THENEWSTACK

vol. 3

AUTOMATION & ORCHESTRATION WITH DOCKER & CONTAINERS

EDITED & CURATED BY ALEX WILLIAMS

The New Stack:
The Docker and Container Ecosystem eBook Series

Alex Williams, Founder & Editor-in-Chief
Benjamin Ball, Technical Editor & Producer
Hoang Dinh, Creative Director
Sam Charrington, Editor, Founder and Principal Analyst of CloudPulse Strategies

Contributors:
Joab Jackson, Managing Editor
Judy Williams, Copy Editor
Lawrence Hecht, Data Research Director
Luke Lefler, Audio Engineer
Michelle Maher, Copy Editor

TABLE OF CONTENTS

AUTOMATION & ORCHESTRATION WITH DOCKER & CONTAINERS

AUTOMATION & ORCHESTRATION DIRECTORY

INTRODUCTION

The container ecosystem has become a very crowded space. Increasing rates of container adoption and usage in production means that the management and orchestration space is just that much more important.

In this ebook, The New Stack explores the ways that vendors and practitioners are automating, orchestrating, and managing their container technologies in multiple environments. We take a closer look at comparisons of major orchestration tools, including orchestration platforms and containers as a service, and how orchestrating containers is a critical component for advancing a wider usage pattern for containers.

We cover the basic components that go into orchestrating containers, and how orchestration involves an array of different functionalities including cluster management, scheduling, service discovery, configuration management, and even beyond into the role of infrastructure and platforms. We discuss how this space compares to virtual machine management, and how many of the same concepts apply within the realms of software delivery, networking, storage and management.

Another major focus of the ebook is on original research surrounding the perceptions of end users and vendors. We created an orchestration survey that looked primarily to measure the expectations of our audience about the functionality of orchestration tools; we also asked respondents to report their current and projected usage of tools, level of container adoption, usage of container technologies in production environments, and more. The resulting data paints a very interesting picture of the current market for orchestration tools and services, and we look to build upon and analyze this data even more in the coming months.

This is our third ebook in the Docker and Container Ecosystem series, and the middle of the pack in our total of five ebooks. Since the next ebook is going to address more around networking, storage and security, it was important to establish the goals of orchestration and management in this book. The primary concepts around container management are prevalent throughout the rest of the book series, and this ebook provides a necessary foundation for understanding the choices users will need to make about their technology and practices.

This book represented an enormous challenge for us in some ways, as the orchestration space is perhaps the most contested in the container ecosystem. It intersects many different tool and service categories, and represents a bountiful area for open source software. Orchestration itself involves so many types of behaviors and practices, that it's no wonder vendors are so eager to capture the attention of Dev and Ops alike.

As I've said before, we're constantly looking for new topics in need of greater focus and education, and we welcome any feedback on what areas we should tackle next, even beyond the container ecosystem. Thanks so much for your interest in our ebook series. Please reach out to our team any time with feedback, thoughts, and ideas for the future.

Thanks,
Ben

Benjamin Ball
Technical Editor and Producer

SPONSORS

We are grateful for the support of the following ebook series sponsors:

CISCO

docker

IBM

And the following sponsors for this ebook:

APCERA

cloudsoft

MESOSPHERE

KEY CONCEPTS IN ORCHESTRATION

by **JANAKIRAM MSV**

The Docker platform and surrounding ecosystem contain many tools to manage the lifecycle of a container. Just one example, Docker Command Line Interface (CLI) supports the following container activities:

- Pulling a repository from the registry.
- Running the container and optionally attaching a terminal to it.
- Committing the container to a new image.
- Uploading the image to the registry.
- Terminating a running container.

While the CLI meets the needs of managing one container on one host, it falls short when it comes to managing multiple containers deployed on multiple hosts. To go beyond the management of individual containers, we must turn to orchestration tools.

> “Orchestration tools extend lifecycle management capabilities to complex, multi-container workloads deployed on a cluster of machines.

By abstracting the host infrastructure, orchestration tools allow users to treat the entire cluster as a single deployment target.

Baseline Features

The process of orchestration typically involves tooling that can automate all aspects of application management from initial placement, scheduling and deployment to steady-state activities such as update, deployment, update and and health monitoring functions that support scaling and failover. These capabilities have come to characterize some of the core features users expectations offor modern container orchestration tools.

Declarative Configuration

Orchestration tools provide an option for DevOps teams to declare the blueprint for an application workload and its configuration in a standard schema, using languages such as YAML or JSON. These definitions also carry crucial information about the repositories, networking (ports), storage (volumes) and logs that support the workload. This approach allows orchestration tools to apply the same configuration multiple times and always yield the same result on the target system. It also allows the tools to accept different configurations for the same application during the various stages of development, testing and production for different target environments.

Rules and Constraints

Workloads often have special policies or requirements for host placement, performance and high availability. For example, it's pointless to provision the master and slave database container on the same host; it defeats the purpose. Similarly, it may be a good idea to place in-memory cache on the same host as the web server. Orchestration tools support mechanisms for defining the affinity and constraints of container placement.

Provisioning

Provisioning, or scheduling, deals with negotiating the placement of containers within the cluster and launching them. This process involves selecting an appropriate host based on the configuration. Apart from a container-provisioning API, orchestration tools will invoke the infrastructure APIs specific to the host environment.

Discovery

In a distributed deployment consisting of containers running on multiple hosts, container discovery becomes critical. Web servers need to dynamically discover the database servers, and load balancers need to discover and register web servers. Orchestration tools provide, or expect, a distributed key-value store, a lightweight DNS or some other mechanism to enable the discovery of containers.

Health Monitoring

Since orchestration tools are aware of the desired configuration of the system, they are uniquely able to track and monitor the health of the system's containers and hosts. In the event of host failure, the tools can relocate the container. Similarly, when a container crashes, orchestration tools can launch a replacement. Orchestration tools ensure that the deployment always matches the desired state declared by the developer or operator.

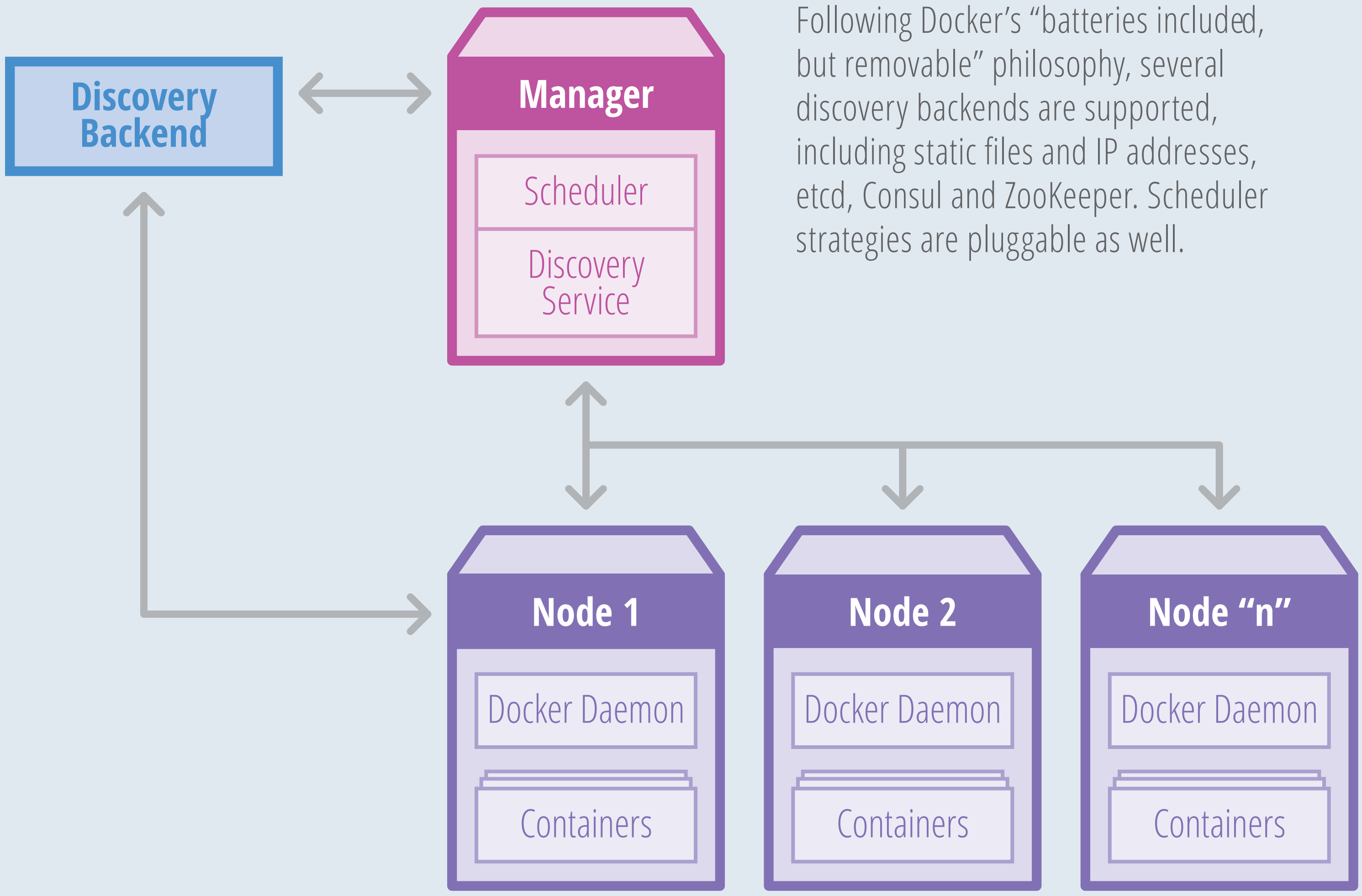
A Closer Look at Three Popular Orchestration Platforms

Docker Swarm

The objective of Docker Swarm is to use the same Docker API that works with the core Docker Engine. Instead of targeting an API endpoint

FIG 1: *Docker Swarm features pluggable scheduling algorithms and broad registry support. Communication between manager and nodes uses standard Docker APIs.*

Docker Swarm: Swap, Plug, and Play



Following Docker's "batteries included, but removable" philosophy, several discovery backends are supported, including static files and IP addresses, etcd, Consul and ZooKeeper. Scheduler strategies are pluggable as well.

Source: The New Stack.

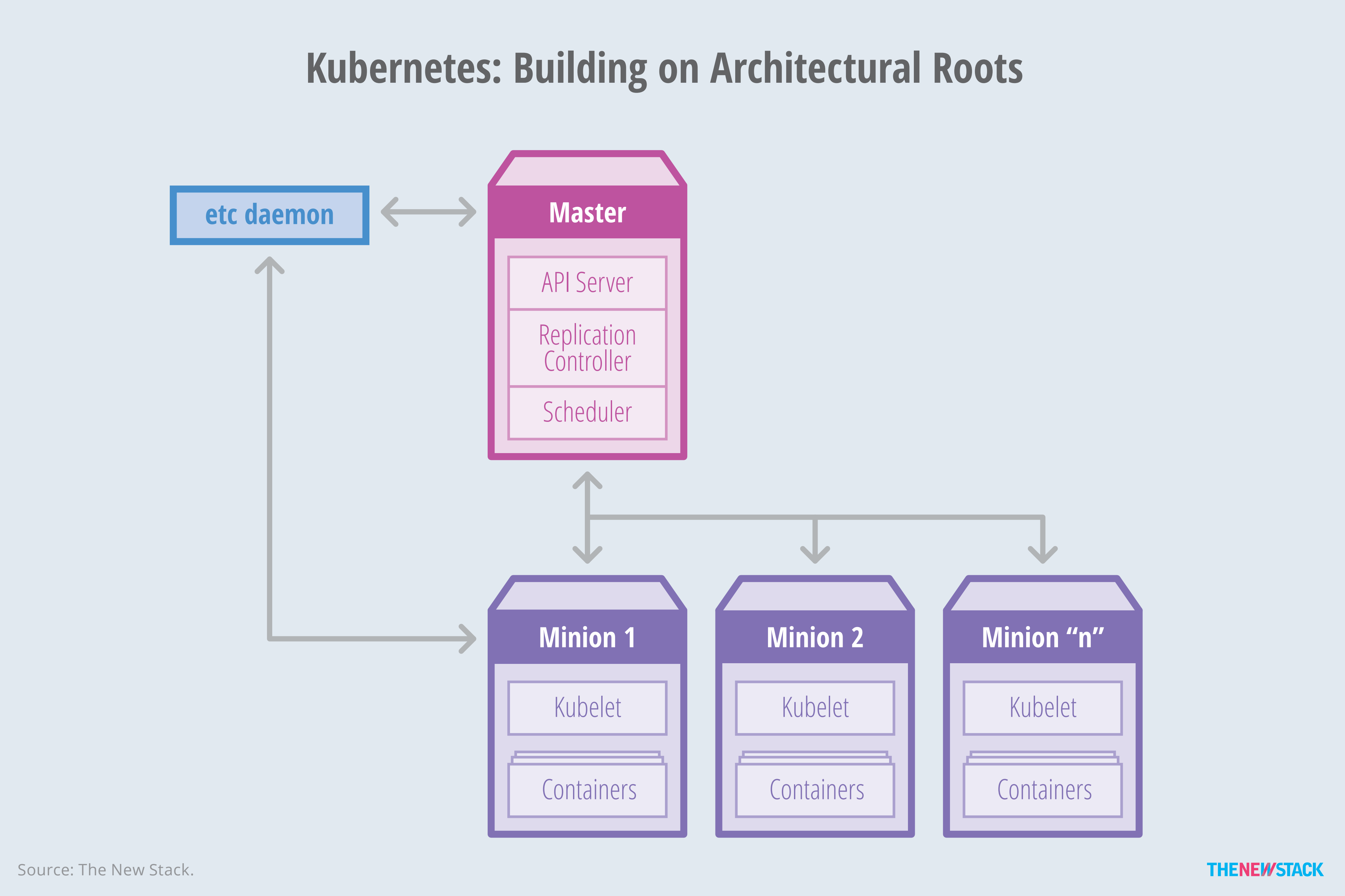
representing one Docker Engine, Swarm transparently deals with an endpoint associated with a pool of Docker Engines. The key advantage to this approach is that the existing tools and APIs will continue to work with a cluster in the same way they work with a single instance. Docker's tooling/CLI and Compose are how developers create their applications and therefore they don't have to be recoded to accommodate an orchestrator.

Docker Swarm comes with several built-in scheduling strategies, giving users the ability to guide container placement so as to maximize or minimize the spread of containers across the cluster. Random placement is supported as well.

Docker seeks to follow the principle of "batteries included but removable," meaning that while it currently ships with only a handful of simple scheduling backends, in the future it may support additional backends through a pluggable interface. Based on the scale and complexity of a given use case, developers or operations staff might choose to plug in an appropriate backend.

Docker Swarm supports constraints and affinities to determine the placement of containers on specific hosts. Constraints define requirements to select a subset of nodes that should be considered for scheduling. They can be based on attributes like storage type, geographic location, environment and kernel version. Affinity defines requirements to collocate containers on hosts.

For discovering containers on each host, Swarm uses a pluggable backend architecture that works with a simple hosted discovery service, static files, lists of IPs, etcd, Consul and ZooKeeper.



Source: The New Stack.

FIG 2: *Kubernetes traces its architectural lineage to Google Borg, an internal cluster management system responsible for launching over 2 billion containers per day.*

Swarm supports basic health monitoring, which prevents provisioning containers on faulty hosts.

Kubernetes

Coming from Google — a company that claims to deal with two billion containers every day — Kubernetes enjoys unique credibility.

Kubernetes' architecture is based on a master server with multiple minions. The command line tool, called kubecfg, connects to the API endpoint of the master to manage and orchestrate the minions. Below is the definition of each component that runs within the Kubernetes environment:

- **Master:** The server that runs the Kubernetes management processes, including the API service, replication controller and scheduler.
- **Minion:** The host that runs the kubelet service and the Docker Engine. Minions receive commands from the master.
- **Kubelet:** The node-level manager in Kubernetes; it runs on a minion.
- **Pod:** The collection of containers deployed on the same minion.
- **Replication controller:** Defines the number of pods or containers that need to be running.
- **Service:** A definition that allows the discovery of services/ports published by each container, along with the external proxy used for communications.
- **Kubecfg:** The command line interface that talks to the master to manage a Kubernetes deployment.

The service definition, along with the rules and constraints, is described in a JSON file. For service discovery, Kubernetes provides a stable IP address and DNS name that corresponds to a dynamic set of pods. When a container running in a Kubernetes pod connects to this address, the connection is forwarded by a local agent (called the kube-proxy) running on the source machine to one of the corresponding backend containers.

Kubernetes supports user-implemented application health checks. These checks are performed by the kubelet running on each minion to ensure that the application is operating correctly. Currently, Kubernetes supports three types of health checks:

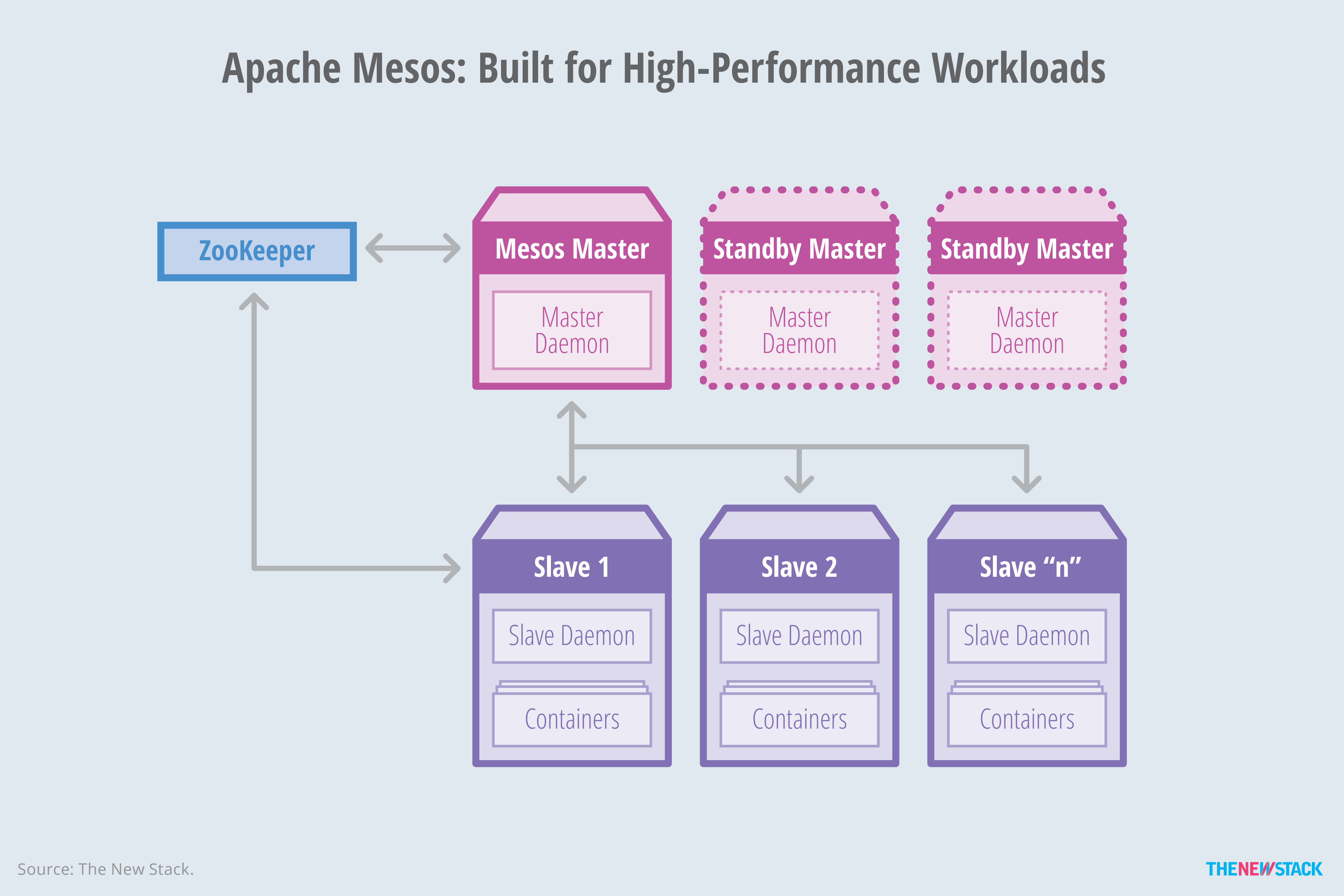
- **HTTP health check:** The kubelet will call a web endpoint. If the response code is between 200 and 399, it is considered a success.

- **Container exec:** The kubelet will execute a command within the container. If it returns “OK,” it is considered a success.
- **TCP socket:** The kubelet will attempt to open a socket to the container and establish a connection. If the connection is made, it is considered healthy.

Apache Mesos

Apache Mesos is an open source cluster manager that simplifies the complexity of running tasks on a shared pool of servers. Originally

FIG 3: *With roots in the high performance computing world, Mesos supports Hadoop, Spark and more in addition to Docker and containers.*



Source: The New Stack.

designed to support high-performance computing workloads, Mesos added support for Docker in the 0.20.0 release.

A typical Mesos cluster consists of one or more servers running the mesos-master and a cluster of servers running the mesos-slave component. Each slave is registered with the master to offer resources. The master interacts with deployed frameworks to delegate tasks to slaves. Below is an overview of Mesos' architecture:

- **Master daemon:** The mesos-master service runs on a master node and manages slave daemons.
- **Slave daemon:** The mesos-slave service runs on each slave node to run tasks that belong to a framework.
- **Framework:** An application definition consisting of a scheduler that registers with the master to receive resource offers, along with one or more executors to launch tasks on the slaves.
- **Offer:** The list of a slave node's resources. Each slave node sends offers to the master, and the master provides offers to registered application frameworks.
- **Task:** The unit of work scheduled by a framework to be executed on a slave node.
- **Apache ZooKeeper:** The software used to coordinate the collection of master nodes.

Unlike other tools, Mesos ensures high availability of the master nodes using Apache ZooKeeper, which replicates the masters to form a quorum. A high availability deployment requires at least three master nodes. All nodes in the system, including masters and slaves, communicate with ZooKeeper to determine which master is the current leading master. The leader performs health checks on all the slaves and proactively deactivates any that fail.

When Mesos is used in conjunction with Marathon, service discovery can be enabled based on the HAProxy TCP/HTTP load balancer, along with an assistant script that uses Marathon's REST API to periodically regenerate a HAProxy configuration file. Alternatively, Mesos-DNS, a DNS-based service discovery mechanism, has recently been released in beta.

Summary

The container ecosystem is growing rapidly. From major infrastructure companies to PaaS vendors to early-stage startups and even in serverless computing, everyone is clamoring to stake out their place in the ecosystem. There are many contributors working on container orchestration tools, as these are essential for deploying real-world applications, thus driving the adoption of Docker and containers. We attempted to highlight some of the key contributors building orchestration tools, but there is more to it than just explicit orchestration tools — it's also important to look at the build, deployment, CI/CD, PaaS, and other tools that orchestrators interact with, which we cover at great length in the Automation and Orchestration Directory.

BRIDGING REALITIES: ORCHESTRATION AND PROGRAMMABLE INFRASTRUCTURE

by **ALEX WILLIAMS** *and* **BENJAMIN BALL**

The continuous shifting of the container ecosystem means there are contrary views surrounding VMs, containers and the roles they play in automated, scaled-out environments. There are also many unknowns, as most technical professionals have little experience developing on distributed platforms. Most people have developed on single hosts or managed systems as administrators, managing machines and their virtualized environments.

Running clusters and orchestrating thousands of containers is an entirely new game. It puts software at the center stage, with a greater need to monitor components than the application itself. It's a world that requires an advanced appreciation of automation via APIs, and orchestration that takes into account the need for scheduling, cluster management and a host of other matters, such as securing nodes, health checks and prioritization.

> “Running clusters and orchestrating thousands of containers is an entirely new game.

Today we see how orchestration for containers changes the lens of how we view applications. Microservices orient developers to focus on services and how those services differentiate applications. Containers are catalysts for making these applications behave according to the components on distributed clusters. Apache Mesos, Docker Swarm, HashiCorp Nomad and Kubernetes are good examples of this. They help systems operators model their clusters and make services accessible.

Many software companies have built platforms that are almost entirely ephemeral. For companies adopting elastic, scaled-out platforms, the concept is often about how to eliminate the VM entirely, build on bare metal and create container-based clusters.

This all adds up to a change in how we think about automation and orchestration. But one principle remains: there’s a push to implement programmable infrastructure, applying methods and tooling established in software development onto the management of IT infrastructure, across thousands of clouds. It’s a movement that will lead to any number of new practices and discoveries.

Facing the Current, Complex Reality

The software industry at large has failed to truly implement the ideas behind programmable infrastructure, said Solomon Hykes, Docker founder and chief technical officer. It’s difficult to become an effective developer without ten years or more of programming experience. Scale needs to be at the scale of the Internet, Hykes said.

“So clustering, orchestration, networking between containers, storage across more than one container, that shared storage, security, things like that, verification of contents so that you can track the authenticity and provenance of the containers you’re about to deploy,” Hykes said. “All of these are part of the problem of building and deploying distributed applications. So it’s a big list of problems and you need a combination of tools to address these problems. And you need those tools to integrate. They need to work together to form a platform, otherwise you’ve got a giant puzzle and the pieces will never fit together. You’ll never complete the puzzle.”

Hykes will tell you that Docker takes an incremental approach with tool development and does not try to “boil the ocean” by developing one platform to do it all. It’s an approach that applies to the overall Docker and container ecosystem. That is, in part, due to the complexity that scale brings and the very nature of container technology itself.

Docker and containers are processes. They make delivering components easier. A container does not carry an operating system with it. That makes the container lighter and easier to manage. It does not require configuration to the machine itself. And a container is disposable — it embodies the concept of immutable infrastructure; live instances are never directly changed, but rather replaced as their configuration changes. Due to their light weight and easy reproducibility, containers, in their current packaged form, introduce new complexities previously not understood.

> “A container is disposable — it embodies the concept of immutable infrastructure.

With containers comes a change in the role that data centers will play. The research and development that has existed internally will move outwards into open source communities. This kind of change could have happened with VMs, but due to the great tooling and speed at which containers are easily managed and ported, it makes it more natural than with VMs.

The software will, in turn, continue to get developed at an even more rapid pace, built as integrated technologies for the developer and the operations person, who can no longer separate their duties. There is no wall anymore between Dev and Ops, just a pipeline of continuous delivery.

Removing that wall between Dev and Ops is the most important shift of all that will come as container adoption becomes more widespread, accelerated by open source development. The software to manage clusters will be the orchestration platforms, working on data planes that make container-based clusters fully mobile. The infrastructure itself becomes centered on the application.

Cloud-Native and Programmable Infrastructure

The concept of cloud native has set the stage for how organizations develop a programmable infrastructure, and the complexity is astounding.

There are a number of overlapping domains, especially when we think about container management, said Chris Ferris, distinguished engineer and chief technical officer of open technology at IBM Cloud, in an interview with The New Stack. There is Kubernetes, Swarm, Mesos, initiatives around OpenStack Magnum, etc., all to orchestrate, manage and schedule containers. Then you've got the likes of Cloud Foundry, which is also doing container scheduling and orchestration, but it's a little bit more hidden.

> *"All of these technologies are independently developed in independent groups, in independent communities, ...ultimately these things have to start coalescing, coming together, or at least providing the ability that we can integrate between OpenStack and Kubernetes. For instance, if I'm running containers in a Kubernetes pod and I want to integrate those capabilities with something I've got running in a VM and OpenStack, how do I do that from a networking and storage perspective, how do I share the networking storage across those platforms?"*

There are arguments about what goes underneath, in the middle or on top, Ferris said. OpenStack Magnum is getting built to provision the likes of Kubernetes and Mesos.

This may lead to different integrations that allow customers to build from components that suit their needs, and potentially an architecture that everybody can agree upon. What is needed is fault tolerance, self healing, easy roll-outs, versioning, and the ability to easily scale up or down. Containers should run on a cloud service or your own hardware — and have them just run at whatever scale is necessary, never going down and never paging the Ops team. This is what people call orchestration.

Defining Automation and Orchestration

Doug Davis gave us his thoughts on defining the automation and orchestration space:

> “*I think largely, when we think about automation, you think about writing the scripts that are integrated into a platform, like Chef, Jenkins or Ansible ...that is actually driving the actual behavior; and we think about orchestration as the platforms themselves that are providing that facility to be able to orchestrate the order in which things are going. That’s the orchestration. The automation itself is just the actual execution of the point-in-time script.”*

Many of our first questions to experts addressed this same distinction between automation and orchestration, and experts had numerous ways of thinking about it. Ben Schumacher, Innovation Architect at Cisco, agreed with the inherent relationship between automation and orchestration. “We quickly learned that, while there is some separation into what experts and users most closely associate with each label, they are serving essentially the same purpose,” he said. His colleague, Ken Owens, chief technology officer of cloud infrastructure services (CIS), described a more detailed thinking about the two strategies:

“As you move ... to this new container ecosystem, you’re seeing all of that underlying infrastructure becoming infrastructure as code,” Owens said. “And the ecosystem around containers, and Mesos, and then Kubernetes around orchestration and scheduling with Marathon as well, brings in a whole new interesting layer. And from Cisco’s standpoint, we’re very interested in not just what’s happening in that layer from a cloud-native development standpoint, but we're also interested in what are the enterprise-like feature sets around quality, and around workload and cluster management, to grant the granularity that our customers require. And how do we enhance that with better networking capability, better service discovery and service management capability, and better security capability?"

Creating New Architectures and Pipelines

Orchestration helps to complete the end-to-end DevOps pipeline in many ways. For the developer, it starts with the local environment on their laptop. But for the platform to work, it necessitates automation of the entire infrastructure.

It is not just about continuous integration and continuous delivery. It's about container operations, which speaks to the concepts of DCOS, the datacenter operating system developed by Mesosphere.

DCOS provides an operating system that abstracts the resources of an entire cluster of machines and makes them available to the developer like one big box, said Michael Hausenblas, a datacenter application architect and DevOps advocate at Mesosphere. Frameworks run on DCOS.

> *"For many people out there, I think the main thing, really, is how can you realize this next step after Docker build, Docker run? How to really put the containers into production and keep it running and fulfill these AppOps [application operations] tasks? Part of it is naturally the CI/CD pipeline. Again, for us, that's the thing where Mesos is kind of unique and able to run all kinds of different, what we call, frameworks."*

For example, Jenkins may cover the CI/CD pipeline, together with Cassandra, Kafka and Spark that handle analytics, with some web server from NGINX that serves the website. All of these different applications and frameworks in your entire life cycle can run together in one cluster.

