arguably the most influential platform today for getting businesses to adopt microservices.

Similar to how virtual machines streamlined software development and testing by providing multiple instances of an OS to end-users from one server, containers add an extra abstraction layer between an application and the host OS. The big difference is that containers don't require a hypervisor and only run one instance of an operating system; overall, this equates to far less memory and faster run time.

As with developing any application, logging is a central part of the process and especially useful when things go wrong. But logging in the world of containerized apps is different than with traditional applications. Logging Docker effectively means not only logging the application and the host OS, but also the Docker service.

There are a number of logging techniques and approaches to keep in mind when working with Dockerized apps. We outline the top five best practices in more detail below.

APPLICATION-BASED LOGGING

In an application-based approach, the application inside the containers uses a logging framework to handle the logging process. For instance, a Java application might use Log4j 2 to format and send log files to a remote server and bypass the Docker environment and OS altogether.

While application-based logging gives developers the most control over the logging event, the approach also creates a lot of overhead on the application process.

This approach might be useful for those who are working within more traditional application environments since it allows developers to continue using the application's

Logging Docker effectively means not only logging the application and the host OS, but also the Docker service.

logging framework (i.e., Log4j 2) without having to add logging functionality to the host.

USING DATA VOLUMES

Containers by nature are transient, meaning that any files inside the container will be lost if the container shuts down. Instead, containers must either forward log events to a centralized logging service (such as Loggly) or store log events in a data volume. A data volume is defined as "a marked directory inside of a container that exists to hold persistent or commonly shared data."

The advantage of using data volumes to log events is that since they link to a directory on the host, the log data persists and can be shared with other containers. The advantage of this approach is that it decreases the likelihood of losing data when a container fails or shuts down.

Instructions for setting up a Docker data volume in Ubuntu can be found here.

DOCKER LOGGING DRIVER

A third approach to logging events in Docker is by using the platform's logging drivers to forward the log events to a syslog instance running on the host. The Docker logging driver reads log events directly from the container's stdout and stderr output; this eliminates the need to read to and write from log files, which translates into a performance gain.

However, there are a few drawbacks to using the Docker logging driver: 1) it doesn't allow for log parsing, only log forwarding; 2) Docker log commands work only with log driver JSON files; 3) containers terminate when the TCP server becomes unreachable.

Instructions for configuring the default logging driver for Docker may be found here.

DEDICATED LOGGING CONTAINER

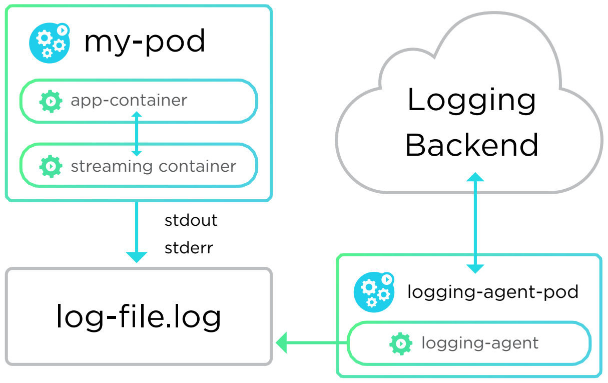
This approach has the primary advantage of allowing log events to be managed fully within the Docker environment. Since a dedicated logging container can gather log events from other containers, aggregate them, then store or forward the events to a third-party service, this approach eliminates the dependencies on a host.

Additional advantages of dedicated logging containers are: 1) automatically collect, monitor, and analyze log events; 2) scale your log events automatically without configuration; 3) retrieve logs through multiple streams of log events, stats, and Docker API data.

SIDECAR APPROACH

Sidecars have become a popular approach to managing microservices architectures. The idea of a sidecar comes from the analogy of a how a motorcycle sidecar is attached to a motorcycle. To quote one source, "A sidecar runs alongside your service as a second process and provides 'platform infrastructure features' exposed via a homogeneous interface such as a REST-like API over HTTP."

From a logging standpoint, the advantage of a sidecar approach is that each container is linked to its own logging container (the application container saves the log events and the logging container tags and forwards them to a logging management system like Loggly).



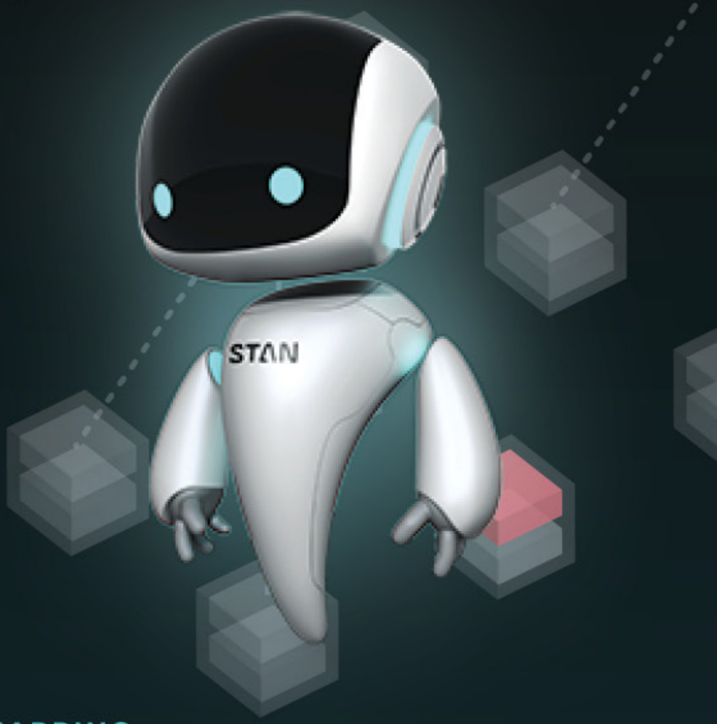
See image inspiration here

The sidecar approach is especially useful for larger deployments where more specialized logging information and custom tags are necessary. Though, setting up sidecars are more notably complex and difficult to scale.

Jeffrey Walker is the Founder and Owner of StartUPLabs, a Boston-based company that provides Strategic Advising, Prototyping, Web Design and Development, and Marketing Services to help guide and launch successful businesses. He has worked in IT for the past decade, including 3 years as an emerging technologies researcher for a Fortune 100 company in Boston. During 2009-2010 he served as founder and CEO of a small startup specializing in open source human-robot interaction (HRI). Jeffrey holds multiple master's degrees, including an Ed.M. from Columbia University.

INSTANA

The Six Pillars of Managing Containerized Applications



Containers have changed Applications forever, and that includes management tools. Managing containerized apps requires a new APM tool - one built on these six pillars of functionality.

