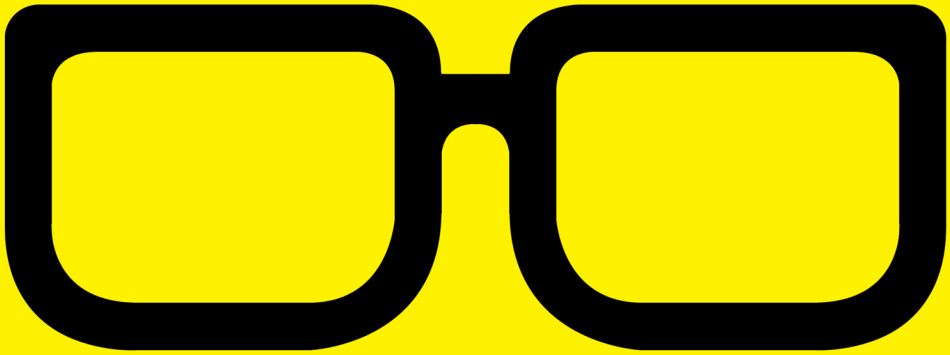
SPONSORED BY

GEEK GUIDE



Tame the Docker Life Cycle with SUSE

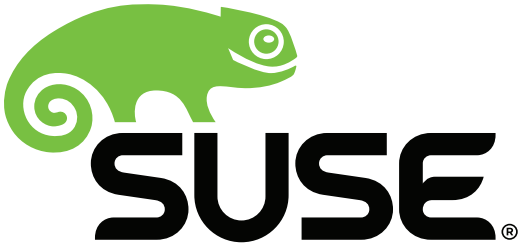
LINUX JOURNAL

Table of Contents

JOHN S. TONELLO is the Director of IT and Communications Manager for NYSERNet, New York's regional optical networking company, serving the state's colleges, universities and research centers. He's been a Linux user and enthusiast since building his first Slackware system from diskette more than 20 years ago.

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

About the Sponsor



SUSE®, a Micro Focus company, provides and supports enterprise-grade Linux and open-source solutions with exceptional service, value and flexibility. With partners and communities, we innovate, adapt and deliver secure Linux, cloud infrastructure and storage software to create solutions for mixed enterprise IT environments. We help customers harness the benefits and power of an open enterprise that can empower their possibilities.

Tame the Docker Life Cycle with SUSE

JOHN S. TONELLO

It's no accident or mere passing fad that containers are revolutionizing how IT shops of all sizes do their work. Whether you're looking to make better use of existing data-center resources or improve portability to the cloud, Docker and the new-found freedom it offers to use virtual environments for everything from development to enterprise applications holds a lot of promise.

The challenge is figuring out how best to move beyond a standard Docker install to an enterprise-worthy solution

that's secure, easy to manage and scalable. It's also important to find ways to manage all your containers easily as well as the images you modify and plan to reuse. After all, containers are only part of any enterprise, which is now a healthy mix of bare-metal boxes, virtual machines, containers and on- and off-premises clouds. Tools that can help provide a common framework—and familiar interfaces—are critical.

With SUSE Enterprise Linux Server 12 and the tools it offers, you and your team can begin to solve real-world problems, tame the Docker life cycle, and create, run and maintain containers at nearly any scale.

The Container Revolution

Anyone managing hardware—from a few blades to full data centers—knows that bare-metal server deployments are costly, time-consuming and not very efficient. Even if you could still afford it, the idea of running one or two services on a single physical server—maybe a database here, a website there—is just not practical. Even if you're the best system administrator out there, you can really make only educated guesses about the maximum amount of CPU, memory and storage a particular service will need over time. Once you do the math and purchase the hardware, you know there surely will be hours, days and weeks when your physical server's capacity is idle and of no use to you.

Virtual machines changed all that by enabling more efficient use of that same physical server's resources by sharing them across separate instances of Linux

and Windows servers. With the advent of VMware and Hyper-V and open-source KVM and Xen, suddenly you could place multiple servers on a single physical box, quickly move them between clusters, more easily run backups and restores, clone them and manage them all from a single interface.

Even with the dramatic resource savings VMs offer, it's clear that many services still don't need all the overhead of their own operating systems. In these cases, the service is what's critical, not the OS, so why burn so much costly cloud CPU, memory and storage capacity on operating system overhead?

Fortunately, lots of smart people asked that same question and came up with the container concept, first releasing Linux Containers (LXC) in 2008. The company that eventually would become Docker initially used the LXC technology as the base of its Platform-as-a-Service offering a year later, and in early 2013, it released the open-source Docker platform that's in widespread use today.

Since then, containers have become widely adopted, the breakthrough coming with Docker's simplicity, which gives just about anyone the ability to deploy containers quickly and easily from the Docker Hub. Docker created the de facto platform-agnostic standard and bolstered it with an ever-growing community-driven public container repository.

The Dockerized World

Unlike virtual machines, containers create virtual environments, which share the host's underlying kernel, instead of replicating it, while remaining fully isolated

For services that quickly need to scale, such as MySQL and nginx, a Docker instance running on a single Linux VM can host dozens of containerized services, acquiring CPU, memory, networking and storage from the host, but using only sips rather than gulps of those critical resources.

from each other and the host system. Each container is based on an image that contains the application you want to run and all its dependencies. For services that quickly need to scale, such as MySQL and nginx, a Docker instance running on a single Linux VM can host dozens of containerized services, acquiring CPU, memory, networking and storage from the host, but using only sips rather than gulps of those critical resources.

Ideally, the containers live not just on a single VM or bare-metal server, but in a clustered environment, drawing the resources they need on the fly from all the available CPU, memory, storage and network. Instead of allocating a couple CPU cores here and this much memory there, containers take what they need from the cluster.

Of course, containers have limitations that make them good additions to, not replacements for, your VMs. Perhaps key among these is the need to run Docker on a VM or

bare-metal host. Docker containers can't yet be spun up natively, but they are stateless and can be spun up quickly on other servers and platforms.

The other drawback for anyone looking to deploy containers in an enterprise is the nature of the public repositories. The Docker Hub contains more than 14,000 official and public images. Official images are offered by SUSE, MariaDB and other companies that make the software. Public images are submitted by "unofficial" developers.

Since security is critical to any enterprise, some may be uncomfortable not knowing what's inside public containers. It works the other way too. If you have any proprietary or custom images you don't want to share in the cloud, you can't fully rely on the public Docker Hub for all your needs. For that, you'll need to deploy your own local repository.

Fortunately, companies like SUSE and its partners have created integrated solutions that allow you to take full advantage of Docker containers and minimize the hassles that go with them.

