HOL-2433-01-MAP

Containers and Kubernetes 101 on VMware Tanzu

vmware®

Table of contents

Lab Overview - HOL-2433-01-MAP - Containers and Kubernetes 101 on VMware Tanzu

Lab Description [2]

This lab is an overview of Dockers and Kubernetes and is demonstrating the Kubernetes capability which is available with vSphere with Tanzu. This lab is using vSphere with Tanzu for hosting Kubernetes.

In this lab we will introduce the basic concepts of containers in general and docker in particular. We will look at how to implement a simple application in docker and how to use container networking and docker volumes. We will also introduce the concept of container management with a brief overview of Kubernetes. We will briefly touch on the VMware Tanzu portfolio of products and VMware Tanzu Kubernetes Grid (TKG) which is the underlying platform upon which this labs docker and kubernetes platform is utilizing.

Lastly we will take a quick tour of some of the free educational resources which VMware provides for learning about Kubernetes and the Tanzu Portfolio, Introduction to VMware Learning Platform & Kube Academy and Introduction to TechZone.

Lab Guidance [3]

Welcome! This lab is available for you to repeat as many times as you want. To start somewhere other than the beginning, use the Table of Contents in the upper right-hand corner of the Lab Manual or click on one of the modules below.

- Module 1 - Introduction to Containers (15 minutes) (Basic)
- Module 2 - A Quick Tour of Docker (15 minutes) (Basic)
- Module 3 - Introduction to Kubernetes (15 minutes) (Basic)
- Module 4 - Introduction to VMware Learning Platform & Kube Academy
- (15 minutes) (Basic)
- Module 5 - Introduction to VMware TechZone (15 minutes) (Basic)

Lab Captains:

- Kevin Brady, Staff Partner Solutions Architect USA

This lab manual can be downloaded from the Hands-on Labs document site found here:

http://docs.hol.vmware.com

This lab may be available in other languages. To set your language preference and view a localized manual deployed with your lab, utilize this document to guide you through the process:

http://docs.hol.vmware.com/announcements/nee-default-language.pdf

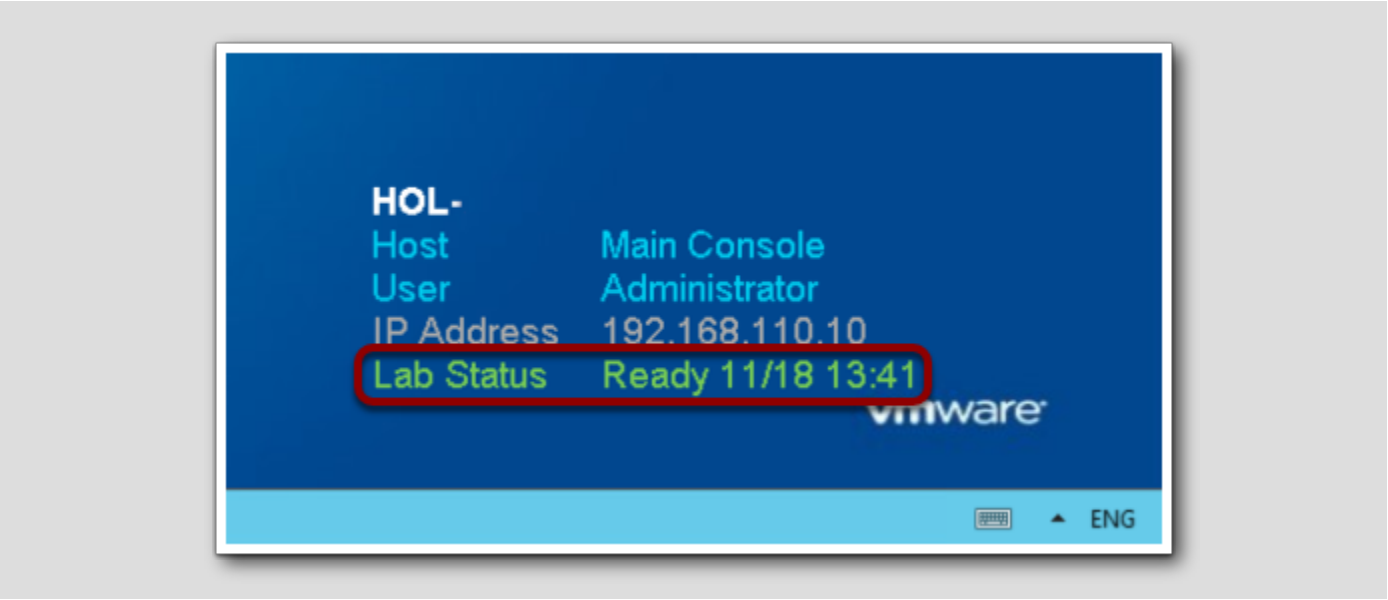
First time using Hands-on Labs? [4]

Welcome! If this is your first time taking a lab review the VMware Learning Platform interface and features before proceeding.

For returning users, feel free to start your lab by clicking next in the manual.

You are ready....is your lab?

[5]



The lab console will indicate when your lab has finished all the startup routines and is ready for you to start. If you see anything other than "Ready", please wait for the status to update. If after 5 minutes your lab has not changed to "Ready", please ask for assistance.

Module 1 -Introduction to Containers (15 minutes) Basic

vSphere with Tanzu Introduction [7]

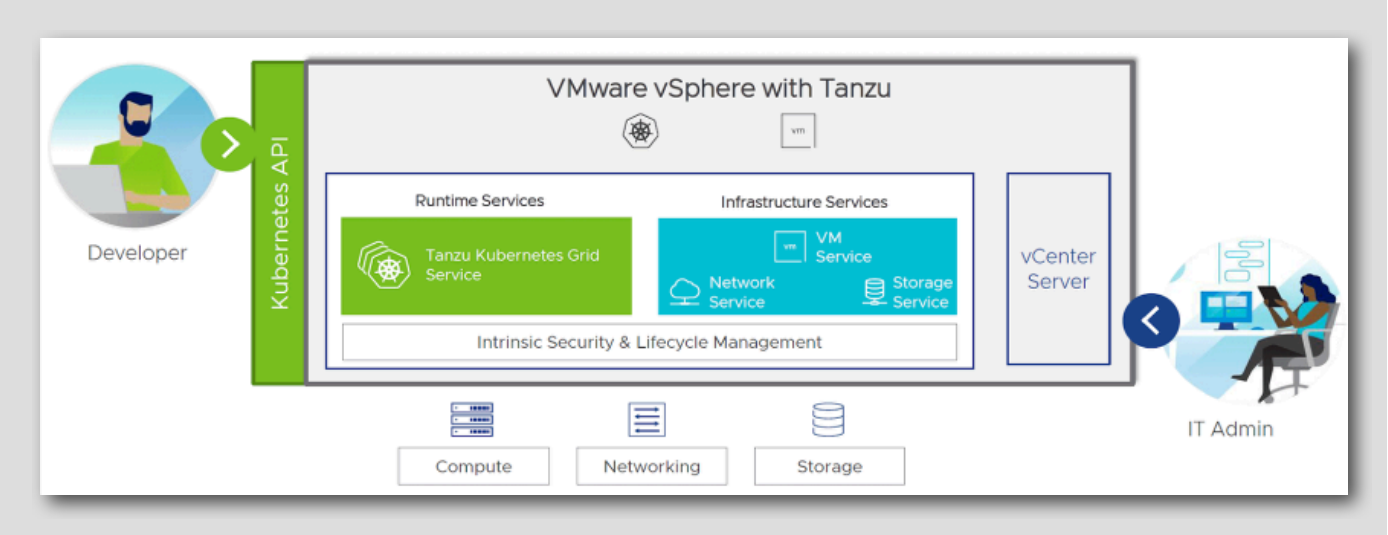
In this lab we are using vSphere with Tanzu for hosting our kubernetes environment.

This section provides a brief overview of vSphere with Tanzu which is covered in more detail in the vSphere with Tanzu (2413) lab.

Kubernetes is now built into vSphere with Tanzu which allows developers to continue using the same industry-standard tools and interfaces they've been using to create modern applications. vSphere Admins also benefit because they can help manage the Kubernetes infrastructure using the same tools and skills they have developed around vSphere. To help bridge these two worlds we've introduced a new vSphere construct called Namespaces, allowing vSphere Admins to create a logical set of resources, permissions, and policies that enable an application-centric approach.

What is vSphere with Tanzu? [8]

VMware vSphere with Tanzu delivers developer ready infrastructure and application-focused management for streamlined development, agile operations, and accelerated innovation. It's a flexible environment for modern applications that are built from microservices and run across heterogenous environments.



With vSphere with Tanzu, VMware delivers embedded Tanzu Kubernetes Grid Service for fully compliant and conformant Kubernetes capabilities for containerized applications. This approach provides Kubernetes APIs to developers, enabling CI/CD (continuous integration / continuous delivery) processes across a global infrastructure including on-premises data centers, hyperscalers, and Managed Service Providers (MSP) infrastructure. It unites the data center and the cloud with an integrated cloud operating model. Now enterprises can increase the productivity of developers and operators, enabling faster time-to-innovation combined with the security, stability, and governance, and avoid cost proliferation due to multiple stacks of IT infrastructure or cloud services.

vSphere 8 with Tanzu vs vSphere 7 [9]

This lab includes some features that are available only in vSphere 8. Specifically the multi-cluster / multi-zone supervisor and workload clusters and new TKG cluster class spec are only available in vSphere 8. Aside from those features and some minor UI changes, the Tanzu functionality shown in this lab is also available in vSphere 7

Streamlined Development of Kubernetes Applications [10]

vSphere with Tanzu enables the DevOps model with infrastructure access for developers through Kubernetes APIs. It includes the Tanzu Kubernetes Grid Service, which is VMware's compliant and conformant Kubernetes implementation for building modern containerized applications. In addition, the vSphere Pod Service compliments the Tanzu Kubernetes Grid Service for application container instances requiring VM-like isolation benefits of improved performance and security of a solution built into the hypervisor.

Agile Operations for Kubernetes [11]

We are introducing a lot of value in vSphere with Tanzu for the VI admin. We deliver a new way to manage infrastructure, called application-focused management. This enables VI admins to organize multiple objects into a logical group and then apply policies to the entire group. For example, an administrator can apply security policies and storage limits to a group of virtual machines and Kubernetes clusters that represent an application, rather than to all of the VMs and clusters individually.

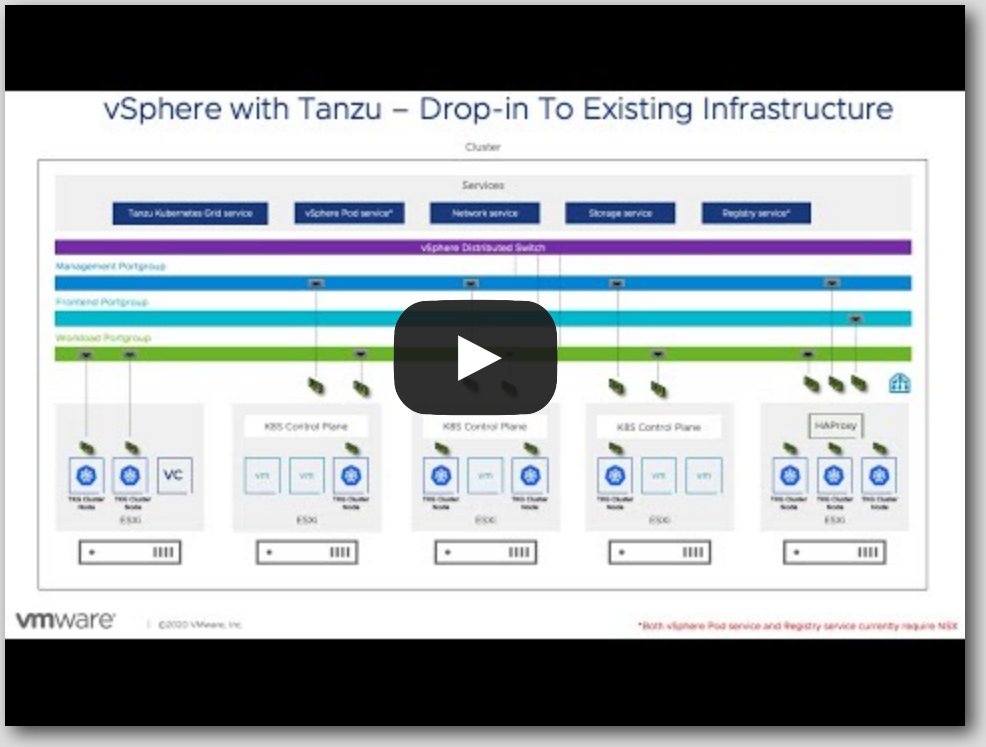
vSphere with Tanzu Delivers Essential Services for Hybrid Cloud [12]

VMware solved the challenges faced by traditional apps across heterogeneous architectures with the introduction of VMware vSphere. With vSphere, we’ re delivering the essential services for the modern hybrid cloud. VMware hyperconverged infrastructure (HCI) stack combines compute, storage, and networking with unified management—vSphere with Tanzu powers the innovation behind developer ready infrastructure. With the new Kubernetes and RESTful API surface, developers can streamline their work and IT administrators can improve productivity using application-focused management.

Technical Overview of vSphere with Tanzu (4:25)

[13]

https://www.youtube.com/watch?v=0pI65Kn9AKk



vSphere with Tanzu delivers Developer ready infrastructure by allowing IT teams to use their existing vSphere environments to rapidly deploy Kubernetes clusters for their development teams. A simple enablement wizard gets you up and running in minutes. Learn how **Networking with vSphere Distributed Switch** works in this quick video

Introduction to Containers

[14]

In this Chapter, we will explain containers and how they enable third party platform application architectures to be run efficiently in distributed environments.

Notice: This section is all reading. If you are eager to get to typing on the keyboard, you can move on to the next module, but you will miss out on learning some very important concepts

Brief History of Containers

[15]

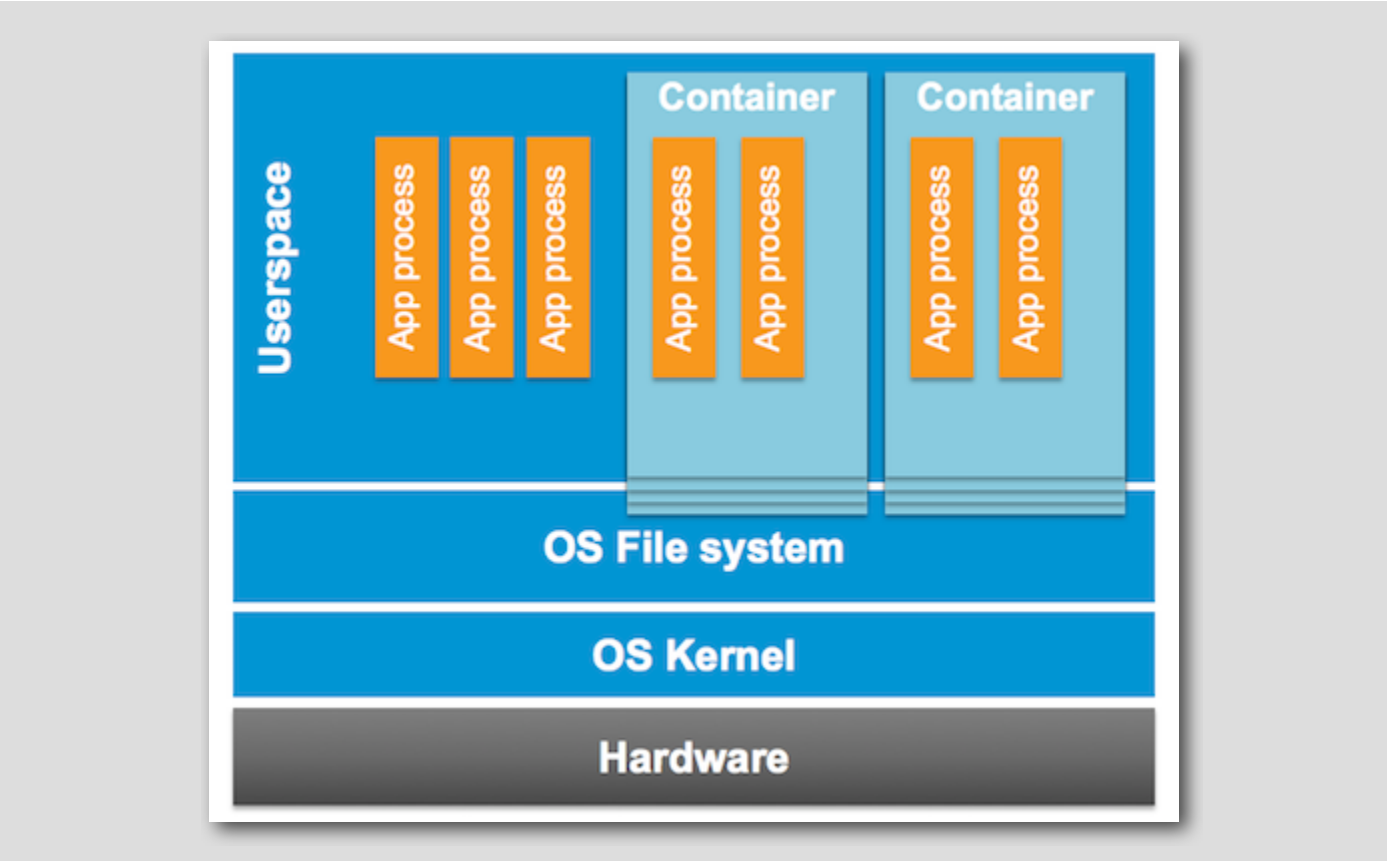
While containers are certainly a very popular topic right now, containers themselves are not new. They have existed for many years. FreeBSD, Solaris Zones, LXC…there are many incarnations of containerization technology.

You may ask - then why are containers and in particular docker so popular? For a few good reasons, but mainly because containers offer a very convenient form factor for distributing and sharing code. Very complex applications can be run using a single command line, the containerization ensures that the application behaves the same regardless of the underlying OS, hardware and networking environment. This has made it very popular among developers and testers who need to quickly spin up complex environments.

Containers are not without challenges. While they work extremely well in development, organizations are still struggling with issues like isolation, security, network virtualization and monitoring of containers. As you will discover throughout this lab, with the Tanzu portfolio of products VMware is fully embracing the idea of concept of containers and is providing solutions to many of the problems we just discussed.

What are Containers?

[16]

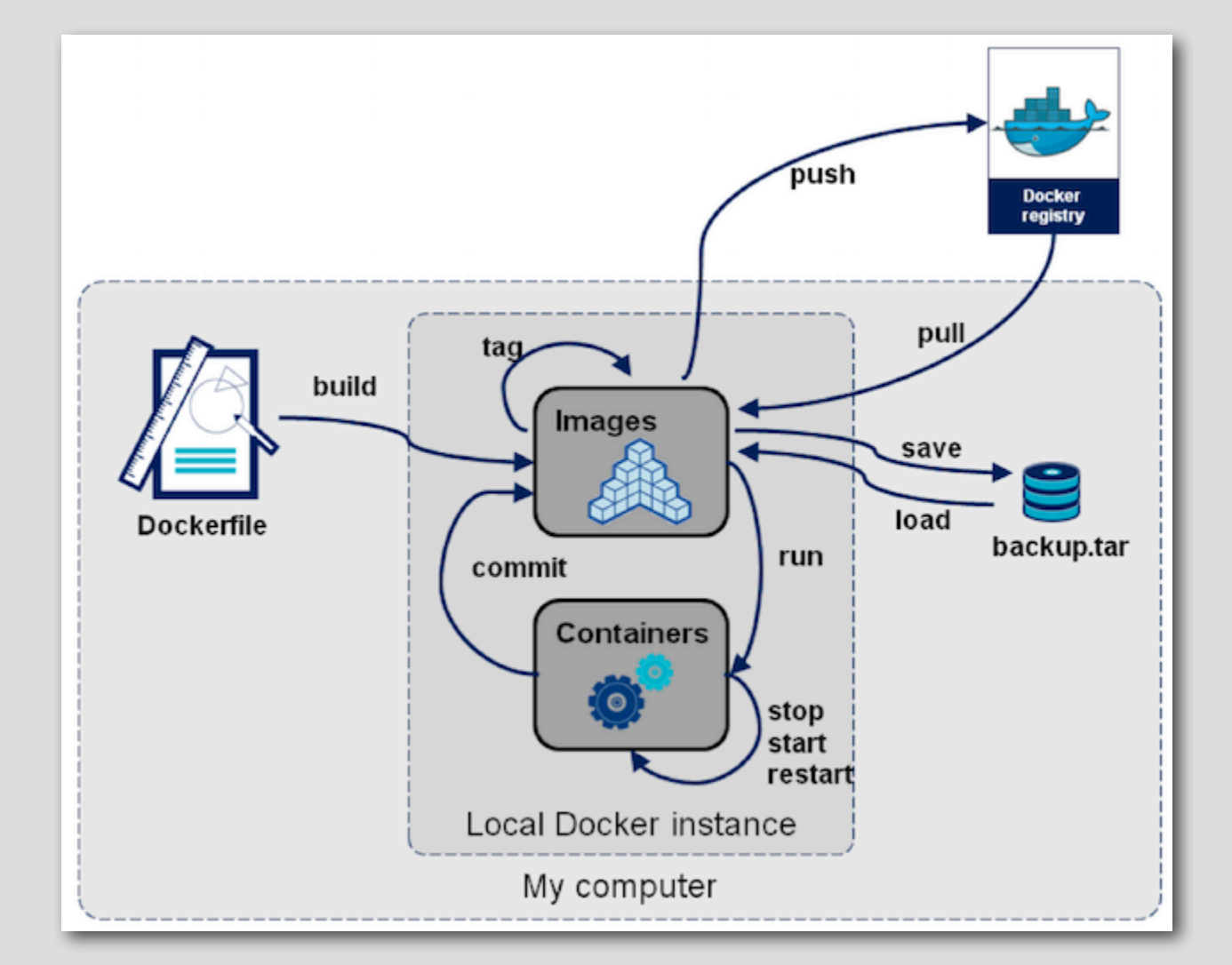


Containers are an OS-level virtualization method in which the kernel of an operating system allows for multiple isolated user-space instances, instead of just one. The primary benefits of using containers include limited overhead, increased flexibility and efficient use of storage; the container looks like a regular OS instance from the user's perspective. Changes to the image can be made very quickly and pushed to a repository to share with others for further development and utilization.

Note that Kubernetes supports many container runtimes, including containerd, Docker Engine, CRI-O, and Mirantis Container Runtime. Docker is the most frequently used Kubernetes container runtime and the container runtime we will be discussing in this manual and will be using in this lab.

What is Docker?

[17]



Docker containers wrap up a piece of software in a complete filesystem that contains everything it needs to run: code, runtime, system tools, system libraries – anything you can install on a server. This guarantees that it will always run the same, regardless of the environment it is running in.

Containers running on a single machine all share the same operating system kernel so they start instantly and make more efficient use of RAM. Images are constructed from layered filesystems so they can share common files, making disk usage and image downloads much more efficient. Docker containers are based on open standards allowing containers to run on all major Linux distributions and Microsoft operating systems.

Containers include the application and all of its dependencies, but share the kernel with other containers. They run as an isolated process in userspace on the host operating system.

Docker is a natural fit for microservice-based architectures.

How do Containers and Virtual Machines Differ?

[18]

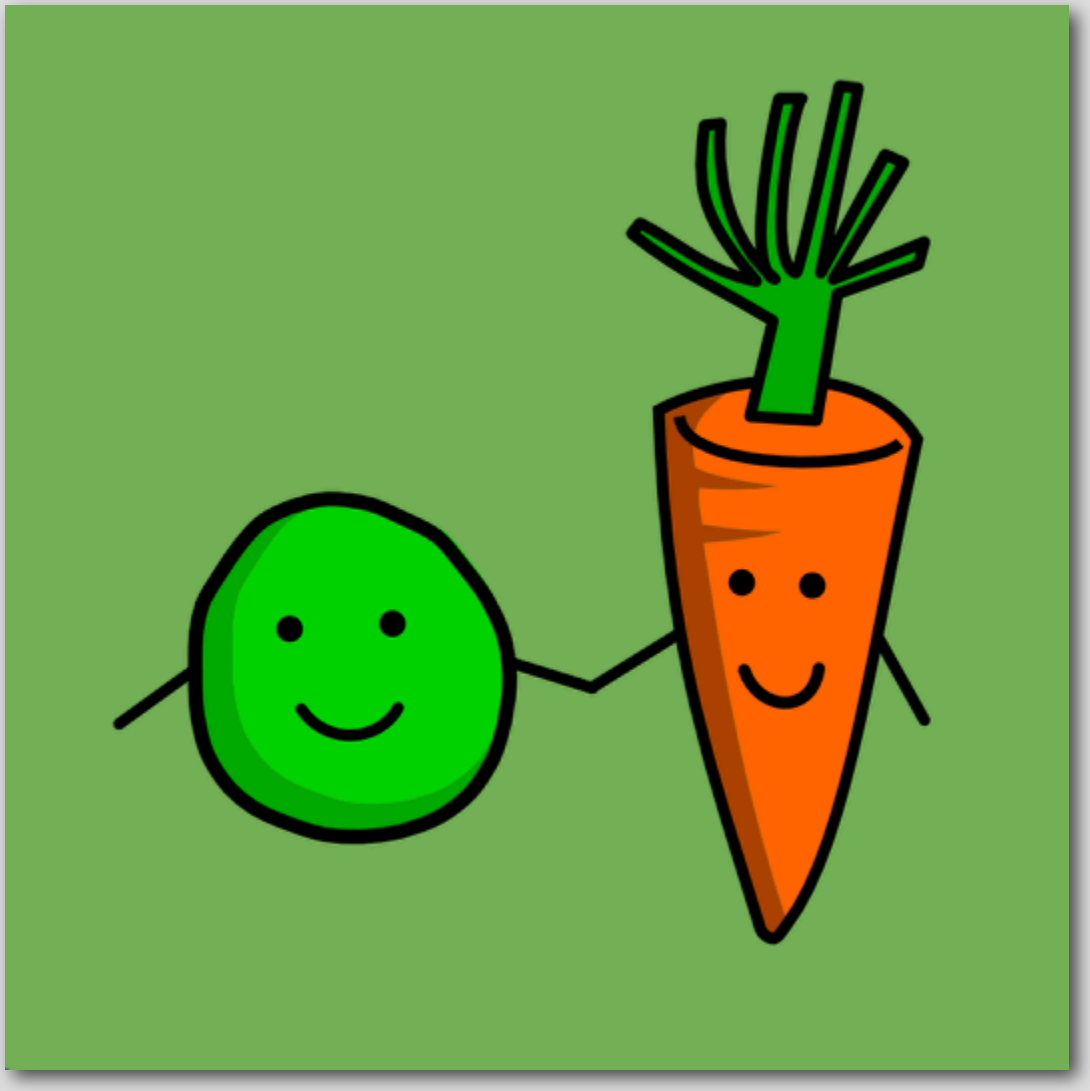
This sounds a lot like virtual machines, doesn't it? Aren't they just variations on the same theme?

Well, not really… A virtual machine is an emulation of a piece of hardware. When you run a virtual machine, you run an entire operating system including kernel and BIOS that behaves as if it interacted with physical hardware. A container, on the other hand, runs in an emulated operating system with multiple containers sharing a single kernel. Containers run on top of a host operating system, which, in turn can run on either physical or virtual hardware.

A container is intended to run a single application. Containers are typically very specific, intended to run MySQL, Nginx, Redis, or some other application. So what happens if you need to run two distinct applications or services in a containerized environment? The recommendation is usually to use two separate containers. The low overhead and quick start-up times make running multiple containers trivial, thus they are typically scoped to a single application.

A virtual machine, on the other hand, has a broader range, and can run almost any operating system. As you are likely aware, the virtual machine serves as an extremely firm boundary between OS instances that's enforced by a robust hypervisor, and connects to Enterprise-level storage, network and compute systems in a trusted, well-defined and secure manner. Virtual machines have traditionally lent themselves to running more traditional 3-tier (Web - App - Database) applications that compromise 99% of the application space today.

Virtual machines and containers: better together [19]



Containers provide great application portability, enabling the consistent provisioning of the application across infrastructures. However, applications and data alone are rarely the major barrier to workload mobility. Instead, operational requirements such as performance and capacity management, security, and various management tool integrations can make redeploying workloads to new environments a significant challenge. So while containers help with portability, they're again only a piece of a bigger puzzle.

Due to the fundamental differences in architecture (namely the ESXi hypervisor used by virtual machines versus the shared kernel space leveraged by containers), Linux containers will not achieve the same level of isolation and security. Furthermore, the toolsets available in the virtual machine ecosystem are battle-tested and Enterprise-grade, enabling scores of benefits (stability, compliance, integrated operations, etc) that are indispensable to operations and infrastructure teams.

For these reasons, VMware provides the best of both worlds by offering an optimized operating system built for containers to run with minimal overhead. By dedicating an extremely lightweight operating system to run containerized workloads, we don't have to choose one or the other - we can have both! By taking advantage of memory sharing, a core feature of the ESXi hypervisor, we drastically reduce the operating system overhead while enabling the application flexibility promised by containers.

Application Delivery with Containers

[20]

Notice: This section is all reading. If you are eager to get to typing on the keyboard, you can move on to the next module, but you will miss out on learning some very important concepts.

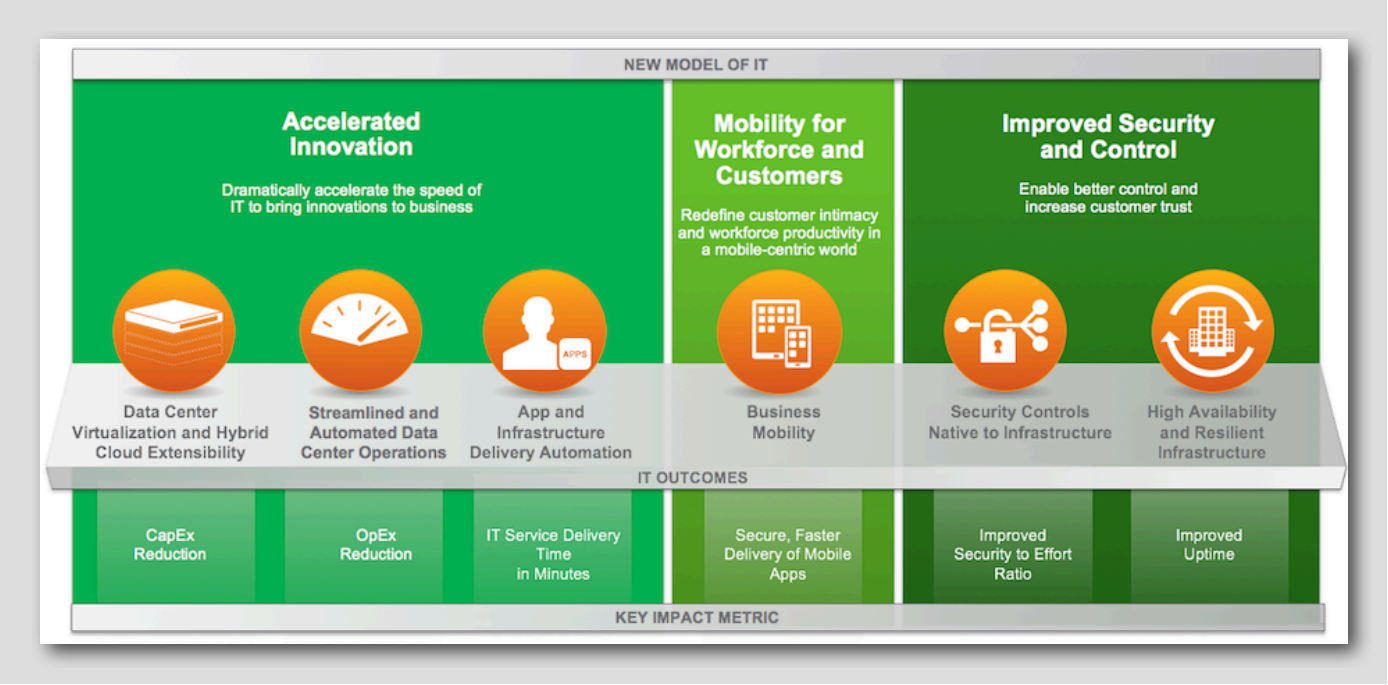
Containers enable you to deliver an application as something called a micro service. This isn't a new concept. In fact, it's quite old. Back in the mainframe days if you used something like IBM CICS, your code would be divided into transactions, each responsible for carrying out some specific business function. For example, withdrawing money from an account would be one transaction, depositing would be another and so on. Similar ideas appeared in the 90s with CORBA and the early 2000s with SOA.

At its core, the concept is simple: Each business function is encapsulated in a self-contained service that exposes a well-defined and published interface describing how to interact with it. Since the services are self-contained and independent, they can be managed and upgraded independently. For example, if the credit card functionality of an application needs to be replaced, we can just replace the corresponding micro service. As long as we expose the same interface, any application component using the service will continue running without even noticing the change.

As we discussed above, this is nothing new. What's new is that the container form factor makes delivering and managing micro services a lot easier.

Application Development and Delivery

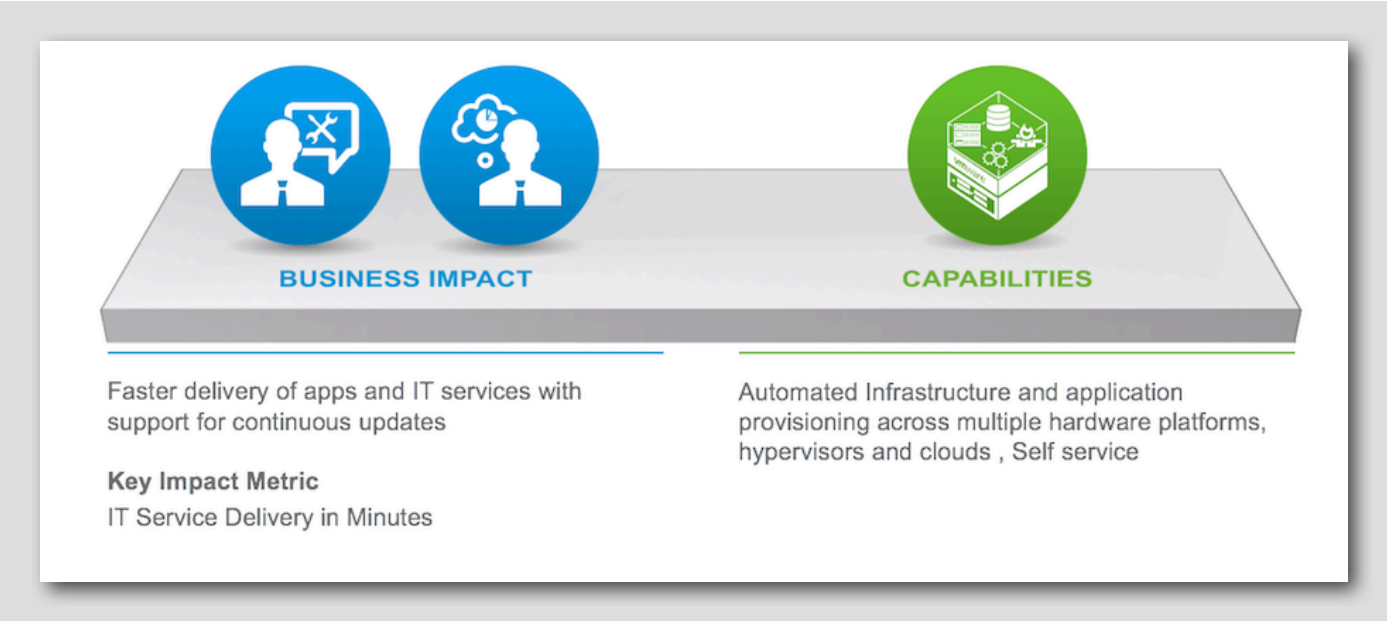
[21]



If we look at the outcomes which are delivered from a new model of IT, Businesses are increasing their focus on **App and Infrastructure Delivery Automation** throughout the datacenter.

App and Infrastructure Delivery Automation

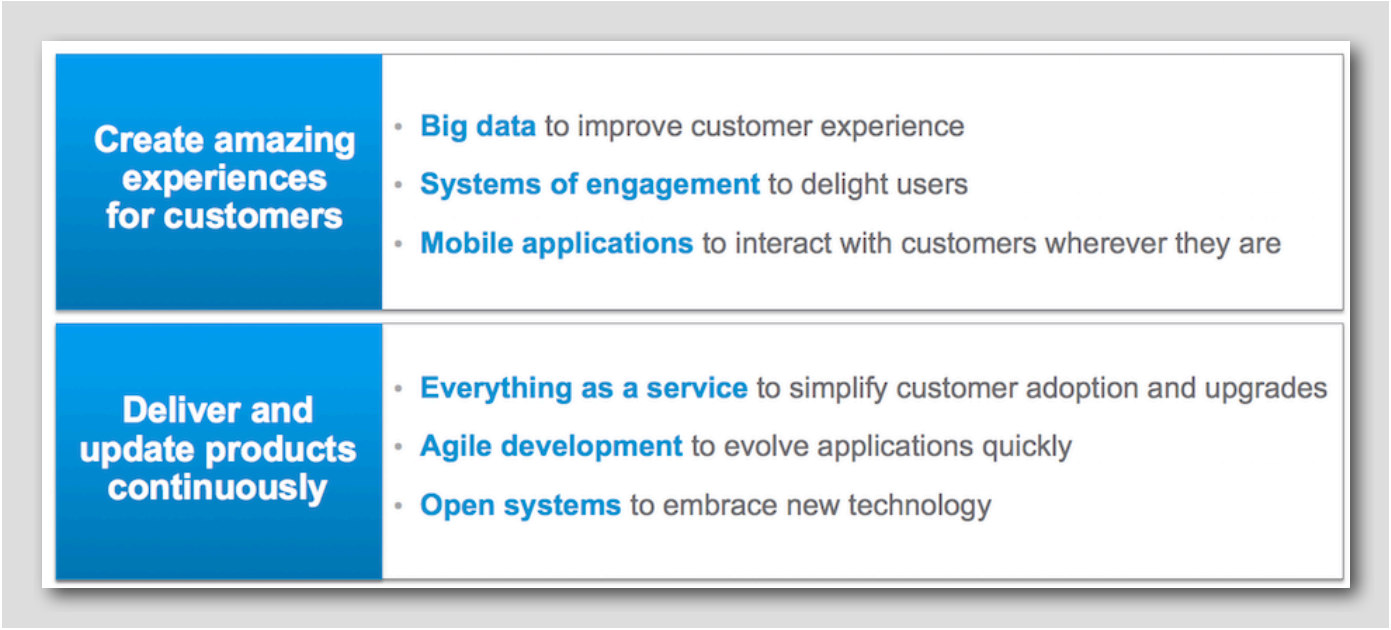
[22]



IT is making strides to provide the ability to enable faster delivery of application and IT services leveraging capabilities derived from automated infrastructure and application provisioning.

New Business Imperative

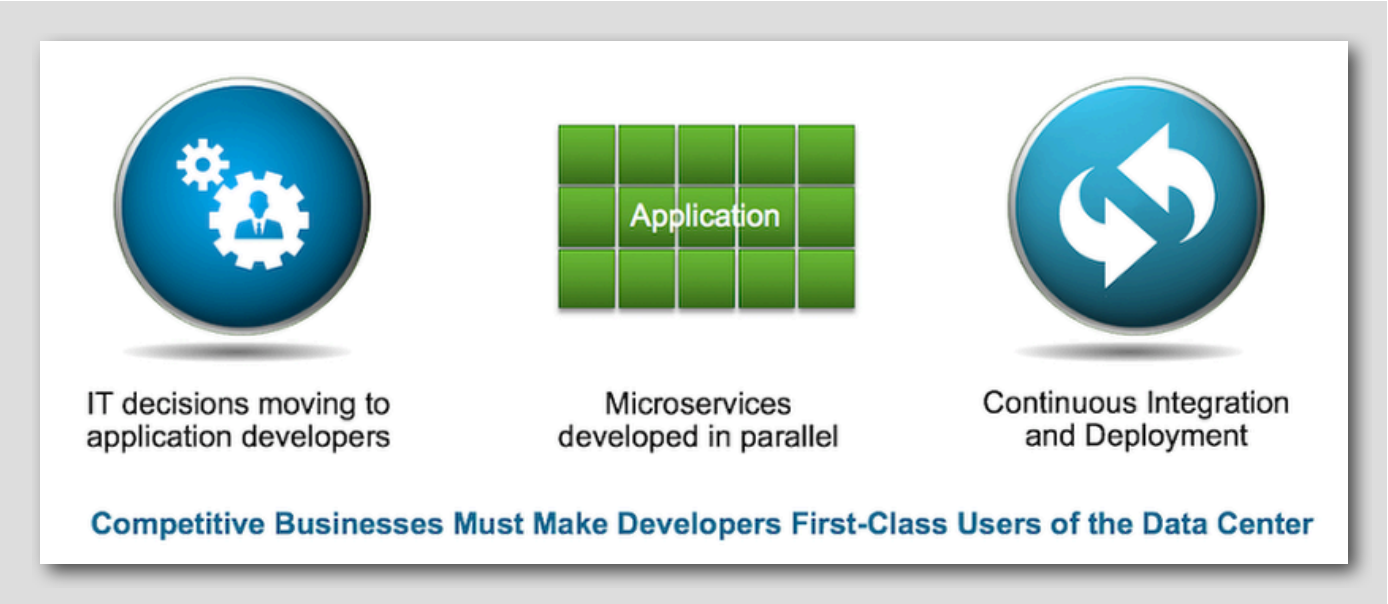
[23]



Competitive businesses are delivering new applications to market in increasingly faster cycles, ushering in technologies like Linux containers and microservices. Next-generation applications are being built on infrastructure assumed to be dynamic and elastic. To keep our customers agile, our App Modernization group builds infrastructure technologies to open, common standards that preserve security, performance, and ease-of-use, from developer desktop to the production stack.

Moving Faster Requires Design and Culture Changes

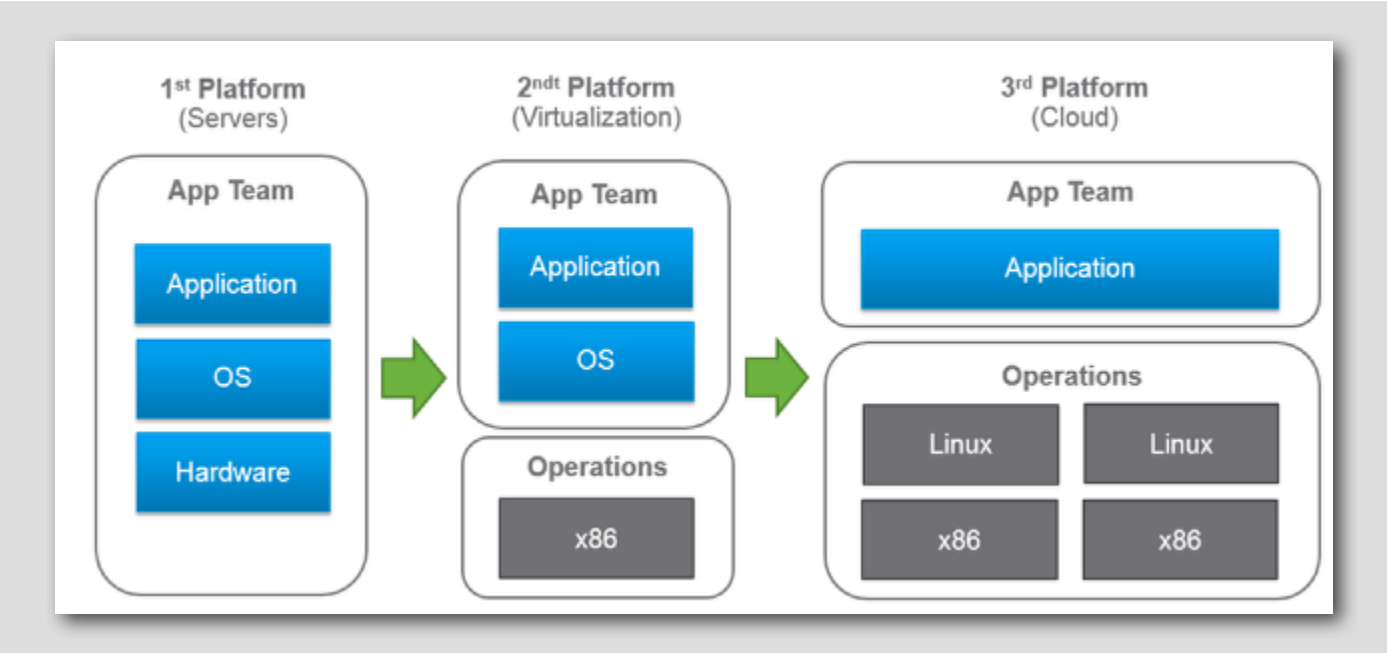
[24]



To move faster, businesses implement a variety of cultural, design, and engineering changes. At VMware, we are striving to make the Developer a first class citizen of the Data Center and help align them with IT's journey to achieve streamlined App and Infrastructure Delivery Automation.

History of Platforms

[25]



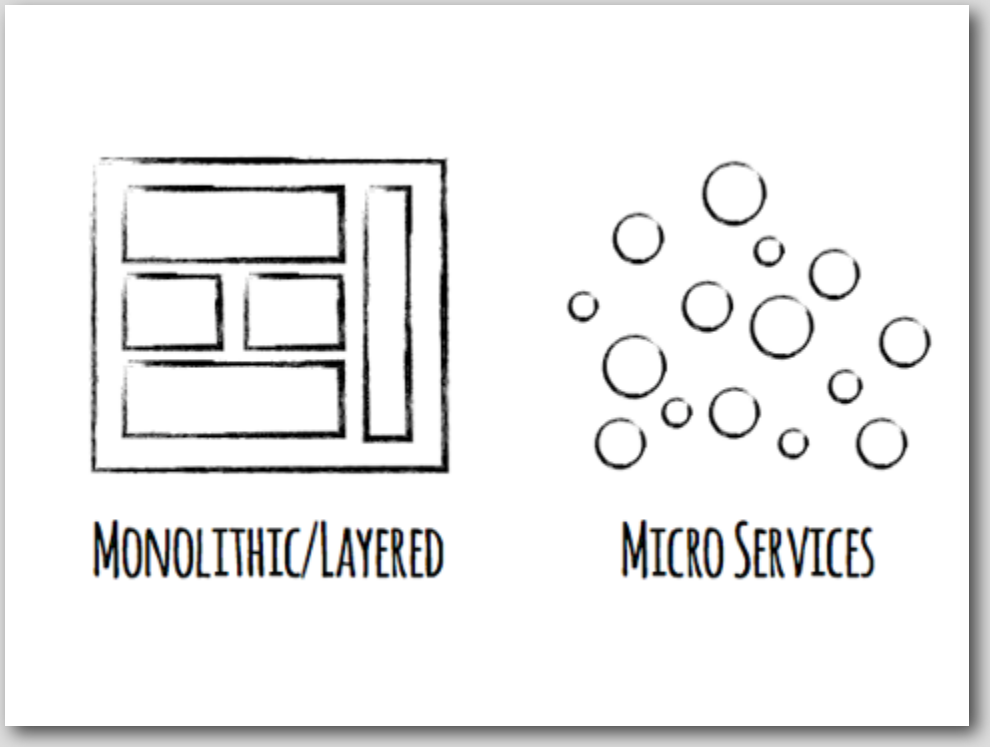
1st Platform systems were based around mainframes and traditional servers without virtualization. Consolidation was a serious issue and it was normal to run one application per physical server.

2nd Platform architectures have been the standard mode for quite a while. This is the traditional Client/Server/Database model with which you are likely very familiar, leveraging the virtualization of x86 hardware to increase consolidation ratios, add high availability and extremely flexible and powerful management of workloads.

3rd Platform moves up the stack, standardizing on Linux Operating Systems primarily, which allows developers to focus on the application exclusively. Portability, scalability and highly dynamic environments are valued highly in this space. We will focus on this for the rest of this module.

3rd Platform - Microservice Architecture

[26]



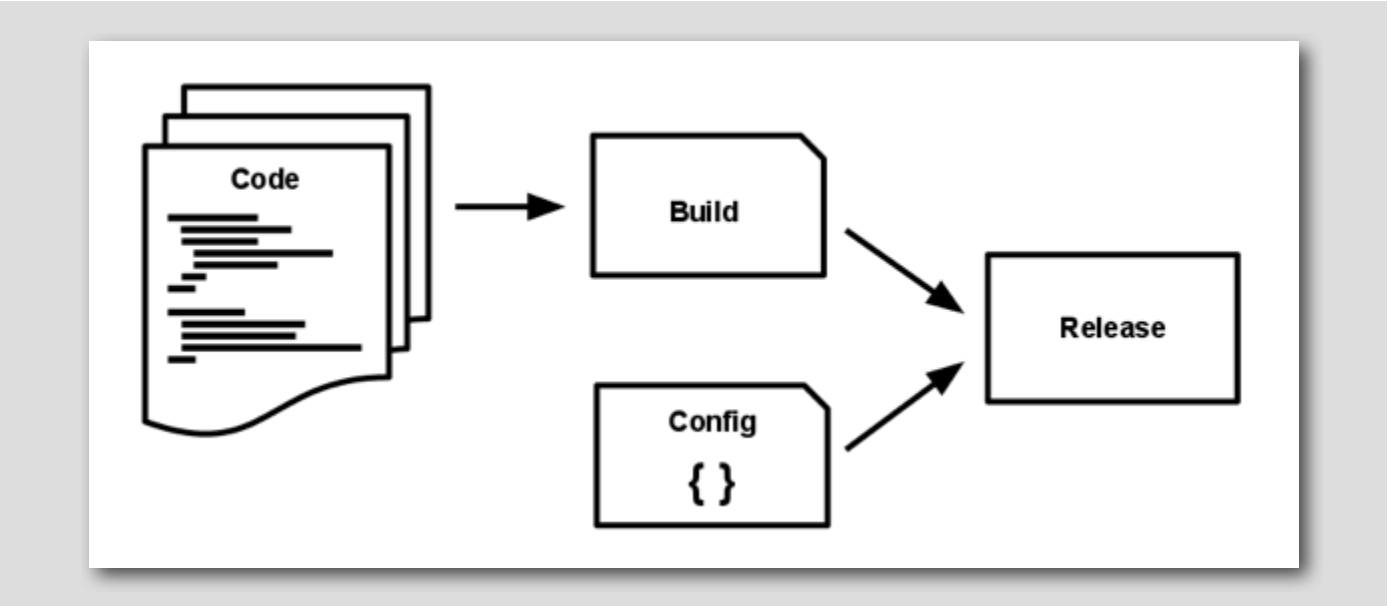
Microservices are growing in popularity, due in no small part to companies like Netflix and Paypal that have embraced this relatively new model. When we consider microservices, we need to understand both the benefits and the limitations inherent in the model, as well as ensure we fully understand the business drivers.

At its heart, the microservice architecture is about doing one thing and doing it well. Each microservice has one job. This is clearly in stark contrast to the monolithic applications many of us are used to; using microservices, we can update components of the application quickly without forcing a full recompile of the entire application. But it is not a "free ride" - this model poses new challenges to application developers and operations teams as many assumptions no longer hold true.

The recent rise of containerization has directly contributed to the uptake of microservices, as it is now very easy to quickly spin up new, lightweight run-time environment(s) for the application.

The ability to provide single-purpose components with clean APIs between them is an essential design requirement for microservices architecture. At their core, microservices have two main characteristics: they are stateless and distributed. To achieve this, let's take a closer look at the Twelve-Factor App methodology in more detail to help explain microservices architecture as a whole.

The Twelve-Factor App [27]



To allow the developer maximum flexibility in their choice of programming languages and back-end services, Software-as-a-Service web applications should be designed with the following characteristics:

- Use of a declarative format to attempt to minimize or eliminate side effects by describing what the program should accomplish, rather than describing how to go about it. At a high level it's the variance between a section of code and a configuration file.
- Clean Contract with the underlying Operating Systems which enables portability to run and execute on any infrastructure. API's are commonly used to achieve this functionality.
- Ability to be deployed into modern cloud platforms; removing the dependencies on underlying hardware and platform.
- Keep development, staging, and production as similar as possible. Minimize the deviation between the two environments for continuous development.
- Ability to scale up (and down) as the application requires without needing to change the tool sets, architecture or development practices.

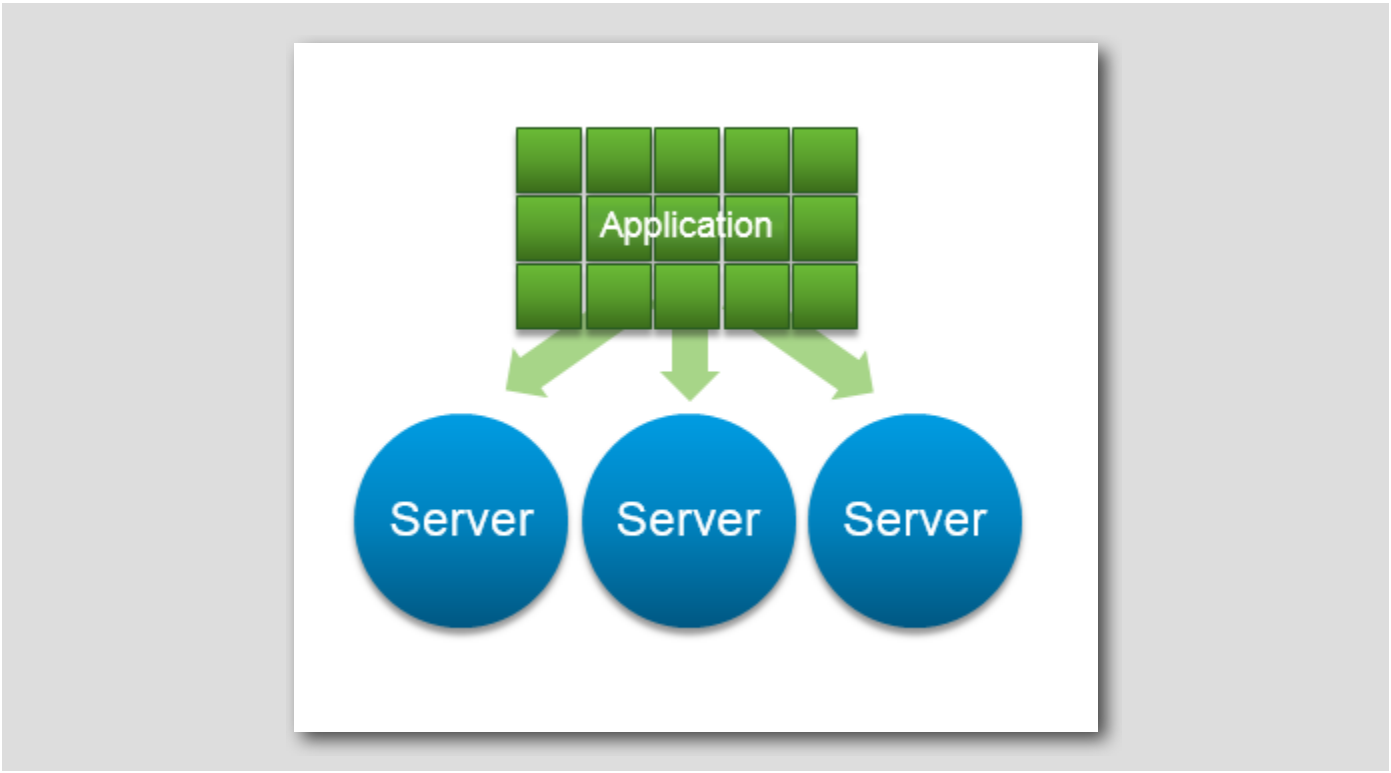
At a high level, the 12 Factors that are used to achieve these characteristics are:

1. Codebase - One codebase tracked in revision control, many deploys
2. Dependencies - Explicitly declare and isolate dependencies
3. Config - Store config in the environment
4. Backing Services - Treat backing services as attached resources
5. Build, release, run - Strictly separate build and run stages
6. Process - Execute the app as one or more stateless processes
7. Port Binding - Export services via port binding
8. Concurrency - Scale out via the process model
9. Disposability - Maximize robustness with fast startup and graceful shutdown
10. Dev/Pro Parity - Keep development, staging, and production as similar as possible
11. Logs - Treat logs as event streams
12. Admin Process - Run admin/management tasks as one-off processes

For additional detailed information on these factors, check out *12factor.net*.

