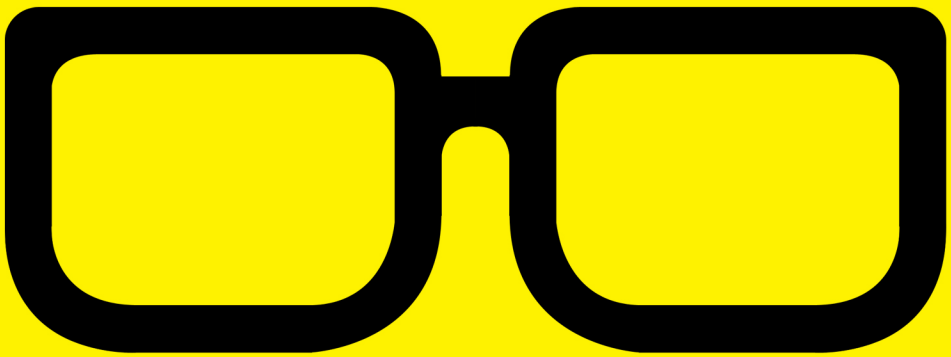
SPONSORED BY

puppet

GEEK GUIDE



Containers 101

LINUX JOURNAL

Table of Contents

What Is a Container and How are Containers Used? 5

What Are the Values of Containers? 6

Who Are the Container Providers? 8

Do Companies Need to Leave the VM Structure Entirely, or Can There Be Hybrid Approaches? 10

Why Are Some Firms Waiting to Use Containers? 11

What's Involved in Managing Containers? 14

Who Are Some of the Major Players in the Container Runtime Space? 16

Benefits Gained by Switching to Containers—Case Studies 18

How Does Configuration Management Apply to Containers, and How Does Puppet Accelerate the Adoption of Container Technologies? 19

Conclusion 25

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 33,000 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. Based in Portland, Oregon, Puppet is a privately held company with more than 470 employees around the world. Learn more at http://puppet.com.

Containers 101

SOL LEDERMAN

What Is a Container and How Are Containers 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.*

A container is a format that encapsulates a set of software and its dependencies, the minimal set of runtime resources the software needs to do its function. A container is a form of virtualization that is similar to a virtual machine (VM) in some ways and different in others. VMs 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.

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

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

> Containers are a solution to the problem of how to get software to run reliably when moved from one computing environment to another. 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.

Containers, because they don't encapsulate an entire OS and its services, are an order or two of magnitude smaller than VMs. Because they're lightweight and have a minimal OS component, containers have some major advantages. They start up quickly and move easily from one platform to another compatible one, and a number of containers can fit into the disk footprint of a single VM.

What Are the Values of Containers?

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 new code.

4. Move code to the staging site.

5. Verify operation in the staging environment.

6. Move the new code to production.

These cycles occur frequently and involve multiple parties (developers, QA staff, system administrators and perhaps DevOps staff), and any bottlenecks in provisioning the resources an application needs will delay releases. Beyond the concerns of moving software through the life cycle quickly are the risks introduced by differences in the development, testbed, staging and production environments. Of particular concern are bugs introduced into production caused by differences in environments. The later in the life cycle bugs are caught, the more expensive and time consuming they are to correct.

Containers are particularly valuable in rapid development cycles for three reasons. First, containers are much quicker to spin-up than servers or VMs. Second, smaller footprints make better utilization

of server hardware than VMs. And third, because containers enclose their running environment including all of their dependencies, containers greatly minimize the risks of inconsistencies in environments. Later in this ebook, I discuss the importance of carefully managing the complexities of a container environment to speed deployments while minimizing risks.

Who Are the Container Providers?

Table 1 shows the results of ClusterHQ's "2016 Container Market Adoption Survey" (https://clusterhq.com/assets/pdfs/state-of-container-usage-june-2016.pdf), which lists utilization percentages of the seven most popular container technologies according to survey respondents (note that some enterprises use more than one container technology, so the percent total is greater than 100%).

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

Docker	**94%**
LXC	**15%**
rkt	**10%**
FreeBSD Jails	**5%**
Solaris Zones	**5%**
Other	**5%**
LXD	**4%**

Docker is by far the most widely adopted container technology. The ConvoxBlog explains why in its post "Why Docker? The Image API is Everywhere" (https://convox.com/blog/why-docker):

> The reason to use Docker is for its modern packaging and runtime APIs. The Docker Image and Container APIs have become a de facto standard. Every major computing platform—from OS X to AWS—now has native support for working with Docker Images and Containers.