Deploying Containers and Images with SUSE

The popularity of Docker's container solution has spread across all platforms, which means you have a lot of choices. That's good and bad. Yes, you have the freedom to deploy Docker containers anywhere (making them highly portable) and on any freely available Linux distro, but the sometimes quirky, CLI-heavy nature of deploying containers makes most newcomers crave integrated tools, preferably some with graphical interfaces.

SUSE is tackling that issue with Linux Enterprise Server 12, which uses the popular YaST tool for managing many of the services you need. It has tools that make it a snap to customize your OS environment and roll it up into a bootable .iso, OEM image, VM template or container.

To make it easier to get started with Docker on SLES12, SUSE offers a handy Quick Start guide, which shows how to prepare your server, install Docker, configure the service, create and install SUSE images, and deploy your own private—and secure—container registry with Portus, an open-source browser-based registry management tool.

I used the guide and other public resources to run the

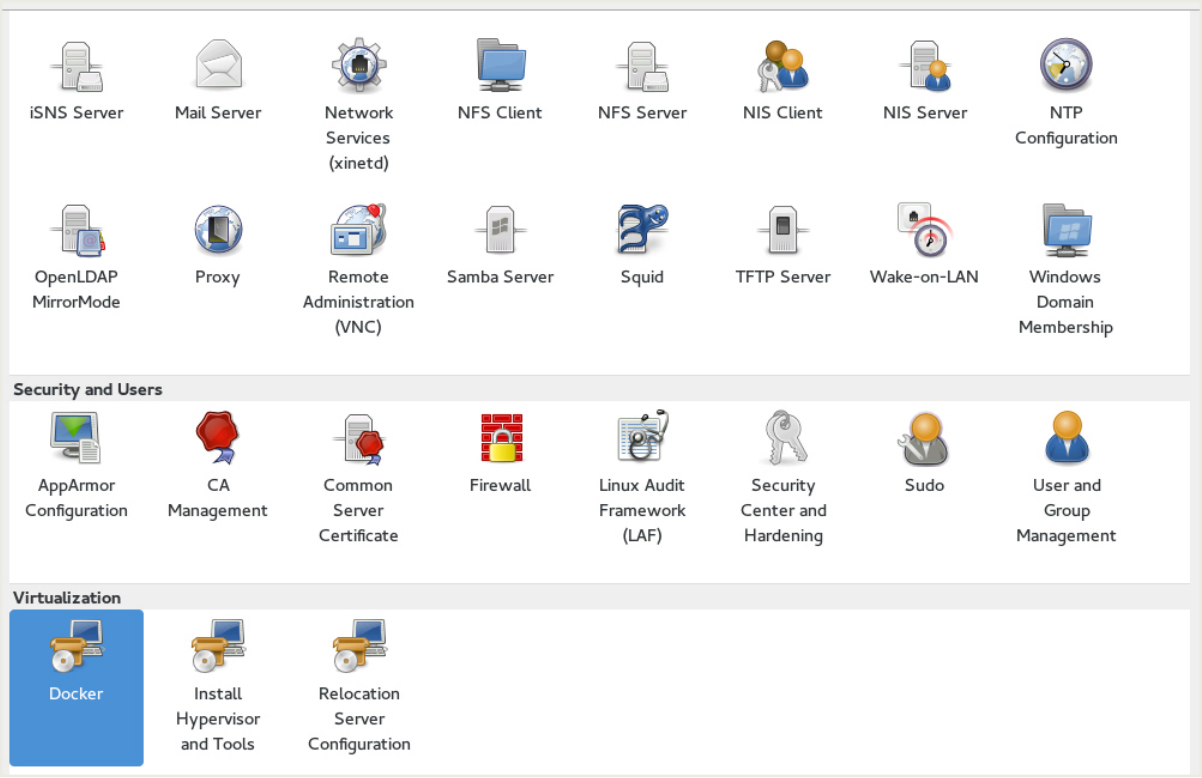


FIGURE 1. This is a view of YaST under SUSE Enterprise Linux Server 12, including the yast2-docker module.

SUSE container solution through its paces. I signed up for a free 60-day SUSE software trial and, for testing purposes, ran a full version of SUSE Linux Enterprise Server 12 SP1. The Disk 1 .iso file is 2.95GB, expanding to about 5GB as a full-functioning VM. I also deployed SUSE's JeOS VM version, which was a mere 267MB.

SUSE's 60-day trial gives you access to all the repositories for the Docker deployment—and everything else the SLES12 server has to offer. Since SUSE is RPM-based, you can use zypper and YaST's Software Management module to handle installs.

I started by deploying the server VM and adding the Container Module to my new system's software repositories. That added the Docker package sources so I could install Docker with a simple:

```
$ sudo zypper install docker
```

I added my system user name to the docker group (so I could run it under my personal login) and started the service. All the dependencies were handled by the Container Module, so it was easy to have Docker up and running in moments.

Of course, the real fun begins when you start pulling down and running container images. If you've never done this before, it's worth taking some time to explore the Docker Hub and run some apps. With the built-in isolation, you never have to worry about messing up your SUSE host install when you run containers. You even can run multiple versions of applications side by side, which makes Docker a great way to test-drive all sorts of goodies.

For most, the speed at which containers boot is the first surprise. I ran the standard "hello-world" container to verify my environment and then pulled down MySQL, which took just seconds to launch. The speed of the first run of any container depends on your network speed, but once a container is downloaded, all future boots are local and they're very fast. I started the full SLES12 container in about two seconds.

Pulling and running containers from the command line is a snap, and SUSE offers a graphical module for YaST called yast2-docker that shows you all of your Docker images and containers, and gives you ways to manipulate them. For instance, my running MySQL container appeared in