Container orchestration will mostly require developers to just start making new projects, tools, and solutions, Hausenblas said. How people learn to manage container orchestration will, in turn, transform how they think about automated environments.

Containers make for a realistic mechanism to build these new architectures, but with implementation comes a need for new tooling and self-healing to manage how the systems work across distributed platforms.

“To me, an orchestration platform is a platform that can orchestrate multiple other system tools, orchestration engines — I would be talking about specific schedulers at that point, things like Zookeeper and Mesos and Marathon,” Owens said. “Things that are kind of the end point to what you’re trying to orchestrate, plus configuration management systems, plus automation tools, tooling systems or platforms. So it’s kind of like orchestrator of orchestrators, orchestrator of runtime and configuration management tools or toolsets.”

And there are so many tools still needed. Yelp, for instance, discovered issues with “zombie processes” spawned by signaling issues when using containers. To eliminate the signaling issues, Yelp developed dumb-init, an initialization system that runs inside of Docker containers.

The clear need for tooling comes with the assumption of DevOps practices in container orchestration environments. As we have stated in our own research, it becomes apparent that addressing the needs of specific job roles is essential when considering tool environments. In our survey of container users, 58 percent said integrated tools for both application development and IT operations are extremely important.

Discovering What Tools Are Needed

The tooling question will take some time to define as the use cases for container orchestration are just emerging. And due to the immaturity of the platforms, many expert users are just now discovering what tools are needed for these distributed platforms to operate efficiently.

Cluster management for scaled-out container orchestration can still be rudimentary. Its immaturity means that issues, such as prioritization of workloads, are yet to be completely defined.

Definitions for cluster management require policies for who can manage the clusters, and when, where and what mechanisms are needed for it, said Ken Robertson, lead architect at Apcera. These issues can be defined with cluster administration tools, but then there are the problems that come with rolling out and the automated communications between different aspects of the technology stack.

> “It’s a lot of automation in terms of managing the individual blocks, but also the entire facade.

Despite the effort to move to application architectures, it’s the machines making up the clusters, that defines how the provisioning is managed.

"A lot of that falls on — what are the machines making up this cluster environment?" Robertson said. "If you're running on AWS, GCE, or cloud environments, you have APIs to be able to provision more machines. In some case, say you provision machines with a lot of CPU and RAM, but are talking about hardware mapping, you have something that can only be consumed by one thing at a time."

This means the user will have a limit on how many of these workloads they can run at a time. The operator will need visibility into the limitation around that resource and the ability to provision more if needed. But the resources involved in provisioning aren't infinite.

"How do you turn on more, and how do you — in some cases — procure more of them where there might be a few weeks lead time to get them?" Robertson asked.

Herein lies the complexity of distributed systems. There are different generations of hardware architecture that have to be managed. And with cluster management, there's also the interest in managing cost. It may be that a customer wants to be notified if there is a price drop on a service. The input would be fed into the orchestration platform to take advantage of the pricing. The integrations would come through the third-party services, eliminating the need for one end-all interface.

Addressing the Complexity of Scale

The complexities with scale speak to the need for autonomic systems, said Alex Heneveld, co-founder and chief technology officer at Cloudsoft. It's a concept dreamed of for decades, but with container-based workloads, there's at least now a discussion point, as humans can't manage scaled-out architectures and fix them manually. There becomes the need for self-healing environments that can break complex tasks into smaller tasks, subsystems that generally do what they need to do.

In autonomic systems, a sensor is emitted when there's a problem. A management platform analyzes the sensors. Effectors, also described as levers, exist to make changes. It's this combination of sensors giving metrics, and effectors letting users control things, that is the encapsulation of a system.

"Systems can be managed by another system that itself has sensors and effectors," Heneveld said. "Within it, it's doing the monitoring, analyzing, planning and executing to take care of the systems underneath it. But you can almost view it like an org chart or a hierarchy. We have it looser than some very strict theories in this realm, but the idea is you can make very complex systems just by composing simpler systems. If we look at what's happening around microservices, we're building up complex systems from simpler systems."

Apache Brooklyn is an open source project that, at its core, is built on the principles of autonomous systems. It allows the user to unpack a complex system to look inside it. The user can keep unpacking the system to look deeper inside, all the way to the plumbing underneath, if need be.

"If I'm lucky and someone is hosting some of my substrate, I don't need to look that far," Heneveld said. "If I've got a hosted container service, for instance, I'm not interested in the plumbing underneath it. But I do want to know that every one of those containers is doing the right thing. So, the metrics that we get back from each piece becomes absolutely essential to informing that model, allowing us to maintain a healthy system, and correct it, and improve it when it’s needed, whether that’s scaling, or failing over, or DR [disaster recovery]. And in our world, those are implemented as policies, where this ‘monitor, analyze, plan, execute’ logic sits within the autonomic framework.”

Conclusion

For users and vendors existing between a VM-dominated world and a container-based one, it’s not a problem; rather, it’s a space rich with emerging practices, technologies, and solutions to problems. This balance of current forces is bringing the Dev and Ops teams closer together than ever before. And even as the space begins to change, and reviews about the role of VMs and containers in automated environments at scale matures, there is still much to be learned about managing virtualized environments; this space is still a new one for many users and vendors alike, and there are many unknowns, but that just places greater emphasis on the need to manage and orchestrate the various components.

From automation and orchestration to scheduling, cluster management, service discovery, security and more, it all adds up to a change in how we think about automation and orchestration. Embracing this change, and learning to focus on developing best practices, is a larger journey for the entire container ecosystem; a journey fueled by open source software, the need for connectivity, passionate communities of users, and economic factors in the business models of vendors.

SHAPING CONTAINER USAGE: EDUCATION, OPEN SOURCE & STANDARDS

Davis talks about IBM’s role early on in the life of containers, discovering how to improve core technologies (through projects like runC, the Open Container Initiative, and the Cloud Native Computing Foundation), and how IBM helps customers assess orchestration tools. **Listen on SoundCloud** or **Listen on YouTube**

***Doug Davis** is a senior technical staff in the cloud and open source standards division of IBM. He is also involved in open source projects and standards such as Cloud Foundry, Apache Axis, CIMI, and SOAP-related specs, as well as standards bodies such as W3C, OASIS and DMTF.*

Ferris talks about the success of products based on how they can generate and sustain an open source ecosystem around their tooling, and about obstacles facing the container ecosystem at large, including feedback about the functionality of networking and storage technologies. **Listen on SoundCloud** or **Listen on YouTube**

***Christopher Ferris**, a distinguished engineer and CTO of open technology in IBM Cloud, has technical responsibility for strategic open technology initiatives: OpenStack, Cloud Foundry, Open Container Initiative, Cloud Native Computing Foundation and more.*

PEER PERSPECTIVES ON CONTAINER ORCHESTRATION SURVEY

by **LAWRENCE HECHT**

After watching a year's worth of conference presentations, it is easy to become inured by the hype around products that orchestrate containers. Yet, there still confusion in the broader tech community about what functionality is involved with orchestrating containers. The New Stack ran a survey that was able to target users of container management technology, with 70 percent of respondents using containers to some degree. We found that there is indeed uncertainty about what it means to manage and orchestrate containers. Yet, there is also a developing consensus that scheduling, cluster management and service discovery are necessary to orchestrate the use of containers in production.

Over 40 percent of respondents using or trialing containers said an orchestration platform was the primary way containers are managed in their organization. However, the actual products being used or planned for were strongly influenced by a respondent's stage of container adoption. People that have non-production use of containers were more likely to cite configuration management tools like Ansible and Chef as a method of orchestration. Platforms as a service (PaaS) solutions, OpenShift, and HashiCorp products were cited more often in their roadmaps among those evaluating containers.

There is a wide mix of companies and services in users' plans. It remains to be seen what tools will actually win out. For now, it appears that many of the container orchestration-specific tools, like Kubernetes, Mesos and Docker Swarm, as well as the Containers as a Service offerings, are making sure that they provide a wide range of open source tools within their bundled offerings. Yet vendors should realize that most users are not looking for one solution to solve all their needs — only 29 percent said it is important for a container orchestration tool to be able to handle non-containerized and legacy applications. Instead, what they care most about it is integrating tooling for both developers and IT operations (ops).

Sample and Methodology

How the Data Was Collected

The survey was administered via the web from February 21 through March 14, 2016. As with all anonymous web-based surveys, the results are likely affected by self-selection bias. All responses were manually reviewed for integrity and several were discarded. The data in this chapter is based on 309 respondents, 75 percent of which completed the entire survey.

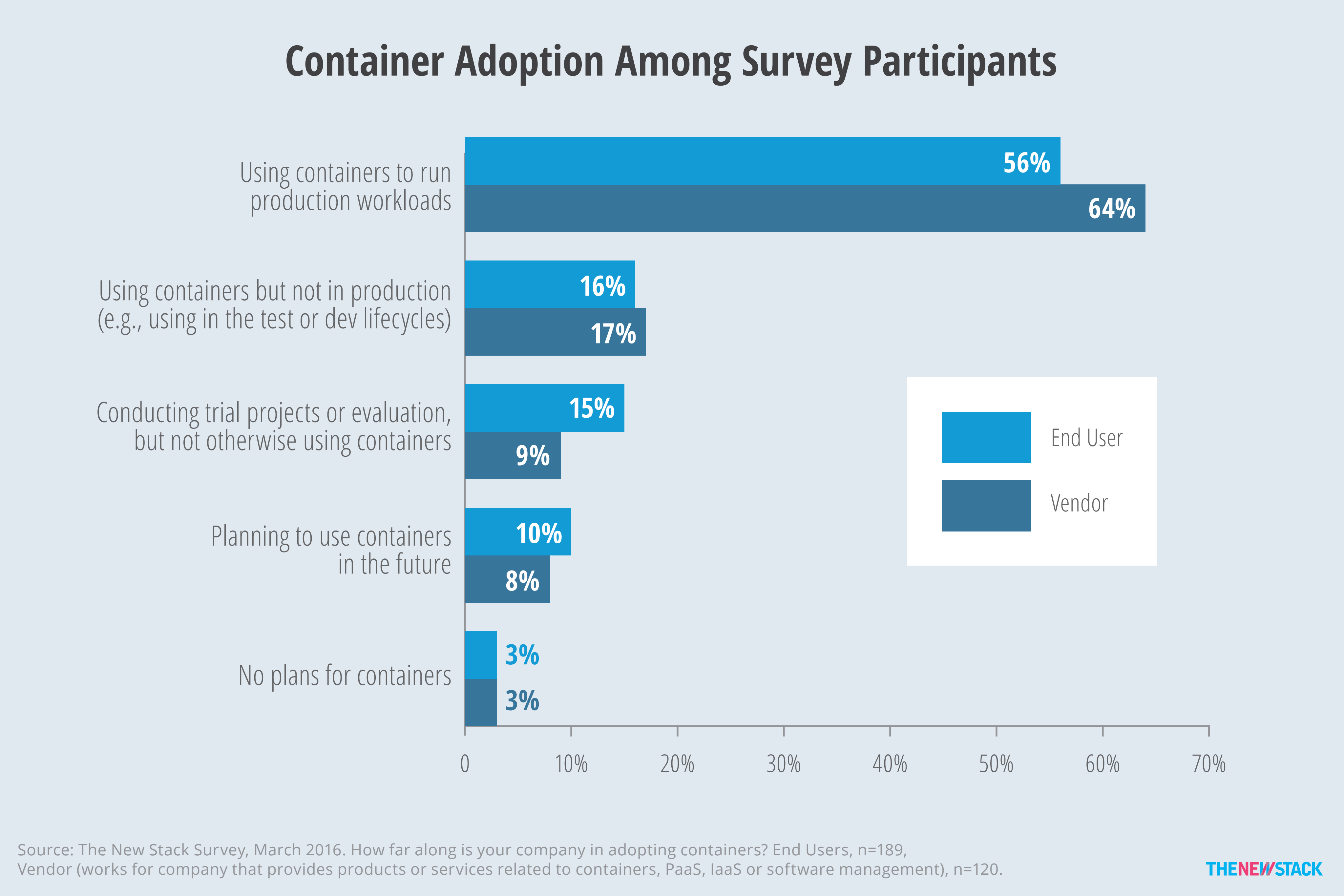
The New Stack readership was asked to participate in the survey. The study was also promoted via social media. In addition, we received 54 responses due to publicity in DevOps Weekly and another 42 as a result of an email sent by an executive in Mesosphere's marketing department. When relevant, the analysis notes how this may impact the findings.

Profile of Study Participants

Sixty-one percent of respondents were end users, with the remainder working for vendors and service providers associated with containers, PaaS, infrastructure as a service (IaaS) or software management.

In order to prevent vendors from voting for their own products and services, questions about tools or products being used or considered were only asked of end users. Furthermore, these questions were only presented to the 164 end users that were using and/or at least trialing or evaluating containers.

FIG 1: *Seventy-one percent of surveyed end users are using containers to some degree.*

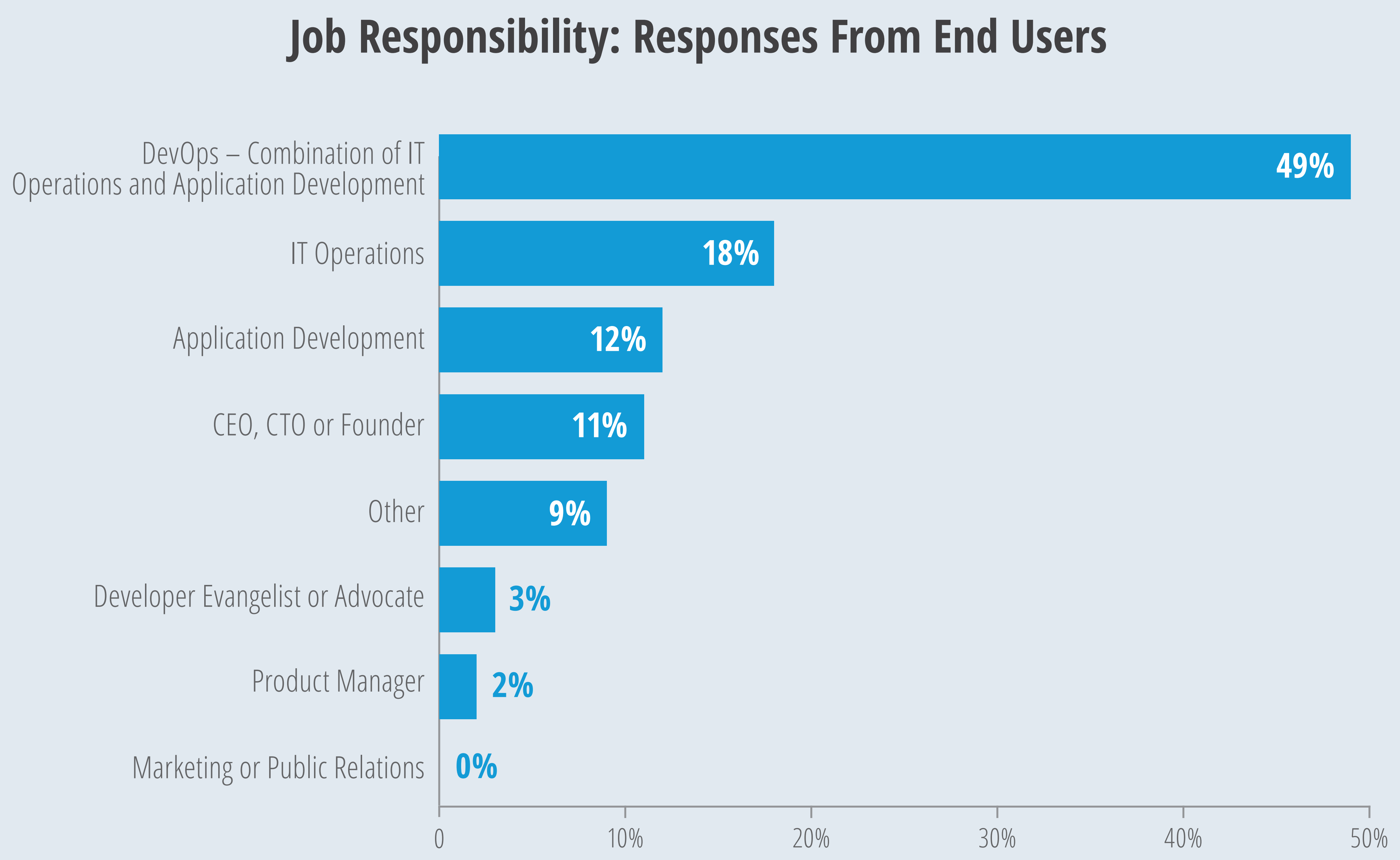


Source: The New Stack Survey, March 2016. How far along is your company in adopting containers? End Users, n=189, Vendor (works for company that provides products or services related to containers, PaaS, IaaS or software management), n=120.

Compared to the broader market, the survey was more likely to talk to early adopters, as 59 percent said they use containers in production, with another 11 percent using containers in a more limited manner, such as in the test part of the development lifecycle. It is somewhat surprising that only 64 percent of vendors said they are using containers in production.

Almost half of the end users described themselves as DevOps, and upon investigation we found this was not because of responses received from DevOps Weekly subscribers. Although the growth of DevOps is real, the role is mostly likely strongly represented in this study because container orchestration is an area that requires coordination between application development and IT operations.

FIG 2: *DevOps pros are well represented.*



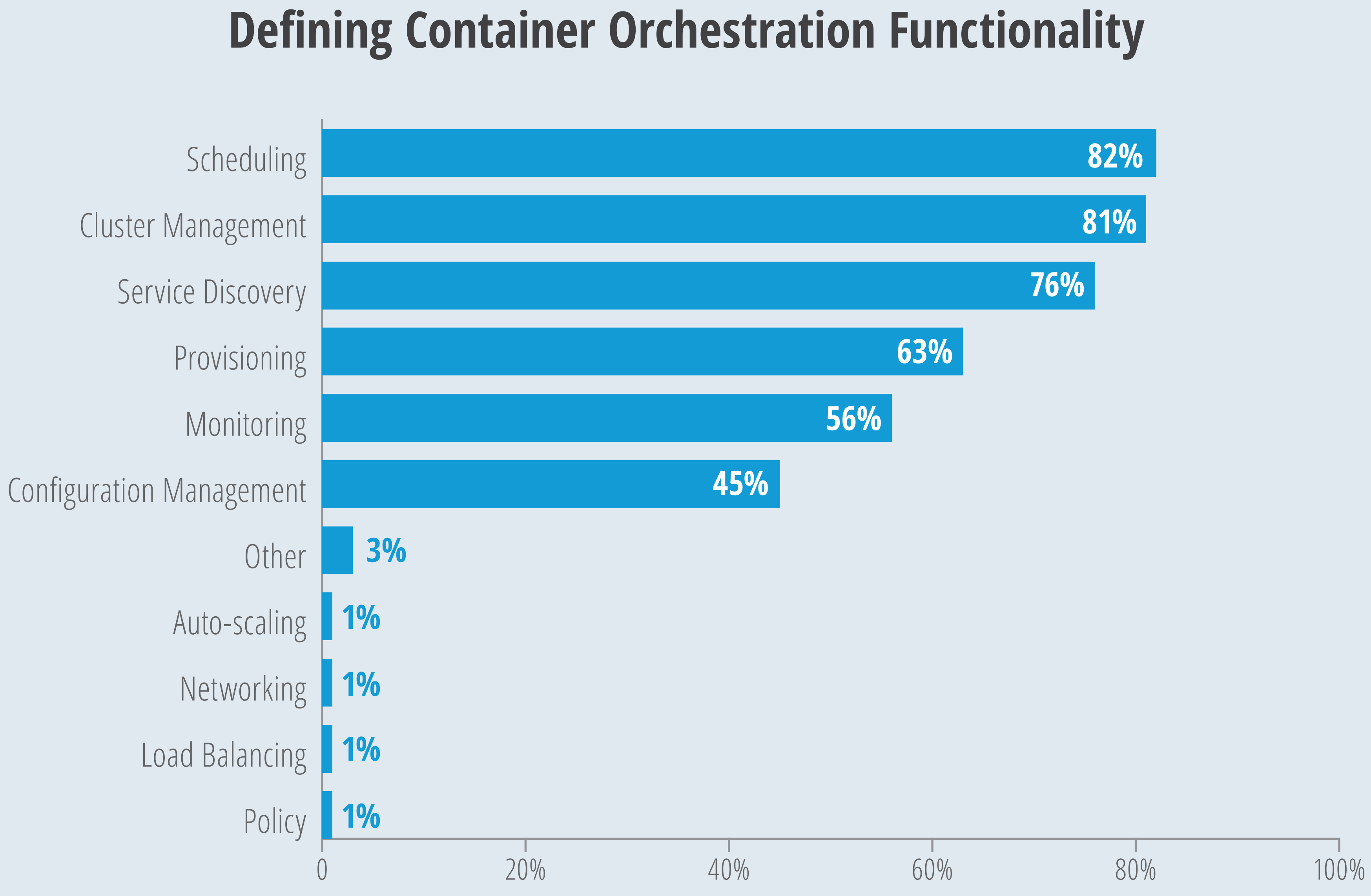
Source: The New Stack Survey, March 2016. What is your primary job role? n=179.

Defining Container Orchestration Functionality

When we started writing this book, there was little agreement about what exactly defines container orchestration. So, instead of trying to create a top-down definition of the market, we decided to ask the community about their opinion.

There is consensus that scheduling, cluster management and service discovery are core. Four out of five people said scheduling and cluster management are expected in a product or service related to container orchestration. Close behind, 76 percent associated service discovery with

FIG 3: *Only 45 percent of respondents consider configuration management to be part of a container orchestration product.*



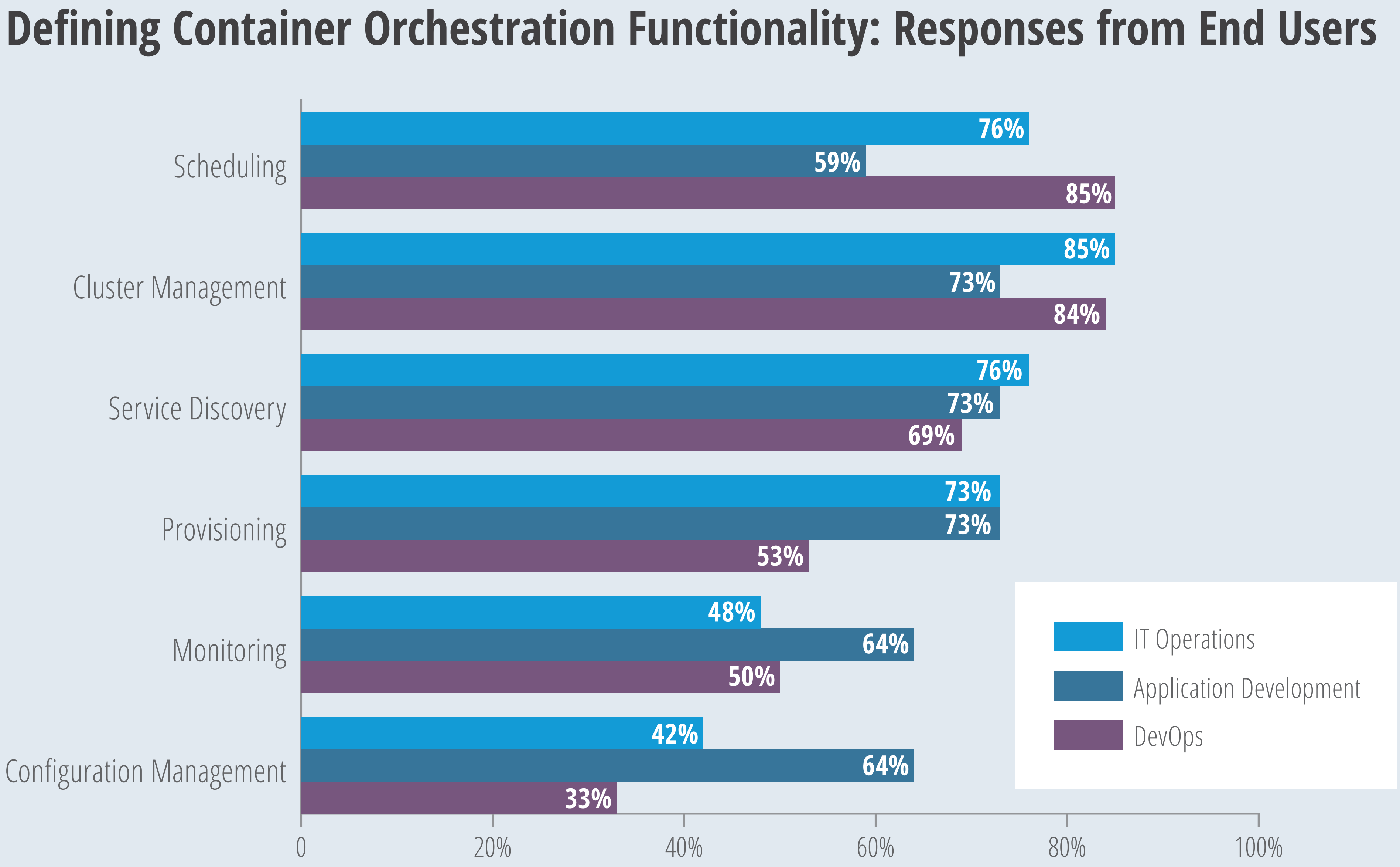
Source: The New Stack Survey, March 2016. What functionality do you expect to be in a product described as a container orchestration tool? Select all that apply. n=307.

THENEWSTACK

container orchestration. Monitoring, and especially configuration management, were the least likely functionality associated with container orchestration. Auto-scaling, load balancing and networking were all definitions that were proffered by multiple people in the the "Other" box.

A couple of variables affected perceptions, with job role having a significant impact on how people define container orchestration. In particular, self-identified application developers were much less likely to associate it with scheduling (59 percent) and more likely to associate it with monitoring (64 percent) and configuration management (64 percent). Their lack of involvement with monitoring and configuration management may be why they cited the functionality more often. In addition to job role,

FIG 4: *Scheduling is not top-of-mind when app developers think of container orchestration.*



Source: The New Stack Survey, March 2016. What functionality do you expect to be in a product described as a container orchestration tool? Select all that apply. IT Operations, n=33; Application Development, n=22; DevOps – Combination of IT Operations and Application Development, n=94. Open-ended responses excluded from the chart.

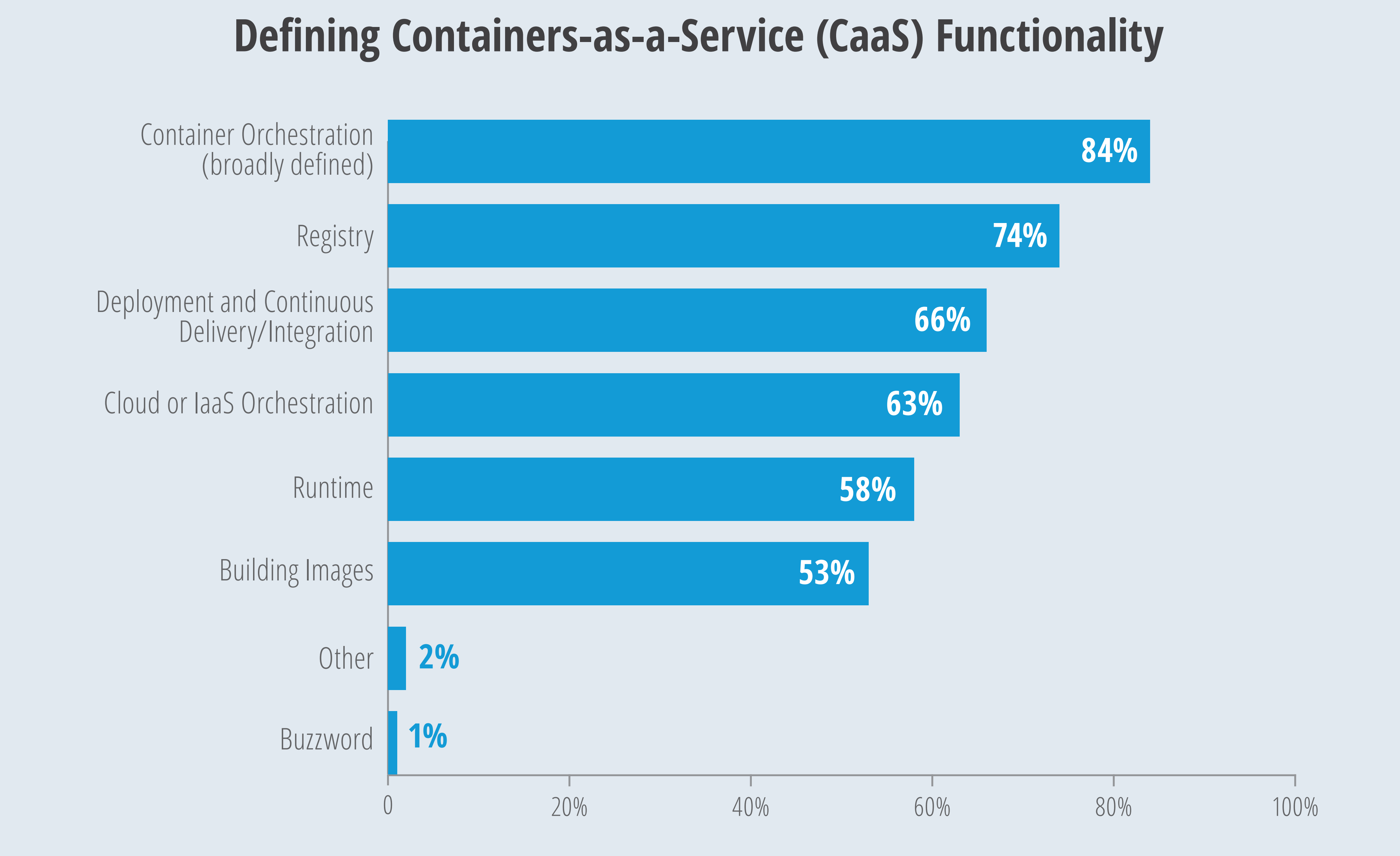
THENEWSTACK

we also found that the people with a Mesosphere perspective – those that participated in the survey due to a company employee's promotional email – were more likely to answer scheduling (93 percent) and monitoring (74 percent). Among those Mesosphere-centric respondents, 72 percent said they use Apache Marathon for scheduling.

