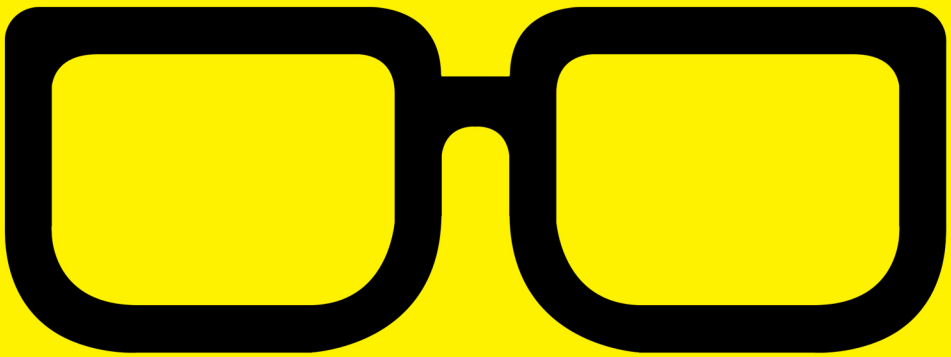
SPONSORED BY

puppet

GEEK GUIDE



Containers 101

LINUX JOURNAL

Table of Contents

SOL LEDERMAN is a technical people-oriented professional with more than thirty years of broad experience in system administration, software design and development, technical support, training, documentation, troubleshooting and customer management. Sol currently divides his time between running IT for a software firm and providing a variety of tech services to the federal government.

GEEK GUIDES:

Mission-critical information for the most technical people on the planet.

About the Sponsor

Puppet

Puppet is driving the movement to a world of unconstrained software change. Its revolutionary platform is the industry standard for automating the delivery and operation of the software that powers everything around us. More than 40,000 companies — including more than 75 percent of the Fortune 100 — use Puppet's open source and commercial solutions to adopt DevOps practices, achieve situational awareness and drive software change with confidence. Headquartered in Portland, Oregon, Puppet is a privately held company with more than 500 employees around the world. Learn more at puppet.com.

Containers 101

SOL LEDERMAN

What Is a Container and How Are They Used?

A starting point for an exploration of containers and how they're used is this simple definition: a container is a packaging format for a unit of software that ships together.

More specifically, a container is a format that encapsulates a set of software and its dependencies, the minimal set of runtime resources an application needs to function. A container is a form of virtualization that is similar to a virtual machine (VM) in some ways and different in others. Like containers, VMs also encapsulate functionality in the form of the application platform and its dependencies. The key difference between VMs and containers is that each VM has its own full-sized OS, while containers typically have a more minimal OS.

An article on CIO.com "What are containers and why do you need them?" (cio.com/article/2924995/enterprise-software/what-are-containers-and-why-do-you-need-them.html) explains the motivation driving container use:

Containers are a solution to the problem of how to reliably deliver applications across a variety of differently-configured environments. This could be from a developer's laptop to a test environment, from a staging environment into production, and perhaps from a physical machine in a data center to a virtual machine in a private or public cloud.

Because they minimize the OS overhead, containers are an order of magnitude smaller than VMs. Containers are lightweight, which gives them major advantages. They start up quickly and deploy easily. Requiring less storage, multiple containers can fit into the disk footprint of a single VM.

Containers, because they don't encapsulate an entire OS and its services, are an order or two of magnitude smaller than VMs.

What Is the Value of Container Adoption?

Containers are particularly useful in rapid development environments. The develop/deploy process cycles through these six steps:

1. Develop or update an application.
2. Deploy the application to the testbed.
3. QA the new code.
4. Move the code to the staging site.
5. Verify operation in the staging environment.
6. Deploy changes to production environments.

These cycles occur frequently and involve multiple parties — developers, QA staff, system administrators, and perhaps DevOps staff.

With a traditional develop/deploy cycle, any bottleneck in provisioning the resources an application needs will delay release. Beyond the concerns of moving software through the lifecycle quickly are the risks introduced by differences in the development, testbed, staging, and production environments. The later in the lifecycle that issues are detected, the more expensive and time-consuming they are to correct. Adoption of containers can help mitigate these issues.

Containers are particularly valuable in rapid development cycles for three reasons. First, containers are much faster to deploy than servers or VMs. Second, smaller footprints make better utilization of server and cloud resources. And third, because containers enclose their running environment including all of their dependencies, they minimize the risk of inconsistencies across environments.

Because their self-contained nature makes them such a convenient delivery method, containers are a key component in automating testing, integration, and deployment processes. Later in this e-book, we explore how containers are a key component of continuous integration/continuous deployment (CI/CD) strategies.

How Are Containers Being Managed?

Though there were many contenders for leading container technology, Docker has emerged as the clear winner. While there are still cases where other containerization systems like LXC are in use, most orchestration tools are built around Docker. In particular, orchestration tools used by cloud providers emphasize Docker.