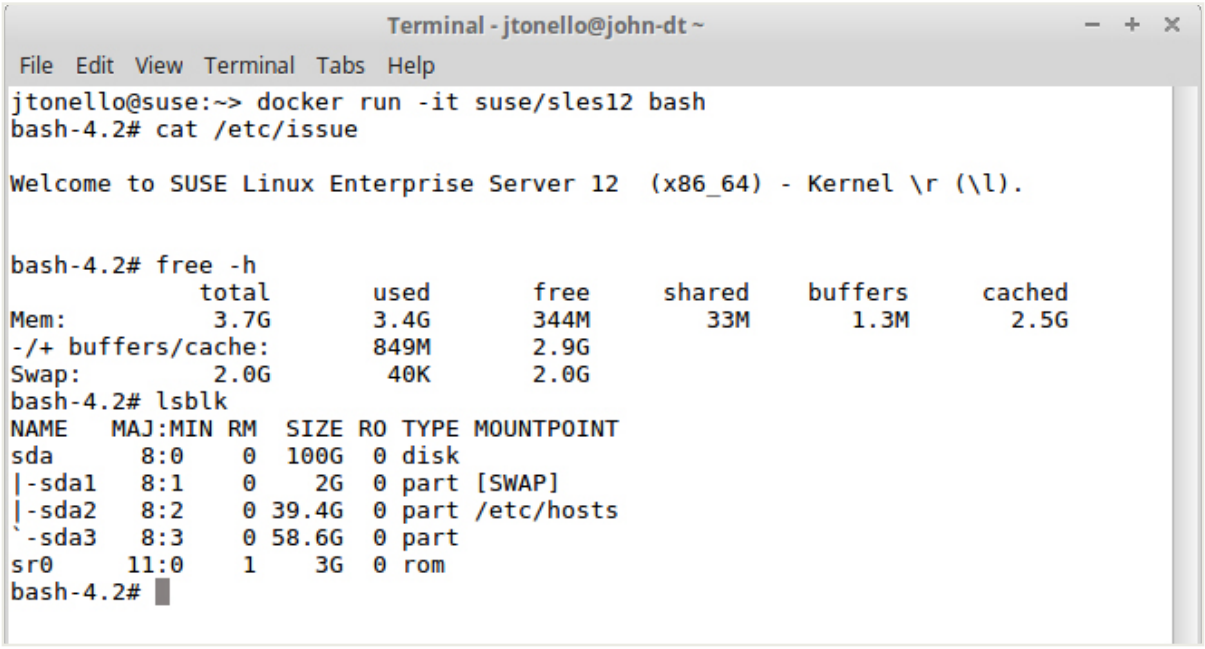


FIGURE 2. Running a simple `docker run` command starts up SLES12 and drops to a bash shell. Running a couple common Linux commands shows that the container is, indeed, SLES12 and that it has full access to the host VM's memory and disks.

yast2-docker's Images and Containers inventory tabs, which graphically replicate the function of typing `docker images` and `docker ps` on the command line.

More valuable, perhaps, is yast2-docker's ability to inject a terminal graphically into any running container or commit a container to a public or private Docker registry.

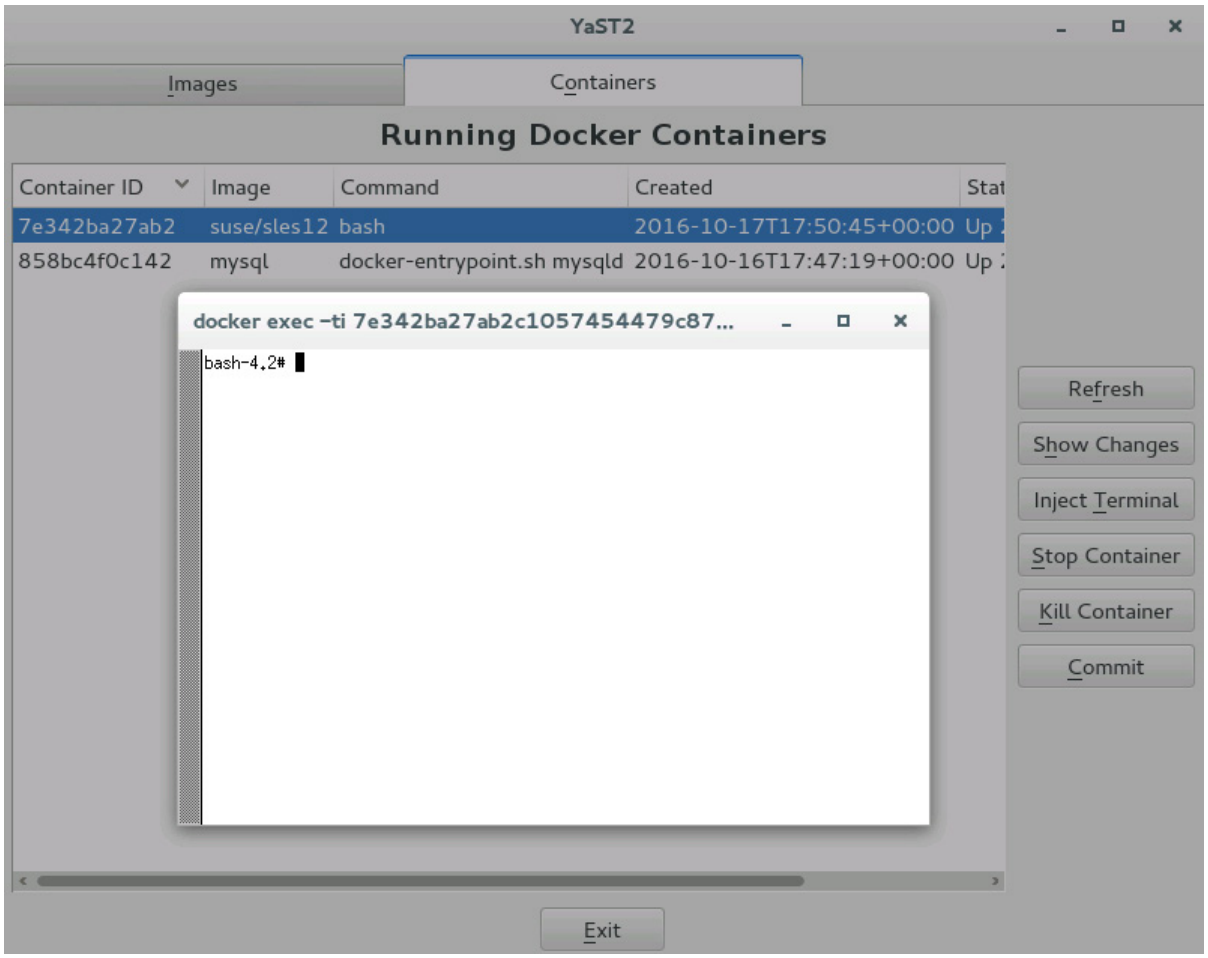


FIGURE 3. SUSE Enterprise Linux 12 uses YaST for most administrative tasks, and the yast2-docker module offers a graphical way to manage containers, including injecting a terminal to anything that's running.

Create a Local Repository with Portus

Your newly created custom images are nice, but they're not much use if you can't share them beyond a single local repository. For that, you need to publish them to a shared repository. By default, all images pushed to the Docker Hub are public, and if you're a commercial user, you have to pay a fee to make more than one of them private.

It's not just an issue of cost. For some, pushing assets to a cloud service they don't own or control, like Docker Hub, is against company security policies.

You can get past those limitations by deploying your own on-premises Docker repository that will allow you to push and pull container images and still update and maintain them with little hassle.