Defining Containers as a Service Functionality

Like PaaS before it, there is considerable disagreement about what Containers as a Service (CaaS) means. It is important to note that The New Stack asked about some, but not all the types of CaaS functions. We found that container orchestration was the functionality most associated with

FIG 5: *Container orchestration and registries are most associated with CaaS.*



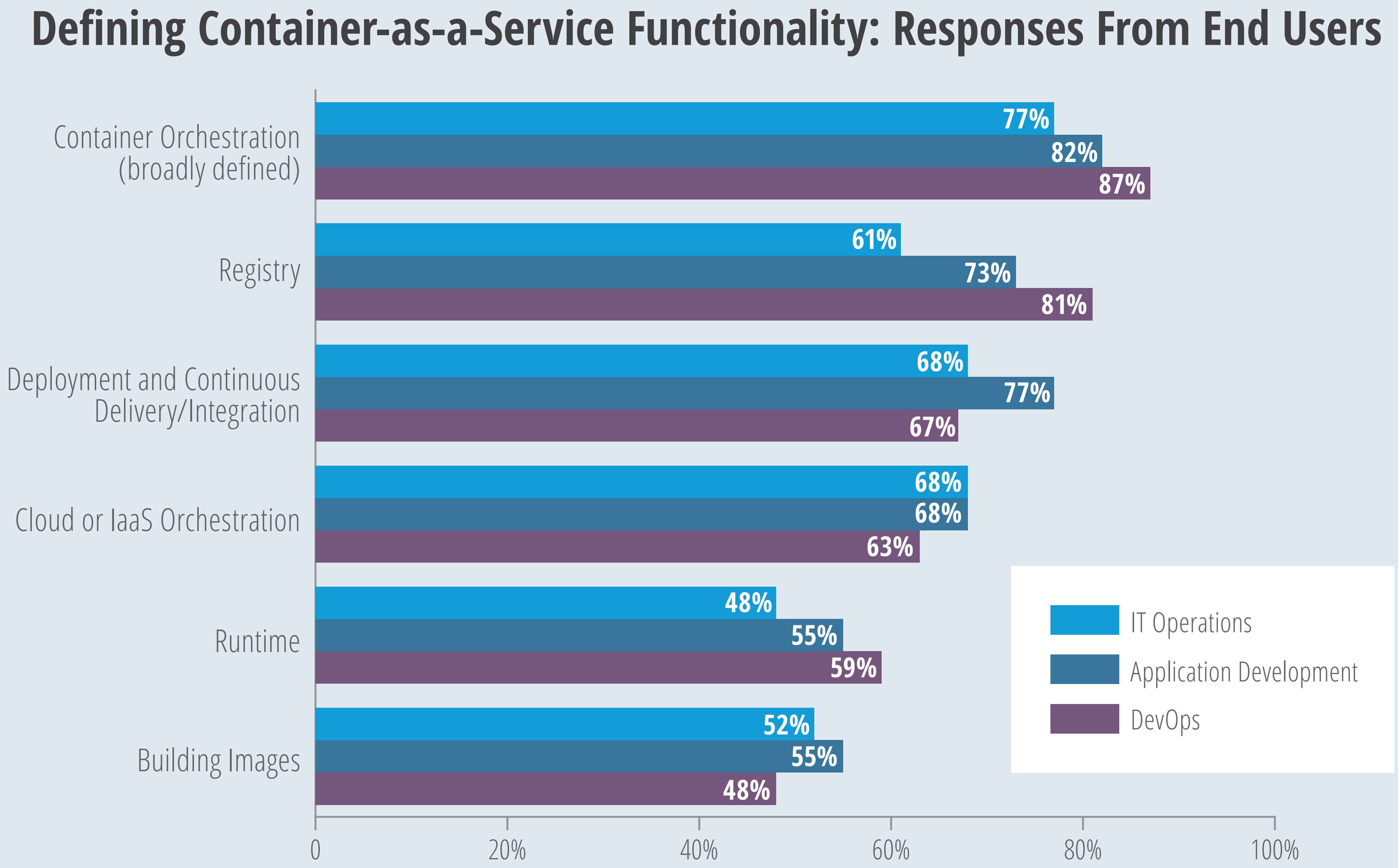
Source: The New Stack Survey, March 2016. What functionality do you expect to be in something described as Containers-as-a-Service (CaaS)? Select all that apply. n=298.

THENEWSTACK

Containers as a Service. However, since the survey's topic was container orchestration, it is likely that the responses were biased towards that response. Registry, the place where containerized content is accessed, was also strongly correlated with CaaS. Building images and runtime were least likely to be associated with CaaS, yet still was a majority perception. In other words, CaaS is viewed by a majority as an end-to-end offering.

Job role also affected understanding about CaaS. Compared to DevOps, people responsible for IT operations were less than likely to say registry (61 vs 81 percent), and to a lesser degree runtime and container orchestration, are part of a CaaS package. Those responsible for IT operations were less likely to be using containers, so they were probably less informed about why registries can be an important part of a container

FIG 6: *IT operations teams' perspective on CaaS differ from those doing DevOps.*



Defining Container-as-a-Service Functionality: Responses From End Users

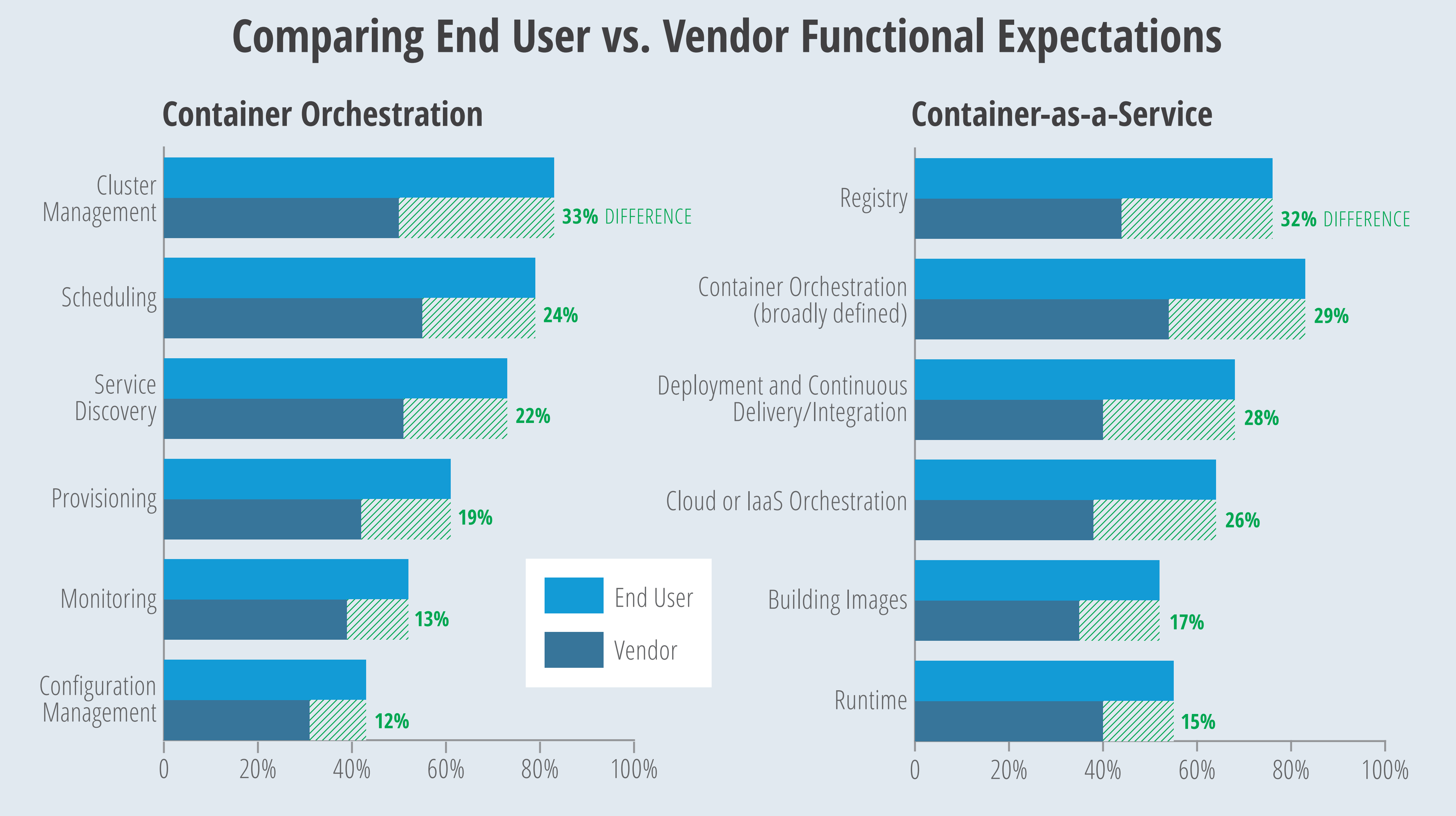
Source: The New Stack Survey, March 2016. What functionality do you expect to be in something described as Containers-as-a-Service (CaaS)? Select all that apply. IT Operations, n=31; Application Development, n=22; DevOps – Combination of IT Operations and Application Development, n=91. Open-ended responses excluded from the chart.

service. Possibly because they are not as tied into the container ecosystem, participants solicited from the DevOps Weekly newsletter were less likely to say container orchestration and runtime are part of CaaS.

Outside of Vendor Territory, Confusion Abounds

Employees of companies associated with the container ecosystem were much less likely to check most of the boxes when asked to define functionality associated with either container orchestration or CaaS. In particular, fewer vendors said cluster management equates to container orchestration, and that registry, deployment and cloud orchestration are components of CaaS. One reason employees of vendors involved with

FIG 7: *Vendors were parsimonious when defining container orchestration and CaaS.*



Source: The New Stack Survey, March 2016. What functionality do you expect to be in a product described as a container orchestration tool? Select all that apply. End Users, n=188, Vendor (works for company that provides products or services related to containers, PaaS, IaaS or software management), n=119. What functionality do you expect to be in something described as Containers-as-a-Service? Select all that apply. End Users, n=183, Vendors, n=115. Open-ended responses excluded from chart.

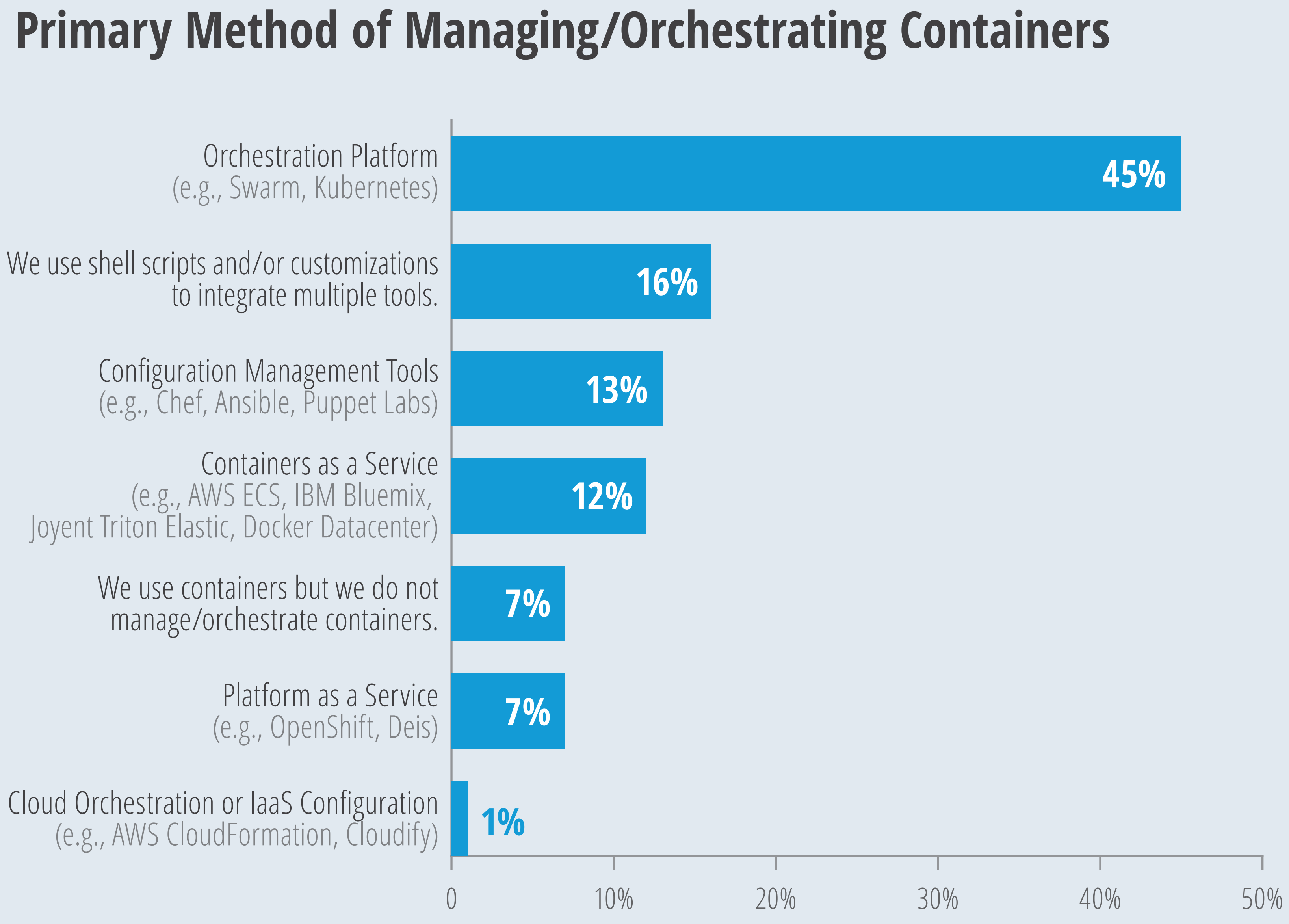
THENEWSTACK

containers are more discriminating in their definitions is that they have likely spent more time attending events and reading about the subject. Another reason may be that a broader definition of the market does not fit into how they are marketing their own products. No matter why the differences arise, they point to the fact that beyond industry insiders, there is a continuing need to define the market.

Products/Services Used to Manage and Orchestrate Containers

As noted in several of our previous articles, no matter how you slice it, the use of containers is rising and with it the need to orchestrate their

FIG 8: *Platforms focused on orchestration are the primary method of managing containers for 45 percent of end users that are using or trialing containers.*



Source: The New Stack Survey, March 2016. What is the primary way containers are managed or orchestrated in your organization? Please select all that apply. n=138.

THENEWSTACK

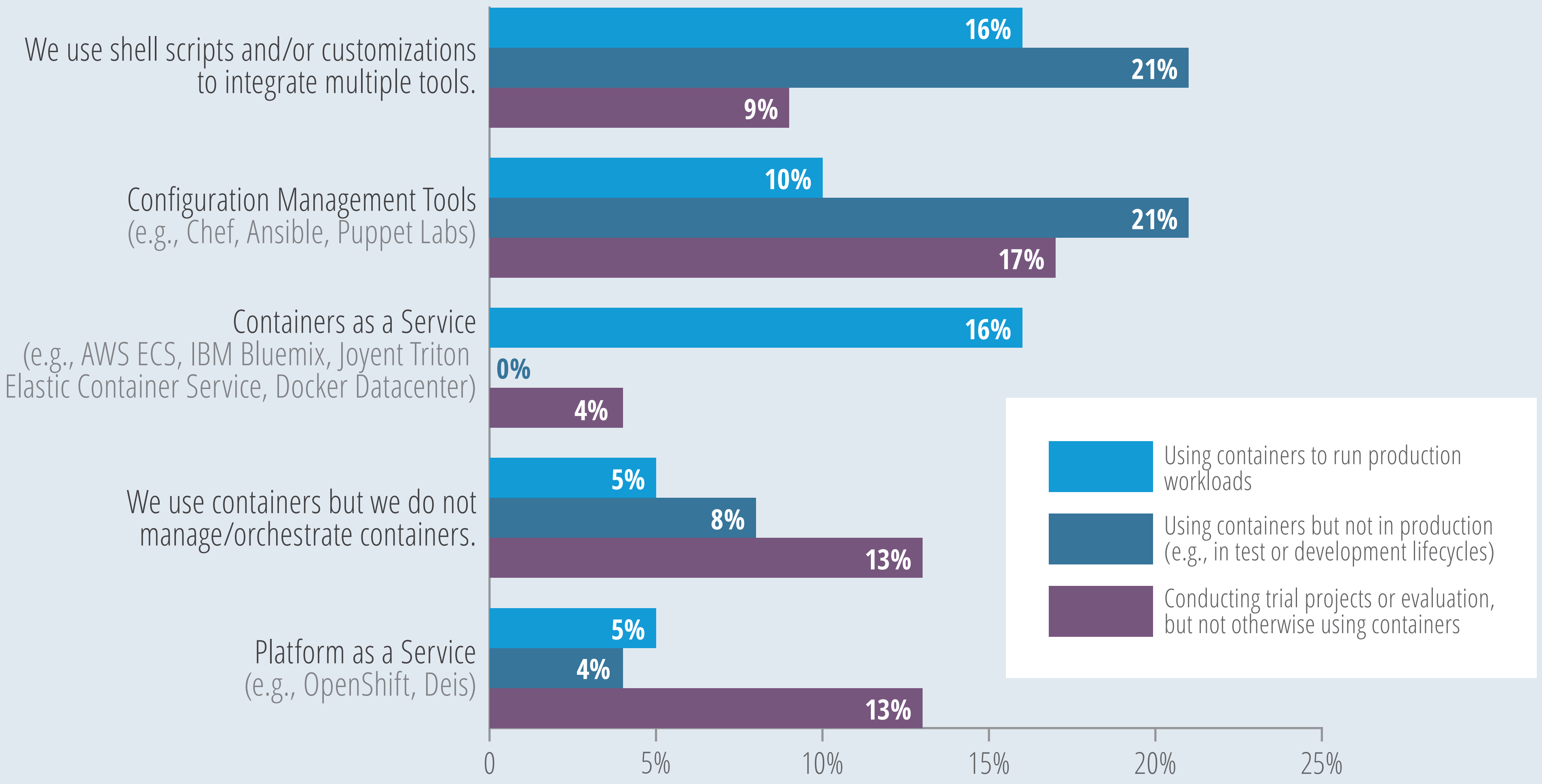
deployment in production. Since there is still confusion about what container orchestration means, we made sure to ask broadly about what is being used to either manage or orchestrate containers. The questionnaire first asked about the primary method of orchestration or management, and then asked about the top three products or services to be used in the next year. This methodology forced respondents to prioritize between multiple methods and offer some specificity about their future roadmaps.

One consequence of this approach is that only 16 percent of respondents primarily manage containers with shell scripts or customizations, which is significantly lower than other studies where respondents were able to provide multiple choices. While customizations may glue together multiple tools, they are usually not in and of themselves the main way to manage something.

Status of container adoption also correlates with the primary way of orchestrating containers. Respondents that have started to use containers, but not yet in production, were a bit more likely to use configuration management tools (21 percent) and shell scripts (21 percent). However, as users move forward on their “container journey,” the numbers drop to 16 and 10 percent respectively, possibly because the need for a more consistent, scalable solution becomes apparent as production use increases. Note that subscribers to DevOps Weekly were more likely to be using configuration management tools (29 percent), which is not surprising considering its publisher’s day job is with Puppet Labs.

Looking at PaaS and CaaS, it appears that PaaS brand names like OpenShift and Deis are more likely (13 percent) to be top-of-mind among those conducting evaluations. Production users are more likely to say

Primary Method of Managing/Orchestrating Containers: Differences Based on Implementing Status



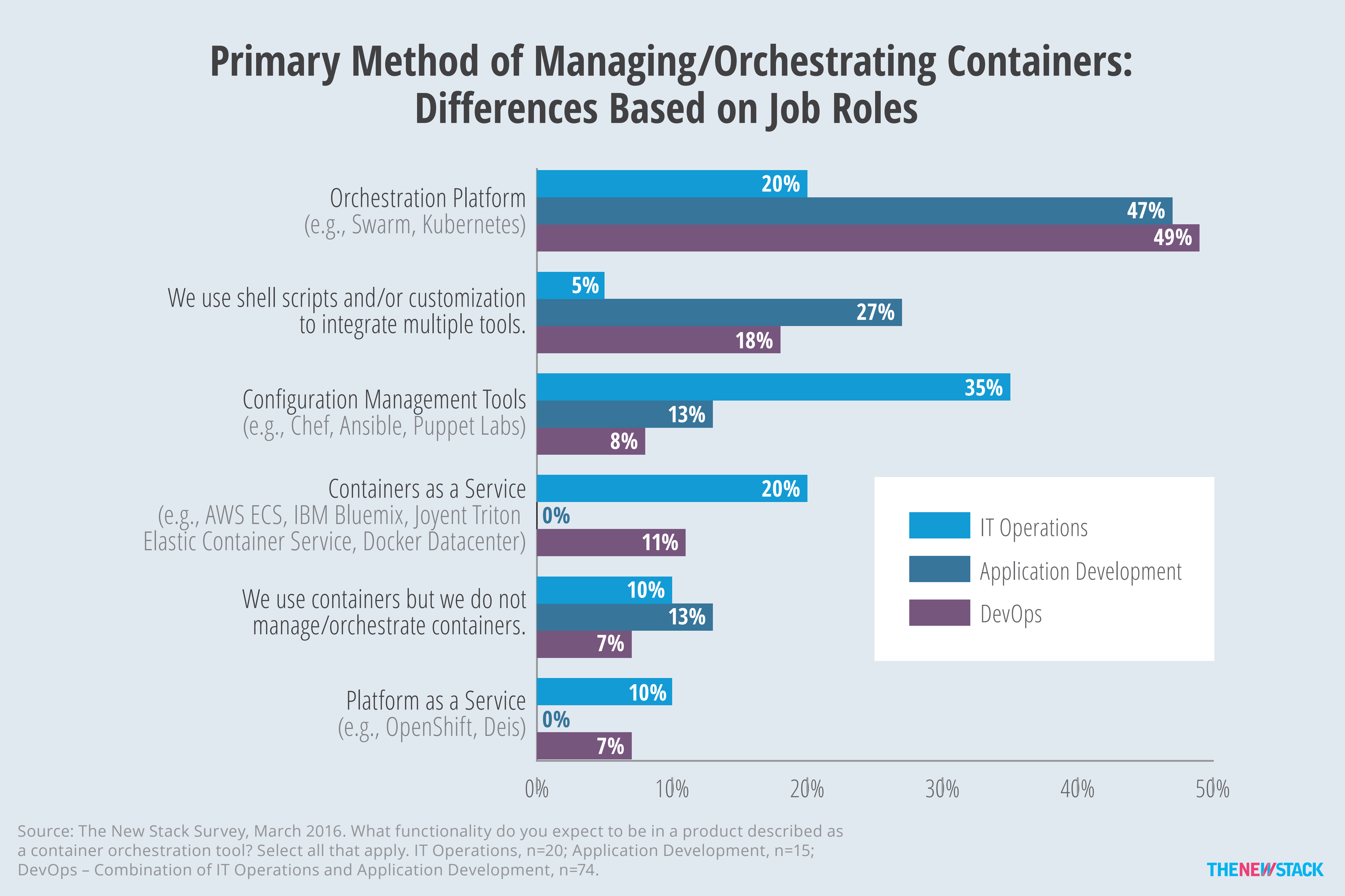
Source: The New Stack Survey, March 2016. What is the primary way containers are managed or orchestrated in your organization? Select all that apply. Using in production, n=91; Using, but not in production, n=24; Conducting trials/evaluation, n=23.

THENEWSTACK

FIG 9: *Use of configuration management tools to manage containers declines as enterprises move into production use.*

CaaS is their primary management method. Of course, as we discussed earlier, the definition of Containers as a Service is still up for debate. Some people may consider it to be the provision of container services or container hosting, while others may have a broader definition.

Despite all these other methods of managing containers, the leading choice by far was orchestration platforms like Swarm, Kubernetes and Mesos, which were cited by 45 percent of respondents as their primary method of managing containers. Also described as a framework, orchestration platforms offer users the ability to execute several types of orchestration functionality (i.e., cluster management, scheduling). At least for now, it appears that purpose-built platforms are the orchestration



Source: The New Stack Survey, March 2016. What functionality do you expect to be in a product described as a container orchestration tool? Select all that apply. IT Operations, n=20; Application Development, n=15; DevOps – Combination of IT Operations and Application Development, n=74.

FIG 10: *CaaS and orchestration platforms like Swarm, Kubernetes and Mesos compete to win over IT Operations from its configuration management tools.*

method of choice, but that may be due to the field's newness rather than because of a preference for one type of tool or another.

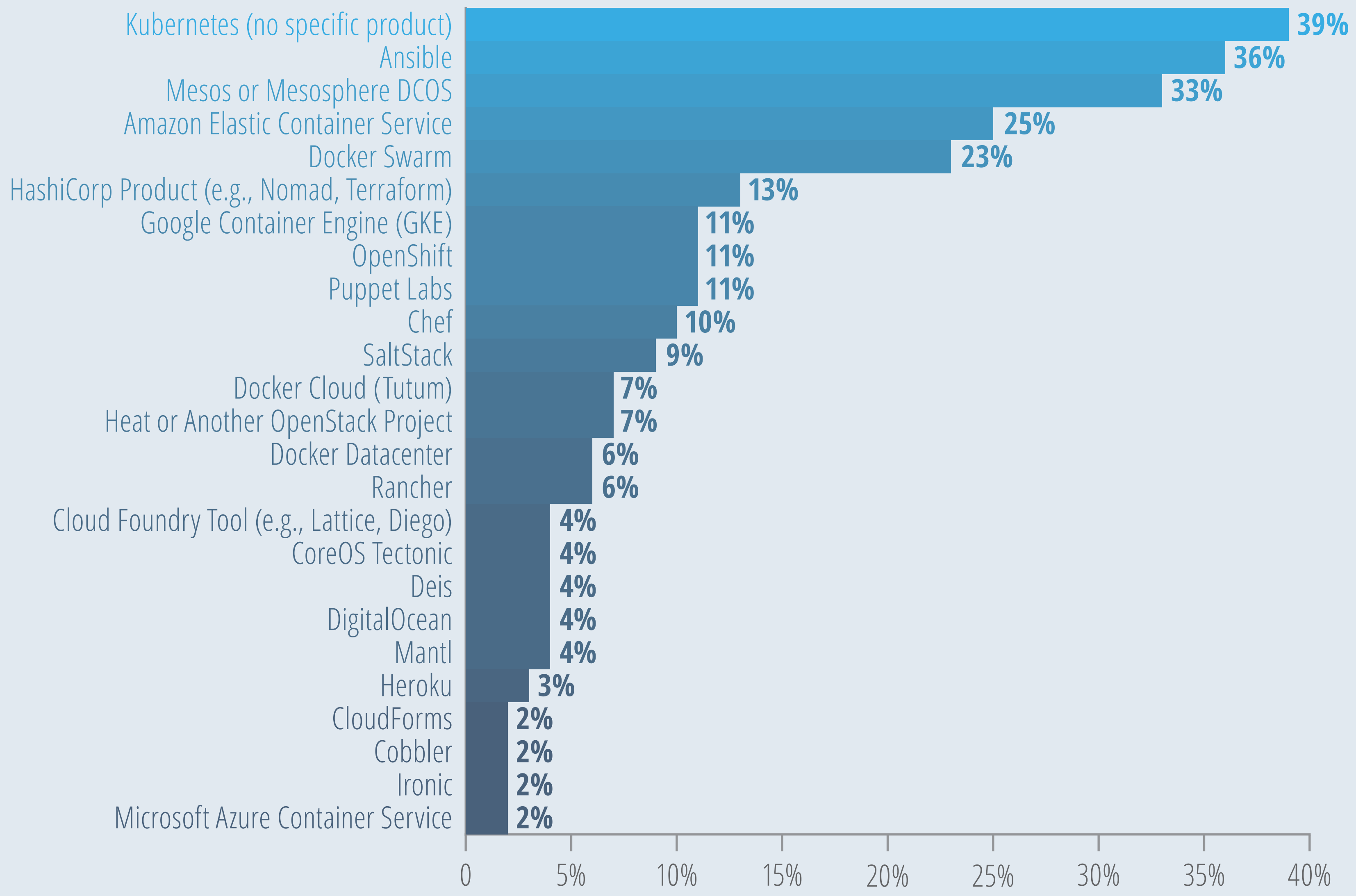
IT operations is a job role that doesn't seem as inclined to use orchestration platforms — fewer than half as many cited them as compared to everyone else. Instead, 35 percent of IT ops primarily use configuration management tools, compared to only nine percent among everyone else. In addition, IT operations is just as likely to look towards CaaS as to orchestration platforms. In comparison to those with an operations focus, respondents with application development roles were more likely (27 percent) to use shell scripts, which is not surprising since they are likely very comfortable hacking a custom solution together.

The names behind the categories reported in the charts were revealed by asking about the top three products in users' near-term plans as well as those currently being used. The top five choices were Kubernetes, Ansible, Mesos or Mesosphere DCOS, Amazon Elastic Container Service (ECS) and Docker Swarm. Kubernetes leads the list with 39 percent of respondents planning to use it. However, since it is not a purchasable product, it is hard to compare that result with the other entries in Figure 10. Another way to read the chart would be to identify products like Google Container Engine (GKE) and CoreOS Tectonic that are closely related to Kubernetes. Since many users of Amazon ECS also use Kubernetes, it is hard to determine how many of the 25 percent citing Amazon Web Services (AWS) plan to use it as their primary method of orchestration. The same issue arises with Mesosphere DCOS, which can be run on top of Amazon EC2.

One out of three respondents expect to use either Apache Mesos or Mesosphere DCOS. While the two are different, they were combined in order to account for respondents that think of them as synonymous. The sample likely over-represents plans to use Mesos because almost 50 percent of respondents citing Mesos or Mesosphere DCOS were solicited for participation by Mesosphere. However, even if those respondents are excluded, Mesos/Mesosphere would still be tied for fourth place with 23 percent.

The use of Docker for orchestration is understated if you only look at the 23 percent figure for Swarm. As stated earlier, the production-ready version of Swarm was announced while the survey was being conducted, but so was a new product called Docker Datacenter. Furthermore, Tutum was renamed as Docker Cloud as a production version went live. There was no overlap between people citing Docker Datacenter and Docker

Top Orchestration Products Based on Expected Usage Within Next Year



Source: The New Stack Survey, March 2016. Within the next year, what are the top three products or services you expect to utilize to manage or orchestrate containers? n=114.