Benefits of Microservices

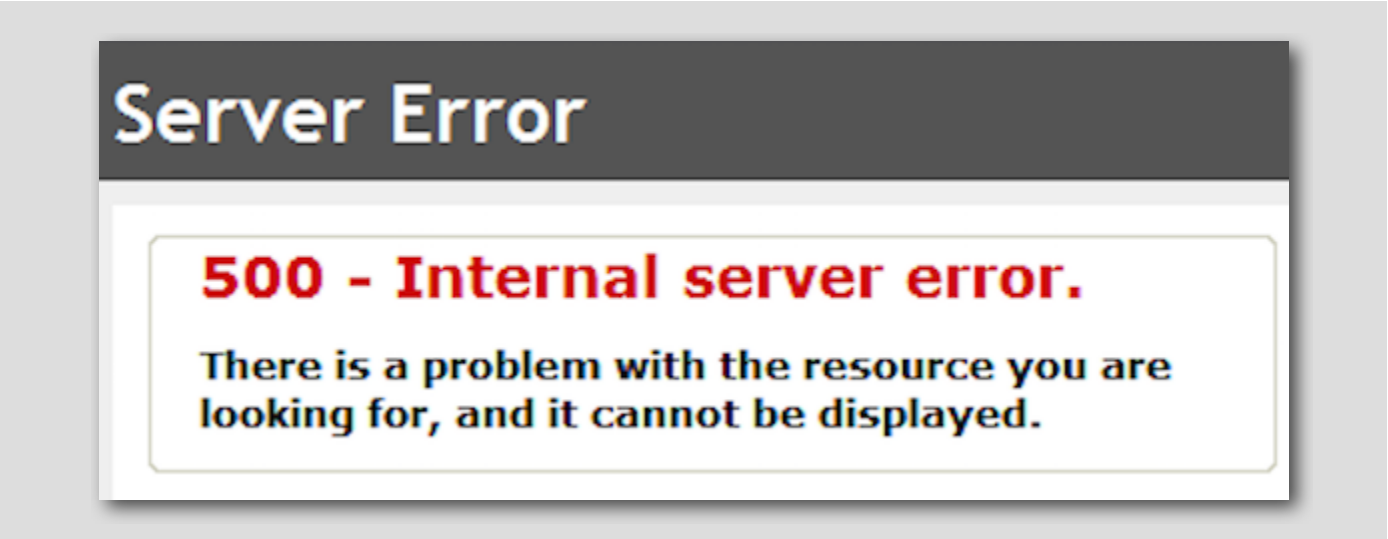
[28]



Microservice architecture has benefits and challenges. If the development and operating models in the company do not change, or only partially change, things could get muddled very quickly. Decomposing an existing app into hundreds of independent services requires some choreography and a well thought-out plan. So why are teams considering this move? Because there are considerable benefits!

Resilience

[29]



Note the above image is showing a example of a service failure and is not a error in the manual.

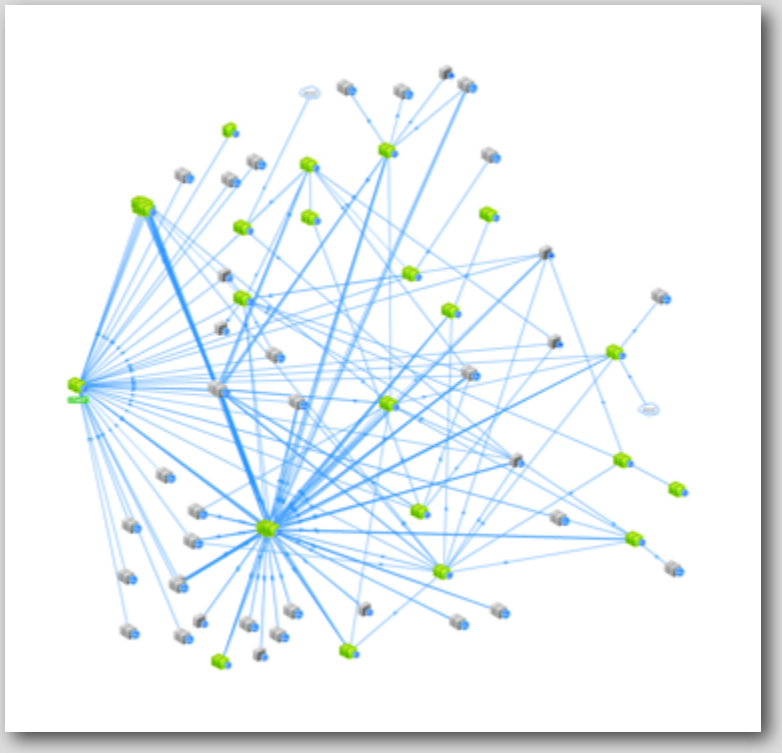
With a properly architected microservice-based application, the individual services will function similarly to a bulkhead in a ship. Individual components can fail, but this does not mean the ship will sink. The following tenet is held closely by many development teams - "Fail fast, fail often." The quicker a team is able to identify a malfunctioning module, the faster they can repair it and return to full operation.

Consider an online music player application - as a user, I might only care about playing artists in my library. The loss of the search functionality may not bother me at all. In the event that the Search service goes down, it would be nice if the rest of the application stays functional. The dev team is then able to fix the misbehaving feature independently of the rest of the application.

Defining "Service Boundaries" is important when architecting a microservice-based application!

Scaling

[30]



If a particular service is causing latency in your application, it's trivial to scale up instances of that specific service if the application is designed to take full advantage of microservices. This is a huge improvement over monolithic applications.

Similar to the Resilience topic, with a monolithic application, one poorly-performing component can slow down the entire application. With microservices, it is almost trivial to scale up the service that is causing the latency. Once again, this scalability must be built into the application's DNA to function properly.

Deployment

[31]

Once again, microservices allow components to be upgraded and even changed out for entirely new, heterogeneous pieces of technology without bringing down the entire application. Netflix pushes updates constantly to production code in exactly this manner.

Misbehaving code can be isolated and rolled back immediately. Upgrades can be pushed out, tested, and either rolled back or pushed out further if they have been successful.

Organizational

[32]

"Organizations which design systems ... are constrained to produce designs which are copies of the communication structures of these organizations" --Melvin Conway

The underlying premise here is that the application should align to the business drivers, not to the fragmentation of the teams. Microservices allow for the creation of right-sized, more flexible teams that can more easily align to the business drivers behind the application. Hence, ideas like the "two pizza rule" in which teams should be limited to the number of people that can finish two pizzas in a sitting (conventional wisdom says this is eight or less...though my personal research has proved two pizzas do not feed more than four people.)

No Silver Bullet!

[33]

Microservices can be accompanied by additional operations overhead compared to the monolithic application provisioned to an application server cluster. When each service is separately built out, they could each potentially require clustering for fail over and high availability. When you add in load balancing, logging and messaging layers between these services, the real-estate starts to become sizable even in comparison to a large off the shelf application. Microservices also require a considerable amount of DevOps and Release Automation skills. The responsibility of ownership of the application does not end when the code is released into production, the Developer of the application essentially owns the application until it is retired. The natural evolution of the code and collaborative style in which it is developed can lend itself to challenges when making a major change to the components of the application. This can be partially solved with backwards compatibility but it is not the panacea that some in the industry may claim.

Microservices can only be utilized in certain use cases and even then, Microservices open up a world of new possibilities that come with new challenges and operational hurdles. How do we handle stateful services? What about orchestration? What is the best way to store data in this model? How do we guarantee a data persistence model? Precisely how do I scale an application properly? What about "simple" things like DNS and content management? Some of these questions do not have definitive solutions yet. A distributed system can also introduce a new level of complexity that may not have been such a large concern like network latency, fault tolerance, versioning, and unpredictable loads in the application. The operational cost of application developers needing to consider these potential issues in new scenarios can be high and should be expected throughout the development process.

When considering the adoption of a Microservices, ensure that the use case is sound, the team is aware of the potential challenges and above all, the benefits of this model outweigh the cost.

Recommended reading: If you would like to learn more about the operational and feasibility considerations of Microservices, look up ***Benjamin Wootton*** *and read some of his publications on the topic, specifically 'Microservices - Not A Free Lunch!'.*

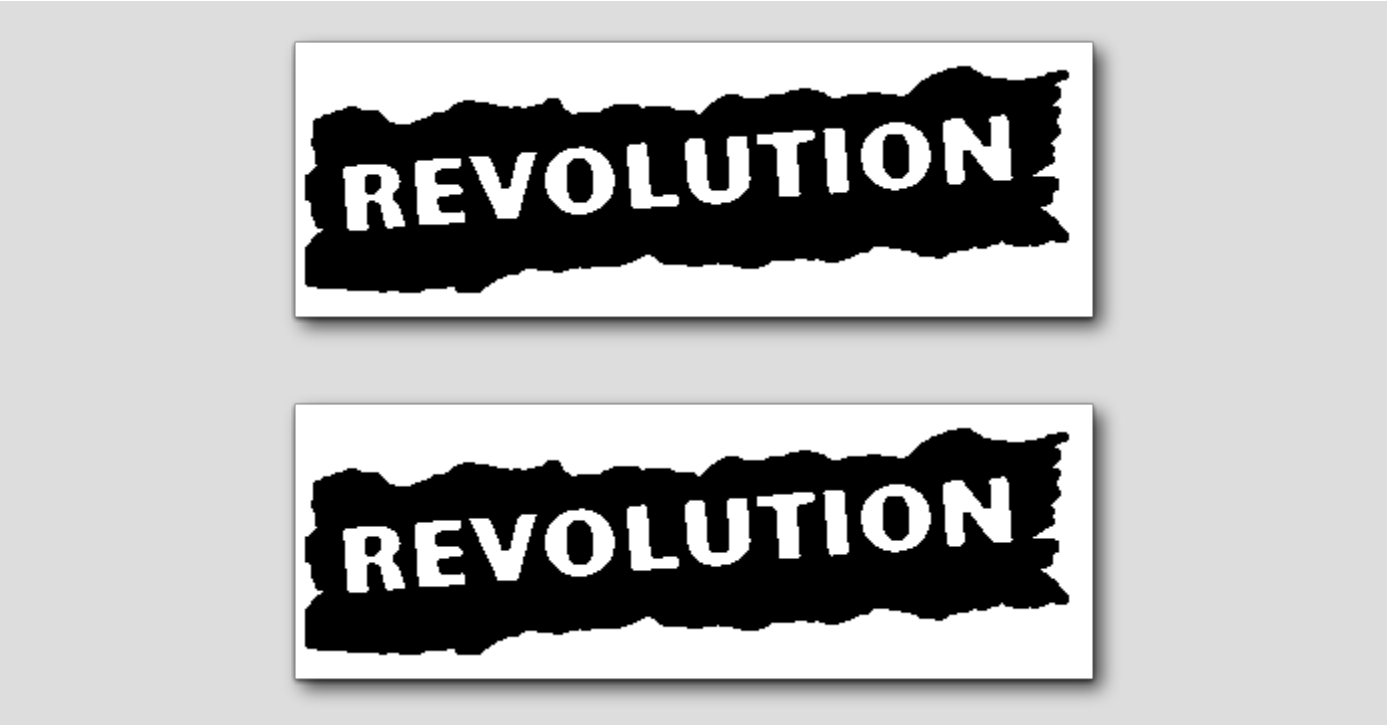
The Docker Revolution

[34]

In this lesson we will discuss the tremendous impact containers are having on the IT industry with the benefits and challenges.

Notice: This section is all reading. If you are eager to get to typing on the keyboard, you can move on to the next module, but you will

miss out on learning some very important concepts



What Just Happened?

The reason you're taking this lab is probably that you've heard talk of containers swirling around and are eager to understand what all the buzz is about. Let's take a minute to try to understand what has happened over the last few years.

Revenge of the Developer

Some of the most successful startups today are driven by developers and coders rather than business people. This is a radical change from how things used to be done. The business used to come up with ideas and push them down to development to have them translated into code. That is about to change and has already changed in some industries. Developers now constantly push the boundaries of what's possible and send their ideas to the business who builds offerings around them. This shift fundamentally changes how IT operates and has driven the demand for new tools and technologies.

From Months to Minutes

Under the old paradigm, releases once every quarter was considered frequent. Today, some industries need to make several releases an hour to keep up. It would be impossible to roll out all of these changes to a system that's constantly used by millions of people and that doesn't tolerate any downtime. Frequent changes means smaller changes, which means easier rollouts. In addition, if you keep the changes well encapsulated and independent from each other, they become much easier to roll back if something should go wrong.

It's been reported that the big social media and web portal companies deploy hundreds of changes every day. Clearly, something in the way we build and deploy code has to change fundamentally!

DevOps and CI/CD

To get to the speed and agility needed, we need to change our processes fundamentally.

You may have heard of DevOps. This is the idea that we can no longer have developers simply throwing code over the fence to operations and hope that it gets deployed. It's too slow and error prone. Instead we need to streamline the handover and even eliminate it. DevOps is the idea of integrating development and operations so they behave as one unit.

Sounds good, but how do we get there? One way is to implement a Continuous Integration / Continuous Deployment process. The idea is that whenever a developer commits a piece of code and tags it as ready for release, the deployment process automatically kicks off. In some cases, this process can be made completely automatic by introducing automated testing and deployment.

The challenge to implementing a CI/CD pipeline is that it requires you to set up and tear down complete environments very quickly.

Docker to the Rescue!

Docker is no silver bullet, but it offers some attractive features helping anyone who is trying to increase speed and agility:

- Containers are fast to deploy and undeploy.
- The container form factor is convenient for shipping small changes.
- It's relatively easy to use containers as part of a CI/CD pipeline.
- Container schedulers such as Kubernetes facilitate low-impact upgrades.
- Auto-scaling becomes easier, since containers can be created and destroyed very quickly.

The Challenges

So why isn't everyone doing this right now? Because problems arise when you try to operationalize this model. By default, docker does not contain the tools necessary for monitoring and debugging containers and most IT operations organizations lack the knowledge and processes for monitoring, troubleshooting and handling containers. Some of the challenges IT operations is struggling with are:

- Isolation between containers can be weaker than between virtual machines.
- Backup and restore of containers is different than virtual machines and requires different tools and processes.

- Network virtualization and security are in the early stages of development.
- Kubernetes is great for orchestration and scheduling but can be difficult to manage.

To assist IT operations in addressing these problems VMware introduced the Tanzu portfolio of products.

VMware Tanzu is a portfolio of products and solutions that allow customers to Build, Run, and Manage Kubernetes controlled container-based applications. With VMware Tanzu, you can ready your infrastructure for modern apps with consistent, conformant Kubernetes everywhere. Provide a self-service, compliant experience for developers that clears their path to production. Then centrally manage, govern and observe all clusters and apps across clouds. It' s that simple.

You can learn more about VMware Tanzu here: *https://tanzu.vmware.com/tanzu*

Conclusion

[36]

Congratulations on completing Module 1. You should now have a solid foundational understanding of container technologies and the docker implementation.

VMware provides a host of Modern Apps FREE learning resources hosted on our *Tanzu website*.

You have finished Module 1

[37]

Congratulations on completing Module 1.

Based on your interests, please proceed to any module below:

- Module 2 - A Quick Tour of Docker - (15 minutes) (Basic) In this module, we're getting hands on with docker and explore how to create simple containers and how to leverage docker container networking. Finally, we'll look at how you can build your own images using a Dockerfile.
- Module 3 - Introduction to Kubernetes - (30 minutes) (Basic) In this module we will explain the place where container management fits and take a high level walk thru the Kubernetes architecture. We will walk through some worked examples using a kubernetes instance deployed on Tanzu Kubernetes Grid (TKG).
- Module 4 - Introduction to Harbor - (30 minutes) (Advanced) In this module we will walk through some of the capabilities of the Harbor secure image repository and explore some worked examples using images pre-populated into this labs Harbor instance
- Module 5 - Introduction to Tech Zone VMware Learning Platform & Kube Academy - (15 minutes) (Basic) In this module we will introduce some of the many FREE learning resources which VMware provides for users looking to start their journey into modern apps and kubernetes or looking to build on the knowledge and experience they already have.

From here you can:

1. Click to advance to the next page and continue with the next lab module
2. Open the TABLE OF CONTENTS to jump to any module or lesson in this lab manual
3. End your lab and come back and start it again in the future

Module 2 - A Quick Tour of Docker (15 minutes) Basic

A Quick Tour of Docker

[39]

In this lesson we will cover the following:

- Starting a simple container
- Running a simple service as a container
- Container Networking

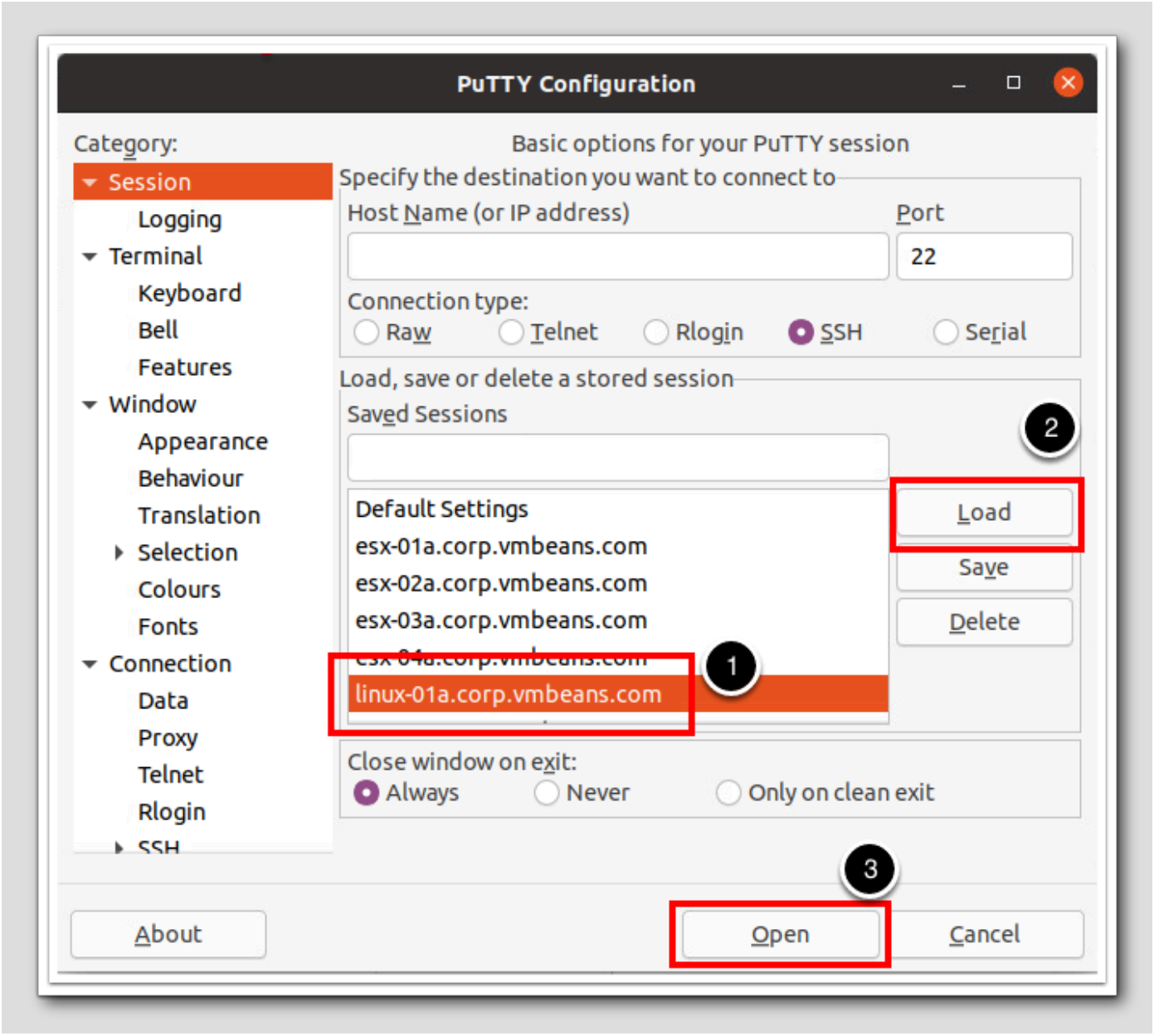
Connect to An Ubuntu Image Running Docker

[40]

1. On the taskbar - double click on the **Putty** icon if it is not already open.

For this lab we have pre-configured a docker image on an Linux image appropriately called "Linux-01a". If you would like to experiment with docker it can be loaded on most laptop or desktop machines, even a machine as small as a Raspberry Pi. See *https://docs.docker.com.*

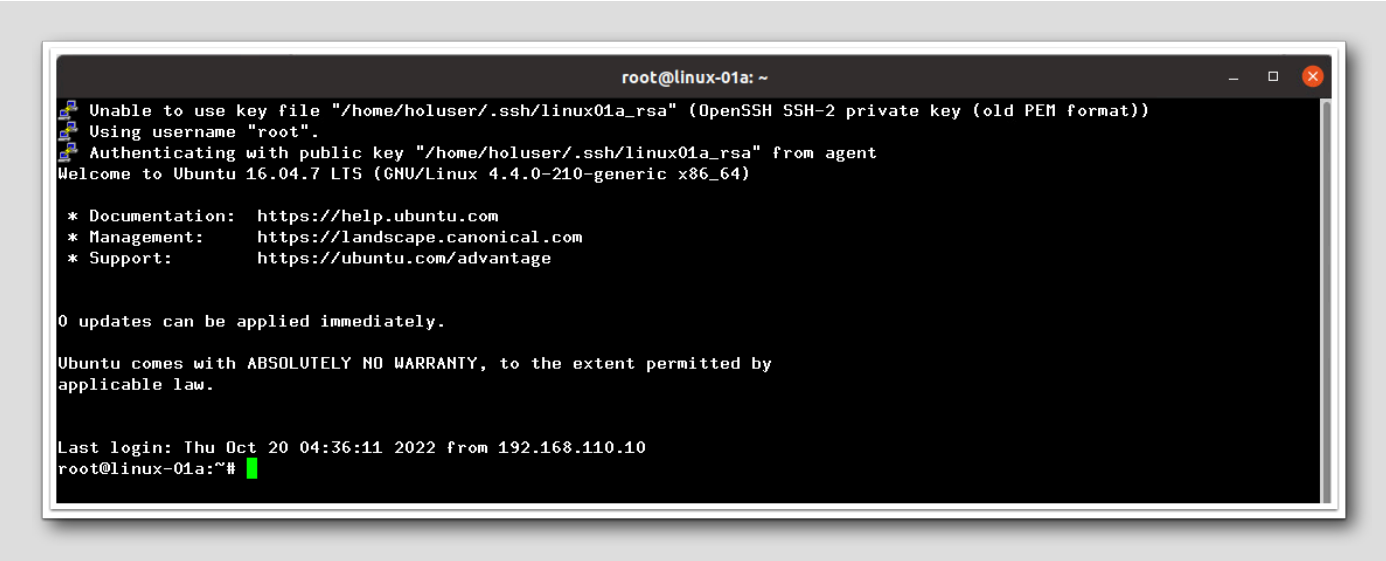
Note: *For security, access to the external internet is limited in the Hands-on Labs environment. You will need another device to access this link to download and install docker.*


PuTTY Configuration
Category:
Session
Logging
Terminal
Keyboard
Bell
Features
Window
Appearance
Behaviour
Translation
Selection
Colours
Fonts
Connection
Data
Proxy
Telnet
Rlogin
SSH
Basic options for your PuTTY session
Specify the destination you want to connect to
Host Name (or IP address)
Port
22
Connection type:
Raw
Telnet
Rlogin
SSH
Serial
Load, save or delete a stored session
Saved Sessions
Default Settings
esx-01a.corp.vmbeans.com
esx-02a.corp.vmbeans.com
esx-03a.corp.vmbeans.com
linux-01a.corp.vmbeans.com
Load
Save
Delete
Close window on exit:
Always
Never
Only on clean exit
About
Open
Cancel
1
2
3

Login using SSH to Linux-01a VM:

1. Click on **linux-01a.corp.vmbeans.com** under Saved Sessions.
2. Click the **Load** button.
3. Click **Open**.

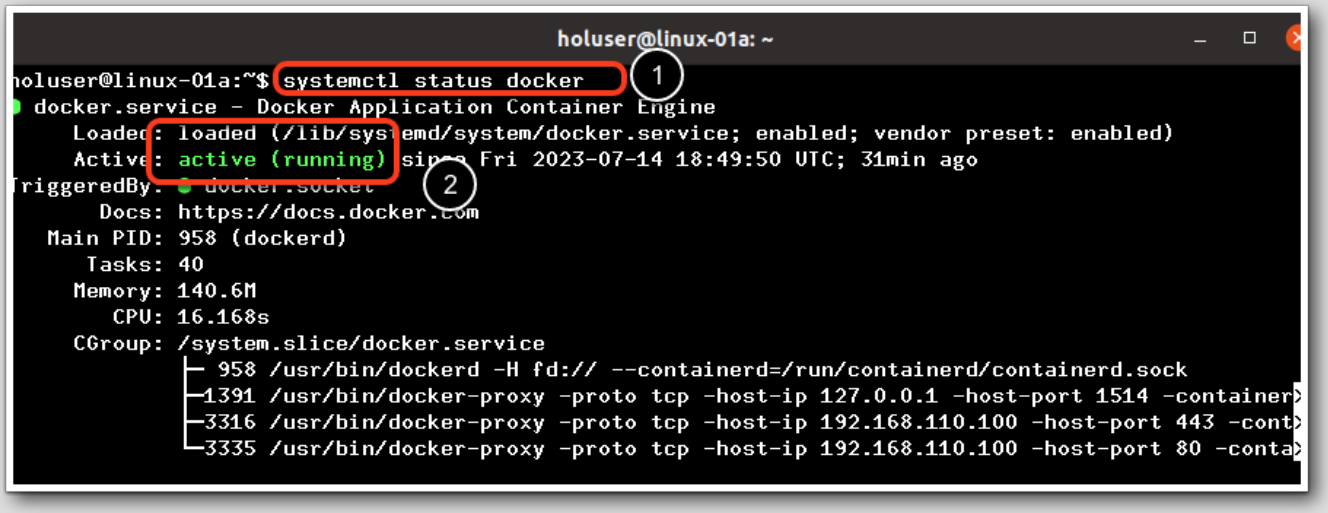
After a few seconds, you should be logged in as the **root** user. You should not be asked for a password as that is handled through key based authentication.



After a few seconds, you should be logged in as the **root** user. You should not be asked for a password as that is handled through key based authentication.

Verify Docker

[41]

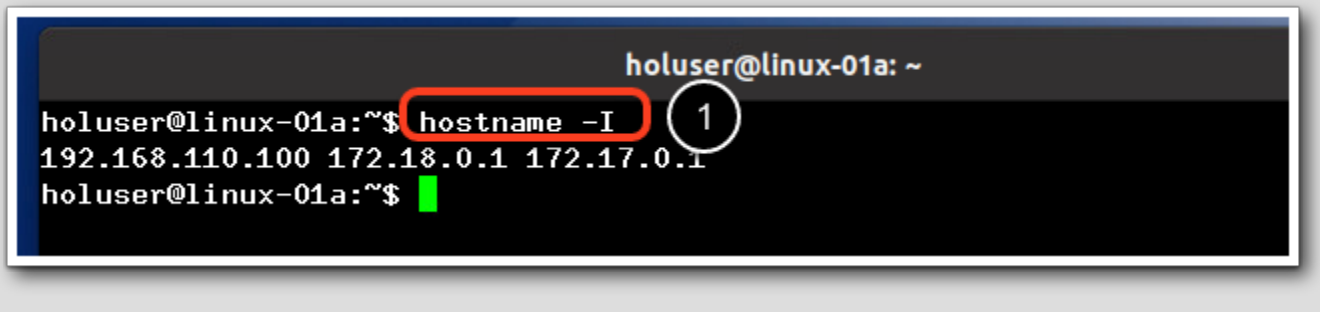


Verify that docker is Running and Active with the following command:

1. Type `systemctl status docker` and press enter
2. Verify the docker system is **active**
3. **Press Ctrl-C** to return to the prompt.

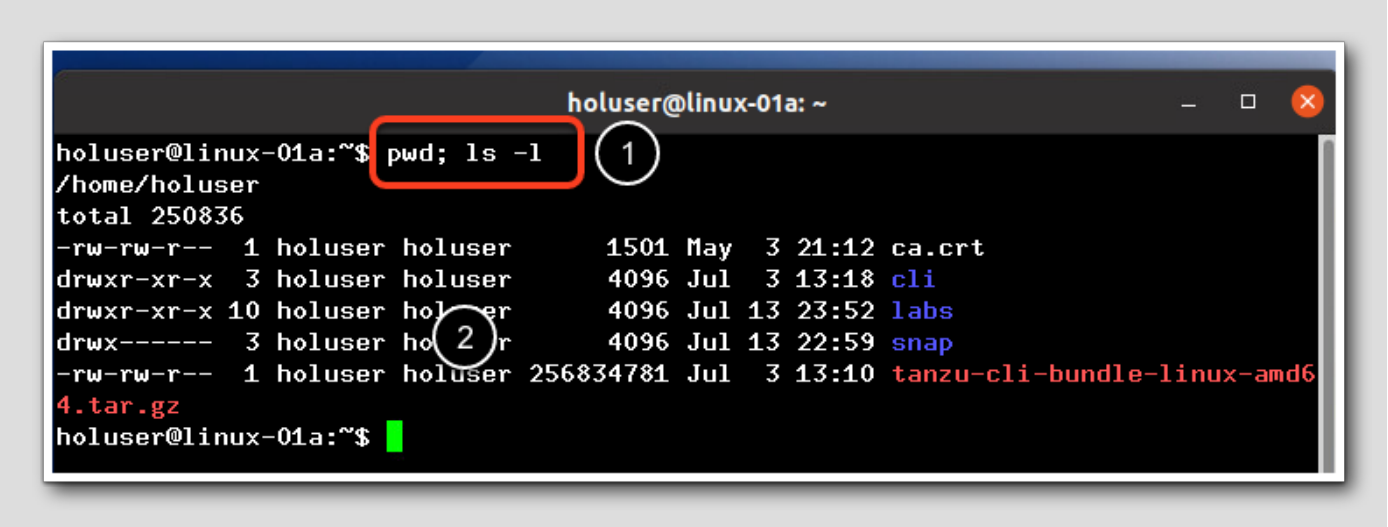
Starting our first container

[42]



At the command line, type the following command followed by Enter:

1. `hostname -I` and take note of the output.



Check the current directory and the contents of the local directory:

1. `pwd;ls -l`

Note the current directory and the contents of the file system.

In linux you can have multiple commands on the same line separated by a semi-colon (;) . Linux will execute the first command, then the next, etc…

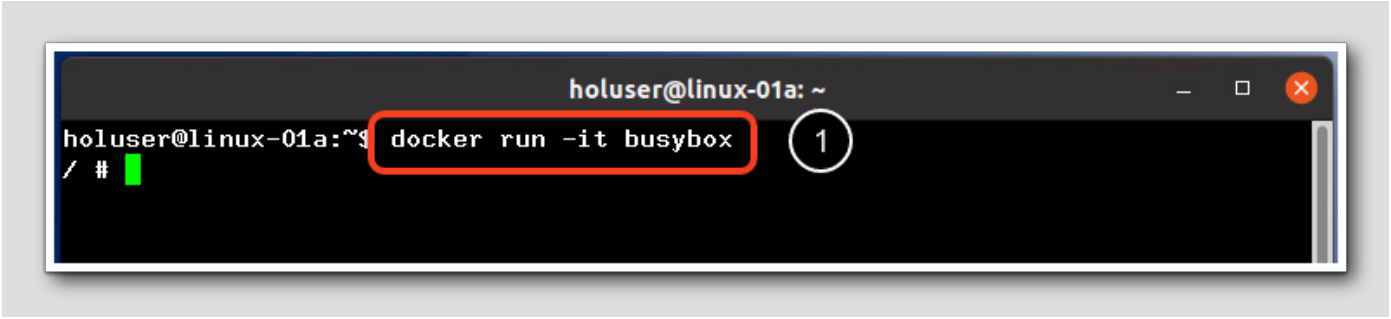
We could achieve the same result by typing the commands on separate lines

pwd

ls

Start Our First Docker Container

[43]



At the command prompt, type the command below, followed by Enter:

1. `docker run -it busybox`

Congratulations ! You have now created your first container. The command above created a docker container with the Linux "busybox" application. BusyBox is a Linux utility that combines tiny versions of many common UNIX utilities into a single small executable.

Note how quickly the container was created, this is due to the fact that containers are much lighter weight then virtual machines. Also note that docker did not need to download the busybox application. This is due to the fact that the busybox application was previously downloaded and was cached by docker. We'll discuss docker images and the cache later but just make note that busybox launched immediately because it was cached.

To start an application inside of a container using docker you need to have a container image. The container image acts as a packaging mechanism for distributing an application, and includes the application software, operating system files, libraries and other software required by the application to run.

Prebuilt container images for applications, or a base container image upon which you may build your own container image, are distributed using image registries.

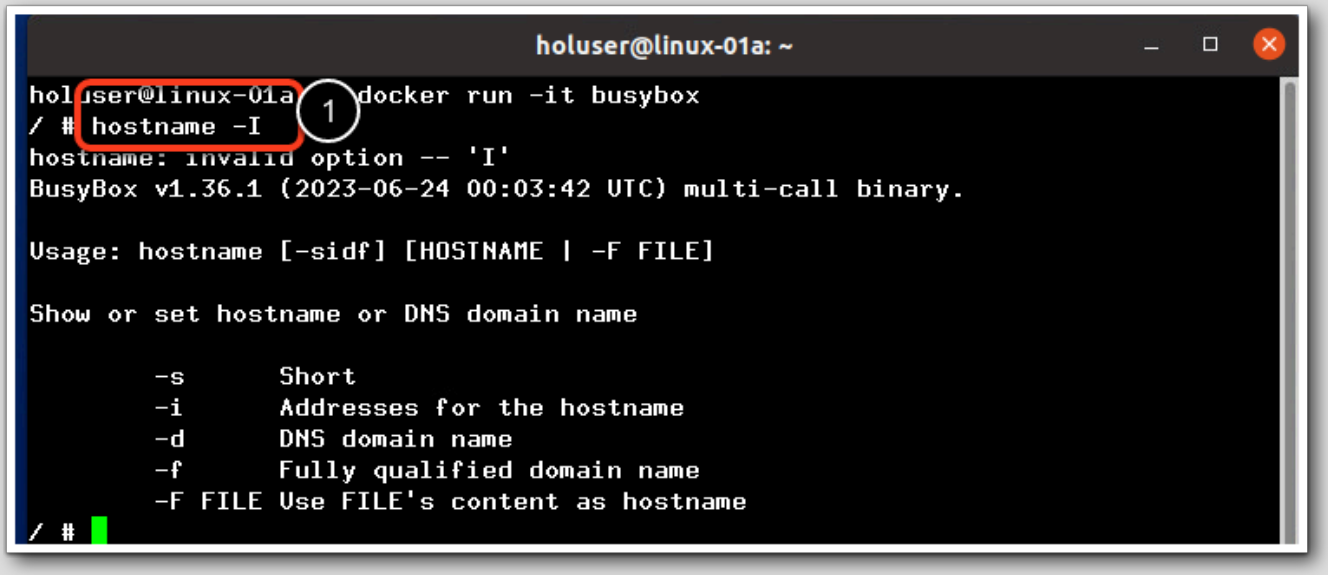
The two main hosted image registry services that exist are *Docker Hub* and *Quay.io*. Support for hosting and distributing container images is also available from Git repository hosting services such as *GitHub* and *GitLab*.

In this case, docker pulled the image for "busybox" from its image cache, created a container and kicked off "busybox". Busybox is a popular utility which combines tiny versions of many common UNIX utilities into a single small executable.

Note the Prompt (#). The -it flag to docker run caused docker to launch an image and start an interactive shell, the "#" is the shell prompt for the docker Busybox image.

Explore the world inside the container

[44]



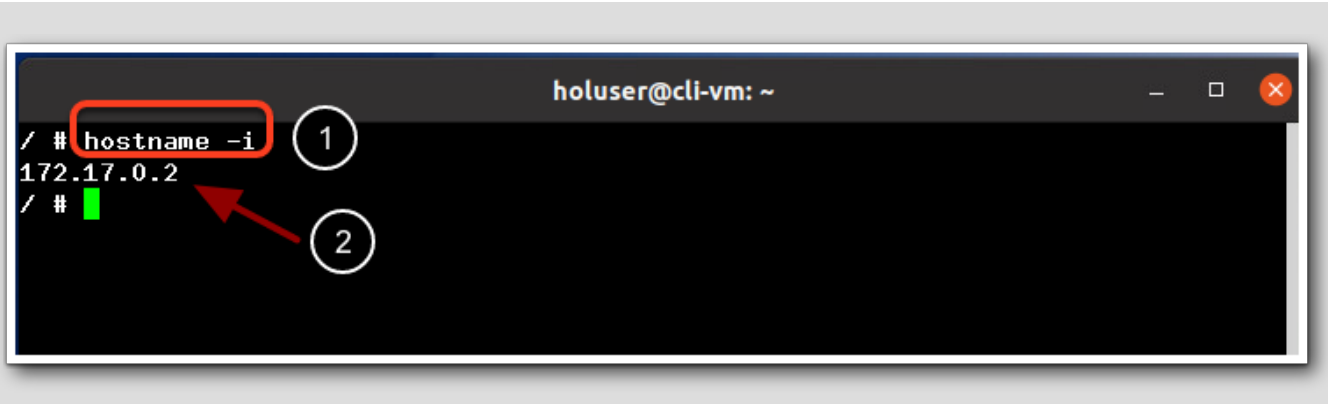
You are at a command prompt inside the container. Let's repeat the hostname command from the previous step:

1. `hostname -I`

And we get a error ??

This is due to the fact that we are running busybox inside a container and we are executing commands inside busybox.

Let's try hostname again.



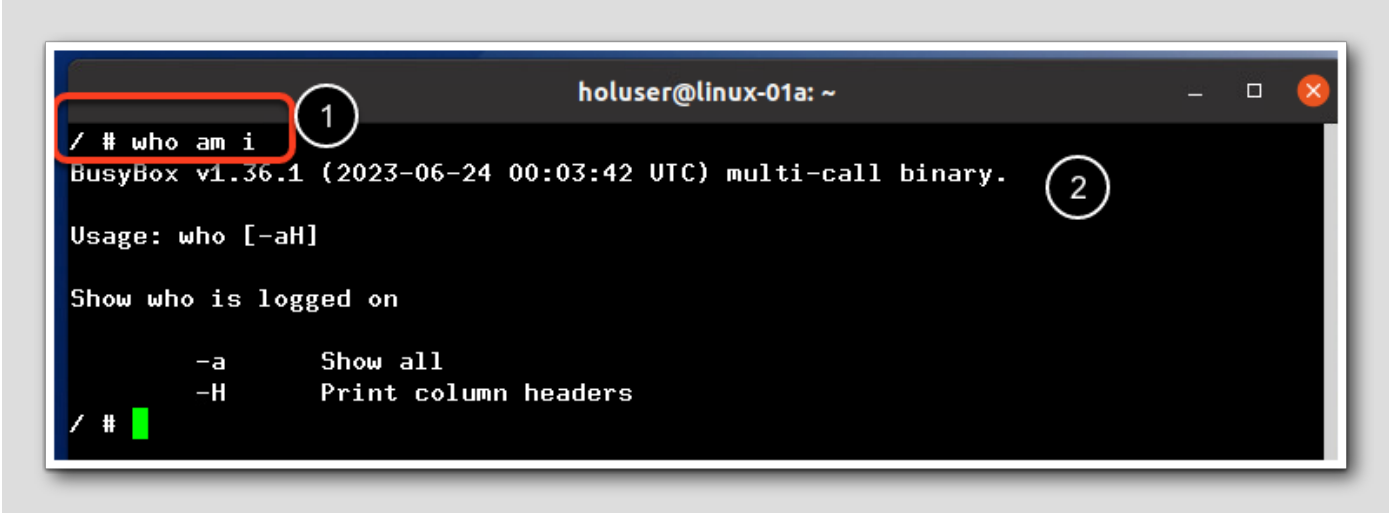
1. `hostname -i`

Note the address is different this is due to the fact that the docker container is using it's own internal network.

If I type :

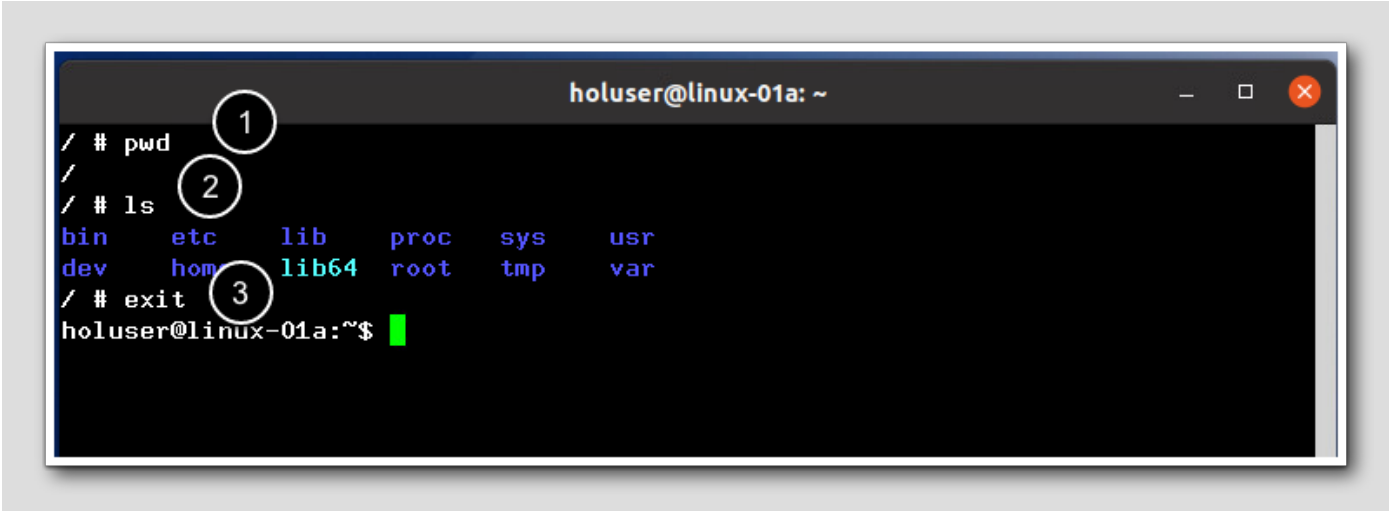
1. `who am i`

You will see you are running an instance of BusyBox.



Explore the environment inside BusyBox

[45]



Repeat the commands we typed before:

1. `pwd`
2. `ls`
3. `exit`

Note that the results are different then the results you received previously this is because busybox is running inside a docker container with its own filesystem.

1. Type **`exit`** followed by Enter. You are now back to the command prompt of the host VM.

What happened?

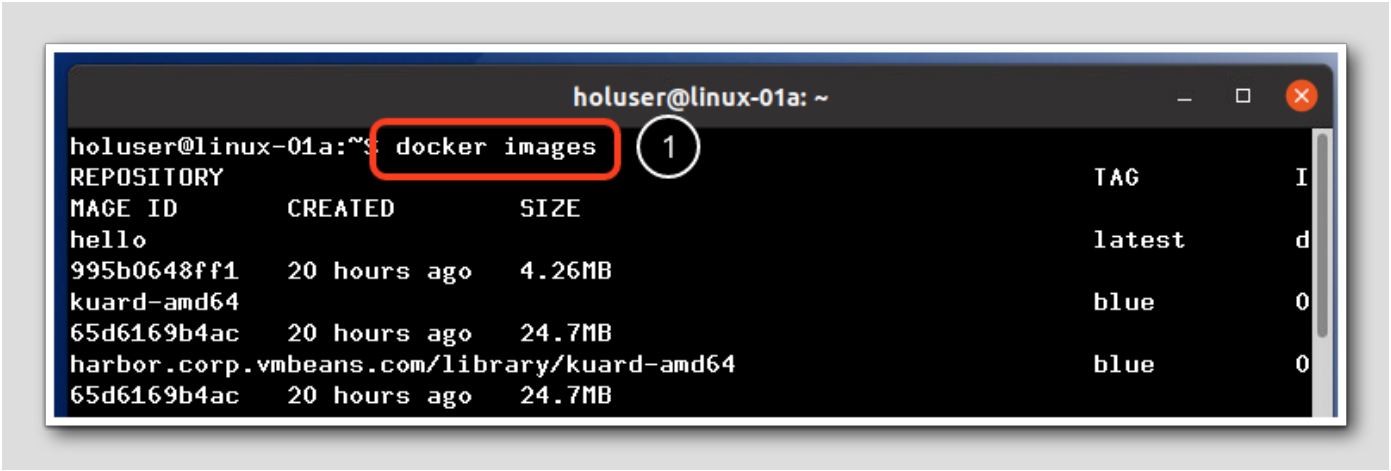
[46]

Congratulations! You just created and ran your first container! But what actually happened?

The command **`docker run -it busybox`** told docker to run a Busybox environment. This is a tiny Linux environment typically used for smaller devices, such as routers. If the docker image for busybox didn't exist in your local cache, it would have downloaded it. The `-it` flag simply meant that you want an interactive process with the current terminal connected to it.

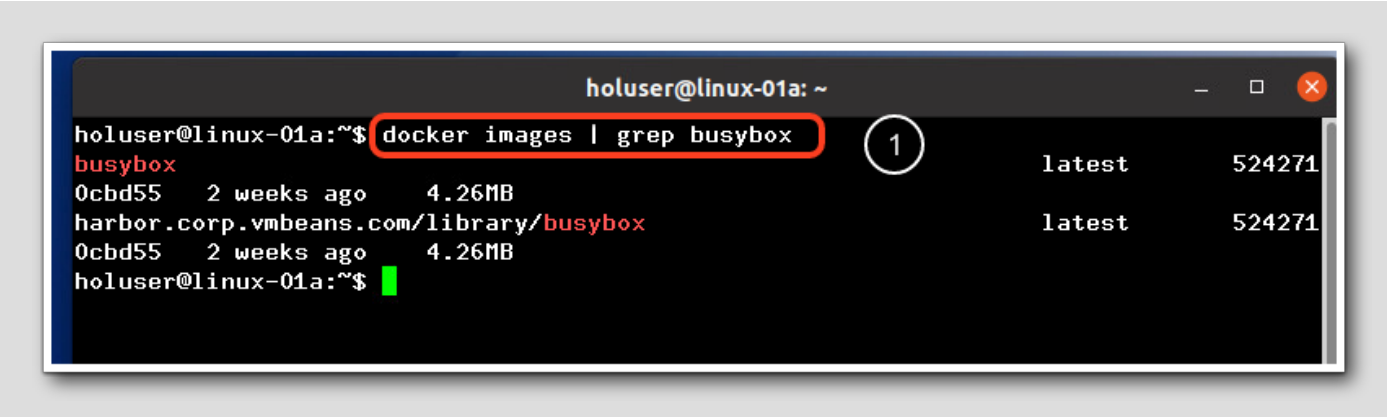
So this dropped you into the shell of busybox. You also noticed that you were given a different file system. This is because each docker container comes with its own file system that's completely isolated from the host. You also noticed you were connected to the 172.x.x.x network instead of the 192.168.110.100. This is because docker creates an ad-hoc virtual network for your containers so that ports you open in different containers won't interfere with each other.

As noted above, docker will look in its cache of images before looking to pull an image from a repository. You can see the images which docker has cached with the docker images command. Once you have returned to the main cli-vm prompt you can type:



1. `docker images`

To examine the images which docker has in its image cache.



You can view just the busybox image by typing:

1. `docker images | grep busybox`

If you are not familiar with "grep" , in Linux and Unix Systems Grep, short for global regular expression print, is a command used in searching and matching text files contained in the regular expressions. So we were able to view just the images that matched the text string "busybox".

Time to define some terms!

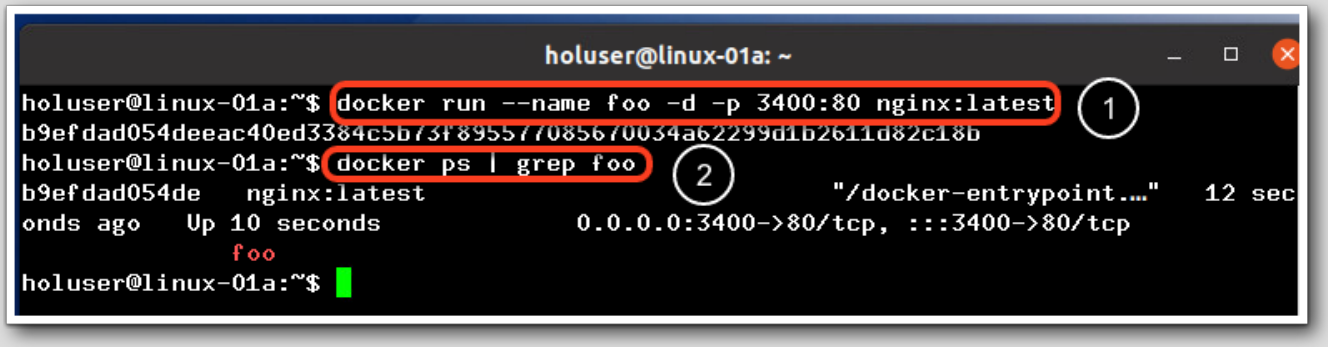
[47]

We've already used the terms container and image. Let's define them and some other concepts a bit more in detail.

- Image - The code you are running in a container packaged as a virtual file system known as a "layered file system". More about that later.
- Container - The running instance of an image. This is somewhat analogous to a virtual machine but is isolated at the OS level rather than using virtualized hardware. Each container has its own file system and virtual network.
- Repository - An image or set of images packaged in such a way that they can be published and reused.
- Registry - A catalog service for repositories. When you type `docker run -it busybox`, a query is sent to the registry service to locate the image(s) making up busybox. The registry server delivers the content of the images if they're not already in the local cache.
- Docker host - The machine (virtual and physical) hosting your containers.

Starting a service in a container

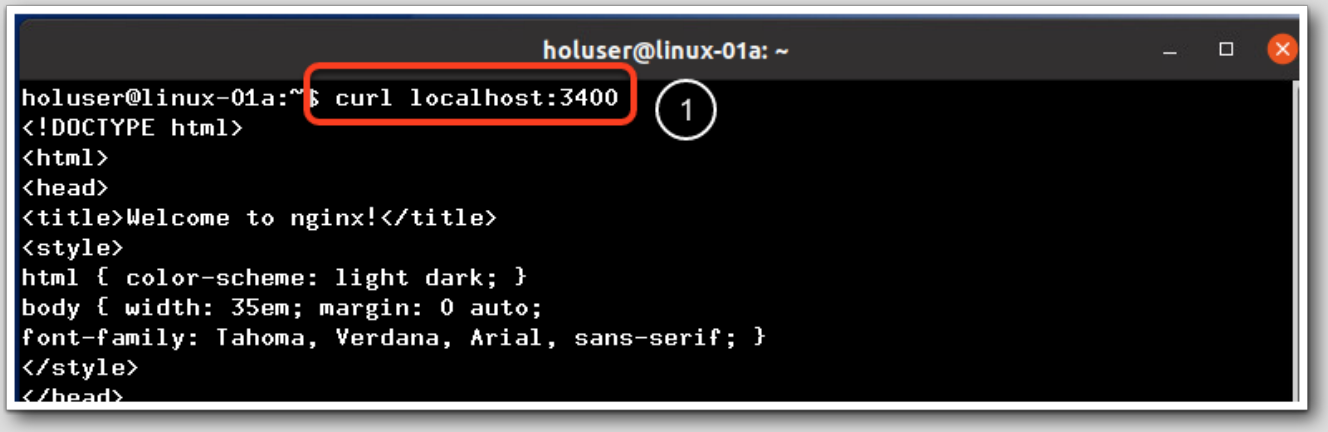
[48]



Let's get hands on again and do something a bit more useful. How about starting a nginx web proxy? That can be a bit of a daunting task when doing it in a traditional VM. With docker, it's a bit easier.

Let's do it. Type:

1. `docker run --name foo -d  -p 3400:80 nginx:latest`
2. You have now started an nginx web proxy! Now let's look at what's running. Type : **`docker ps | grep foo.`** You will see nginx running and listning on port 3400. When we executed the docker run command we specified port 3400:80 with the -p option.



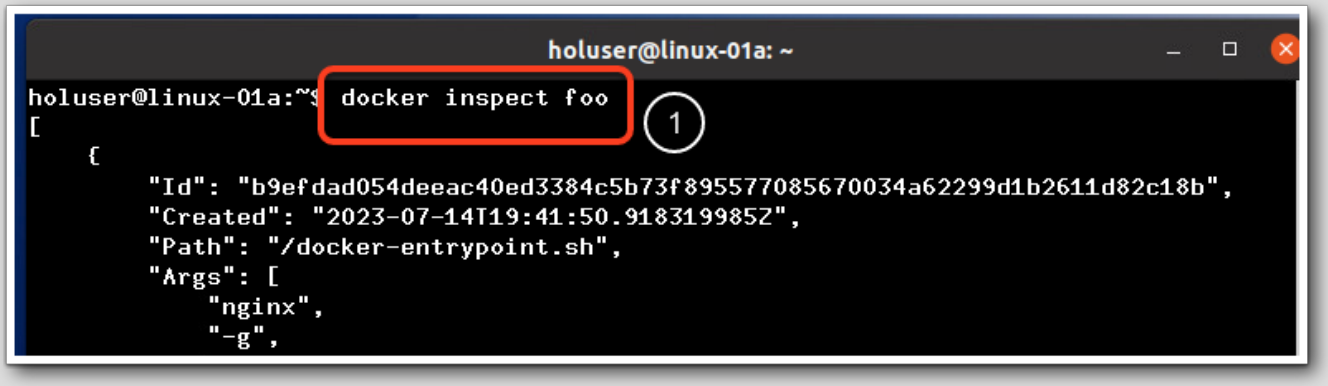
Let's see if it responds, Type this command, followed by enter:

`curl localhost:3400`

You should see a short HTML document printed in your terminal window.