CONTINUOUS AUTOMATIC DISCOVERY & MAPPING

With containers, the only constant is change. APM must automatically discover & map all application components; and automatically update the map when changes occur.

ONE-SECOND METRIC GRANULARITY

Containers can spin up and decommission in an instant. Metric granularity longer than one second leads to missing data and – worse – missing problems.

THREE-SECOND PROBLEM NOTIFICATION

Don't wait to see if a new container behaves properly. Get near real-time notification so that any problems can be rolled back or fixed immediately.

FULL STACK TRACING OF EVERY REQUEST

Every request is important in its own way – and every request has the potential to fail or slow down. To guarantee that every bad trace is captured for debugging, you must trace every single request.

FOCUS ON WHAT MATTERS

Deal with the sheer scale, complexity and dynamic nature of containerized apps by filtering out extraneous systems and requests, focusing only on the systems you're investigating.

USE MACHINE LEARNING FOR SERVICE QUALITY MANAGEMENT

As element counts grow into the hundreds or thousands, humans cannot practically manage application performance on their own. APM should learn which Key Performance Indicators (KPIs) to collect, when KPIs indicate service incidents, and determine the likely root cause of problems – all without human interaction.

To allow your Operations and Development teams to operate effectively, look for tools that have all six pillars – anything less will jeopardize your container projects. To learn more about how Instana can help you achieve better service quality, visit Instana.com

Continuous Discovery: The Key to Continuous Delivery

Agile development, CI/CD, and the use of containers create constant application change in code, architecture, and even which systems are running.

This constant change, especially in containerized applications, makes Continuous Discovery a must-have feature for in order for effective monitoring tools to:

- Discover and map the components that make up the application
- Automatically monitor the health of each component on its own and as part of the app

Configuring Agents is a key obstacle to achieving continuous discovery. To be specific, agent configuration has always been difficult, but dynamic applications make any configuration work obsolete almost as soon as it's complete.

The key to making continuous discovery work within monitoring tools is automation.

Automating agent configuration requires a change in the way agents are built and deployed, especially the way agents collect and transmit information through technology sensors, such as:

- Configuration data
- Events
- Traces
- Metrics

Agents should automatically recognize every component and deploy the proper monitoring sensors, automatically collect the right data, and provide a real-time health score.

This goes beyond code. Every technology component needs its own expert monitoring. So far at Instana, we've expanded that list to nine languages and almost 70 unique technologies.

Why is this essential for Continuous Delivery?

1. **Speed**: Nobody has time to configure (and reconfigure) tools with the rate of change.
2. **Real-time Mapping**: Containerized applications are constantly changing so data flow and interactions can't be known in advance. Application maps must be built in real time.
3. **Instant Feedback**: Results from changes (including deployments) should be known in seconds. A delay of even a few minutes could be devastating.

You've created an agile development process. You're investing in containers. Don't let your monitoring tools prevent you from achieving your ultimate goals of Continuous Delivery.

WRITTEN BY PAVLO BARON
CO-FOUNDER AND CHIEF TECHNOLOGY OFFICER , **INSTANA**

PARTNER SPOTLIGHT

Instana

INSTANA

Dynamic APM for Containerized Microservice Applications

CATEGORY
Application Performance Management for Containerized Applications

NEW RELEASES
Several times per year

OPEN SOURCE
No

CASE STUDY
To go mobile, a real-time service provider moved from a monolithic environment to containerized microservices. They got their scalability, but they now had to manage 25 different technologies. Both the APM and Infrastructure monitoring tools struggled to keep up with application changes.

Enter Instana. Instana automatically mapped the entire app and infrastructure in just 15 minutes, instantly providing health reports for each component. Metrics were provided along with all the meta information from Docker and Mesosphere.

DevOps finally has what they need to manage infrastructure, components, and the application from one place — without any human configuration. The best part? They're ready for the next big architectural change, too.

STRENGTHS
- Containerized application discovery and mapping
- Automatic health monitoring of 60+ technologies — with no-touch configuration
- Full stack tracing of every request through every technology component
- One-second granularity for real-time performance data
- AI-assisted service quality management and troubleshooting

NOTABLE CUSTOMERS
- Douglas Online
- Conrad.com
- DriveNow
- Booxware

WEBSITE instana.com | **TWITTER** @InstanaHQ | **BLOG** instana.com/blog

How Does IT Ops Spell T-R-O-U-B-L-E? Container-Driven Technology Diversity

BY **PAVLO BARON**
FOUNDER AND CTO, **INSTANA**

QUICK VIEW

01 Containers make it easier for Dev teams to create massively diverse application architectures.

02 Classic Ops teams aren't built to handle the polyglot of application languages or the sheer number of deployed technologies.

03 Automation is the only way Operations can keep up with the pace of Agile and CD.

04 Even Container Orchestration tools are missing key performance metrics to optimize user experience.

CONTAINERS ARE GREAT!

You already know this, since you're reading this report and this article, but even the most ardent fan of containers probably doesn't understand just what they've enabled in the world of application development and application operations.

Essentially, they make it easier to deploy almost anything. Yes, there are the efficiency benefits of containers, inching closer to using (and paying for) only the resources you need. But it's the ease of deployment of practically any application or IT system that can truly change the IT world. Unfortunately, being able to easily deploy almost anything means that it's especially easy to deploy almost EVERYTHING.

AGILITY + CONTINUOUS DELIVERY + TECHNOLOGY DIVERSITY = OPERATIONAL NIGHTMARE

While that ability, in itself, is nothing to worry about, the embraced concepts of extreme agility and Continuous Delivery from development teams — coupled with this insanely easy method of deploying any and everything — has put an onerous burden on the shoulders of IT Operations. As the CTO of a SaaS company, I live in this space between Development and Operations every day, since I manage both.

Now what do we mean by everything? Of course, there are hundreds of different technology components that are being used to build applications quickly, more efficiently, and more responsive to users. But in reality, most of us are using some combination of about 80 things: different languages, database systems, middleware app servers, messaging technology, storage, web servers, and much more.

From the Development perspective, when you couple the enabled ability to use whatever technology platform and/or language you want with the drive to do things faster and more continuously, you get each team (and each developer) making the optimal decision for themselves regarding technology usage. So, while 60 percent of development uses MySQL for their database, 20% might be using SQL Server, another 20% might be using MongoDB, and the other 20% might be using another three different databases.

For Operations, the Developer's "paradise" has become the Operator's nightmare. It will no longer be sufficient to have a MySQL expert on the team with everyone else understanding the basics of configuring and maintaining a MySQL Server. Instead, everyone has to be able to handle whatever might come up with a whole host of different database systems.

Now multiply that problem by the number of systems in operation: security, web server, app server, storage, messaging, transactions, directories, search tools, and more. With all of this mass diversity, how can Ops even hope to cope?