Containers may be useful on their own, but orchestration is what really puts their capabilities to use. Container orchestration covers the distribution, deployment, provisioning, and monitoring of clusters of containers across underlying hosts. An orchestration tool consists of a number of components, such as service discovery or load balancing, and automates much of the tedium of container management.

Table 1. Utilization Percentages of the Seven Most Popular Container Technologies

Amazon ECS	**44%**
Azure AKS	**43%**
IBM IKS	**40%**
Google GKE	**36%**
Docker Swarm	**15%**
Pure Kubernetes	**10%**

Table 1 shows the results of the Portworx 2018 Container Adoption Survey (portworx.com/wp-content/uploads/2018/12/Portworx-Container-Adoption-Survey-Report-2018.pdf). It lists the adoption rates of the most popular container orchestration tools. Note that Kubernetes is clearly in the lead, whether it's pure open-source Kubernetes or a vendor-specific Kubernetes-based platform such as Azure AKS, Amazon EKS, Google GKE, IBM IKS, or Red Hat OpenShift. Portworx suspects that most of the respondents who answered Amazon Elastic Container Service (ECS) probably meant Amazon EKS instead, further growing the Kubernetes market share. As the leading container orchestration platform, Kubernetes functions as the new "operating system" for cloud-native applications, handling operational concerns such as workload scheduling, log aggregation, scaling, health monitoring, and seamless application upgrades.

Because the container orchestration space continues to see experimentation and growth, there is a possibility that other contenders may become more popular. Both Amazon ECS and Docker Swarm could show promising growth in the future, particularly with Swarm's integration into the Docker core in version 1.12.0. In addition, over 70 percent of respondents to Portworx's survey reported that their organization used more than one container orchestration platform. This means that it's still important to keep an eye on the changing space for container orchestration.

As the leading container orchestration platform, Kubernetes functions as the new "operating system" for cloud-native applications.

Do Companies Need to Deploy All Applications in Containers, or Can There Be Hybrid Approaches?

Using containers is not an all-or-nothing proposition. For some applications, it doesn't make sense to update or refactor to run in containers. The migration from traditional OS-packaged applications to containerized applications can be a planned and gradual one.

Enterprises are faced with a mix of technologies; some run in containers, some in VMs, and some on bare metal. To add to the complexity, some cloud-native applications are running in public clouds, and others are running in private clouds. The New Stack introduces the hybrid cloud and container virtual networking as the new normal in its article "How Overlay Networks Pave the Way for Seamless Hybrid Clouds" (thenewstack.io/containerizing-makes-hybrid-cloud-easier-adopt):

The question then becomes how to weave together all the containers running on and off premises and connect them to other non-containerized services, without this turning into a configuration hairball and security nightmare.

Container overlay networks are essentially the de facto standard now because they avoid the configuration hairball. The container virtual network rides on top of the underlying IP networks, so it looks the same to the application regardless of differences in the underlying network technology. A data center network doesn't work the same way as, say, AWS Virtual Private Cloud. Essentially, the problem of managing the differences is pushed down into the container networking layer. That means there is no configuration or code required in the application itself: It can just use regular networking constructs like TCP/IP and DNS.

Planning a migration path will require major effort, but fortunately, the tools and practices to enable such an effort are evolving rapidly.

What's Involved in Managing Containers?

Managing containers requires leading-edge skills and tools, beyond an understanding of the container paradigm. Digital Ocean published an excellent five-part tutorial series (digitalocean.com/community/tutorials/the-docker-ecosystem-an-introduction-to-common-components) introducing containers, service discovery, distributed configuration, networking, communication, scheduling, and orchestration. Here is a summary of the major components the tutorial introduces.

1. **Service discovery and distributed configuration stores:** Service discovery helps containers deploy and scale without human intervention. Discovery assists in interaction with other containers by finding available services that other containers provide. Distributed configuration storage enables dynamic scaling and configuration of containers without requiring static configuration.
2. **Networking:** Containers need to communicate with each other and with their hosts. With the number of containers scaling up and down, and with the potential for a large number of containers in an application, robust networking service is paramount. Secure communication between application components is another concern. Additionally, the networking service must handle subnetting, gateways, MAC addresses, and other tasks.

3. **Scheduling:** The scheduler needs to be able to determine an appropriate host for an application component and start a container on it. The scheduling service relies on information in the distributed configuration stores when making decisions. The service needs to handle potential constraints about whether to run multiple containers on a particular host, whether to run more than one container on a given host, whether to start the container on the least busy host, and any other constraints the administrator places on the application.
4. **Cluster management:** Cluster management is closely related to scheduling. A particular unit of work may consist of containers, hosts, services, and their interactions. It may be desirable to abstract away the management of that workload. Clusters are the abstraction for the resources and the interactions that are required to perform that work. Cluster management tools can operate on those abstractions.
5. **Orchestration:** Orchestration is often used interchangeably with the terms "scheduling" and "cluster management." Orchestration also involves provisioning, which is the process of creating and configuring a container and starting it so that it may perform work.