Get More Details About Our Container

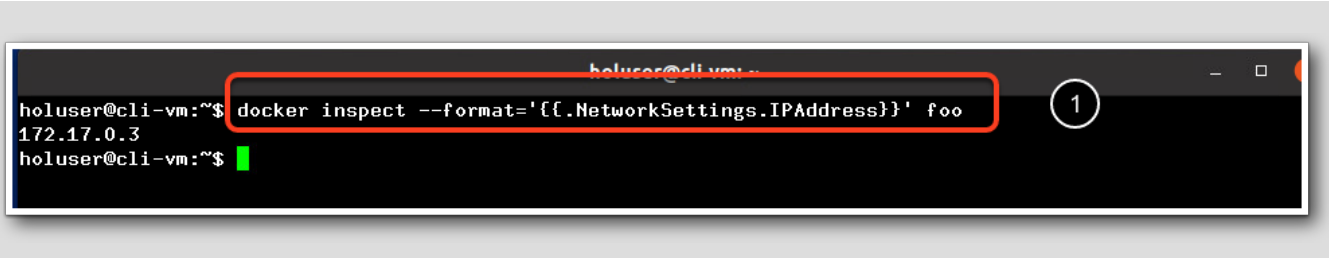
[49]



Let's get some in-depth details about our container and the image running inside. Type:

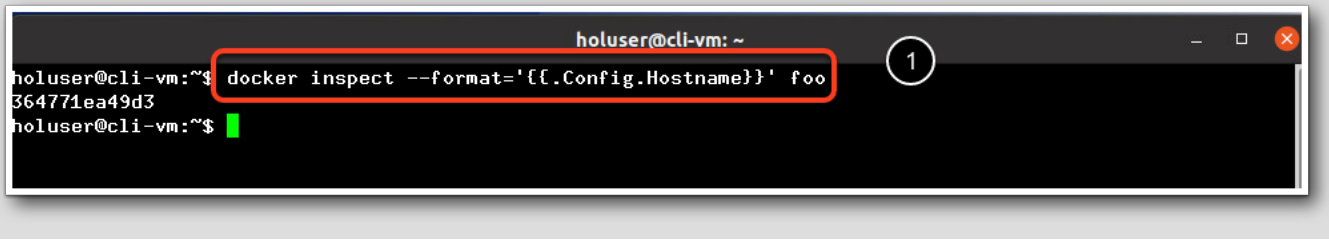
1. `docker inspect foo`

And take a minute to review the output. It's a lot of output so lets look at some ways to get more specific information with the `docker inspect` command.



1. `docker inspect --format='{{.NetworkSettings.IPAddress}}' foo`

This command will show just the IP address of the instance.



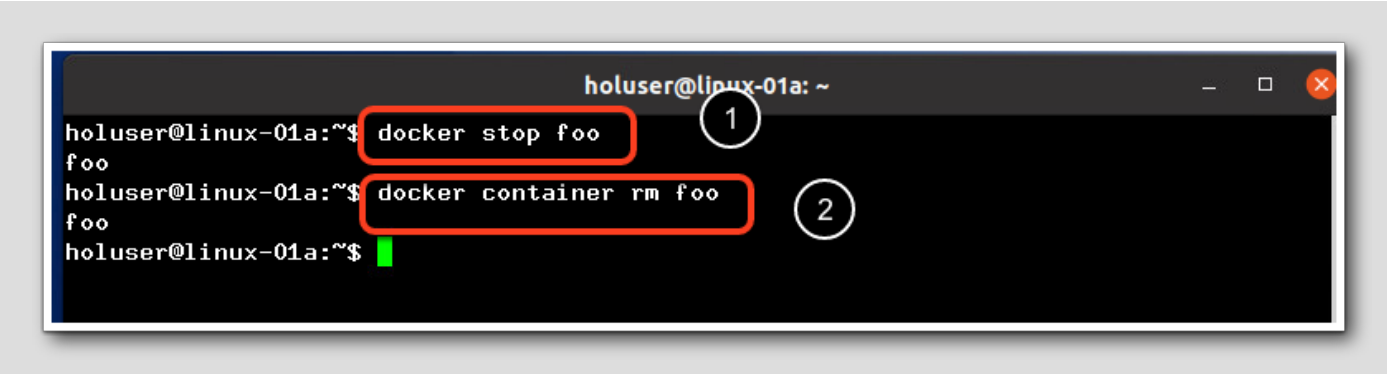
1. `docker inspect --format='{{.Config.Hostname}}' foo`

Will show the hostname.

There are many other docker inspect options available all of which are documented in the docker command line reference (*https://docs.docker.com/).*

Cleanup

[50]



Let's be good citizens and clean up !

1. `docker stop foo`
2. `docker container rm foo`

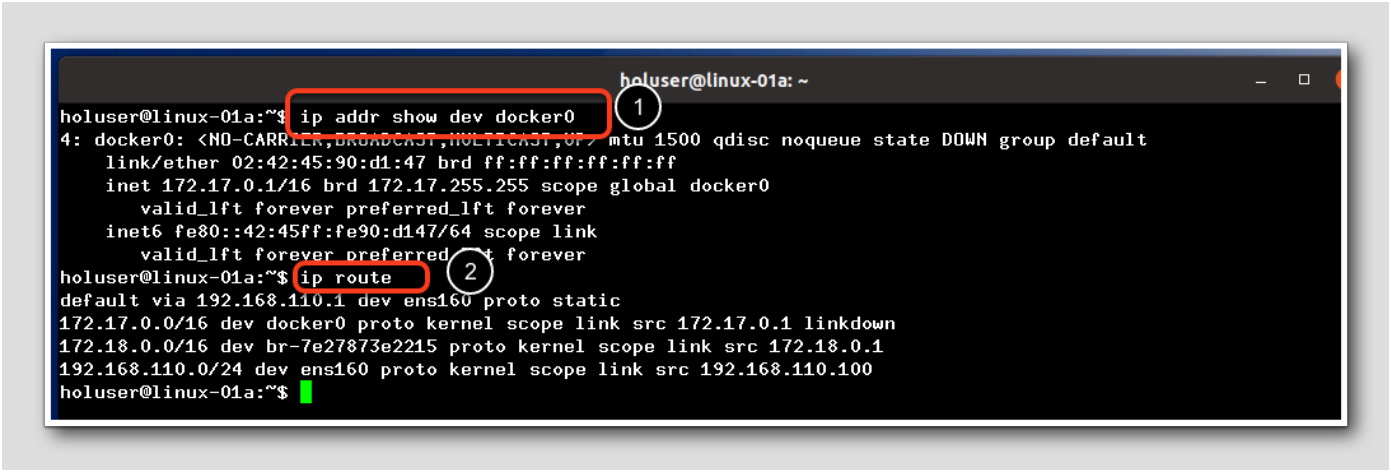
Docker stop foo terminates the running container named foo and docker container rm foo deletes the container named foo.

Daemons and port mappings

[51]

You probably noticed two things: You weren't dropped into the command prompt of the container and you could access the service on the http port on the docker host itself (localhost). It's all in the command options! Note that your IP addresses may be different then the screenshot above.

- -d means detached mode. In other words, the container is started as a background process and doesn't take over the terminal window.
- -p causes ports to the private container network to be mapped to ports on the docker host.
- -p 80:80 simply means "map port 80 on the private network to port 80 on the docker host".



Speaking of private container networks, let's have a quick look at how that works.

1. Type this command followed by Enter:

   `ip addr show dev docker0`

Docker has created a virtual network interface for the containers running on this host. Take note of the "inet addr" of this interface. It should be on the 172.x.x.x network.

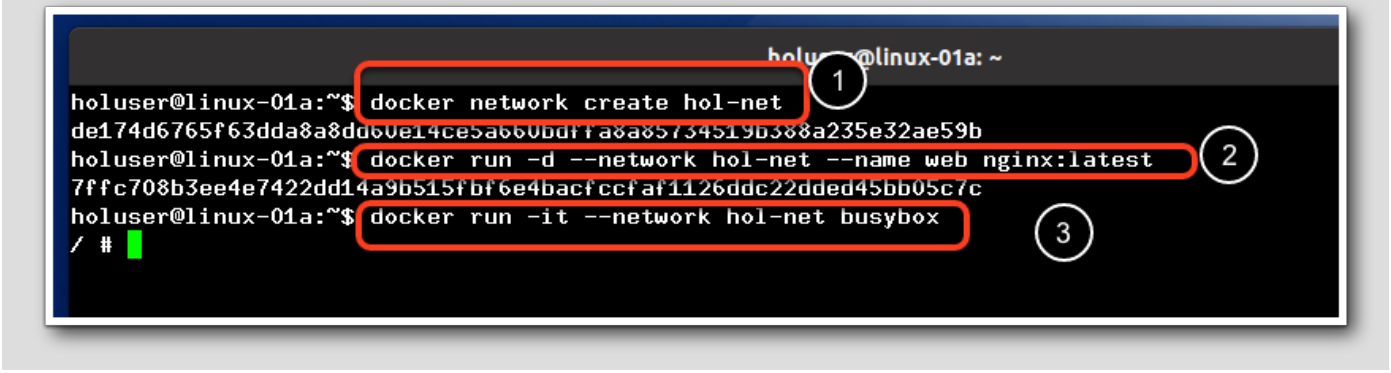
2. Type this command followed by Enter:

   `ip route`

This will display the routing table of the current host. Notice that you will see one or more 172.x.x.x addresses in this table. These entries were added by docker to inform the host how to route traffic to the private container network.

Creating a container network

[52]



We learned in the previous section how docker automatically creates a private container network for communication between your containers. But what if we want to create a separate network for a specific group of containers? That's where the **docker network create** command comes in handy!

Let's spin up a new network, then create a nginx proxy and a busybox and hook them both up to our newly created network.

1. Let's start with the container network. Type this command, followed by Enter:

   `docker network create hol-net`

2. We now have a new container network called hol-net. Let's start a nginx proxy and connect it to that network! Type this command followed by Enter:

   `docker run -d --network hol-net --name web nginx:latest` The **--name** parameter assigns a name to the container. This name can be used as a host name for containers wanting to connect to the container. We're calling the container "**web**".

3. Let's start the busybox! Type this command followed by Enter:

   `docker run -it --network hol-net busybox`

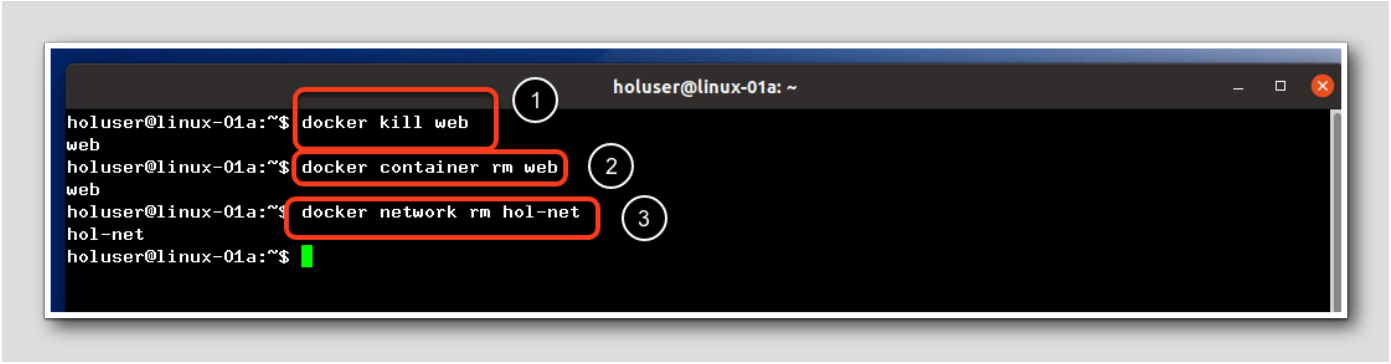
You will now drop into the shell of the busybox. Let's see if we can communicate with the nginx proxy. Remember we gave it the name "web", so we should be able to use that. Type this command, followed by enter:

1. **`ping -c 1 web`**

You should see a successful packet transmission.

Clean up

[53]



Let's clean up after this exercise and get ready for the next!

Type these three commands, each followed by Enter.

1. `exit`
2. Terminate the "web" container we just created. `docker kill web`
3. Delete the "web" container `docker container rm web`
4. Finally - finish by cleaning up the **hol-net** docker network that was created in step 1. `docker network rm hol-net`

The Dockerfile And Docker Build

[54]

So far, we've only been using pre-built images. But how were those images created in the first place? And what if you want to build your own customized image?

That's where the docker build command comes into the picture. This command reads a definition file, calls the Dockerfile and uses the instructions in the file to build a new image. You can see the Dockerfile as the source code of an image.

Let's build a simple application inside an image. Our "code" is the file "hello.sh" and our job is to build a container based on busybox, copying our application code to it and instruct Docker to run that code when the container is started.

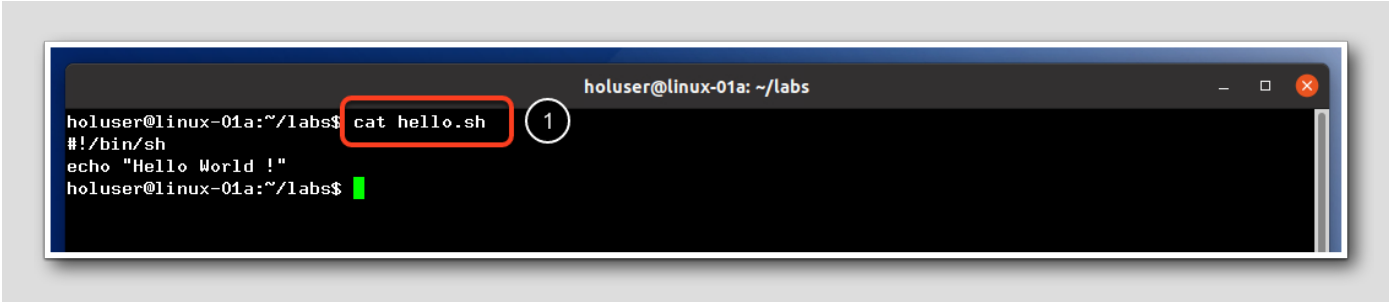
1. Examine the Dockerfile by typing the following two commands and pressing Enter after each line.

   `cd $HOME/labs`
2. `cat Dockerfile`
3. There are three directives in this Dockerfile:

   FROM - Tells docker which image to base this on. In our case, it's the busybox which was already placed in the docker image cache for us, if the image wasn't in the cache

   COPY - This copies the file hello.sh from our file system into the file system of the image.

   ENTRYPOINT - Tells docker which file to run when the container is started.

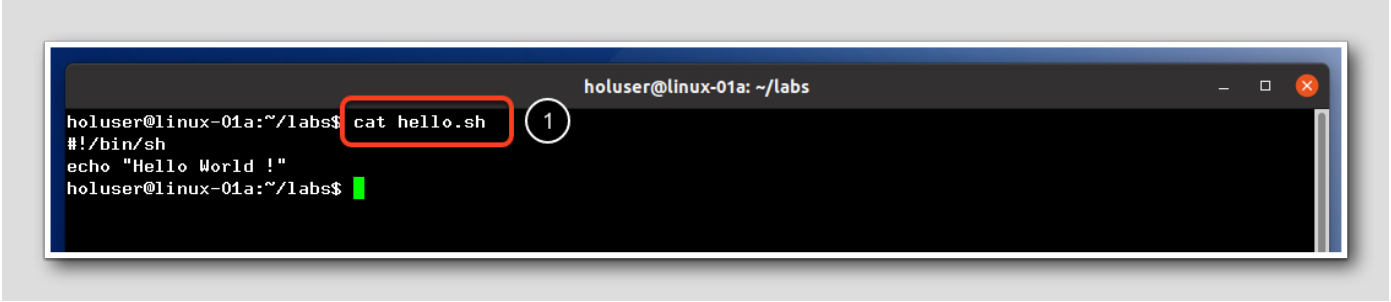


Let's examine the ENTRYPOINT:

1. `cat hello.sh`

This is a simple script that outputs "Hello World!".

As mentioned above, the Dockerfile is copying this file to the container in the second directive in Dockerfile. The third directive in Dockerfile is setting this script to run when this container is started. Hence, when a container is created with this Dockerfile it will start, print out "Hello World!" and then exit.



Connect to The Ubuntu Image

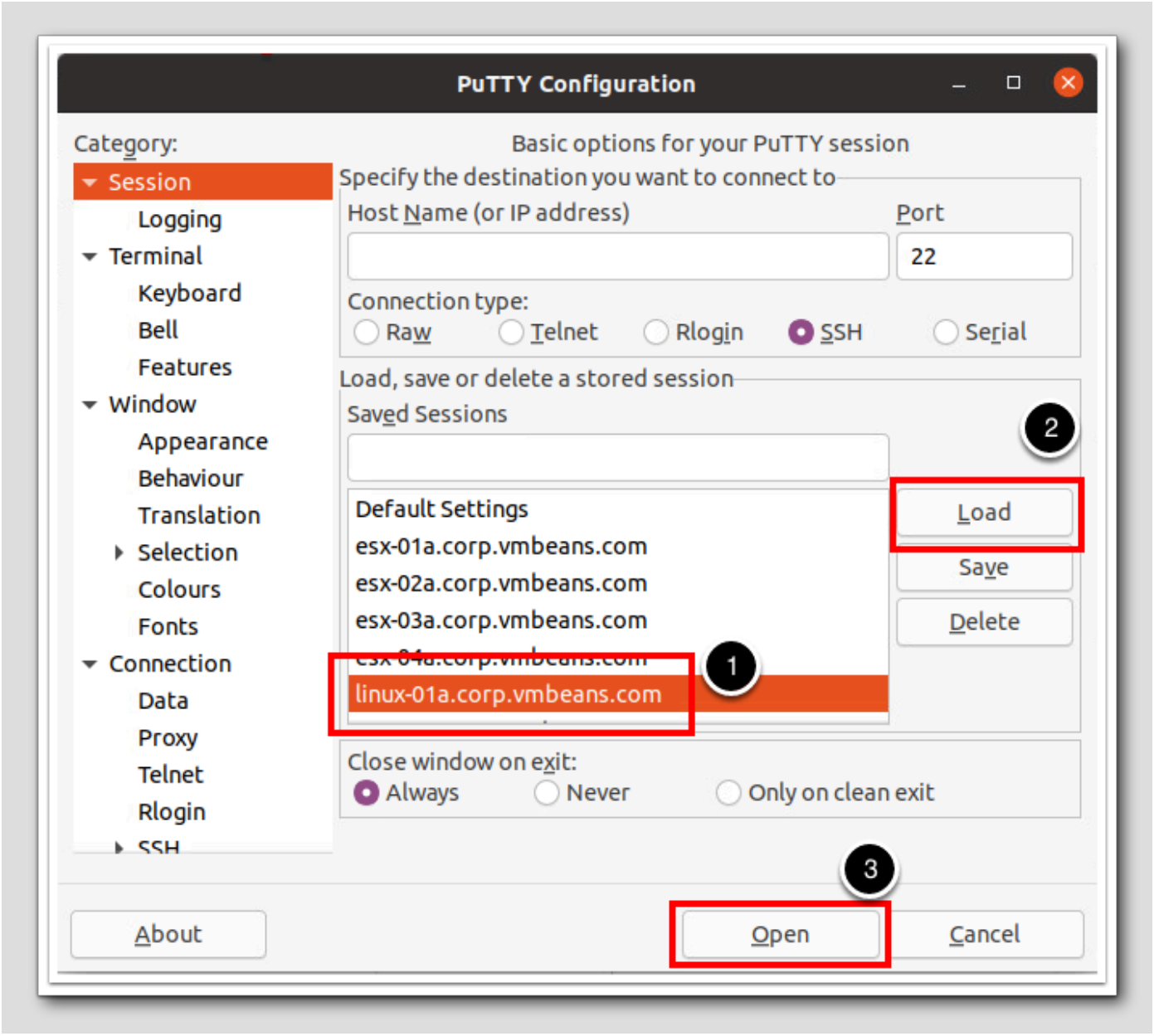
[55]

For the purposes of this lab we have already installed docker on an Ubuntu Linux image. Feel free to check *https://docs.docker.com/get-docker/*

If you would like to install docker on your desktop or laptop.

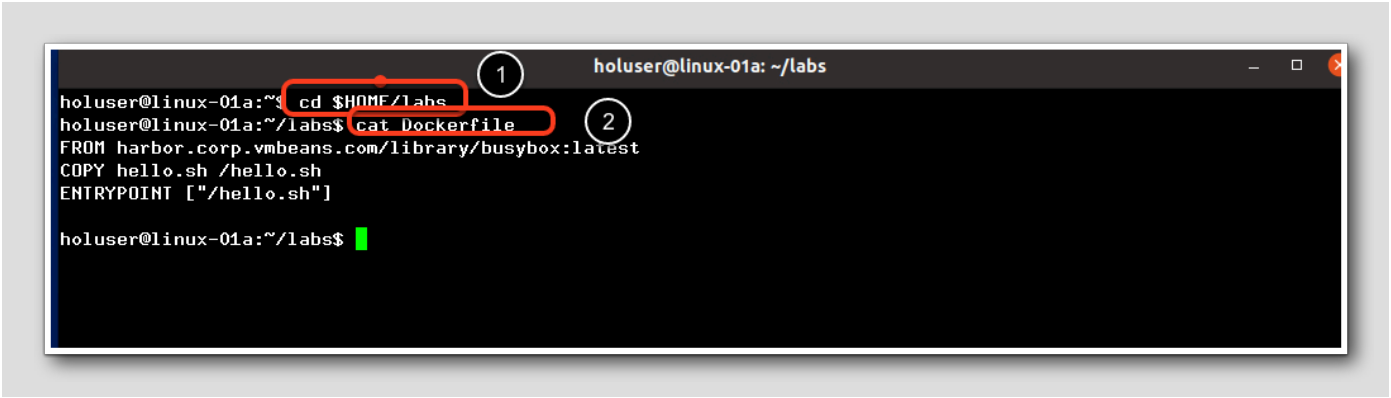
NOTE: If you are already connected to the Ubuntu cli-vm image skip this step.

1. Click on the **PuTTY** icon in the taskbar to open a new SSH session.


PuTTY Configuration
Category:
Session
Logging
Terminal
Keyboard
Bell
Features
Window
Appearance
Behaviour
Translation
Selection
Colours
Fonts
Connection
Data
Proxy
Telnet
Rlogin
SSH
Basic options for your PuTTY session
Specify the destination you want to connect to
Host Name (or IP address)
Port
22
Connection type:
Raw
Telnet
Rlogin
SSH
Serial
Load, save or delete a stored session
Saved Sessions
Default Settings
esx-01a.corp.vmbeans.com
esx-02a.corp.vmbeans.com
esx-03a.corp.vmbeans.com
linux-01a.corp.vmbeans.com
Load
Save
Delete
Close window on exit:
Always
Never
Only on clean exit
About
Open
Cancel
1
2
3

Login using SSH to Linux-01a VM:

1. Click on **linux-01a.corp.vmbeans.com** under Saved Sessions.
2. Click the **Load** button.
3. Click**Open.**
4. After a few seconds, you should be logged in as the **root** user. You should not be asked for a password as that is handled through key based authentication.
5. Supervisor Cluster authentication is integrated with vSphere Single Sign On through the vSphere Kubernetes Plugin. This plugin is already installed in the Linux-01 workstation.

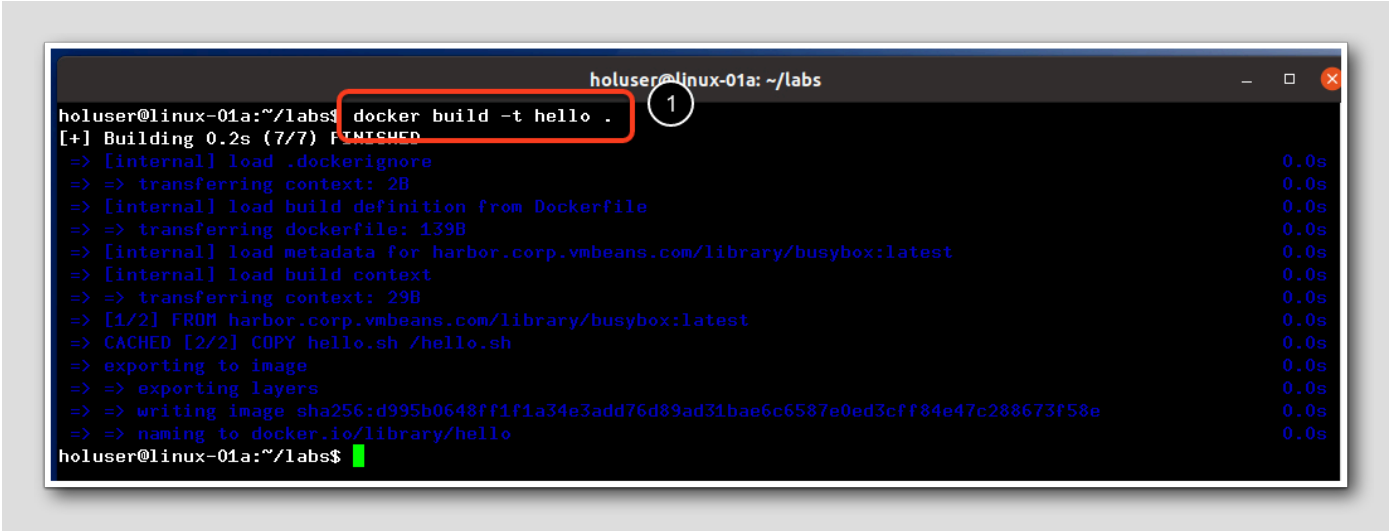


Examine a Simple Dockerfile

[56]

Build a SImple Docker Image

[57]



Let's build the new image! Type the following and press Enter. (Notice the dot at the end!)

1. `docker build -t hello .`

The -t option allows us to give the image a name (tag). We'll use that when running it.

Note that there is a period/dot "." at the end of the command line.

What are Images

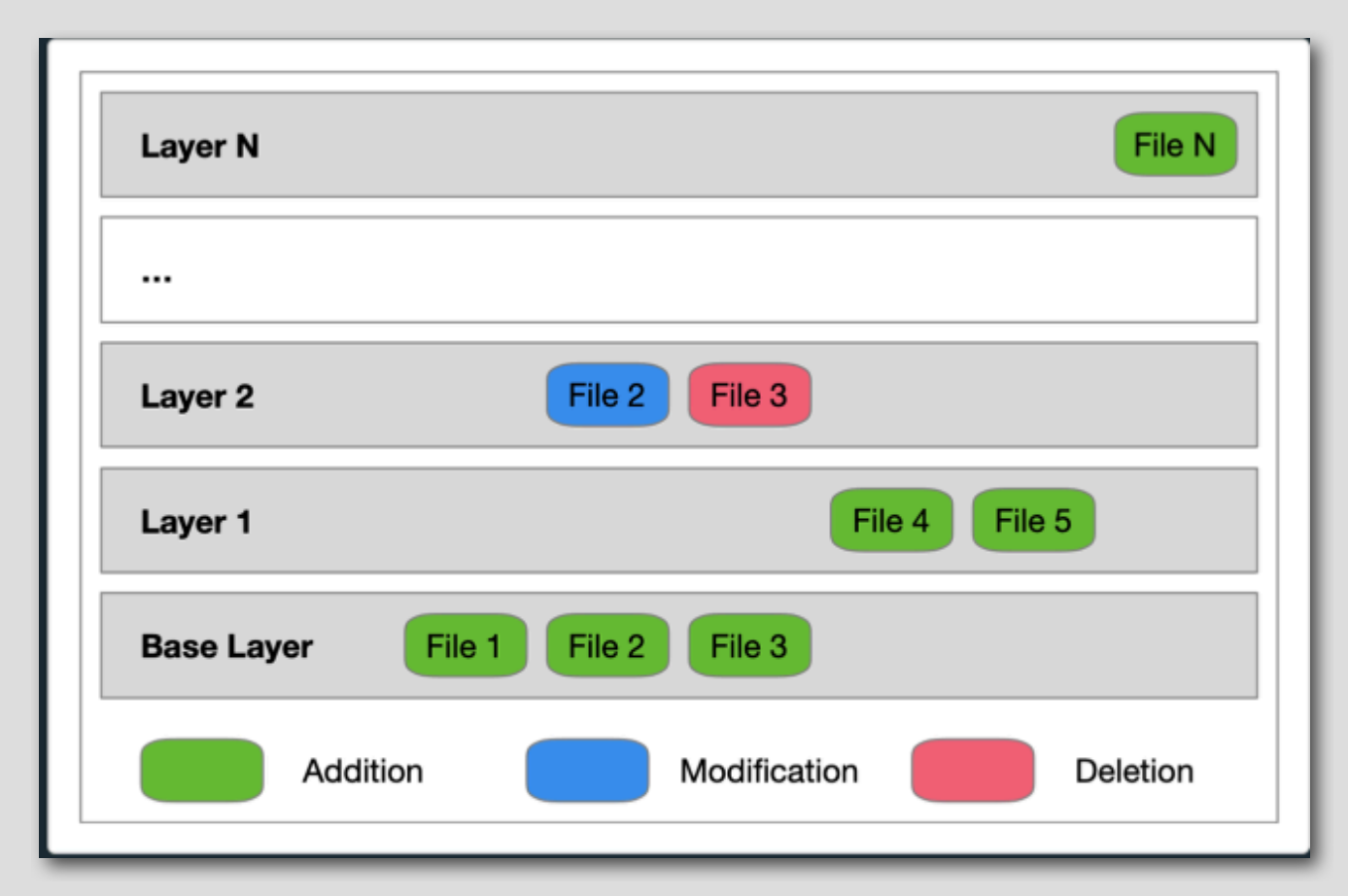
[58]

You created a running container from a container image. The container image you used was pulled down from an image registry. Although there are lots of container images available for commonly used software applications, it is more likely that you will want to create a container image for your own application.

A container image was previously described as being a packaging mechanism for distributing an application, including the application software, operating system files, libraries and other software required by the application to run.

In that sense, a container image can be seen as being a fancy form of tarball or zip file which is unpacked to populate a file system. In actuality it is a bit more complicated but this analogy is close.

Rather than being a single package containing all the required files, a container image is a set of packages, where each contains the files from a single layer of the container image.



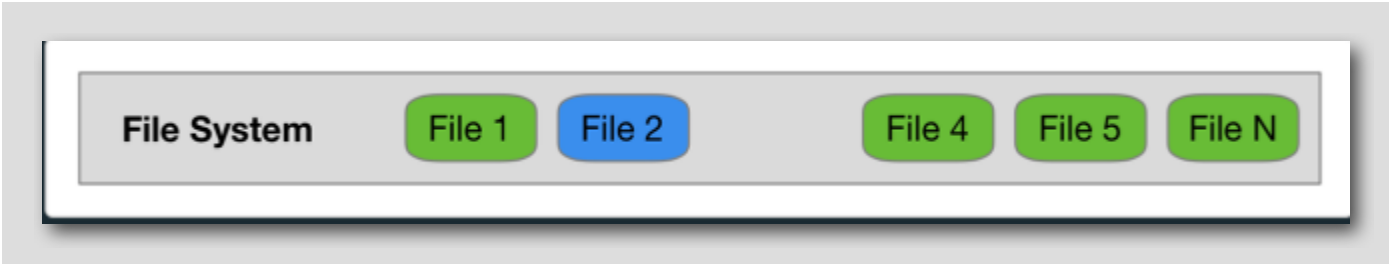
For each container image, one starts out with a base layer. This would usually consist of all the base operating system files, applications and libraries.

On top of this additional layers are added. In each layer you might add more files, modify existing files, or delete files.

When existing files are modified or deleted, it is not the original file in the lower layer that is changed. Instead, for a modification, a new copy of the file with the changes exists in the new layer, with the original in the lower layer still existing as it was. In the case of a file deletion, the metadata of the layer records that the file was deleted in that layer, with the original still existing in the lower layer.

Changes or deletions are handled in this way because each layer is immutable. The only way to affect a change is in a new layer.

When the container image is used in creating a container, a special type of file system is used which overlays each layer from the container image on top of each other to form a composite view of the layers.

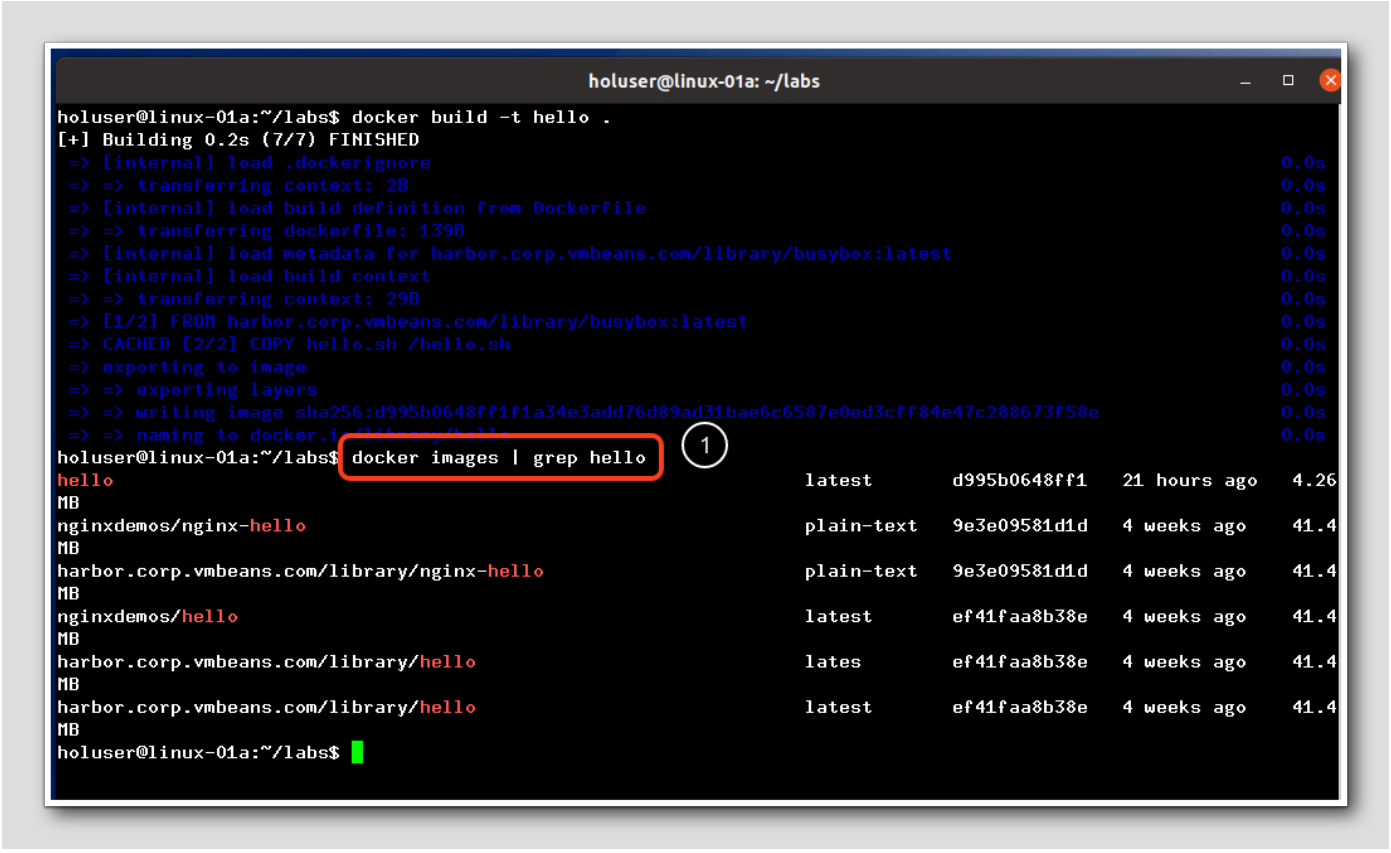


This final composite view is itself still read only. For changes made to files within the container, they will exist in yet another layer which exists for the life of the container.

The actual specification for the container image format is defined by the OCI Image Format Specification. How the image is unpacked, interpreted and used to run an application in a container is defined by the OCI Runtime Specification. Both specifications are managed under the governance of the Open Container Initiative, a Linux Foundation sponsored organization.

Examine The Docker Images

[59]



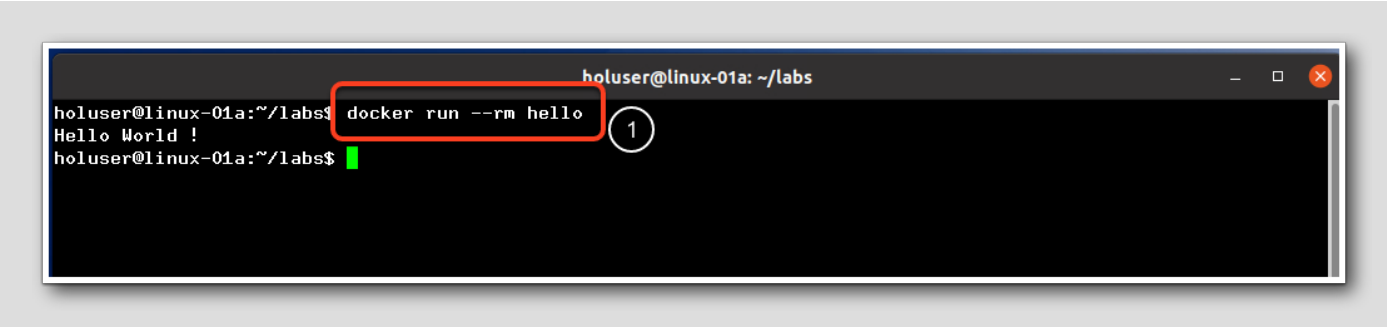
You can see all of the tagged docker images by typing "docker images" to reduce the amount of output from "docker images" lets pipe the output into the Linux "grep" utility.

1. `docker images | grep hello`

You see the docker "hello" image we created with the docker build command previously.

Execute The Docker Image

[60]



Run the Hello world application. Type the following and press Enter:

1. `docker run --rm hello`

Docker ran the container we created with the docker build command and executed the hello.sh command which we specified as the ENTRYPOINT. The --rm option told docker to delete the container after it ran it.

Building an Image

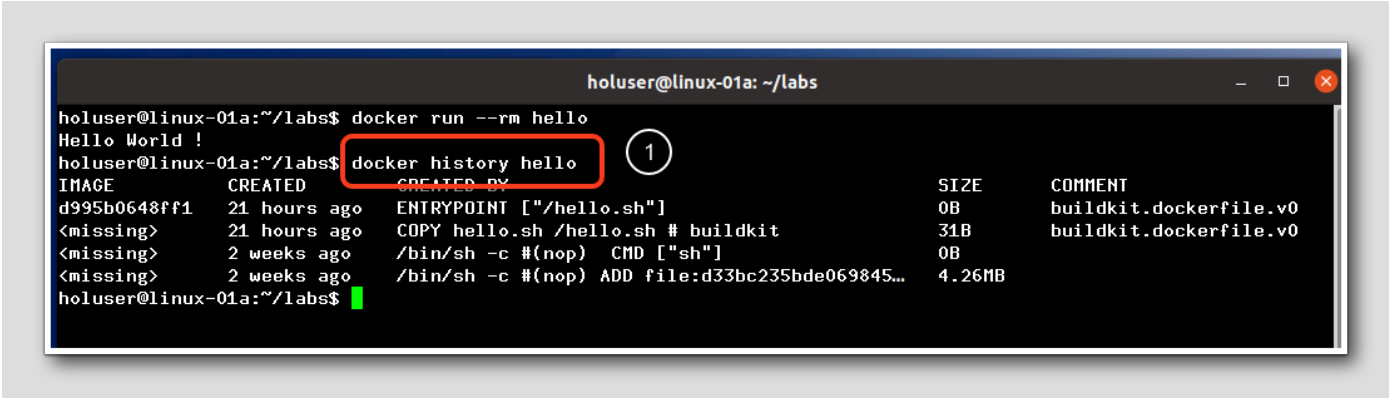
[61]

There are three principal ways of building a container image. These are:

- Interactively building an image by starting a container using an existing image, making changes in the container, and saving the result as a new container image. We executed this with the Busybox example previously.
- Creating a container image as a batch process using a set of scripted instructions in an input file. The canonical example of this is using a Dockerfile to build a container image. We executed this with the Dockerfile example above.
- Creating a container image by importing a copy of the file system from a tarball.

Inspecting the Layers of Your Image

[62]



To inspect the layers of the container image you just created, run:

```
docker history hello
```

Because you constructed the container image from a running container, there is no way to see what individual commands you ran, or what files you copied into the container.

To see the metadata for the container image, you can also run:

```
docker inspect hello
```

Choosing a Base Image

[63]

In the Dockerfile for the "hello" example we used the busybox container image. This image is convenient for demonstrations because it is small, but for real world applications it isn't really suitable.

For real world applications you will want to choose a more complete base image built from one of the more popular Linux distributions.

Popular choices for base images are:

- Fedora
- CentOS
- Debian
- Ubuntu
- Photon OS

Photon OS is an open source minimal Linux container host placed into the Open Source community by VMware. We will be using Photon OS in other modules in this lab.

Deploying with Docker Compose [64]

In this lesson, we will learn how to deploy a multi-node application using the docker-compose tool.

Specifically, we will discuss:

- Structure of the application
- The docker-compose.yaml file
- How to start the application
- Testing the application

Structure of the application [65]

In this section, we will deploy a more meaningful application. We're going to build a multi-container Wordpress Application. It includes a MYSQL database container and a Wordpress container itself. We are going to deploy each of these containers and make them communicate on a private container network. The Wordpress container will expose port 8000 to the public network.

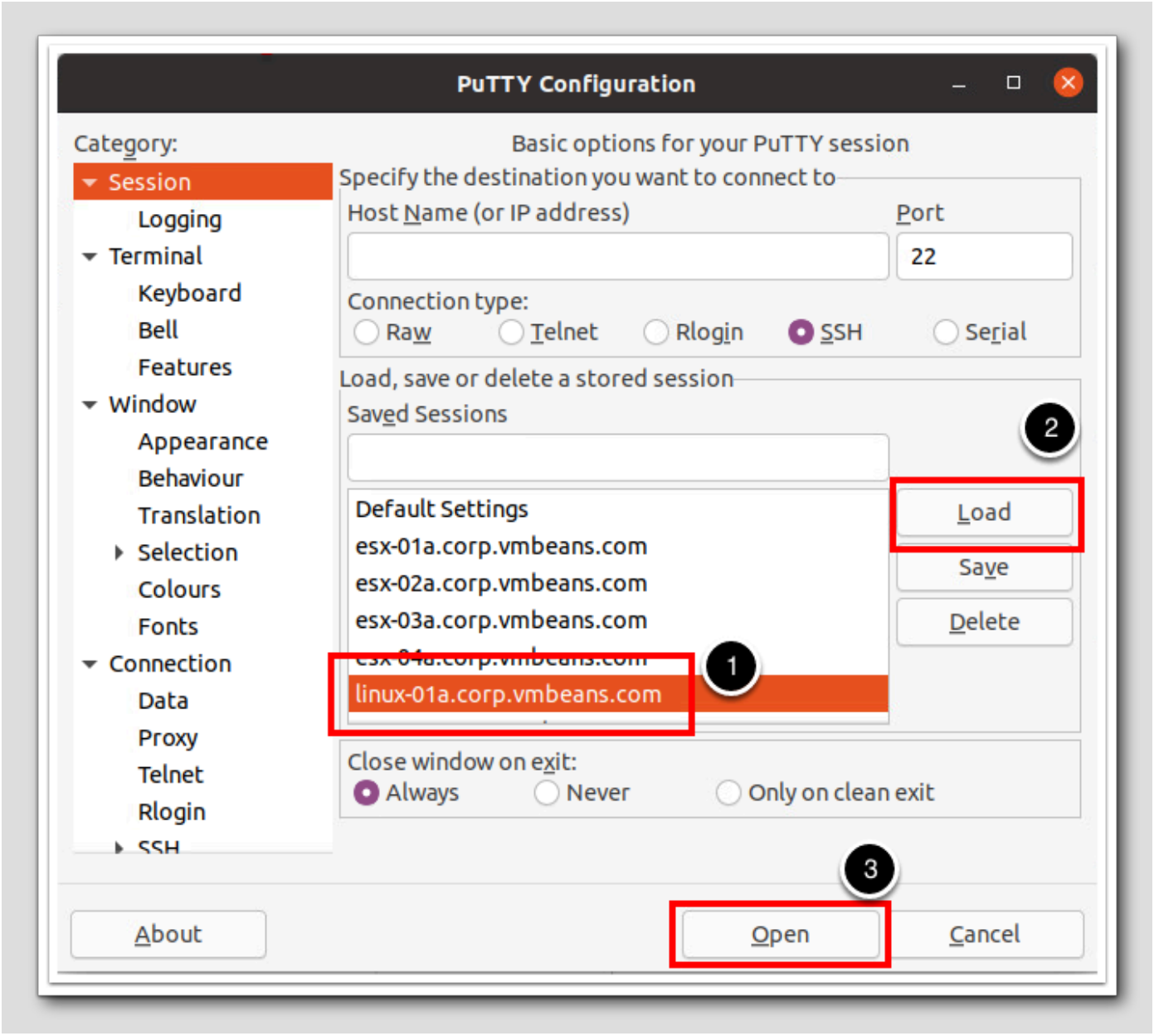
One of the challenges we're going to face is how to deploy a multi-tier application without having to manually deploy each tier and carefully tying them together. Luckily, there are several tools for doing that in an enterprise environment. Kubernetes is probably the most well-known one in that space.

In this exercise, we're going for a simpler and more lightweight solution using Docker Compose. This tool lets us define the container making up the tiers of the application in a single YAML file and bring the entire application up using a single command.

Connect to Dev Workstation [66]

NOTE: If you are already connected to the Ubuntu cli-vm image skip this step.

1. Click on the **PuTTY** icon in the taskbar to open a new SSH session.

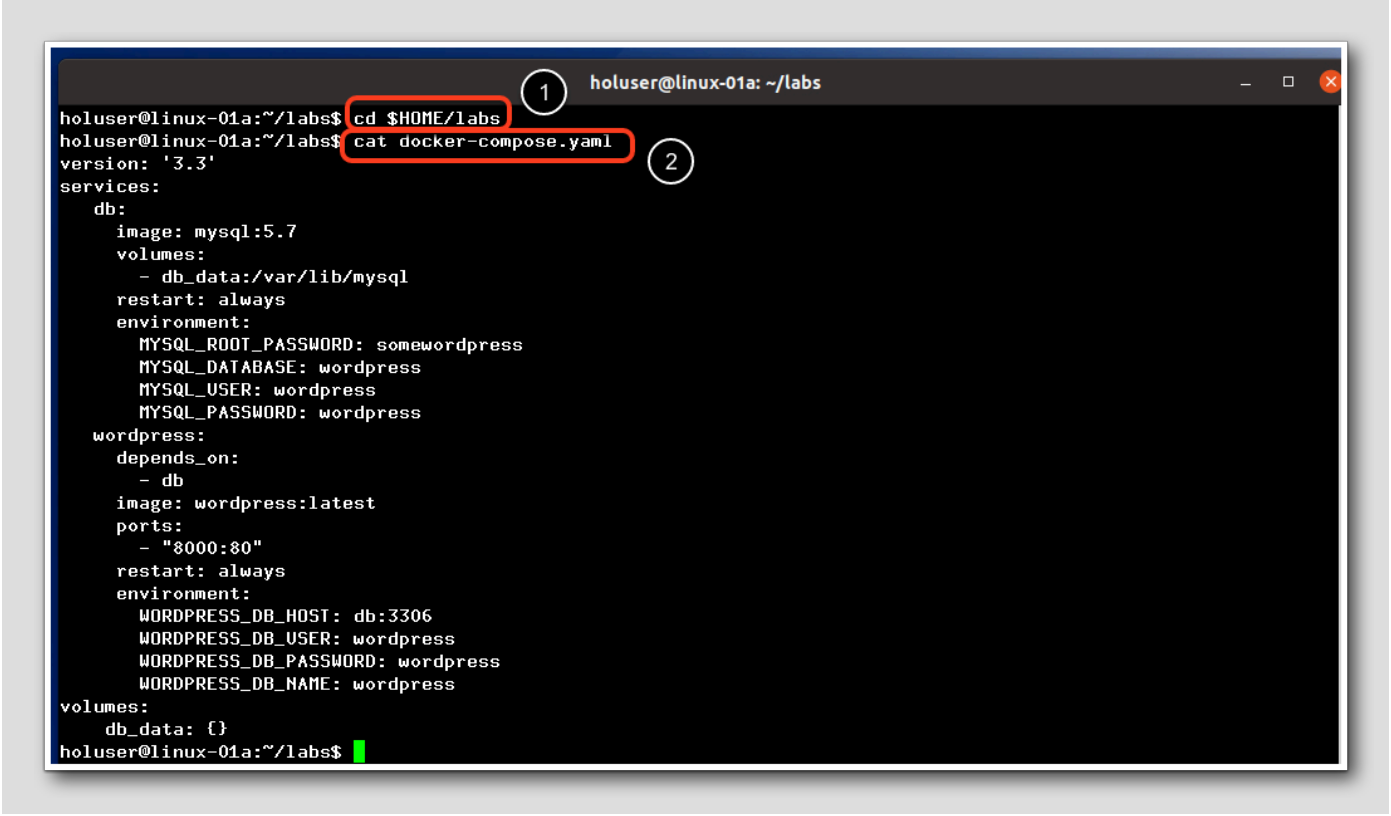

PuTTY Configuration
Category:
Session
Logging
Terminal
Keyboard
Bell
Features
Window
Appearance
Behaviour
Translation
Selection
Colours
Fonts
Connection
Data
Proxy
Telnet
Rlogin
SSH
Basic options for your PuTTY session
Specify the destination you want to connect to
Host Name (or IP address)
Port
22
Connection type:
Raw
Telnet
Rlogin
SSH
Serial
Load, save or delete a stored session
Saved Sessions
Default Settings
esx-01a.corp.vmbeans.com
esx-02a.corp.vmbeans.com
esx-03a.corp.vmbeans.com
linux-01a.corp.vmbeans.com
Load
Save
Delete
Close window on exit:
Always
Never
Only on clean exit
About
Open
Cancel
1
2
3

1. Login using SSH to Linux-01a VM:
2. Click on **linux-01a.corp.vmbeans.com** under Saved Sessions.
3. Click the **Load** button.
4. Click**Open**.
5. After a few seconds, you should be logged in as the **root** user. You should not be asked for a password as that is handled through key based authentication.

Supervisor Cluster authentication is integrated with vSphere Single Sign On through the vSphere Kubernetes Plugin. This plugin is already installed in the Linux-01 workstation.

Examining the docker-compose.yaml file

[67]



We have already provided a docker-compose file for you to use with this exercise. Let's familiarize ourselves with the structure of this file.

1. Type the following two commands, each followed by Enter:

   `cd $HOME/labs`

2. `cat docker-compose.yaml`

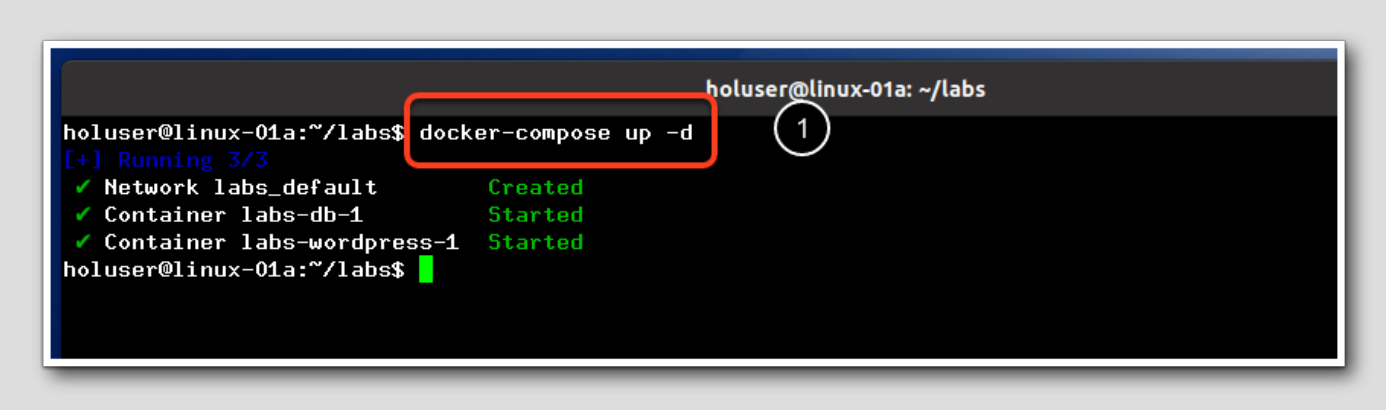
You should see a short text file. Let's examine the contents!

- services - Lists the services (containers) the application will spin up.
- image - The name of the docker image to spin up when the service starts.
- ports - Any ports we want to expose publicly. In this case, we're mapping the internal port 80 to port 8000 on the public network.

Since we have dependencies between the nodes, docker will automatically place them on a container network and link them together. It will also set up a simple naming service so that the code in the frontend can refer to the nodes by name instead of using IP addresses.

Starting the application

[68]



Now for the fun (and easy) part. Let's start the application. Since the application is fully described by the docker-compose file, we can bring it up without adding any additional parameters.

1. Type the following command followed by Enter:

   `docker-compose up -d`

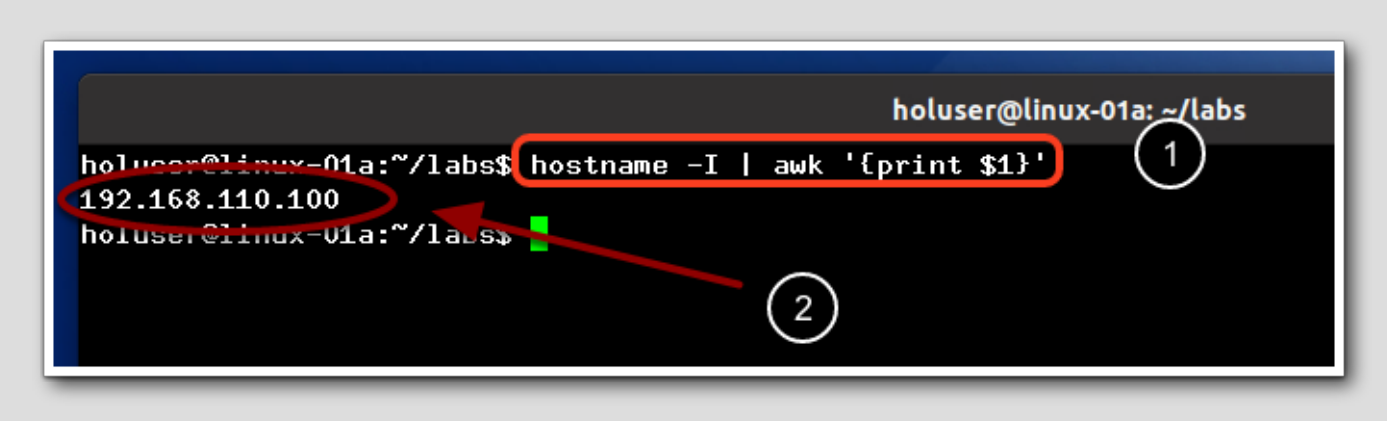
2. You should see some status messages and the application should be ready for use within a few seconds.

The -d option just tells docker-compose to start the containers in background mode without assigning the current terminal to them.