Automation isn't a panacea to fix all problems. But in this hyper-agile application world, it's an absolute requirement.

AUTOMATION IN OPERATIONS

Automation isn't a panacea to fix all problems. But in this hyper-agile application world, it's an absolute requirement. This isn't the automation of the nineties, when we were all hypothesizing about dark room IT Ops. No, this is automation that's required simply so your Operations team can actually do the things they need to do over the lifecycle of your applications.

What should be automated? Some would like to say "everything," but let's start with the most important aspects of Operations: the ability to deliver application services to end users at the proper scale and proper performance levels (or service levels). So, monitoring has to be one of the first things Operations should automate.

AUTOMATED CONTAINER MONITORING

Which brings us full circle back to those little containers. Not only do they make it easy to overrun the operations team with this mass diversity of systems, but they add a layer that's difficult to penetrate when it comes to understanding how the overall system (or application) is performing.

The first thing to automate is understanding what's actually running in a container (the app technology), how it fits within the overall application architecture (dependency map), and how both the container and the technology inside are executing their responsibilities to deliver the appropriate service levels.

The next step in automation is monitoring the health (again, both of the container and the thing that's running inside it). This is best done by machines, whether an expert system (understanding what should be measured, and what different metrics mean to the overall health of the system) or further down the AI path to machine learning, where the monitoring system gleans patterns of execution, determines causal correlation between end-user service levels and individual component measurements, and other systemic ways to handle what would be an impossible task for human operators.

SPEED KILLS (MONITORING EFFECTIVENESS)

The other thing that happens with containerized applications, especially in an agile/Continuous Delivery environment, is the rate at which containers are provisioned and destroyed as they are needed (and then not needed). Traditional monitoring tools (15-minute metrics, 1-hour notifications) and APM tools (1-minute metrics, 5-minute notifications) are going to miss anywhere from 10-50% of containers that even execute.

And while container orchestration tools are a way to better manage the deployment of individual (or even sets of) containers, these tools aren't considering the overall performance of the application, nor can they take into account outside influences (or resource suckers) that make their allocations suboptimal.

The embraced concepts of extreme agility and Continuous Delivery from development teams — coupled with this insanely easy method of deploying any and everything — has put an onerous burden on the shoulders of IT Operations.

A new breed of monitoring/performance management tools are needed that recognize both the overall system and the individual components running inside containers (and even a look at the containers themselves). These new tools should also be able to work hand-in-hand with orchestration and other container management systems to ensure that everything operationally works together to meet the purpose of the containerized apps in the first place — fantastic user experience.

Pavlo Baron is the co-founder and Chief Technology Officer of Instana. Pavlo is a 25-year veteran of IT, especially regarding application delivery and performance analysis. His experience includes being an enterprise architect and engineering lead at companies ranging from small to enterprise, such as Sixt and UniCredit. In addition to his career as a technologist and entrepreneur, Pavlo has authored a book on software support.

TECTONIC

by CoreOS

ENTERPRISE
UPSTREAM
KUBERNETES

CoreOS.com/Tectonic

Automating Operations With CoreOS Tectonic

No one is happy when roadblocks and delays stand in the way of getting your product to market. Much like a marathon runner who loses a race because they have to stop every mile to do the mundane task of tying their shoes, each time your business needs to slow down to manually service your IT infrastructure, the competition is gaining, or worse, passing you. CoreOS Tectonic provides automated operations for your container infrastructure, which drastically reduces time-to-market and reduces the risk of a service outage, all while freeing up your IT team to focus on providing value to your organization.

IT teams spend time and resources to perform infrastructure maintenance tasks, requiring personnel to focus on mundane, repeatable processes which provide no value.

The myopic view of automated infrastructure operations is that it will take jobs out of the IT organization, but it does no such thing: Tectonic augments the ops team's role. Automated operations liberate IT by automating time consuming, repetitive tasks and allow the whole organization to deploy as quickly as they innovate.

All companies are becoming more like software companies in that they need to deliver more and better products at a faster rate than was expected even just a few years ago. The move to cloud-native, containerized infrastructure delivers access to flexible and scalable environments, bringing web-based offerings within reach for many more organizations than before. By automating the functions of infrastructure management and maintenance with Tectonic, teams can adjust their focus away from their old tasks of maintenance and fire-alarm responses and onto innovating and creating value.

WRITTEN BY BY ROB SZUMSKI
TECTONIC PRODUCT MANAGER, **COREOS**

PARTNER SPOTLIGHT

Tectonic by CoreOS

Tectonic is the enterprise-ready Kubernetes platform that delivers scalable, resilient, and secure automated infrastructure

CATEGORY
Automated infrastructure

NEW RELEASES
Monthly

OPEN SOURCE
Yes

CASE STUDY
A large entertainment company is in the process of moving their entire infrastructure to Kubernetes, and has chosen CoreOS's Tectonic as their enterprise-ready Kubernetes platform of choice. They began with their web platform, which was containerized but with a clunky deployment mechanism. It took over 20 minutes to deploy, and the team had low confidence with each deployment. Since implementing Tectonic, deployment speed has improved 20 times, going from 20 minutes to 60 seconds, and it works every time, with a high degree of confidence. Tectonic has allowed their team to deliver software on a daily basis: The team does a standup in the morning, states their plans for the day's releases, and reconvenes at the end of the day to show the output.

Ultimately, Kubernetes and Tectonic empower the entertainment company's makers, creators and visionaries to continue to innovate and stay competitive in a fast moving industry. The entertainment company is fostering a space for their engineering and development teams to innovate and deliver great solutions to the market.

STRENGTHS
- Built on the latest upstream Kubernetes releases
- Easy-to-use installer for Kubernetes
- Intuitive, feature-rich dashboard for efficient infrastructure management
- Automated operations providing software updates for the latest software and security patches

NOTABLE CUSTOMERS
- Ticketmaster
- Concur
- Ebay
- Time Warner Cable
- Verizon

WEBSITE coreos.com | **TWITTER** @CoreOS | **BLOG** coreos.com/blog

Five Things We've Learned About Monitoring Containers and Their Orchestrators

QUICK VIEW

01	Containers are ephemeral and hard to instrument, and that changes the monitoring game.
02	Manual instrumentation isn't feasible in a dynamic environment.
03	Service-oriented data analysis is critical to monitor and manage the performance of apps in a dynamic environment.
04	Monitoring data will be exponentially larger in containerized environments and requires a scalable storage solution.