Docker is also popular because it is tightly integrated with the Linux kernel and runs on all major Linux distributions. Less adopted FreeBSD Jails and Solaris Zones, in contrast, are built on less popular OSes. And, Docker has major community involvement through its cloud-based registry, testing and collaboration service, Docker Hub (https://hub.docker.com).

Because the container space continues to see much experimentation and growth, there are a large number of container technology providers, with a large matrix of overlapping features. No single document could provide a fair comparison of the many offerings. However, Wikipedia has an article, "Operating-system-level virtualization", with a table of 17 implementations (as of September 2016) showing the OS, license and feature information for each (https://en.wikipedia.org/wiki/Operating-system-level_virtualization). This table is an excellent jumping off point for comparing offerings at a very high level.

A bigger question than whether containers and VMs can coexist is how to network them together.

Do Companies Need to Leave the VM Structure Entirely, or Can There Be Hybrid Approaches?

Using containers is not an all-or-nothing proposition for enterprises with a heavy investment in VMs. Plus, it doesn't make sense to update or refactor some applications to run in containers. The migration from VM-centric applications to containerized applications can be a planned and gradual one.

A bigger question than whether containers and VMs can coexist is how to network them together. Enterprises are faced with a mix of technologies, some running in containers, some in VMs and some on bare metal. To add to the complexity, some 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" (http://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 (uncontained) 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.

Why Are Some Firms Waiting to Use Containers?

Enterprises commonly cite one or more of the following six reasons for not rushing to containers:

1. Not all applications fit into the container model. Containers are particularly well suited to tasks that are stateless or can be carved into small units of functionality (that is, microservices) and that benefit from scaling specific tasks rather than the entire application. On the other hand, you will need to consider how to deal with data persistence and storage. There may be no reason to move monolithic applications that run fine in VMs to containers.

2. Security is a major concern because containers share a common OS kernel. A virus that infects one container places other containers that share that kernel at risk. Although SELinux can help mitigate risk, many enterprises are reluctant to deploy containers in production environments. VMs are less susceptible to spreading vulnerabilities, because they are more isolated from the hypervisor, and each runs its own OS. Additionally, the landscape of hardware virtualization is much more stable than that of containers.

3. The container ecosystem is evolving rapidly and may not be ready for production in your organization. Developer-driven container usage can be at odds with operational, security and compliance concerns. While it can be easier to build a container than a VM, you will lose visibility into and control over what's running in your infrastructure. This is problematic if you need to remediate a wide-ranging vulnerability. Some companies are nervous about investing in managing the complexity of containers if they believe they'll need to train their staff in new tools and processes in the next year or two. Related concerns are how to handle changing specifications, backward compatibility and interoperability.

4. Container management, configuration and orchestration is complex. Companies may find it difficult to justify the expense and effort of training their IT personnel (or of hiring additional staff) to

The challenges of provisioning large numbers of containers running different services, of scaling sets of containers up or down, and of managing the interaction among services can be daunting without really solid and proven technologies and workflows.

migrate to containers. The concern is heightened if management has invested heavily in VM environments, especially if they have done so recently.

5. Related to the complexity of managing containers is the concern that management tools have not matured. In a particular deployment there may be hundreds of containers, each running a single service. The challenges of provisioning large numbers of containers running different services, of scaling sets of containers up or down, and of managing the interaction among services can be daunting without really solid and proven technologies and workflows.

6. Windows is the workstation platform of many developers, and Docker for Windows was just recently released. On June 8, 2016, Microsoft announced, "You can now use Docker natively on Windows 10 with Hyper-V Containers, to build, ship and run containers utilizing

the Windows Server 2016 Technical Preview 5 Nano Server container OS image" (https://blogs.windows.com/windowsexperience/2016/06/08/announcing-windows-10-insider-preview-build-14361). It will take time for this offering to prove itself. And, requiring a VM to run on the Windows Server may hinder its adoption. A related nascent area is running Windows in a container (https://msdn.microsoft.com/en-us/virtualization/windowscontainers/about/about_overview).

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 that introduces containers, service discovery, distributed configuration, networking, communication, scheduling and orchestration (https://www.digitalocean.com/community/tutorials/the-docker-ecosystem-an-introduction-to-common-components). Although the tutorial is focused on Docker, it provides a good overview of container technology for users of any container platform. Here is a brief summary of the major components the tutorial introduces:

1. **Service discovery and distributed configuration stores:** service discovery helps containers to scale without human intervention. Discovery assists in interaction with other containers by finding available services that your container provides. Distributed

configuration storage makes possible the dynamic scaling and configuration of containers without requiring the containers themselves to be dependent on some static configuration.

2. **Networking:** containers need to communicate with one another and with their host servers or VMs. 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. And, of course, the networking tools need to provide all of those services in a dynamic environment.