Navigating to the Application

[69]

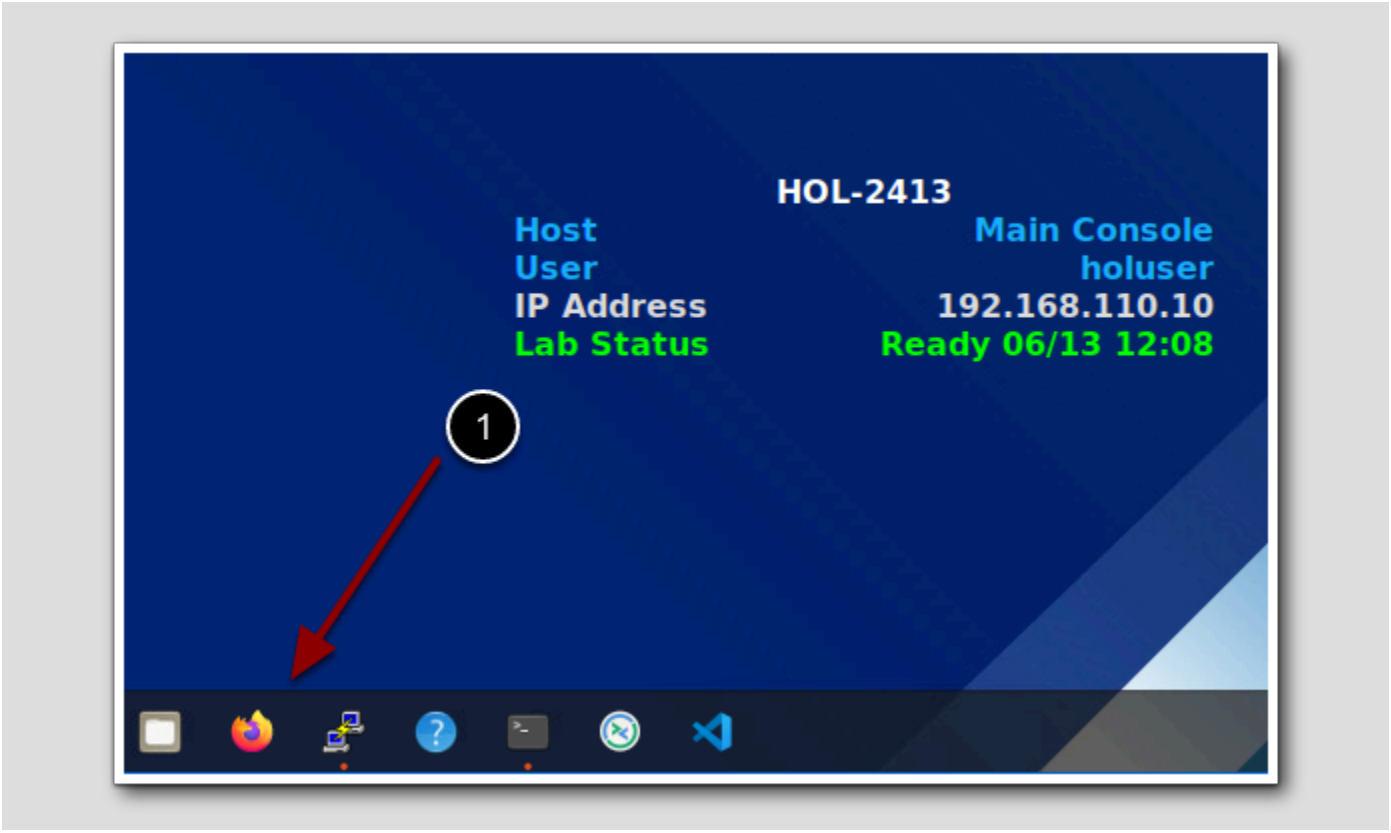
Since we kicked off our Wordpress container in our cli-vm Ubuntu image we need to get the IP address of our Ubuntu image.



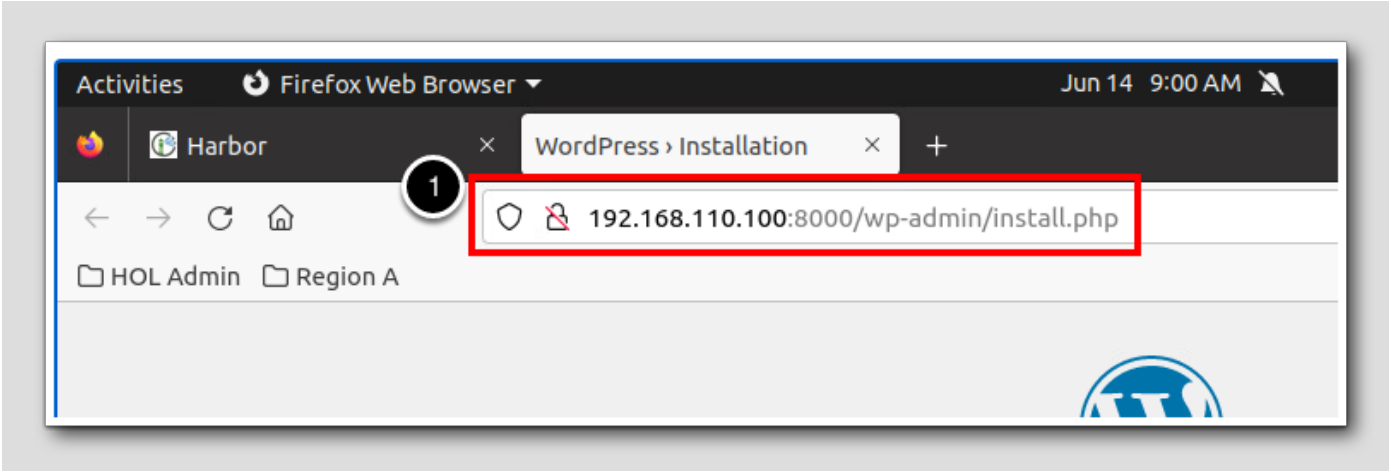
Let's get our IP address.

1. `hostname -I | awk '{print $1}'`

hostname -i returns a lot of information so we use the awk command to filter the output to just the local IP address.



1. Open a Firefox Browser and open a new browser tab.



1. Enter the IP address you obtained from the previous command followed by ":8000"

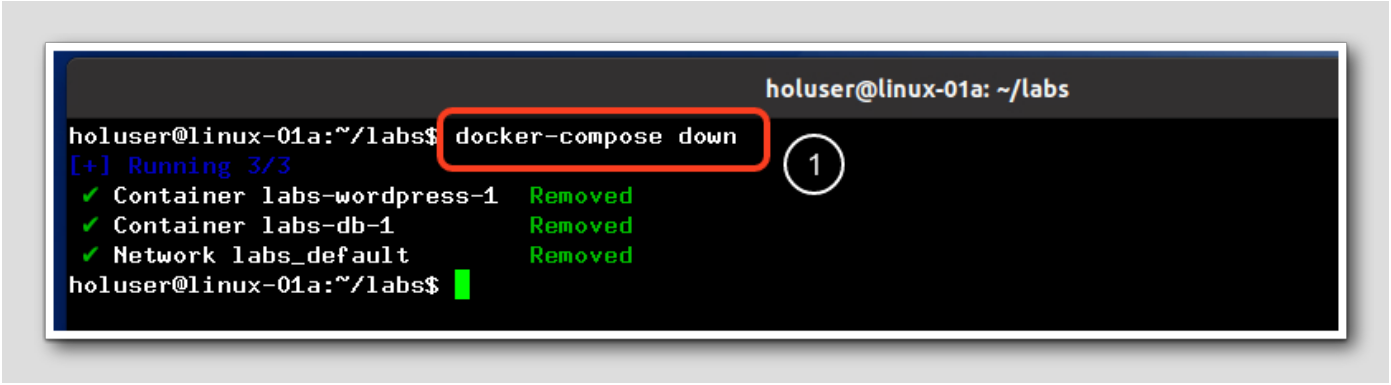
For example 192.168.110.100:8000

Using the IP address we obtained from hostname open the Wordpress container which is listning on port 8000. You should see a Wordpress screen such as the one above.

NOTE: Your labs IP address may be different then the example.

Bringing the application down

[70]



Let's clean up after ourselves by bringing the application down!

1. Back in Putty, type the following and press Enter:

```
docker-compose down
```

Not surprisingly, the "down" command does the exact opposite of "up" and brings down the containers, as well as releasing all resources associated with them. Also unsurprisingly if you click refresh on your browser in you will get a "This site can't be reached" error.

Exploring Docker Commands

[71]

In the previous modules we've examined some of the capabilities of Docker, creating images from the command line and thru a Dockerfile. In this chapter we'll look at some of the more commonly used Docker commands:

- systemctl status docker
- docker version
- docker images
- docker login
- docker pull and push

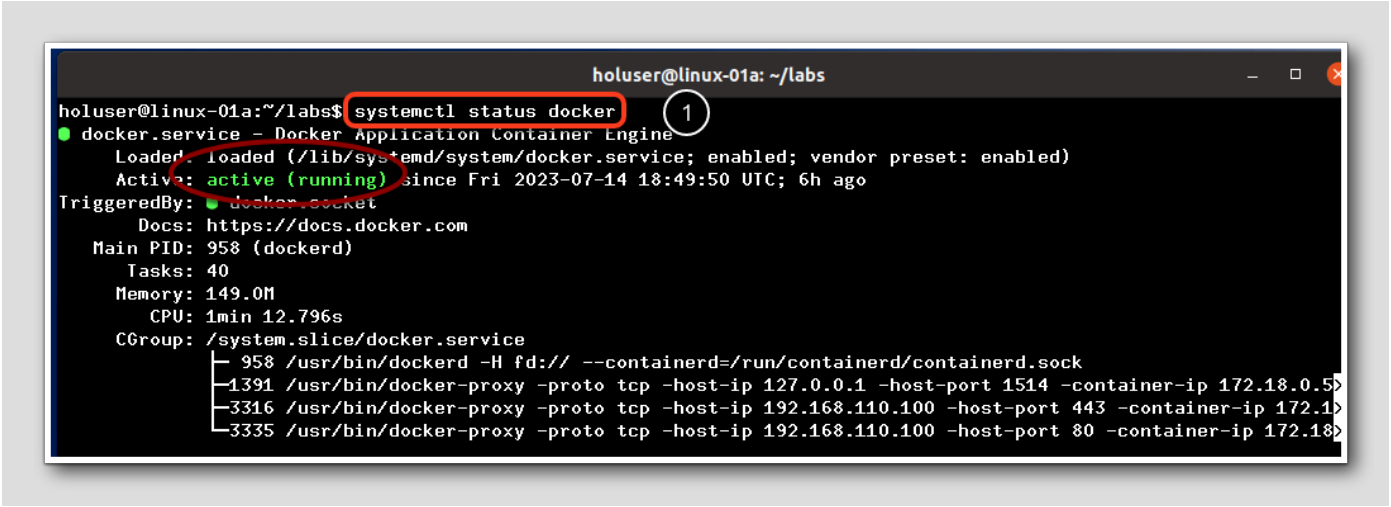
- docker run and docker build
- docker ps
- docker ps -a
- docker stop
- docker kill
- docker commit
- docker rm and rmi

Refer to *https://docs.docker.com/*

For more information on these and other docker commands.

Docker Status

[72]



Verify that docker is dunning and active with the following command:

1. Type **`systemctl status docker`**
2. Press Ctrl-C to return to the prompt.

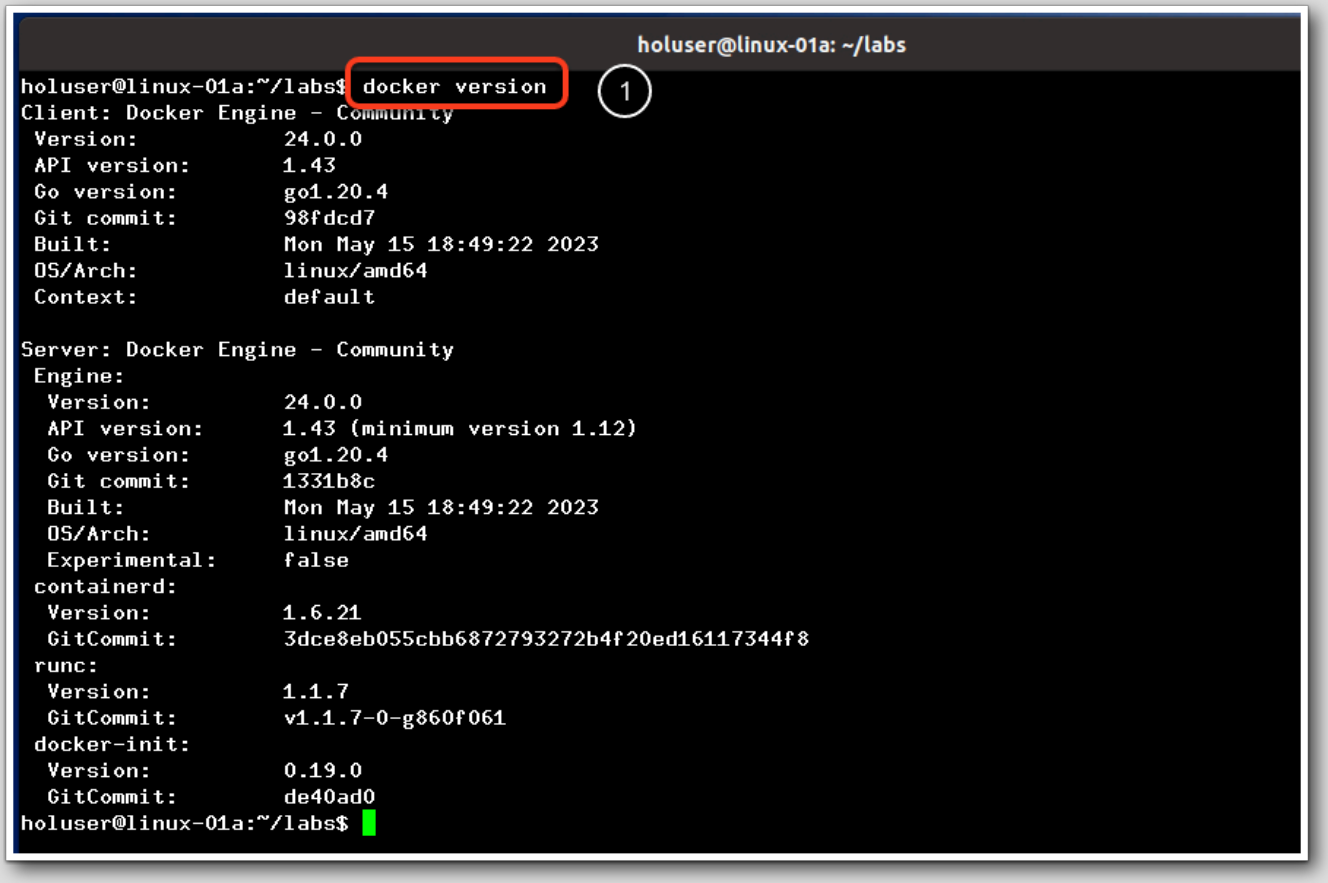
Note that if the docker daemon isn't running you would use the commands:

`sudo systemctl start docker`

`sudo service docker start`

Docker Version

[73]



For compatibility and compliance reasons it's important to know what version of docker is currently installed.

`docker version`

Displays the currently installed version of docker.

Login to The Docker Hub Repository

[74]

Before interacting with the code repository you would need to authenticate yourself with:

`docker login`

However, for security, access to the external internet is limited in the Hands-on Labs environment as such we will not execute this command from this environment.

Pull/Push An Image To/From A Repository

[75]

You can see what images are already in the image cache by typing:

```
docker images
```

In most cases the image you want won't be in the image cache, in this case you will need to obtain the image from the docker repository, for example hub.docker.com being a common repository.

```
docker pull <image>
```

Correspondingly to add an image to a repository:

```
docker push <user/image>
```

Note: *For security, access to the external internet is limited in the Hands-on Labs environment as such the docker pull/push commands cannot be executed from this environment.*

Run/Create A Docker Container

[76]

In previous modules we have created a container from an image :

```
docker run -it busybox
```

Created a container with the busybox image the `-it` flag simply meant that you want an interactive process with the current terminal connected to it.

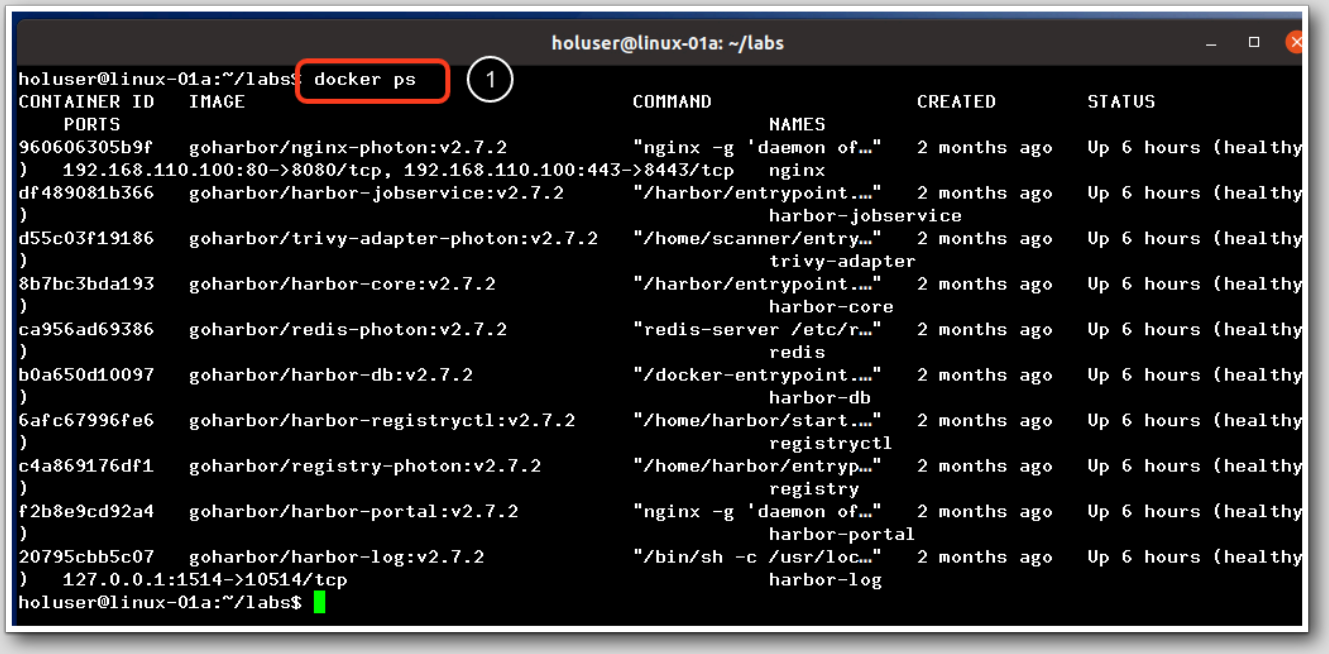
We also created a container using a Dockerfile:

```
docker build -t hello .
```

The -t option allows us to give the image a name (tag). We'll use that when running it.

List the running containers

[77]



Type:

`docker ps`

To display all of the currently running containers.

Adding the -a option displays all of the running and exited containers.

Stop A Running Container

[78]

In the exercises we did in this lab we started several containers but never terminated them. To stop a container execute:

`docker stop <container id>`

Kill A Running Container

[79]

```
docker kill <container id>
```

This command kills the container by stopping its execution immediately. The difference between docker kill and docker stop is that docker stop gives the container time to shutdown gracefully, in situations when it is taking too much time for getting the container to stop, one can opt to kill it.

Delete A Stopped Container

[80]

Stopping or killing a container (docker stop/docker kill) will only terminate the running process. To delete the container execute:

```
docker rm <image>
```

To delete the image from local storage:

```
docker rmi <image>
```

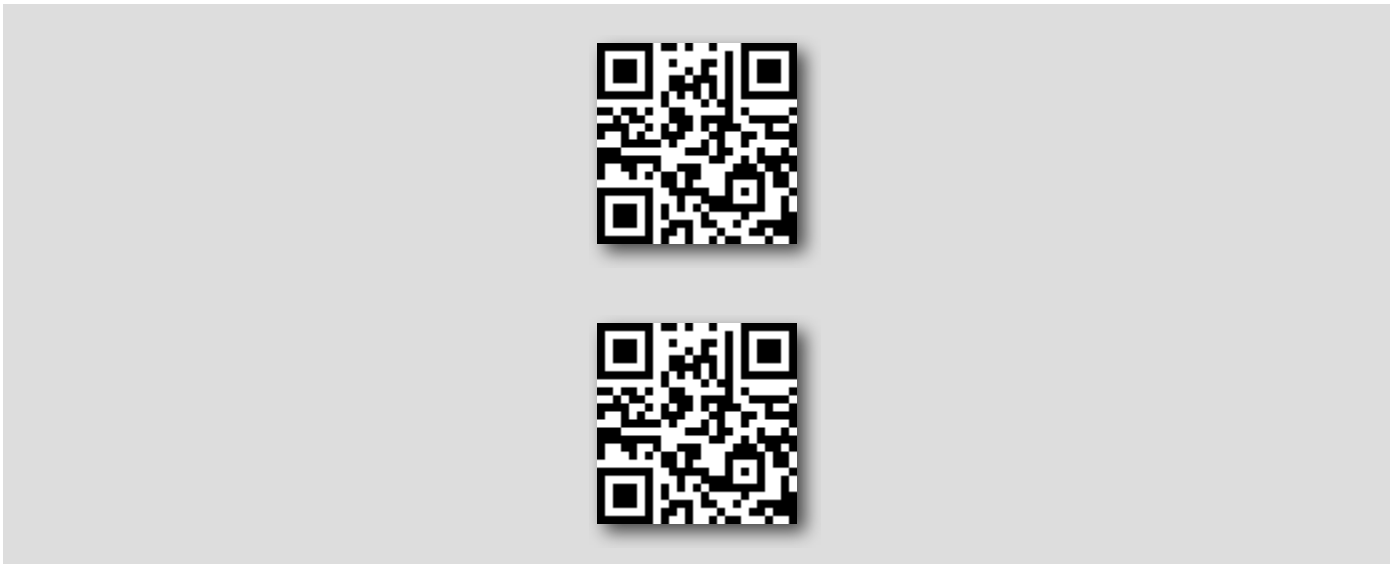
Conclusion

[81]

Congratulations on completing Module 2. You should now have a foundational understanding of container technologies and the docker implementation. If you want to learn more about docker commands they are all documented on the *Docker website*.

Plus VMware provides a host of Modern Apps FREE learning resources hosted on our *Tanzu website* as well as the *Modern Apps Community*.

Please use the QR code for more information.



Congratulations you have finished Module 2

[82]

Based on your interests, please proceed to any module below:

- Module 1 - Introduction to Containers (15 minutes) (Basic) - In this module, we will explain containers and how they enable 3rd Platform application architectures to be run efficiently in distributed environments. We will also delve into the basics of Docker as a container platform. This is a reading module only without any interactive labs.
- Module 3 - Introduction to Kubernetes - (30 minutes) (Basic) In this module we will explain the place where container management fits and take a high level walk thru the Kubernetes architecture. We will walk through some worked examples using a kubernetes instance deployed on Tanzu Kubernetes Grid (TKG).
- Module 4 - Introduction to Harbor - (30 minutes) (Advanced) In this module we will walk through some of the capabilities of the Harbor secure image repository and explore some worked examples using images pre-populated into this labs Harbor instance
- Module 5 - Introduction to Tech Zone VMware Learning Platform & Kube Academy - (15 minutes) (Basic) In this module we will introduce some of the many FREE learning resources which VMware provides for users looking to start their journey into modern apps and kubernetes or looking to build on the knowledge and experience they already have.

From here you can:

1. Click to advance to the next page and continue with the next lab module
2. Open the TABLE OF CONTENTS to jump to any module or lesson in this lab manual
3. End your lab and come back and start it again in the future

Module 3 - Introduction to Kubernetes (15 minutes) Basic

Terminology is a barrier. Kubernetes objects explained

[84]

Many people new to the container space and Kubernetes get hung up on all of the new terminology. Before jumping into the details of the platform, we are going to spend a little time defining some of the terms that will be used later on to describe the function of the platform. The goal is to provide some level of depth on these topics, however if you find that this is more than you need, skip to Module 3 and start using Kubernetes and Tanzu Kubernetes Grid.

We haven't used the term "Tanzu" or "Tanzu kubernetes Grid (TKG)" previously, so what are these.

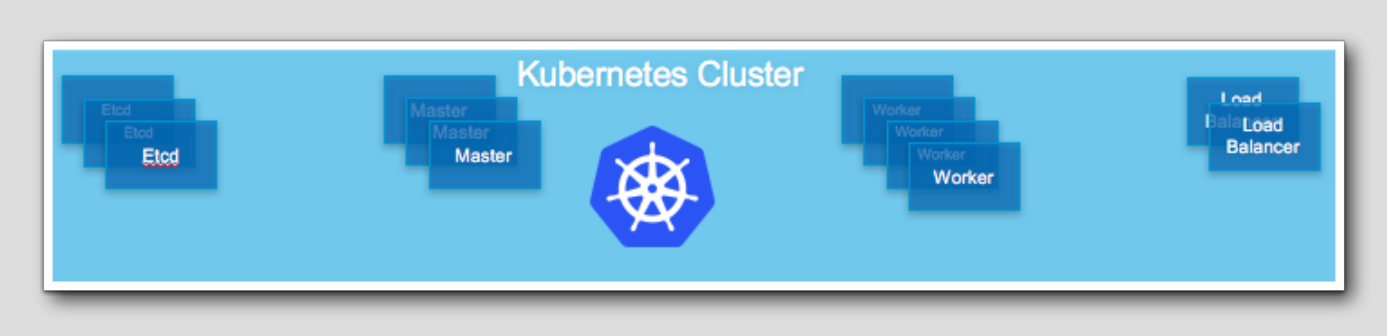
VMware Tanzu is a family of products and services for modernizing your applications and infrastructure with a common goal: deliver better software to production, continuously. The portfolio simplifies multi-cloud operations, while freeing developers to move faster and access the right resources for building the best applications. VMware Tanzu enables development and operations teams to work together in new ways that deliver transformative business results.

Tanzu Kubernetes Grid (TKG) simplifies operation of Kubernetes on-premises by putting cloud native constructs at the virtualization admin's fingertips as part of vSphere 6.7U3 or vSphere 7. It delivers an **open source-aligned Kubernetes distribution**, packaged for the enterprise and delivered as part of your existing infrastructure to support application modernization. We've highlighted open source-aligned as this is key to the modern applications development community, since TKG uses the Cloud Native Kubernetes distribution there is no "vendor lock in" and quite simply "it just works" your existing Yaml files "just work" on TKG (assuming they are CNCF Kubernetes conformant).

This lab is built using VMware TKG, as you will find as you walk through lab you will notice that the standard, native Docker and Kubernetes APIs are exposed to the developer.

Kubernetes Cluster

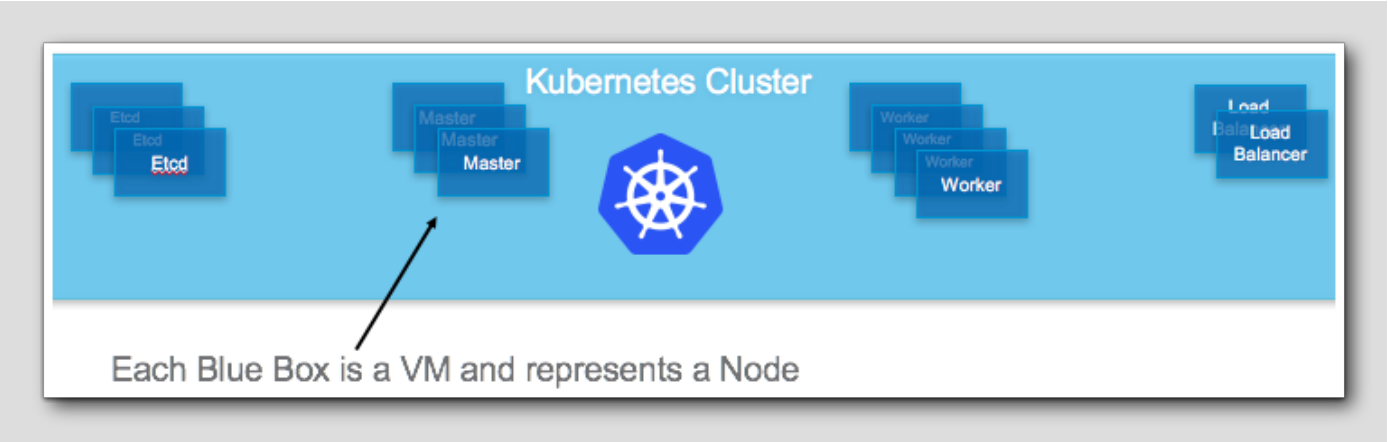
[85]



A cluster is very simply the physical or virtual machines and other infrastructure resources used by Kubernetes to run your applications. You define a set of machines, create networking and attach storage, then install the Kubernetes system services. Now you have a running cluster. This does not mean that there is any sort of traditional clustering technology in the infrastructure sense - nor does it align with vSphere clustering constructs. That has been a point of confusion for many VMware administrators. A cluster is simply a set of VMs, wired together, with attached local or shared storage - and running the Kubernetes System services.

Kubernetes Node

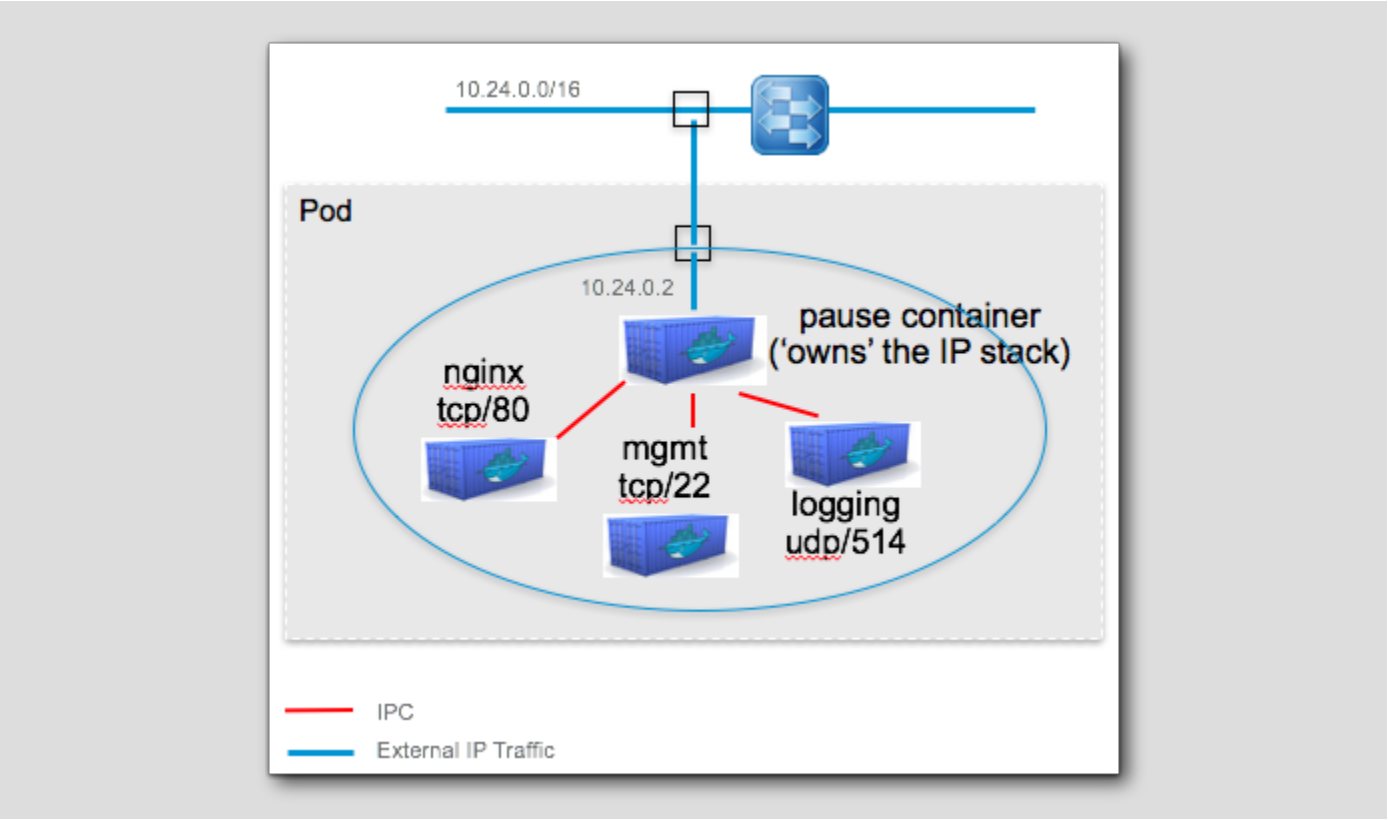
[86]



A node is any of the physical machines or VMs that make up the Kubernetes cluster. Nodes are of two types: Control and Worker. Some of the Control Plane services can be broken out into their own set of VMs and would also be referred to as nodes (we will get to Etcd shortly). The Control plane nodes run the kube-system services. The Worker nodes run an agent and networking proxy, but are primarily thought of as the set of nodes that run the pods.

Pods

[87]

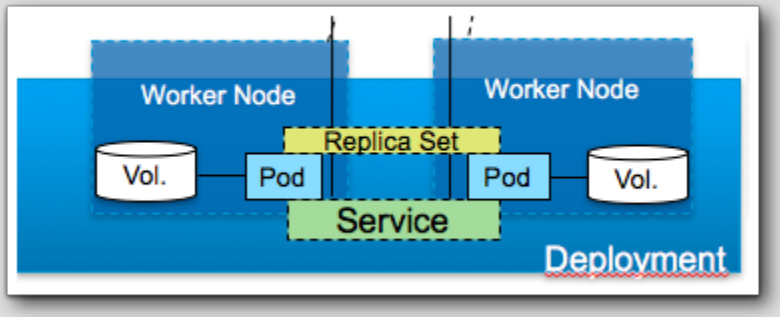


Pods are the smallest deployable units of computing that can be created and managed in Kubernetes. Pods are always co-located and co-scheduled, and run in a shared context. A pod models an application-specific logical host - it contains one or more application containers which are relatively tightly coupled. The shared context of a pod is a set of Linux namespaces, groups, and potentially other facets of isolation - the same things that isolate a Docker container.

In this sample pod, there are three application containers. The Nginx webserver, along with ssh and logging daemons. In a non-container deployment, all three of these would probably run as individual processes on a single VM. Containers generally run a single process to keep them lightweight and avoid the need for init configuration. Notice in the image that there is also a Pause container. This container actually hosts the networking stack, the other three containers will share the IP and listen on different ports. This allows all containers in a pod to communicate via localhost. Notice that the pod in this example has a single IP: 10.24.0.2 on a network that is generally private to the Kubernetes cluster. The pod is a logical abstraction that is managed by Kubernetes. If you log onto a Kubernetes node VM and look for pods, you won't find them through Docker. You will be able to see a set of containers, but no pods. You will find the pods through the Kubernetes CLI or UI.

Replica Sets

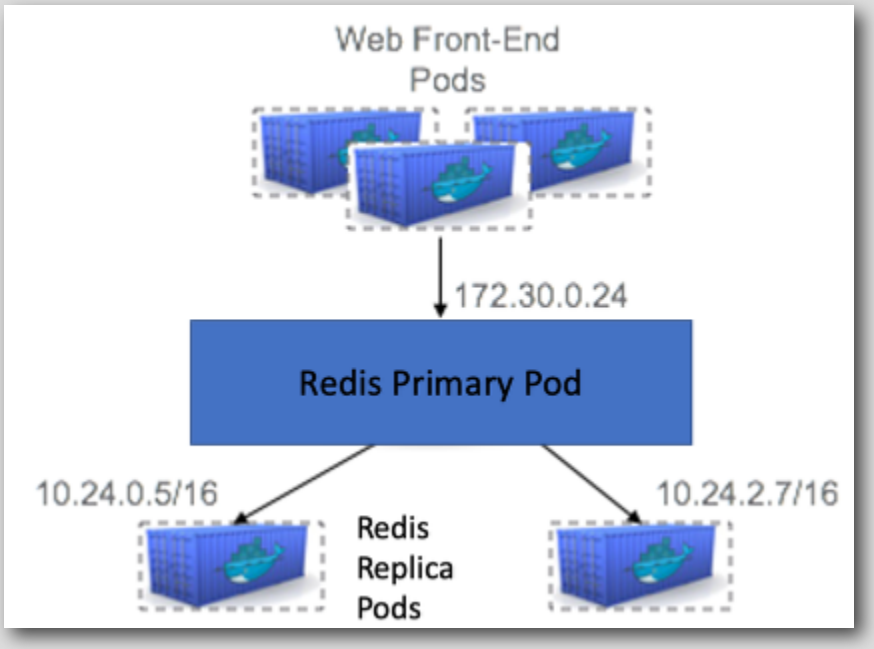
[88]



A Replica Set ensures that a specified number of pod replicas are running at any given time. A replication controller process watches the current state of pods and matches that with the desired state specified in the pod declaration. If there is a difference, because a pod has exited, it attempts to make the desired state and current state consistent by starting another pod. Developers may choose to define replica sets to provide application availability and/or scalability. This definition is handled through a configuration file defined in yaml or json syntax.

Services

[89]



Kubernetes pods are ephemeral. They are created and when they die, they are recreated - not restarted. While each pod gets its own IP address, even those IP addresses cannot be relied upon to be stable over time. This leads to a problem: if some set of pods - like say a Redis replica (Redis is a Key/Value store with a primary/replica architecture) - provides functionality to other pods - like a frontend Webserver - inside the Kubernetes cluster, how do those frontends find and keep track of which backends are in that set?

Enter Services.

A Kubernetes Service is an abstraction which defines a logical set of pods and a policy by which to access them - sometimes called a micro-service. The set of pods targeted by a service is (usually) determined by a label selector (Explained on the next page). A service generally defines a ClusterIP and port for access and provides East/West Load Balancing across the underlying pods.

Let's look at this in the context of the diagram above. There are two Redis-replica pods - each with its own IP (10.24.0.5, 10.24.2.7). When the service is created, it is told that all pods with the label Redis-replica are part of the service. The IPs are updated in the endpoints object for the service. Now when another object references the service (through either the service clusterIP (172.30.0.24) or its DNS entry, it can load balance the request across the set of pods. Kubernetes includes its own DNS for internal domain lookups and each service has a record based on its name (redis-replica).

To this point we have only talked about internal access to the service. What if the service is a web server and users must access it from outside the cluster. Remember that the IPs aren't routable outside the private cluster overlay network. In that case there are several options - Ingress Servers, North/South Load Balancing, and NodePort. In the service declaration, a specification of type NodePort means that each cluster node will be configured so that a single port is exposed for this service. So a user could get access to the frontend web service in the diagram by specifying the IP address of any node in the cluster, along with the NodePort for the frontend service. The service then provides East/West load balancing across the pods that make up the service.

Labels and Selectors

[90]

1. *Labels* are Key:Value pairs that can be attached to any Kubernetes object (pods, nodes, services) to help to identify them. For example to Identify releases (Beta, Prod), Environments (Dev, Prod) or Tiers (Frontend, Backend)
2. *Selectors* are the mechanism for group filtering based on the labels

Kubernetes labels are key/value pairs that can be attached to Kubernetes objects. Labels are meant to specify identifying attributes of Kubernetes objects. Label selectors are exactly what their name says. They allow you to select Kubernetes objects based on labels and do interesting things with them.

Kubernetes is architected to take action on sets of objects. The sets of objects that a particular action might occur on are defined through *labels.* We just saw one example of that where a service knows the set of pods associated with it because a selector (like run:redis-replica) was defined on it and a set of pods was defined with a label of run:redis-replica. This methodology is used throughout Kubernetes to group objects.

Deployments [91]

A deployment is a declarative object for defining your desired Kubernetes application state. It includes the number of replicas and handles the roll-out of application updates. Deployments provide declarative updates for pods and replica sets (the next-generation replication controller). You only need to describe the desired state in a deployment object, and the deployment controller will change the actual state to the desired state at a controlled rate for you. Think of it as a single object that can, among other things, define a set of pods and the number of replicas, while supporting upgrade/rollback of pod image versions.

Namespaces [92]

Namespaces are intended for use in environments with many users spread across multiple teams, or projects. Namespaces provide a scope for names. Names of resources need to be unique within a namespace, but not across namespaces. They are a way to divide cluster resources between multiple uses. As Kubernetes continues to evolve, namespaces will provide true multi-tenancy for your cluster. They are only partially there at this point. By default, all resources in a Kubernetes cluster are created in a default namespace. A pod will run with unbounded CPU and memory requests/limits. A Kubernetes Namespace allows users to partition created resources into a logically named group. Each namespace provides:

- a unique scope for resources to avoid name collisions
- policies to ensure appropriate authority to trusted users
- ability to specify constraints for resource consumption

This allows a Kubernetes cluster to share resources by multiple groups and provide different levels of QoS to each group. Resources created in one namespace are hidden from other namespaces. Multiple namespaces can be created, each potentially with different constraints.

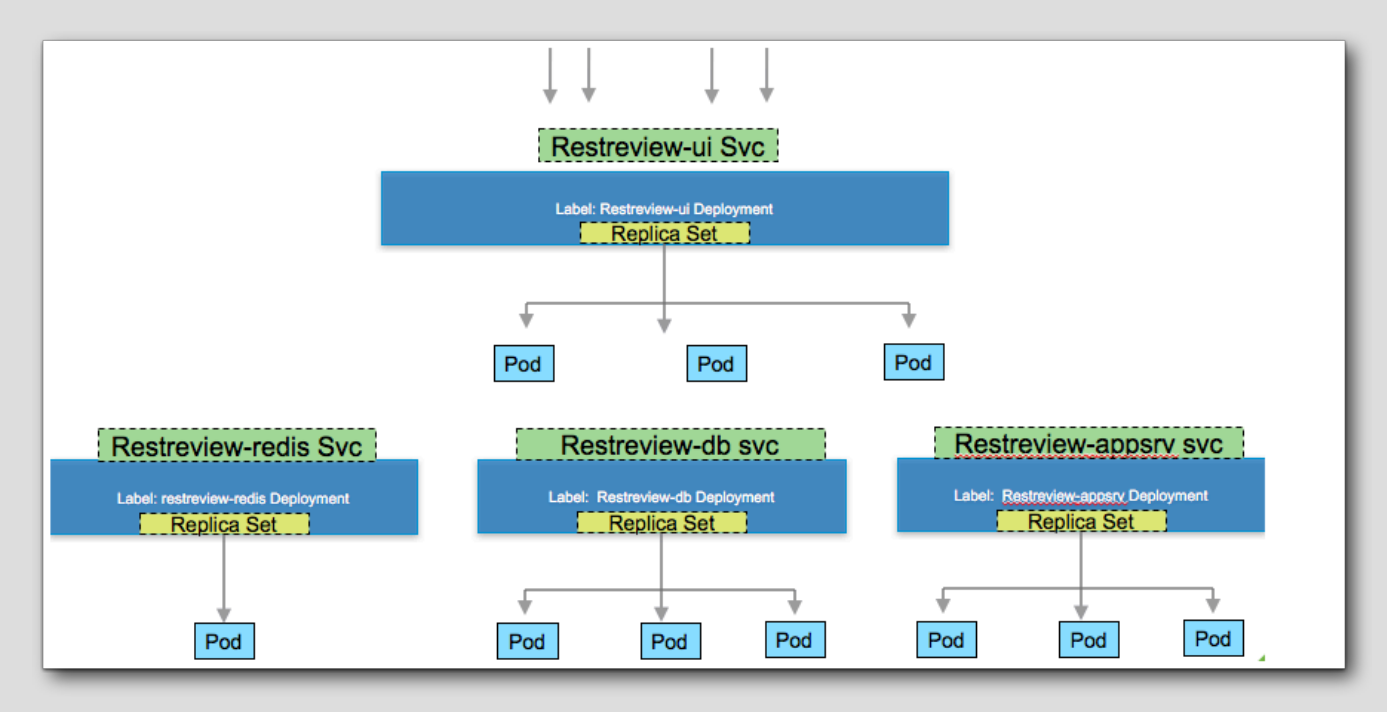
Load Balancing [93]

Load balancing in Kubernetes can be a bit of a confusing topic. The Kubernetes cluster section shows an image with load balancers. Those represent balancing requests to the Kubernetes control plane. Specifically the API Server. But what if you deploy a set of pods and need to load balance access to them? We have previously discussed services. In addition to discovery, services also provide load balancing of requests across the set of pods that make up the service. This is known as East/West load balancing and is internal to the cluster. If there is a need for ingress to a service from an external network, and a requirement to load balance that access, this is known as North/South load balancing. There are three primary implementation options:

- Create service with type 'LoadBalancer'. This is platform dependent and requires that the load balancer distributing inbound traffic is created through an external load balancer service. NSX can provide load balancing for clusters.
- Statically configure an external load balancer (Like F5) that send traffic to a K8s Service over 'NodePort' on specific nodes. In this case, the configuration is done directly on the external load balancer after the service is created and the nodeport is known.
- Create Kubernetes Ingress. This is a Kubernetes object that describes a North/South load balancer. The Kubernetes ingress object is 'watched' by an ingress controller that configures the load balancer datapath. Usually both the ingress controller and the load balancer datapath are running as pods. This requires that an ingress controller be created, but may be the most flexible solution. NSX-T provides an ingress controller.

An Example - A Sample Restaurant Rating Application

[94]



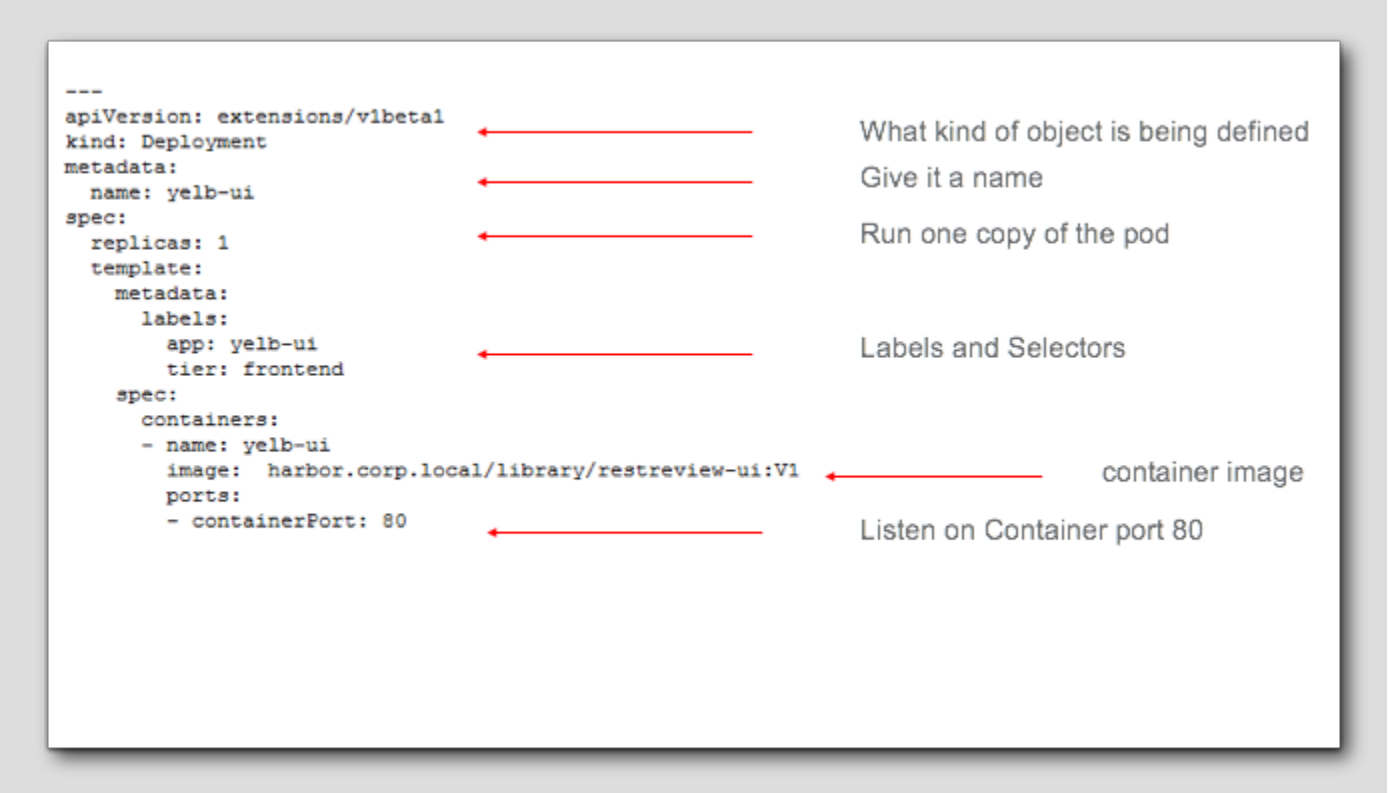
This simple application captures votes for a set of restaurants, provides a running graphical tally and captures the number of page views. It contains four separate deployments- UI, Application Server, Postgres DB and Redis caching server. A deployment provides a declarative method for defining pods, replica sets and other Kubernetes constructs. The UI Deployment includes a UI pod, which runs an Nginx Webserver. It defines a replica set that maintains three running copies of the UI pod. It also defines a UI service that provides an abstraction to the underlying UI pods, including a ClusterIP and Load Balancer that can be used to access the service.

The application is using a Redis Key:Value store to capture page views and a Postgres Database to persist the vote. Let's now dig into the configuration files that would be needed to define this application.

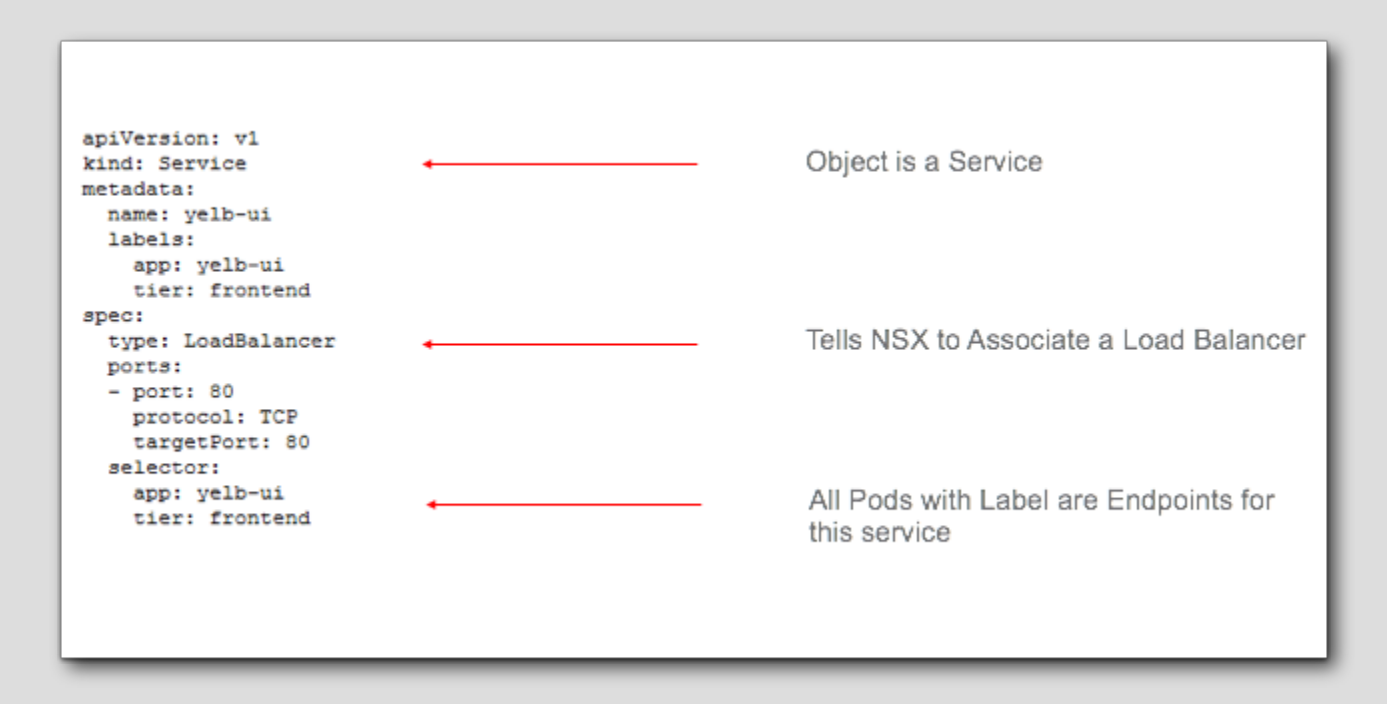
Yaml Files

[95]

The files for creating the deployments and their services can be in yaml or json format. Usually yaml is used because it is easier to read. Below are the yaml files used to create the UI deployment and the UI service.



This file defines the deployment specification. Think of it as the desired state for the deployment. It has a name - yelb-ui. It defines a replica set that includes 1 replica. That means the desired state for this deployment is that 1 copy of the pod is running. Labels are defined for these pods. You will see below that the service definition will use these to define the pods that are covered by the service. The container in the pod will be based on the harbor.corp.local/library/restreview-ui:V1 image. Resources can be constrained for the container based on the requests Key. Lastly, the container will be listening on port 80. Remember that this is container port 80 and must be mapped to some host port in order to access it from an external network.



This file defines the UI service specification. The important pieces are the Type: LoadBalancer and the Selector. Specifying Type: LoadBalancer means that NSX will associate a Load Balancer with this service to provide external access to the application. The service will then route requests to one of the pods that has a label from the service's selector. So all pods with matching labels will be included in this service. Note: NSX actually changes the routing mechanism from what is described here, but it logically works this way.

Containers And Container Management (aka Kubernetes)

[96]

In this module, we will discuss the rise of containers in modern data centers and how Kubernetes is used for container orchestration.

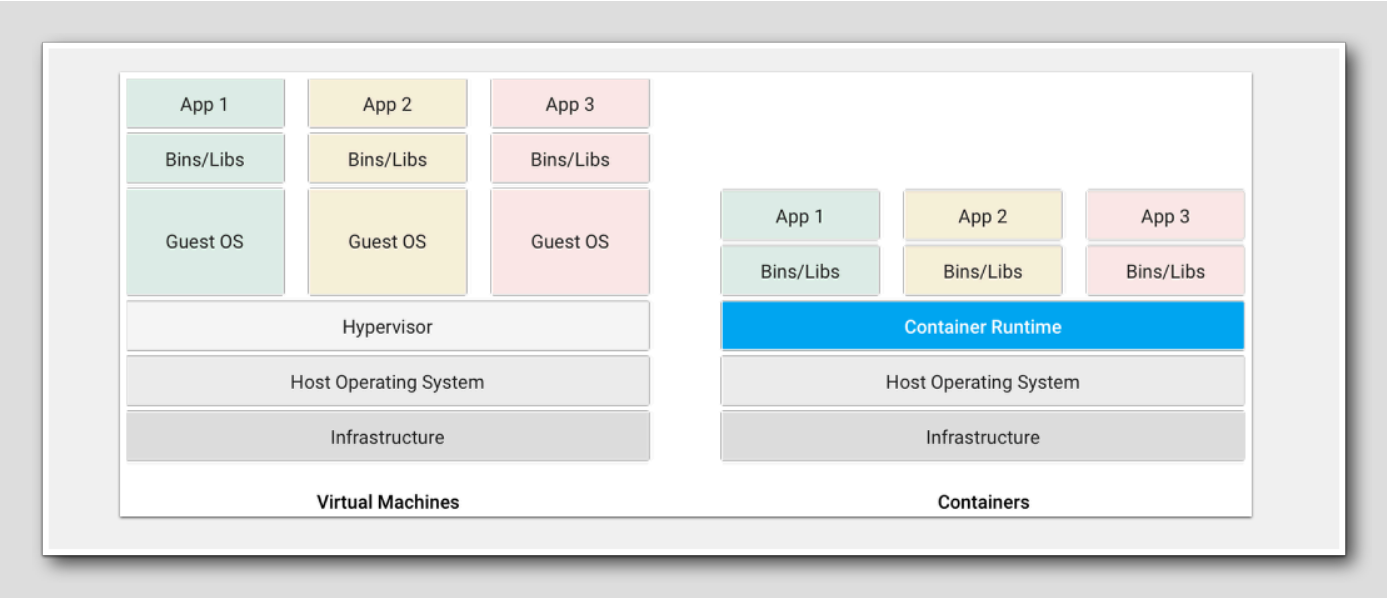
Containers vs. VMs

[97]

To understand containers, it is helpful to compare them to Virtual Machines. What do virtual machines do? They abstract from the physical hardware allowing many servers to consume resources from the same physical machine. Thus, multiple servers could run on a single physical box. With containers we go a step further and abstract at a different layer -- the OS or application layer

Multiple Containers Share an OS Kernel

[98]



So, now multiple containers can share the same OS kernel. Instead of having 3 VMs, each with their own operating system, composing an application, you can conserve resources by deploying 3 containers, each sharing in the common Operating System of the container-enabled host. While each VM may be several gigabytes in size, each container may only be a few megabytes. Moreover container deployment is much faster than VM deployment. Instead of waiting for the OS to start, containerized applications can be realized almost immediately.