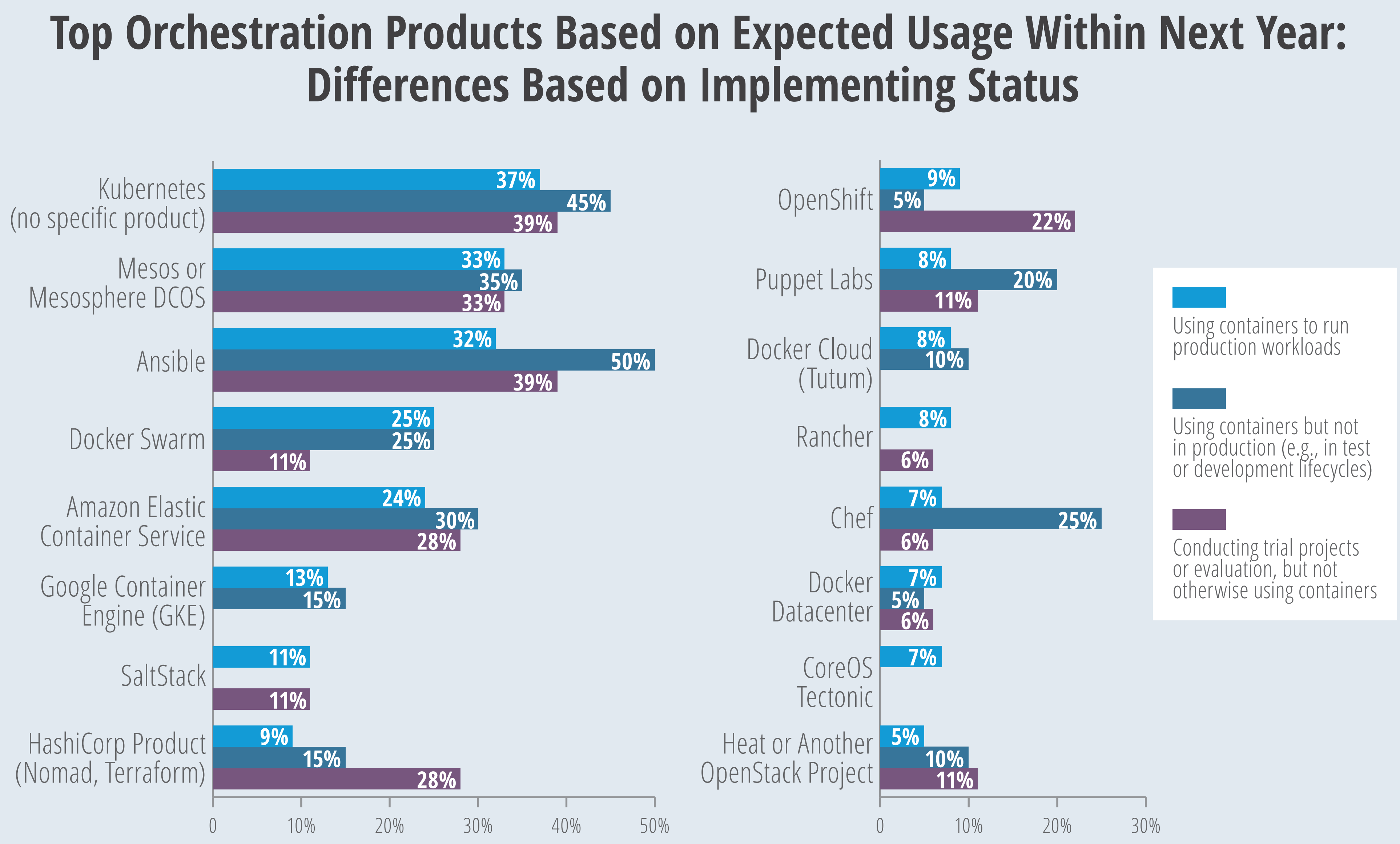
FIG 11: *Kubernetes, Ansible, Mesos/Mesosphere, Amazon ECS and Docker Swarm top users' plans for container orchestration.*

Cloud, but seven of those 15 respondents did cite Swarm as another of their choices. Thus, looking at Docker more broadly, 31 percent plan to use the company's products to manage containers.

Ansible was the choice of 36 percent of respondents, which surprisingly placed it second on users' roadmaps. Among the Ansible respondents, only ten percent chose configuration management as their primary method of orchestrating containers. It appears that Ansible will be used in conjunction with several other tools. Since Puppet Labs and Chef are also configuration management vendors, it is not surprising to see that all three are much more likely to be in the roadmaps of the respondents that are using containers but not in production. However, since Ansible is still number three among those using containers in production, we believe there are other reasons why Ansible does so well. Chief among them is its recent purchase by Red Hat. Already well-regarded, it is possible that many Red Hat customers will choose Ansible over its configuration management rivals as a way to integrate with other Red Hat offerings. This rationale may be the reason that seven of the twelve respondents citing Red Hat's OpenShift also plan to use Ansible.

Among the second tier of choices, people conducting trial projects or evaluations were more likely choose Red Hat's OpenShift and HashiCorp products. This is likely because many end users are already familiar with or using those companies' products. For example, the percentage

FIG 12: *Hashicorp and OpenShift are thought of more often among those conducting trial projects or evaluation.*



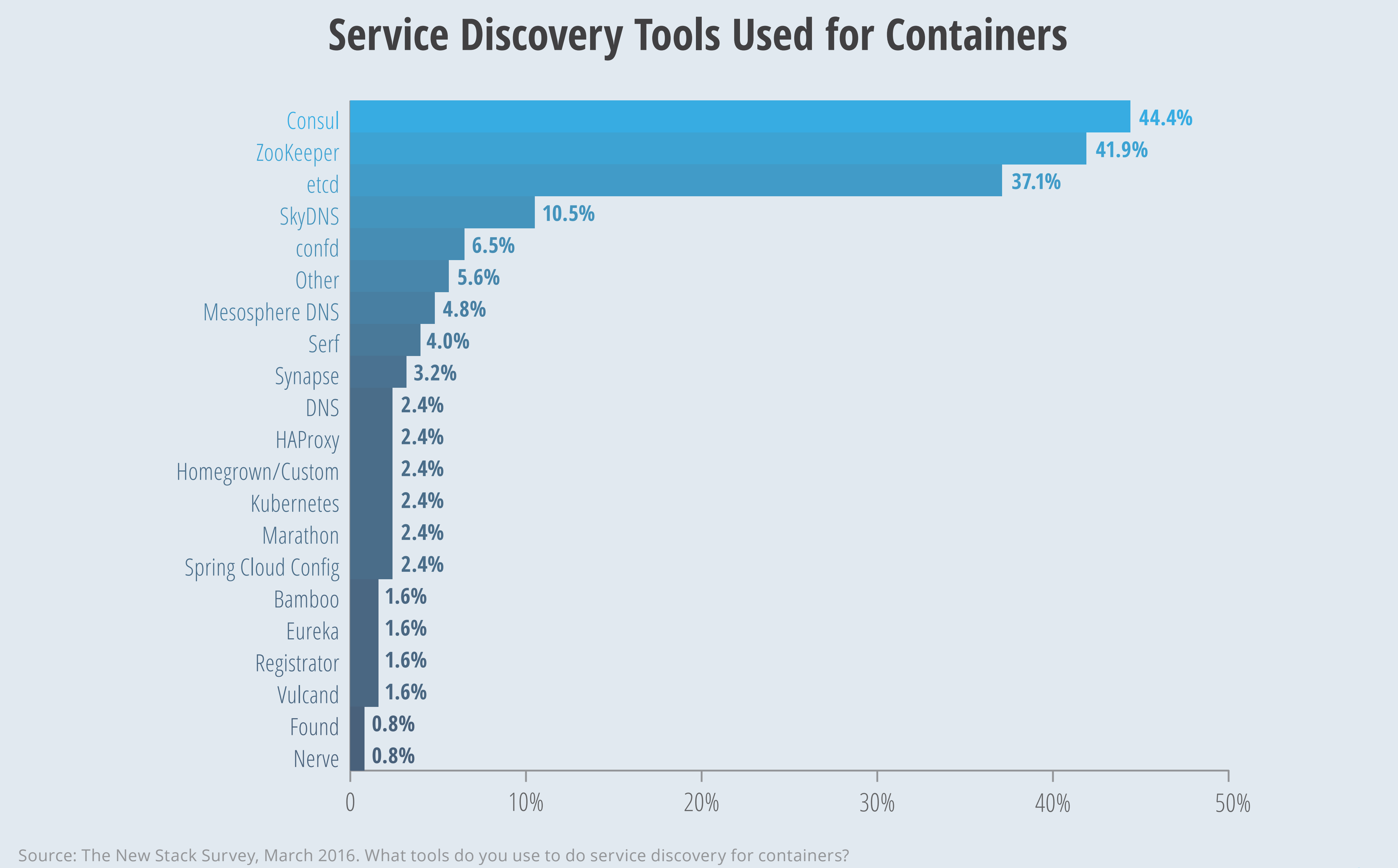
Source: The New Stack Survey, March 2016. Within the next year, what are the top three products or services you expect to utilize to manage or orchestrate containers? Select all that apply. Using in production, n=76; Using, but not in production, n=20; Conducting trials/evaluation, n=18. Choices with less than five responses are not shown.

planning to use a HashiCorp product rose from 13 percent to 21 percent when looking only at people that use HashiCorp's Consul for service discovery. Since HashiCorp's Nomad is meant for larger deployments, its use may go up as the scale of current container usage increases.

Tools Being Used to Perform Specific Functionality

Warning: *The following charts and analysis should not be used to analyze market share. Instead, view them as a gauge of user perceptions. Survey participants were offered a long list of tools that could conceivably be related to container management functionality. Although they were able to offer other, open-ended responses as well, this methodology may have influenced the results. Furthermore, no definition of what service discovery, scheduling or cluster management was provided.*

FIG 13: *Consul, ZooKeeper and etcd are used more often than other tools for service discovery.*



Source: The New Stack Survey, March 2016. What tools do you use to do service discovery for containers? Please select all that apply. n=124. Choices with less than two responses are not shown.

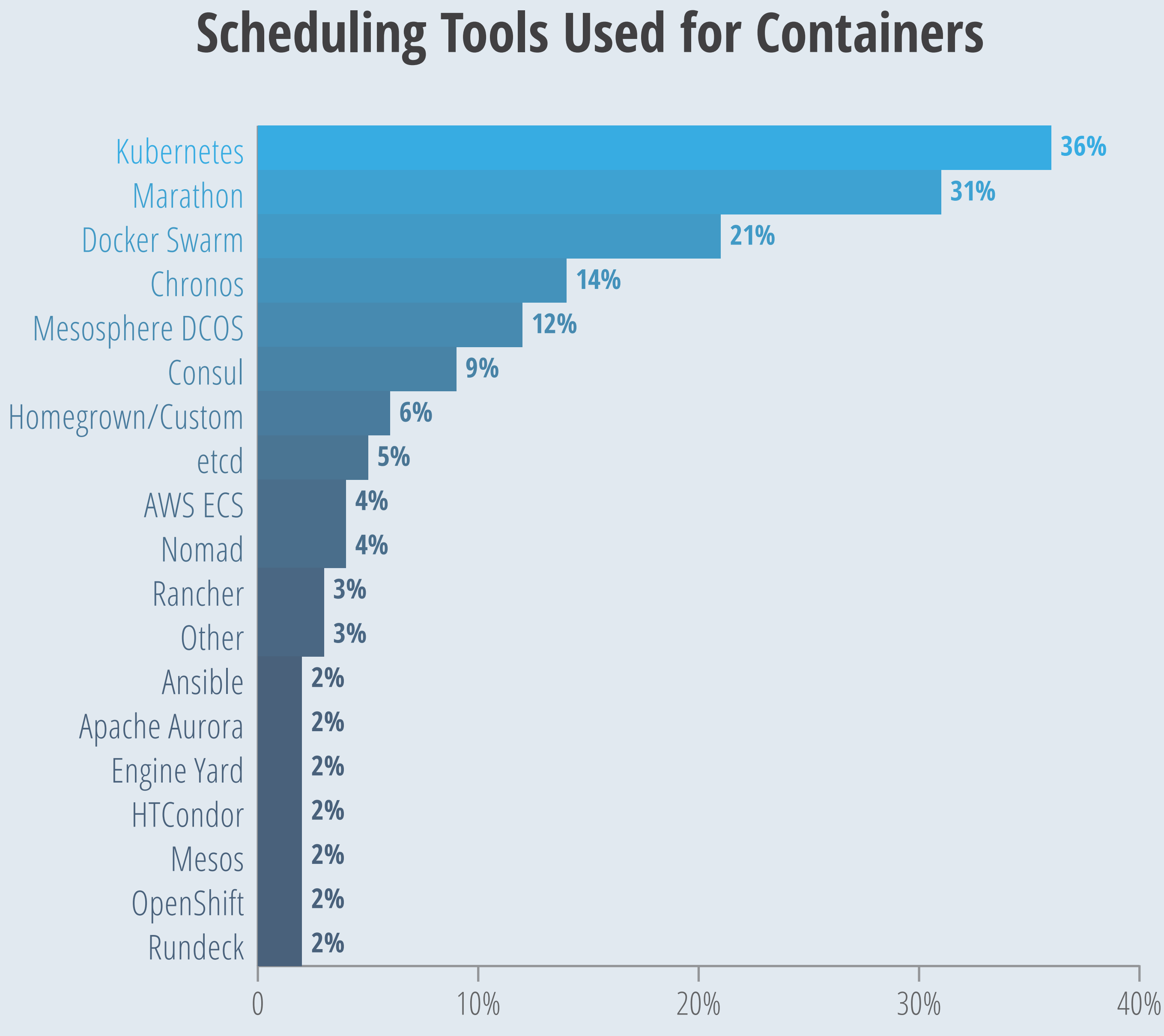
Service Discovery

Consul from HashiCorp, Apache ZooKeeper and etcd (now a CNCF project) were cited most often for service discovery. Since Mesos utilizes ZooKeeper to ensure high availability, it is not surprising that 73 percent of participants solicited through the Mesosphere survey promotion use ZooKeeper. Those that heard about the survey in DevOps Weekly were more likely to use Consul or etcd, perhaps attracted to open source solutions.

Scheduling

Kubernetes, Marathon and Docker Swarm were the tools cited most often for container scheduling. Among the Mesosphere-solicited respondents, 72 percent use Marathon for scheduling and 33 percent use Chronos, while

FIG 14: *Open source Kubernetes, Marathon and Swarm are commonly used to schedule containers.*



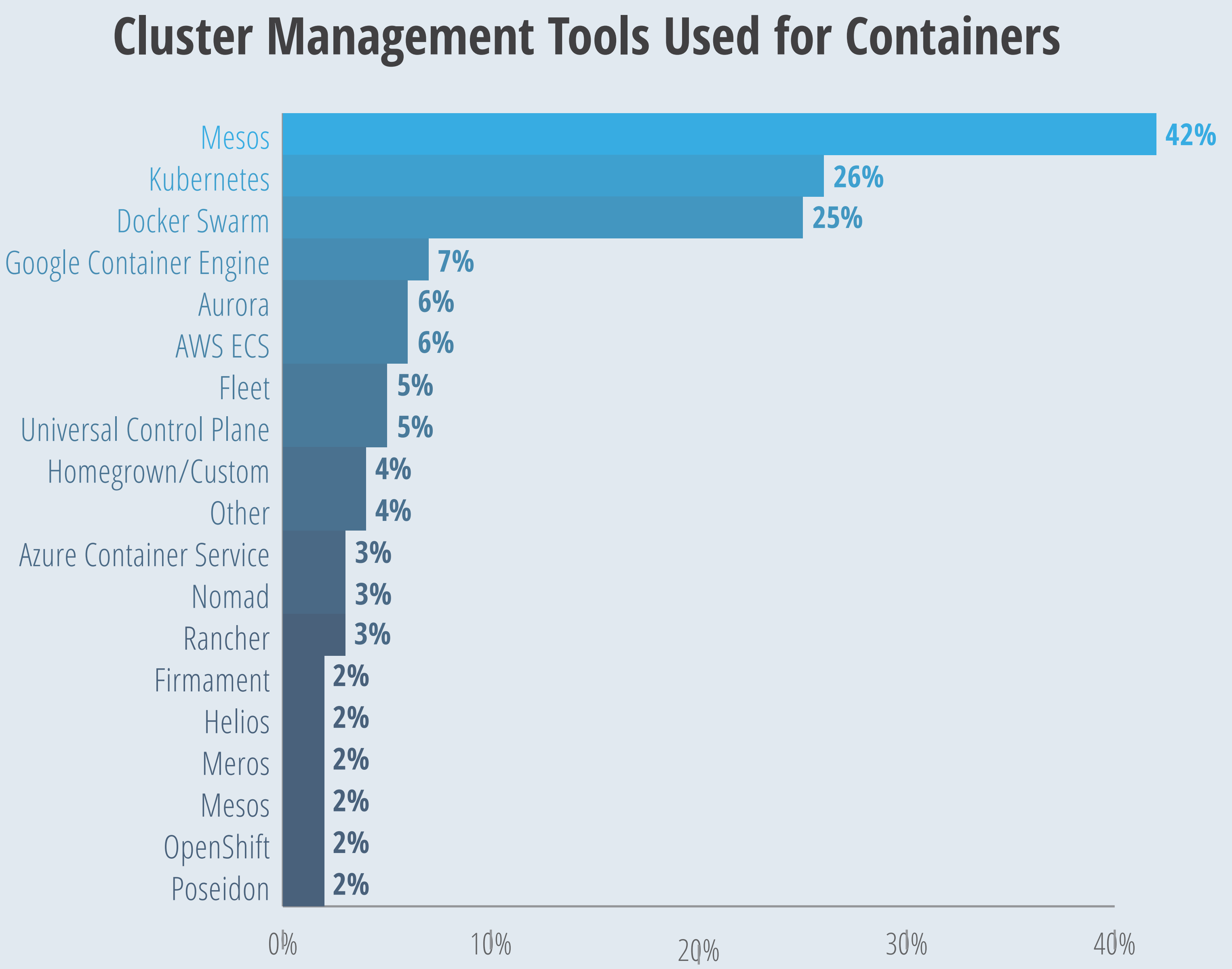
Source: The New Stack Survey, March 2016. What do you use for scheduling for containers? Please select all that apply. n=121. Choices with less than two responses are not shown.

only 6 percent use Kubernetes. If these responses are excluded, Marathon would trade places with Swarm, dropping from the second to the third most used tool. Interestingly, this group was only slightly more likely to use Mesosphere DCOS for scheduling, perhaps because they are not as interested in the enterprise-level services that DCOS recently brought to the market. On the other hand, 43 percent of participating DevOps Weekly subscribers use DCOS for scheduling. It is notable that both AWS ECS, Rancher, and "homegrown" each received at least four write-in votes.

Cluster Management

Mesos, Kubernetes and Docker Swarm were the tools cited most often for cluster management. Every participant that came from Mesosphere said

FIG 15: *Taking into account the survey's slant towards the Mesosphere ecosystem, cluster management is a three-way race between Kubernetes, Swarm and Mesos.*



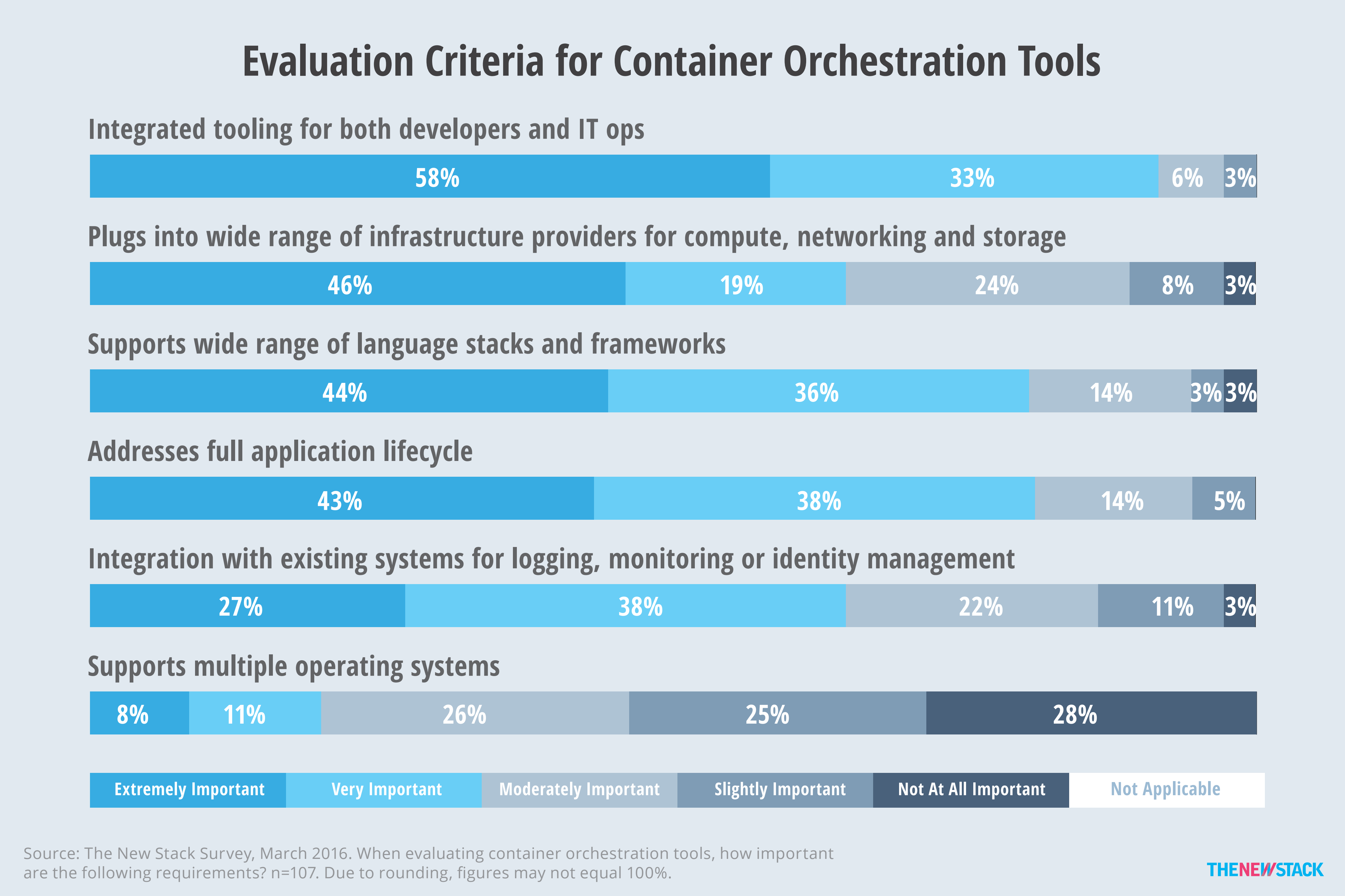
Source: The New Stack Survey, March 2016. What do you use for container cluster management? Please select all that apply. Choices with less than two responses are not shown.

they use Mesos. Yet, even if we exclude these answers, Mesos would still compete with the other two with 22 percent of the remaining sample. DevOps Weekly subscribers were more likely to use Kubernetes (50 percent) and Docker Swarm (43 percent), which may be an indicator of these tools' exposure outside of the early adopter community. Since Swarm moved into general availability in March, it is not surprising that half of Swarm respondents are not using containers in production. Docker's Universal Control Plane (UCP) is also used by five percent of respondents, of which all but one person were using it in conjunction with Swarm to manage clusters of containers.

Evaluation Criteria

In this still nascent field, deciding what product or service to use is not as easy as just looking at cost, performance tests and brand reputation. To find out how container orchestration decisions are being made, we asked about users' evaluation criteria in terms of requirements and abilities.

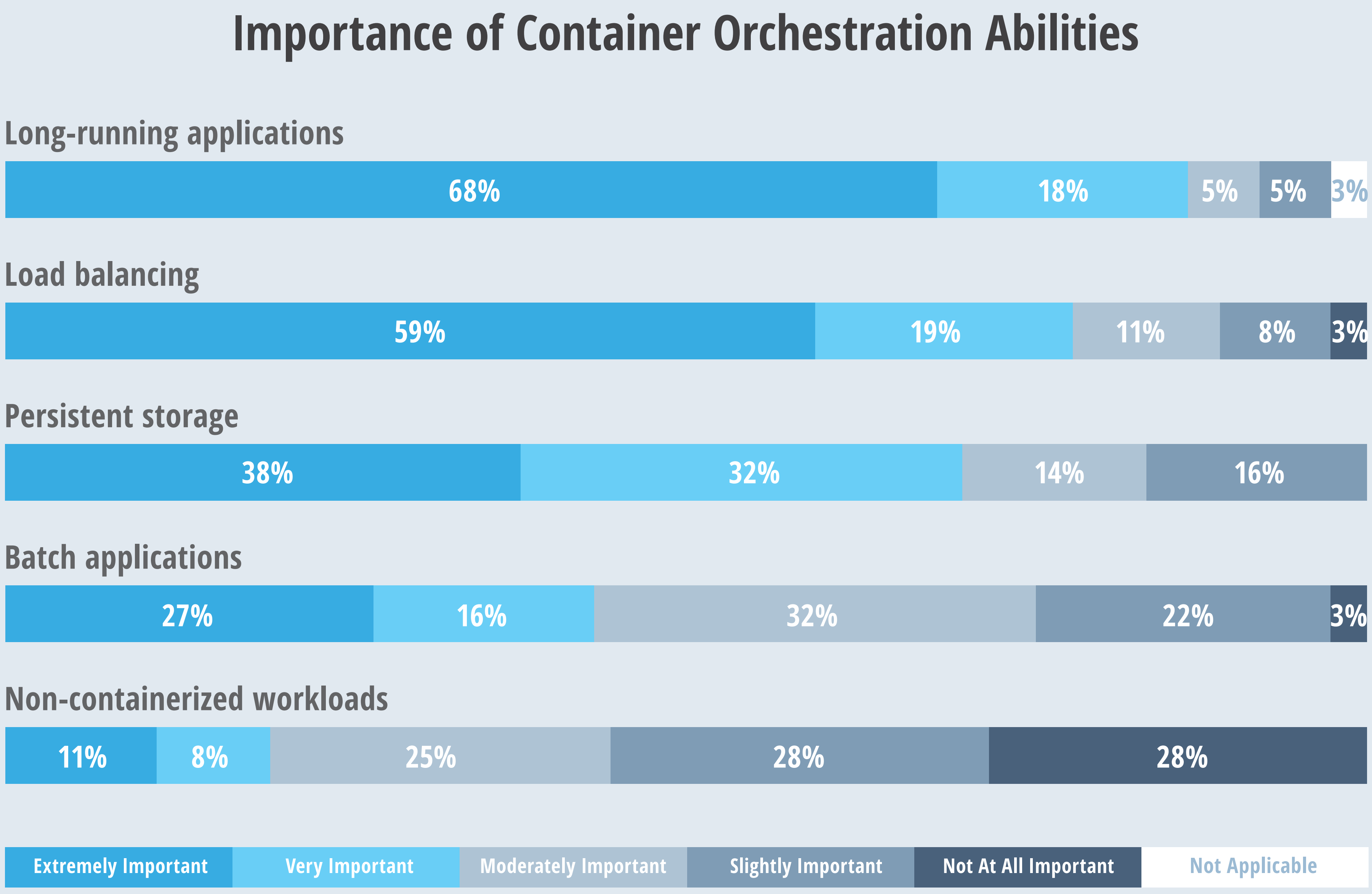
We admired many of the questions Docker asked in its recent user survey of their users, so we adapted one for our own survey. Just like in Docker's results, The New Stack found that integrated tooling (for both developers and IT operations) and addressing the full application lifecycle were among the most likely criteria to be considered important. Also in alignment with the Docker survey, we found that supporting multiple operating systems is the least important. Yet, looking more closely at Figure 16, it becomes apparent that addressing the needs of specific job roles is essential. Fifty-eight percent said integrated tools for both application development and IT operations is extremely important, 12 points higher than any other criteria.



Source: The New Stack Survey, March 2016. When evaluating container orchestration tools, how important are the following requirements? n=107. Due to rounding, figures may not equal 100%.

FIG 16: *Tools that can be used by both developers and IT ops are the most important criteria.*

Usability in many different environments is something that a technology can improve on as it matures. It is harder to address the ability to handle specific use cases. Figure 17 indicates how critical being able to support long-running applications is, with 86 percent saying it is at least very important. In contrast, only 43 percent said the same thing about batch applications. Load balancing was also considered important by more than three quarters of participants. Persistent storage, something that is often cited as a pain point, was as at least very important for 70 percent. However, in comparison to the leading criteria, people were less likely to say it is extremely important. At the bottom of the list is being able to support non-containerized workloads, which indicates that people expect to handle legacy applications separately from those running on containers.



Source: The New Stack Survey, March 2016. When evaluating container orchestration tools, how important are the following requirements? n=107. Due to rounding, figures may not equal 100%.

THENEWSTACK

FIG 17: *Supporting long-running applications and load balancing capabilities are viewed as important.*

Takeaways

Definitions of Container Orchestration

- Scheduling, cluster management and service discovery are widely acknowledged to be part of container orchestration. However, provisioning and monitoring were also considered to be part of orchestration by over half of respondents.

- Docker's Swarm is recognized as the underlying technology among many of those planning to use Docker Cloud and Docker Datacenter.

- While there is still work to be done, many users of Mesos and

Kubernetes understand what products are utilizing the underlying technology.

- Container orchestration platforms remain the most commonly used method of managing containers. However, when looking at specific offerings, users are most likely to say they use Kubernetes which, in and of itself, is not a product.

Containers as a Service and Platform as a Service

- There is agreement that CaaS encapsulates several parts of the software development lifecycle. Container orchestration and registries are the abilities most associated with the term.

- At 25 percent, AWS Elastic Container Service is the fourth most considered option for managing containers in the next year. Google Container Engine and Docker Cloud and Datacenter offerings also are cited often.

- There is still not enough data to determine how much bundling users want in their orchestration tools.

Job Roles

- The intersection between developers and IT management continues to be the container sweet spot. Considering that 49 percent of respondents described their job role as DevOps, it is not surprising that integrated tooling for both developers and IT operations was considered extremely important by 58 percent.

- Application developers are more likely to use custom scripts to manage containers. Perhaps because they do not use them a lot, this group is more likely to think monitoring and configuration management are part of container orchestration tools.

- IT operations are more likely to be using configuration management tools to handle non-production container implementation. There are indications that they will be more likely than other groups to look at CaaS for production use instead of more generic “orchestration platform” offerings.