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 in making its 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 a broad term that 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). I dedicate a later section of this guide to introducing container configuration management and some of the complexities that a good set of tools can help manage.

Who Are Some of the Major Players in the Container Runtime Space?

If you want to run containers, you have to run them on a single machine, VM or computer cluster. Note that you can run containers with just Docker on a single machine. Swarm and the rest are clustered solutions.

The three offerings shown in Table 2 are a good starting place for research.

Table 2. Major Players in the Container Runtime Space

Offering	Website	Description from Website
Docker Swarm	https://docs.docker.com/swarm/overview	"Docker Swarm is native clustering for Docker. It turns a pool of Docker hosts into a single, virtual Docker host."
Kubernetes	http://kubernetes.io	"Kubernetes builds upon 15 years of experience of running production workloads at Google, combined with best-of-breed ideas and practices from the community."
Mesos	http://mesos.apache.org	"Native support for launching containers with Docker and AppC images."

Benefits Gained by Switching to Containers—Case Studies

The Docker website includes several dozen case studies of benefits enterprises gained when they adopted containers (https://www.docker.com/customers).

Beyond Docker's own press, the following is a sampling of the positive experiences of enterprises migrating to containers:

- **Uber:** "While the transition was painful, the end result was what they had hoped for, getting rid of their three greatest pain points that stifled continuous deployment. With Docker, they no longer had to wait for the infrastructure team to write service scaffolding, wait for IT to locate services or wait for infrastructure team to provision services" (http://thenewstack.io/docker-helped-turbocharge-ubers-deployments).

- **eBay:** "The adoption of Kubernetes at eBay is not just about moving to containers to deploy applications, but changing the application lifecycle at the company, which is centered around the infrastructure cloud layer, with provisioning, deploying, monitoring, and remediating issues being the key functions for developers and system administrators to perform" (http://www.nextplatform.com/2015/11/12/inside-ebays-shift-to-kubernetes-and-containers-atop-openstack).

- **Yelp:** "[Docker] provides the developers with more of

the ability to do more of the management of the systems themselves", Sam Eaton, Yelp Director of Operations says. "It makes it easier for them to manage and be responsible for all their own services, without having to ask for operational help" (http://thenewstack.io/docker-helped-yelp-leave-monolith-behind).

- **ADP:** "[ADP] was an early tester of the Docker Datacenter stack and now has it running across 762 server nodes on top of its OpenStack cloud and will also be using the stack to provide a compatibility layer running on the AWS cloud" (http://www.nextplatform.com/2016/02/23/docker-trickles-down-from-hyperscale-to-enterprise).

- **Goldman Sachs:** "When our engineers discovered and started using Docker's open source platform, they were immediately impressed by the portability it provides applications", Duet (a 27-year veteran of the firm) said. "It inspired us to move towards a standardized infrastructure for packaging, shipping and running our applications based on Docker's technology" (http://www.cnbc.com/2015/04/14/goldman-sachs-invests-95-million-in-docker.html).

How Does Configuration Management Apply to Containers, and How Does Puppet Accelerate the Adoption of Container Technologies?

Configuration management tools provide automation to handle the complexities of deploying, managing and upgrading infrastructure software. Sophisticated

By managing an enterprise-wide configuration from a central place, the administrator can quickly update applications, system patches, libraries and other resources.

configuration managers allow IT staff to define the desired state of infrastructure and applications while the manager enforces that state. By managing an enterprise-wide configuration from a central place, the administrator can quickly update applications, system patches, libraries and other resources. The configuration manager handles determining which hosts get which updates, relieving the administrator of the onerous and error-prone task of managing the details. Configuration managers also enforce consistency across environments, paving the way for rapid and continuous software deployment.

Although some dismiss the importance of configuration management tools in a container environment, claiming that container management systems often provide configuration management capabilities and making separate tools redundant, they miss another view. Luke Kanies, Founder and CEO of Puppet, one of the leading providers of configuration management tools, articulates a different perspective (http://searchitoperations.techtarget.com/news/450296682/Configuration-management-tools-seek-foothold-in-containers):

> [While] virtualization made each individual machine

> less necessary and eliminated many of the difficult problems involving managing physical machines, it also increased the number of machines under management about tenfold, Kanies said. Meanwhile, Docker is going to make everybody's infrastructure at least another 10 times bigger. Some people argue IT will have as much as 100 times as many containers as it has VMs to manage—and potentially even more.
>
> So, every application you have just got more complex, more critical, more confusing and more complicated. You need way more management, not way less management.