BY **APURVA B DAVE**

This article will cover how to build a scaled-out, highly reliable monitoring system that works across tens of thousands of containers. It's based on Sysdig's experience building its container monitoring software, but the same design decisions will impact you if you decide to build your own tooling in-house. I'll share a bit about what our infrastructure looks like, the design choices and tradeoffs we've made. The five areas I'll cover are:

- Instrumenting the system
- Mapping your data to your applications, hosts, and containers
- Leveraging orchestrators
- Deciding what data to store
- Enabling troubleshooting in containerized environments

Ok, let's get into the details, starting with the impact containers have had on monitoring systems.

WHY DO CONTAINERS CHANGE THE RULES OF THE MONITORING GAME?

Containers are pretty powerful. They are:

- Simple: mostly individual process
- Small: 1/10th of a VM
- Isolated: fewer dependencies
- Dynamic: can be scaled, killed, and moved quickly.

Containers are simple and great building blocks for microservices, but their simplicity comes at a cost. The ephemeral nature of containers adds to their monitoring complexity. Just knowing that some containers exist is not enough: deep container visibility is critical for ops teams to monitor containers and troubleshoot issues. Let's start breaking down these monitoring challenges.

INSTRUMENTATION NEEDS TO BE TRANSPARENT

In static or virtual environments, an agent is usually run on a host and configured for specific applications. However, this approach doesn't work for containerized environments:

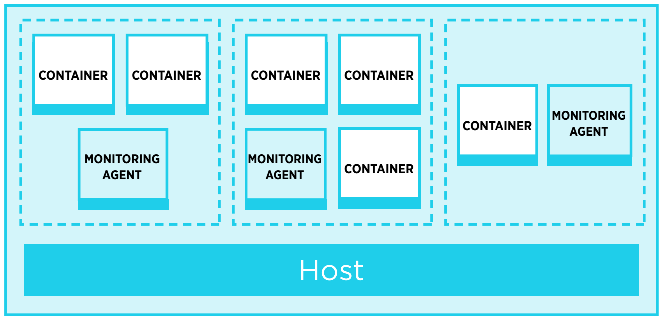
- You can't place an agent within each container.
- Dynamic applications make it challenging to manually configure agent plug-ins to collect metrics.

In containerized environments, you need to make instrumentation as transparent as possible with very limited human interaction. Infrastructure metrics, application metrics, service response times, custom metrics, and resource/network utilization data should be ingested without spinning up additional containers or making any effort from within the container.

There are two possible approaches: First are pods, a concept created by Kubernetes. Containers within each pod can see what other containers are doing. For monitoring agents, this is referred to as a "sidecar" container.

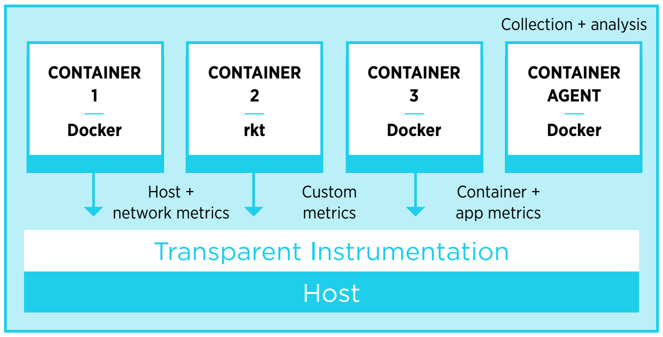
This is relatively easy to do in Kubernetes, but if you have many pods on a machine, this may result in heavy resource consumption and dependency. This can wreak havoc in your application if your monitoring sidecar has performance, stability, or security issues.

Pod Instrumentation



The second model is per-host, transparent instrumentation. This captures all application, container, statsd, and host metrics within a single instrumentation point and sends them to a container per host for processing and transfer. This eliminates the need to convert these metrics into statsd. Unlike sidecar models, per-host agents drastically reduce resource consumption of monitoring agents and require no application code modification. In Sysdig's case, we created a non-blocking kernel module to achieve this. That, however, required a privileged container.

Sysdig ContainerVision



By introducing "ContainerVision," Sysdig chose to do the latter, and herein lies the biggest tradeoff we had to make.

Although running the monitoring agent as a kernel module raises concerns and implementation complexities, this allows us to collect more data with lower overhead — even in high-density container environments — and reduces threats to the environment. Finally, to address these concerns as a third-party software provider, we open sourced our kernel module as part of the Sysdig Linux and container visibility command-line tool. This latter point isn't something you're likely to deal with if you're building your own internal tooling.

HOW TO MAP YOUR DATA TO YOUR APPLICATIONS, HOSTS, CONTAINERS, AND ORCHESTRATORS

As your environment increases in complexity, the ability to filter, segment, and group metrics based on metadata is essential. Tags allow you to represent the logical blueprint of your application architecture in addition to the physical reality of where containers are running.

There are two tagging metrics: explicit (attributes to store) vs. implicit (orchestrator) tags. Explicit tags can be added by your team based on best practices, but implicit tags should be captured by default. The latter is a key element for orchestrators. Each unique combination of tags is a separate metric that you need to store, process, and then recall on demand for your user. We'll discuss the major implications of this in the "Deciding what data to store" section below.

LEVERAGING ORCHESTRATORS

Orchestrators radically changed the scheduling management approach for containers and impacted users' monitoring strategy. Individual containers became less important, while the performance of a service became more important. A service is made up of several containers, and the orchestrator can move containers as needed to meet performance and health requirements. There are two implications for a monitoring system:

- Your monitoring system must implicitly tag all metrics according to the orchestration metadata. This applies to systems, containers, application components, and even custom metrics.

Your developers should output the custom metric, and the monitoring system should keep the state of each metric. You can read more about this topic here.

- Your monitoring agent should auto-discover any application and collect the relevant metrics. This may require you to update your monitoring system to provide these functionalities.

There are two methods to achieving this: depending on the events started by the orchestrator to flag containers, or determining applications based on the heuristics of a container. Sysdig chose the latter approach, as it requires more intelligence in your monitoring system, but produces more reliable results. You can read more on monitoring Kubernetes and orchestrators here.

DECIDING WHAT DATA TO STORE: "ALL THE DATA"

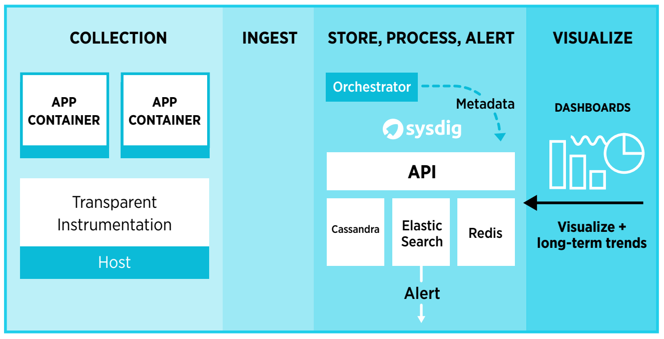
Distributed systems increase monitoring data and the resulting metrics. Although it is appealing to reduce your metric count for cost and simplicity, you'll find that the more complex your infrastructure becomes, the more important it is to have all the data for ad-hoc analysis and troubleshooting. For example, how would you identify an intermittent slow response time in a Node app with missing metrics data? How can you figure out if it's a systemic problem in the code, a container on the fritz, or an issue with AWS? Aggregating all that information via microservices will not give you enough visibility to solve the problem.