Configuration Management

- Configuration management was the functionality least likely to be associated with container orchestration, although the big configuration management vendors are still often cited in users’ plans.

- Instead of being a primary method of orchestration, configuration management vendors can expect to be used in conjunction with other tools. Thus, Ansible, Puppet Labs, Chef and SaltStack all get cited in many users’ roadmaps as a second or third choice.

- Ansible rivaled Kubernetes for number of citations in users’ plans. Red Hat’s acquisition of Ansible may have boosted its position as many people cited both OpenShift and Ansible in their plans.

CLOUD-NATIVE CAPABILITIES AND A DEEPER USER EXPERIENCE

In this episode, Ken Owens and Ben Schumacher talk about how orchestrating infrastructure and cloud has changed with the trend towards infrastructure as code, and the emphasis on new formers of schedulers. Many of the changes in this ecosystem, and in the way tools are managed, have directly influenced Mantl, Cisco's curated offering on how to run a container stack. Other topics include: next generation of cloud-native architectures, changing system complexity, the pace of tooling evolution in the container ecosystem, and a discussion about advancements needed with current orchestration platforms.

Listen on SoundCloud or **Listen on YouTube**

Ken Owens *is CTO of Cloud Infrastructure Services (CIS) at Cisco Systems, responsible for creating and communicating technical/scientific vision and strategy for CIS business.*

Ben Schumacher *is an architect within the CTO's Office for Cisco Intercloud Services (CIS), and a lead on the Mantl and Shipped projects.*

CONSIDERATION FOR CONTAINERS IN PRODUCTION

by **VIVEK JUNEJA**

The benefits of adopting a container-oriented development and deployment workflow are not fully realized if the adoption is only retained within the boundaries of development and test environments. The reluctance to run containers in production stems from concerns surrounding security and isolation, and a general lack of operational expertise in managing containers in a production environment.

In organizations that are at some stage of adopting containers, the decision to move them into production environments is a major consideration. It is easier to take the plunge when adopting containers for a completely new service or application, one that is ideally container native.

What does it mean to be container native? A container-native application is one that is designed and built around the lifecycle of containers, and generally considers containers to be first class citizens of its existence. For applications that have been retrofitted to work with containers, the decision to move to production is usually harder. This refers to legacy applications, which are prone to extensive refactoring in order to adopt container-oriented development and deployment.

Further, the idea of running co-located application services as containers on the same machine could invite more confusion, leading to reluctance to go ahead with the production deployment of containers. This idea stems from concerns around providing strong isolation to each co-located container, with respect to performance, security and service levels. This concern is not a new one. Its origin is from the days when virtual machine (VM) technology was being sold to organizations. There is currently increased concern, however, due to the frightening possibilities of running different tenants on the same VM/physical machine as containers.

Understanding the impact on existing workflows and processes in the organization's production environments is a significant aspect of operating containers in production. Here are some workflows and processes around production that may be impacted by containers:

- Moving a change or feature set from development to production.
- Allowing end users to access the change or feature set deployed in production.
- Debugging an issue reported in production.
- Monitoring applications in production.
- Updating application versions in production.
- Taking backup of data generated.
- Disaster recovery and business continuity.

- Capacity planning for production.
- Setting up a host machine, especially for networking and security configuration.

In this chapter, we look at what it takes to deploy and operate containers in the production environment. The goal is to make reasonable arguments in favor of practices that have worked for some adopters who have successfully run containers in production.

Capacity Planning

Operating a production environment without containers is the norm at most organizations in the midst of container adoption. In such organizations, virtual machines may take precedence as the deployment unit, serving the need for isolation and management of an application's components. Most likely, these individual components are distributed across a set of VMs that run across multiple hosts in redundant configurations, which allows for high availability. If the production environment is shared with other applications, then the VM becomes an important tool for this isolation.

The operations team controls the VM management and lifecycle, with application code usually copied over to VMs via some form of automation and workflow. The VMs are rarely destroyed for new deployments, and are instead reused, as revisions of code keep coming in for new deployments. Occasionally, these VMs also go through changes, ideally by reconstructing the VM itself using a new version of the golden image.

Alternatively, some organizations practice reprovisioning new sets of VMs by discarding the old ones when releasing changes to application or infrastructure in production. Netflix's Aminator, and the practices around it, became a trend setter on how VM-based management of production environments can still use the principles of immutability and disposability.

Proponents of containers increasingly advocate for running containerized workloads on bare metal. This helps avoid performance and context switch penalties that arise from using a hypervisor in between the bare metal and the operating system. This argument is more pronounced if we run single tenant systems that are not going to share the resources of other non-trusted tenants.

> “ Proponents of containers increasingly advocate for running containerized workloads on bare metal.

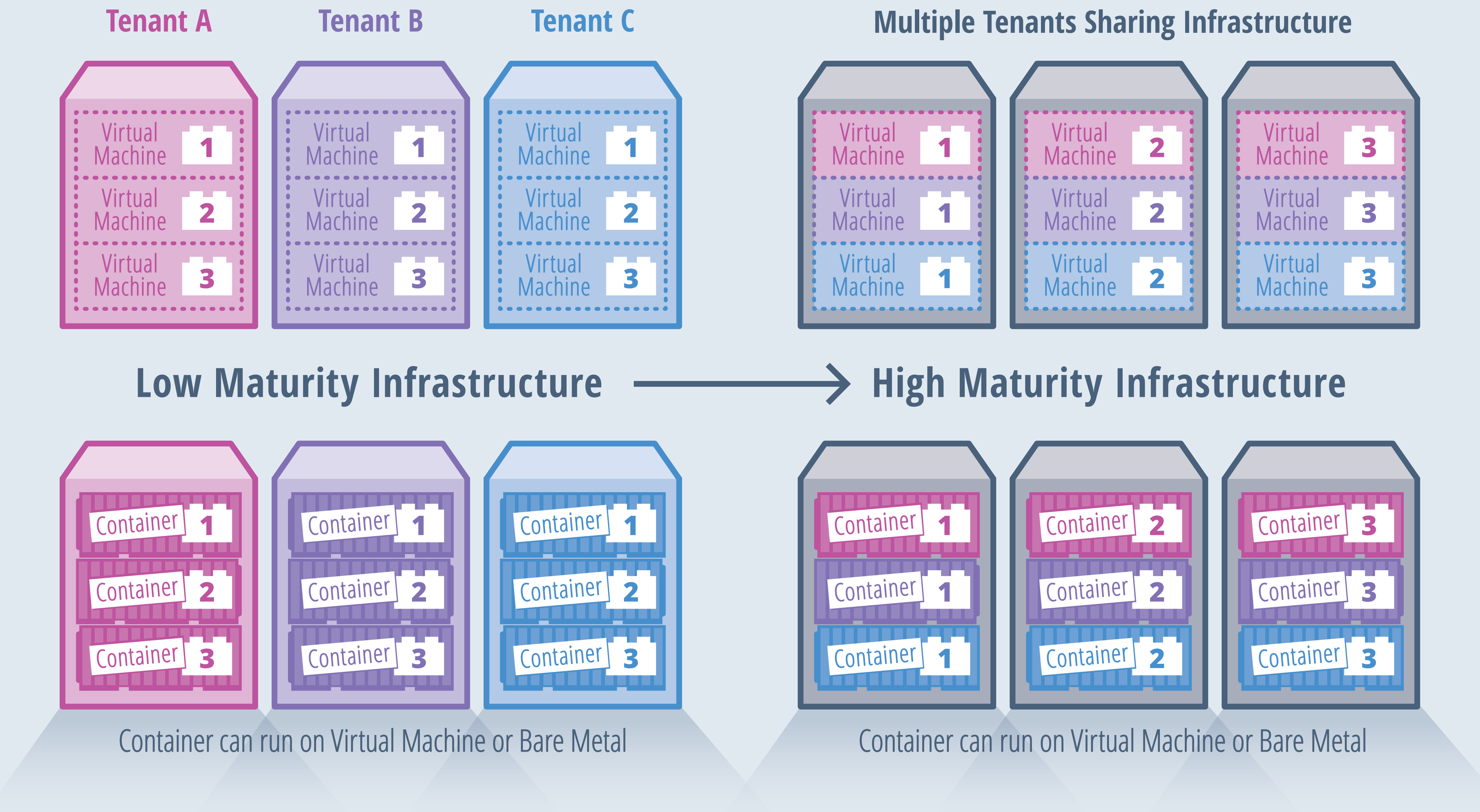
Organizations in the early stages of container adoption can try to use a hybrid strategy to deal with this situation: have a combination of containerized applications running on machines as VMs and some on bare-metal. This combination can be fine-tuned by analyzing the management and performance metric over a period of time. When a certain level of maturity is reached with operating containers, some adopters choose to use only bare metal for running their containers.

When running multiple applications on a shared production infrastructure, the decision to run containers on bare metal should be based on prior experience. A safer option is to have these multiple tenants wrapped around VMs, and let containers be the runtime deployment of your services inside these VMs. Each individual VM may host containers that belong to the same tenant, providing isolation against other tenants.

This concept is demonstrated in Figure 1. In the diagram, at low maturity, the tenants do not share the production infrastructure. This isolation can be a VM or a physical host. At high maturity, components of each tenant application are spread across the shared infrastructure. When considering containers, high maturity represents a shared VM or bare metal infrastructure, where components of each tenant application are spread.

FIG 1: *At low maturity, tenants do not share production infrastructure. This isolation can be a VM or a physical host. At high maturity, components of each tenant application are spread across the shared infrastructure. When considering containers, high maturity represents a shared VM or bare metal infrastructure, where components of each tenant application are spread.*

Transition of Infrastructure from Low to High Maturity



Source: Vivek Juneja

Releasing Applications in Production

The ephemeral nature of containers allows for updates by launching code changes in each new container instance, rather than updating an existing instance. When a particular change is marked to go into production, a new set of containers are created using the new version of the container image, identified by a new tagt. The new tag is ideally generated during the continuous integration (CI) process. The generated image and tag are stored in the container image registry, which is accessible to the production environment.

While it is possible to share the container registry among all environments — including development, test and production — in some cases, a separate container image registry is used exclusively for the production environment. This exclusive container image registry for production is not shared with other environments. In this case, there needs to be a promotion process to take the "candidate" container image from the registry marked for development and test to the registry marked for production. An intermediary system could be used to pull the candidate image from the dev/test registry, and re-tag and push the candidate container image to the registry marked for production. Using this method, there is clear isolation of the images stored and available in the image registries between non-production and production environments.

The transient aspect of containers enforces constraint on the pre-existing practice of using the static port bindings and IP addresses of the host from which the application is deployed. This helps with configuration of the network firewalls and switches. Best practice for releasing changes in production is to adopt rolling upgrades. This requires additional infrastructure in the production environment to modify existing load balancers and proxy configurations, while a new set of container instances are launched on the side of the existing version.

To maintain some level of resilience when deploying containers, it would be wise to leverage the offerings from the container runtime and platform around the orchestration tool. If you're only using the container runtime without any orchestration tool, taking advantage of a runtime configuration like the "--restart" policy is crucial. The recommended restart policy is to toggle between the "on-failure" and "unless-stopped" configurations, or pick one that makes sense for your environment.

Setting Up the Host Environment for Containers in Production

One important aspect in preparing your production infrastructure for running containers is to acknowledge the "sealed" nature of the container image. A container image demands a standardized infrastructure that remains similar in configuration across development, test and production environments. This standardization is crucial to getting a predictable experience when running containers in production. Here are some factors to consider:

- Choice of kernel.
- Container runtime version and choice.
- Network access and firewall configuration.
- Choice of security hardening solution.
- System access permissions.

It may be tempting to sway away from standardizing all of these factors across environments; however, any deviance can lead to unpredictable results that may require investigation. The simplest way to address this is by ensuring that each environment, from development to test to production, is built with the same topology. It may be difficult to achieve at the beginning, but consider it the objective for container adoption. All environments use the same Linux kernel, they use the same host configuration, file system topology, network configuration and user roles. Standardizing all factors helps ensure that the container image which gets built and tested in integration remains the same right through to production.

Certain aspects of the runtime behavior of the container instance will be different, like URLs for accessing other dependent services, or log-levels. These runtime configurations are passed to the container using various options. One such option is to inject the runtime configurations via environment variables to the container instance. The environment variable could point to a service registry, which then points to the right dependent service. Tools like etcd, ZooKeeper and Consul are helpful in implementing this paradigm.

If the host is being provisioned using existing configuration management and provisioning tools, like Chef, Puppet and Ansible, then it would be right to have a singular configuration for all environments, as much as possible. For a scale-out infrastructure, the limited differences remaining across environments is the number of instances that would be required to be provisioned, considering availability and performance requirements.

The host will also have to be occasionally updated with the latest kernel patches, container runtime version, etc., that cannot be sealed inside the container image. This change of host configuration could follow the rolling upgrade, which allows updates in stages, instead of taking a big bang approach.

Finally, like the host, there needs to be similar configuration and topology for supporting infrastructures, such as log aggregation, monitoring, metrics, service routing and discovery for all environments. This allows a level of consistency for the container images, not just with the host, but with all participating services in the respective environment.

Service Discovery of Containers in Production

Standardizing the service discovery mechanism is an important consideration for containerized applications. It would be rare to run containerized applications the same way you would run application on VMs: there would be more than one container running on the same host. In this rare case, if you're using a bridged network mode, you may require a static port assignment for each container. This means that you pre-select the port exposed by the container on the host, and use that to configure the load balancer and proxy.

Reconfiguration of the load balancer and proxy may not be needed if there is only a limited need for adding more containers to the host. In most cases, however, containers are added or removed based on the load through the use of immutable deployments or scaling practices. In this case, it would be difficult to keep a static set of pre-assigned port numbers, and that is precisely where service discovery systems excel.

Production Support: Log Aggregation and Monitoring

The supporting services needed for containers in production remain the same as with non-containerized environments. This includes a way to capture logs from container instances and ship them to a centralized log management system. Built-in log backend support already exists in the Docker daemon, and custom solutions are in place to take care of this.

There are a number of options for managing log data produced by Docker containers. Docker's default settings write logs uncompressed to disk with optional retention settings to limit storage usage. In a production environment, there are a number of logging drivers that ship with the Docker engine and provide flexible options for managing log data transparently to the applications producing it. If you already have a centralized logging solution in place, there is likely a Docker logging driver which can be configured to feed container log data to it. There are logging drivers for various established protocols such as syslog which can be used against a number of ingestion systems as well as cloud or SaaS-specific options.

The ephemeral state of containers is a key behavior to note in container logging and monitoring. New containers replace old ones when deployment happens in production. This is a break from the traditional convention of assuming stateful and long-running compute units. Rotating the container fleet creates new problems for the traditional logging and monitoring mindset. Rather than having compute instances lying around for days to months, containers could be rotated within a window of an hour.

> “The ephemeral state of containers is a key behavior to note in container logging and monitoring.

If you practice continuous delivery (CD), this window could be even shorter. A host-centric logging and monitoring solution cannot scale to the complexity of containers. Rather, due to their short life span, have a declarative way of monitoring them. And it’s better to monitor them as a group, instead of monitoring each container in the fleet.

One way to associate containers within a group is by using the appropriate metadata, such as tags. A tag refers to the “image-tag” that would be running in the production environment. It is recommended to avoid using the misunderstood “latest” tag for Docker containers when implementing this.

For example, suppose you deploy the image of your application “product-api” with the image tag “25” and the environment variable set to “production.” This means this is the 25th image identified by the build system, and the container running is in the production environment. You could have multiple instances of this running across your container infrastructure at any given point in time. The tag will be changed with each new deployment, but the environment variable configuration will remain set to “production.”

Having a monitoring system watch out for container images that are currently running with the environment variable set to “production” creates the impression of a long running production service, insulated from the continuous changes that are being made as a new version is deployed. If you are using orchestration tools, then you have access to a richer vocabulary of tags for use in grouping container instances.

The appropriate monitoring and logging strategy for containers in production is a non-intrusive solution that can stay out of the running container, ideally running as a “side-car” container service. Container monitoring tools need to be container aware, and even container-platform aware. This awareness will help monitoring tools avoid confusion when reporting a failure, such as when a container is stopped on one host and moved over to another by the container platform. The container footprint will lead to an excess of data gathered by the monitoring and logging systems, thereby causing a need for additional data management.

Container monitoring is a space that is growing by leaps and bounds, and we are covering it in our fifth ebook. There are various Software as a Service (SaaS) and on-premises tools that could help you make a decision around this in the meantime.

Approaches for Managing Container Data

The generally accepted advice for managing container data is to have stateless containers running in the production environment that store no data on their own and are purely transactional. Stateless containers store processed data on the outside, beyond the realm of their container space, preferably to a dedicated storage service that is backed by a reliable and available persistence backend. This security concern is even more pronounced with container instances that host storage services, like databases, queues and caches. For these stateful containers, the agreed-upon pattern is to use data containers. The runtime engines of these stateful services get linked at runtime with the data containers. In practical terms, this would mean having a database engine that would run on a container, but using a “data container” that is mounted as a volume to store the state.

If you are running a clustered hosting environment using an orchestration platform, it is important to have a distributed storage solution, like Gluster and Ceph, to provide shared mount points. This is useful if the container instances move around the cluster based on availability.

Container Security and Key Management

Security is often a major concern, especially when running multiple container instances, each for a different tenant on a shared machine. This concern stems from a lack of trust and confidence that container technology will provide the right kind of isolation, as is expected from a VM-based implementation. However, treating containers as a replacement for VMs will not help with this concern. Container implementations, like Docker, provide a security blanket for applications. The container runtime abstracts away the complication of configuring fine-grained permissions on different namespaces, such as user, network and process.

If considering a multitenant deployment of containers on a shared infrastructure, leverage VMs for each tenant separation, and use containers as the isolation medium between a tenant’s application components. As proliferation of container deployments increase across users in the community, it will become necessary to avoid the VM wall and have all tenants share the same infrastructure.

The other aspect of running an immutable container instance is to avoid baking in any keys or credentials that need to be kept secret. There are multiple ways to solve this problem, from passing the secrets via environment variables, to protecting them with encrypted data containers that are mounted as volumes. However, there has been criticism in using these techniques, and there is no available standard from container runtime providers.

The Road Ahead with Containers in Production

An important element in making container adoption successful in production is by having an informed and educated community. Tools and practices will evolve to a state where running containers becomes the norm for most organizations. Until then, the unfinished battle to make this adoption happen lies within the organization. Making every stakeholder, especially operations and security teams, deeply understand their requirements around containers is a task that requires a lot of work. The path to container adoption in production will be different from that of VMs, and will prioritize developer experience and operational simplification in addition to the benefits of resource utilization.

CREATING A FINE-GRAINED USER EXPERIENCE FOR CONTAINER MANAGEMENT

In this podcast with Ken Robertson of Apcera, learn about some of the obstacles associated with cluster management, container management, scheduling, and more. Robertson talks about Apcera's approach to handling multi-user systems, and the high value placed on the user experience of managing workloads. Also discussed: how to manage security and clustering with multi-user services, and the issue of trust when it comes to giving high-level access to hosts and privileges. Many of these issues are related to the need for fine-grained control within clusters. Ken talks to The New Stack about how Apcera can help address those needs.

Listen on SoundCloud or **Listen on YouTube**

***Ken Robertson**, lead architect at Apcera, builds, deploys, manages, and monitors applications and infrastructure at scale. He enjoys blurring the lines between architect, developer, and operations. An active open source contributor, Ken authored Kurma, an advanced implementations of the App Container specification (appc) in 2015, and is an active maintainer on the project. Ken also guides Apcera's work with the OpenContainer Initiative (OCI). His previous development experiences include Telligent (now Zimbra), Involver (acquired by Oracle), and Demandbase.*

BAKERY AS A FOUNDATION: CONTAINER IMAGES AND MICROSERVICES

by **VIVEK JUNEJA**

One of the fundamental ingredients in the adoption of microservices and containers is the notion of immutable changes being packaged in the form of container images. This container image format leverages the transport system of continuous delivery (CD), and allows a progressively better model than virtual machine (VM) images or Amazon Machine Images (AMI), in AWS parlance. The container image format is the new executable or build asset that is built once, and then can run on any compatible microservices infrastructure.

The adoption of machine images as the transportable entity is often attributed to the rise of automation strategies that allowed development teams to produce the image as the deployment artifact. This began with the AWS AMI format, and since then a variety of tools have appeared in the ecosystem that eased the life of developers by promising a quick, reliable and easy way to generate redistributable machine images. This also opened up a debate around what's the best way to distribute the application on infrastructure: whether to conceal all aspects of a machine and code into a factory made image that remains immutable, or to allow a certain degree of self-awareness and self-configuration.

One of the approaches to build the machine image is to bundle it with the application code as the output of a build process. The image is configurable to the extent that application-specific parameters can be passed to it, when a container instance is instantiated. These parameters could be used for service discovery and other application-specific toggles. This approach dictates rebuilding and redeploying the said image whenever any change is required, either to the application or to the application topology.

The other approach of building machine images insists on creating a minimal machine image that is built only when there is a change in infrastructure, runtime or topology. For most code changes, new machine images are not rebuilt. New container instances are spawned from pre-existing machine images, and code is then deployed over it, either through boot scripts or via agents running on the machine instance. This approach prevents the need for continuous generation of new machine images, and instead lets the newly created container instance discover and fetch the changes itself. This mandates autonomous behavior on the part of the container instances.

> “The difference between approaches, however, lies in the scope of what goes inside the build process of machine images.

It is critical to note that in both approaches, the common aspect is to keep creating new instances of infrastructure when propagating changes, instead of relying on a long running infrastructure footprint. The difference between approaches, however, lies in the scope of what goes inside the build process of machine images.

Netflix documented its internal process of creating baked machine images, and open sourced the Aminator tool a while back. Tools, like Packer from HashiCorp, have become commonplace in the modern delivery pipeline. Major PaaS vendors, like IBM Containers, have entire toolsets for managing and creating images in containerized environments. Ventures, like CloudNative.io, which offer organizations a complete AMI lifecycle management tool called Bakery, are pitching this capability similar to the practice at Netflix.

Machine images have new meaning beyond their usual reference to VM images like virtual desktop infrastructure (VDI) or AMI. Container image is the new addition to the meaning of machine image. This addition is drastically faster, more lightweight, and is making more headway into the workflows of ordinary developers. The container image is more pronounced with developers compared to the predecessors, such as machine images, that were primarily part of the machine-based operations toolkit.

Defining the Model of a Bakery

A bakery is a form of infrastructure entity that embodies the process of acquiring, building and releasing machine images to allow repeatable deployments of working code. The output of a bakery is a baked image that is used to spin off instances of machines (VMs or containers) in any compatible infrastructure. The compatible infrastructure represents the required environment in the form of hypervisors or container engines that support the deployment of these baked images.

A form of bakery excels when its output of baked images is compatible to a varied type of infrastructure, reducing the need for bespoke images built for each type. The rise of containers represents the form of standardization that would make container-oriented bakery models more meaningful and widespread.

In any microservices and containers story, the production and maintenance of container images holds an important place. Hence, successful microservices and container adoption is not without elegant handling of the various facets of container images, including:

- Selection of base images.
- Image sharing and isolation across multitenants/multiple projects.
- Container image validation and verification.
- Container image workflow.
- Testing container images.
- Container image storage and lifecycle.
- Container image expiration and depreciation.

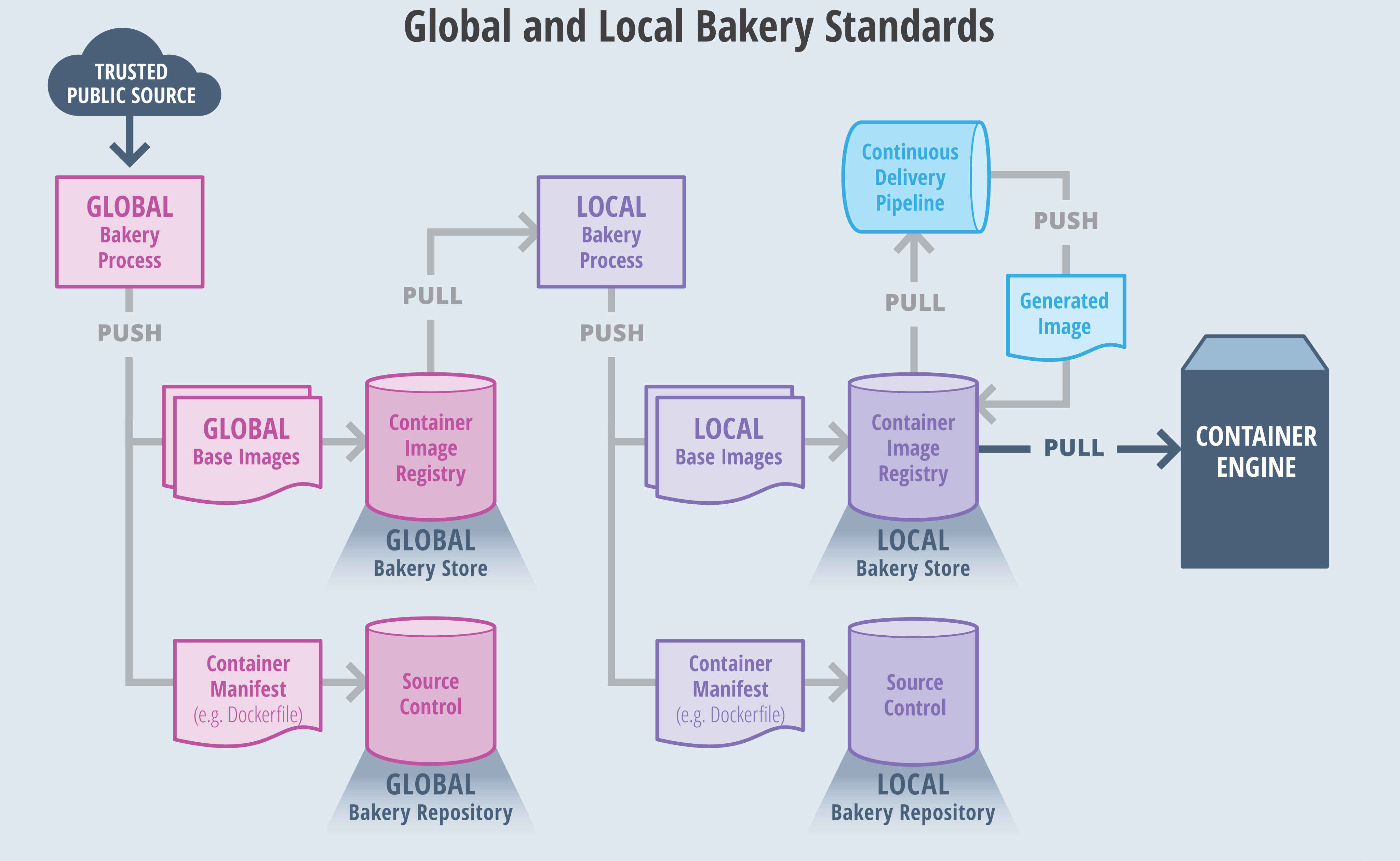
All this is integral to the bakery strategy and implementation in an organization. In this article, we will go over these facets in detail, and demonstrate how automating some of these can simplify the story of microservices and container adoption.

Creating Global and Local Bakery Standards

The story of the bakery model begins with the selection of the appropriate base images that are inducted in the development process. Most likely, the selection of these base images will be based on the size of the image and the trust for distribution. Alpine Linux, and images originating from that, continues to be a predominant choice for base images. It is important to note that each base image selected must be accompanied by a documented history to ensure that the origin of the image can be validated.

In an organization with multiple projects and varied teams, it could be detrimental to have each team select their own base image. Building an

FIG 1: *Diagram illustrating workflow across various systems involved in the Global Bakery and Local Bakery models, and demonstrating how both the continuous delivery process and the container engine pull from the workflow.*

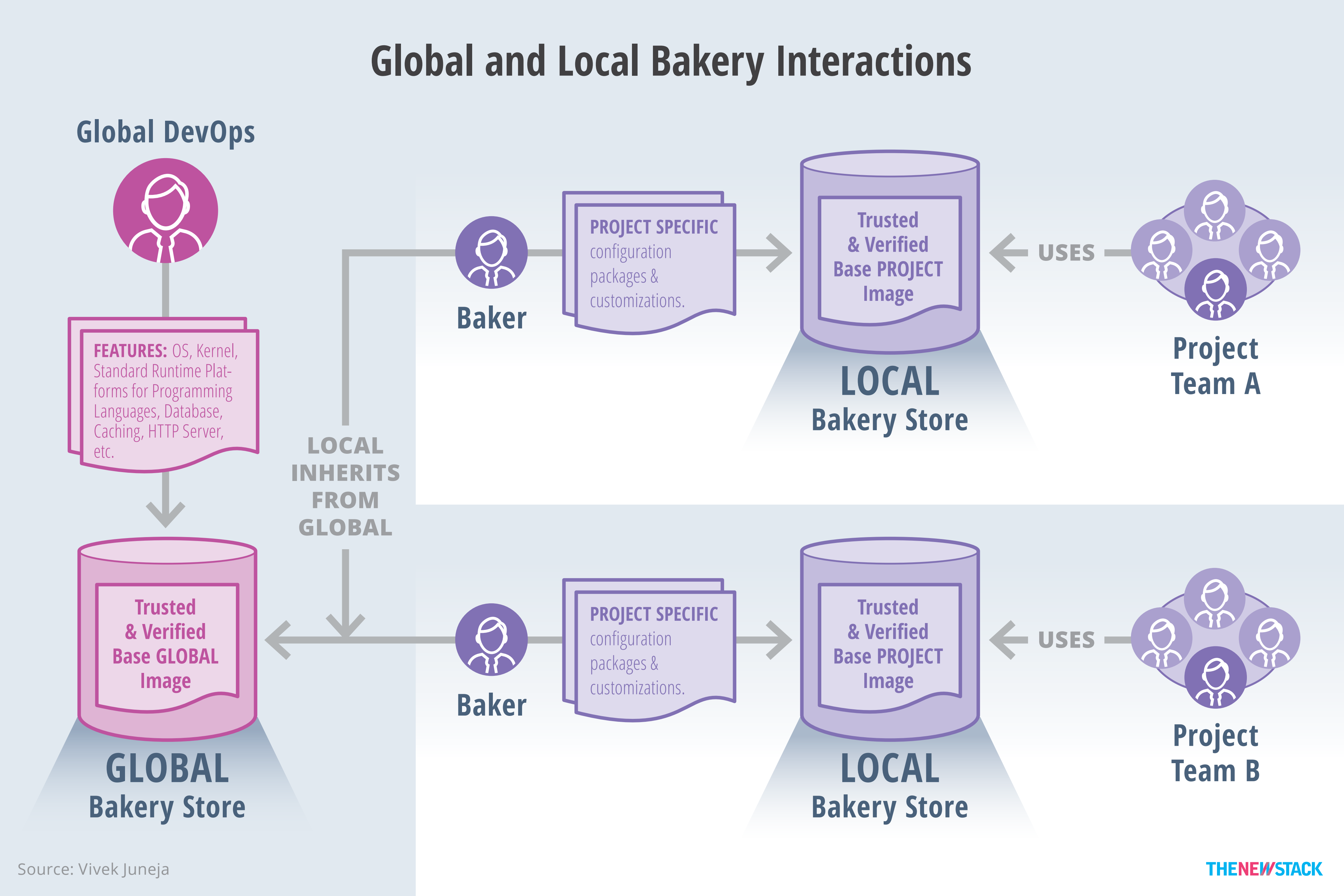


Source: Vivek Juneja

array of trusted base images that could be shared and standardized is an approach that could work for organizations at scale. I call this concept a Global Bakery — a logical entity in an organization that focuses on standardization, governance and the lifecycle of base container images.