Installing a local registry is simple enough. Under SUSE, you can use `zypper` to install `docker-distribution-registry`. Enable it to run at boot time, and you've got an on-premises place to store and share everything you and your team create:

```
$ sudo zypper install docker-distribution-registry
```

Configuration is done via the /etc/registry/config.yml file, which enables you to set the root directory, the port on which the registry runs, certificates, logging and the like. A simple example from Portus shows the basics (Figure 4).

Leaving it there presents a couple problems though. By default, this local Docker registry lacks any form of authentication, which means anyone with access to it

```
version: 0.1
loglevel: debug
storage:
  filesystem:
    rootdirectory: /var/lib/docker-registry
  delete:
    enabled: true
http:
  addr: :5000
  tls:
    certificate: /etc/nginx/ssl/my.registry.crt
    key: /etc/nginx/ssl/my.registry.key
auth:
  token:
    realm: https://my.portus/v2/token
    service: my.registry:5000
    issuer: my.portus
    rootcertbundle: /etc/nginx/ssl/my.registry.crt
notifications:
  endpoints:
    - name: portus
      url: https://my.portus/v2/webhooks/events
      timeout: 500ms
      threshold: 5
      backoff: 1s
```

FIGURE 4. A basic configuration file for `docker-distribution-registry`, which serves as a local Docker repository manageable by Portus.

can push and pull—and even overwrite—images stored there. At the same time, it's very difficult to keep track of the images you push to a local Docker registry. It lacks all the graphical trappings of docker.io and the Docker Hub, including searching and other functions that make sharing and collaboration easy.

You can solve those problems with Portus, an open-source authentication and graphical interface for Docker registry

created by SUSE. Installation is straightforward, because its packages are included in the same SUSE Containers Repository used to install Docker:

```
$ sudo zypper in portus
```

It's important to note that Portus is using docker-registry behind the scenes, not adding an additional registry to your system.

Your on-premises Docker registry can run without certificates (and SSL), but best practice for a production system is to secure it fully. However, during testing and development,

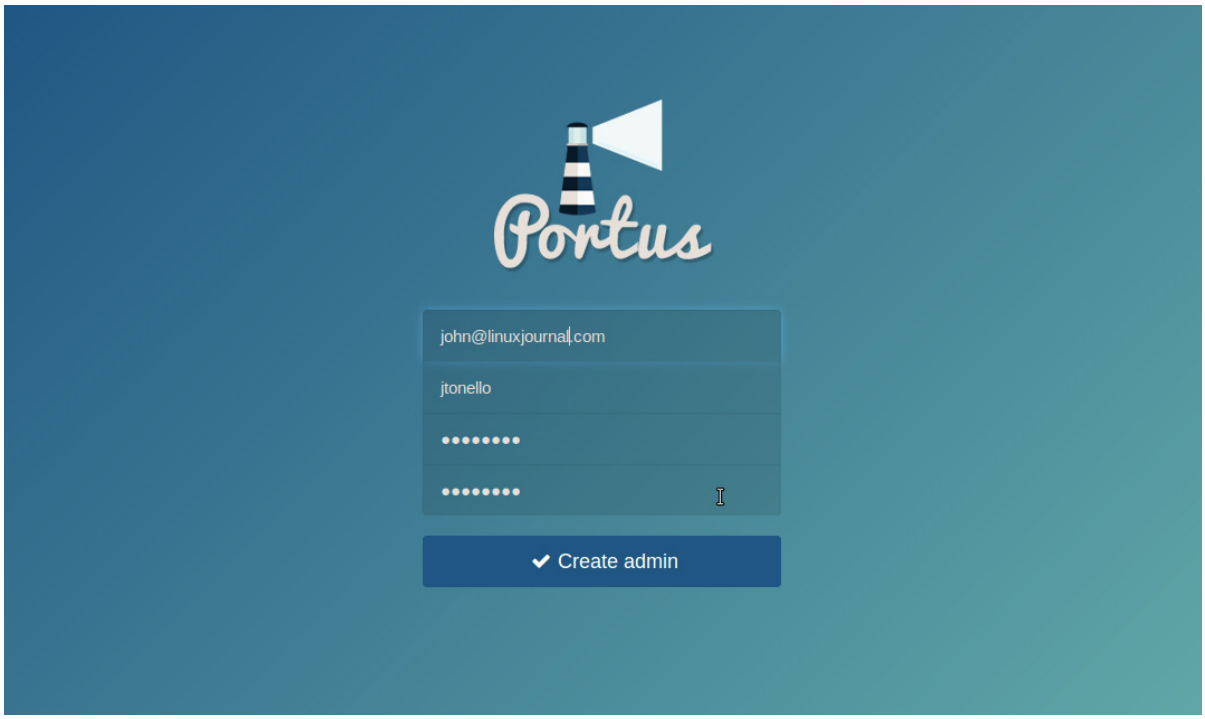


FIGURE 5. When you first open Portus, you're prompted to set up an admin user, which becomes the admin user for logging in to your local registry from the command line.

you can pass an `--insecure-registry` flag in /etc/sysconfig/docker, and your registry still will listen on port 5000, but you won't have to install any certificates to get it working.

Once the docker-registry and Portus are running, you can log in and create an account. If your host server is called my.registry, you'd log in at https://my.registry.

You'll be prompted to create a new admin user, which you can use later to log in to your local registry from the command line. Portus then asks for a name for your new registry and the