This means we will collect a lot of metrics and events data. In order to have this data persisted and accessible to our users, we decided to:

- Build a horizontally scalable backend with the ability for our application to isolate data, dashboards, alerts, etc. based on a user or service.
- Store full resolution data up to six hours and aggregate thereafter.

Our backend consists of horizontally scalable clusters of Cassandra (metrics), ElasticSearch (events), and Redis (intra-service brokering). This provides high reliability and scalability to store data for long-term trending and analysis. All the data is accessible by a REST API. You will likely end up building the same components if you create your own system.

Sysdig Architecture



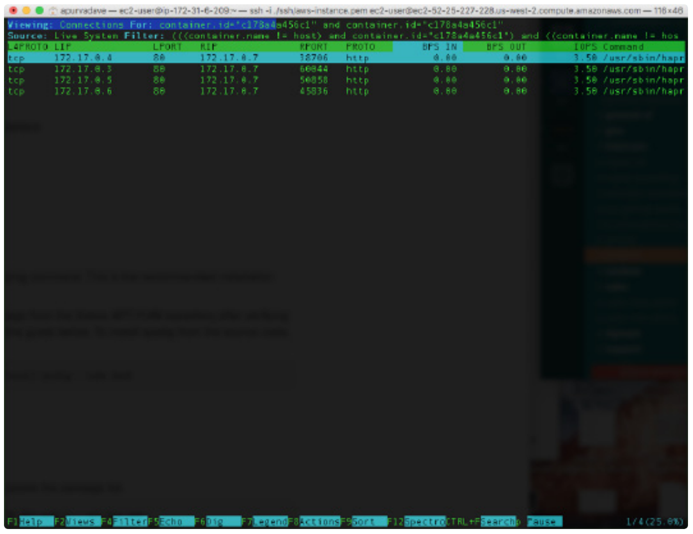
HOW TO ENABLE TROUBLESHOOTING IN CONTAINERIZED ENVIRONMENTS

Containers are ideal for deployment and repeatability, but troubleshooting them is challenging.

Troubleshooting tools — ssh, top, ps, and ifconfig — are neither accessible in PaaS-controlled environments nor available inside containers, and the ephemeral nature of containers adds to this complexity. This is where container troubleshooting tools come into play, with the ability to capture every single system call on a host giving deep visibility into how any application, container, host, or network performs.

Interfacing with the orchestration master provides relevant metadata that is not just limited to the state of a machine, but also provides the ability to capture the state and context of the distributed system. All this data is captured in a file, allowing you to troubleshoot production issues on your laptop and run post-mortem analyses at ease.

For example, when an alert is triggered by a spike in network connections on a particular container, all system calls on the host are recorded. Troubleshooting this alert via cSysdig provides all the relevant context and helps identify the root cause by drilling down to the network connections:



```
apurvadave — ec2-user@ip-172-31-6-209:~ — ssh -i ./ssh/aws-instance.pem ec2-user@ec2-52-25-227-228.us-west-2.compute.amazonaws.com — 116×48
Viewing: Connections For: container.id="c178a4a456c1" and container.id="c178a4a456c1"
Source: Live System Filter: (((container.name != host) and container.id="c178a4a456c1") and ((container.name != hos
L4PROTO LIP          LPORT  RIP           RPORT  PROTO      BPS IN     BPS OUT       IOPS Command
tcp     172.17.0.4   80     172.17.0.7    38706  http         0.00        0.00       3.50 /usr/sbin/hapr
tcp     172.17.0.3   80     172.17.0.7    60844  http         0.00        0.00       3.50 /usr/sbin/hapr
tcp     172.17.0.5   80     172.17.0.7    50858  http         0.00        0.00       3.50 /usr/sbin/hapr
tcp     172.17.0.6   80     172.17.0.7    45836  http         0.00        0.00       3.50 /usr/sbin/hapr

F1Help  F2Views F4FilterF5Echo  F6Dig   F7LegendF8ActionsF9Sort  F12SpectroCTRL+FSearch Pause        1/4(25.0%)
```

Check out Understanding how Kubernetes DNS Services work for more on open source Sysdig.

CONCLUSION

Building a highly scalable, distributed monitoring system is not an easy task. Whether you choose to do it yourself or leverage someone else's system, I believe you're going to have to make many of the same choices we made.

Apurva B Dave is the VP of marketing at Sysdig. He's in marketing and (gasp!) not afraid of a command line. He's been helping people analyze and accelerate infrastructure for the better part of two decades. He previously worked at Riverbed on both WAN acceleration and Network Analysis products, and at Inktomi on infrastructure products. He has a computer science degree from Brown University and an MBA from UC Berkley.

Comparing Container Image Registries

BY RITESH PATEL

CO-FOUNDER AND VP OF PRODUCTS AT NIRMATA

DOCKERHUB

If you have used Docker containers, then you probably know about DockerHub. It is one of the most widely used tools, since it is the default registry service for the Docker engine. DockerHub's collaboration model is similar to that of GitHub: you can create organizations and add add individual collaborators to each repository. Docker Hub provides major features like image repository management, webhooks, organizations, GitHub and BitBucket integration with automated builds, etc.

PROS:

- Uses a very familiar collaboration model, and is therefore very easy to use, especially for GitHub users
- Provides public and private repositories
- Quickly create organizations and add users or create groups of users to collaborate with your repositories.
- Allows users to set permissions to restrict access or set different permission levels like read, ride, and admin to different users.
- Fairly inexpensive with usage-based pricing.
- Security scanning available at additional cost.

CONS:

- Lacks fine-grained access control.
- Does not provide any insight into registry usage.
- Lacks LDAP, SAML, and OAuth support.
- Registry performance can be inconsistent.

AMAZON EC2 REGISTRY

No matter which architecture you choose, you must think about mediation and orchestration layers. Here's a list of the most popular integration frameworks implementing EIP patterns:

PROS:

- Familiar to AWS users and easy to use.
- Highly secure, as policies can be configured to manage permissions and control access to a user's images using AWS IAM without having to manage credentials directly on EC2 instances.
- No upfront fees or commitments. You pay only for the amount of data you store in your repositories and data transferred to the Internet.
- Tight integration with Amazon ECS and the Docker CLI, allowing you to simplify your development and production workflows.