This excerpt from the Puppet web page introducing its Docker integration lists a number of the many moving parts to manage in a container environment (https://puppet.com/product/managed-technology/docker):

> Puppet Application Orchestration allows you to model the relationships between application services—for example, databases, API servers, and message queues. Relationships can be modeled between any mix of containers, microservices, persistent infrastructure, monoliths, devices, or whatever else makes up your application's architecture. With a model to reference, it's easier to understand what to re-architect and where you need to re-architect to incorporate containers and microservices.

Gareth Rushgrove, senior software engineer at Puppet,

makes the case for an even greater need for configuration management of containers even when there is support in powerful management tools, such as Google's Kubernetes container manager (https://puppet.com/blog/managing-kubernetes-configuration-puppet):

> Some people look at configuration management (and tools like Puppet) as a way of managing host-bound resources like files, services, packages, users or groups. Kubernetes introduces higher-level primitives, like Pods and Replication Controllers, aimed at making the management of distributed and scalable systems drastically easier. The story goes that you no longer need configuration management with those new primitives.
>
> ...
>
> The problems associated with those capabilities are present with systems like Kubernetes too, and only partially addressed by current native tooling—problems like managing configuration drift, having a single well-audited change control mechanism, having a model of your infrastructure outside Kubernetes, etc. This becomes even more important as deployments hit production, as well as in heterogeneous (read real-world) environments, where multiple generations of technology run side by side.

Beyond the complexity of managing the relationship between application services is the complexity of working with leading-edge technologies like Docker,

Additionally, Puppet provides tools, in the form of Docker images, that allow Puppet to run on hosts that run Linux containers and on top of container managers.

Kubernetes and Mesos. Puppet has launched Project Blueshift (https://puppet.com/product/managed-technology/blueshift) as the vehicle to facilitate engagement with Puppet's user and technology provider communities.

Docker support includes Puppet's Docker module to install, configure and manage Docker plus its host and the services running on that host (https://forge.puppet.com/puppetlabs/docker_platform/readme). Additionally, Puppet provides tools, in the form of Docker images, that allow Puppet to run on hosts that run Linux containers and on top of container managers. These tools also facilitate the creation of a local Puppet development environment (https://puppet.com/blog/puppet-docker-running-puppet-container-centric-infrastructure). And, when Docker announced general availability of Docker Universal Control Plane, a tool to deploy and manage dockerized applications, Puppet immediately announced its corresponding docker_ucp module (https://puppet.com/blog/install-docker-universal-control-plane-puppet).

Puppet can create resources in Google's Kubernetes container manager via its Kubernetes module (https://forge.puppet.com/garethr/kubernetes/readme).

Per Puppet's announcement, the module:

> ...allows you to use the Puppet domain specific language to manage resources in Kubernetes—for instance Replication Controllers, Services and Pods. This means: (1) It is easier to manage the state of Kubernetes resources over time, using source code that can be versioned alongside your application code. (2) You can be sure of the state of your Kubernetes infrastructure by taking advantage of Puppet's built-in reporting and tools like PuppetDB. (3) If you're already using Puppet, the Kubernetes Puppet module provides a convenient way of managing Kubernetes alongside your other infrastructure.

And, Puppet users can create higher level abstractions for Kubernetes (https://puppet.com/blog/building-your-own-abstractions-for-kubernetes-puppet).

The Puppet community has developed modules for installing and managing Apache Mesos, open-source software that allows enterprises to abstract away system resources in order to build application-centric elastic distributed systems. Installing Mesos on a compute cluster, installing several of the most popular Mesos frameworks, and using Mesos with Puppet are introduced in the "Using Puppet with Mesos" article (https://puppet.com/blog/using-puppet-mesos). This article also includes a number of links to modules and examples.

Project Blueshift also includes modules to work with the open-source CoreOS Linux distribution, which includes the rkt container engine, and the project also distributes

modules to work with Consul, the open-source tool for discovering services on networks. As new container technologies are developed and prove themselves, Project Blueshift will develop modules, relationships with providers and community support to accelerate their integration into the enterprise.

Conclusion

I'd like to close this guide with a second quote from Gareth Rushgrove, senior software engineer at Puppet. This statement focuses, at the surface, on Mesos and Puppet. But, it speaks to the heart of Puppet's priority of fostering community. One could argue that Docker is wildly successful because it combines great technology with a strong emphasis in community. Puppet is following this same recipe for success:

> This collection of modules for managing Mesos with Puppet is another great example of the ingenuity of the Puppet community. Thank you to everyone who has contributed to these modules and to making Puppet a useful tool for managing Mesos. All of this is also a good example of how Puppet is often used to help people adopt newer technologies, bringing a consistency and confidence to how new software is installed, configured and managed. If you have other examples of great Puppet and Mesos integrations, or if you're happily using both together, please do let us know.■