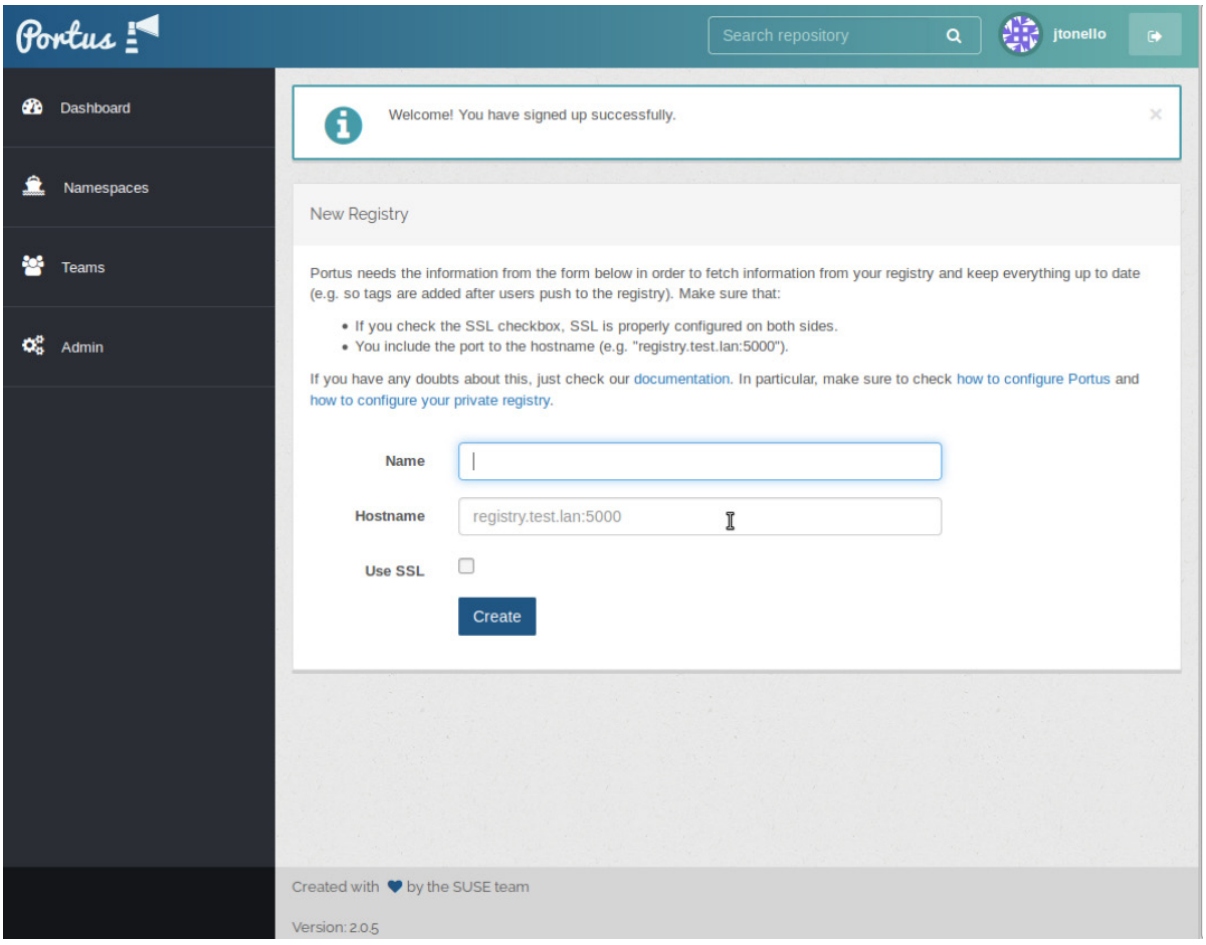


FIGURE 6. **Connecting Portus to your local docker-registry is done by giving it a name, and the hostname and port number.**

hostname, including the port number (Figure 6).

When you return to the command line to push and pull images, you connect Docker to your local registry by logging in using the domain name and port number—but no https:

```
$ docker login my.registry:5000
```

When prompted for a user name and password, use the credentials you just created at the Portus portal.

It's critical that the domain name and port you create in Portus match the domain name and port you put in /etc/registry/config.yml. If they don't, you'll get a connection error when you try to push your images into your on-premises registry.

For illustration purposes, I pushed the mysql image I created (and modified) into my on-premises Docker repository. Pushing images to Docker is as simple as:

```
$ docker push image-name:tag
```

However, if you try that on an image, you'll send it to the public Docker Hub, not your on-premises repository. Instead, you first need to tag your image. I called my SUSE 12 image "myimage" and tagged it like this:

```
$ docker tag myimage:latest suse.home:5000/jtonello/myimage:latest
```

Tagging works "silently"; you won't see any output unless there's an error. Once you've tagged your image, you can push

it to your local repository and manage it through Portus:

```
$ docker push suse.home:5000/jtonello/myimage:latest
```

On the screen, you'll see the pushing begin and a count-up of the MB. When it's complete, you'll see some digest output. If you rush over to Portus, you won't see your image right away; wait a few minutes for the Portus Dashboard or Namespaces tabs to refresh.

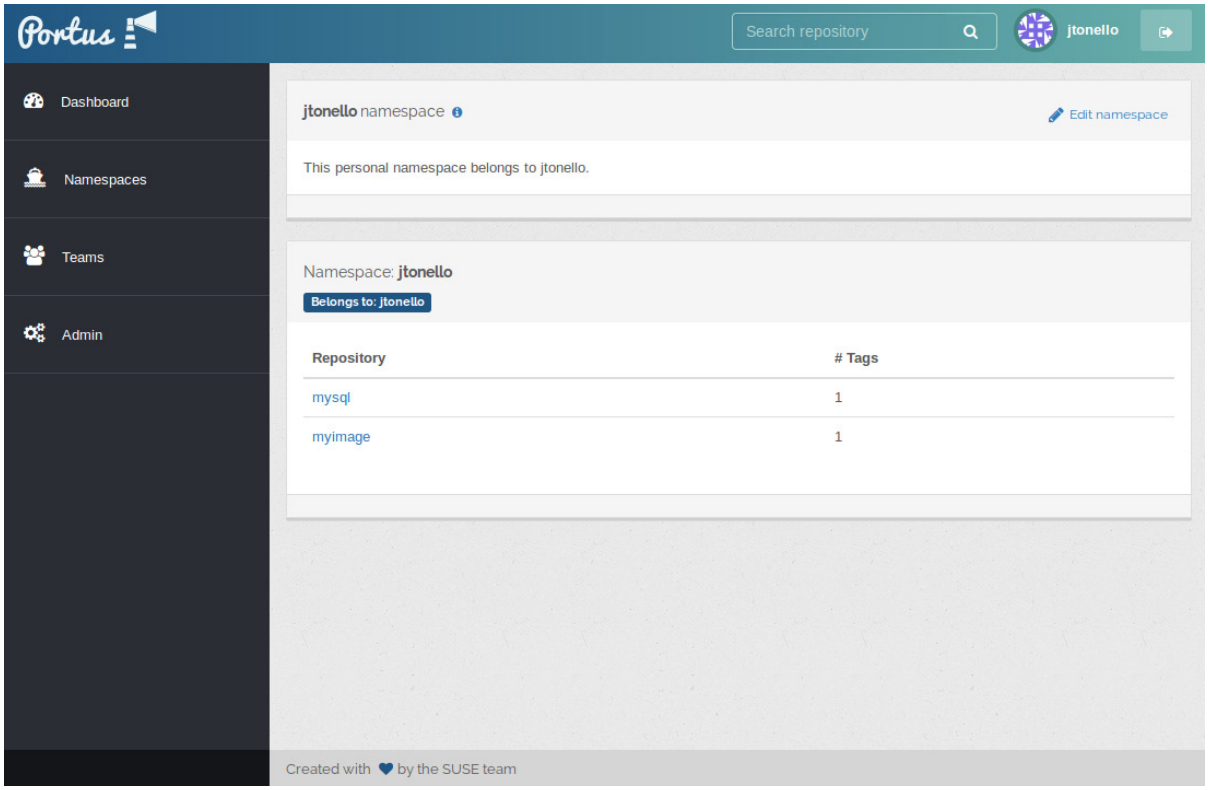


FIGURE 7. A running version of Portus provides a clean graphical interface, plus security and other features, to the locally installed docker-registry. The repository contains two images ready for sharing.