These shared base images could then be further customized to each team's requirements and inducted into the development workflow. From the team's point of view, the focus is less about inter-team sharing and more about making the container image useful as the deployable base. I call this concept the Local Bakery — an entity that inherits the base images from the Global Bakery counterpart, and is required to work at the scale of an individual team.

FIG 2: *Diagram illustrating scope of features that go into a base image during the bakery model process, including the ability to facilitate multiple Local Bakery teams.*


Global and Local Bakery Interactions
Global DevOps
FEATURES: OS, Kernel, Standard Runtime Platforms for Programming Languages, Database, Caching, HTTP Server, etc.
Trusted & Verified Base GLOBAL Image
GLOBAL Bakery Store
LOCAL INHERITS FROM GLOBAL
Baker
PROJECT SPECIFIC configuration packages & customizations.
Trusted & Verified Base PROJECT Image
LOCAL Bakery Store
USES
Project Team A
Baker
PROJECT SPECIFIC configuration packages & customizations.
Trusted & Verified Base PROJECT Image
LOCAL Bakery Store
USES
Project Team B
THENEWSTACK

Source: Vivek Juneja

The output of the Global Bakery is a standard pool of base images, each of which has been verified, to add to the base image registry. The images are then ready to be used by all teams in a service-oriented organization.

The output of the Local Bakery is the base image that is customizable to the individual service stack and deployable for the service. Both Global and Local Bakeries use continuous integration (CI) to build and test the container images before shipping (or deploying) them to their registry. A simple tagging strategy is useful to ensure container images are easily searchable and identifiable. Usually, the build ID generated by the CI server is used as the basis to arrive at the tag. Both Global and Local Bakery entities could use this premise to avoid confusion of the "latest" tag.

Discovering and Implementing Bakery Model Practices

Testing and Security with Images

Testing base images is crucial to the success of container adoption. Base images will get propagated across the entire infrastructure and will be used for further customization and extension by developers. Behavior-driven development (BDD) can be employed to test automated infrastructure. By using a service like Serverspec, you can create RSpec tests to check if a given server is configured correctly. Serverspec scripts can be run against the container instances spawned from base images during the CI process. This form of script allows verification against a prescribed set of configured states that a container has when it's run.

Penetration Testing and Security Assessment

The base images and the workflow surrounding them provide a good foundation to conduct an automated security assessment. Tools, such as

Twistlock and Docker's Project Nautilus, can integrate with the CI and deployment pipelines for baked images and perform continuous scanning to ensure the images are secured. If a vulnerability is found and fixed, a new base image can be generated from the Global Bakery, and then passed on to the various Local Bakeries. Each Local Bakery then leverages continuous delivery (CD) pipelines to release new containers based on the changed base image. This allows rapid remediation to be in effect as soon as a fix is introduced in the base images.

Avoiding the Need to Bake in Secrets

One commonly recurring practice for production deployment of containers is the storage of secrets, like credentials and auth keys, needed by the service running inside the container instance. A bakery can instill a common practice to manage the secrets needed. As part of the governance process, a Global Bakery can integrate key management solutions, like HashiCorp's Vault or Square's Keywhiz, into the base images, thereby providing a common foundation for all container images that are created further down the food chain. This will also remove the need for reinventing the wheel for each service team that runs its own Local Bakery; it allows correct practice of storing and accessing secrets to become institutionalized.

Configuration Management in the Bakery Model

Prior to the rise of mass adoption of containers, configuration management (CM) tools like Ansible, Chef and Puppet helped with converging a given set of machines to a desired configuration. However, with the advent of container images and their extremely small footprints, these assumptions provide little value to the cause of CM tools. Since container images need to be lightweight, a heavier CM approach sounds like an anti-pattern.

CM tools do, however, merit an ordered way of orchestrating installation. In some cases, images are being built with CM tools and are provisioned via a Chef-solo kind of mode, without the need for any external CM servers. The way these two paradigms will converge or diverge depends on the strategy that container engines take in the coming years with regards to configuration management.

Global Bakery Through Automation

As mentioned earlier, the Global Bakery is a logical entity, and is usually used to generate shared base container images that can be adopted by each service team in an organization. The Global Bakery can be implemented as a set of automated tooling that exists between the Local Bakery and the public container images hosted on Docker Hub, Quay.io, IBM Containers, or other registry services. An automated Global Bakery must perform the following in an organization setting:

- Validate base images from the origin to check if a valid history exists.
- Compare alternatives for the base image based on the size of the image and community strength (this refers to the number of people using the image from Docker pulls).
- Perform a security assessment and vulnerability scan.
- Create a new resultant container manifest (Dockerfile) that will install management tooling for debuggability, monitoring, log stream and secret store, and so on.
- Add metadata to the resultant base image in the container manifest.

- Push the resultant container manifest to the versioning system and the generated image to the registry, the Global Bakery store. This is used further by individual teams, each running their own Local Bakery.

Retiring Old Base Images

A base image goes through multiple versions over a lifetime, and eventually is discarded from usage. Release of new versions of a particular global base image can be communicated to individual teams, so that changes could be made to the local images. Alternatively, the CI infrastructure for each service team may check for new versions of the base image on each iteration, and can trigger a warning if an older version is still operational.

The Way Ahead for the Bakery Model

The bakery model is potentially a very important entity in the infrastructure for microservices. There are new initiatives, like improved container workflow around unikernels and changes to Docker image IDs, that promise to make the bakery model even more useful and relevant. If effectively implemented, a bakery can allow service teams to become more independent and deliver on the promise of microservices.

cloudsoft

GAINING VISIBILITY INTO MANAGING CONTAINERIZED APPLICATIONS

Alex Heneveld of Cloudsoft talks about what it really means to automate and manage an application, and how it necessarily builds on visibility into the application and the ability to orchestrate related services. He discusses the origin of Apache Brooklyn and the basis for ideas about autonomic computing for managing complex systems. He sees containers as similar to past movements involved in multi-threading, parallel execution, and chain processing—while containers aren't exactly the same, the idea of easy encapsulation and speed is there. Heneveld goes on to cover a great deal of topics including the idea of "death as architecture," the role of messengers in the container stack, modeling policies, and container networking capabilities. **Listen on SoundCloud** or **Listen on YouTube**

Alex Heneveld *is the co-founder and chief technical officer of Cloudsoft, and has 20 years experience designing software solutions in the enterprise, startup, and academic sectors. Alex was with Enigmatec, where he led the development of what is now the Monterey Middleware Platform. Before that, he founded PocketWatch Systems, commercializing results from his research. Alex holds a PhD (Informatics) and an MSc (Cognitive Science) from the University of Edinburgh.*

CONFIGURATION MANAGEMENT AND ORCHESTRATION

by **SCOTT M. FULTON III**

Orchestration is what distinguishes the modern way of providing digital services from how it was done even five years ago. What's so new about this stack? The answer is the orchestration of workloads. We can now think of the functions our servers perform as workloads, rather than applications with brands, or virtual machines with golden masters and overwrite protection.

Sir Isaac Newton redefined how we think about the physics of the universe by perceiving the motion of objects as units of work that consume energy and produce force. Orchestration assumes a Newtonian redefinition of computing. And like the words of Sir Isaac himself, the paradigm has been difficult for some.

Key Forces that Enabled Organization

The phenomenon of orchestration came about through the confluence of several evolutionary industry trends:

- **Virtualization** unshackles programs from the hardware of the servers that ran them.
- **Web services** enable communication between program functions, without the necessity of middleware.
- **Data stores** free up huge pools of data from having to be processed by using an itinerary specific to any one application, or any number of them.
- **Cloud dynamics** makes it possible for the resources required by a program to be provisioned on-demand, at precise increments.
- **Containerization** shrink-wraps programs, and the few dependencies they require, into tight bundles that can be hosted by their own operating environment.
- **Software-defined networking** makes the environment in which workloads are provisioned and deployed a plastic, pliable, fault-tolerant mesh that's adaptable to the demands of the jobs at hand.

These six forces brought about the new reality of truly distributed computing. What's shocking is that they didn't really conspire to do so intentionally. All six of these information sciences were created to serve their own unique purposes. But once they were brought together in the data center, they produced a completely new opportunity to build and manage systems exclusively and entirely around the work that can be done.

All this being said, orchestrated workloads must coexist with traditional applications, and even legacy software, for reasons that may be substantive or trivial, but are yet undeniable. Organizations seeking to implement workload orchestration within their data centers must develop a strategy for the old and new systems to work together with reasonable efficiency, no loss in productivity, and no degradation in security.

The Mindsets of Orchestration Adopters

Configuration management (CM) is part of the old stack — a tool from the 1970s and the era of version control. Back when changes in software were perceived as hindrances, software configuration management (SCM) devised checkpoints that kept the forces of change in check.

Yet, CM today may play a pivotal role in the establishment of an environment of coexistence, both for servers and the people who manage them.

In traditional virtualization environments, CM has become invaluable. The basic job of CM is to document the resources required by applications hosted on virtual machines, and to apply those documents in determining the resources required to run many applications for multiple tenants simultaneously.

> “In traditional virtualization environments, configuration management has become invaluable.

The job of CM is typically delegated to the IT department. It was when the manifests used by CM tools first evolved into scripts, with terms and functions akin to batch processing languages, that someone came up with the notion that systems operators and software developers were merging into one job function. DevOps is a by-product of the modern evolution of CM.

In the 2000s, virtualization made it necessary for brand-name, commercial CM vendors to move the focus of their tools away from itinerary-based requirements assessment, and toward change management. The ability to rapidly provision virtual machines led engineers to develop open source alternatives that evolved at a brisker pace. Chef and Puppet helped IT and the first DevOps teams set forth a strong set of guidelines and principles for how workloads are deployed in an enterprise.

The continuous integration (CI) and continuous deployment (CD) movement arose from the discovery of efficiencies and productivity gains associated with CM tools such as Puppet and Chef. IT departments became tasked with automating version control in smaller and smaller increments. They began conducting more testing (and assuming some parts of the job of debugging), with application performance management (APM) tools playing a role in measuring the effects of each increment on the data center. The CI/CD movement trickled over to software developers, thanks to the popularity of open source tools like Jenkins, TravisCI or their commercial equivalents.

What Docker Changed and What It Hasn't

Then Docker happened. Docker's native tools are designed to be used by developers, and have helped bring containerization practices to different teams and IT roles. At least one recent survey from Evans Data showed that as many as nine out of ten organizations deploy containerized environments within separate virtualization envelopes, mainly for use by developers, and maintained by IT using traditional CM. This shows how the social order of information workers within enterprises is resistant to change, even in the face of technological metamorphosis.

As a result, there is no preferred methodology for orchestration across organizations. Arguably, it may be best that we not pretend there is one. While the six forces I listed earlier clearly gave rise to containerization, the needs of users and businesses are still changing and will likely shake things up in the future.

> “There is no preferred methodology for orchestration across organizations.

Perhaps the greatest unresolved argument in containerization today is whether a container should be a temporary, transient vehicle for functionality, or rather a persistent encapsulation of functionality — an evolved form of the virtual machine with a master image and a stable state. It’s an important argument with validity on both sides. Many database applications require some functions to be persistent in order to preserve database integrity. Yet microservices require containers to be transient or ephemeral in order to prevent workflows from evolving into meshes of contorted spaghetti code. Another issue is the role of people in the workflow. Enterprises that have embraced the ideal of CI/CD, as well as its technologies, are accustomed to perceiving applications as immutable images. Immutability, in their mindset, is attributable to the software rather than the infrastructure. The choices enterprises make with regard to their orchestration tools speaks to the way their people work with one another, as much as to their interactions with their software.

Five Styles of Workload Orchestration

Configuration management, CI/CD and on-demand cloud service provisioning play roles in modern enterprises to varying degrees. As a result, organizations may adopt different orchestration strategies, and may even employ multiple strategies for different roles and separate divisions.

1. The Docker-Based System

Docker's native orchestration system, from the vantage point of usability, is not complex at all. The system is command-line based, and Linux command lines are familiar to both developers and administrators. But its underlying assumption is that it's being used by developers in the course of staging their own work.

The part of Docker that runs the containers themselves is called a daemon. The component that includes this daemon is called Docker Engine. Its purpose is to run containers within the environment currently set up for it. By default, that's the typical memory address space of one machine (physical or virtual).

The distribution part of the equation enters in with Docker Swarm. It collects multiple address spaces together into a central cluster, performing an act of "reverse virtualization" that abstracts the complexity of that cluster from Docker Engine. As a result, the engine perceives the space created by Swarm as just a very large pool, or a huge computer.

A third component, Docker Compose, introduces one or two elements from CM. A text file, called the Dockerfile, uses YAML code to describe the requirements of the application hosted in the container. This file is spun "up" to the Docker Engine, which executes commands based on the Dockerfile's contents, to produce a fully-active running environment.

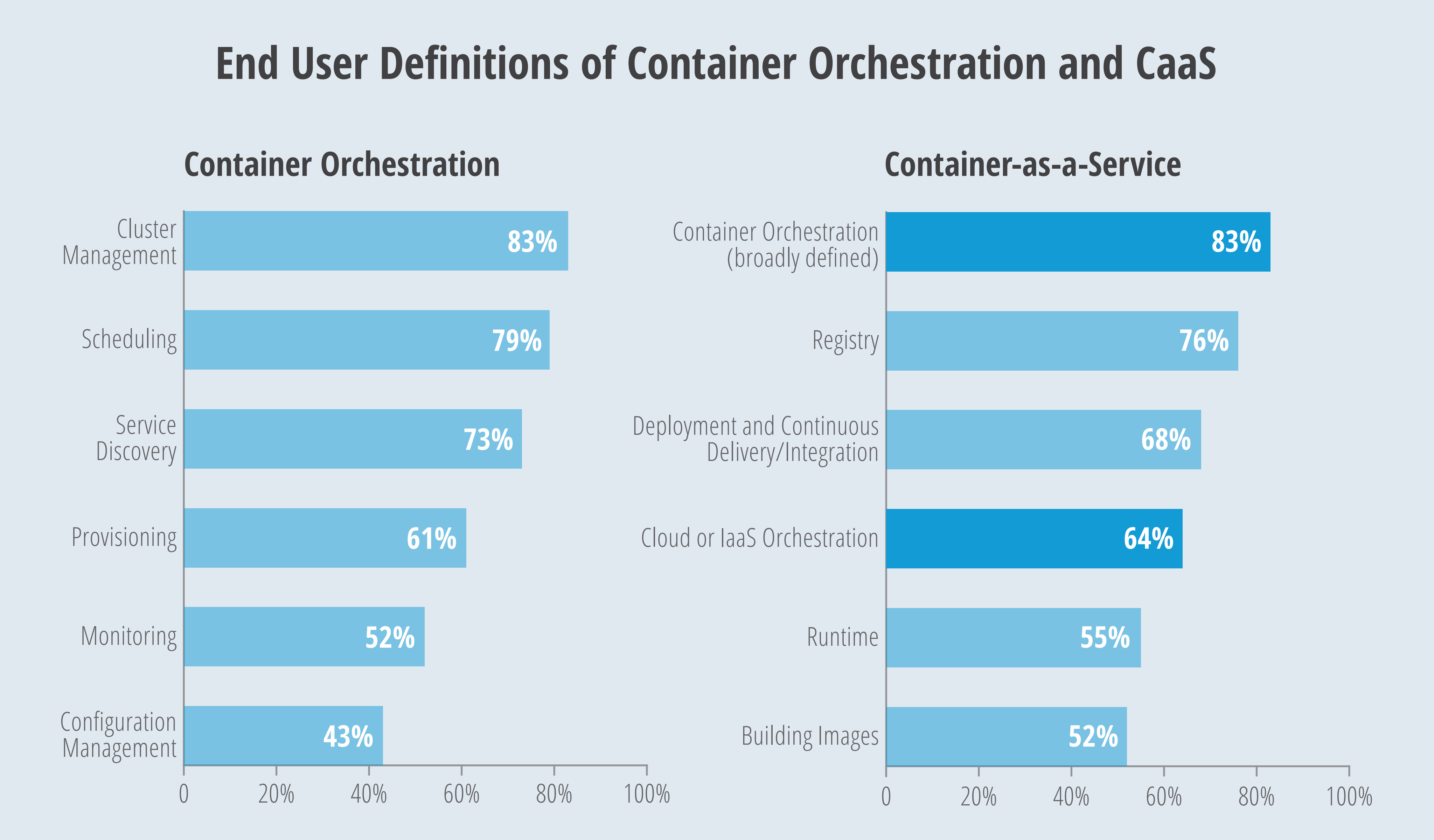
Companies in the business of producing orchestration systems might debate whether these tools constitute a true method for orchestration.

However, the commands used by Compose to start, monitor, stop, and otherwise access running services within containers are regular enough to be automated. By that standard, orchestration is indeed possible. A large focus for us in this ebook has been what companies and users determine to be orchestration, and we broke down what our survey respondents said.

2. Scheduling Multitenant, Microservices Platforms

Some of the most recognized orchestration platforms today are the ones that enable microservices. One of the first organizations to attempt such

FIG 1: *This chart shows what functionality end users expect from container orchestration and Containers as a Service offerings. For CaaS, the relatively informed respondents often expect both orchestration of more than just containers.*



Source: The New Stack Survey, March 2016. What functionality do you expect to be in a product described as a container orchestration tool? Select all that apply. n=188, What functionality do you expect to be in something described as Containers-as-a-Service (CaaS)? Select all that apply. n=183. Open-ended responses excluded from chart.

THENEWSTACK

an architecture was Google, which built an orchestration system called Borg. Its principles were passed on to an open source project called Kubernetes, which Google marshals.

Like Swarm, Kubernetes maintains a pooled, distributed computing environment made up of a cluster of servers, each of which (virtual or physical) is considered a node. Within these nodes, Kubernetes recognizes groups of containers called pods, which may share resources but also run simultaneously.

Pods play a significant role in establishing strategies for distributing workloads. In a microservices-oriented system, individual services and functions may be utilized by more than one application simultaneously. Dividing functions with similar resource needs into pods enables a kind of management scheme where functions within a pod re-provision their own resources based on the demands of the applications running at the time.

This changes the function of load balancing, which began as a matter of replicating servers and became one of the replicating applications. Now, applications don't have to scale as a whole just because their functions do.

Another open source tool widely used for pooling together container resources and scheduling the execution of workloads on that pool is Apache Mesos. Mesosphere produces a commercial orchestration environment that it describes as a "Datacenter Operating System" (DCOS), including a real-time visual representation of multiple workloads running on a cluster of servers.

Marathon, one of the tools that Mesosphere produces for scheduling container workloads on its DCOS platform, was originally offered as an alternative to Kubernetes. Today, the company includes both orchestration systems with DCOS, and gives customers the choice of using either or both in real-world use cases.

3. Jenkins-Integrated Change Control Platforms

Unto itself, Jenkins is not a container orchestrator. It's an open source CI/CD platform that automates the processes of development, testing and staging prior to deploying applications in production. More to the point, Jenkins is intended to regularize the work that people need to do in bringing new software to fruition. In Jenkins, stages of work that can be automated and scripted are called pipelines.

Originally, workloads in Jenkins and other CI/CD systems were encapsulated as virtual machines. As organizations began adopting Docker, it soon became clear that the human work processes associated with containers did not "map" one-to-one with the maintenance of VMs. CloudBees, which produces the commercial version of Jenkins, manufactures a plugin called Workflow that redefines Jenkins' own notion of continuous delivery in order to be more open to containerization. A companion plugin to Workflow is then added to Docker, which enables entry points for pipelines using scripts written in the Groovy language — scripts that are more commensurate with the types of processes that container developers perform daily.

It's important to note here that nothing about the nature of staging workloads in this way, using CloudBees Jenkins, is incompatible with the typical use cases for Kubernetes, Tectonic and Mesosphere/Marathon. While CI/CD pipelining focuses on automating the development process using containers, these other orchestration platforms focus on managing deployment and regular use using containers.

However, within enterprises today, adhering to the ideal of CI/CD runs hand-in-hand with subscribing to the mandates of CM. Jenkins is often integrated with CM tools such as Chef, Puppet, Salt and Ansible. While vendors of these CM tools portray themselves as compatible with orchestration tools, in practice today, many enterprises perceive them as duplicative, and don't use them together.

One alternative orchestrator designed to be compatible at the outset with Jenkins and its pipelined workflows is Docker Cloud. It presents workflow automation in a very straightforward, Amazon-like style that IT administrators may find comfortable.

4. CM-Agnostic Integration Platforms

In almost every software industry there is some platform that presents itself on the virtues of being "end-to-end." For decades, there have been "lifecycle management" tools, and many CM packages have been marketed as such, for better or for worse. But the fact that such platforms are often used together, simultaneously, speaks to the undiscovered fact that "ends," within organizations, tend to move.

Shippable produces a cloud-based continuous integration platform that integrates with Docker, but which avoids Jenkins. It also bypasses typical configuration management, by virtue of how it deploys container images. Choosing to perceive containers as persistent rather than transient, the Shippable approach is to monitor the resources and to adjust their configurations accordingly.

5. Microsegmented, Hybrid VM Platforms

Because containerization platforms have no place for traditional virtual machines, it was inevitable that VMware would directly address the demands of Docker users. The company devised a kind of "embrace and extend" policy that involves wrapping Docker containers in an envelope called "jeVM," such that its existing vSphere environment will accept it just like any other VM.

By including the company's Photon operating system (OS), containers can become hosted by VMware hypervisors instead of a Linux kernel. Orchestration in this scheme becomes a matter of staging container instances, like VMs, in the existing environment. While this disables any prospect for microservices as per the current model, this mode does have the advantage of being instantly integrated with an enterprise's existing CM and CI/CD platforms. However, the relative scalability of container-based applications under this model has yet to be proven.

Summary

This chapter represents the state of configuration management as it exists within the orchestration market, as of the time of its writing; however, we do not cover at length the role of cloud orchestration as a market concept, which is different enough that warrants its own investigation. With Docker itself being only three years old, the facts stated here are likely to become historical in a very short time. What may remain true is the realization that workloads may be manageable independently of the applications through which they were deployed, or the brands of their manufacturers. Orchestration is a new reality today, and several years from now, it may still be a new reality.

CONTAINER ORCHESTRATION AND SCHEDULING: HERDING COMPUTATIONAL CATTLE

by **DANIEL BRYANT**

There is a perfect storm forming within the IT industry that comprises three important trends: the rise of truly programmable infrastructure, which includes cloud, configuration management tooling and containers; the development of adaptable application architectures, including the use of microservices, large-scale distributed message/log processing, and event-driven applications; and the emergence of new processes/methodologies, such as Lean and DevOps.

With all this change, why should we care about a topic like container orchestration and scheduling? We should care for the same reason that the successful farmers from the post-agricultural revolution cared about where they let their cattle graze.

Forgive the analogy, but we are increasingly being asked to think of our infrastructure and applications as cattle. We care about cattle, but perhaps a little less than we would in comparison to our pets. We try not to be reckless with our cattle, and we want them to be grazing on the best land, we want to periodically move them away from crowded or dangerous pasture, and if they do get ill, we want someone to tend them back to full health.

In the world of container platforms, the orchestration and scheduler frameworks perform all of these roles for our application “cattle.” The best application “farmers” maximize resource utilization while balancing the constantly changing demands on their systems with the need for fault-tolerance.

What Type of Container Ranch are You Running?

There is currently a wide range of choice for hosting and deploying containerized applications. If we only consider modern cloud platforms, we can divide the landscape broadly into three categories: Infrastructure as a Service (IaaS), Containers as a Service (CaaS), and Platform as a Service (PaaS). Each category has inherent strengths and weaknesses depending on the types of applications being deployed. Although some organizations will be tempted to choose simply based on intuition, analyst reports or existing vendor relationships, this is a fundamental decision that requires serious evaluation.

The choice of platform used heavily influences the type of orchestration and scheduling that can be implemented. The table following table attempts to provide insight into the key abstractions, control planes and appropriate use cases for each of the three types. It’s important to note that Containers as a Service is still a hotly disputed area — many say that what we’ve traditionally called container services is also a type of Containers as a Service; however, Docker suggests a different set of

Comparing Platforms and Services

Type	IaaS	CaaS / Container Services	PaaS
Description	Virtualized compute resources (CPU, RAM, storage, network, etc.) that must be assembled and ready for app deployment.	Provides an abstraction layer over compute resources that allows the deployment and orchestration of containers.	Provides an abstraction layer over compute infrastructure that allows the deployment and orchestration of applications.
Key abstraction	Compute resources	Containers	Applications
Example	Amazon Web Services, Azure, DigitalOcean, Google Cloud Platform, IBM Cloud Infrastructure, VMware	CoreOS Tectonic, Docker Cloud, Google Container Engine (GKE), HashiCorp Nomad, IBM Container Service, Joyent Triton, Mesosphere Datacenter Operating System (DCOS)	AWS Elastic Beanstalk, Deis, Google App Engine, Heroku, IBM Bluemix, Pivotal Cloud Platform, Red Hat OpenShift
Application Scheduling	Manual or via configuration management (CM) tooling. Resource requirements generally specified by infrastructure selection and configuration. Fault tolerance generally implemented procedurally.	Container scheduling typically baked-in to platform, and capabilities range from crude to sophisticated. Resource requirements are specified declaratively. Fault tolerance handled automatically.	Baked-in to platform. Resource requirements are specified declaratively. Fault tolerance handled automatically.
Service Discovery	Manual or via CM tooling like Puppet, Chef or Ansible. CM tooling typically combined with distributed KV store like Consul or ZooKeeper for service discovery.	Varies from procedural specification (for example, with Mesos, Registrator and srv_router) to declarative specification (for example, with services and labels in Kubernetes).	Typically baked-in to the platform, for example, service discovery provided “as a Service.”
Resource allocation (provisioning)	Manual or via CM tooling like Puppet, Chef or Ansible. Operator decides on resource allocation and distribution for fault tolerance and approach to scaling.	Typically baked-in to platform, and algorithms like bin-packing or spread available to maximize utilization or fault tolerance. Scaling of container instances can be implemented via platform.	Hidden from users. Costs typically based on resource allocation/consumption.
Data Store/ Middleware	Typically Database as a Service (DBaaS) or self-hosted.	Typically IaaS-based DBaaS, potentially efforts to provide containerized databases.	Provided as part of the platform, for example, as a service broker.
Typical Use Case	Organizations performing cloud migrations and migrating on-premises VMs and apps like-for-like, or requiring more fine-grained control over resource assembly and allocation.	Organizations that are already producing cloud-native applications, and are keen to move away from the operational complexity of IaaS, but not wanting to embrace an opinionated PaaS ecosystem.	Ranges from startups looking to cut the costs of maintaining platform infrastructure, to large-scale organizations seeking standardization and developer productivity.

criteria to be considered a CaaS solution, and central to the qualifications is being cloud infrastructure agnostic. We think the larger market interpretation of this product category still has room to be clarified.

Following our farming analogy, we can think of compute resource (CPU, memory, disk, networking, etc.) as land on which our application cattle graze. The role of orchestration and scheduling within a container platform is to match applications to resources — cattle to land — in the most effective and efficient way possible. At first glance, it may not appear to be particularly challenging to efficiently match applications to resources, but the bin-packing approach to optimizing resource usage is computationally an NP-hard problem. When we combine this fact with the volatile and ephemeral nature of the underlying cloud fabric, we definitely have a challenge on our hands.

Corralling Containerized Cattle

Creating a containerized application platform from an IaaS is akin to buying a bare plot of land and building your own ranch from scratch -- fairly arduous, but you have maximum control. Utilizing CaaS is much like buying grazing land, and employing a professional herdsperson to build fences and shepherd the cattle around. This allows you to have visibility of the land, declare your intentions, and have someone else make the moment-to-moment decisions. Deploying applications to a PaaS is effectively trusting your cattle to another farmer. Arguably, here you get to focus on the stuff that really matters (buying and breeding the best stock, taking your cattle to market quickly), but you may not have visibility of the grazing land, and you may not always agree with how the other farmer runs their farm.

In my work at OpenCredo, we have worked with multiple clients to design, build and operate containerized applications and platforms. The following sections share details from two case studies we produced. These examples are meant to demonstrate the decisions that go into creating these orchestration stacks.

Running Spring Boot Services on Mesos and Google Cloud Platform

With our first client, Apache Mesos was the orchestrator being implemented. The client had a desire to mix long-running containerized application services and Spark-based batch data processing. Mesos itself is effectively a “datacenter kernel,” in that it abstracts compute resources from the cluster of infrastructure, and offers these resources to frameworks, such as Mesosphere’s Marathon for long-running services, Chronos for batch jobs, and Spark for data processing workloads. Deploying and running Mesos can be operationally complex; however, it has been proven to work at scale by the likes of Twitter, Airbnb and eBay, and the ability to mix workloads and schedule applications across the entire datacenter is an attractive proposition.

Mesos was deployed and managed with Ansible, and containerized applications built and deployed into the Marathon framework via Jenkins. Service discovery was provided with the combination of Consul, Registrator and srv_router, and service restarts or re-allocations for the purpose of fault-tolerance were provided with Marathon. The QA team were able to spin up personal instances of the Mesos platform within Google Cloud Platform (GCP), which facilitated testing by reducing contention on resources (previously an issue). However, several hard lessons were learned, as the resource-constrained personal environments scheduled workloads differently to the full platform. For example, running multiple Mesos frameworks within a cluster of one machine often led to resource starvation for one of the frameworks.