CONS:

- Lack of insight into registry usage.
- Difficult to use with the Docker client, as it requires the creation of a temporary token.
- Potentially expensive if the deployed containers are not on AWS.

JFROG ARTIFACTORY

JFrog Artifactory is an enterprise-ready Universal Artifact Repository Manager supporting secure, clustered, highly available Docker registries. It integrates with all major CI/CD and DevOps tools, and provides an end-to-end, automated, and bullet-proof solution for tracking artifacts from development to production.

PROS

- Supports different artifacts created by any language or tools.
- Fairly easy to use.
- Clustering and High Availability are supported, which means that replication to another instance of Artifactory (multi-site) is easily possible.
- Flexible deployment options such as SaaS and on-premise.
- Out-of-the-box integrations with most CI/CD and DevOps tools.
- Security scanning is also available at an additional cost.

CONS

- The on-premise version needs to be managed and upgraded by the end user.
- Could be more expensive compared to hosted options.

Modern applications require a new approach to connectivity.
Tigera delivers it.
TIGERA
Learn more at www.tigera.io/dzone

Cloud Native: Preparing for Security & Scale

Imagine letting your development teams work in their preferred programming language, or not having to worry about having the right environment setup, or being able to scale at a moment's notice. Imagine being able to do all this, and much more - faster.

Of course, you've probably already imagined this - and know that many companies are working to make it a reality. After all, compared to traditional monolithic systems - the benefits of microservices are incredible!

However, like any new technology, microservice architectures bring with them their own set of challenges. Moving away from monoliths to containers means shifting focus from maintaining your AS400 to deploying and orchestrating hundreds to billions of constantly moving parts, while managing your network and focusing on security.

These challenges are complicated by the speed containers are being created at — 900x the speed of virtual machines — with a 25x faster churn. Not to mention the significant increase in attack surface and heavy reliance on web services such as REST APIs.

60% of web services contain high risk vulnerabilities.

This means you need to be well prepared, and understand that while microservices bring real, tangible benefits - they also bring inherent risks. Thankfully companies like Google, IBM, and Tigera are working to mitigate them with solutions such as Kubernetes (the most popular orchestrator), Istio, and Project Calico. Along with these open source tools — and Tigera's expertise — you can ensure a successful move to microservices in production.

To learn more about application connectivity, check out the Secure Networking for Kubernetes checklist, or visit tigera.io.

WRITTEN BY MIKE STOWE
DIRECTOR OF COMMUNITY, **TIGERA**

PARTNER SPOTLIGHT

Tigera Essentials for Kubernetes

TIGERA

Production-ready, cloud native application connectivity

CATEGORY
Application Connectivity

NEW RELEASES
Every two months

OPEN SOURCE
Yes

CASE STUDY
As a large multibillion dollar SaaS Provider moved from their monolithic infrastructure to microservices, they found that their existing network firewall was unable to scale and limited growth - they also needed a dynamic security solution that would enforce policy in both their Kubernetes environment as well as on their existing bare metal servers. They decided to choose Tigera Essentials for Kubernetes. With Tigera's expertise and professional grade SLA, they were able to do both confidently - increasing not only their ability to scale, but maintain security within their environment with greater insight and transparency.

STRENGTHS
- Builds on popular and proven open source technologies
- Policy query utility, auditing, and violation alerting
- Expert advice and implementation support
- Production Support 24x7 SLA with 30 Minute Response Time

DID YOU KNOW?
6 of the Forbes top 10 cloud operators are using Tigera's technologies.

WEBSITE tigera.io | **TWITTER** @tigeraio | **BLOG** blog.tigera.io

Executive Insights on Orchestrating and Deploying Containers

BY **TOM SMITH**
RESEARCH ANALYST, **DZONE**

QUICK VIEW

01 Containers improve speed to market with automated, secure development and deployment. However, initial set-up can be complicated.

02 Containers can be more secure than traditional virtual machines, with less developer input, but they cannot be forgotten or ignored since you're exposing a lot more data.

03 There's a lot of hype around containers, but the reality of what's available and actually being done is lagging behind the hype by a couple of years.

To gather insights on how companies are currently orchestrating and deploying containers, we spoke with 15 executives who are familiar with container deployment today. Here's who we talked to:

MARK THIELE CHIEF STRATEGY OFFICER, APCERA

RANGA RAJAGOPALAN CTO AND CO-FOUNDER, AVI NETWORKS

CHANDRA SEKAR V.P. MARKETING, AVI NETWORKS

DUSTIN KIRKLAND UBUNTU PRODUCT AND STRATEGY, CANONICAL

PHIL DOUGHERTY CEO, CONTAINERSHIP

ANDERS WALLGREN CTO, ELECTRICCLOUD

LUCA RAVAZZOLO PRODUCT MANAGER, INTERSYSTEMS

RAJESH GANESAN DIR. OF PRODUCT MANAGEMENT, MANAGEENGINE

JIM SCOTT DIR. OF ENTERPRISE STRATEGY & ARCHITECTURE, MAPR

BROOKS CRICHLOW V.P. PRODUCT MARKETING, MONGODB

DEREK SMITH CEO, NAVEEGO

FEI HUANG CEO, NEUVECTOR

CHRIS BRANDON CEO, STORAGEOS

WEI LIEN DANG V.P. OF PRODUCTS, STACKROX

JOHN MORELLO CTO, TWISTLOCK

KEY FINDINGS

01 The most important elements of orchestrating and deploying containers are **wide-ranging and require a platform, support framework, infrastructure, or ecosystem.** The elements of this ecosystem should include: storage, network, security, management, load balancing, provisioning, orchestration/scheduling, lifecycle management, patch management, deployment, automation, bursting, monitoring, log aggregation, and routing. It's important to understand these services are an integrated ecosystem that are automatically managed with predetermined policies aligned with the problems that need to be solved.

02 **Kubernetes** was the most frequently mentioned tool that respondents used to orchestrate and deploy containers. **Go and Java** were the most popular programming languages for applications built with containers. However, those were just three of 26 solutions mentioned.

03 **The automation of application deployment and developing has accelerated the speed of the orchestration and deployment of containers.** This results in high-quality and secure applications being deployed more quickly. Containers provide a standard unit of packaging for applications, so developers can focus on individual components and collaborate when building out the application. Developers can move these standard units across a variety of environments. Package lifecycles can be easily automated from development to deployment to runtime to teardown. This results in greater reliability and stability, as well as the ability to push new apps quickly.