If you're unwilling to wait for your local docker-registry to refresh, you can sync the registry with the Portus database manually by running a simple command; see the Resources section at the end of this ebook for more information.

Pulling images from your private repository is essentially the `push` directive issued with `pull`:

```
$ docker pull suse.home:5000/jtonello/myimage:latest
```

If my public Docker Hub namespace was also "jtonello", and this image existed there, I would `tag`, `push` and `pull` my images much the same way:

```
$ docker tag myimage:latest jtonello/myimage:latest
```

and:

```
$ docker push jtonello/myimage:latest
```

and:

```
$ docker pull jtonello/myimage:latest
```

The only thing missing in these last examples is the docker.io domain name, which is appended invisibly when you're pushing to the public Docker Hub.

Maintaining It All with zypper and zypper-docker

One of the challenges of managing containers, and the

images that spawn them, is keeping them up to date. Fortunately, SLES12 includes zypper-docker, a simple command-line tool that enables you to list your images, look for updates and patches, and bring them up to date without destroying the originals.

It does all that by applying updates to a new image, never overwriting the old. In fact, zypper-docker will fail if you attempt to overwrite an existing Docker image with the same name.

You can find out which updates are available by listing the updates and patches.

In this case, I had it look for updates of the standard SLES12 image I pulled:

```
$ zypper-docker list-updates suse/sles12
```

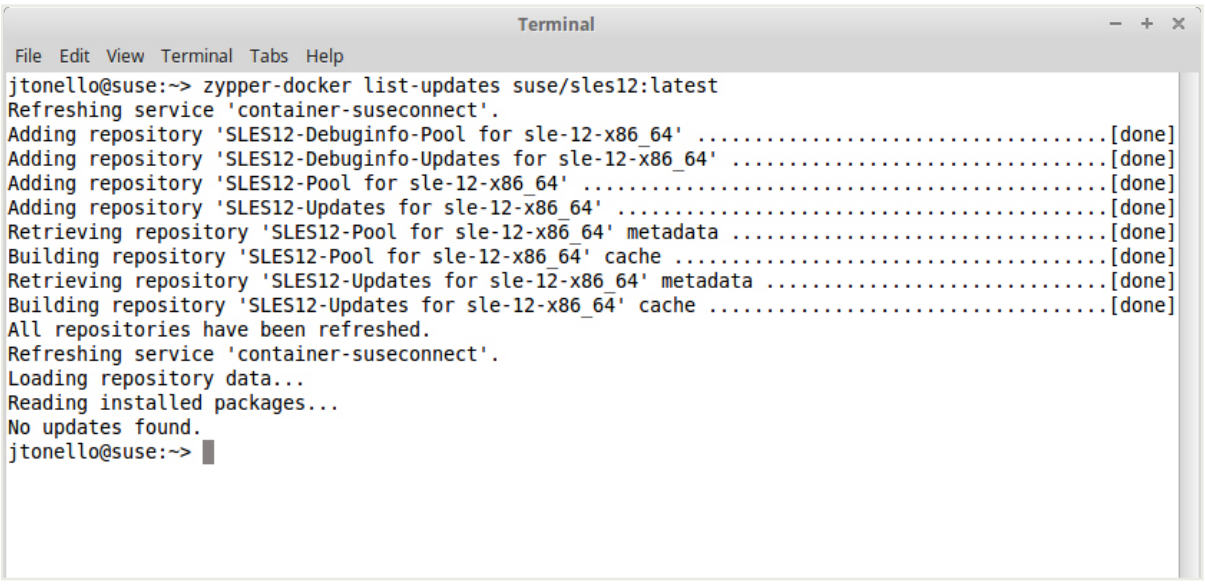


FIGURE 8. The **SUSE** tool zypper-docker lets you check your Docker images for updates and patches.

Since the container was new, I really didn't expect any updates and the results of zypper-docker showed none. The result was different for my custom image called "myimage", which I created earlier:

```
$ zypper-docker list-updates myimage:latest
```