The Benefits of Containers

[99]

With the ease and speed of deployment (and destruction), containers also offer elasticity, allowing current demand to direct deployment scale so that containers are deployed as needed. With the ability to share OS resources and to scale on-demand and as-required, containers are the lightweight, efficient, and agile atoms of the modern application platform. Containers are compact units of software that package code and application dependencies together. Because containers are abstracted from the OS and a particular environment, a container-based app can be deployed reliably, quickly, and consistently, regardless of environment type (on-prem data center, public cloud, personal laptop, etc.) This makes containers the ideal software unit for an ever-evolving and in-flux infrastructure.

What is Kubernetes?

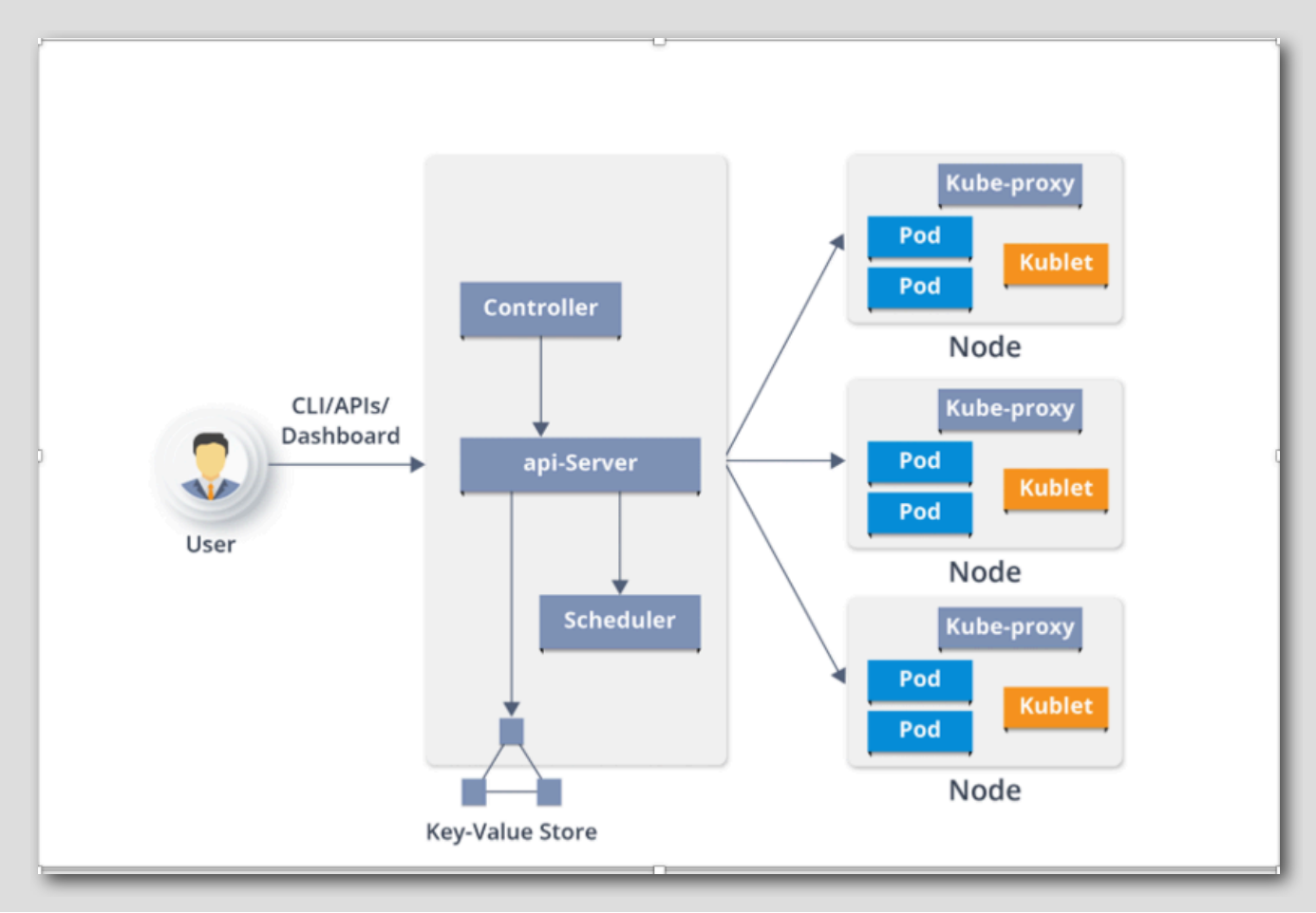
[100]

While containers are replacing traditional servers as the prime unit for the modern application, application management must also change. Since we are no longer managing applications at the OS level, we need an organized system to monitor, deploy, scale, patch, and administer these new containerized applications. In short, we need an entire new ecosystem to effectively manage containers, and this is where Kubernetes enters the picture.

Kubernetes (commonly stylized as K8s) is an open-source container-orchestration system for automating deployment, scaling and management of containerized applications. It was originally designed by Google and is now maintained by the Cloud Native Computing Foundation.

Kubernetes Architecture

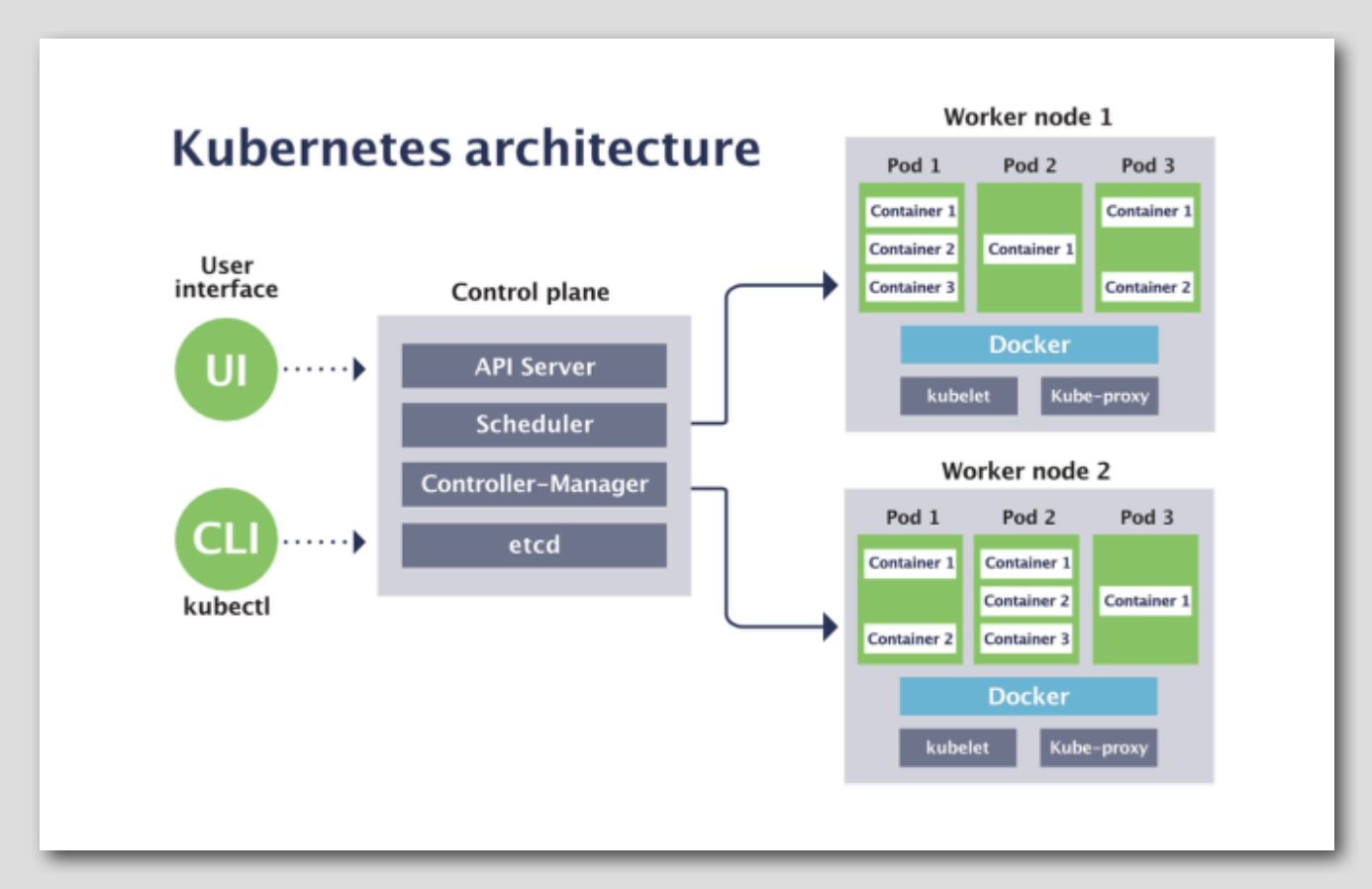
[101]



Above is the basic Kubernetes architecture complete with the following: **Kubernetes Control Plane Node** (Orchestration node), **Worker Nodes** (or just nodes where containers run), and the **Key Value Store (etcd)** that stores cluster data and participates in Kubernetes Control Plane functions.

Control Plane Node Architecture

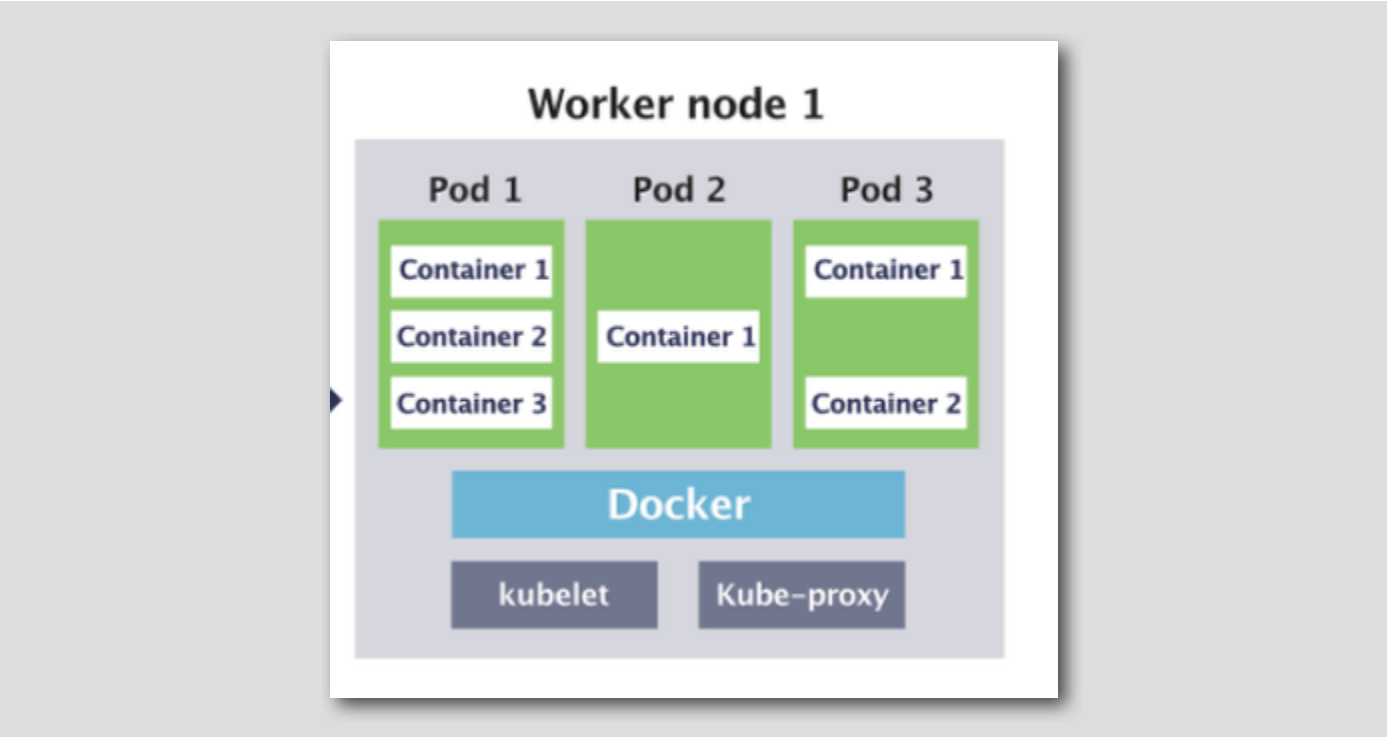
[102]



1. *Kubernetes API Server*
    - Exposes K8s REST API
    - Validates and configures data for API objects
    - Entry point for administrative tasks
2. *Controller*
    - A collection of several managers that monitor the state of the environment
    - Works to steer the cluster into the desired state
    - Acts as the Control Plane for Kubernetes
3. *Scheduler*
    - Component responsible for placing pods into nodes
    - Must consider factors like resource requirements, service requirements, hardware/software restrictions, data locality, deadlines, ect
4. etcd
    - Distributed Data Store for Kubernetes
    - Stores the "Desired" state of the system
    - Monitors the "Actual" state of the system
    - Can exists external to the master node itself.

Worker Node Architecture

[103]



1. *Kubelet*
    - Responsible for running the Pods (containers)
    - Acts as the Kubernetes agent running on each node
    - Ensures that the pod's containers are running and healthy
    - Provides the status of containers and pods on the node.
2. *Container Runtime (c runtime)*
    - Responsible for running the containers
    - Different flavors are supported, but the most common Container runtime is Docker.
    - Acts as the Data Plane or realized or "runtime" state
3. *Kube-Proxy*
    - Performs basic network functions on each node, including TCP and UDP forwarding
    - Implements East/West load-balancing using IPTables

Kubernetes Namespace

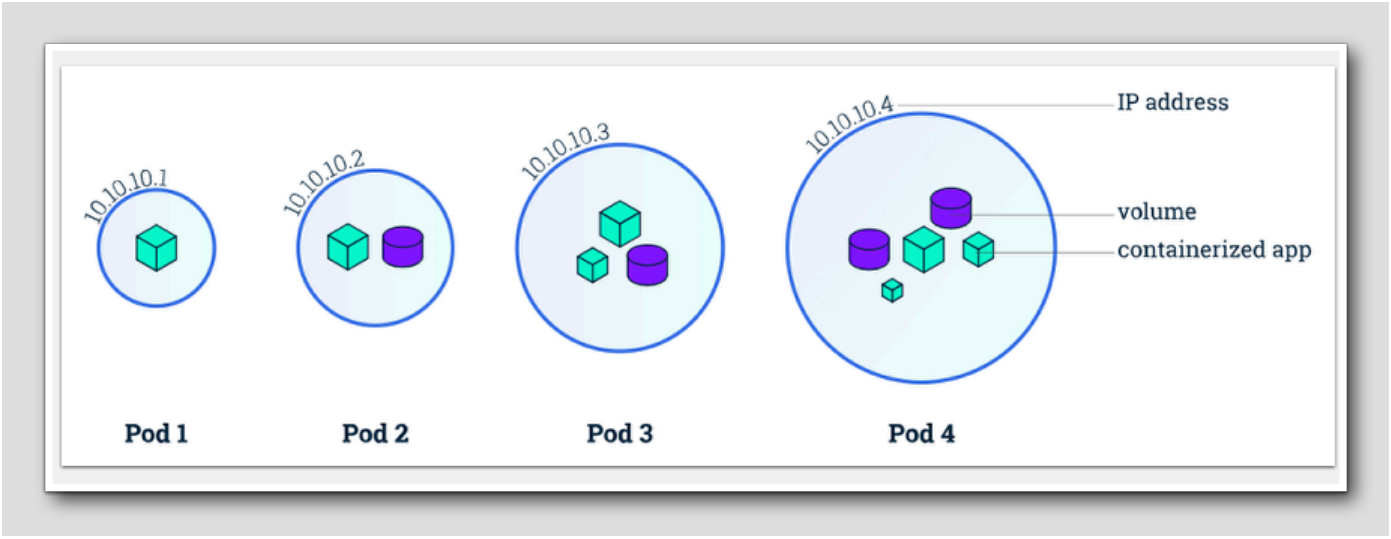
[104]

```
$ kubectl get namespaces
NAME          STATUS    AGE
default       Active    1d
kube-system   Active    1d
kube-public   Active    1d
```

A namespace is a virtual cluster that outlines the scope of the name. Each namespace is completely isolated from all other namespaces, or virtual clusters -- they can only communicate through public (known) interfaces. Names of resources should be unique within a namespace, but not across namespaces. Thus, different namepsaces can share the same resources.

Kubernetes Pod

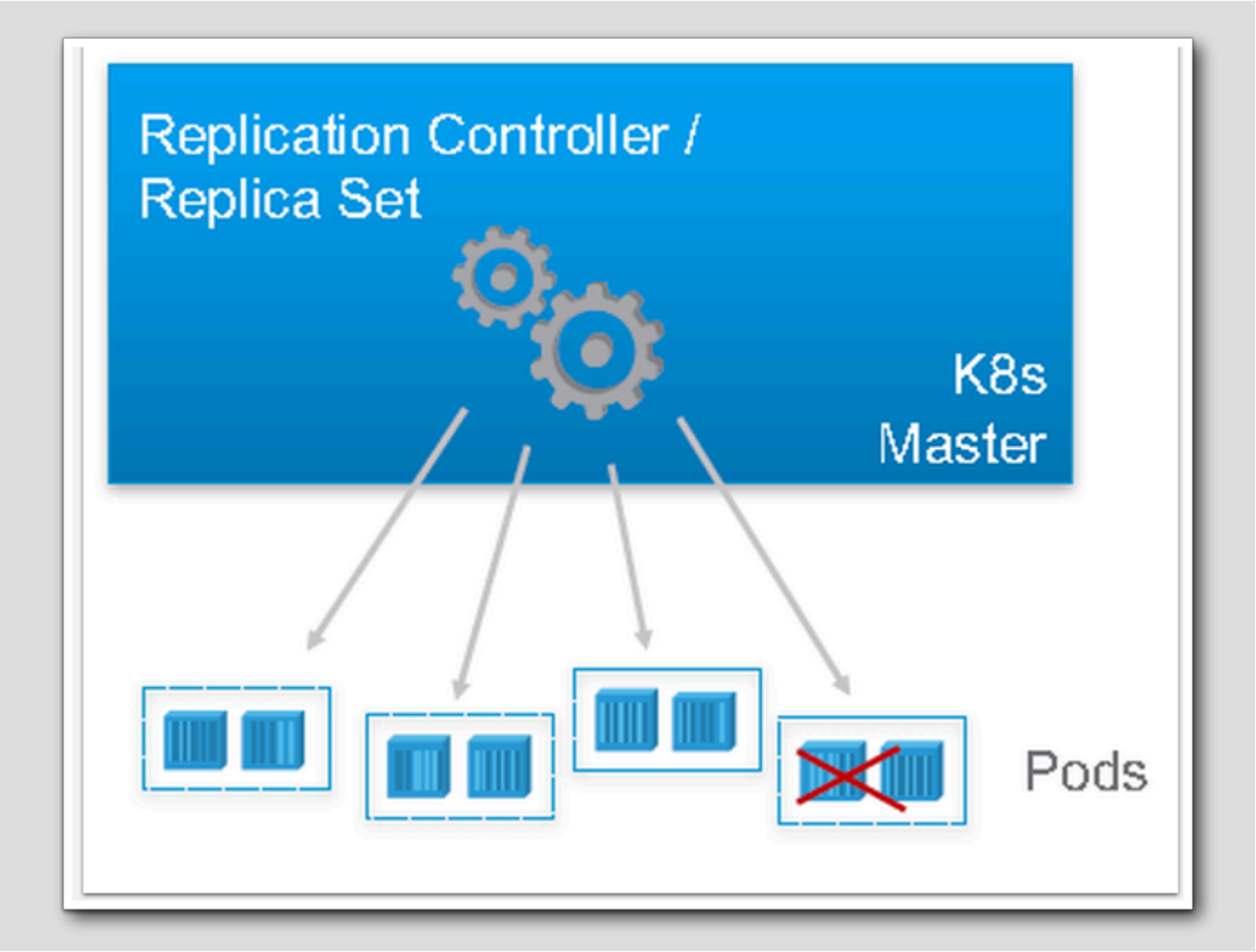
[105]



A Kubernetes Pod is a group of one or more containers. Pods are always scheduled on the same node. All containers in the pod share the same IP address and port space; they can communicate using localhost or standard inter-process communication. Moreover, all containers in a single pod have access to the same shared local storage on the node. Pods are a model of the pattern of multiple cooperating processes which form a cohesive unit of service and serve as unit of deployment, horizontal scaling, and replication. Co-Location (co-scheduling), shared fate (e.g. termination), coordinated replication, resource sharing, and dependency management are handled automatically for containers in a Pod.

Kubernetes Controller

[106]



The Kubernetes Controller runs on the control plane node and monitors the "actual state" of the system on the worker nodes.

Kubernetes Controller Components

[107]

A *Replication Controller* enforces the 'desired' state of a collection of Pods. E.g. it makes sure that 4 Pods are always running in the cluster. If there are too many Pods, it will kill some. If there are too few, the Replication Controller will start more. Unlike manually created pods, the pods maintained by a Replication Controller are automatically replaced if they fail, get deleted, or are terminated.

A *Replica Set* is the next-generation Replication Controller. The only difference between a ReplicaSet and a Replication Controller right now is the selector support. ReplicaSet supports the new set-based selector requirements whereas a Replication Controller only supports equality-based selector requirements.

A *Deployment Controller* provides declarative updates for Pods and ReplicaSets. You describe a desired state in a Deployment object, and the Deployment controller changes the actual state to the desired state at a controlled rate. You can define Deployments to create new ReplicaSets, or to remove existing Deployments and adopt all their resources with new Deployments.

Kubernetes Service

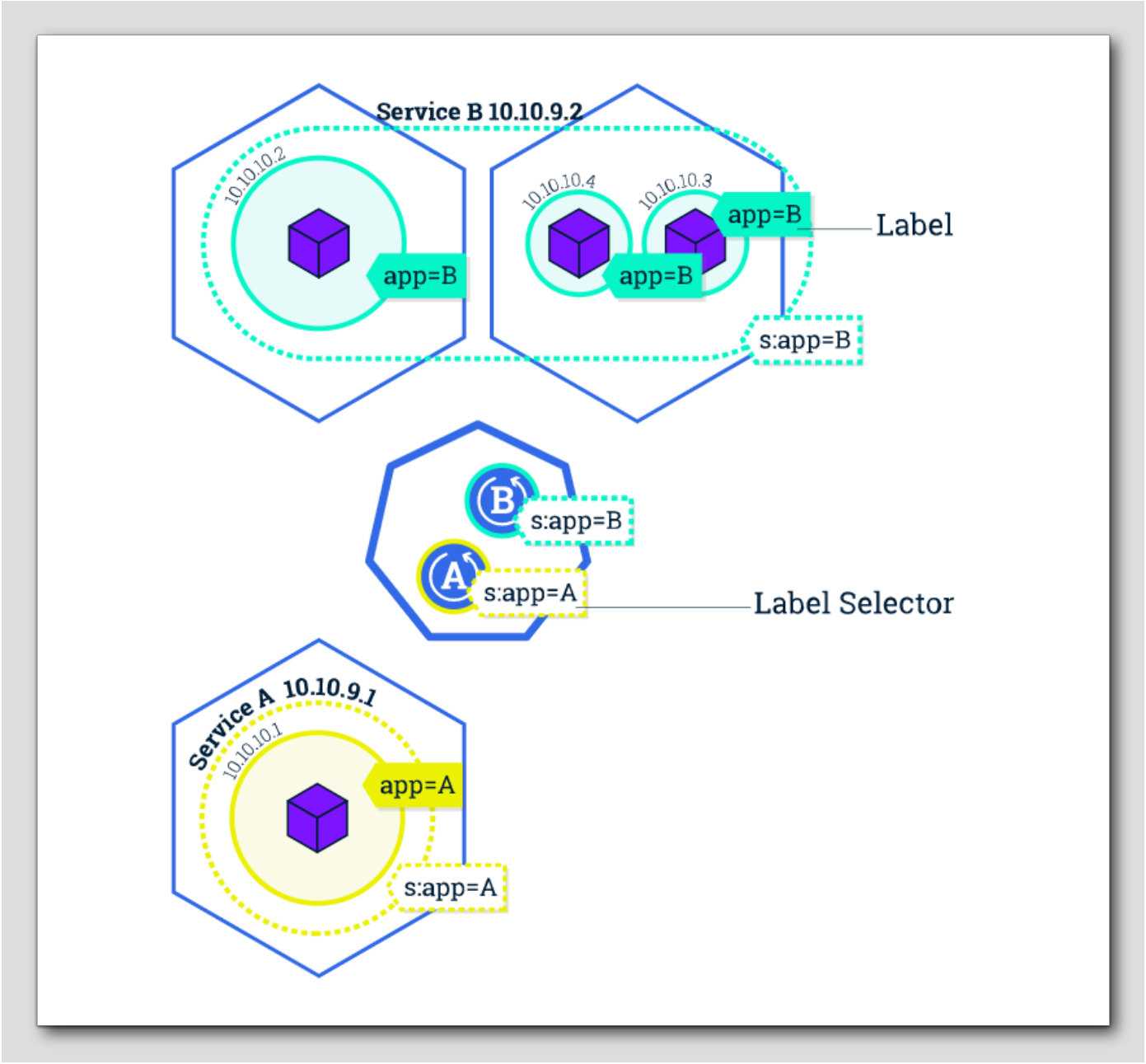
[108]

Just as a group of VMs may compose an application (or service), a group of Kubernetes pods composes a service in Kubernetes. A Service in Kubernetes is an abstraction which defines a logical set of Pods and a policy to which to access them. Services enable a loose coupling between dependent pods. The set of Pods targeted by a Service is usually determined by a LabelSelector.

Without defining a service, the pods will remain inaccessible and hidden from sources outside the Kubernetes cluster. Even though all pods are created with a unique IP, that IP is not exposed externally. Services clear a frontend path and allow access to the backend pods that make up an application. Thus, a service is essential for allowing access to a group of pods that together, perform a function or constitute an application.

Kubernetes Service Example

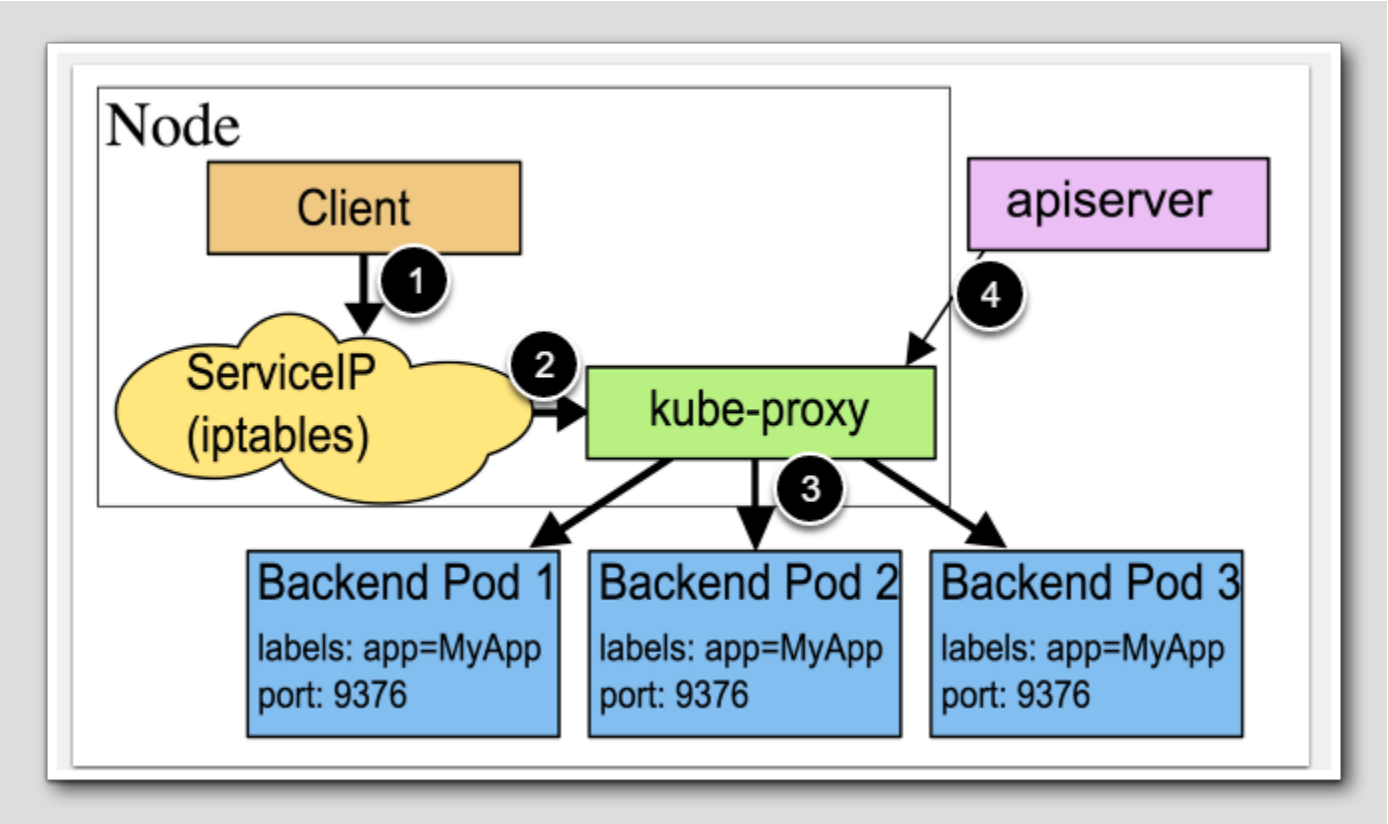
[109]



In the example seen in the previous image, we have two defined services: Service A with Label Selector "s:app=A" and Service B with Label Selector "s:app=B."
Service B is composed of 3 distinct pods that all share the same Label Selector. While external clients cannot access the internal IPs of the pods, they can access the IP exposed for Service B at 10.10.9.2. To access Service A, clients would need to reach 10.10.9.1. Therefore, while the backend pods may be in flux -- going up and down as required and different IP addresses being allocated each time, the service can remain constant. We get the best of both worlds -- a service that provides a consistent user experience and the desired flexibility and elasticity for the modern application.

Kubernetes Services Workflow

[110]



1. In the workflow image seen above, we can see that a **Client will access the application via the ServiceIP**, that is the externally exposed ingress point.
2. From there, the **Service IP directs traffic to the kube-proxy service**, which provides routing and load-balancing services to the backend pods.
3. The local **Kube-proxy service can then deliver the traffic to the appropriate pod(s)**.
4. **Kube-proxy also listens to the k8s apiserver** on the Master Node for environmental changes and adjusts accordingly.

Kube-Proxy

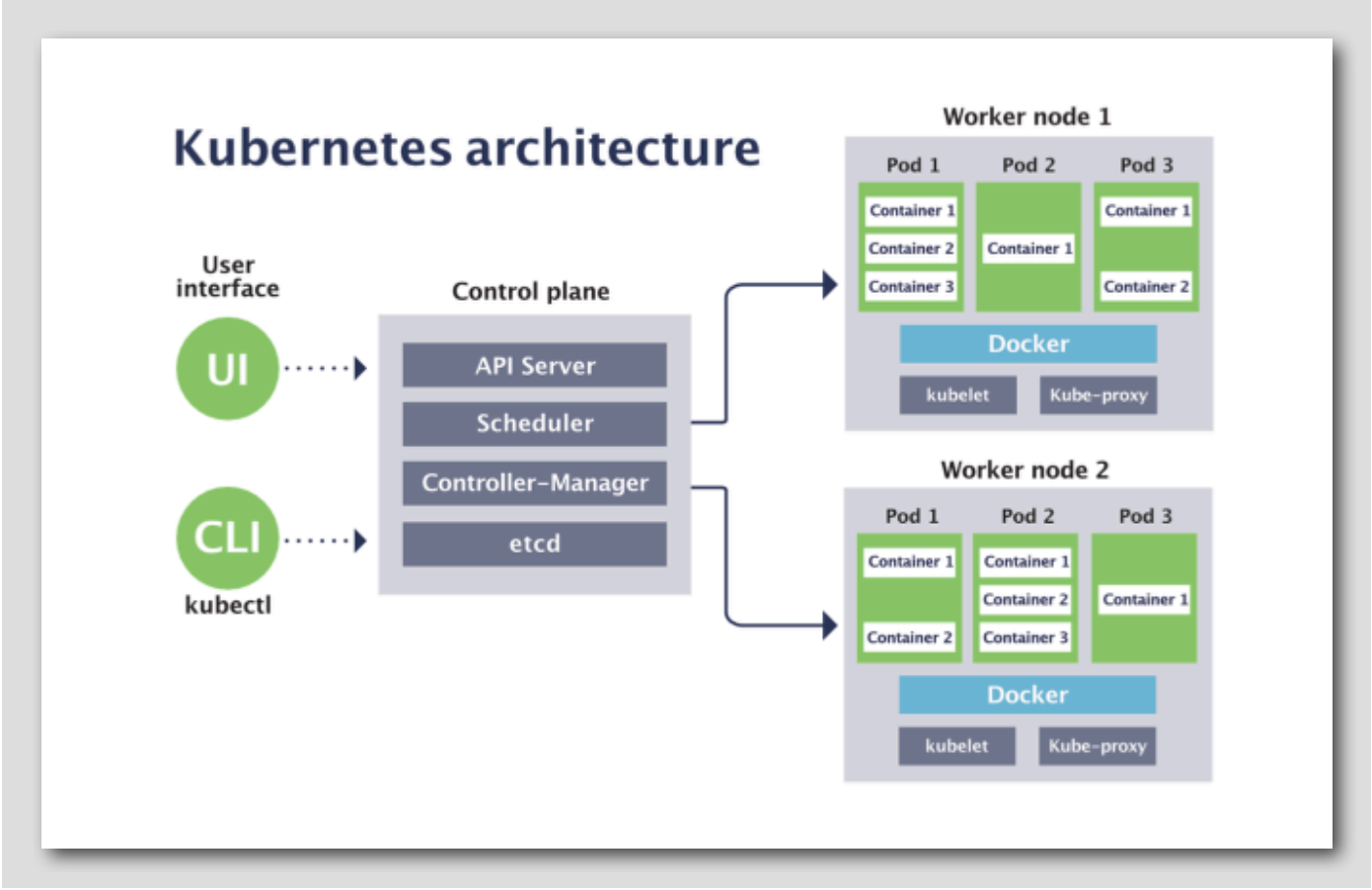
[111]

The kube-proxy watches the Kubernetes master for the addition and removal of Service and Endpoints objects. For each Service, it installs iptables rules which capture traffic to the Service’ s clusterIP (which is virtual) and Port and redirects that traffic to one of the Service’ s backend sets. For each Endpoints object, it installs iptables rules which select a backend Pod. By default, the choice of backend is random.

Kubernetes Architecture Deep Dive

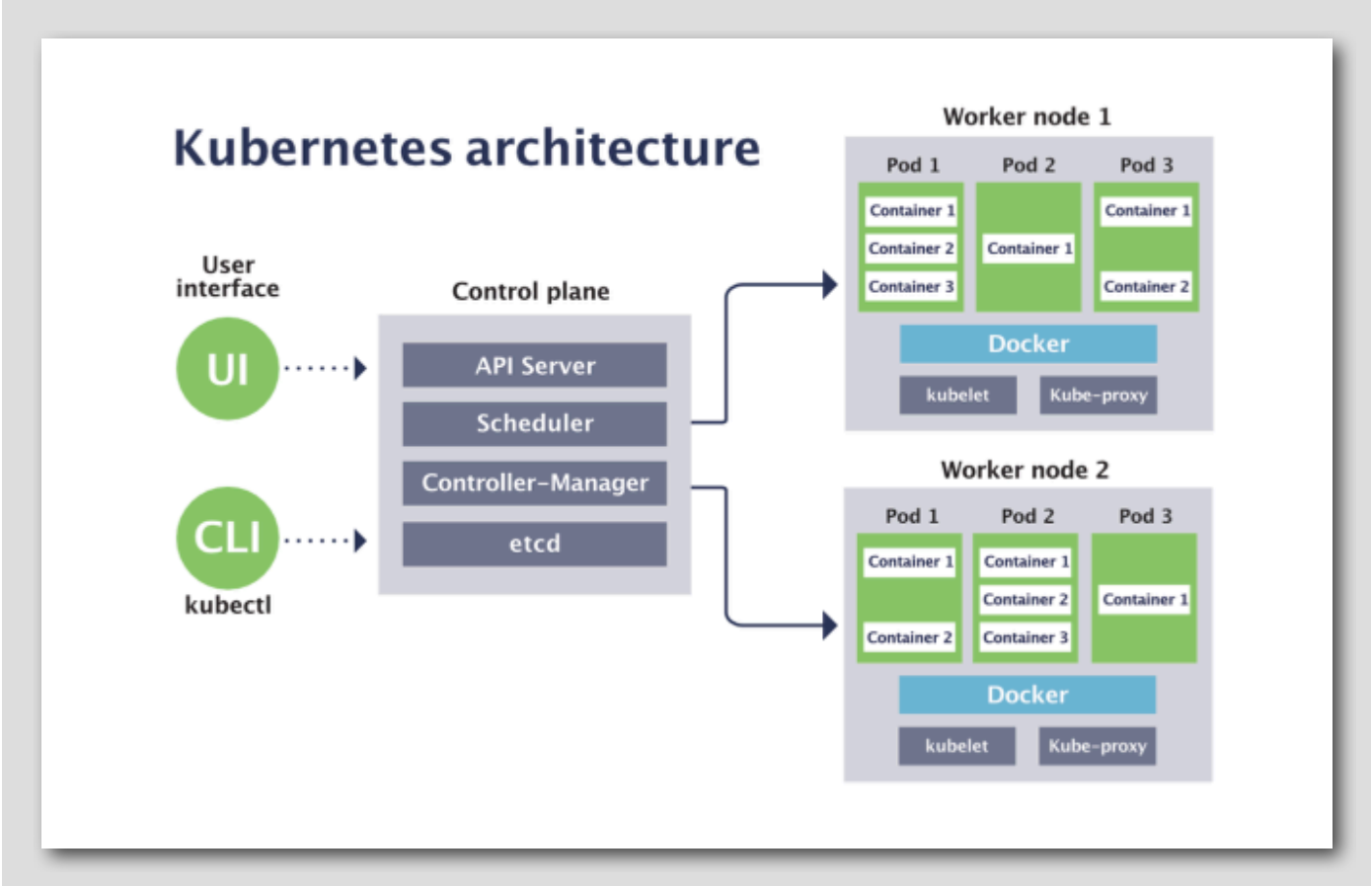
[112]

At a very high level, the Kubernetes cluster contains a set of services that may be contained in a single VM or broken out into multiple VMs. The Kubernetes control plane includes the Kubernetes API, which is a set of services used for all internal and external communications. Etcd is a distributed key value store that holds all persistent meta data for the Kubernetes cluster. The scheduler is a control plane service that is responsible for scheduling container workloads onto the Worker nodes. Worker nodes are VMs that are placed across ESXi hosts. Your applications run as a set of containers on the worker nodes. Kubernetes defines a container abstraction called a pod, which can include one or more containers. Worker nodes run the Kubernetes agent, called **Kubelet**, which proxies calls to the container runtime daemon (Docker or others) for container create/stop/start/etc. etcd provides an interesting capability for "Watches" to be defined on its data so that any service that must act when meta data changes simply watches that key:value and takes its appropriate action.

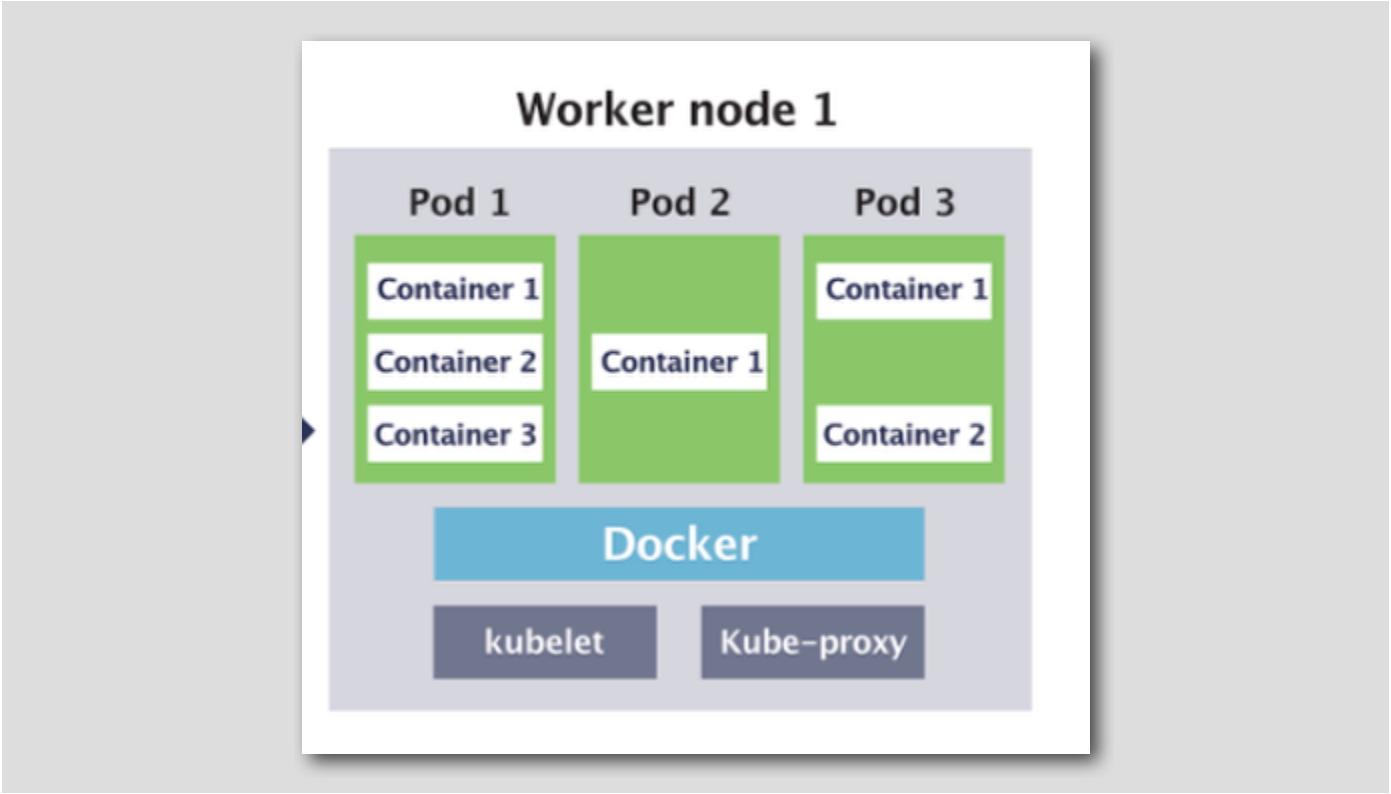


A Kubernetes cluster can have one or more control plane VMs and generally will have etcd deployed redundantly across three VMs.

- **API Server**: Target for all operations to the data model. External API clients like the Kubernetes CLI client, the dashboard Web-Service, as well as all external and internal components interact with the API Server by 'watching' and 'setting' resources
- **Scheduler**: Monitors container (pod) resources on the API Server, and assigns Worker nodes to run the pods based on filters
- **Controller Manager**: Embeds the core control loops shipped with Kubernetes. In Kubernetes, a controller is a control loop that watches the shared state of the cluster through the API Server and makes changes attempting to move the current state towards the desired state
- **Etcd**: Is used as the distributed key-value store of Kubernetes
- **Watching**: In etcd and Kubernetes everything is centered around 'watching' resources. Every resource can be watched on etcd through the API Server



Kubernetes Worker Nodes [113]



- **Kubelet:** The Kubelet agent on the nodes is watching for ‘PodSpecs’ to determine what it is supposed to run and Instructs container runtimes to run containers through the container runtime API interface. PodSpecs are defined through the yaml configuration files seen earlier.
- **Docker:** Is the most used container runtime in Kubernetes. However K8s is ‘runtime agnostic’, and the goal is to support any runtime through a standard interface (CRI-O)
- **Rkt:** Besides Docker, Rkt by CoreOS is the most visible alternative, and CoreOS drives a lot of standards like CNI and CRI-O (Check out https://www.cncf.io/ for more on these standards)
- **Kube-Proxy:** Is a daemon watching the K8s ‘services’ on the API Server and implements east/west load-balancing on the nodes using NAT in IPTables

Let's look at a sample workflow. This is a high level view and may not represent the exact workflow, but is a close approximation. A user wants to create a pod through the CLI, UI or using the API through their own code. The request comes to the Kubernetes API Server. The API Server instantiates a pod object and updates etcd with the information. The scheduler is watching for pod objects that have no node associated with it. The scheduler sees the new pod object and goes through its algorithm for finding a node to place the pod (available resources, node selector criteria, etc.). Scheduler updates the pod information (through the API Server) to include the placement node. On that node, Kubelet is watching etcd for a pod object that contains its node. Once it sees the new pod object, it begins to instantiate the pod. Kubelet will call the container runtime engine to instantiate the set of containers that make up the pod. Once the pod is running and has an IP address, that information is updated in etcd so that the new Endpoint can be found.

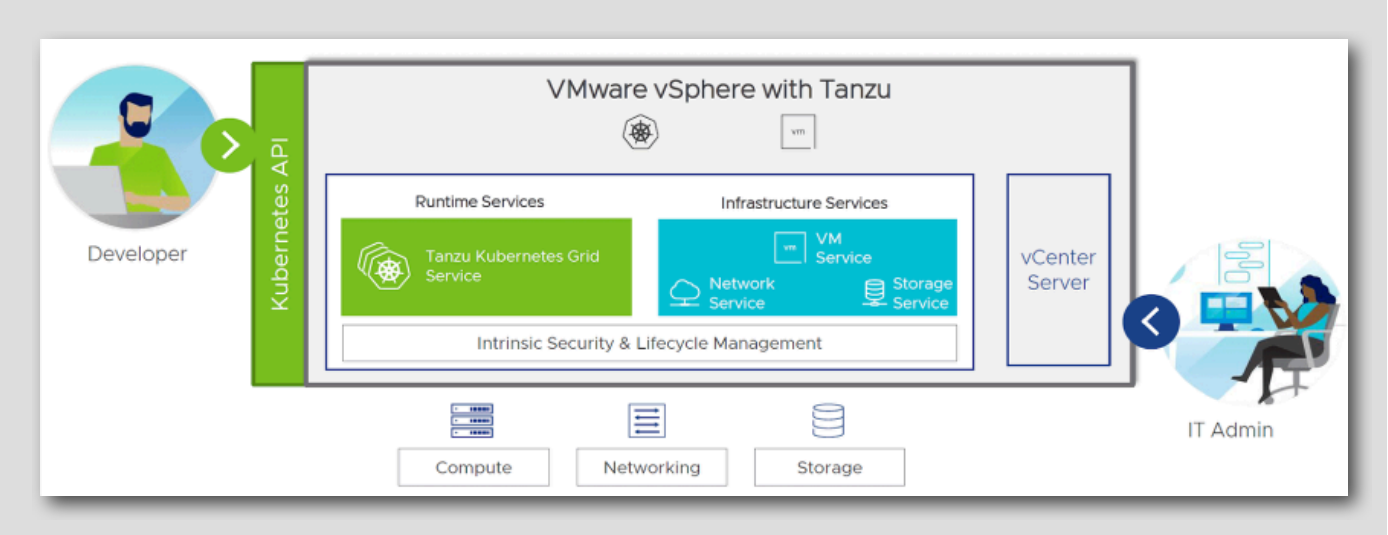
vSphere with Tanzu Introduction [114]

Common Platform for Running both Kubernetes/Containerized Workloads and VMs

Kubernetes is now built into vSphere with Tanzu which allows developers to continue using the same industry-standard tools and interfaces they've been using to create modern applications. vSphere Admins also benefit because they can help manage the Kubernetes infrastructure using the same tools and skills they have developed around vSphere. To help bridge these two worlds we've introduced a new vSphere construct called Namespaces, allowing vSphere Admins to create a logical set of resources, permissions, and policies that enable an application-centric approach.

What is vSphere with Tanzu? [115]

VMware vSphere with Tanzu delivers developer ready infrastructure and application-focused management for streamlined development, agile operations, and accelerated innovation. It's a flexible environment for modern applications that are built from microservices and run across heterogenous environments.



With vSphere with Tanzu, VMware delivers embedded Tanzu Kubernetes Grid Service for fully compliant and conformant Kubernetes capabilities for containerized applications. This approach provides Kubernetes APIs to developers, enabling CI/CD (continuous integration / continuous delivery) processes across a global infrastructure including on-premises data centers, hyperscalers, and Managed Service Providers (MSP) infrastructure. It unites the data center and the cloud with an integrated cloud operating model. Now enterprises can increase the productivity of developers and operators, enabling faster time-to-innovation combined with the security, stability, and governance, and avoid cost proliferation due to multiple stacks of IT infrastructure or cloud services.

Streamlined Development of Kubernetes Applications [116]

vSphere with Tanzu enables the DevOps model with infrastructure access for developers through Kubernetes APIs. It includes the Tanzu Kubernetes Grid Service, which is VMware's compliant and conformant Kubernetes implementation for building modern containerized applications. In addition, the vSphere Pod Service compliments the Tanzu Kubernetes Grid Service for application container instances requiring VM-like isolation benefits of improved performance and security of a solution built into the hypervisor.

Agile Operations for Kubernetes [117]

We are introducing a lot of value in vSphere with Tanzu for the VI admin. We deliver a new way to manage infrastructure, called application-focused management. This enables VI admins to organize multiple objects into a logical group and then apply policies to the entire group. For example, an administrator can apply security policies and storage limits to a group of virtual machines and Kubernetes clusters that represent an application, rather than to all of the VMs and clusters individually.

vSphere with Tanzu Delivers Essential Services for Hybrid Cloud

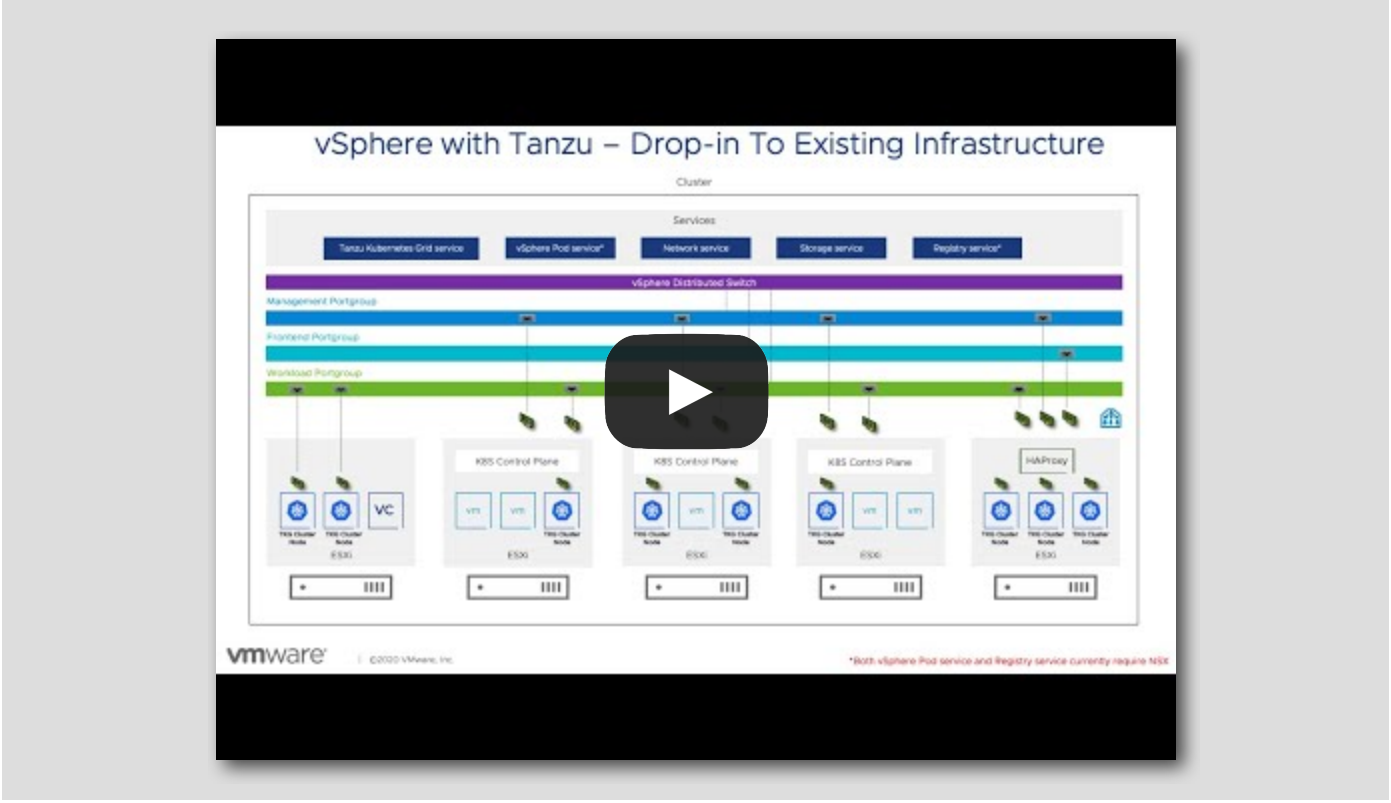
[118]

VMware solved the challenges faced by traditional apps across heterogeneous architectures with the introduction of VMware vSphere. With vSphere, we're delivering the essential services for the modern hybrid cloud. VMware hyperconverged infrastructure (HCI) stack combines compute, storage, and networking with unified management—vSphere with Tanzu powers the innovation behind developer ready infrastructure. With the new Kubernetes and RESTful API surface, developers can streamline their work and IT administrators can improve productivity using application-focused management.

Technical Overview of vSphere with Tanzu (4:25)

[119]

https://www.youtube.com/watch?v=0pI65Kn9AKk



vSphere with Tanzu delivers Developer ready infrastructure by allowing IT teams to use their existing vSphere environments to rapidly deploy Kubernetes clusters for their development teams. A simple enablement wizard gets you up and running in minutes. Learn how **Networking with vSphere Distributed Switch** works in this quick video

VMware vSphere with Tanzu Services

[120]

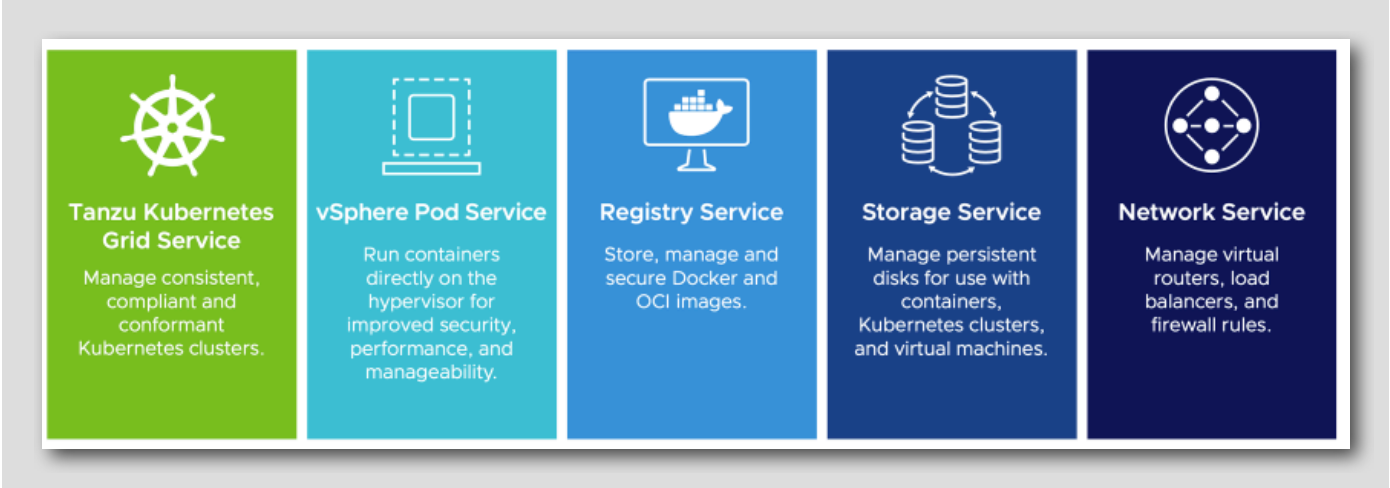
Powered by innovations in vSphere with Tanzu, vSphere with Tanzu Services is a new, integrated Kubernetes and RESTful API surface that enables you to drive API access to all core services.