Deploying Microservices to Kubernetes on AWS

In this project, Kubernetes was deployed onto Amazon Web Services (AWS) via Terraform and Ansible, with the primary goal of providing a level of abstraction for developers above IaaS, but without investing wholesale into a PaaS. Google Container Engine (GKE) could not be used due to client/vendor restrictions. It is easily argued that the vast majority of organizations ultimately create some form of PaaS, because many of the PaaS offerings are required by a typical development team, e.g., testing, deployment, service discovery, storage, database integration and developer community facilitation. Accordingly, this case study did implement many of these features.

Kubernetes provided namespace isolation, which was useful for running multiple test and staging deployments on the same cluster. Separate clusters were used for true isolation. Deployment was managed via the integration of the Kubernetes API or kubectl CLI tooling. Continuous integration framework Jenkins was used, which was also responsible for building and testing our application services. Service discovery was provided out-of-the-box, as was a good level of fault tolerance, with the provision of node health checks and application/container-level liveness probe health checks. Storage and database integration was provided by the underlying IaaS layer, and development patterns were shared and re-used with the combination of a version control system (VCS) and an internal, well-maintained wiki.

Lessons Learned From our Time on the Ranch

The aim of this article is to provide input on choosing a container platform and, based on your requirements, the associated orchestration and scheduling mechanisms. However, this is based primarily on two example client case studies. There are numerous other product combinations and resulting stacks, everything from Docker Swarm mixed with Mesos to focuses on PaaS-based cloud orchestration builds with IBM.

Here are some of our key learning experiences so far at OpenCredo:

Tooling

- Automate as much as possible, with the goal of time saving, repeatability at scale, and the reduction in deviation of infrastructure/configuration.

- Discourage, but allow, manual intervention, especially when introducing this new container technology into an organization. We all know that you shouldn't SSH into production hosts, but leaving this method of ingress open as a last resort measure can be a business saver.

Performance Errors

- Semantic errors — read "business impact" errors — are much more important in the cloud and container world than underlying infrastructure failures. Ultimately, it doesn't matter if you lose half of your production environment, as long as the system gracefully degrades and customers can still get value from the applications. Too many times we have seen operations teams inundated with infrastructure failure alerts, but they do not know the actual user-facing impact.

- Learn about Brendan Gregg's Utilization, Saturation and Error (USE) and Tom Wilkie's Rate, Error and Duration (RED) methodologies. We found these approaches to be incredibly useful at addressing performance issues.

Monitoring and Resource Contention

- There must be visibility at all levels. Container monitoring and management is a major concern for companies implementing containers in production.

- Instrument applications appropriately, and log only essential information.

- Provide business-level metrics. This is especially useful for driving adoption with key stakeholders, and also helps greatly with failures.

- Network: make sure you appropriately size your compute instance if creating your own platform. For example, your application may only require the CPU and memory provided by a "small" instance, but have you confirmed the associated network bandwidth?

- Extensive monitoring is essential at the disk and CPU levels. With the CPU, watch for steal time if you're deploying onto a public cloud.

- Lack of entropy is surprisingly common in an environment where multiple containers are contending over the host /dev/random. Containerized applications will often hang on for no apparent reason, and upon debugging it is observed that a security operation (e.g., session or cryptographic token generation) is blocking it.

UNIFYING AND ORCHESTRATING OPINIONATED FRAMEWORKS AND SERVICES

In this podcast, Michael Hausenblas talks about the critical need to unify components and integrate across systems and frameworks. It's not just about integrating these components, but having an opinionated distribution that can point to services, say how they should be configured, what resources to pull, what nodes to run on, and so on. Michael talks about the vision behind Mesosphere Datacenter Operating System (DCOS), how other components like Marathon and Chronos fit into the picture, and later topics such as self-healing systems and big data.

Listen on SoundCloud or **Listen on YouTube**

Michael Hausenblas *is a developer and cloud advocate at Mesosphere, where he helps AppOps build and operate distributed applications. He shares his experience through demos, blog posts and public speaking engagements, and by contributing to open source software: Apache Mesos, Apache Myriad, Kubernetes. Prior to Mesosphere, Michael was chief data engineer at MapR Technologies.*

THE EMERGING CONTAINERS AS A SERVICE MARKETPLACE

by **ALEX WILLIAMS, SUSAN HALL** *and* **JOAB JACKSON**

While many developers are enthusiastic about the way containers can speed up deployments, administrators and operators may be a bit more wary, given the considerable amount of retooling that their internal systems may need to go through in order to support container-based pipelines.

Which is why the emerging Containers as a Service (CaaS) approach may prove popular to both camps.

> “ CaaS changes the dynamic for how containers are perceived by operations teams.

CaaS changes the dynamic for how containers are perceived by operations teams that must otherwise build-out platforms that manage complex environments. At scale, containers bring with them the need for new tooling, services and platforms. And for most businesses, the expertise is missing to manage complex, scaled-out platforms built on container-based clusters.

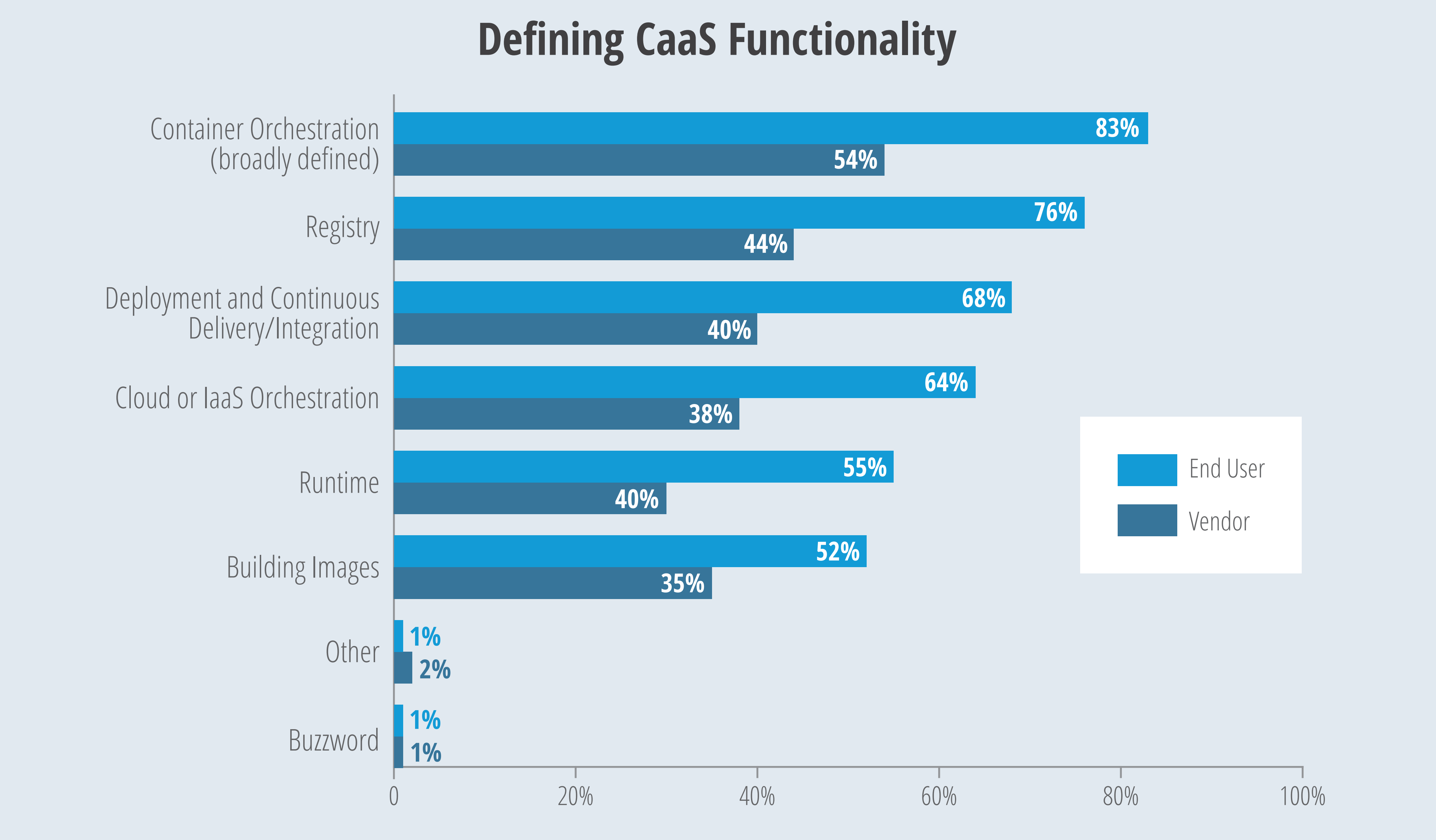
Colm Keegan, senior analyst for Enterprise Strategy Group, explained to The New Stack last year that developers have largely driven the use of containers and have done so somewhat independently. Operations teams haven't had much exposure to container technology. They're unfamiliar with how containers work and the tooling required to manage application-centric architectures. "As a consequence of that, developers, out of necessity, started managing these environments themselves," Keegan said.

CaaS abstracts the complexities of building a scaled-out platform. Further, it means that IT can sanction it as an enabled service and put all the needed controls — such as security and governance — around it, providing the service not only in the local data center, but also possibly across public clouds.

Birth of a True Product Category

The year 2015 brought widespread interest from operations organizations about how to best deploy and manage containers. In the past two years, many cloud services providers have developed their own CaaS platforms. CaaS is becoming the new Platform as a Service (PaaS), writes Janakiram MSV, an analyst and writer for The New Stack in an article for Forbes.

The services point to some distinctions for how CaaS, like other "as a service" offers, can have so many different meanings. It's important to note that Containers as a Service is still a hotly disputed area — many say that what we've traditionally called container services is also a type of



Source: The New Stack Survey, March 2016. What functionality do you expect to be in something described as Containers-as-a-Service (CaaS)? Select all that apply. End Users, n=183, Vendor (works for company that provides products or services related to containers, PaaS, IaaS or software management), n=115.

THENEWSTACK

FIG 1: *The New Stack's 2016 container orchestration survey looked at end users' and vendors' perspectives on what functionality they would expect from Containers as a Service.*

Containers as a Service; however, Docker suggests a different set of criteria to be considered a CaaS solution, and central to the qualifications is being cloud infrastructure agnostic. We think the larger market interpretation of this product category still has room to be clarified.

Janakiram MSV writes that Azure Container Service is an interface between its own assets and the orchestration layer. That allows for the mapping of the block storage, the VMs and other assets to the orchestration layer. The goal: give the developer the ability to launch a container cluster from the command line as simply as they can launch an "HDInsight Hadoop cluster on Linux."

Google Container Engine (GKE) also serves as the separation between Google Compute Engine and Kubernetes. It acts as a thin layer that bridges the gap between Google Compute Engine and the orchestration engine powered by Kubernetes. GKE schedules containers into the cluster and manages them automatically based on the user-defined requirements, such as CPU and memory. It's built on the Kubernetes system, giving flexibility to take advantage of on-premises, hybrid or public cloud infrastructure.

According to Janakiram MSV, clusters on GKE are comprised of virtual machines. As he writes, this is pretty conventional, as most infrastructure is a pool of virtual machines.

Amazon EC2 Container Service (ECS) runs on the EC2 Infrastructure as a Service (IaaS) layer to deliver hosted container services. Amazon's service is entirely proprietary. It supports Docker containers and allows the user to run applications on a managed cluster of Amazon Elastic Compute Cloud (EC2) instances.

You do not need to install, operate, and scale your own cluster management infrastructure. With simple API calls, you can launch and stop container-enabled applications and discover the state of your cluster. You can use Amazon EC2 with other AWS services, and you can take advantage of familiar features, such as Elastic Load Balancing, EBS volumes, EC2 security groups, and IAM roles.

IBM launched IBM Containers on Bluemix in June 2015, providing one of the first hosted Docker container services. Bluemix is IBM's cloud platform providing access to PaaS, IaaS, and now CaaS providing choices to meet the user's requirements. With IBM Containers, the user's Docker experience begins with deploying containers as opposed to deploying a cluster of virtual machines as Docker hosts. IBM Containers allows the administrators to specify user quota and determine which images from the private, hosted Docker registry can be deployed. The offering also provides integrated monitoring and logging at the container level, overlay networking, scalable groups with integrated load balancer and auto-recovery, storage volumes for persistent data needs, content and policy vulnerability insight to every Docker image in the registry, and the ability to bind to any of the 150+ services in the Bluemix catalog.

Joyent has Triton, a service that treats containers natively and allows containers to run on bare metal while eliminating the VM abstraction layer, which Bryan Cantrill, chief technical officer at Joyent, calls "an unnecessary layer of fat." The challenge to that, he said in a story on The New Stack, has been the Linux substrate used for containers — namely namespaces and cgroups — was really not designed for multi-tenancy, creating security issues that make it either difficult or unwise to deploy Linux containers on bare metal.

The Triton service instead uses Joyent's SmartOS substrate, which builds on some previous security work done by Sun Microsystems in the open source variant of Solaris.

Rackspace launched Carina, a CaaS offering that Rackspace hosts and manages for the user. Carina's web-based console is designed to instantiate and deploy scalable container images. From there, Rackspace's administrators take over the roles of scheduling and orchestration, scaling up workloads as necessary and assuming responsibility for all the maintenance.

Carina uses the Open Container Initiative, allowing it to absorb containers and follow a customer's instructions about how they are to be deployed, including their choice of bare metal or VM-based infrastructure.

OpenStack's Nova scheduler recognizes host aggregates, or clusters of physical servers designed to be separated from other availability zones. Servers within these host aggregates may be a particular "flavor," which (theoretically) enables Windows hosting on OpenStack. It also allows for container hosting.

Within these host aggregate clusters, Magnum pools together Nova instances into what it calls bays. These individual bays can conceivably be orchestrated using Docker Swarm, Kubernetes and Mesos. As Carina is built out, users will be allowed their preference of orchestration engine.

In early 2016, Docker released software for running CaaS operations with the Docker Datacenter (DDC). It provides a way to deploy applications without worrying about issues that might arise from moving a codebase from development into production. The integrated package is a collection of open source technologies for deploying and managing containers.

"It's not just the recognition that we need to manage these containers, but offering a suite of products to manage those successfully," Scott Johnston, Docker chief operating officer, said.

Applications are driving this next wave, and operations wants to support it, not get in the way of it. Users need products that can provide governance over those compute, network and storage resources, and at the same time provide the flexibility, agility and portability of applications that the development team are producing.

> “Ops is writing the checks, but they’re trying to be a partner in this workflow that is largely driven by the application development teams.

Johnston used security as an example, saying 2016 will bring more requirements for end-to-end security solutions. Teams need security to be a consideration from the beginning, not something that’s only a consideration for production environments. If you’re waiting that long to consider the security of your containers, it can never be more than a patch or a quick fix.

The security-related technology image signing will be an early draw. As a containerized application moves through development, different departments can sign it, such as QA, providing a trail of the steps showing where the software has traveled.

“That allows the Ops team to make real-time decisions — and ultimately automated decisions — on how to deploy that application,” Johnston said. One container may have a “super-secret IP,” so it must stay in-house, whereas a simple test application can be deployed to a public cloud. This early consideration for security is a critical investment in how your stack performs in the future. “Security is a great example of how doing things at the beginning of the pipeline gives you huge benefits downstream.”

Image signing is just one example of how Containers as a Service provides the agility that developers are looking for and the control that operations is looking for. Johnston sees similar developments with networking and storage.

"Enterprises are saying, 'Don't cause me to break my app as it moves through the application stages from dev to QA to prod. Let my app maintain its logical relationships and swap out the actual implementation as it goes through those stages,'" Johnston said. "This is putting pressure on storage and networking vendors to produce implementation plugins, and has given us great feedback in terms of what those APIs for networking and storage, from an application viewpoint, need to look like."

Interoperability for the Future

Cantrill sees the real challenge as clearing up the confusion surrounding containers. The impediments to containers are seemingly rival solutions in all these different parts of the stack, and many don't understand how the pieces compete with one another or fit in with one another, Cantrill said. Interoperability has been a tertiary concern.

As a result, interoperability will be a much larger concern in 2016, he predicted. There is a critical need to define how these solutions interface, or developers won't be able to innovate beneath that level of integration.

Cantrill says this concern for interoperability is not what some vendors want to hear, because they'd like to think they can provide that next "magic stack" as a singular solution, without the help of others. Cantrill and many others don't see this one-solution approach being feasible — orchestration and management will need the collaboration of many tools.

> “It’s going to be much more heterogeneous,
> much more composable,
> much more interoperable than that.

Summary

The introduction of Containers as a Service to the marketplace, and its growing user base, represents a maturity in container management that appeals to both developer and operations teams. It represents the possibility of a radically shorter lifecycle for adopting containers across the entire business, instead of the crawl between development, operations, and true usage in production. Offering container services allows for pre-packaged security and governance, providing the service in the local data center and across public clouds.

As this new type of product and service evolves over time, it’s important that users and vendors look to define the functionality that you expect, and that you know your team needs. Meeting functional needs in an interoperable package is going to be critical to the future of the entire orchestration market.

BUILDING OPEN SOURCE TOOLS FOR THE DEMOCRATIC INTERNET

Solomon Hykes of Docker talks about the need for greater public access to programming skills and tools — part of what he refers to as a democratization of the Internet. Plumbing the Internet as a distributed utility has a huge potential to unlock innovation. Hykes talks about how his industry has otherwise failed to achieve these accessibility goals, but points out that he believes Docker, and the tools they are producing, are a small step in the right direction. He believes that Docker creates a foundational technology that could enable a new change. Later, he addresses criticisms of Docker's role in infrastructure, Docker's approach to building new tools, and finding common frameworks to integrate all your components. **Listen on SoundCloud** or **Listen on YouTube**

Solomon Hykes *is the founder, chief technology officer and chief architect of Docker. In his role at Docker, he is focused on building a platform for developers and system administrators to build, ship, run and orchestrate distributed applications. A Forbes 30 under 30 and YCombinator alumni, Solomon led dotCloud as CEO through five years of fundraising, business operations and product launches before focusing entirely on Docker. Prior to dotCloud, Solomon held engineering, IT infrastructure management and software development roles at CEIS and SmartJog.*

AUTOMATION
& ORCHESTRATION
DIRECTORY

BUILD/DEPLOY (INCLUDING CONTINUOUS INTEGRATION/ DELIVERY)

	Product/Project (Company or Supporting Org.)	Also Categorized In:
	Active Deploy (IBM) Active Deploy allows you to release a new version of your software with no downtime. If at any time during the release a problem occurs, Active Deploy allows you to quickly revert back to the original version. You finalize the new version only when it has shown to work properly.	
Open Source	**Ant (Apache Software Foundation)** Apache Ant is a Java library and command-line tool used to drive processes described in build files as targets and extension points dependent upon each other. Ant supplies a number of built-in tasks allowing users to compile, assemble, test and run applications.	
	AppVeyor (AppVeyor) AppVeyor is a continuous delivery service for Windows, focusing on .NET developers.	
	Artifactory (JFrog) JFrog provides software developers with a binary repository management solution that integrates into CI/CD pipelines.	
	Atlas (HashiCorp) Atlas unites HashiCorp development and infrastructure management tools to create a version control system for infrastructure.	
	AWS CodeDeploy (Amazon Web Services) AWS CodeDeploy is a deployment service that enables developers to automate the deployment of applications to instances and to update the applications as required.	
	AWS OpsWorks (Amazon Web Services) AWS OpsWorks provides a simple and flexible way to create and manage stacks and applications.	
	Bamboo (Atlassian) Bamboo is a continuous integration and delivery tool that ties automated builds, tests and releases together in a single workflow.	

	Product/Project (Company or Supporting Org.)	Also Categorized In:
	Bluemix (IBM) Bluemix is IBM's digital application platform, enabling you to develop, deploy, manage and run cloud applications. With Bluemix you can utilize Docker containers, virtual machines and Cloud Foundry applications using open APIs. It has a catalog of over 150 services, including support for microservices, logging and monitoring and Watson analytics.	» Platforms/PaaS
Open Source	**Boxupp (Paxcel Technologies)** Boxupp is an interface which simplifies creation of virtual machines and containers by using powerful tools like Vagrant, Puppet and Docker.	
Open Source	**Brooklyn (Apache Software Foundation)** Apache Brooklyn is a library and control plane for deploying and managing distributed applications.	
Open Source	**Buildbot (N/A)** Buildbot is a Python-based open source framework for automating software build, test and release processes.	
	BuildMaster (Inedo) BuildMaster is an application release automation tool for DevOps that creates complex release pipelines — all managed from one central dashboard.	
	Calm.io (Calm.io) Calm.io is a platform to deploy, manage and maintain distributed applications in private and public cloud environments. It claims to offer full-stack orchestration, run item lifecycle management, and policy-based governance.	» Orchestration
Open Source	**Capistrano (N/A)** Capistrano is a remote server automation and deployment tool written in Ruby.	
Open Source	**Capsule (N/A)** Capsule is a packaging and deployment tool for JVM applications.	
Open Source	**Centurion (New Relic)** Centurion is a deployment tool for Docker. It ships containers to and from a registry, running them on a fleet of hosts.	
	Chef Delivery (Chef) Chef Delivery provides a workflow and visualization tool to help manage the continuous delivery pipeline.	
	CircleCI (CircleCI) CircleCI provides a continuous delivery process for Docker applications.	

	Product/Project (Company or Supporting Org.)	Also Categorized In:
	Cloud 66 (Cloud 66) Cloud 66 is an application provisioning and management service that allows you to build Docker stacks from scratch on any public or private cloud vendor or your own infrastructure.	» Orchestration
Open Source	**Cloud Foundry Lattice (Cloud Foundry Foundation)** Lattice is an open source project for running containerized workloads on a cluster. Lattice bundles up HTTP load balancing, a cluster scheduler, log aggregation, and streaming and health management into an easy-to-deploy and easy-to-use package.	» Orchestration
	CloudBees Jenkins Platform (CloudBees) CloudBees Jenkins Platform is a PaaS that supports continuous integration and delivery of mobile and web applications. It's the enterprise version of Jenkins that has multiple plugins to support Docker.	
Open Source	**Cloudbreak (Hortonworks)** Cloudbreak helps users launch on-demand Hadoop clusters in the cloud or to any environment that supports Docker containers.	
Open Source	**CloudSlang (Hewlett-Packard Enterprise)** CloudSlang is a flow-based orchestration tool for managing deployed applications. It aims to automate DevOps workflows and offers Docker capabilities and integrations.	» Orchestration
	Codefresh (Codefresh) Codefresh combines Docker tools with a web IDE based on Eclipse's Orion. The result is what they call a complete development environment for Node.js with DevOps and QA teams.	
	Codenvy (Codenvy) Codenvy provisions, shares and scales developer environments. Users can use a Dockerfile to launch a custom stack.	
	Codeship (Codeship) Codeship provides a hosted continuous delivery platform for web applications. It can be used to test Docker apps on different operating systems.	
	ContainerShip (ContainerShip) ContainerShip is a self-hosted container management platform, capable of running on any cloud, and used to manage containers from development to production.	» Platforms/PaaS
Open Source	**Continuum (Apache Software Foundation)** Apache Continuum is an enterprise-ready continuous integration server with features such as automated builds, release management, role-based security, and integration with popular build tools and source control management systems.	

	Product/Project (Company or Supporting Org.)	Also Categorized In:
	DCHQ (DCHQ) DCHQ is a deployment automation and governance platform for Docker-based application development. It provides self-service access to Docker-based applications using an agent-based architecture for orchestration.	» Orchestration
	Delivery Pipelines (IBM) As you develop an app in the cloud, you can choose from several build types. You provide the build script, and IBM Bluemix DevOps Services runs it. With one click, you can automatically deploy your app to one or many Bluemix spaces, public Cloud Foundry servers, or Docker containers on IBM Containers for Bluemix.	
	Distelli (Distelli) Distelli manages your applications as they progress through the development lifecycle with complete visibility of who did what and when along the way.	
Open Source	**Docker (Engine) (Docker)** Docker Engine is a lightweight runtime and tooling that builds and runs your Docker containers.	» Orchestration
	Docker Cloud (Docker) Docker Cloud is the new cloud service by Docker that expands on the features of Tutum and brings a tighter integration with Docker Hub; it is a SaaS service for deploying and managing Dockerized applications.	» Orchestration, Platforms/PaaS
Open Source	**Docker Compose (Docker)** Compose is a tool for defining and running multi-container applications with Docker.	» Orchestration
	Docker Datacenter (Docker) Docker Datacenter provides on-premises container management and deployment services to enterprises with a production-ready platform supported by Docker and hosted locally behind the firewall.	» Orchestration, Platforms/PaaS
Open Source	**Docker Distribution (Docker)** A toolset to pack, ship, store and deliver content. It supersedes the docker/docker-registry project with a new API design, focused around security and performance.	
Open Source	**Docker Engine for SmartDataCenter (Joyent)** A Docker Engine for SmartDataCenter, where the data center is exposed as a single Docker host. The Docker remote API is served from a "docker" core SDC zone built from this repo.	
	Docker Hub (Docker) Docker Hub is a cloud-based registry service for building and shipping application or service containers. It provides a centralized resource for container image discovery, distribution and change management.	
	Docker Image Build Service (IBM) Docker Image Build Service is a cloud-hosted build service published to a private Docker registry on Bluemix.	

	Product/Project (Company or Supporting Org.)	Also Categorized In:
Open Source	**Docker Machine (Docker)** Docker Machine lets you create Docker hosts on your computer.	» Orchestration
Open Source	**dockersh (Yelp)** Dockersh is a login shell for machines with multiple users; it gives access to multiple users but allows for isolation between them.	
	Drone (Drone.io) Drone is a continuous integration system built on top of Docker. Drone uses Docker containers to provision isolated testing environments.	
Open Source	**Dusty (GameChanger Media)** Dusty is a development environment that provides OS X support for containers. Their documentation compares the tool to Vagrant and says it uses Docker Compose to orchestrate containers.	
	ElasticBox (ElasticBox) ElasticBox is a service for connecting CI/CD pipelines, configuration management tools, and deploying cloud applications.	
	ElectricFlow (Electric Cloud) ElectricFlow orchestrates and automates DevOps tools in software delivery processes.	
Open Source	**fabric8 (Red Hat)** fabric8 is an open source DevOps and integration platform that is built as a set of microservices that run on top of Kubernetes and OpenShift V3. Its continuous delivery is based on Jenkins, Nexus and SonarQube. Its iPaaS integrates with Apache ActiveMQ, Camel and CXF.	» Platforms/PaaS
Open Source	**Ferry (N/A)** Ferry is a big data development environment. It lets you define, run and deploy big data applications on AWS, OpenStack and your local machine using Docker.	
	Flockport Apps (Flockport) Flockport is a Linux container-sharing website. It also provides tools that make it easier to install and use LXC containers.	
	Microscaling Systems (Microscaling Systems) Microscaling Systems scales containers in real time in response to demand.	» Orchestration
	Giant Swarm (Giant Swarm) Giant Swarm is a hosted container solution to build, deploy and manage containerized services.	» Platforms/PaaS

	Product/Project (Company or Supporting Org.)	Also Categorized In:
	GitLab Continuous Integration **(GitLab)** GitLab CI is the part of GitLab that assists in testing, building and deploying code.	
Open Source	Go Continuous Delivery **(ThoughtWorks)** Go Continuous Delivery (GoCD) helps you automate and streamline the build-test-release cycle for worry-free, continuous delivery of your product. This is a Java/JRuby on Rails project. It is an open source continuous delivery server with pipelines at its core.	
Open Source	Gradle **(Gradle)** Gradle is a build automation system. Gradle has been designed to support build automation across multiple languages and platforms — including Java, Scala, Android, C/C++ and Groovy — and is closely integrated with development tools and continuous integration servers, including Eclipse, IntelliJ, and Jenkins.	
Open Source	Gump **(Apache Software Foundation)** The Apache Gump continuous integration tool is written in Python and fully supports Apache Ant, Apache Maven and other build tools. Gump is unique in that it builds and compiles software against the latest development versions of those projects.	
	Hook.io **(Hook.io)** Hook.io is an open source hosting platform for microservices.	» Platforms/PaaS
	Hortonworks Data Platform **(Hortonworks)** Hortonworks Data Platform is a commercial Hadoop offering. With its 2015 purchase of SequenceIQ, Hortonworks acquired Cloudbreak's ability to launch on-demand Hadoop clusters in the cloud or to any environment that supports Docker containers.	
Open Source	Hudson Continuous Integration Server **(Eclipse Foundation)** Hudson monitors the execution of repeated jobs, such as building a software project or jobs run by cron.	
Open Source	Instainer **(Instainer)** Instainer allows for fast deployment using Heroku-style Git deployment.	
Open Source	ION-Roller **(Gilt)** ION-Roller is an AWS immutable deployment framework for web services.	
Open Source	Janky **(GitHub)** Janky is a continuous integration server built on top of Jenkins, controlled by Hubot, and designed for GitHub.	
Open Source	Jenkins **(N/A)** Jenkins is an extensible open source continuous integration server.	

	Product/Project (Company or Supporting Org.)	Also Categorized In:
Open Source	**Kafka Deploy (N/A)** Kafka Deploy enables automated deployment for a Kafka cluster on AWS.	
	LaunchDarkly (LaunchDarkly) LaunchDarkly is a service for launching, controlling and measuring features, including performance testing, feature flagging and controlled rollouts.	
Open Source	**Leiningen (N/A)** Leiningen is a build manager for Clojure.	
Open Source	**Mantl (Cisco)** Mantl is an open source platform for rapidly deploying globally distributed services. It works with tools such as Marathon, Mesos, Docker and Consul.	» Platforms/PaaS
Open Source	**Maven (N/A)** Apache Maven is a software project management and comprehension tool. Based on the concept of a project object model (POM), Maven can manage a project's build, reporting and documentation from a central piece of information.	
Open Source	**MSBuild (Microsoft)** The Microsoft Build Engine is a platform for building applications. This engine, which is also known as MSBuild, provides an XML schema for a project file that controls how the build platform processes and builds software. Visual Studio uses MSBuild, but MSBuild does not depend on Visual Studio.	
	Navops Launch (Univa) Navops Launch allows users to build container clusters using Kubernetes.	
	Nitrous (Nitrous) With Nitrous, developers can utilize pre-built development environment templates from the Nitrous standard library, or use their companies' own private templates.	
Open Source	**nscale (nearForm)** nscale is an open toolkit which supports configuration, build and deployment of connected container sets.	
Open Source	**Octopus Deploy (Octopus)** Octopus is a deployment automation tool for .NET developers. It works with your build server to enable reliable, secure, automated releases of ASP.NET applications and Windows Services into test, staging and production environments.	
Open Source	**Otto (HashiCorp)** Otto is a development and deployment tool that is the successor to Vagrant.	