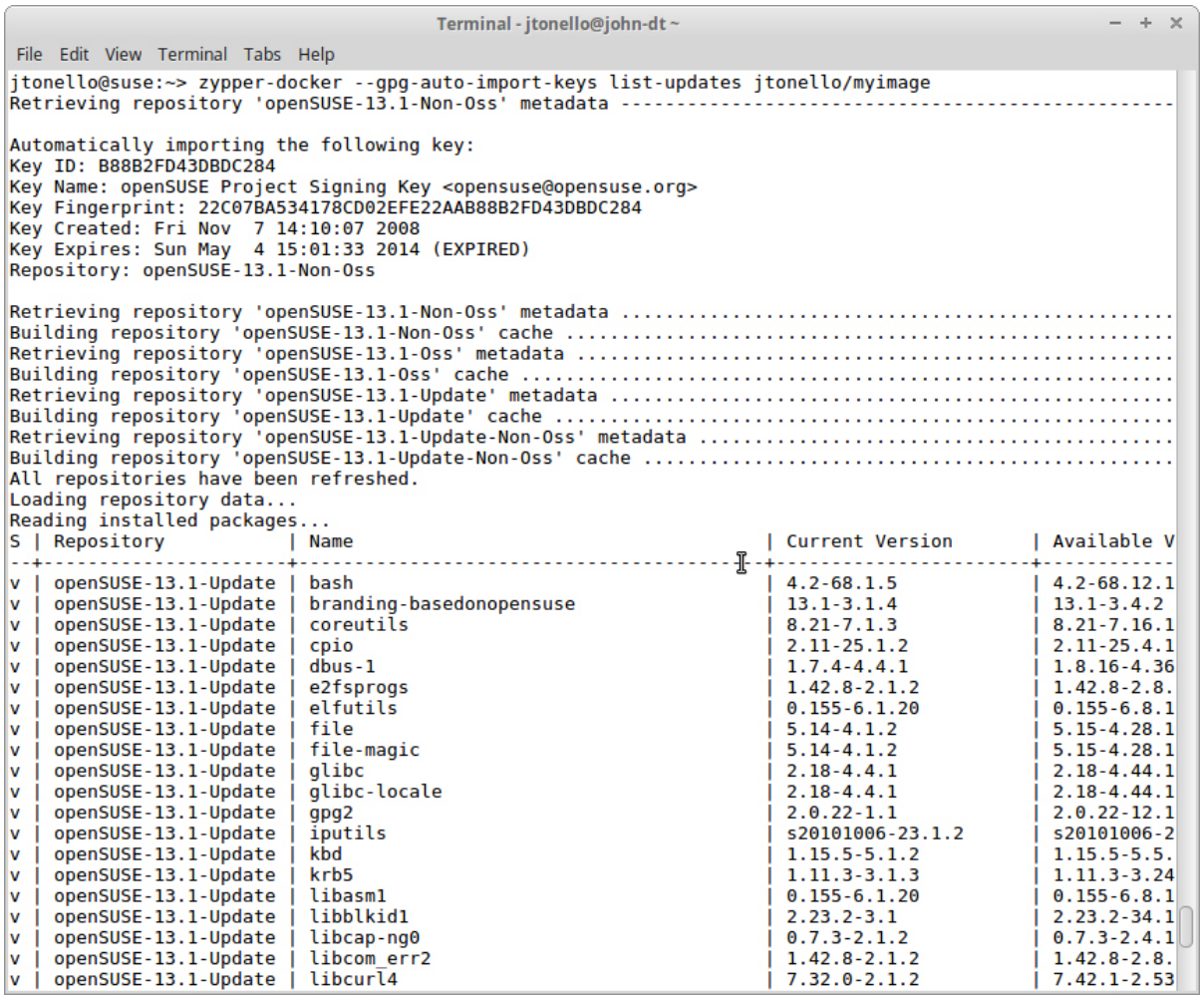


FIGURE 9. **SUSE's zypper-docker tool examines a custom image for updates and finds more than 100, showing both current and available versions of installed packages.**

It found 106 updates available for this image, so I went ahead and did a full update:

```
$ zypper-docker update myimage myimage-new
```

Since updates for this container were available, I ran the `zypper-docker update` command, which takes two arguments: the current name of the image and a name for

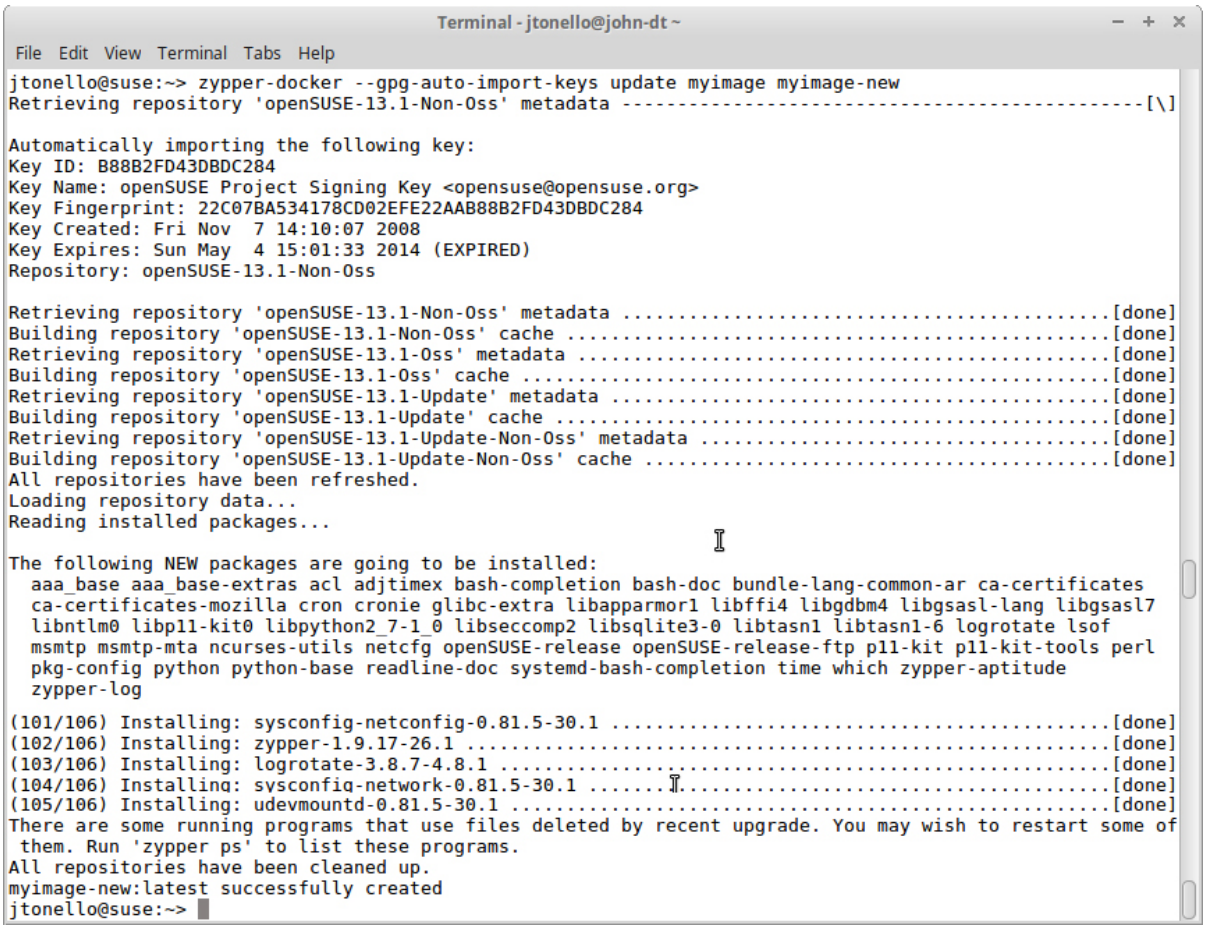


FIGURE 10. The zypper-docker tool runs updates on the container just as it would for a **SUSE** environment, retrieving and installing new packages and outputting a new, updated image.

the newly updated container about to be created. By leaving off the container tags (`:latest`) for each of the two image names, zypper-docker assumes `:latest` and does its thing. Patches are done in a similar way.

If you forget that you've updated or patched a particular container and run it, nothing will immediately tell you that you've got a new version available. However, issuing `zypper-docker ps` will tell you that you're running an image that's been updated and recommend that you stop it and use the fresher one.