04 **Security of containers relies on best practices and scans,** as well as companies' own solutions. Best practices include: a secure operating system beneath the container, image scanning, host security, trusted registries, access controls, and integration with run-time security. Companies should only deploy what has passed predetermined security protocols. Some companies

have built their own platforms to secure containers, while others are using Twistlock, Hashicorp Vault, 256bit AES encryption or TLS for encryption of traffic between nodes, and SSL for frontend web security.

05 Containers are helping companies across many industries accelerate software development and deployment at scale, while reducing costs and saving IT departments time. Customers can instantly launch, migrate, and scale applications based on granular cost-benefit analysis. Solutions providers are reducing the time and effort by reducing the IT workload environment by 85%, building an infrastructure for log management, and reducing risk through automation, which provides healthcare, financial services, telecom, and entertainment companies with greater flexibility to deliver against their business objectives.

06 The complexity and speed of change around developers' tools and security, are the most common issues affecting the orchestration and deployment of containers. The rapid evolution of container platform components such as orchestration, storage, networking, and systems services like load balancing are making the entire stack a moving target. This makes it difficult to have a stable application or service on top of them. The industry will need to standardize, consolidate, and simplify containers for mass adoption.

Security and visibility are concerns as well. Containers are great in a test environment, but they can be tougher to roll out into production and scale for enterprise-grade services. The orchestration platform is a substantial attack surface.

07 The greatest concerns regarding the current state of containers are complexity, security, and hype. Once it's in place, a container-based CI/CD pipeline is incredibly powerful; however, getting there is not easy. Orchestration systems are not easy to set up and automate. We need containers talking to each other via registries and services in a standardized way.

Security is an unknown frontier. We need an ecosystem of container technology companies to work together to make it easier for the average enterprise to adopt containers securely. The security exposure is at the application layer, which is frequently changing and scaling. We need to think about how security is architected and implemented from the beginning. We are very concerned about security, high availability, disaster recovery, and policy and roles management.

The actual capabilities lag behind the the hype by a couple of years. This noise and misinformation makes it that much more complicated for companies to pursue and implement a containers-based infrastructure. An elementary technology has gotten overcomplicated with too many vendors blowing smoke.

08 We're at an inflection point similar to where virtual machines were 10 years ago. We have convenient development and packaging tools for developers that are spreading into IT and operations. Ultimately this will lead to a better user experience (UX). We are moving toward the immutable image of deployment where we have reliability and consistency. We're making headway on the APIs and the standards around them. Docker made containers so popular but the addition of too many things resulted in stability problems and tools like Rocket evolving from Docker. What happens to Docker customers as they scale?

As everything scales, there is greater opportunity for hackers and bots. Keys and security must be maintained, as well as access policies. The declarative nature of containers will make securely spinning up containers as easy as spinning up a cloud server instance today. There's still speed and efficiency to be rung out of the application.

What's next? Serverless with the lambda functionality offered by AWS and Azure along with remotely scheduled processes. Hosting constraints are bringing us very close to a serverless environment. This will become a special discipline within containers.

09 Developers need to know a lot when working on orchestrating and deploying containers. **Start by working with the architect or operations to map how the application will work from end-to-end.** Know the workflow of development, orchestration, and deployment. Be aware of how the application will handle run-time situations in a microservices architecture. Learn how to properly build a clean Docker container without unnecessary components. Remember to protect the data of your application – storage and data security must be included in the initial design. Container systems do not have native persistent storage. Containers can be more secure than traditional virtual machines with less developer input; however, security cannot be forgotten or ignored.

10 Additional considerations involve security, data, and automation:

- There is no one solution to container security. The complexity of virtualized environments requires multiple levels of security to be in place. There are many best practices for preparing a secure environment for containers, but the real challenge is getting the security and visibility needed when containers are actively running in production and there are suspicious activities happening in real time. Ultimately, we'll have active penetration testing of containers in real time.

- Data fabric and centralized data storage is overlooked and developers are missing the benefits. How are people managing their data when it's in a container? How do containers talk to each other? Data is the most critical piece of the scaling platform you are building out. How are you managing the central repository in the data center? When there are multiple data centers, we mirror volumes of data with data repositories; however, a lot of people overlook the management of these repositories.

- I'm not sure organizations realize the importance of organizational change to get value from a cloud-native approach. CI/CD is less about the build and orchestration software you're using and more about the mindset shift of automating everything and tearing down the traditional friction points in deployment.

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.

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	Demo available by request	anchore.com
Anchore	Anchore Navigator	Container monitoring	Free solution	anchore.io
Apache Software Foundation	Mesos	Cluster management software	Open source	mesos.apache.org
Apcera	Apcera	PaaS, container platform	Free tier available	apcera.com/platform
Apprenda	Apprenda Platform	PaaS, container platform, Kubernetes-as-a-Service	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	Available by request	stacksmith.bitnami.com
CA Technologies	Automic Continuous Service	Service orchestration	Available by request	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
ClusterHQ	FlockerHub	Docker Volume Repository	In open beta	clusterhq.com/flockerhub/introduction

COMPANY	PRODUCT	PRODUCT TYPE	FREE TRIAL	WEBSITE
Codefresh	Codefresh	CI/CD platform for containers	Free tier available	codefresh.io
CoreOS	Quay Enterprise	Private container image registry	30 days	coreos.com/quay-enterprise
CoreOS	Clair	Container image security	Open source	coreos.com/clair/docs/latest
CoreOS	Tectonic	Containers-as-a-Service	Free up to ten nodes	coreos.com/tectonic
CoreOS	Flannel	Container-defined networking	Open source	coreos.com/flannel/docs/latest
Datadog	Datadog	Server monitoring, container monitoring	14 days	datadoghq.com
Diamanti	Diamanti	Container-defined storage hardware	N/A	diamanti.com/products
Docker	Docker	Container platform	Free tier available	docker.com/get-docker
Docker	Docker Swarm	Container orchestration and clustering	Open source	github.com/docker/swarm
Docker	Docker Compose	Multi-container application tool	Free solution	docs.docker.com/compose/install
Docker	Docker Hub	Container image registry	Free for public repos	hub.docker.com
Docker	Docker Trusted Registry	Private container image registry	30 days	docker.com/enterprise-edition#/container_management
Docker	Docker Cloud	Containers-as-a-Service	Available by request	cloud.docker.com
Docker	Docker EE	Containers-as-a-Service, container security	Free tier available	docker.com/enterprise-edition
Google	Kubernetes	Container orchestration	Open source	kubernetes.io
Google	Google Cloud Container Registry	Container image registry	Free based on use	cloud.google.com/container-registry
Google	Google Container Engine	Containers-as-a-Service, container orchestration	$300 free credit	cloud.google.com/container-engine
Hedvig	Hedvig Distributed Storage Platform	Container-defined storage	Demo available by request	hedviginc.com/product#hedvig-distributed
IBM	Bluemix Containers	Containers-as-a-Service, Kubernetes-as-a-Service	Free tier available	ibm.com/cloud-computing/bluemix/containers