VMware vSphere with Tanzu Services Explained

[121]

VMware vSphere with Tanzu consists of two families of services: Tanzu Runtime Services and Infrastructure Services.

- Tanzu Runtime Services deliver core Kubernetes development services, including an up-to-date distribution of Tanzu Kubernetes Grid.
- Infrastructure Services include full Kubernetes and RESTful API access that spans creating and manipulating virtual machines, containers, storage, networking, and other core capabilities.



* vSphere Pod Service requires NSX; In vSphere 8u1 the Registry Service can now be used with vSphere networking or NSX

Supervisor Services (new in vSphere 8 update 1)

[122]

Supervisor Services are vSphere certified Kubernetes operators that deliver Infrastructure-as-a-Service components and tightly-integrated Independent Software Vendor services to developers. You can install and manage Supervisor Services on vSphere with Tanzu and make them available for use with Kubernetes workloads. When Supervisor Services are installed on Supervisor clusters, DevOps engineers can use the service APIs to create instances on Supervisors in their user namespaces. These instances can then be consumed in vSphere Pods and Tanzu Kubernetes Grid clusters.

Tanzu Kubernetes Grid Service [123]

A Tanzu Kubernetes Grid (TKG) cluster is a Kubernetes (K8s) cluster that runs inside virtual machines on the Supervisor layer and not on vSphere Pods. It is enabled via the Tanzu Kubernetes Grid Service for vSphere. Since a TKG cluster is fully upstream-compliant with open-source Kubernetes it is guaranteed to work with all your K8s applications and tools. Tanzu Kubernetes clusters are the primary way the customers will deploy Kubernetes based applications.

TKG clusters in vSphere use the open source Cluster API project for lifecycle management, which allows developers and operators to manage the lifecycle (create, scale, upgrade, destroy) of conformant Kubernetes clusters using the same declarative, Kubernetes-style API that is used to deploy applications on Kubernetes.

VM Service [124]

The VM Service allows a developer (or any DevOps, platform operations, or Kubernetes user) to deploy and manage virtual machines using Kubernetes standard APIs, while simultaneously allowing the IT administrator to govern resource consumption and service availability.

This capability joins the Network Service and Storage Service that have been available since VMware originally released vSphere with Tanzu last year, collectively giving developers self-service access to the compute, network, and storage resources they need to run their applications, using a Kubernetes API surface. With the VM Service, vSphere with Tanzu further empowers organizations to modernize their applications and enhance the cloud-native experience for their developers. The VM Service was added in vSphere 7 U2a.

Storage Service [125]

The Storage Service exposes vSphere or vSAN based storage and provides developers and operators the capability to manage persistent disks for use with containers, Kubernetes clusters and virtual machines.

Network Service [126]

The Network Service abstracts the underlying virtual networking infrastructure from the Kubernetes environment. It can be implemented using VMware vSphere Networking or NSX. It provides network services for Supervisor Cluster control plane VMs as well as TKG cluster control plane and worker nodes. If NSX is leveraged there are additional capabilities around Kubernetes service load balancing and network security.

vSphere Pod Service* [127]

The vSphere Pod Service is a service that runs on a VMware managed Kubernetes control plane over your ESXi cluster. It allows you to run native Kubernetes workloads directly on ESXi. The ESXi hosts become the Kubernetes Nodes and vSphere Pods are the components that run the app workloads.

The vSphere Pod Service provides a purpose-built lightweight Linux kernel that is responsible for running containers inside the guest. It provides the Linux Application Binary Interface (ABI) necessary to run Linux applications. Since this Linux kernel is provided by the hypervisor, VMware has been able to make numerous optimizations to boost its performance and efficiency. When users need the security and performance isolation of a VM, orchestrated as a set of containers through Kubernetes, they should use the vSphere Pod Service.

* vSphere Pod Service requires NSX

Registry Service* [128]

The Registry Service allows developers to store, manage and better secure Docker and OCI (Open Container Initiative) images using Harbor as a private registry. The lifecycle of projects and members of the private image registry is automatically managed and is linked to the lifecycle of namespaces and user or group permissions in namespaces created in vCenter.

* Registry Service requires NSX pre 8.0u1. The Supervisor Service for Harbor can be leveraged post 8.0u1 to deploy container based Harbor to the environment.

Quick Tour of Kubernetes [129]

Now that we've had a tour of the architecture and some of the terminology let's take a quick tour of some of the basic Kubernetes functionality.

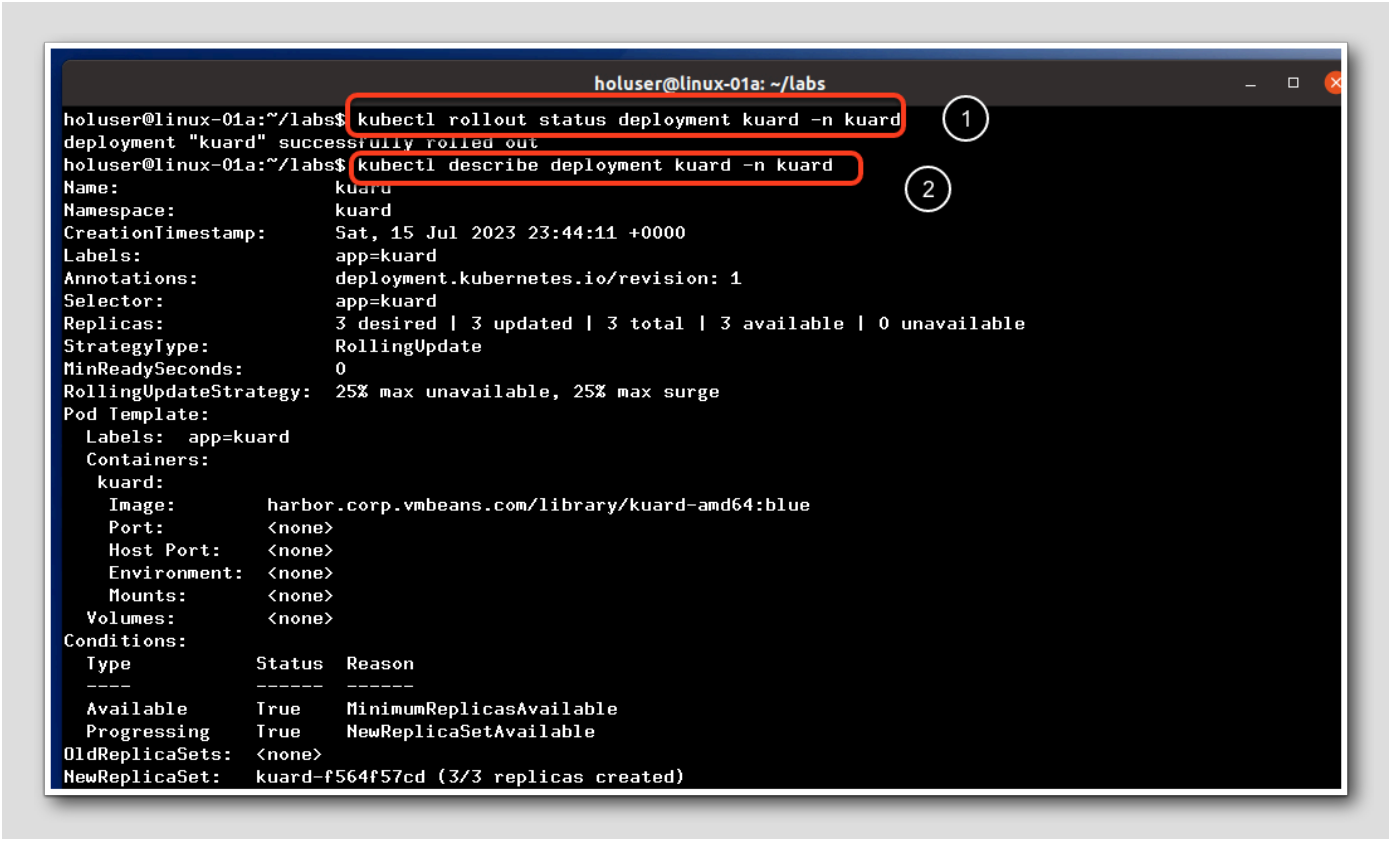
Try this: append the command above with a "-w"

```
kubectl get pods -n kuard -w
```

You will note that the command lists the pods and doesn't return to the command promot.

The "-w" is the WAIT option, it will show the status and wait for any additional actions. Since our pods are already deployed this command will never return. The "-w"is useful with executing a larger job with multiple deployment steps.

Hit CTRL-C to return to the command prompt.



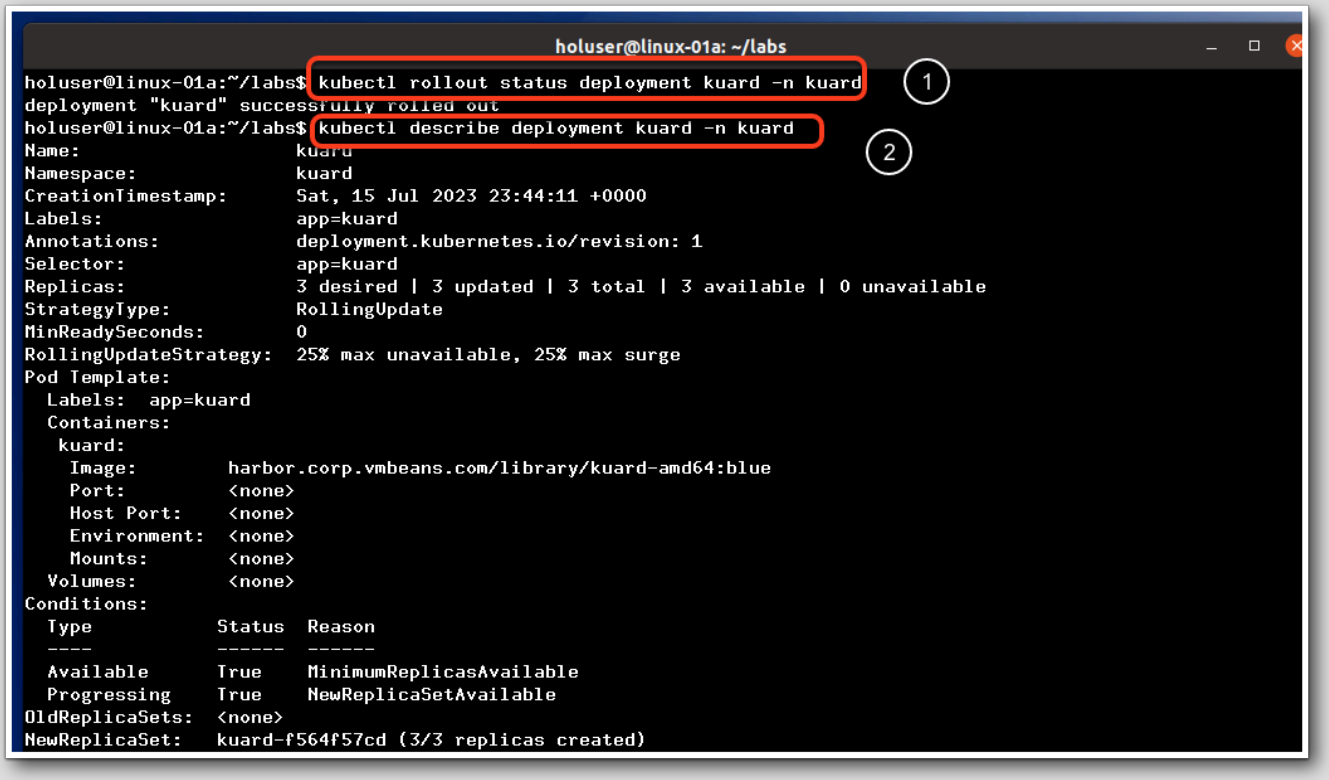
We can also check the status of the kuard deployment several ways:

1. `kubectl rollout status deployment kuard -n kuard`
2. `kubectl describe deployment kuard -n kuard`

Take a minute or so to look through the information displayed by the kubectl describe command.

The kubectl rollout status deployment command displays the rollout status along with the rollout progress.

The kubectl describe deployment command provides detailed information about a specific Deployment resource in Kubernetes. The command will display information like the number of replicas, the strategy used for updates, the image being used, the current status of the Deployment, any events related to the Deployment, and more.



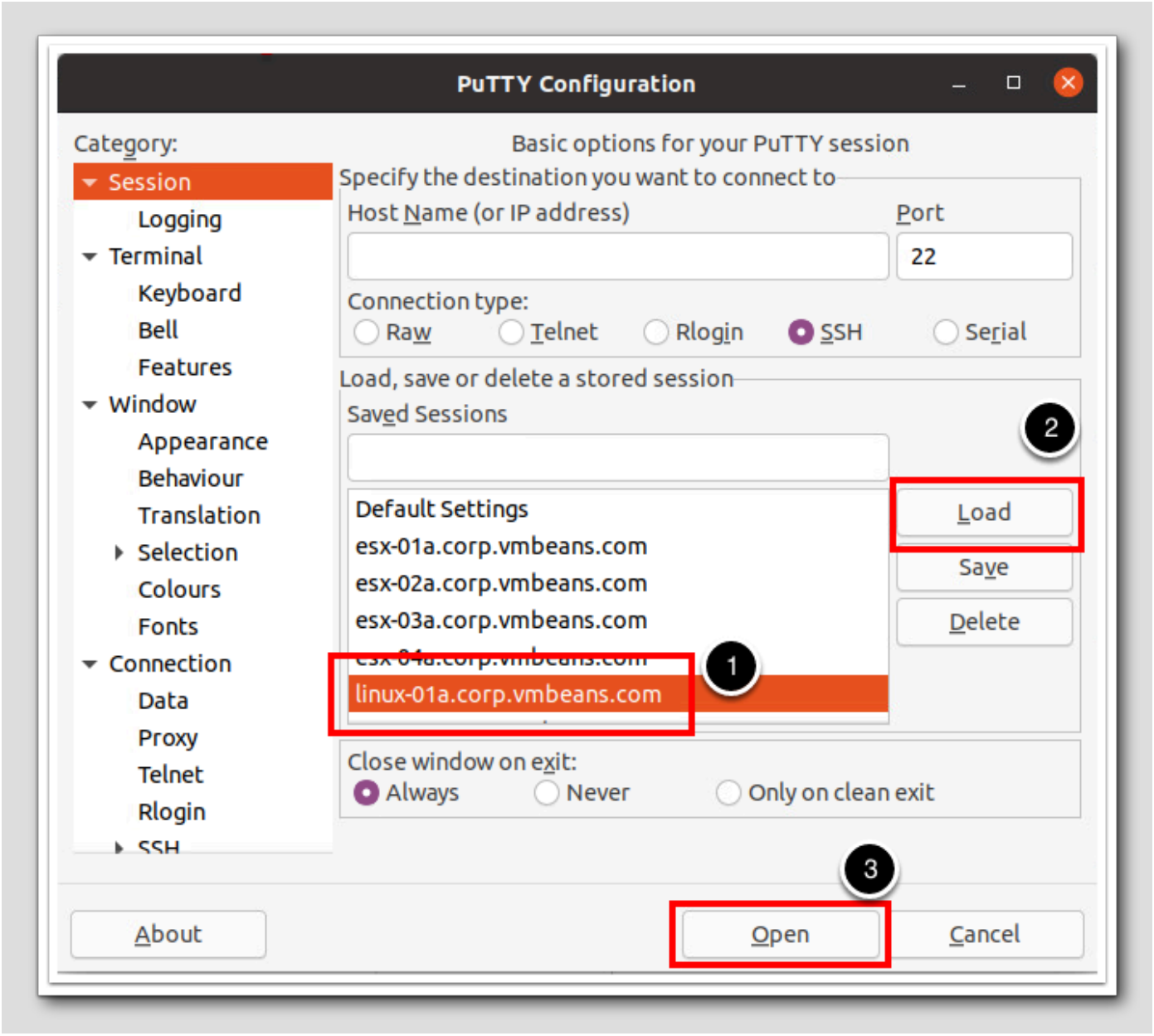
Connect to A Ubuntu Image Running Kubernetes

[130]

In our lab, we have a Linux server configured with all the tools needed to get through the lab lessons. We access this box via PuTTY, which already has a saved session for this machine.

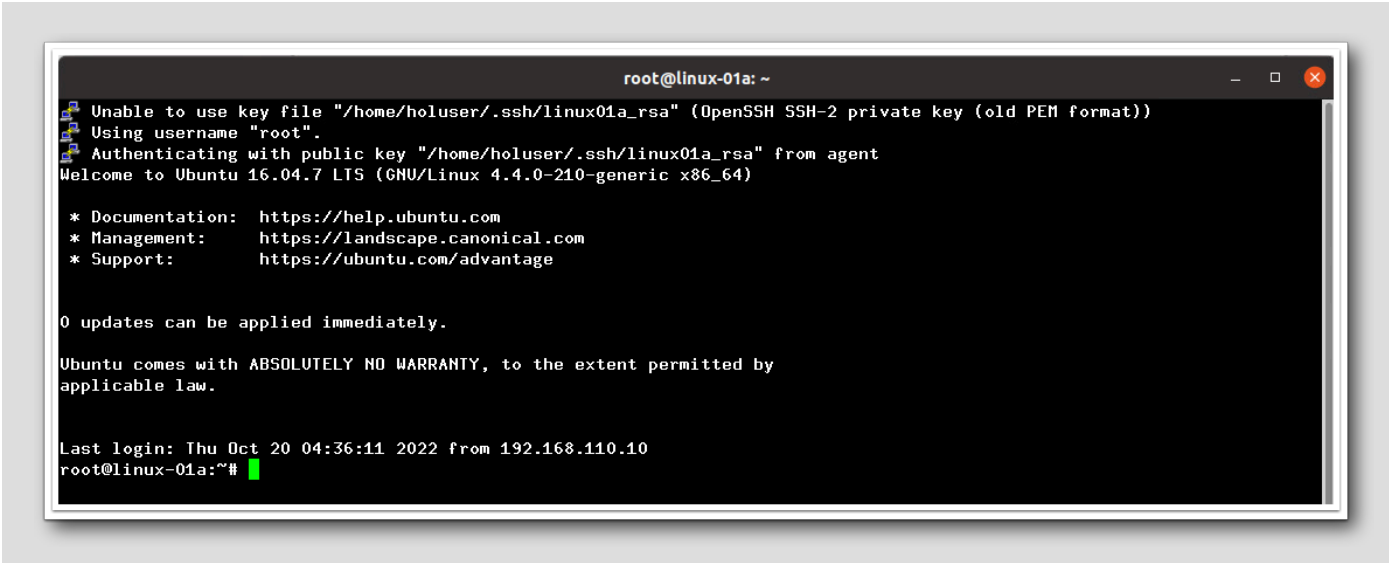
NOTE: If you are already connected to the linux-01 image skip this step.

1. Click on the**PuTTY**icon in the taskbar to open a new SSH session.



Login using SSH to Linux-01a VM:

1. Click on **linux-01a.corp.vmbeans.com** under Saved Sessions.
2. Click the **Load** button.
3. Click **Open**.



After a few seconds, you should be logged in as the root user. You should not be asked for a password as that is handled through key based authentication.

Login To The vSphere Kubernetes Instance

[131]

This lab is deployed on vSphere 8 and is using Tanzu Kubernetes Grid (TKG)

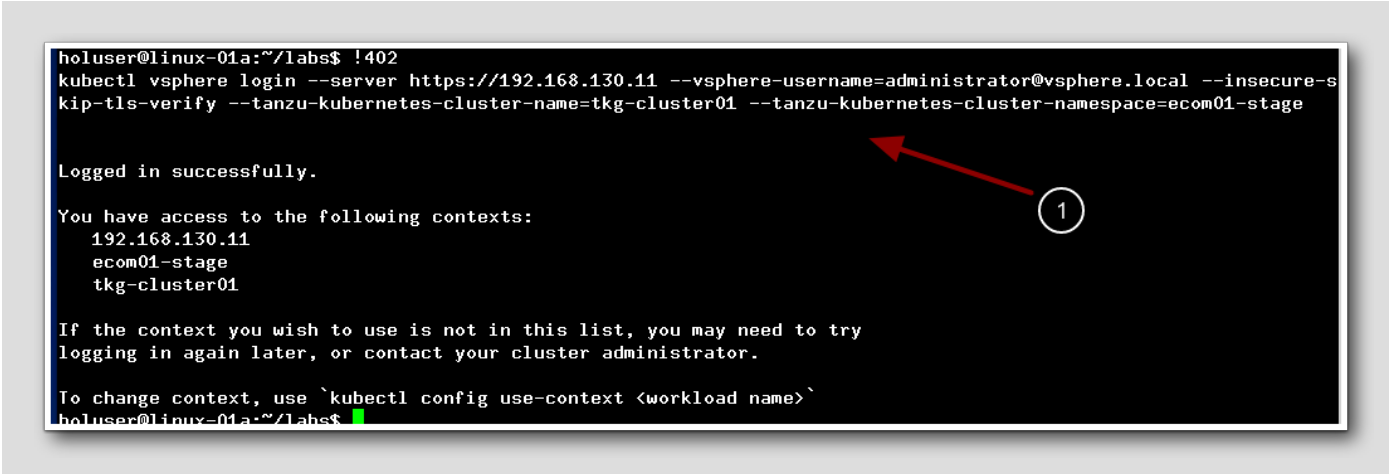
Before we access our kubernetes environment we must login to the Tanzu Kubernetes cluster. Cluster authentication is integrated with vSphere Single Sign On through the vSphere Kubernetes Plugin. This plugin is already installed in the Linux-01 workstation.

Log in to Supervisor Cluster:

1. Type or copy/paste the following command into Putty: `kubectl vsphere login --server=https://192.168.130.11 -u administrator@vsphere.local --tanzu-kubernetes-cluster-namespace=ecom01-stage --tanzu-kubernetes-cluster-name tkg-cluster01 --insecure-skip-tls-verify`

The supervisor Cluster authentication is integrated with vSphere Single Sign On through the vSphere Kubernetes Plugin. This plugin is already installed in the Linux-01 workstation.

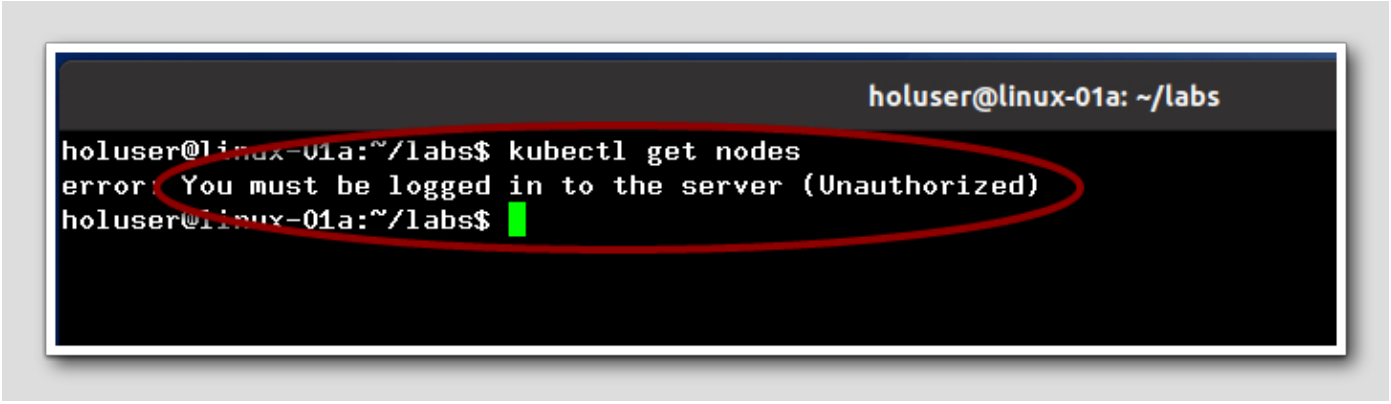
You should not be prompted for a password since it's been saved for you as a shell variable but if you are prompted for a password type or copy/paste the password:VMware1!



Important: The login tokens for the supervisor cluster are good for 12 hours so you should not to login to the supervisor cluster again for the duration of this lab.

Since all of the modules in this lab were made to be taken independently you may see another place where you are asked to login to the supervisor cluster, if you have already logged in you can ignore that instruction.

However if you see the message below when executing a kubectl comand you need to repeat the supervisor cluster login.



Verify The Version of Kubernetes And Check The Environment

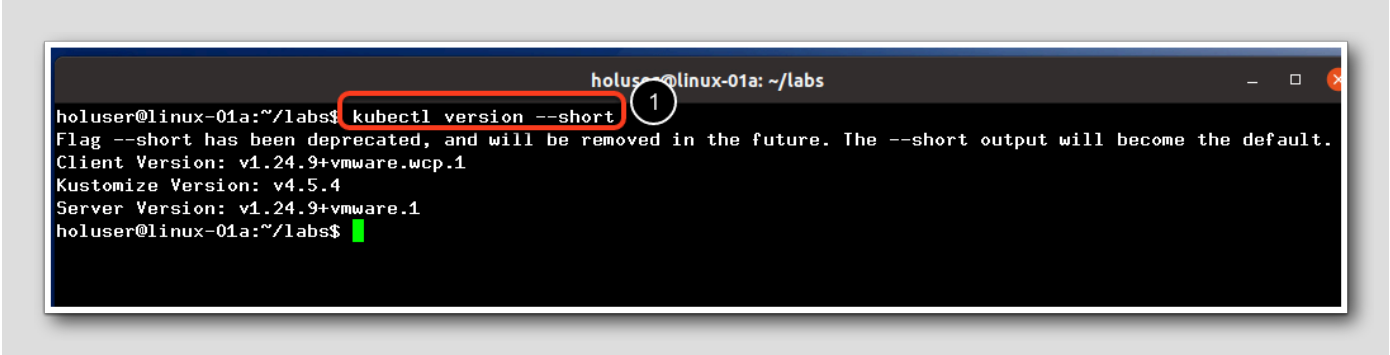
[132]

Before deploying the application lets verify that the kubernetes cluster is up and running and verify the version of kubernetes which is deployed in this lab. You can find information on the current version of kubernetes at: *https://kubernetes.io/*

1. Type: **`kubectl version --short`**

The --short option limits the output. Feel free to try the same command without the --short option and observe the difference.

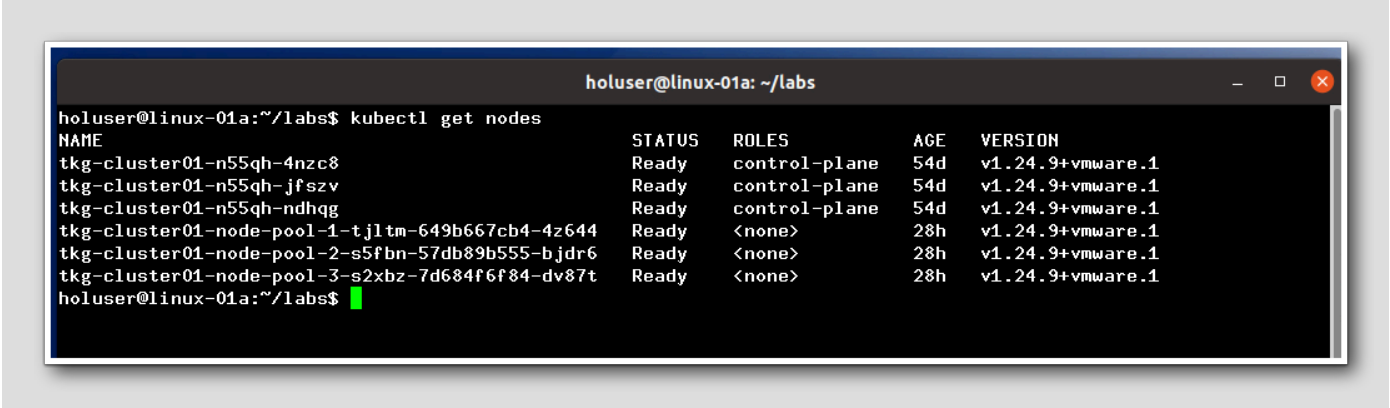
Ignore the message that the --short flag is being deprecated. Kubernetes is a rapidly evolving platform and as with any such software, the feature set naturally evolves over time, and sometimes a feature may need to be removed. This could include an API, a flag, or even an entire feature.To avoid breaking existing users, Kubernetes follows a deprecation policy for aspects of the system that are slated to be removed.



1. Type: **`kubectl get nodes`**

You will notice there are 6 nodes, three control plane node and 3 worker nodes in your deployed cluster.

Note this is a limited environment due to the limitations of the Hands On Labs environment, in a true production environment you would deploy multiple control plane nodes and additional worker nodes.



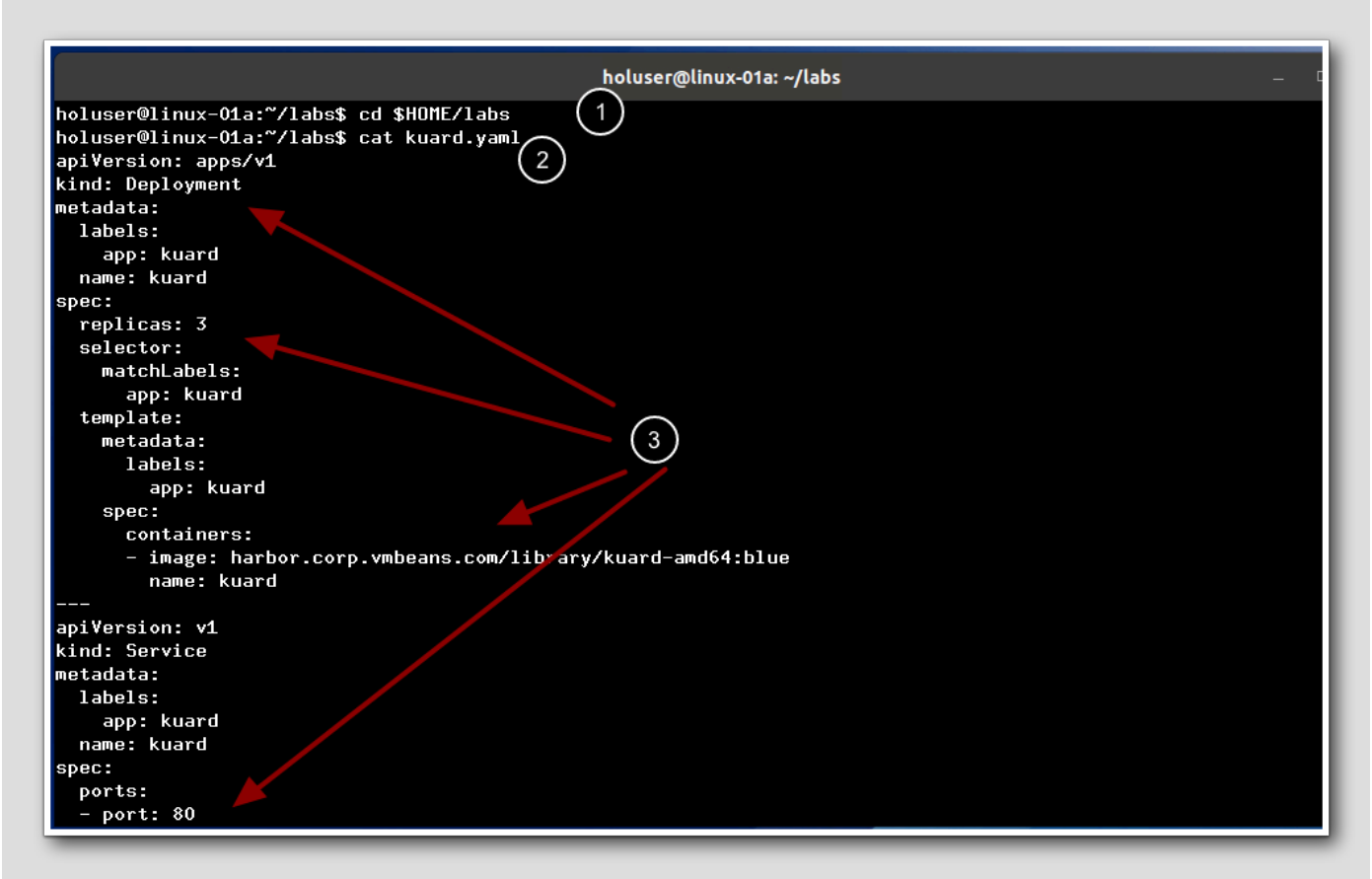
Deploy A Simple Kubernetes Application

[133]

Let's take a look at a sample Yaml file, YAML(YAMLAin't Markup Language) is a data-oriented language structure used as the input format for diverse software applications.

Typically Yaml is used in Kubernetes for deployment of resources such as such as pods, services, and deployments.

1. `cd $HOME/labs`
2. `cat kuard.yaml` and press Enter



To get familiar with some basic Kubernetes commands we will deploy a simple application and then examine that application using some standard commands.

We will deploy a simple application called kuard , it's demo application for Kubernetes, and stands for "Kubernetes up and Running Demo".

We will start by looking at the yaml deployment file, if you reviewed the previous module you will remember that YAML stands for *Yet Another Markup Language* and is the standard way in which Kubernetes pods and services are deployed.

Let's look at the components that make up this particular application yaml file and some of their settings

3 -kind: Deployment- A deployment declares the desired state of your Pods.

replicas: This is the number of Pod replicas that are desired

image: The location and image for the pods to use. Note that harbor-01.corp.local is our local image registry

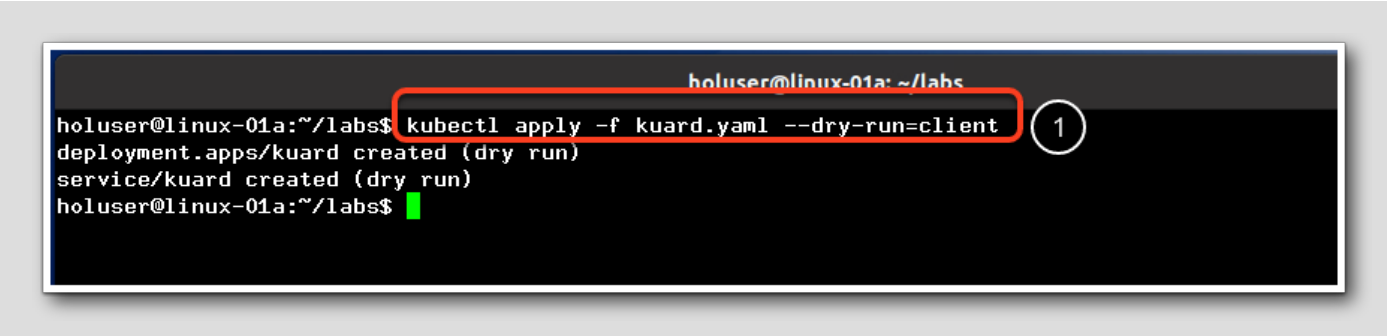
kind: Service- An abstract way to expose an application running on a set of Pods as a network service.

port:This is the port exposing the Kubernetes service. We will use this port to connect to the Pods

Deploy The Kuard Application - Part 1

[134]

Before we deploy the application let's take a look at the resources which executing the kuard.yaml will deploy. Kubernetes gives us a simple way to do this, take a dry run with a command which is appropriately named "dry-run".



Type : **`kubectl apply -f kuard.yaml --dry-run=client`**

Observe the output, this is what *WOULD* be deployed.

Kubernetes is a large system with many components and many contributors. As with any such software, the feature set naturally evolves over time, and sometimes a feature may need to be removed. This could include an API, a flag, or even an entire feature. To avoid breaking existing users, Kubernetes follows a deprecation policy for aspects of the system that are slated to be removed.

Deploy The Kuard Application - Part 2

[135]

Now that we have seen what *would be* deployed when we execute the kuard.yaml file let's actually deploy it. We'll start by creating a namespace to run the application in.

Namespaces in Kubernetes help segregate resources for different projects, groups, etc. Let's take a look at our existing namespaces and create one for our kuard deployment.

Let's make sure that we are in the default namespace:

1. type: `kubectl config set-context --current --namespace=default`
2. Type **`kubectl get namespaces`**. Note the existing namespaces

This command sets the context for the current namespace. As we discussed previously kubernetes has the concept of namespaces.

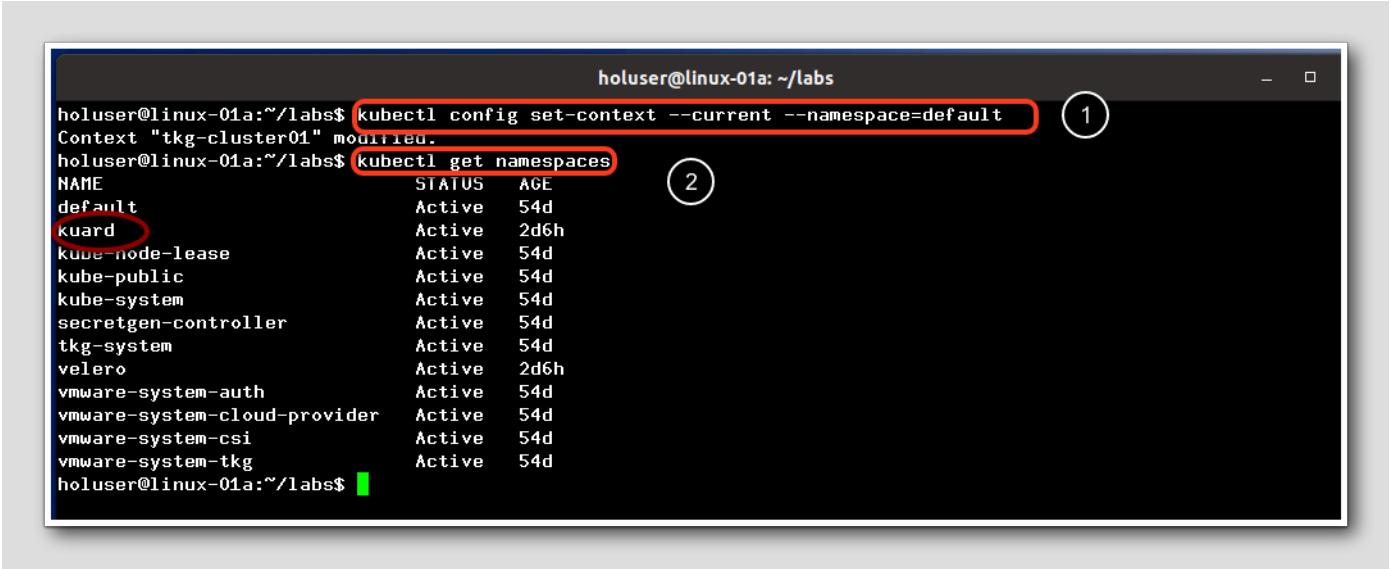
A Kubernetes context is**a group of access parameters that define which cluster you're interacting with, which user you're using, and which namespace you're working in**. It's helpful if you need to access different clusters for different purposes or if you want to limit your access to certain parts of a cluster.

Namespaces are intended for use in environments with many users spread across multiple teams, or projects. For clusters with a few to tens of users, you should not need to create or think about namespaces at all. Start using namespaces when you need the features they provide.

Namespaces provide a scope for names. Names of resources need to be unique within a namespace, but not across namespaces. Namespaces cannot be nested inside one another and each Kubernetes resource can only be in one namespace.

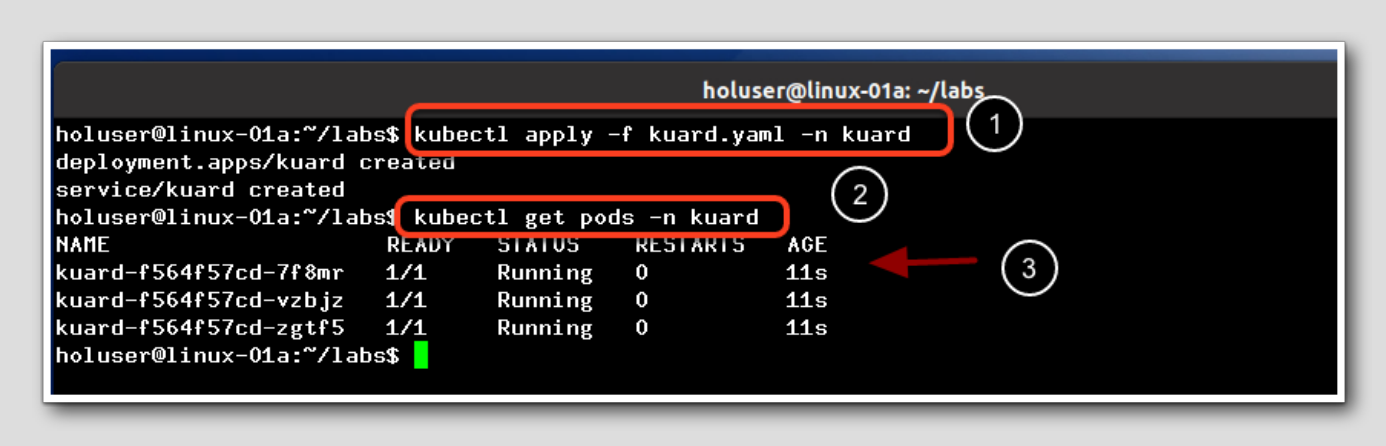
Namespaces are a way to divide cluster resources between multiple users.

The default namespace for kubernetes is called, you guessed it "default".



You should have a "kuard" namespace we've pre-created it for you. However if for some reason that namespace was deleted you cab create it:

1. Type **`kubectl create namespace kuard`** and press Enter



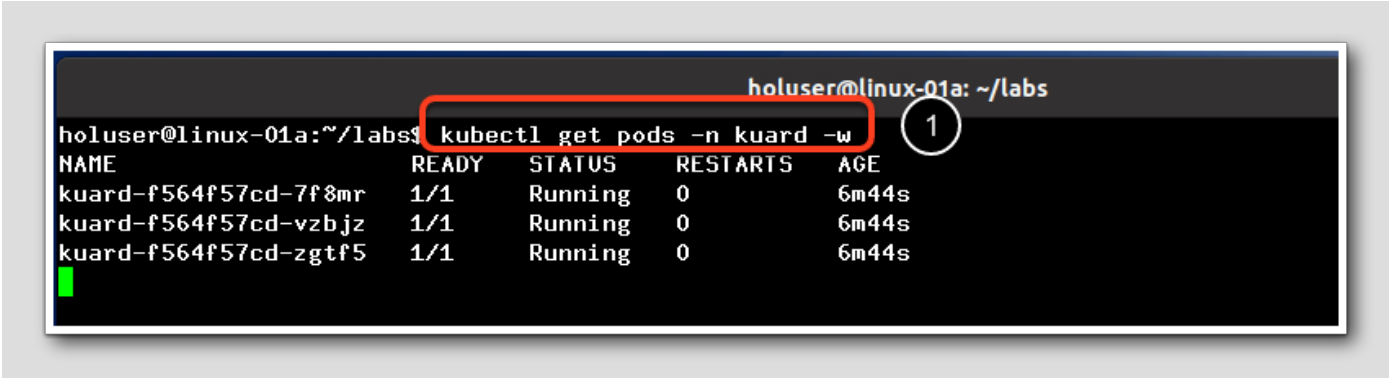
1. Type **`kubectl apply -f kuard.yaml -n kuard`** and press Enter
2. Type **`kubectl get pods -n kuard`** and press. Enter.
3. Note we have 3 replicas/pods in our deployment. if you recall when we looked at the kuard.yaml file there were 3 deployments in the Yaml file.

Wait for all pods to have a status of "running" before moving on to the next step.

Kubernetes operators are declarative, which means they reconcile the current state of Kubernetes cluster with the desired state of the Kubernetes cluster, conceptually in a continuous loop, the control loop. A declarative approach expresses computation without explicitly expressing a sequence of steps. In the kuard.yamp file we created a file that describes the configuration for the particular resource and then apply the content of the file to the Kubernetes cluster

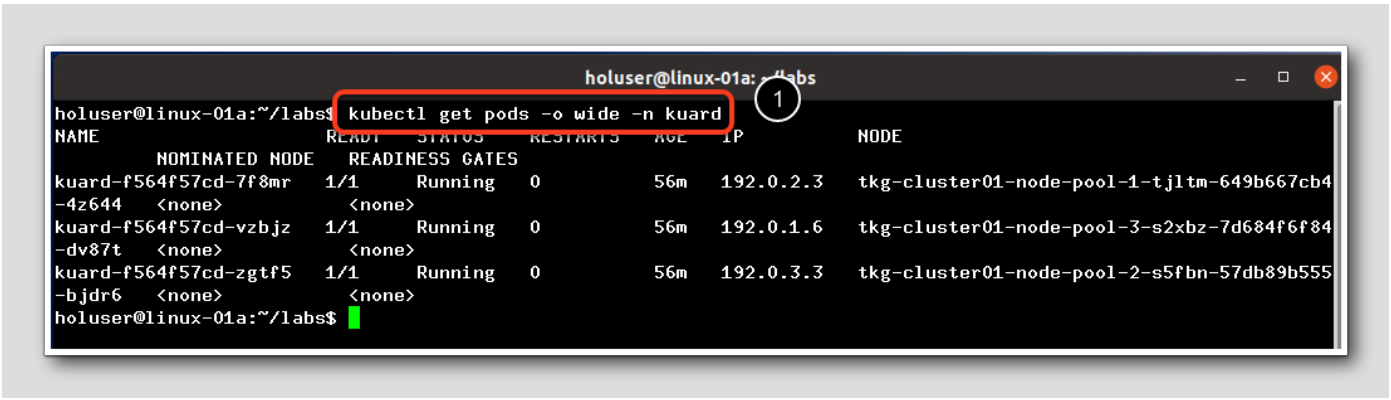
Looking at our Kuard deployment

[136]



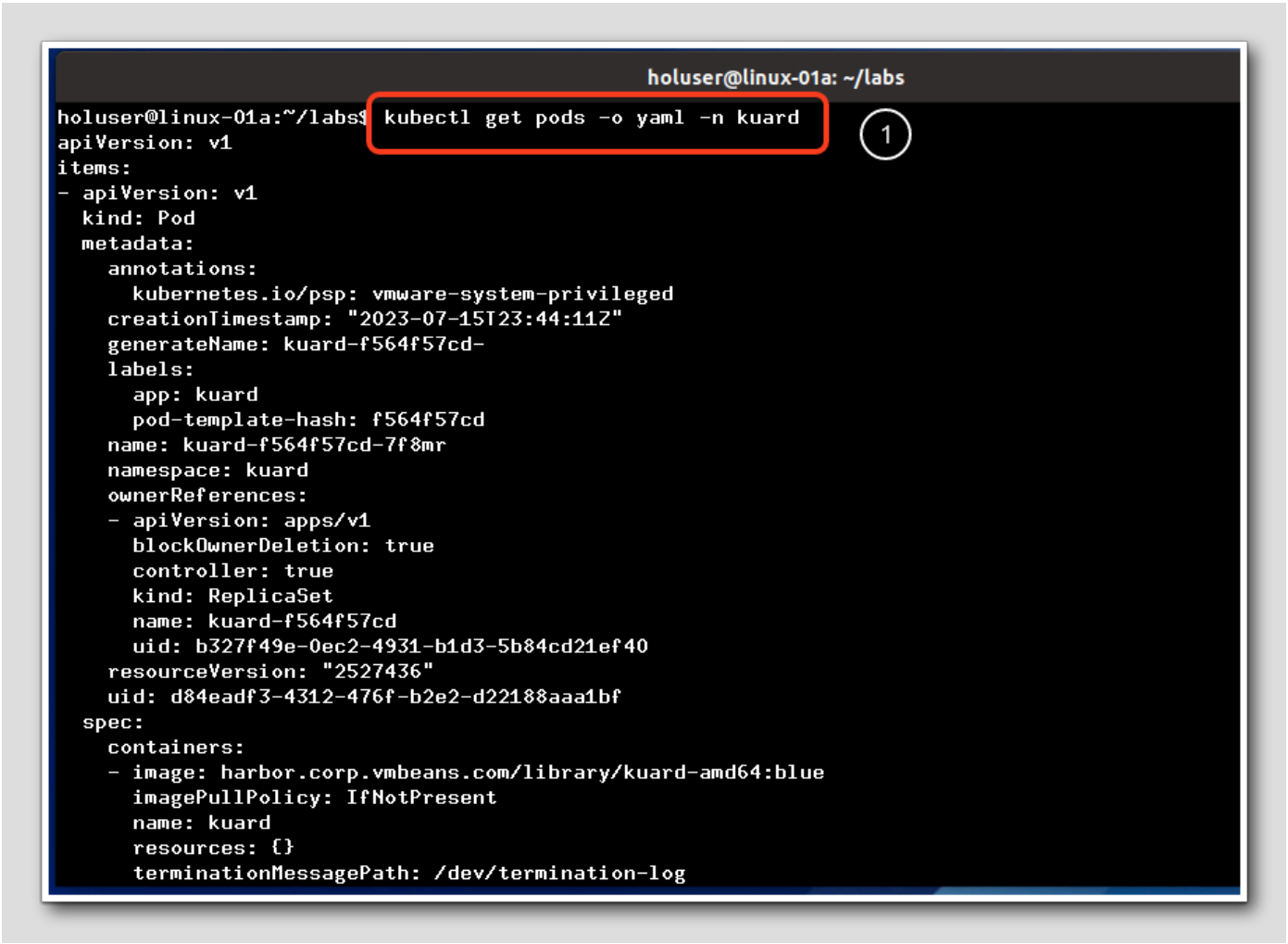
Digging Deeper

[137]



1. `kubectl get pods -o wide -n kuard`

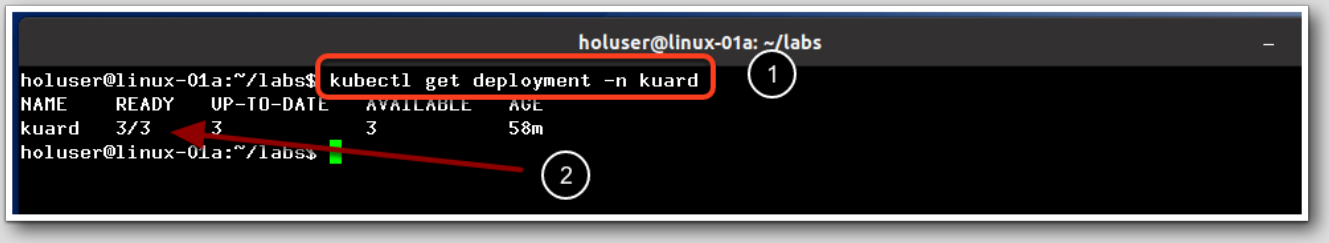
The -o option specifies the output type of the get pods command, the wide option returns additional information about the command.

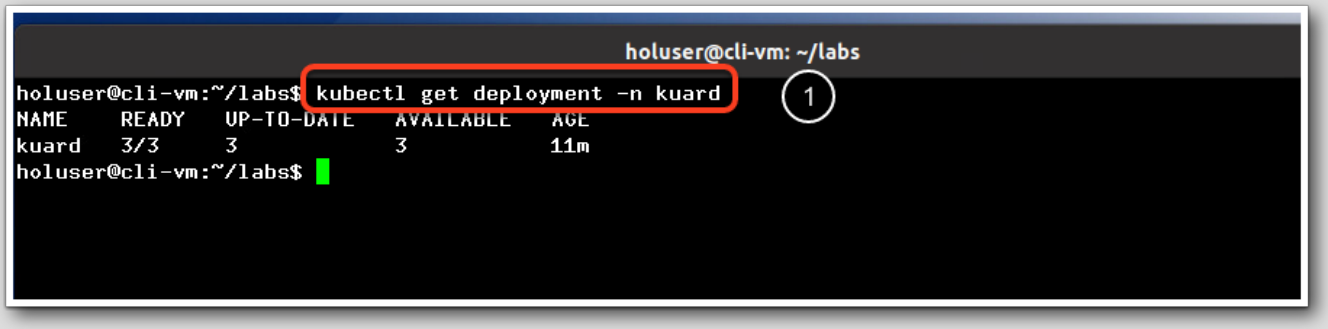


If we wanted to see the Yaml file that was used to create the pod we can use the -o yaml option.

1. `kubectl get pods -o yaml -n kuard`

This is often used as a handy way to generate a yaml file from an existing pod, by redirecting the output into a file I have a quick and easy way to generate a yaml file from a running pod.

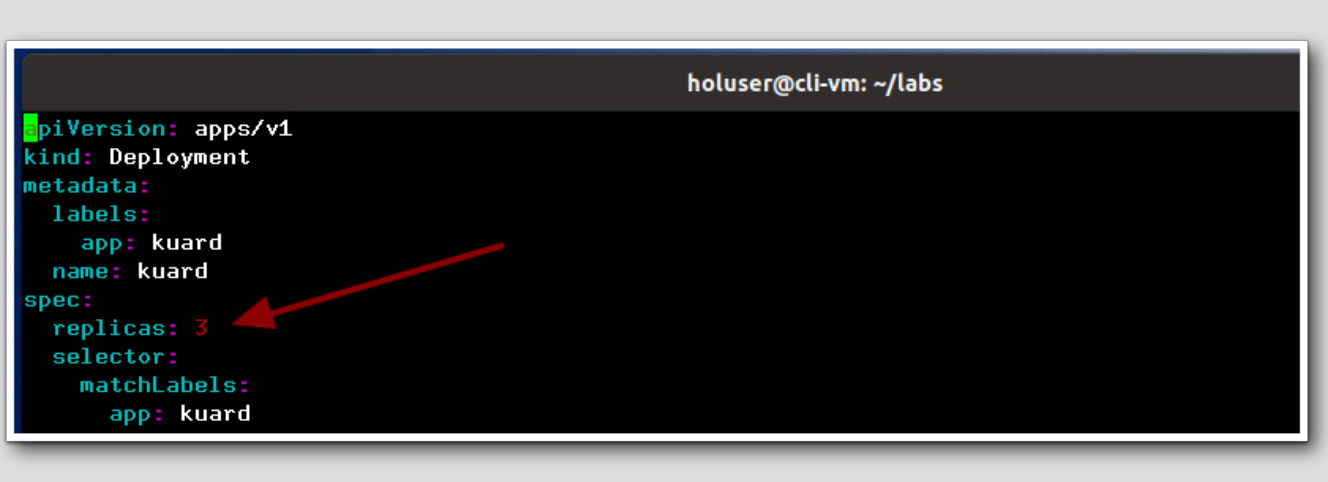




We can get additional information on our kubernetes deployment by typing :

1. `kubectl get deployment --namespace kuard`
2. Note we have 3 pods which matches the number of replicas we requested in the kuard.yaml file we examined earlier.