	Product/Project (Company or Supporting Org.)	Also Categorized In:
Open Source	**Packer (HashiCorp)** Packer is a tool for creating machine and container images for multiple platforms from a single source configuration.	
	Pertino for Docker (Pertino) Pertino for Docker lets developers build container-level virtual private cloud networks.	
Open Source	**Powerstrip (ClusterHQ)** Powerstrip is a tool for prototyping Docker extensions.	
Open Source	**Project 6 (Diamanti)** Project 6 is software for deploying and managing Docker containers across a cluster of hosts, with a focus on simplifying network and storage configurations for on-premises environments. It builds on Google's Kubernetes, Docker and CoreOS's etcd.	» Orchestration
	Project Shipped (Cisco) Cisco's Project Shipped is a model for deploying microservices to the cloud. To reduce environmental inconsistencies, Project Shipped also emulates the cloud environment on your development workstation. Vagrant, Consul and Cisco Microservices Infrastructure are components.	
	PureApplication (IBM) PureApplication offers solutions that use automated patterns to rapidly deploy applications, reduce cost and complexity, and ease management. Docker containers can be included in patterns along with non-container components to take advantage of the PureApplication enterprise-grade lifecycle management. Support includes a private Docker registry pattern deployable as a shared service.	
	QuickBuild (PMEase) QuickBuild is a build tool for CI/CD pipelines, pre-commit testing, issue tracking and more.	
	RapidDeploy (MidVision) RapidDeploy is an application release and deployment automation tool.	
Open Source	**RPM Maven Plugin (N/A)** The RPM Maven Plugin allows artifacts from one or more projects to be packaged in an RPM for distribution.	
	Runnable (Runnable) Runnable works with GitHub and other tools to automatically deploy commits and launch containers in your sandbox when branches are created.	
Open Source	**sbt (Lightblend)** sbt is an open source build tool for Scala and Java projects, similar to Java's Maven or Ant.	

	Product/Project (Company or Supporting Org.)	Also Categorized In:
	Semaphore (Rendered Text) Semaphore is a hosted continuous integration service.	
	Shippable (Shippable) Shippable is a continuous integration platform built natively on Docker and using Docker Hub to deploy.	
Open Source	**Shutit (N/A)** ShutIt is a tool for managing the Docker image building process; it expands on the capabilities of Dockerfiles.	
	Snap (ThoughtWorks) Snap continuous delivery tools let users set up hosted deployment pipelines from Github to platforms like Heroku, AWS and DigitalOcean.	
Open Source	**Spinnaker (Netflix)** Spinnaker is an open source, multi-cloud, continuous delivery platform for releasing software changes with high velocity and confidence.	
	Turbo (Turbo.net) Turbo is a platform for building, testing and deploying Windows applications and services in isolated containers.	
Open Source	**Spring Boot (Pivotal Software)** Spring Boot makes it easy to create stand-alone, production-grade Spring-based applications.	
	StackEngine (Oracle) StackEngine is an end-to-end container application management system that provides a way for dev and enterprise IT teams to deploy Docker applications.	
Open Source	**StackHut (Stackhut)** StackHut is a platform to develop and deploy microservices without writing any server-logic.	
	Stacksmith (Bitnami) Bitnami Stacksmith allow users to create container images. Bitnami is a library of server applications and development environments that can be installed with one click.	
	Team Foundation Server (Microsoft) An enterprise-grade server for teams to share code, track work and ship software — in any language — all in a single package.	
	TeamCity (JetBrains) TeamCity is a continuous integration and build server for developers and DevOps.	

	Product/Project (Company or Supporting Org.)	Also Categorized In:
Open Source	**Terraform (HashiCorp)** Terraform is a tool to build and launch infrastructure, including containers.	» Platforms/PaaS
Open Source	**Totem (N/A)** Totem is a continuous delivery pipeline tool designed for microservices.	
	Travis CI (Travis CI) Travis CI is an open source continuous deployment platform; it is able to run on Docker-based infrastructures.	
	Turbo.net (Turbo.net) Available as both a web-based service and on-site server, Turbo.net allows customers to build, test, deploy and manage desktop application containers.	
	UrbanCode Build (IBM) UrbanCode Build is a continuous integration and build management server optimized for the enterprise. It is designed to make it easy to scale and seamlessly plug into development, testing and release tooling. It supports Docker build and integration with Docker registries.	
	UrbanCode Deploy (IBM) UrbanCode Deploy orchestrates the deployment of applications across environments, coordinating the deployment of many individual components with inventory tracking. This includes support for Docker Containers, Docker Registries, and IBM Container Service on Bluemix via a community plugin.	
	UrbanCode Release (IBM) UrbanCode Release is a collaborative release management tool that handles the growing number and complexity of releases.	
Open Source	**Vagrant (HashiCorp)** Vagrant is a tool for building and distributing development environments. Development environments managed by Vagrant can run on containers.	
Open Source	**Vamp (Magnetic.io)** Vamp stands for Very Awesome Microservices Platform. It helps developers build, deploy and manage microservices. Vamp's core features are a platform-agnostic microservices DSL, powerful A/B testing, canary releasing, autoscaling and an integrated metrics/event engine.	
	Velocity (Mesosphere) Velocity is a recently announced tool that will serve as a kind of "Jenkins-as-a-Service," or quite literally, a sort of "ephemeral Jenkins," where staging environments are spun up and wound down as necessary for particular projects.	

	Product/Project (Company or Supporting Org.)	Also Categorized In:
Open Source	**Vessel (N/A)** Vessel automates the setup and use of Dockerized development environments. It requires both OS X and Vagrant to work properly.	
	Virtuozzo (Odin) Virtuozzo is a container virtualization platform sold to providers of cloud services.	
	Visual Build (Kinook Software) Visual Build enables developers and build masters to easily create an automated, repeatable build process.	
	Weave Flux (Weaveworks) Weave Flux provides routing and control for microservices applications.	
	Wercker (Wercker) Wercker is a platform for automating the creation and deployment of applications and microservices.	
	xDock (Xervmon) Xervmon is a cloud management platform. Its xDock lets users deploy, manage and monitor Docker images in the cloud.	
	XL Deploy (XebiaLabs) XL Deploy provides application release automation to standardize complex deployments, speed up development time, and gain visibility into the pipeline.	
	XL Release (XebiaLabs) XL Release automates, orchestrates and provides visibility into release pipelines. It identifies bottlenecks, reduces errors and lowers the risk of release failures.	
Open Source	**Zodiac (CenturyLink)** Zodiac is a lightweight tool, built on top of Docker Compose, for easy deployment and rollback of "Dockerized" applications.	

ORCHESTRATION: CLUSTER MANAGEMENT

	Product/Project (Company or Supporting Org.)	Also Categorized In:
Open Source	**Aurora (Apache Software Foundation)** Apache Aurora lets you use an Apache Mesos cluster as a private cloud. It supports long-running services, cron jobs, and ad hoc jobs.	
	Azure Container Service (Microsoft) Azure Container Service simplifies the creation and configuration of a cluster. The default configuration of this cluster includes Docker and Docker Swarm for code portability and Marathon, Chronos and Apache Mesos to ensure scalability.	» Platforms/PaaS
Open Source	**Cloud Foundry Lattice (Cloud Foundry Foundation)** Lattice is an open source project for running containerized workloads on a cluster. Lattice bundles up HTTP load balancing, a cluster scheduler, log aggregation, and streaming and health management into an easy-to-deploy and easy-to-use package.	» Build/Deploy
Open Source	**Docker Swarm (Docker)** Docker Swarm offers native clustering for Docker by turning multiple Docker hosts into a single, virtual host.	
	Elastigroup (Spotinst) Elastigroup is a cost-oriented cluster that offers reliable and efficient use in the AWS Spot market and Google Preemptible VMs.	
Open Source	**Firmament (University of Cambridge)** Firmament is a cluster manager and scheduling platform.	
Open Source	**Fleet (CoreOS)** Fleet is a distributed init system used to support cluster management and orchestration of containers.	
	Google Container Engine (Google) Google Container Engine is cluster management and orchestration system that lets users run containers on the Google Cloud Platform.	

	Product/Project (Company or Supporting Org.)	Also Categorized In:
Open Source	**Helios (Spotify)** Helios is a Docker container orchestration platform.	
Open Source	**Helix (Apache Software Foundation)** Helix is a generic, cluster management framework used for the automatic management of partitioned, replicated and distributed resources hosted on a cluster of nodes.	
Open Source	**Mesos (Apache Software Foundation)** Apache Mesos is a cluster manager that provides efficient resource isolation and sharing across distributed applications or frameworks.	
Open Source	**Poseidon (University of Cambridge)** Poseidon is Firmament's integration with Kubernetes.	
Open Source	**Project 6 (Diamanti)** Project 6 is software for deploying and managing Docker containers across a cluster of hosts, with a focus on simplifying network and storage configurations for on-premises environments. It builds on Google's Kubernetes, Docker and CoreOS's etcd.	» Build/Deploy
Open Source	**Shipyard (N/A)** Shipyard enables multi-host, Docker cluster management, and is fully compatible with the Docker Remote API.	
	Universal Control Plane (Docker) Universal Control Plane offers an on-premises management solution for Docker apps.	

ORCHESTRATION: CONFIGURATION MANAGEMENT

	Product/Project (Company or Supporting Org.)	Also Categorized In:
Open Source	**Ansible (Ansible)** Ansible is a configuration management tool that can orchestrate the deployment of Docker containers. In addition, it lets users create container images.	
	Ansible Tower (Ansible) Ansible is a configuration management tool that can orchestrate the deployment of Docker containers. In addition, it lets users create container images.	
Open Source	**CFEngine (CFEngine)** CFEngine is a configuration management and automation framework that lets you securely manage your mission critical IT infrastructure.	
Open Source	**Chef (Chef)** Chef is a configuration management and automation platform. It can be used to create, manage and provision Docker containers.	
	Chef Server (Chef) Chef Server is the highly scalable foundation of the Chef automation platform.	
Open Source	**Cobbler (N/A)** Cobbler is a Linux installation server that allows for rapid setup of network installation environments. Cobbler can help with provisioning, managing DNS and DHCP, package updates, power management, configuration management orchestration and more.	
	Control-M (BMC Software) BMC Control-M is a digital enterprise automation solution that simplifies and automates diverse batch application workloads while reducing failure rates, improving SLAs, and accelerating application deployment.	
	Otter (Inedo) Otter acts as “infrastructure as code” by combining server configuration with IT automation.	

	Product/Project (Company or Supporting Org.)	Also Categorized In:
	Puppet Enterprise **(Puppet Labs)** Puppet Enterprise is a configuration management and automation tool, including built-in tools for building Dockerfiles.	
Open Source	Rudder **(Normation)** Rudder is a web-driven, role-based solution for IT infrastructure automation and compliance.	
	SaltStack Enterprise **(SaltStack)** SaltStack is an orchestration and automation tool that can be used to manage Docker containers.	

ORCHESTRATION: SCHEDULING

	Product/Project (Company or Supporting Org.)	Also Categorized In:
	Apcera Platform (Apcera) Apcera manages access to compute resources across a cluster of servers. By focusing on managing policies across multiple environments, it aims to secure workloads and containers in enterprise production environments.	» Platforms/PaaS
Open Source	**BOSH (Cloud Foundry Foundation)** BOSH is an open source toolchain for orchestration of large-scale distributed services that can be used with Docker containers. BOSH installs and updates software packages on large numbers of VMs over many IaaS providers.	
Open Source	**Chronos (Apache Software Foundation)** Chronos is a distributed and fault-tolerant scheduler that runs on top of Mesos that can be used for job orchestration. Chronos is natively able to schedule jobs that run inside Docker containers.	
Open Source	**Cloud Foundry Containers Service Broker (Cloud Foundry Foundation)** This is a container services broker for the Cloud Foundry v2 services API. It allows for the provisioning of services and binding of applications to a container backend.	
Open Source	**Cloud Foundry Diego (Cloud Foundry Foundation)** Diego is Cloud Foundry's next generation runtime. It is made up of a variety of small components, each with its own repository. It supports the scheduling and monitoring of Docker workloads.	
Open Source	**CloudSlang (Hewlett-Packard Enterprise)** CloudSlang is a flow-based orchestration tool for managing deployed applications. It aims to automate DevOps workflows and offers Docker capabilities and integrations.	» Build/Deploy
	Cloudsoft Application Management Platform (Cloudsoft) Cloudsoft's application management platform, based on the open source Apache Brooklyn project, orchestrates services, platforms and infrastructure, including deployment to containers.	» Platforms/PaaS

	Product/Project (Company or Supporting Org.)	Also Categorized In:
Open Source	**Crane (N/A)** Crane is a lightweight wrapper around the Docker CLI that is used to orchestrate Docker containers.	
	DCHQ (DCHQ) DCHQ is a deployment automation and governance platform for Docker-based application development. The solution provides self-service access to Docker-based applications using an agent-based architecture for orchestration.	» Build/Deploy
	DC/OS (Mesosphere) Mesosphere's DC/OS is a commercial version of the Mesos OS for managing data centers. It supports both Kubernetes and Docker.	
Open Source	**Docker Swarm (Docker)** Docker Swarm offers native clustering for Docker by turning multiple Docker hosts into a single, virtual host.	
Open Source	**Dray (CenturyLink)** Dray is an engine for managing the execution of container-based workflows. It is a Go application that provides a RESTful API for managing jobs and is most commonly used for containers hosting long-running services.	
	Elasticsearch (Elastic) Elasticsearch is a search and analytics engine based on Lucene.	
	Engine Yard (Engine Yard) Engine Yard is a cloud orchestration PaaS for deploying, monitoring and scaling applications.	
Open Source	**Fenzo (Netflix)** Fenzo is an extensible scheduler for Mesos frameworks.	
Open Source	**HTCondor (University of Wisconsin)** HTCondor is a specialized workload management system for compute-intensive jobs. It provides a job queueing mechanism, scheduling policy, priority scheme, resource monitoring, and resource management.	
	IronMQ (Iron.io) IronMQ is a messaging queue that provides scheduling and communication between services and components.	
	IronWorker (Iron.io) IronWorker is a platform that isolates the code and dependencies of individual tasks to be processed on demand in a containerized environment.	

	Product/Project (Company or Supporting Org.)	Also Categorized In:
	Jelastic (Jelastic) Jelastic provides a PaaS and container-based IaaS on a singular platform that includes container orchestration.	» Platforms/PaaS
Open Source	**Kong (Mashape)** Kong is a management layer for APIs. It has the capability of orchestrating Dockerfiles.	
Open Source	**Kontena (Kontena)** Kontena is a container orchestration tool. It abstracts containers into application services and establishes an internal network between linked services, making it easy to deploy and scale applications across multiple hosts.	
Open Source	**Kubernetes (Cloud Native Computing Foundation)** Kubernetes is an open source Docker orchestration tool. Google initially developed Kubernetes to help manage its own LXC containers.	
Open Source	**MaestroNG (SignalFx)** MaestroNG is an orchestrator for Docker-based, multi-host environments sponsored by SignalFx.	
Open Source	**Magnum (OpenStack Foundation)** Magnum is an OpenStack project that offers container orchestration engines for deploying and managing containers as first class resources in OpenStack.	
Open Source	**Marathon (Apache Software Foundation)** Marathon is an Apache Mesos framework for long-running applications. Marathon provides a REST API for starting, stopping and scaling applications. It lets users deploy, run and scale Docker containers.	
	Navops Command (Univa) Navops Command provides automated scheduling combined with policy controls.	
Open Source	**Nomad (HashiCorp)** Nomad is a distributed, highly available, datacenter-aware scheduler.	
	StackEngine (Oracle) StackEngine is an end-to-end container application management system that provides a way for dev and enterprise IT teams to deploy Docker applications.	» Build/Deploy
	Tectonic (CoreOS) Tectonic, which is currently being previewed, will be an enterprise version of Kubernetes.	

	Product/Project (Company or Supporting Org.)	Also Categorized In:
	xDock (Xervmon) Xervmon is a cloud management platform. Its xDock lets users deploy, manage and monitor Docker images in the cloud.	» Build/Deploy
Open Source	**Zenoss Control Center (Zenoss)** Zenoss Control Center is an application management and orchestration system. It works with the Zenoss platform and other Docker applications. Serviced is a popular repository in this project that provides a PaaS runtime.	

ORCHESTRATION: SERVICE DISCOVERY

	Product/Project (Company or Supporting Org.)	Also Categorized In:
Open Source	**confd (N/A)** Confd is a project aimed at allowing dynamic reconfiguration of arbitrary applications based on changes in the service discovery portal. The system involves a tool to watch relevant endpoints for changes, a templating system to build new configuration files based on the information gathered, and the ability to reload affected applications.	
Open Source	**Consul (HashiCorp)** Consul is a tool for service discovery and configuration. Consul is distributed, highly available and extremely scalable.	
Open Source	**ContainerPilot (Joyent)** A service for autodiscovery and configuration of applications running in containers.	
Open Source	**Denominator (Netflix)** Denominator allows uses to portably control DNS clouds using Java or Bash.	
Open Source	**etcd (CoreOS)** etcd is a distributed, consistent, key-value store for shared configuration and service discovery.	
Open Source	**Eureka (Netflix)** Eureka is a REST-based service that is primarily used in the AWS cloud for locating services for the purpose of load balancing and failover of middle-tier servers.	
	Elastic Cloud (Elastic) After Elastic acquired Found, it was implemented into their Elasticsearch as a Service offering called Elastic Cloud. It can be used by the Docker community for search and discovery.	
Open Source	**Nerve (Airbnb)** Nerve is a utility for tracking the status of machines and services. It runs locally on the boxes which make up a distributed system, and reports state information to a distributed key-value store. It is used in conjunction with Synapse.	

	Product/Project (Company or Supporting Org.)	Also Categorized In:
Open Source	**Serf (HashiCorp)** Serf is a decentralized solution for service discovery and orchestration that is lightweight, highly available and fault tolerant.	
	Service Discovery (IBM) Service discovery is a service that enables developers to easily register and find instances of their microservices.	
Open Source	**SkyDNS (N/A)** SkyDNS is a distributed service for announcement and discovery of services built on top of etcd.	
Open Source	**SmartStack (Airbnb)** SmartStack is Airbnb's automated service discovery and registration framework.	
Open Source	**Spring Cloud Config (Pivotal Software)** Spring Cloud Config provides server- and client-side support for externalized configuration in a distributed system.	
Open Source	**Synapse (Airbnb)** Synapse is Airbnb's system for service discovery. Synapse solves the problem of automated failover in the cloud, where failover via network re-configuration is impossible. The end result is the ability to connect internal services together in a scalable, fault-tolerant way.	
Open Source	**Vulcand (N/A)** Vulcand is a programmatic load balancer backed by Etcd.	
	Weave Flux (Weaveworks) Weave Flux provides routing and control for microservices applications.	» Build/Deploy
Open Source	**ZooKeeper (Apache Software Foundation)** ZooKeeper is an open source server that enables highly reliable, distributed coordination.	

ORCHESTRATION: OTHER ORCHESTRATION-RELATED TOOLS

	Product/Project (Company or Supporting Org.)	Also Categorized In:
	Bluemix (IBM)	» Build/Deploy, Platforms/PaaS
	Bluemix is IBM's digital application platform, enabling you to develop, deploy, manage and run cloud applications. With Bluemix you can utilize Docker containers, virtual machines and Cloud Foundry applications using open APIs. It has a catalog of over 150 services, including support for microservices, logging and monitoring and Watson analytics.	
	Calm.io (Calm.io)	» Build/Deploy
	Calm.io is a platform to deploy, manage and maintain distributed applications in private and public cloud environments. It claims to offer full-stack orchestration, run item lifecycle management, and policy-based governance.	
	Cloud 66 (Cloud 66)	» Build/Deploy
	Cloud 66 is an application provisioning and management service that allows you to build Docker stacks from scratch on any public or private cloud vendor or your own infrastructure.	
Open Source	**Docker (Engine) (Docker)**	» Build/Deploy
	Docker Engine is a lightweight runtime and tooling that builds and runs your Docker containers.	
	Docker Cloud (Docker)	» Build/Deploy, Platforms/PaaS
	Docker Cloud is the name of the new cloud service by Docker that expands on the features of Tutum and brings a tighter integration with Docker Hub; it is a SaaS service for deploying and managing Dockerized applications.	
Open Source	**Docker Compose (Docker)**	» Build/Deploy
	Compose is a tool for defining and running multi-container applications with Docker.	
	Docker Datacenter (Docker)	» Build/Deploy, Platforms/PaaS
	Docker Datacenter provides on-premises container management and deployment services to enterprises with a production-ready platform supported by Docker and hosted locally behind the firewall.	
Open Source	**Docker Machine (Docker)**	» Build/Deploy
	Docker Machine lets you create Docker hosts on your computer.	

	Product/Project (Company or Supporting Org.)	Also Categorized In:
Open Source	Dockerize **(Dockerize)** Dockerize simplifies the "Dockerization" of applications by automatically generating configuration files using templates.	
Open Source	Flocker **(ClusterHQ)** Flocker is a data volume manager for Dockerized applications.	
	Microscaling Systems **(Microscaling Systems)** Microscaling Systems scales containers in real time in response to demand.	» Build/Deploy
Open Source	Garden **(Cloud Foundry Foundation)** Garden provides a platform-neutral API for containerization. Backends implement support for various specific platforms.	
	Kismatic **(Kismatic)** Kismatic provides enterprise support and a production platform tooling for Kubernetes and Docker.	
	Service Proxy **(IBM)** Service Proxy is a service that automatically balances workloads across instances to improve performance.	
Open Source	Spring Cloud Data Flow **(Pivotal Software)** The Spring Cloud Data Flow project provides orchestration for data microservices, including both stream and task processing modules.	
	StackPointCloud **(StackPointCloud)** StackPointCloud is a beta service that allows users to easily create, scale and manage Kubernetes clusters of any size at the cloud provider of their choice.	
	XL Deploy **(XebiaLabs)** XL Deploy provides application release automation to standardize complex deployments, speed up development time, and gain visibility into the pipeline.	» Build/Deploy
	XL Release **(XebiaLabs)** XL Release automates, orchestrates and provides visibility into release pipelines. It identifies bottlenecks, reduces errors and lowers the risk of release failures.	» Build/Deploy

PLATFORMS/PaaS (INCLUDING CaaS)

Product/Project (Company or Supporting Org.)	Also Categorized In:
Alauda (Alauda) Alauda is a cloud platform that provides Containers as a Service, cloud hosting and an image registry.	
Apcera Platform (Apcera) Apcera manages access to compute resources across a cluster of servers. By focusing on managing policies across multiple environments, it aims to secure workloads and containers in enterprise production environments.	» Orchestration
Apprenda (Apprenda) Apprenda provides a PaaS for enterprises that supports the hosting of containers.	
Azure Container Service (Microsoft) Azure Container Service simplifies the creation and configuration of a cluster. The default configuration of this cluster includes Docker and Docker Swarm for code portability and Marathon, Chronos and Apache Mesos to ensure scalability.	» Orchestration
Bluemix (IBM) Bluemix is IBM's digital application platform, enabling you to develop, deploy, manage and run cloud applications. With Bluemix you can utilize Docker containers, virtual machines and Cloud Foundry applications using open APIs. It has a catalog of over 150 services, including support for microservices, logging and monitoring and Watson analytics.	» Build/Deploy, Orchestration
Bluemix Dedicated (IBM) Bluemix Dedicated is a fully managed, single tenant cloud service running within an IBM datacenter.	
Bluemix Local (IBM) Bluemix Local is fully managed cloud service, running in a customer's datacenter and on their hardware.	
Built.io Flow (Built.io) Built.io Flow is an Integration Platform-as-a-Service for automating processes, connecting data and devices, and integrating with various services.	

	Product/Project (Company or Supporting Org.)	Also Categorized In:
	Carina (Rackspace) Carina provides a "zero infrastructure" hosted container environment, where users no longer worry about building, managing and updating their container environment.	
	Cloudsoft Application Management Platform (Cloudsoft) Cloudsoft's application management platform, based on the open source Apache Brooklyn project, orchestrates services, platforms and infrastructure, including deployment to containers.	» Orchestration
	ContainerShip (ContainerShip) ContainerShip is a self-hosted container management platform, capable of running on any cloud, and used to manage containers from development to production.	» Build/Deploy
Open Source	**Deis (Engine Yard)** Deis is a lightweight, open source PaaS, built on Docker and CoreOS, that makes it easy to deploy and manage applications on your own servers.	
	DigitalOcean (DigitalOcean) DigitalOcean is an IaaS provider that targets developers. It provides "one-click installs" to deploy Docker.	
	Docker Cloud (Docker) Docker Cloud is the name of the new cloud service by Docker that expands on the features of Tutum and brings a tighter integration with Docker Hub; it is a SaaS service for deploying and managing Dockerized applications.	» Build/Deploy, Orchestration
	Docker Datacenter (Docker) Docker Datacenter provides on-premises container management and deployment services to enterprises with a production-ready platform supported by Docker and hosted locally behind the firewall.	» Build/Deploy, Orchestration
Open Source	**Dokku (Engine Yard)** Dokku is a mini-PaaS that is inspired by Heroku's workflow and is now sponsored through Engine Yard's Deis.	
	EC2 Container Service (Amazon Web Services) Amazon EC2 Container Service helps companies manage clusters of containers on AWS infrastructure.	
Open Source	**fabric8 (Red Hat)** fabric8 is an open source DevOps and integration platform that is built as a set of microservices that run on top of Kubernetes and OpenShift V3. Its continuous delivery is based on Jenkins, Nexus and SonarQube. Its iPaaS integrates with Apache ActiveMQ, Camel and CXF.	» Build/Deploy

	Product/Project (Company or Supporting Org.)	Also Categorized In:
Open Source	**Flynn (Prime Directive, Inc.)** Flynn is a micro-PaaS built on Docker containers and optimized for service-oriented applications.	
	Giant Swarm (Giant Swarm) Giant Swarm is a hosted container solution to build, deploy and manage containerized services.	» Build/Deploy
	Helion (Hewlett-Packard Enterprise) Helion is HPE's hybrid cloud offering that includes a PaaS for developers. HPE recently acquired Stackato from ActiveState and plans to integrate it into future offerings.	
	Heroku Buildpacks (Salesforce) The Heroku PaaS provides "buildpacks" for compiling applications. Customers can pay for managed containers that Heroku calls "Dynos."	
	Hook.io (Hook.io) Hook.io is an open source hosting platform for microservices.	» Build/Deploy
	Jelastic (Jelastic) Jelastic provides a PaaS and container-based IaaS on a singular platform that includes container orchestration.	» Orchestration
	Joyent Triton Elastic Container Service (Joyent) Triton Elastic Container Service allows you to securely deploy and operate containers with bare metal speed on container-native infrastructure; Joyent provides the Triton Elastic Container service as part of its larger IaaS offerings.	
	Kyup Cloud Hosting (Kyup) Kyup provides scalable cloud-based container hosting on Linux containers.	
Open Source	**Mantl (Cisco)** Mantl is an open source platform for rapidly deploying globally distributed services. It works with tools such as Marathon, Mesos, Docker and Consul.	» Build/Deploy
	Nirmata (Nirmata) Nirmata is a cloud-based platform for managing microservices.	
	OpenShift Dedicated (Red Hat) OpenShift Dedicated is the hosted, supported version of the container platform.	

	Product/Project (Company or Supporting Org.)	Also Categorized In:
Open Source	**OpenShift Origin (Red Hat)** OpenShift Origin is the upstream open source version of OpenShift and is meant to allow for development of cloud-native applications. OpenShift is a PaaS built on Docker containers that orchestrates with Kubernetes. It also has Atomic and Red Hat Linux components.	
	Packet (Packet Host) Packet is a bare metal PaaS that supports Docker, CoreOS, Deis, Mesosphere and Rancher.	
	Pivotal Cloud Foundry (Pivotal Software) Pivotal Cloud Foundry is a cloud-native enterprise PaaS based on Cloud Foundry. It offers support for building applications as deployable containers.	
Open Source	**Project Atomic (Red Hat)** Project Atomic hosts run applications in Docker containers with components based on RHEL, Fedora and CentOS. In addition to Atomic Host, the project includes Nulecule, a container-based app spec that enables the use of existing containers as building blocks for new applications.	
	PureApplication (IBM) PureApplication offers solutions that use automated patterns to rapidly deploy applications, reduce cost and complexity, and ease management. Docker containers can be included in patterns along with non-container components to take advantage of the PureApplication enterprise-grade lifecycle management. Support includes a private Docker registry pattern deployable as a shared service.	» Build/Deploy
	Rancher (Rancher Labs) Rancher is a complete infrastructure platform for running containers in production.	
	Scalingo (Scalingo) Scalingo is a PaaS for containers; users push their code to Scalingo and it creates an image and allocates resources to run the application in its cloud.	
	SoftLayer (IBM) SoftLayer provides infrastructure as a service (IaaS) including bare metal and virtual servers, networking, turnkey big data solutions, and private cloud solutions. SoftLayer is supported as a provider behind Docker Machine for quickly standing up a cloud-hosted Docker host.	
	sloppy (sloppy.io) Sloppy provides cloud hosting for containers.	
Open Source	**StackHut (StackHut)** StackHut is a platform to develop and deploy microservices without writing any server-logic.	» Build/Deploy

	Product/Project (Company or Supporting Org.)	Also Categorized In:
Open Source	**Terraform (Hashicorp)**	» Build/Deploy
	Terraform is a tool to build and launch infrastructure, including containers.	
Open Source	**Vamp (Magnetic.io)**	» Build/Deploy
	Vamp stands for Very Awesome Microservices Platform. It helps developers build, deploy and manage microservices. Vamp's core features are a platform-agnostic microservices DSL, powerful A/B testing, canary releasing, autoscaling and an integrated metrics/event engine.	
	Virtuozzo (Odin)	» Build/Deploy
	Virtuozzo is a container virtualization platform sold to providers of cloud services.	
	WaveMaker (WaveMaker)	
	WaveMaker provides a PaaS for development and management of custom enterprise apps on private infrastructure. It supports the running of Docker applications.	

DISCLOSURES

The following companies mentioned in this ebook are sponsors of The New Stack: Arcadia Data, Bitnami, Capital One, Cloud Foundry, Codenvy, CoreOS, DigitalOcean, GoDaddy, HPE, Intel, Iron.io, Joyent, New Relic, Sysdig, Teridion, VMware, Weaveworks.