COMPANY	PRODUCT	PRODUCT TYPE	FREE TRIAL	WEBSITE
Instana	Instana Infrastructure Quality Management	Infrastructure monitoring	14 days	instana.com/infrastructure-management
JFrog	Artifactory	Artifact Repository Manager	30 days	jfrog.com/artifactory
Joyent	Triton	Container networking and management platform	$250 credit	joyent.com/triton/compute
Kontena	Kontena Platform	Container platform	Open source	kontena.io/platform
Loggly	Loggly	Log management and analytics	Available by request	loggly.com
Mesosphere	DC/OS	Container orchestration	Open source	dcos.io
Mesosphere	Enterprise DC/OS	Container orchestration and 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	Free tier available	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	Free tier available	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
Portworx	Portworx	Container data services and networking	Demo available by request	portworx.com
Prometheus	Prometheus	Container monitoring	Open source	prometheus.io

COMPANY	PRODUCT	PRODUCT TYPE	FREE TRIAL	WEBSITE
Rancher Labs	Rancher	Container management platform	Open source	rancher.com/rancher
Rancher Labs	RancherOS	Container OS	Open source	rancher.com/rancher-os
Rapid7	Logentries	Log management and analytics	30 days	logentries.com
Red Hat	Atomic Host	Container OS	Open source	projectatomic.io
Red Hat	Atomic Registry	Private container image registry	Open source	projectatomic.io/registry
Red Hat	Atomic Scan	Container image security	Open source	github.com/projectatomic/atomic
Red Hat	OpenShift Container Platform	PaaS, container platform	Free tier available	openshift.com/container-platform
Sematext	Docker Agent	Container monitoring and log management	30 days	sematext.com/docker
Sematext	Kubernetes Agent	Kubernetes monitoring and log management	30 days	sematext.com/kubernetes
Sensu	Sensu Core	Container monitoring	Open source	sensuapp.org
Splunk	Splunk Cloud	Log management and analytics	15 days	splunk.com/en_us/products/splunk-cloud.html
Sumo Logic	Sumo Logic	Log management and analytics	Free tier available	sumologic.com
Sysdig	Sysdig Falco	Behavioral activity monitor with container support	Open source	sysdig.org/falco
Sysdig	Sysdig Monitor	Container Monitoring and Troubleshooting	14 Days	sysdig.com
Sysdig	Sysdig Open Source	Linux system level visibility	Open source	sysdig.org
Tigera	Canal	Container-defined networking	Open source	github.com/projectcalico/canal
Twistlock	Twistlock Trust	Container image security	Free tier available	twistlock.com
VMware	Photon OS	Linux container host	Open source	vmware.github.io/photon/
Wavefront	Wavefront	Cloud monitoring and analytics	Available by request	wavefront.com/product
Weaveworks	Weave Net	Container-defined networking	Open source	weave.works/oss/net

Diving Deeper

INTO CONTAINERS

TOP #CONTAINERS TWITTER ACCOUNTS

@ericschabell

@rhatdan

@imesh

@csanchez

@JimBugwadia

@proudboffin

@launchany

@nathankpeck

@siruslan

@RealGeneKim

CONTAINER-RELATED 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.

Integration dzone.com/integration
The Integration Zone focuses on communication architectures, message brokers, enterprise applications, ESBs, integration protocols, web services, service-oriented architecture (SOA), message-oriented middleware (MOM), and API management.

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.

TOP CONTAINERS REFCARDZ

Getting Started With Docker
Teaches you typical Docker workflows, building images, creating Dockerfiles, and includes helpful commands to easily automate infrastructure and contain your distributed application.

Java Containerization
Java + Docker = separation of concerns the way it was meant to be. This Refcard includes suggested configurations and extensive code snippets to get your Java application up and running inside a Docker-deployed Linux container.

Getting Started With Kubernetes
Containers weighing you down? Kubernetes can scale them. In order to run and maintain successful containerized applications, organization is key. Kubernetes is a powerful system that provides a method for managing Docker and Rocket containers across multiple hosts. This Refcard includes all you need to know about Kubernetes including how to begin using it, how to successfully build your first pod and scale intelligently, and more.

TOP CONTAINERS VIDEOS

Introduction to Docker and Containers
youtube.com/watch?v=IEGPzmxyIpo

Cloud Computing Explained
youtube.com/watch?v=QJncFirhjPg

Demystifying Docker
youtube.com/watch?v=q1qEYM_SESI

TOP CONTAINERS RESOURCES

What Are Containers?
Running containers in the AWS Cloud allows you to build robust, scalable applications and services by leveraging the benefits of the AWS Cloud such as elasticity, availability, security, and economies of scale.

Containers vs. Virtual Machines
VMs and Containers differ on quite a few dimensions, but primarily because containers provide a way to virtualize an OS in order for multiple workloads to run on a single OS instance, whereas with VMs, the hardware is being virtualized to run multiple OS instances.

Indexed Containers
The search for an expressive calculus of datatypes in which canonical algorithms can be easily written and proven correct has proved to be an enduring challenge to the theoretical computer science community.

GLOSSARY

APPLICATION PERFORMANCE MONITORING (APM)
Combines metrics on all factors that might affect application performance (within an application and/or web server, between database and application server, on a single machine, between client and server, etc.); usually (but not always) higher-level.

APPLICATION RELEASE AUTOMATION (ARA)
The process of packaging and deploying software from development to production, where software is moved through different environments and releases are coordinated automatically.

BUILD ARTIFACT
The resulting application or object created by a build process. Typically this 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.

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
A container image is a essentially a snapshot of a container. They are created with a build command and produce a container that you can later run.

CONTINUOUS DELIVERY
A software engineering approach in which continuous integration, automated testing, and automated deployment capabilities allow software to be developed and deployed rapidly, reliably, and repeatedly with minimal human intervention.

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

DEVOPS
An IT organizational methodology where all teams in the 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.

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

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

METADATA
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.

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.

SERVICE MESH
A set of proxies providing resilience and other network capabilities to applications without application interference.

SIDECAR
An application or proxy co-deployed with the main application or service to provide enhanced functionality to the main application.

INTRODUCING THE

AI Zone

What's new with Predictive Analytics, Natural Language Processing, and Neural Nets? Check out DZone's new AI Zone – a central place for AI resources and tutorials.

Keep a pulse on the industry and go beyond the hype. This Zone gives you access to practical applications and use cases for AI technologies like: Machine Learning, Cognitive Computing, and Chatbots.

Visit the Zone

MACHINE LEARNING

COGNITIVE COMPUTING

CHATBOTS

DEEP LEARNING