Once again we can add the "-o wide" option if we want to display additional information.



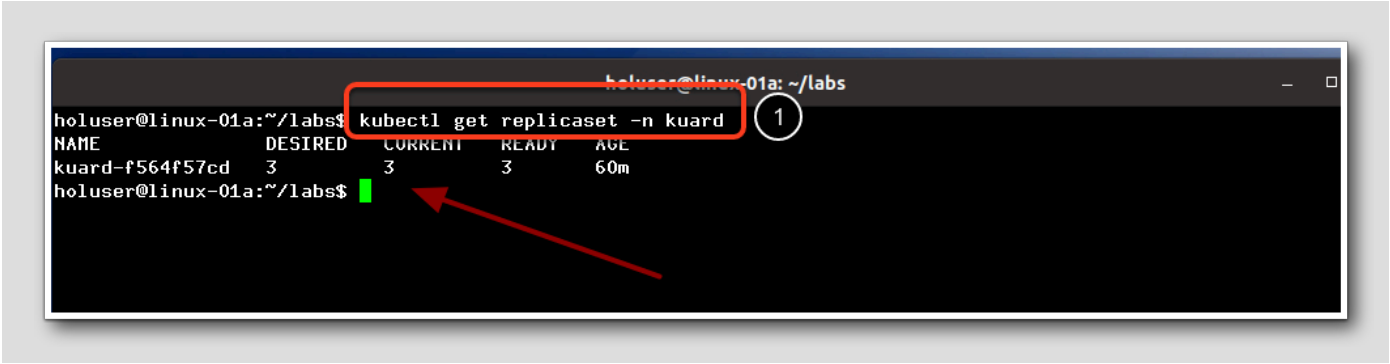
Scale The Kuard Deployment - Command Line

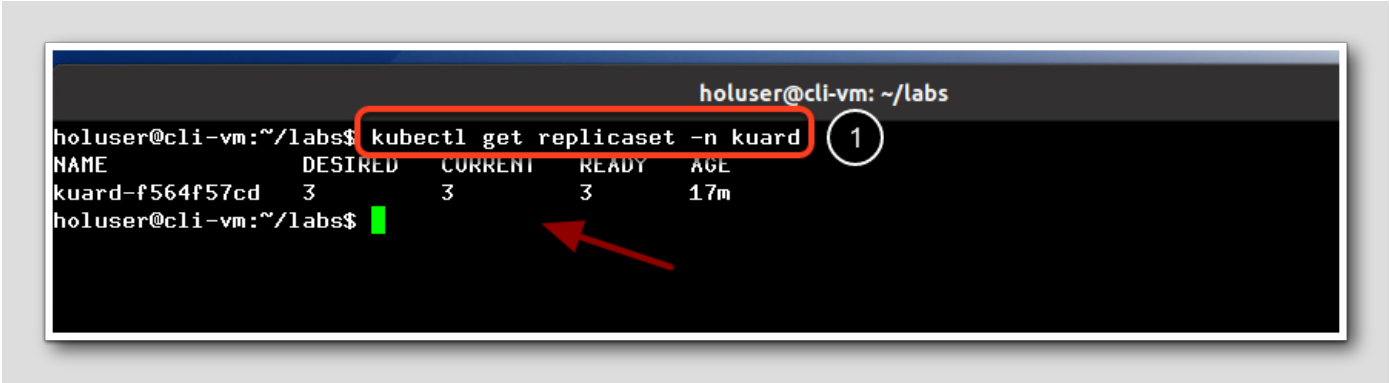
[138]

Sometimes we need to scale an application up or down for performance reasons. For example we may want to create a few additional instances of the application. Kubernetes gives us a few different ways to do this.

If you read the previous module you will recall that a replica set's purpose is to maintain a stable set of replica Pods running at any given time. As such, it is often used to guarantee the availability of a specified number of identical Pods.

Lets start by looking at the number of pods we currently have running.





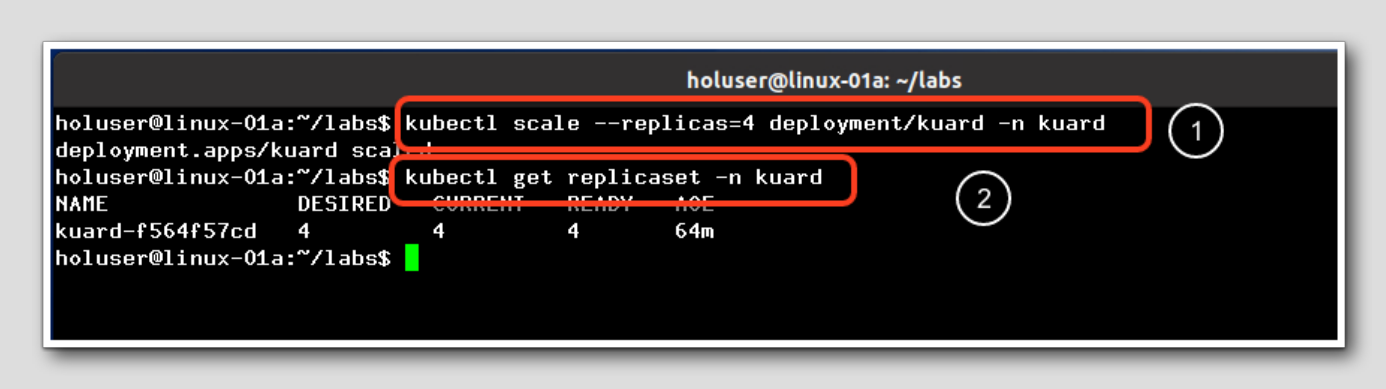
Type:

1. **kubectl get replicaset -n kuard**

We currently have three copies of the kuard pod running.

In Kubernetes, a ReplicaSet is a resource object used to manage and ensure the availability and scalability of a specific number of replicated Pods. A ReplicaSet defines the desired number of Pod replicas, and it continuously monitors the running Pods and makes adjustments to maintain the desired state. If a Pod fails or is terminated, the ReplicaSet replaces it with a new Pod to ensure the specified number of replicas is always available. Likewise if we ask kubernetes to increase the size of the replicaset it will autmatically add (or delete) the desired number of pods. Remember that you can both increase and decrease a replicaset.

Let's say we have encountered a performance problem with our application and need to add an additional pod, increase the replicaset to 4.

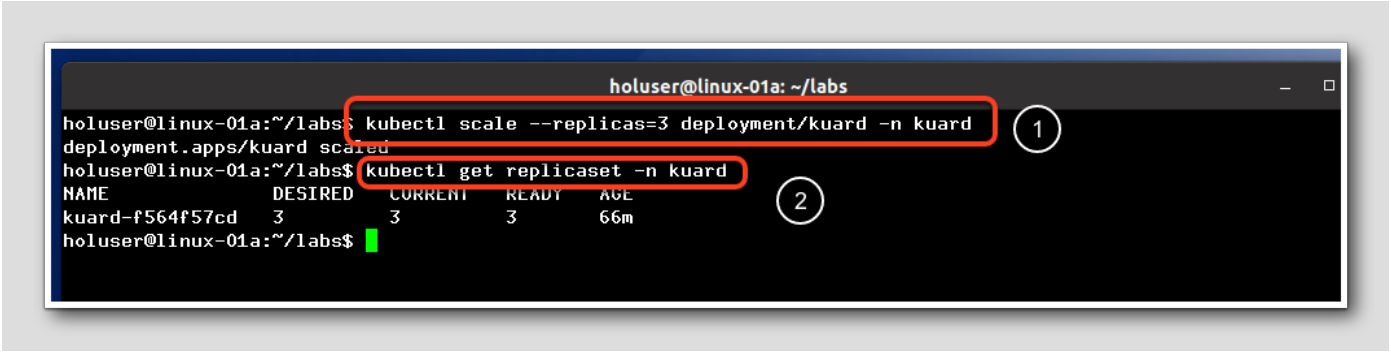


We could do this from the command line:

1. Type : **kubectl scale --replicas=4 deployment/kuard -n kuard**
2. Type: **kubectl get replicaset -n kuard**

Observe how quickly another instance of the kuard pod was deployed, one of the meny benefits of microservice based applications is the ability to rapidly scale a component of the application.

Scaling from the command line has a significant downside and would not represent a best practice since our original kuard Yaml file, kuard.yaml would still show this was deployed with 3 replicas. Also this command line increase only lasts for the duration of the replica set, if somebody deletes the namespace and recreates the kuard pod it would be deployed with 3 replicas since that is what the Yaml file specifies.



Let's scale it back down before proceeding:

1. `kubectl scale --replicas=3 deployment/kuard -n kuard`
2. `kubectl get replicaset -n kuard`

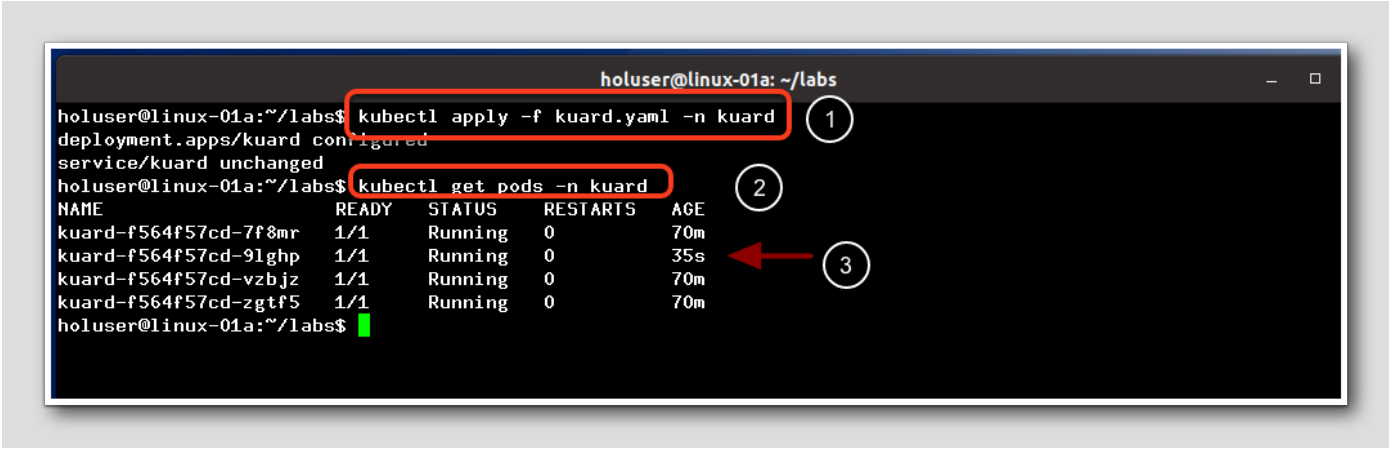
Scale The Kuard Deployment - The Yaml File

[139]

The preferred way to make changes to a kubernetes pod, unless the change is only temporary , would be to edit the Yaml file. This way the change is documented. If you are not confortable using a editor in a Linux environment feel free to skip executing this step.

1. Type *`vi kuard.yaml`* and press Enter to start vi editor
2. Use the arrow keys to move to the**"3"**in the**"replicas: 3"**text, and type **i** to enter Insert mode.
3. Press the "Delete" key and then type 4
4. Press the "Esc" key to exit Insert mode, then type **:wq** and press the Enter key. The ":" indicates command mode to vi, the "w" writes to the file, then the "q" quits vi editor. Note that this is "**:wq**"

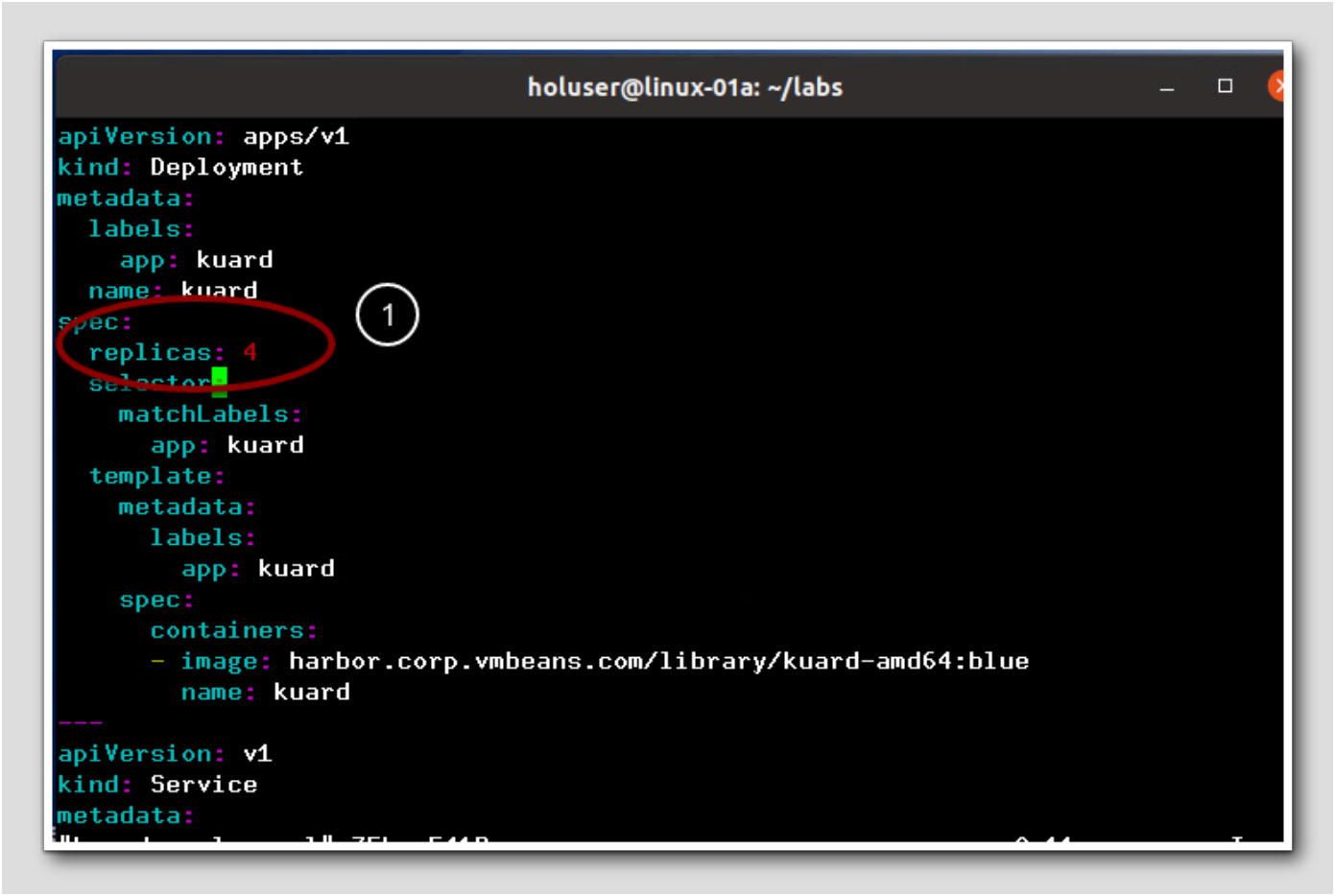
Now lets apply our changes to our deployment



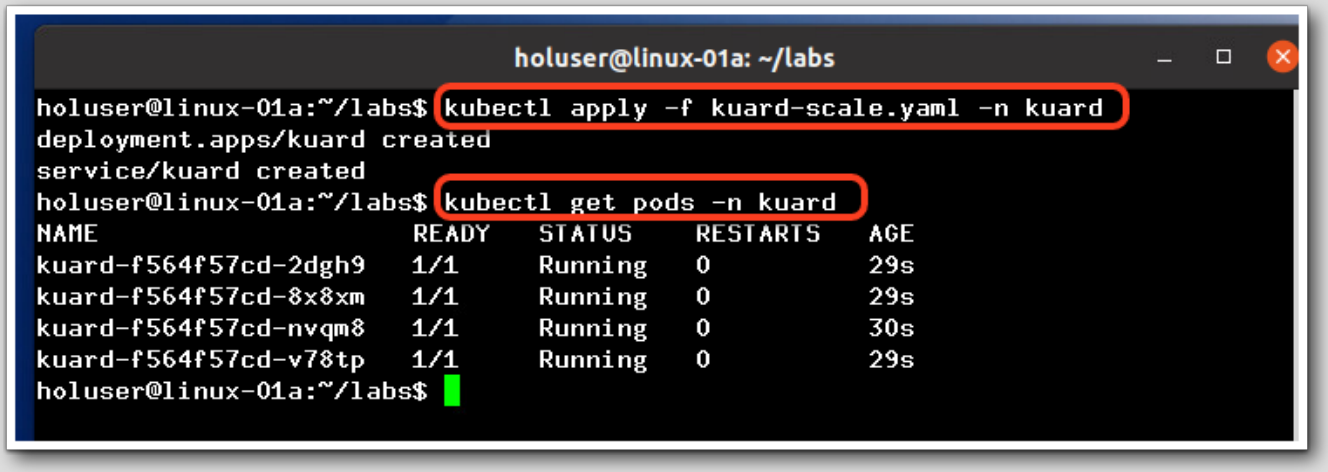
1. Type: **`kubectl apply -f kuard.yaml -n kuard`** and press Enter
2. Type: **`kubectl get pods -n kuard`** and press Enter.
3. Note that we now have 4 pods running. Also note the pod that was just created.

Scale The Kuard Deployment - The kuard-scale.yaml File [140]

If you are not comfortable with making changes using vi or some other editor we have made the changes for you in a file called kuard-scale.yaml.



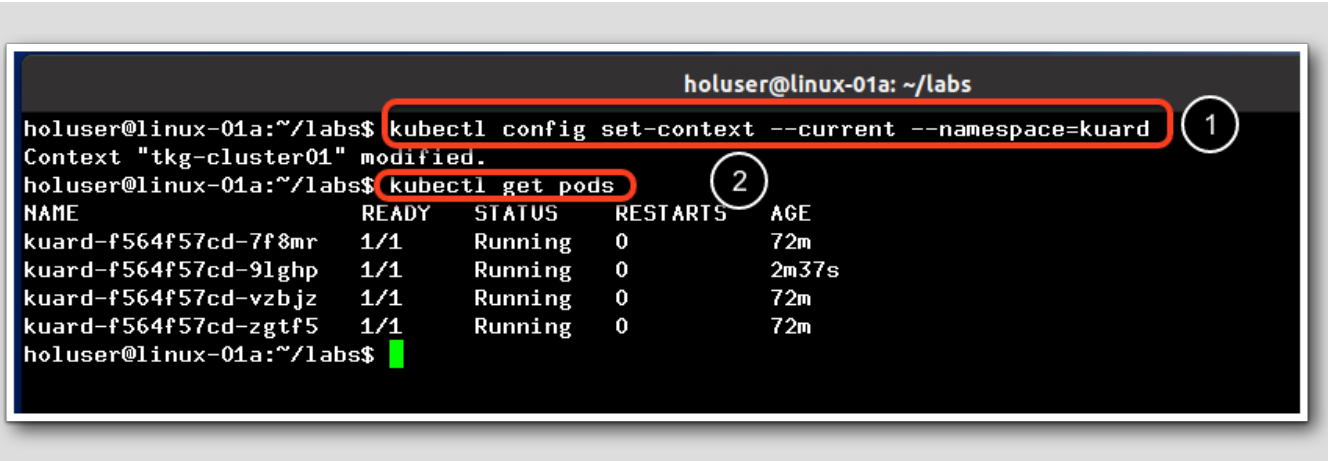
Note that we changed the number of replicas from 3 to 4.



1. Type: **`kubectl apply -f kuard-scale.yaml -n kuard`** and press Enter
2. Type: **`kubectl get pods -n kuard`** and press Enter.
3. Note that we now have 4 pods running.

Setting Context

[141]



Up to now we've been adding the namespace flag to all of our commands to tell the kubectl command that we want to work in the "kuard" namespace. We need do that because our default context is not set to "kuard" Alternatively we could just change our curent namespace.

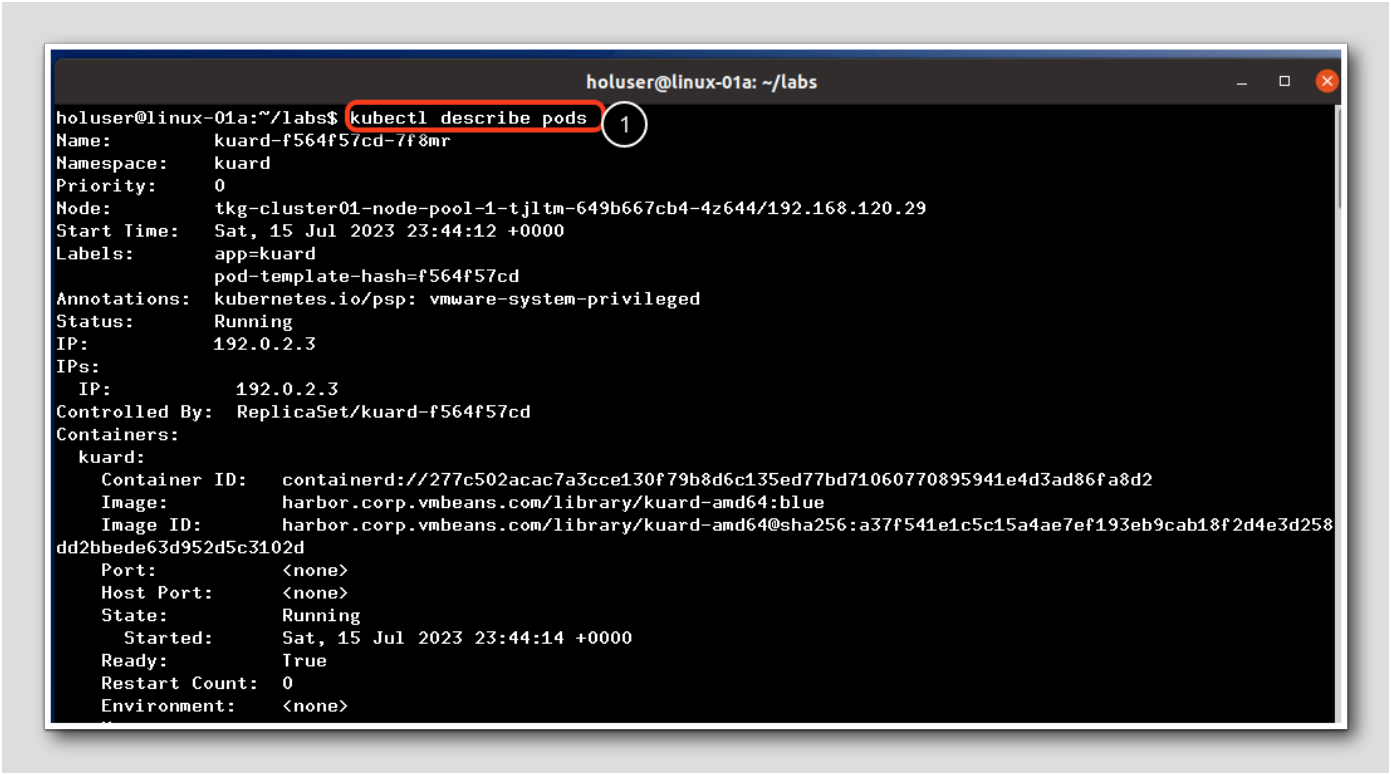
1. Type : **`kubectl config set-context --current --namespace=kuard`**
2. Now when we type: **`kubectl get pods`**

We don't need to specify the namespace.

Getting More Information on Our Pods

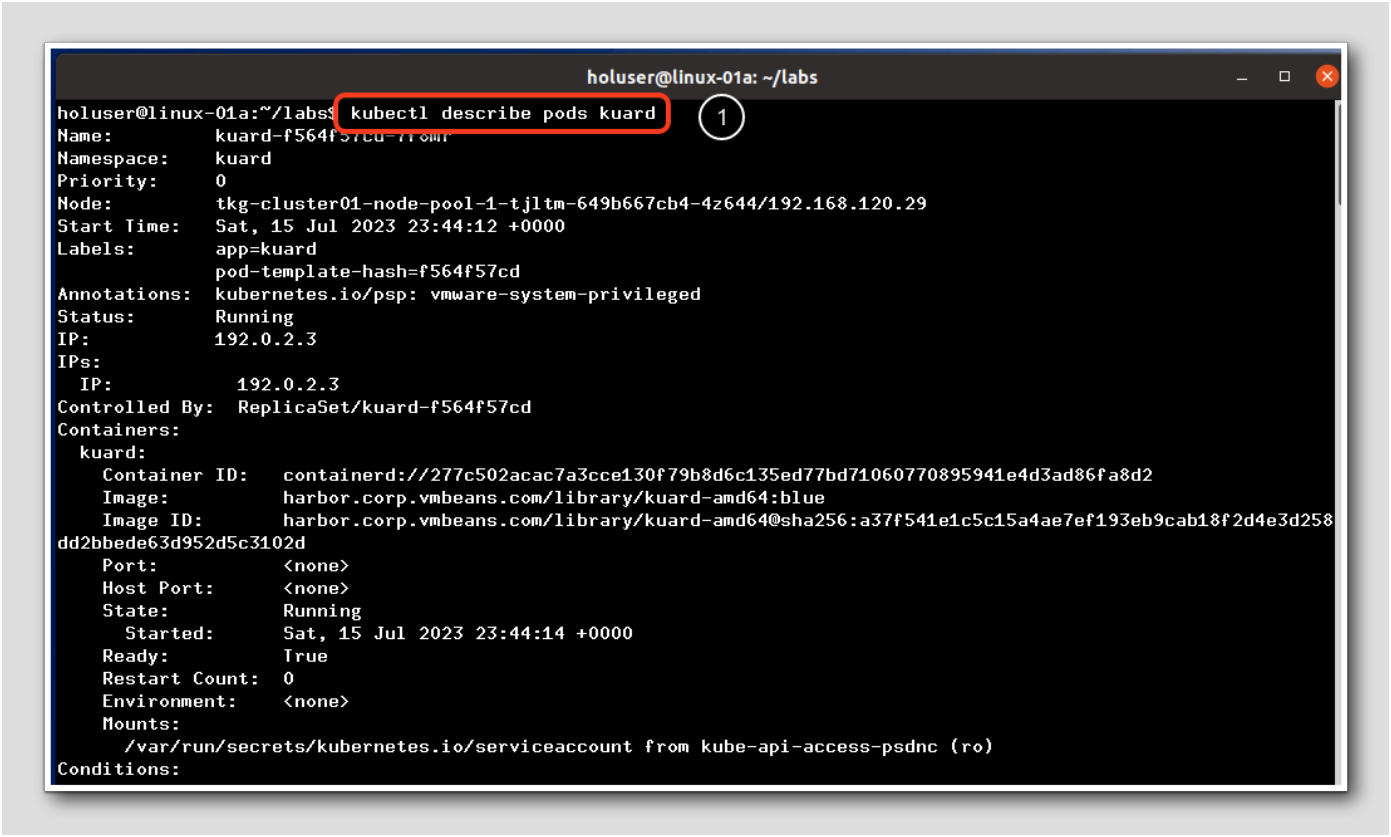
[142]

Let's look at a few simple commands that will provide us with some more information on our pods.



We can get detailed information on our pod(s) with the **`kubectl describe pods`** command. To do this we need to provide the name of a pod, we can type that in by hand or have the shell add it in for us. Lets start by typing:

1. **`kubectl describe pods`**



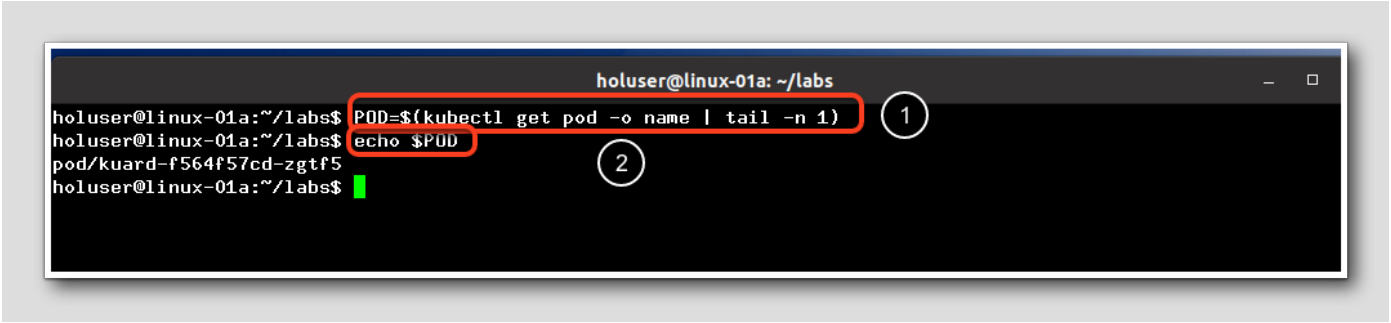
I can get detailed information on all the pods in the replica set by using the name of the replica set after the kubectl describe command.

1. Try: **`kubectl describe pods kuard`**

Feel free to take some time to examine the output from the kubectl describe pods command.

Note that the output from these two commands was the same, in this case there was only one deployment in the kuard namespace, the kuard application. However in most cases a namespace would have more then one application deployed and adding the name of the deployment will simplify the output.

Examining Kubernetes Pods



You can examine the logfile information on a pod with the **`kubectl logs`** command.

NOTE: Do not type the command above.

Let's start by setting a shell variable to reflect the name of one of our kuard pods.

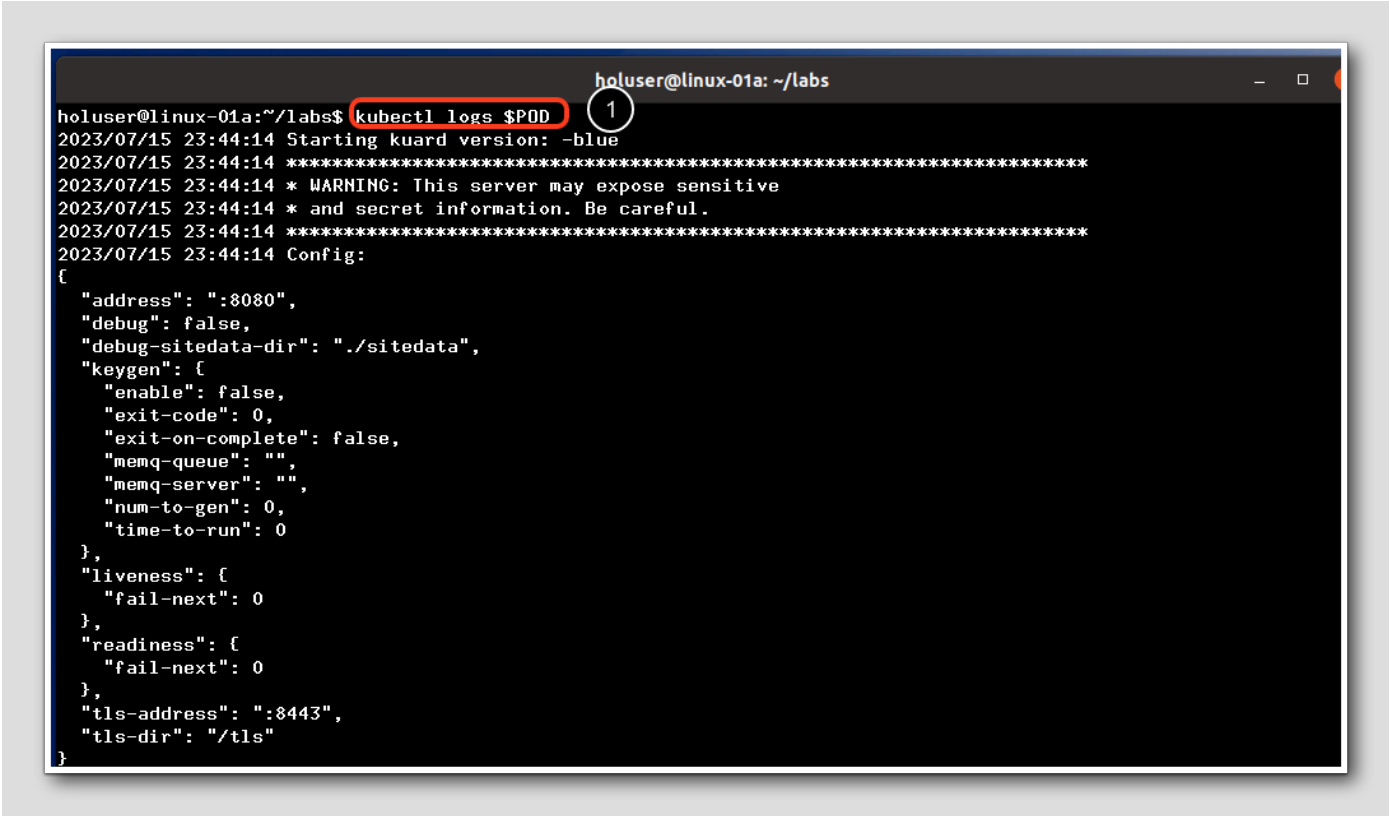
1. Type: **`POD=$(kubectl get pod -o name | tail -n 1)`**
2. Type: **`echo $POD`**

This sets a shell variable. POD to the output of the command string above.

The kubectl command: **`kubectl get pod -o name`** gets the name of the pod, stripping off the other information.

The **`tail -n 1`**

Reduces the output to a single line, recall we have 3 pods in the deployment. . Now we won't have to type in the lengthy pod names.



1. Type: **`kubectl logs $POD`**

And review the displayed log information.

I can view/edit the pod on the fly with:

`kubectl edit $POD`

The edit command allows you to directly edit any API resource you can retrieve via the command line tools. It will open the editor defined by your KUBE_EDITOR, or EDITOR environment variables, or fall back to 'vi' for Linux or 'notepad' for Windows. You can edit multiple objects, although changes are applied one at a time. The command accepts filenames as well as command line arguments, although the files you point to must be previously saved versions of resources.

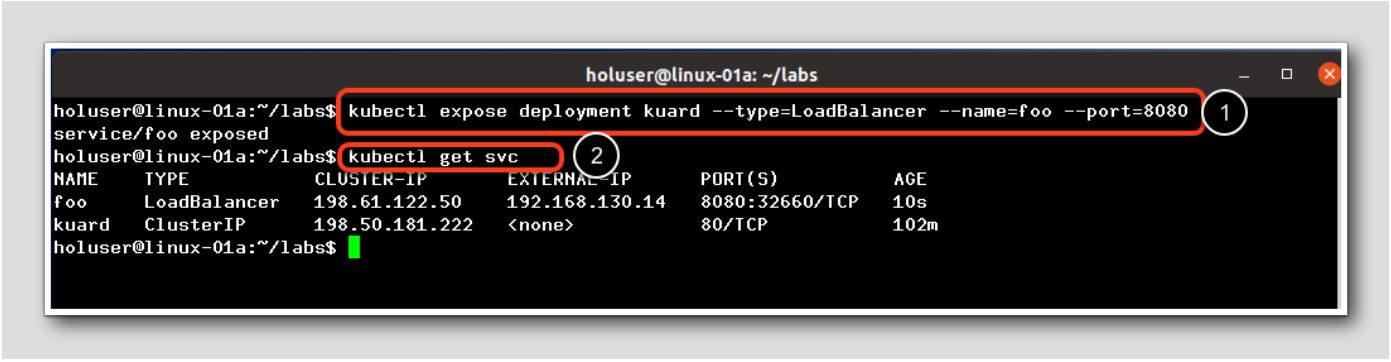
If you are comfortable with the VI editor feel free to examine the file. When done exit the editor by hitting ESC followed by q! (quit w/o write).

Access The Kuard Application

[144]

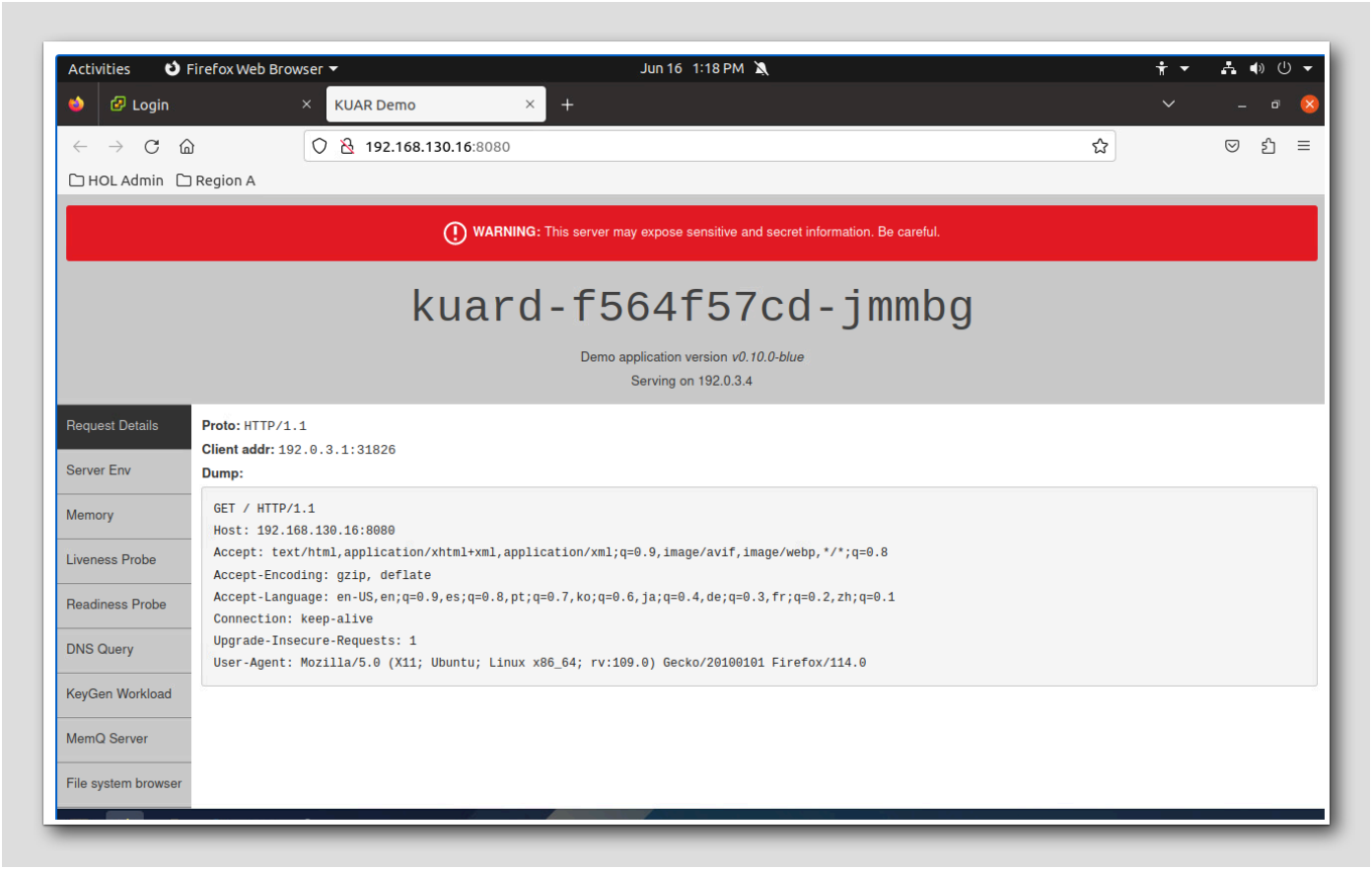
To access our application we will have to provide some way to expose the application so it is accessable from a web browser. A simple way to do this is to add a Load Balancer. A Load Balancer is a kubernetes service that will balance incoming web traffic between pods and also provides an external IP address. In this case we aren't balancing traffic but want to expose the kuard application so it can be accessed via a web browser.

If the command: kubectl expose deployment kuard --type=LoadBalancer --name=foo --port=8080 fails with a error indicating that the service already exists just skip this step in the lab.



1. Type `kubectl expose deployment kuard --type=LoadBalancer --name=foo --port=8080`
2. Type **`kubectl get svc`**
3. Note the EXTERNAL-IP (in the example 192.168.130.16) and PORTS (in the example 8080) provided . We will use these to connect to the kuard application UI.

Note: Your IP may be different then the example.



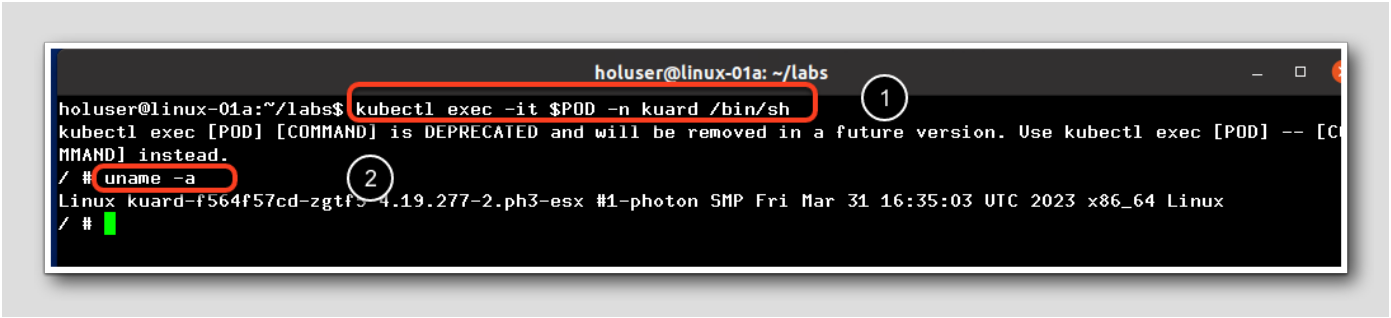
1. Click the Firefox icon to open a browser
2. In the address bar, type the IP address you noted from the kuard deployment, for example http://192.168.100.52:8080 ,and press Enter
3. Feel free to explore the"Kubernetes Up and Running" application by clicking on the tabs on the left-hand side

NOTE: It may take some time before the Kuard GUI appears feel free to move on if you do not want to wait.

Access The Kuard Application Another Way

[145]

Kubernetes also allows you to exec into a container this is useful for debugging or for when you want to interact with the container interactively. Let's try this with kuard.



As we did with previous commands that require a specific kubernetes pod you need to use a specific pod name on the command line. We set the variable POD to reflect the name of one of the pods in the kuard deployment previously.

Again ignore the message that the comand is deprecated.

1. Type: **`kubectl exec -it $POD -n kuard /bin/sh`** You will see a "$" prompt, you are now executing commands inside the kuard pod.
2. Type : **`uname -a`**

And you will see you are executing inside the kuard pod.

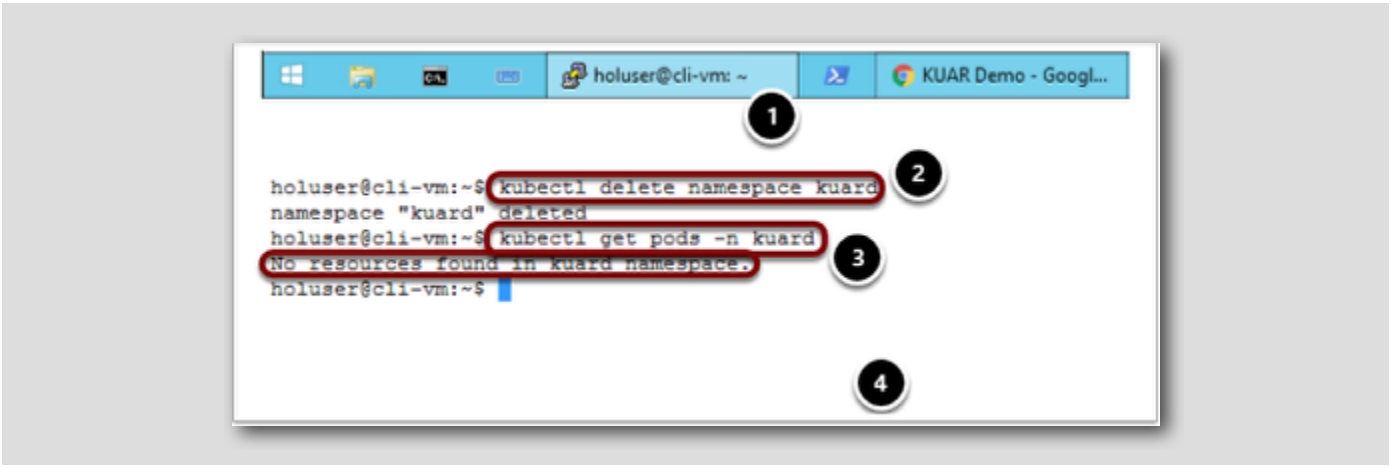
Feel free to explore with Linux commands if you are comfortable with Linux.

Type *exit* when you are done to return to kubernetes.

Cleanup

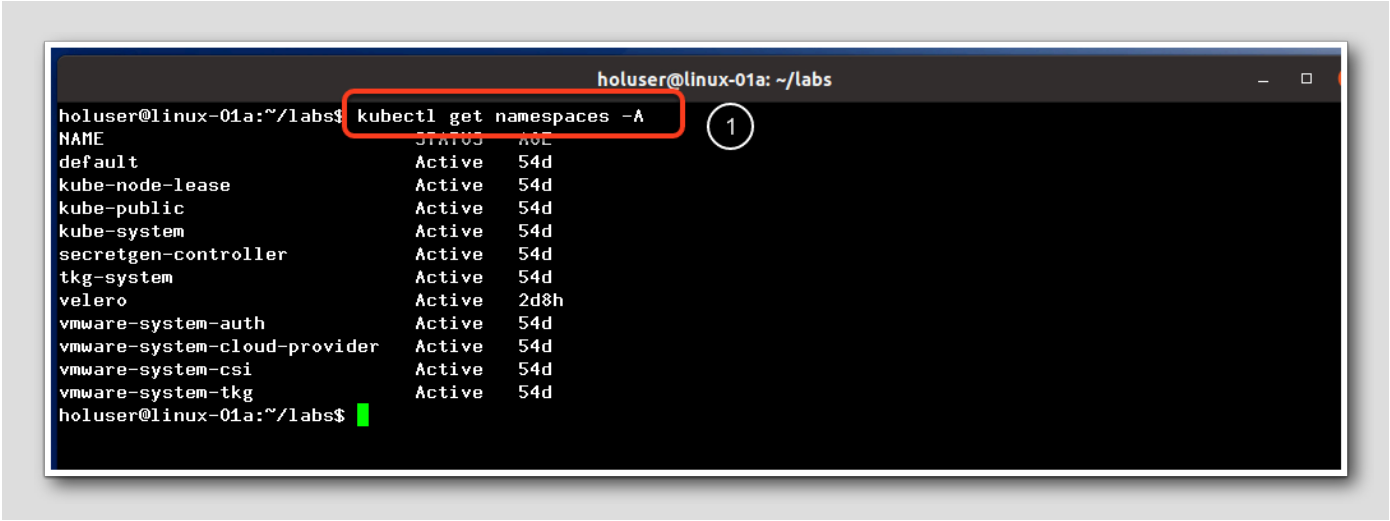
[146]

When you are done let's be good citizens and clean up our pod.



Deleting a namespace also deletes any resources in it. We will delete the namespace to remove the application

1. Return to the vm-cli PuTTY session by clicking the"holuser@cli-vm: ~"icon on the taskbar.
2. At the command line, type **`kubectl delete namespace kuard`** and press<Enter>
3. Type **`kubectl get pods -n kuard`** and press<Enter>
4. You should see "No resources found in kuard namespace."



To verify that the kuard namespace is gone type:

1. `kubectl get namespace -A`

The -A flag to the comand tells kubectl to show ALL namespaces. You will see that the kuard namespace is now gone.

Summary

[147]

In this module we walked through interactive examples of the following typical kubernetes commands

kubectl version

kubectl apply

kubectl get namespace

kubectl get pods

kubectl get deployments

kubectl get replicasets

kubectl create namespace

kubectl scale

kubectl config set-context

kubectl describe pods

kubectl logs

kubectl edit

kubectl expose

kubectl get svc (or get service)

kubectl exec

kubectl delete

kubectl describe deployment

kubectl rollout status

kubectl get-events

Conclusion [148]

Hopefully you found this exercise useful and helpful in learning more about kubernetes.

It can sound complex and complicated but adding Kubernetes to your infrastructure doesn't have to add complexity. With VMware Tanzu, you can ready your infrastructure for modern apps with consistent, conformant Kubernetes everywhere. Provide a self-service, compliant experience for developers that clears their path to production. Then centrally manage, govern and observe all clusters and apps across clouds. Its that simple.

You can learn more about Kubernetes and the VMware Tanzu suite of products at the following web sites, additionally module

https://www.vmware.com/topics/glossary/content/kubernetes

https://tanzu.vmware.com/

https://tanzu.vmware.com/developer/

https://apps-cloudmgmt.techzone.vmware.com/tanzu-techzone

https://communities.vmware.com/t5/Modern-Apps/ct-p/29000-home

Deploying a Microservices Based Application - Spring Petclinic [149]

Spring Petclinic Deployment Introduction [150]

In this lab we will build upon the steps from the previous tasks and do something more advanced and deploy a slightly more complicated micro services application. We be deploying the Spring Petclinic application to demonstrate several vSphere with Tanzu and kubernetes features. This application utilizes the VM Operator Service (VM Service) feature of Tanzu to deploy a Centos Mysql VM. We also use Tanzu Kubernetes Grid to deploy the application containers and services. Finally we leverage NSX Advanced Load Balancer and the Network Service to expose the application to users.

Deploy Centos Mysql VM using VM Service [151]

The VM Service leverages the same Kubernetes API to deploy and manage virtual machines that you use to deploy Kubernetes clusters, pods and services. This allows developers to manage virtual machines for application components that are not containerized. This capability allows you to use the same gitops approach to both virtual machines and containers and is one of the many advantages of deploying TKG clusters on vSphere with Tanzu.

Navigate to the **~/labs/petclinic** folder in the Linux-01 workstation. You'll find two folders. The first folder, **mysql-vmservice** contains the manifests for the mysql database VM Service. The second folder, **petclinic-k8s** contains the manifests for the Kubernetes deployments and microservice components for the petclinic application.

1. Change to the petclinic Application folder: `cd ~/labs/petclinic`
2. List the directories for the VM Service and micro-services: `ls`

```
holuser@linux-01a:~$ cd labs/petclinic/
holuser@linux-01a:~/labs/petclinic$ ls
mysql-vmservice  petclinic-k8s
holuser@linux-01a:~/labs/petclinic$
```

3. The first step is to deploy the mysql Database VM Service which will serve data for multiple micro-services in the Spring Petclinic application. Make sure your kubectl context is set to the Supervisor Namespace named **ecom01-stage**. **Type or copy/paste** the following command into Putty to set the correct context: `kubectx ecom01-stage`

```
holuser@linux-01a:~/labs/petclinic$ kubectx ecom01-stage
✔ Switched to context "ecom01-stage".
holuser@linux-01a:~/labs/petclinic$ kubectx
192.168.130.11
ecom01-stage
tkg-cluster01
```

4. **Type or copy/paste** the following command into Putty to deploy the mysql-centosvm and its associated Kubernetes Service:
`kubectl apply -f mysql-vmservice/mysql-vmsvc-vm-vds.yaml`

```
holuser@linux-01a:~/labs/petclinic$ k apply -f mysql-vmservice/mysql-vmsvc-vm-vds.yaml
virtualmachine.vmoperator.vmware.com/mysql-centosvm created
configmap/centos-cloudinit created
virtualmachineservice.vmoperator.vmware.com/mysql-vmservice created
```

5. To examine the Virtual Machine you just deployed, and ensure the health of the mysql vmservice is running, **Type or copy/paste** the following command into Putty : `kubectl get vm`

```
holuser@linux-01a:~/labs/petclinic$ kubectl get vm
NAME                                                POWER-STATE   AGE
mysql-centosvm                                      poweredOn     39m
tkg-cluster01-n55qh-4nzc8                           poweredOn     2d8h
tkg-cluster01-n55qh-jfszv                           poweredOn     2d8h
tkg-cluster01-n55qh-ndhqg                           poweredOn     2d8h
tkg-cluster01-node-pool-1-tjltm-649b667cb4-qc4df    poweredOn     2d8h
tkg-cluster01-node-pool-2-s5fbn-57db89b555-6gs97    poweredOn     2d8h
tkg-cluster01-node-pool-3-s2xbz-7d684f6f84-zllhv    poweredOn     2d8h
```

6. To get more detailed status of the mysql-centosvm vmservice Virtual Machine, Type or copy/paste the following command into Putty : `kubectl get vm mysql-centosvm -o yaml | grep conditions: -A 12`

```
holuser@linux-01a:~/labs/petclinic$ k get vm mysql-centosvm -o yaml | grep conditions: -A 12
  conditions:
  - lastTransitionTime: "2023-05-24T22:32:35Z"
    status: "True"
    type: Ready
  - lastTransitionTime: "2023-05-24T22:31:46Z"
    status: "True"
    type: GuestCustomization
  - lastTransitionTime: "2023-05-24T22:30:15Z"
    status: "True"
    type: VirtualMachinePrereqReady
  - lastTransitionTime: "2023-05-24T22:31:56Z"
    status: "True"
    type: VirtualMachineTools
```

7. To get the status of the Kubernetes Service that was created on ALB for our mysql VM service VM, Type or copy/paste the following command into Putty: `kubectl get svc mysql-vmservice`

```
holuser@linux-01a:~/labs/petclinic$ kubectl get svc mysql-vmservice
NAME              TYPE           CLUSTER-IP    EXTERNAL-IP       PORT(S)                        AGE
mysql-vmservice   LoadBalancer   10.96.0.14    192.168.130.100   22:32387/TCP,3306:30561/TCP    41m
```

NOTE: We can see that an External-IP has been assigned by NSX ALB and that we are exposing SSH and the mysql port 3306 on the Service.

Deploy the Spring Petclinic Kubernetes objects

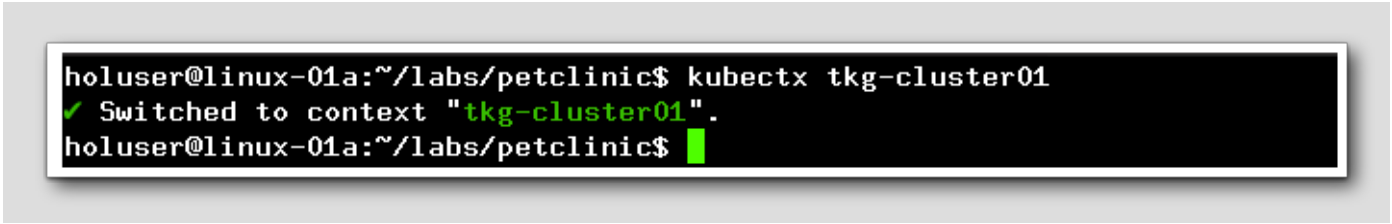
[152]

Now that we have a mysql database running as a Virtual Machine and managed by the vmoperator in Tanzu and a Kubernetes Service managed by ALB exposing the mysql port we are ready to spin up the Petclinic microservices on our Kubernetes cluster **tkg-cluster01** which is also running in the **ecom01-stage** Supervisor Namespace.