Another nice feature of zypper-docker is its ability to apply patches in a surgical way. Sometimes you don't want to

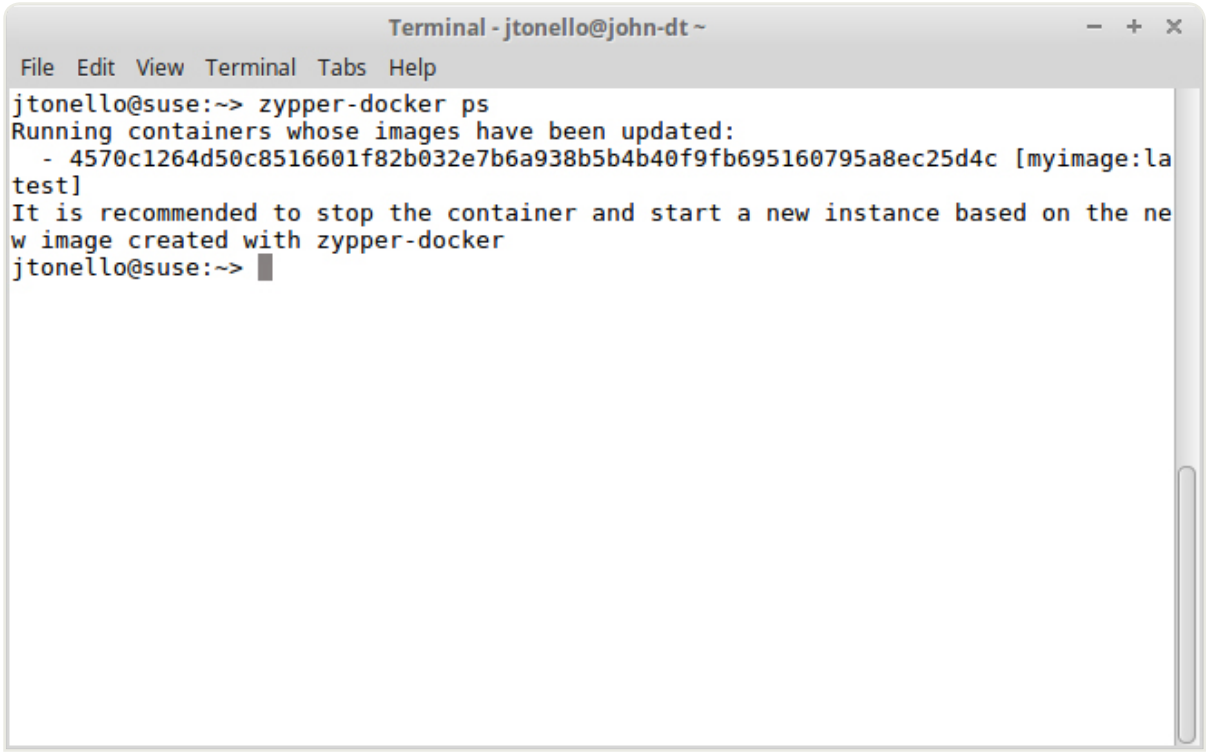


FIGURE 11. The zypper-docker tool can reveal that you're running an outdated container.

apply all the updates available because that could break your application. For those cases, zypper-docker allows you to apply only the patches or updates that, say, fix security issues.

The Docker philosophy is that the right way to update an image is to rebuild it from scratch. It's cleaner. However, sometimes you just can't do that, such as when a user is stuck waiting for an image maintainer to apply security fixes. With zypper-docker, you can fix that security issue without breaking the application running inside the container and allow the user to continue working—safely—until the maintainer of the Docker image delivers a new release.

Manage Your Containers with Orchestration Tools

It's one thing to run a few containers and another thing entirely to build your enterprise infrastructure with them. If you have only a handful, sure, but for true scaling, you need to employ an orchestration tool like Kubernetes or OpenStack.

Each of those is more than a mere container orchestration tool; they can be used to build entire private and public clouds. Kubernetes grew out of Google's need to manage thousands of instances scattered around the world, and OpenStack has gained traction as an open-source alternative to costly licensed cloud platforms. Both are platform-agnostic, meaning you can run them on just about anything, on-premises and in the cloud.

As you begin to scale up your container use, orchestration tools help "objectize" your environment. That is, Kubernetes and OpenStack (and other tools like them) decouple your

SUSE will soon launch a Kubernetes orchestration solution that will further enhance the Docker life cycle deployment with SLES12.

infrastructure from your services and applications. For example, web and database servers can be run in separate, low-overhead containers, but share compute and network namespaces, giving you centralized logic to create, maintain and heal your container infrastructure. These tools can assign network addresses and DNS names on the fly, and respond to your need to, say, scale up during business hours and scale down after dark.

It comes as no surprise that the combination of Docker containers and orchestration tools can dramatically change how you provide resources and services. You can manage horizontal scaling, automated roll-outs, storage management, workload management and self-healing clusters—and use browser-based interfaces to move beyond command-line tools and scripts alone.

SUSE will soon launch a Kubernetes orchestration solution that will further enhance the Docker life cycle deployment with SLES12. That's good news as you begin to evaluate, test and deploy containers in your environment. With it, the folks at SUSE provide you and your team a complete and largely pain-free way to manage containers and the Docker life cycle.

At the OpenStack Summit in Barcelona, SUSE announced

its new OpenStack Cloud 7. This is the first release to include Magnum, an OpenStack component that makes it easy to deploy Kubernetes clusters. That's good news for anyone looking for a painless way to deploy Kubernetes, which can be difficult and time-consuming.

If you have a multi-tenant environment, Magnum also makes it easy to give each tenant its own private and isolated Kubernetes cluster. This "Kubernetes as a Service" can be done with just a few clicks.

If you're looking to deploy Kubernetes outside OpenStack, SUSE recently announced its Container as a Service Platform, known as SUSE CASP. It will be based on MicroOS, a flavor of SUSE Enterprise Linux that's smaller than JeOS and optimized to run Linux containers and Kubernetes.

Try Out the SUSE Container Stack

You can get a free 60-day trial of SUSE Enterprise Linux Server 12 (and SLES SP1) and test everything discussed in this ebook. SUSE supports most platforms, including x86_64, z and Power and also gives you free access to SUSE OpenStack Cloud (for x86_64), SUSE Manager and SUSE Enterprise Storage. Check the Resources section for the URL.■

Resources

- Try SUSE for Free: https://www.suse.com/products/server/download
- Docker Registry: https://docs.docker.com/registry/insecure
- SUSE Docker Quick Start Guide: https://www.suse.com/documentation/sles-12/dockerquick/data/dockerquick.html

- Docker—Tag, Push and Pull: https://docs.docker.com/engine/getstarted/step_six
- Kiwi: https://doc.opensuse.org/projects/kiwi/doc
- Kiwi Docker Example: http://flavio.castelli.name/2014/05/06/building-docker-containers-with-kiwi
- Portus: http://port.us.org
- Portus First Steps: http://port.us.org/docs/first-steps.html
- Portus and Docker Registry Sync: http://port.us.org/features/1_Synchronizing-the-Registry-and-Portus.html
- zypper-docker: https://github.com/SUSE/zypper-docker
- Containers as a Service Platform: https://www.suse.com/communities/blog/introducing-suse-containers-service-platform