An additional major aspect of working with containers is configuration management. This involves making sure that the right versions of OS level as well as application software and libraries are installed and managed (for example, for upgrades). Later in this guide, we introduce container configuration management and some of the complexities that a good set of tools can help manage.

Whereas a well-oiled CI/CD practice may be nice perk for traditional applications that are deployed a few times a month or quarter, cloud-native applications need to deploy on-demand and frequently. In the latter case, CI/CD becomes imperative.

Container Management Beyond Orchestration

Container management goes beyond orchestration. Today, containers are deployed across diverse infrastructures of private data centers and multiple clouds. Modern DevOps environments require orchestration to stretch beyond the bounds of just one isolated Kubernetes-managed cluster.

Kubernetes is great at a lot of things, such as workload scheduling, log aggregation, scaling, health monitoring, and seamless application upgrades. However, when an application is deployed across multiple Kubernetes clusters in multiple data centers, you'll want a tool that automates the application build and deployment lifecycle for all cloud-native and on-prem containerized applications.

Enterprises that are heavily invested in containers can end up using hundreds or even thousands of containers across multiple clusters. This inevitably results in far more containers than one can reasonably keep track of. Discovery is a crucial part both in container management and in migrating applications to container-based deployment. You can learn more about the discovery process and how it's used with containers in this Linux Journal article (linuxjournal.com/content/puppets-cloud-discovery-know-whats-running-your-cloud).

Benefits Gained by Switching to Containers: Case Studies

The Docker website includes several dozen case studies of benefits enterprises gained when they adopted containers (docker.com/customers). Beyond Docker's own press, the following is a sampling of the positive experiences of enterprises migrating to containers:

Splunk: "Splunk leverages Docker Enterprise as the foundation for their CI/CD and test infrastructure, integrating with Jenkins to automate performance tests." (docker.com/customers/splunk)

ADP: "To ensure the utmost security, ADP trusts Docker Enterprise to build an automated and secure software supply chain for their application development process and a uniform operating model for agility." (docker.com/customers/adp)

Kubernetes also includes a number of case studies that enumerate the benefits of container orchestration.

Adform: "Kubernetes enabled the self-healing and immutable infrastructure. We can do faster releases, so our developers are really happy. They can ship our features faster than before, and that makes our clients happier." (kubernetes.io/case-studies/adform/)

Squarespace: "Since Squarespace moved to Kubernetes, in conjunction with modernizing its networking stack, deployment time has been reduced by almost 85%." (kubernetes.io/case-studies/squarespace/)

Wikimedia Foundation: "Wikimedia Tool Labs has seen great success with the initial Kubernetes deployment. Old code is being simplified and eliminated, contributing developers don't have to change the way they write their tools and bots, and those tools and bots run in a more stable fashion than they have in the past." (kubernetes.io/case-studies/wikimedia/)

How to Simplify Continuous Delivery for Containers

While containers have reshaped the way we think about delivering software, to get the most value out of them, it is vital to adopt a continuous delivery (CD) paradigm. Whereas a well-oiled CI/CD practice may be nice perk for traditional applications that are deployed a few times a month or quarter, cloud-native applications need to deploy on-demand and frequently. In the latter case, CI/CD becomes imperative.

CD automates the build/test cycle for every code check-in, and deploys successful changes to the container orchestration system.

With CD, a development team is able to shorten the time between releases through powerful automation that affects every stage of the software development lifecycle. Since containers are self-contained images, there is no issue with package dependencies or unexpected necessary upgrades. In fact, CD leverages containers in such a way that software releases don't rely on traditional package management at all. With containers, your application just needs to be built once to run on all of your target environments.

CD covers a wide variety of software release aspects, from configuration management to automated testing to environment management. The CD process is often paired with continuous integration (CI) – you will see the two practices referred to together in this paper and elsewhere as "CI/CD." Together, they enable smaller, more frequent releases with fewer bugs and reduced deployment risk.

Automating the entire application build and deployment process in order to take advantage of CD can be a challenge. In order for all of the benefits to be available, it's important that your CD process and tooling is built for cloud-native applications. It needs to support self-service deployments without requiring significant custom development work, as is often the case with traditional CI tools.

Conclusion

Containers have become the leading technology for managing enterprise-class applications. Success in building applications or adopting and scaling today's DevOps practices relies on understanding how containers operate and moving to a continuous delivery approach to coding and releasing software. While the cloud-native space may still be rapidly evolving, now is the time to be thinking about how you can realize the benefits that come with adopting cloud-native application architectures.