1. Make sure you are in the **petclinic** folder: `cd ~/labs/petclinic`

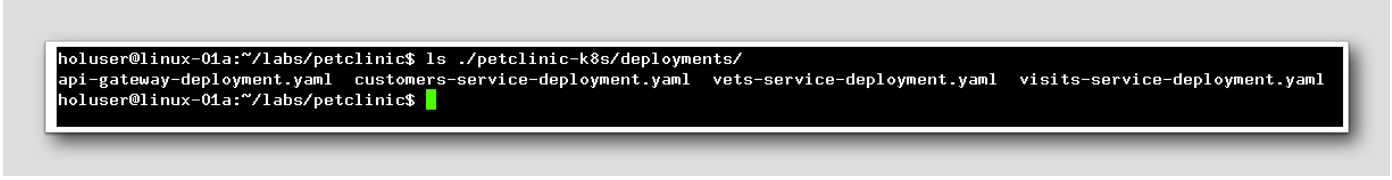
2. **Type or copy/paste** the following command into Putty to change your kubectl context to the **tkg-cluster01** Kubernetes cluster:

   `kubectx tkg-cluster01`



3. Examine the contents of the **petclinic-k8s services** folder by **Typing or copy/pasting** the following command into Putty: `ls ./petclinic-k8s/services`

4. **Type or copy/paste** the following command into Putty to see the contents of the petclinic-k8s deployments folder: `ls ./petclinic-k8s/deployments`



5. We are now ready to create a Kubernetes namespace for our spring-petclinic Kubernetes objects in the tkg-cluster01 Kubernetes cluster. Type or copy/paste the following command into Putty: `kubectl apply -f ./petclinic-k8s/01-namespace.yaml`
6. Type or copy/paste the following command into Putty to see the Namespace you created: `kubectl get namespace`

```
holuser@linux-01a:~/labs/petclinic$ kubectl apply -f ./petclinic-k8s/01-namespace.yaml
namespace/spring-petclinic created
holuser@linux-01a:~/labs/petclinic$ kubectl get namespace
NAME                              STATUS   AGE
default                           Active   14d
kube-node-lease                   Active   14d
kube-public                       Active   14d
kube-system                       Active   14d
secretgen-controller              Active   14d
spring-petclinic                  Active   13s
tkg-system                        Active   14d
vmware-system-auth                Active   14d
vmware-system-cloud-provider      Active   14d
vmware-system-csi                 Active   14d
vmware-system-tkg                 Active   14d
```

7. Type or copy/paste the following command into Putty to create the required Kubernetes objects for the Petclinic application including Secrets, Configmaps, roles, and Kubernetes services for each of the microservice components: `kubectl apply -f ./petclinic-k8s/services/`
8. Examine the Kubernetes services you created for internal application communication and for external access to the web front end. Type or copy/paste the following command into Putty: `kubectl get svc -n spring-petclinic`

```
holuser@linux-01a:~/labs/petclinic$ kubectl apply -f ./petclinic-k8s/services/
secret/mysql-db-info created
configmap/petclinic-config created
role.rbac.authorization.k8s.io/namespace-reader created
rolebinding.rbac.authorization.k8s.io/namespace-reader-binding created
service/api-gateway created
service/customers-service created
service/vets-service created
service/visits-service created
holuser@linux-01a:~/labs/petclinic$ kubectl get svc -n spring-petclinic
NAME                TYPE           CLUSTER-IP      EXTERNAL-IP      PORT(S)          AGE
api-gateway         LoadBalancer   198.63.12.63    192.168.130.15   80:30756/TCP     21s
customers-service   ClusterIP      198.57.148.99   <none>           8080/TCP         21s
vets-service        ClusterIP      198.57.43.231   <none>           8080/TCP         21s
visits-service      ClusterIP      198.52.237.34   <none>           8080/TCP         20s
holuser@linux-01a:~/labs/petclinic$
```

9. Type or copy/paste the following command into Putty to see the Service endpoints for the Petclinic application: `kubectl get ep -n spring-petclinic`

Notice how the ENDPOINTS are blank for the 4 services. This is because we have not deployed the actual Kubernetes Deployments and associated Pods for our application.

```
holuser@linux-01a:~/labs/petclinic$ kubectl get ep -n spring-petclinic
NAME                ENDPOINTS   AGE
api-gateway         <none>      75s
customers-service   <none>      75s
vets-service        <none>      75s
visits-service      <none>      74s
holuser@linux-01a:~/labs/petclinic$
```

10. We can now deploy the actual Kubernetes Deployments and Pods for the Petclinic Application. Type or copy/paste the following command into Putty: `kubectl apply -f ./petclinic/deployments/`
11. Type or copy/paste the following command into Putty to see the current status of the 4 microservices for the Petclinic application: `kubectl get deploy,po -n spring-petclinic`

```
holuser@linux-01a:~/labs/petclinic$ kubectl get deploy,po -n spring-petclinic
NAME                                READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/api-gateway         1/1     1            1           54s
deployment.apps/customers-service   1/1     1            1           54s
deployment.apps/vets-service        1/1     1            1           54s
deployment.apps/visits-service      1/1     1            1           54s

NAME                                    READY   STATUS    RESTARTS   AGE
pod/api-gateway-575599b9d9-tpv4w        1/1     Running   0          20s
pod/customers-service-64575d6985-wm8xw  1/1     Running   0          20s
pod/vets-service-6467fd69df-xkwkj       1/1     Running   0          20s
pod/visits-service-846d78d6c7-9s7f1     1/1     Running   0          20s
holuser@linux-01a:~/labs/petclinic$
```

We will need to wait until the 4 PODs are in a Ready State, which indicated the initialization scripts for the database have run and the Containers are responding on the correct Port(8080).

TIP: You can Type or copy/paste the following command into Putty to see the realtime status of our PODs: `kubectl get po -n spring-petclinic -w`

When all 4 Pods indicate 1/1 for Readiness you can Type the following to stop the watch: `CTL + C`

Test the Spring Petclinic Application

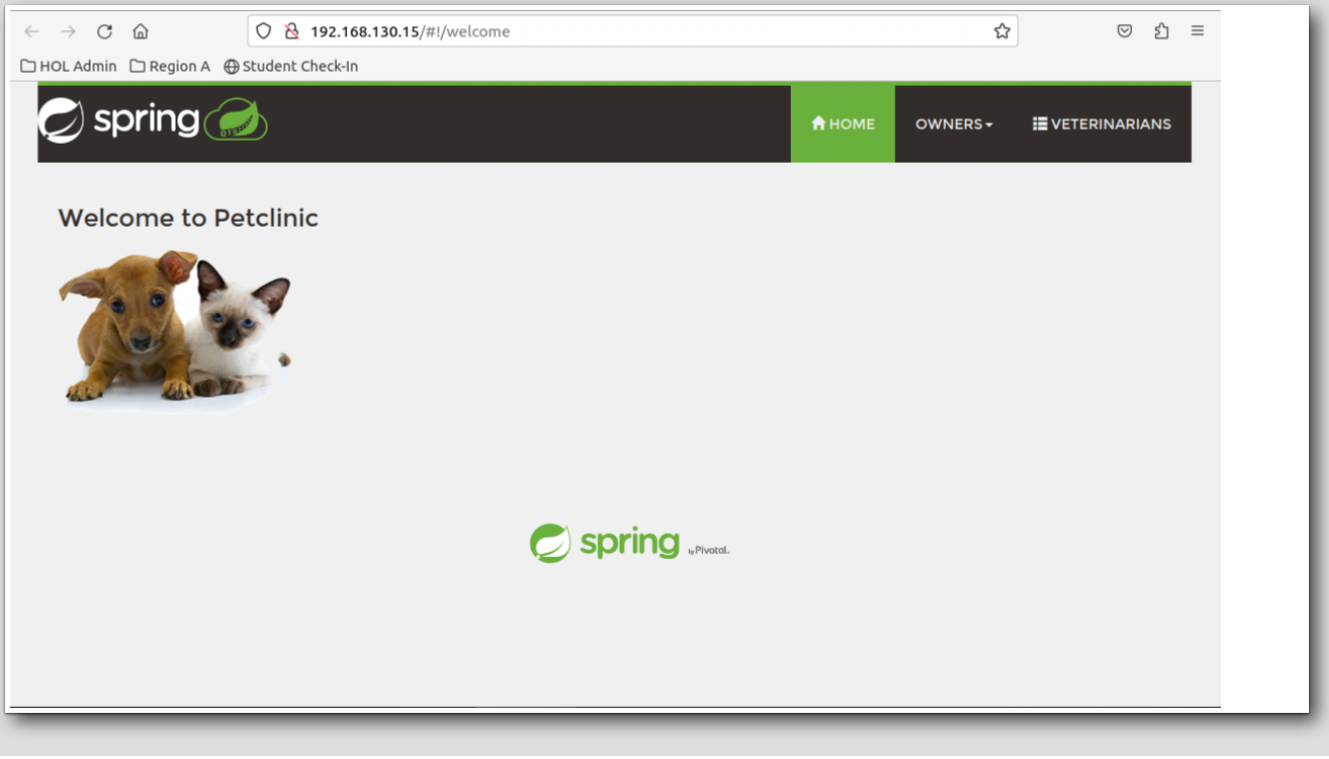
[153]

1. Type or copy/paste the following command into Putty to obtain the EXTERNAL-IP of the Spring Petclinic Frontend Gateway and copy this address: `kubectl get svc -n spring-petclinic`

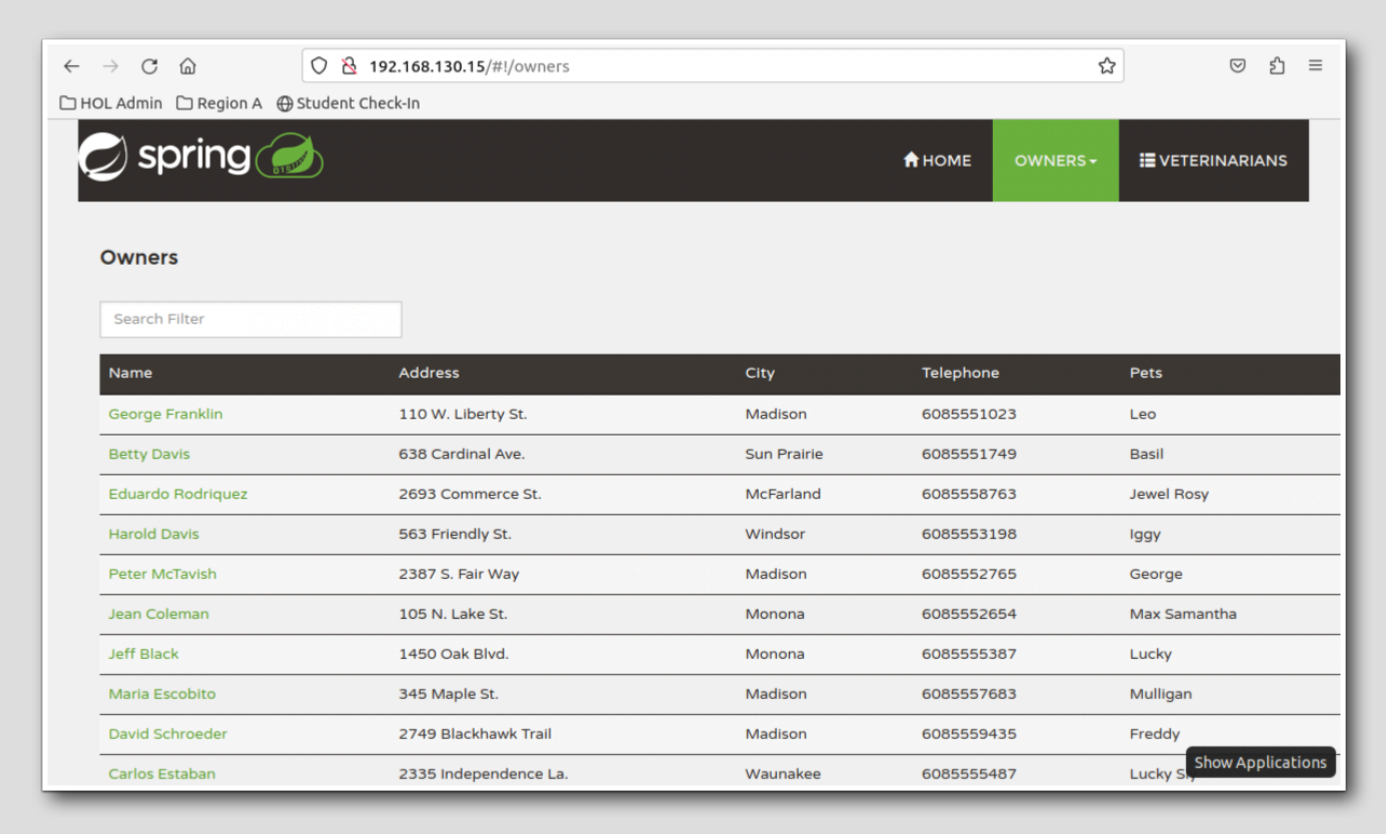
2. Back on the Desktop, start a browser session to access the application:

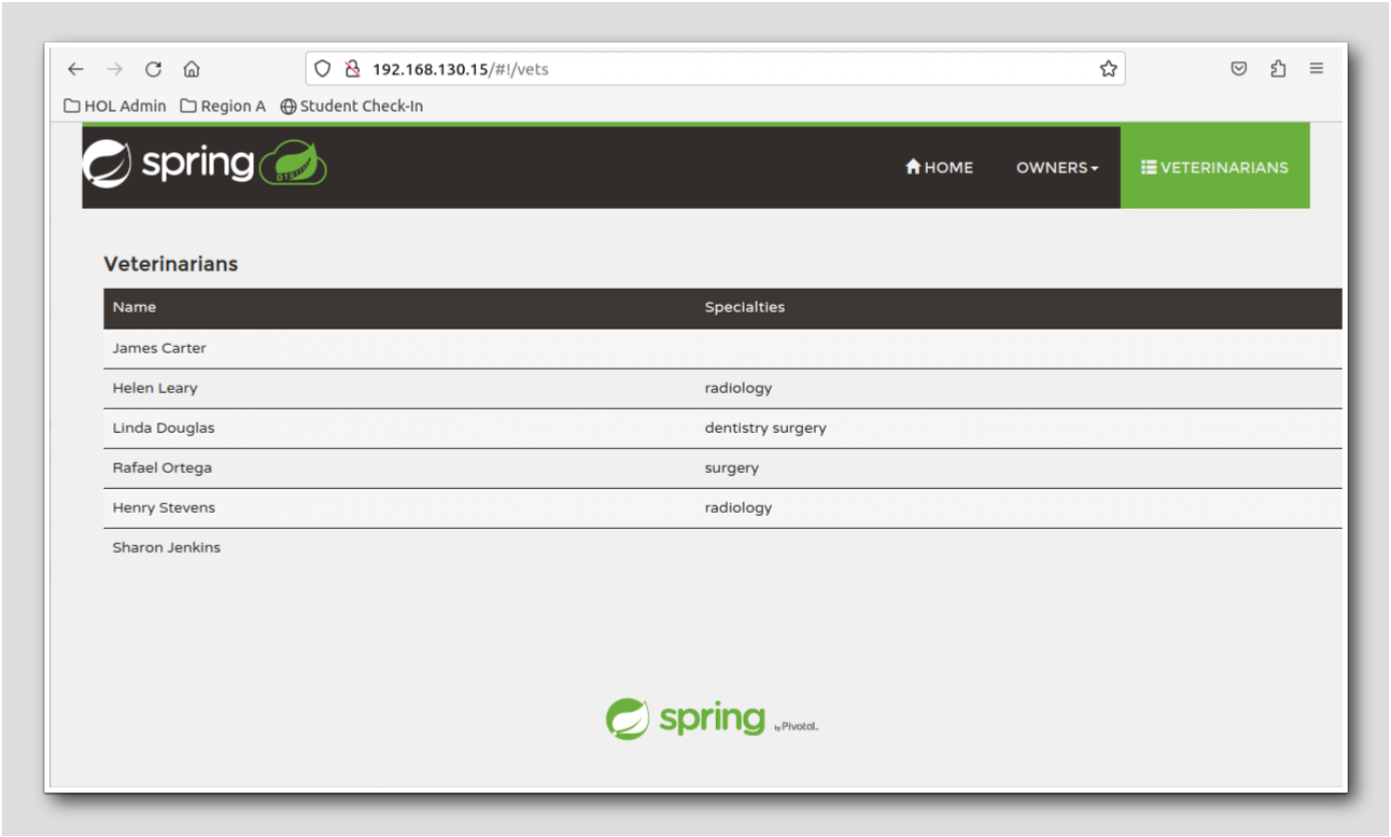
```
holuser@linux-01a:~/labs/petclinic$ kubectl get svc -n spring-petclinic
NAME                TYPE           CLUSTER-IP       EXTERNAL-IP      PORT(S)          AGE
api-gateway         LoadBalancer   198.57.197.214   192.168.130.15   80:30012/TCP     6m18s
customers-service   ClusterIP      198.51.195.109   <none>           8080/TCP         6m17s
vets-service        ClusterIP      198.58.88.114    <none>           8080/TCP         6m17s
visits-service      ClusterIP      198.58.243.118   <none>           8080/TCP         6m17s
```

3. Open a new Tab in Firefox and enter the EXTERNAL-IP from the previous command: `http://192.168.130.15`



4. Explore different sections of the application which are handled by the individual microservices we applied to our Kubernetes cluster and backed by mysql tables in our mysql-centosvm Virtual Machine that we deployed in the previous steps.





Conclusion

[154]

Congratulations on completing Module 3. You should now have a solid foundational understanding of container orchestration and Kubernetes.

VMware provides a host of Modern Apps FREE learning resources hosted on our *Tanzu website* as well as the *Modern Apps Community*.

Please use the QR code for more information.

Congratulations you have finished Module 3

[155]

Based on your interests, please proceed to any module below:

- Module 1 - Introduction to Containers (15 minutes) (Basic) - In this module, we will explain containers and how they enable 3rd Platform application architectures to be run efficiently in distributed environments. We will also delve into the basics of Docker as a container platform. This is a reading module only without any interactive labs.
- Module 2 - A Quick Tour of Docker - (15 minutes) (Basic) In this module, we're getting hands on with docker and explore how to create simple containers and how to leverage docker container networking. Finally, we'll look at how you can build your own images using a Dockerfile.
- Module 4 - Introduction to Harbor - (30 minutes) (Advanced) In this module we will walk through some of the capabilities of the Harbor secure image repository and explore some worked examples using images pre-populated into this labs Harbor instance
- Module 5 - Introduction to Tech Zone VMware Learning Platform & Kube Academy - (15 minutes) (Basic) In this module we will introduce some of the many FREE learning resources which VMware provides for users looking to start their journey into modern apps and kubernetes or looking to build on the knowledge and experience they already have.

From here you can:

1. Click to advance to the next page and continue with the next lab module
2. Open the TABLE OF CONTENTS to jump to any module or lesson in this lab manual
3. End your lab and come back and start it again in the future

Module 4 - Introduction to Harbor (15 minutes) Basic

Introduction to Harbor

[157]

This module contains the following lesson:

- Walk through the Harbor image repository and explore it's key features.

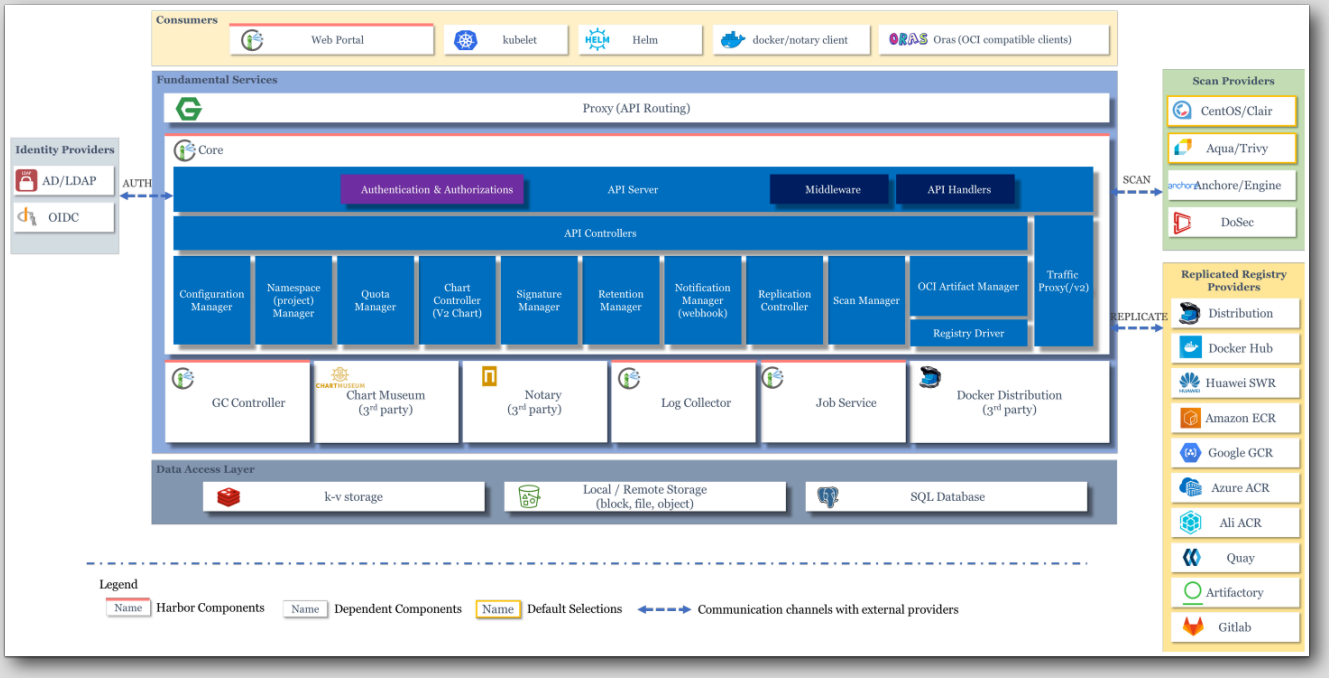
Harbor is an open source registry that secures artifacts with policies and role-based access control, ensures images are scanned and free from vulnerabilities, and signs images as trusted. Harbor, a CNCF Graduated project, delivers compliance, performance, and interoperability to help you consistently and securely manage artifacts across cloud native compute platforms like Kubernetes and Docker. Harbor was the first OCI compliant registry to gradaute the CNCF and has 18k stars on GitHub.

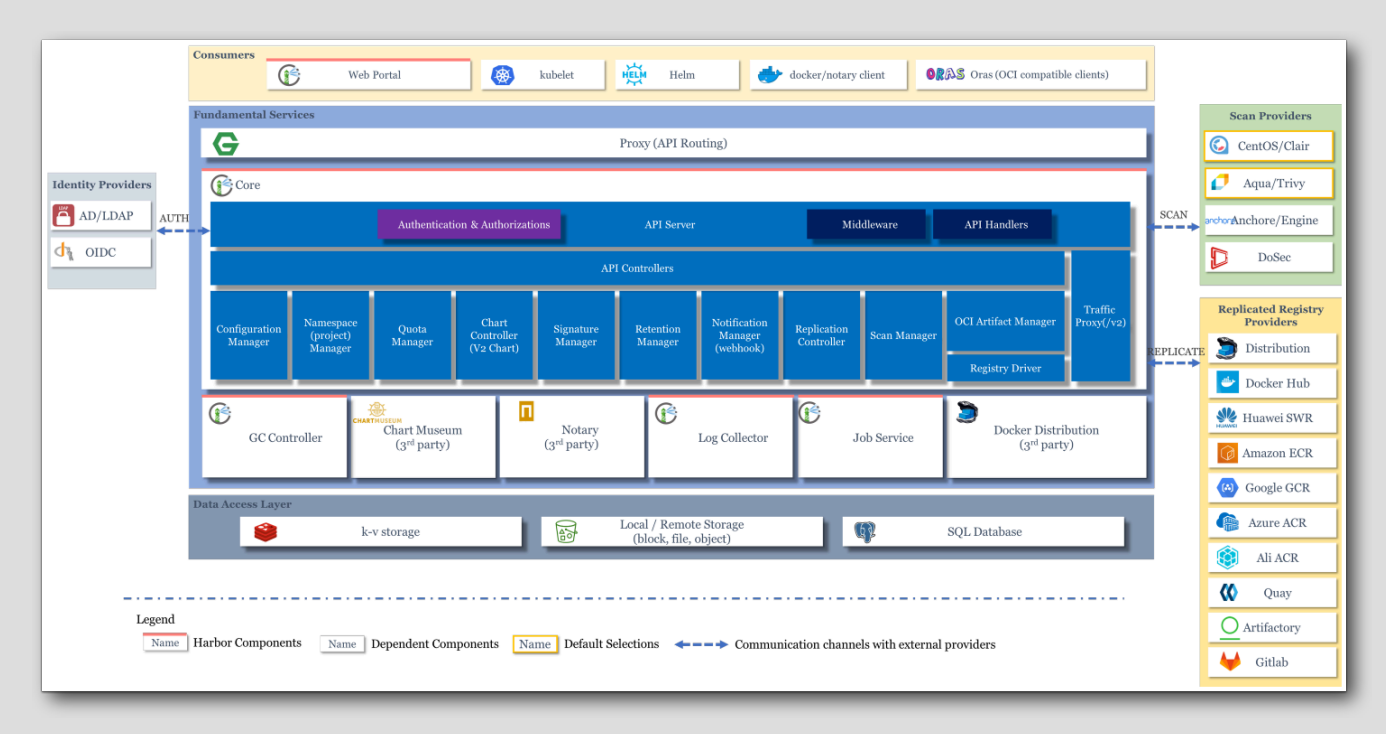
Harbor is included with Tanzu for Kubernetes Operations and provides an enterprise-class registry server that stores and distributes container images. Harbor allows you to store and manage images and provides the functionalities usually required by an enterprise, such as security, identity, and management. As an enterprise private registry, Harbor offers enhanced performance and security.

Harbor includes the following key features:

- Replicate projects: Harbor supports image replication to replicate repositories from one Harbor instance to another.
- Manage role by LDAP group: Harbor administrators can import an LDAP/AD group to Harbor and assign project roles to it.
- Manage Labels: Harbor provides labels to isolate image resources globally or at the project level.
- Manage Helm Charts: Harbor provides management of Helm charts isolated by projects and controlled by RBAC.
- Integrated UAA Authentication: Harbor can share UAA authentication with VMware Tanzu Application Service for VMs (TAS for VMs) and TKGI.
- Role-Based Access Control: Users and repositories are organized into projects. Users can have different permissions for the images in different projects.
- Policy-Based Image Replication: Images can be synchronized between multiple registry instances with auto-retry on errors, offering support for load balancing, high availability, multi-datacenter, hybrid, and multi-cloud scenarios.
- Vulnerability Scanning: Harbor uses Clair or Trivy to scan images regularly and warn users of vulnerabilities.
- LDAP/Active Directory (AD) Support: Harbor integrates with enterprise LDAP/AD systems for user authentication and management.
- Image Deletion and Garbage Collection: Images can be deleted and their space can be recycled.
- Notary: Image authenticity can be ensured by using Docker Notary.

- **Graphical User Portal**: Users can easily browse, search repositories, and manage projects.
- **Auditing**: All the operations to the repositories are tracked.
- **RESTful API**: RESTful APIs for most administrative operations, easy to integrate with external systems.
- **Easy deployment:** Harbor can be deployed via Docker compose, Helm Chart, Harbor Kubernetes Operator, or installed onto a VM





Exploring the Capabilities Of The Harbor Enterprise Container Registry

[158]

The application deployments in this lab make use of a private container registry. We are using a graduated Cloud Native Computing Foundation open source project , Harbor, as our registry. Harbor was donated to the CNCF by VMware and is included as an enterprise supported product with VMware Tanzu solution. Harbor is also a stand alone open source project stewarded by the CNCF.

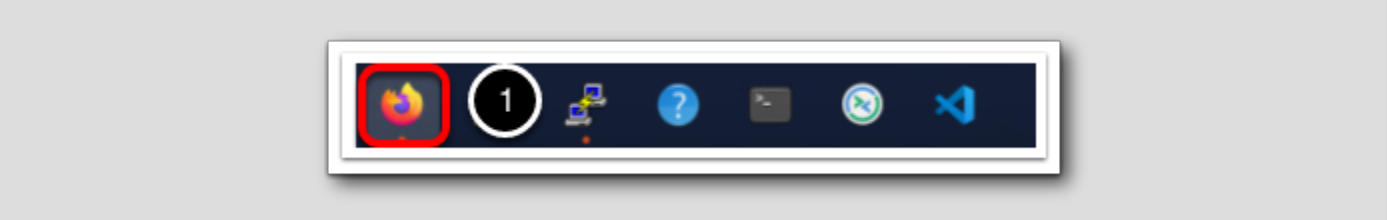
You can get more information and download harbor from *here*.

In this section, you will become familiar with a few of the core capabilities of Harbor. You will see how to scan images for vulnerabilities using an image vulnerability scanner called Trivy and how to push and pull images from the repositories (also known as repos for short). You will also enable content trust so images are signed by the publisher and only signed images may be pulled from the project repo. Most organizations will use a private registry rather than public Docker hub to improve security and latency for their applications. Although Harbor can and would be in production deployed as a highly available application, due to resource constraints we have not done that for this lab.

If you took previous modules in this lab some of the images used were pulled from Harbor.

Open Firefox

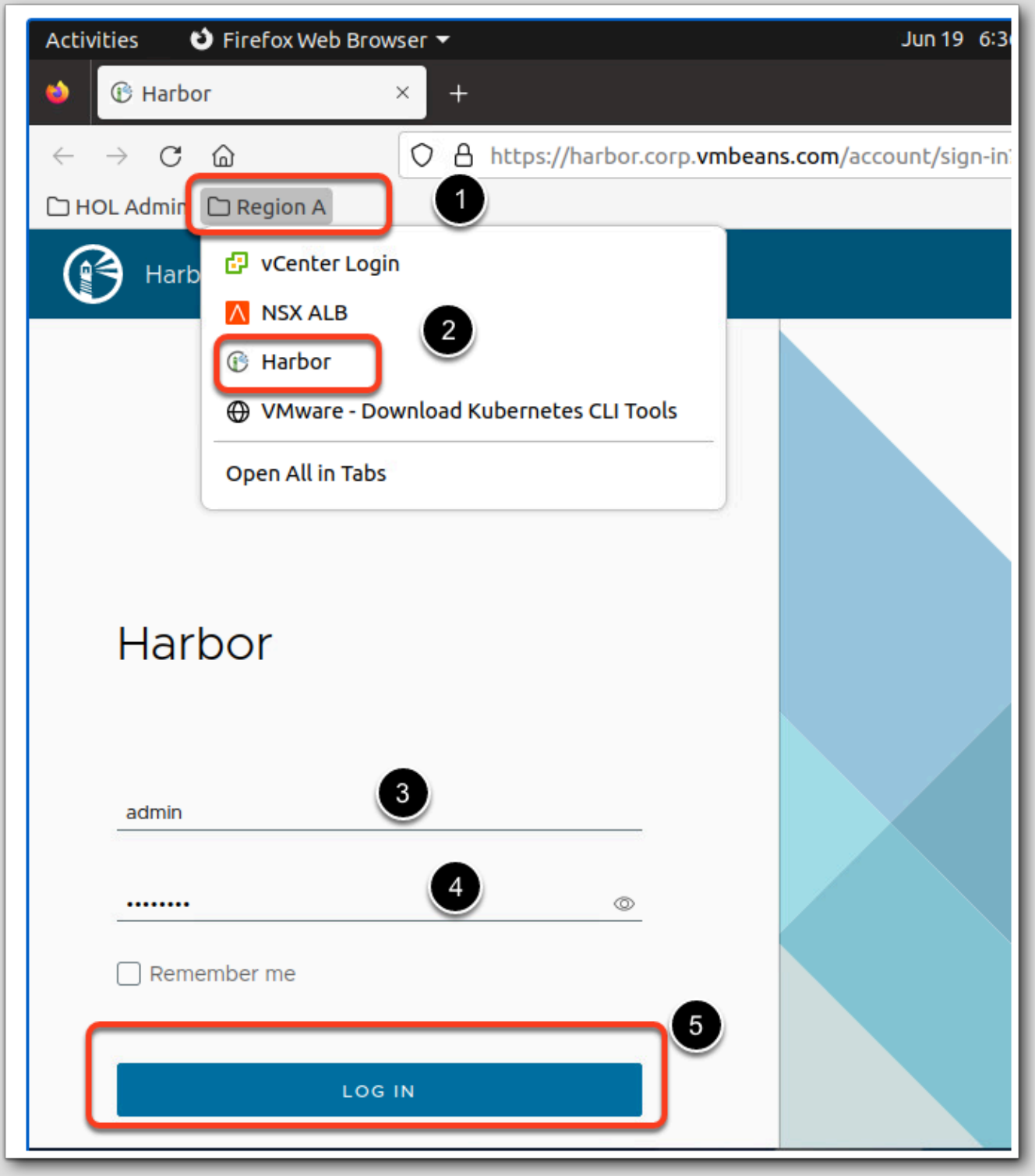
[159]



If you don't already have Firefox open from a previous module then :

1. Click on the Firefox Icon on the Quick Launch Task Bar.

Log in to Harbor


Activities
Firefox Web Browser
Jun 19
Harbor
https://harbor.corp.vmbeans.com/account/sign-in
HOL Admin
Region A
1
vCenter Login
NSX ALB
2
Harbor
VMware - Download Kubernetes CLI Tools
Open All in Tabs
Harbor
admin
3
4
Remember me
5
LOG IN

Start by opening a new tab on the firefox web browser and logging into Harbor:

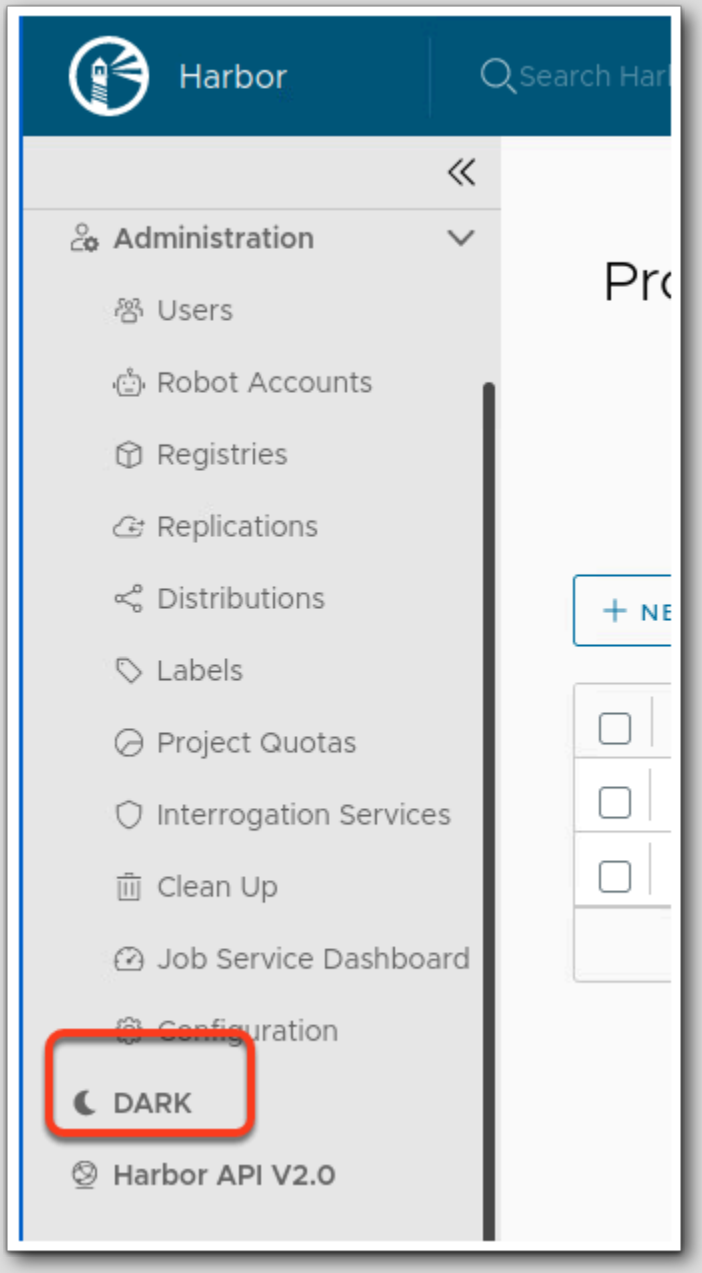
1. Click on the **Region A** Folder in the Bookmark toolbar.
2. Click on **Harbor** link in the Bookmark toolbar.
3. Verify the login is set to "admin".
4. Verify the password is saved.
5. Click the **Login** Button. The credentials should be saved for you.

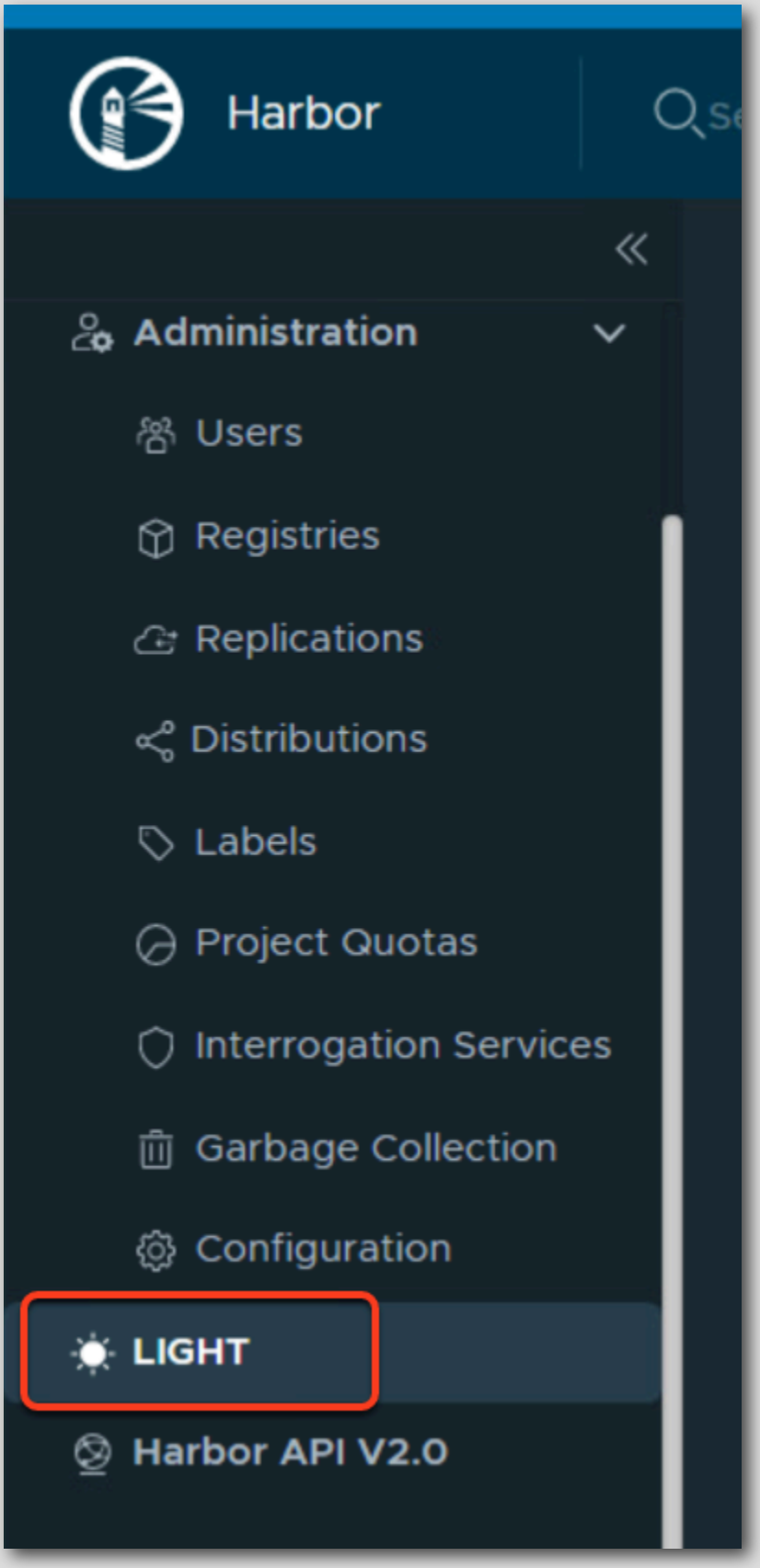
If credentials aren't saved, use the following:

- username: admin
- password: VMware1!

Set the Theme

[161]




Harbor
Administration
Users
Registries
Replications
Distributions
Labels
Project Quotas
Interrogation Services
Garbage Collection
Configuration
LIGHT
Harbor API V2.0

You can toggle the theme for the User Interface from Light Mode to Dark mode based on your preference. Clicking on the LIGHT will toggle to DARK and vice versa. In this lab we used LIGHT mode.

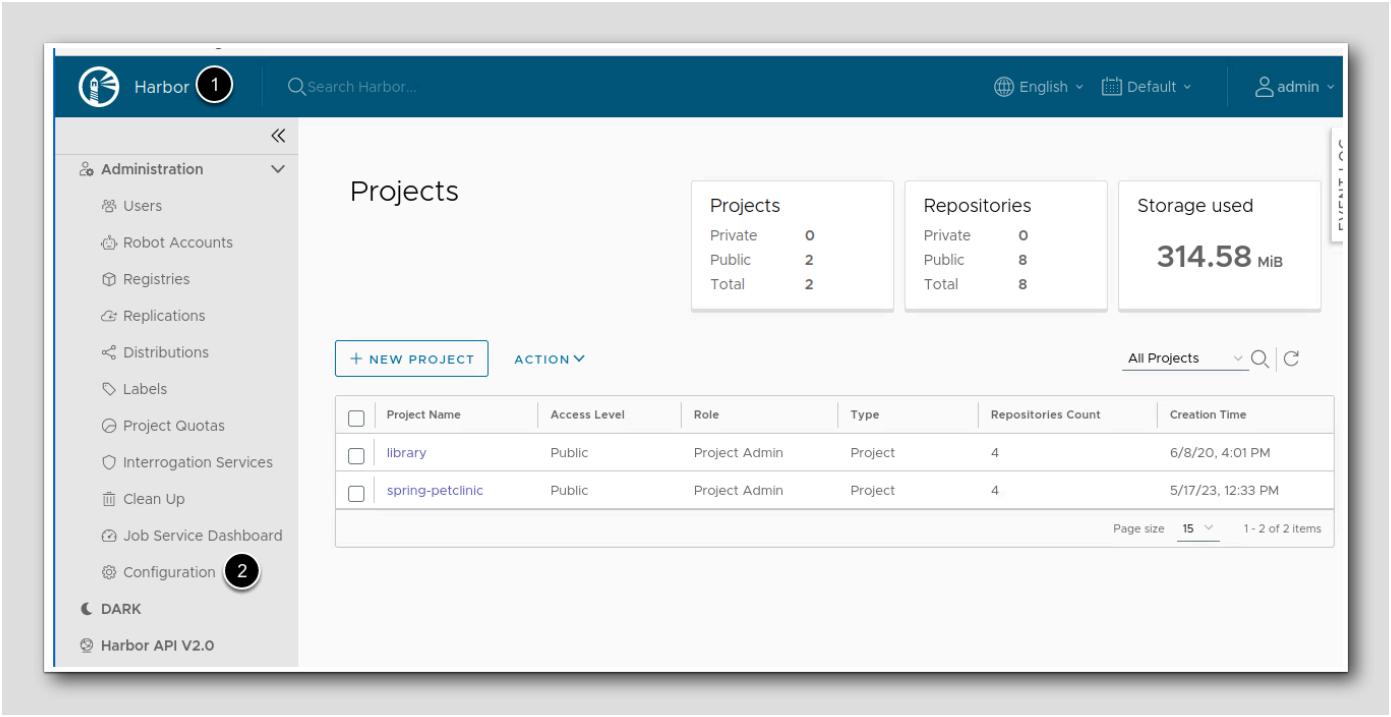
Examine some of the Capabilities of Harbor

[162]

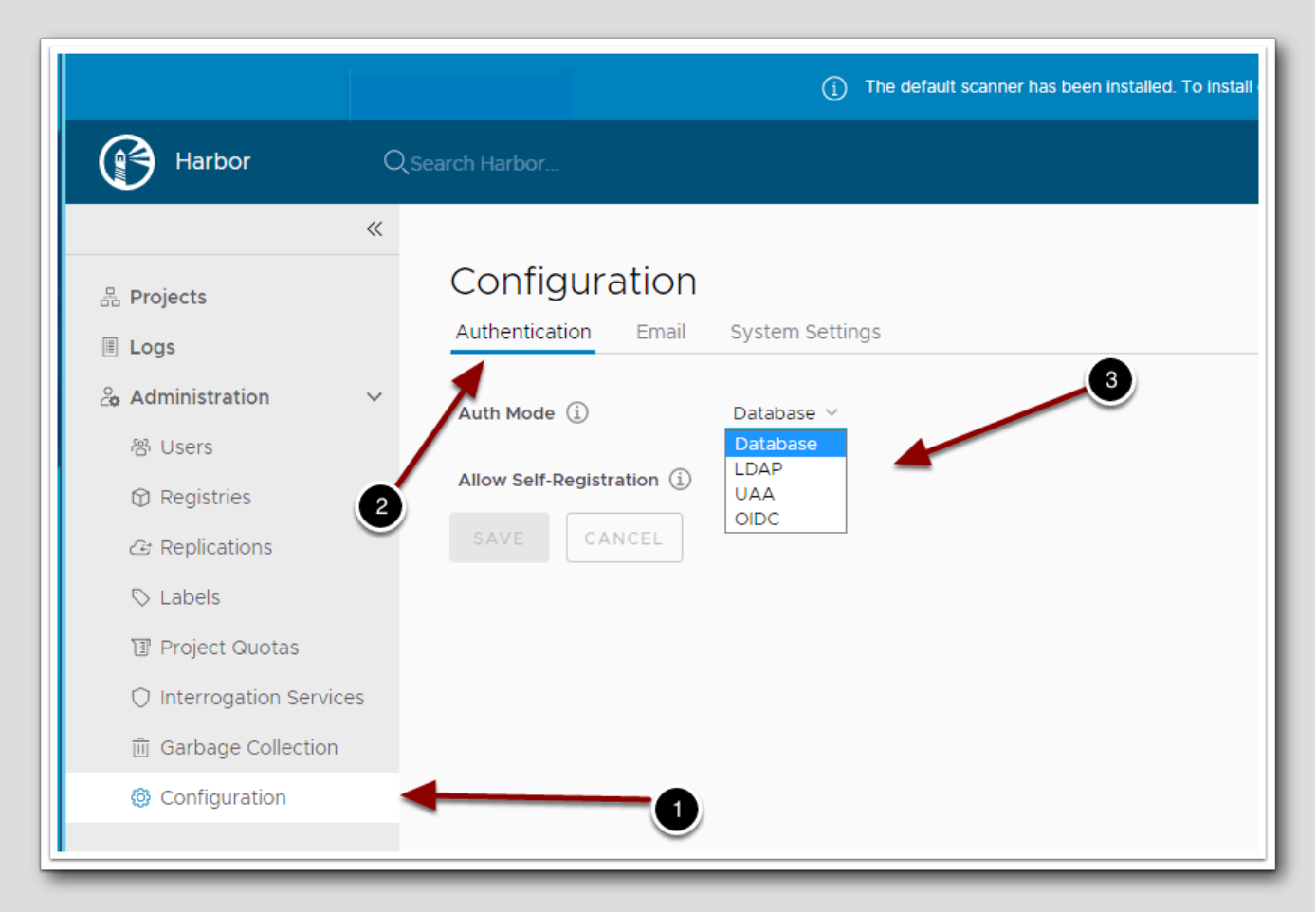
Harbor is an Open Container Initiative (OCI) compliant registry that secures artifacts with policies and role-based access control, ensures images are scanned and free from vulnerabilities, and signs images as trusted. Harbor delivers compliance, performance, and interoperability, to consistently and securely manage artifacts across cloud native compute platforms like Kubernetes and Docker. Let's take a look at some of the capabilities of Harbor that make it ideal for use as a secure code repository.

Configuring Harbor Authentication

[163]



1. if you are not on the landing page for Harbor click the lighthouse symbol. *NOTE: At any time clicking the lighthouse returns you to the landing page.*
2. Click **Configuration**



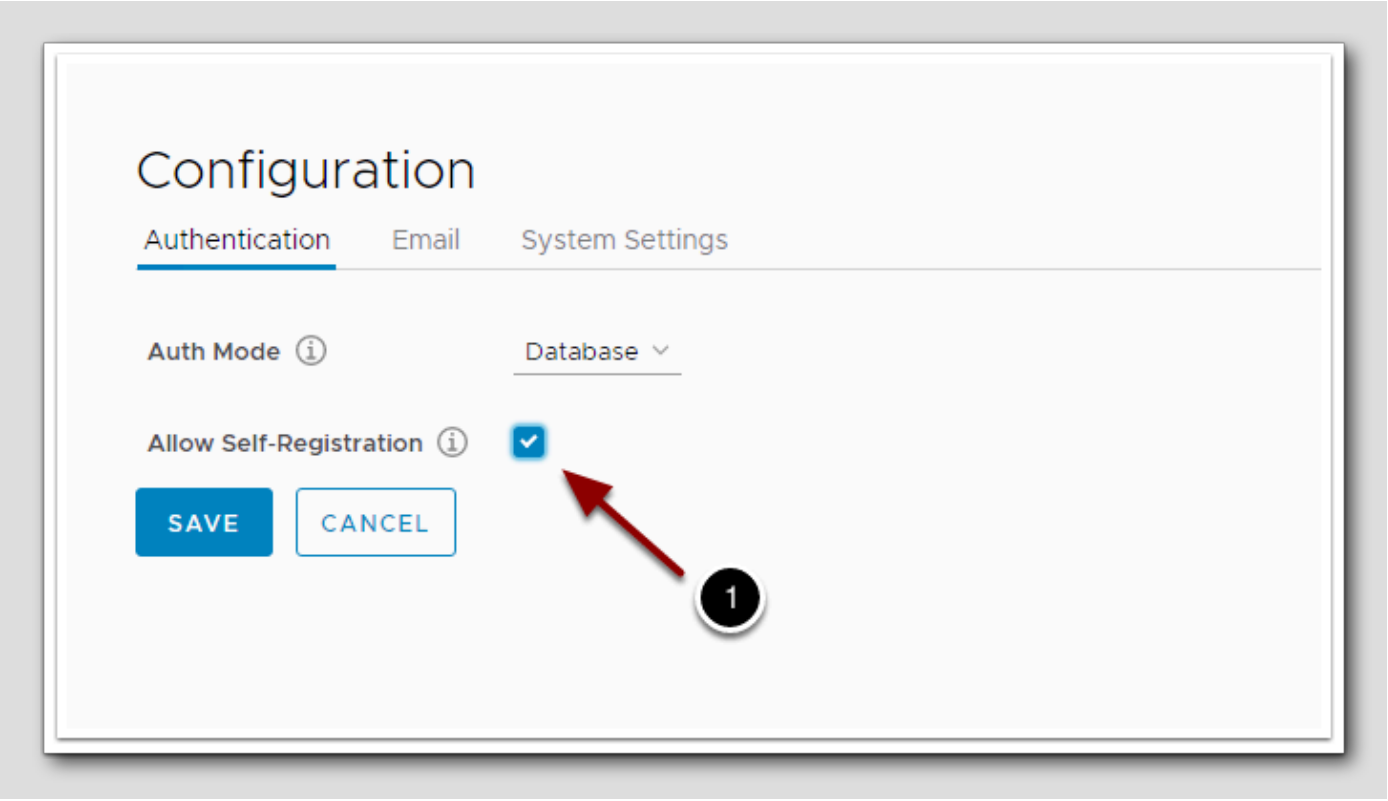
1. Click **Authentication**

Here you can set the authentication mode. For the purposes of this lab we've set Harbor to "Database" which means it's using a internal user database. However, in production you will likely use one of the other authentication options.

Note that this lab is set to "Database" by default and no other options are available, the screenshot is showing the options that would be available in a typical production deployment.

Allow Self-Registration

[164]

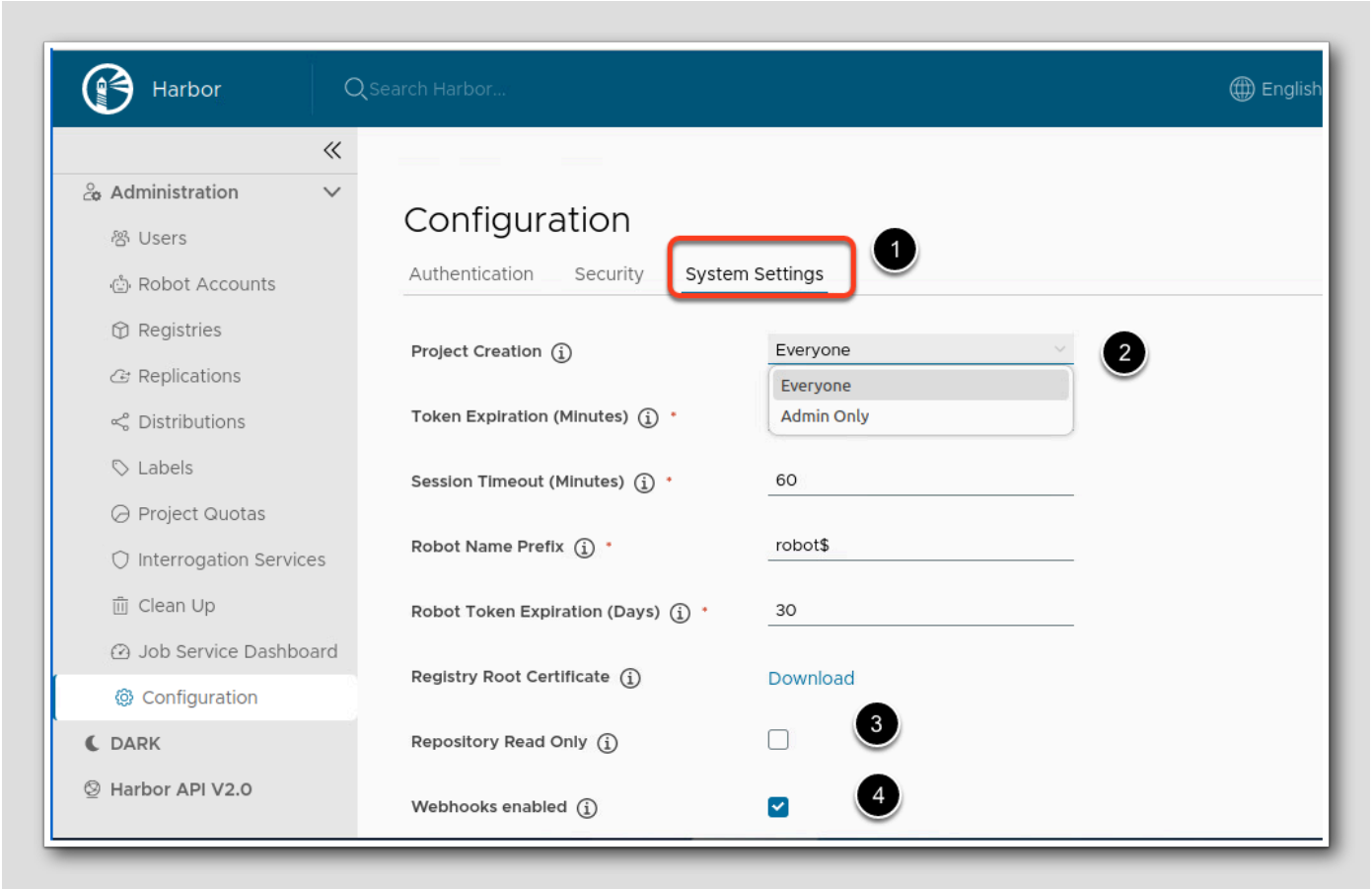


1. You can optionally select self registration by clicking the **Allow Self-Registration** checkbox

If you enable the self-registration option, users can register themselves in Harbor. Self-registration is disabled by default. If you enable self-registration, unregistered users can sign up for a Harbor account by clicking Sign up for an account on the Harbor login page.

Make the Registry Read Only

[165]

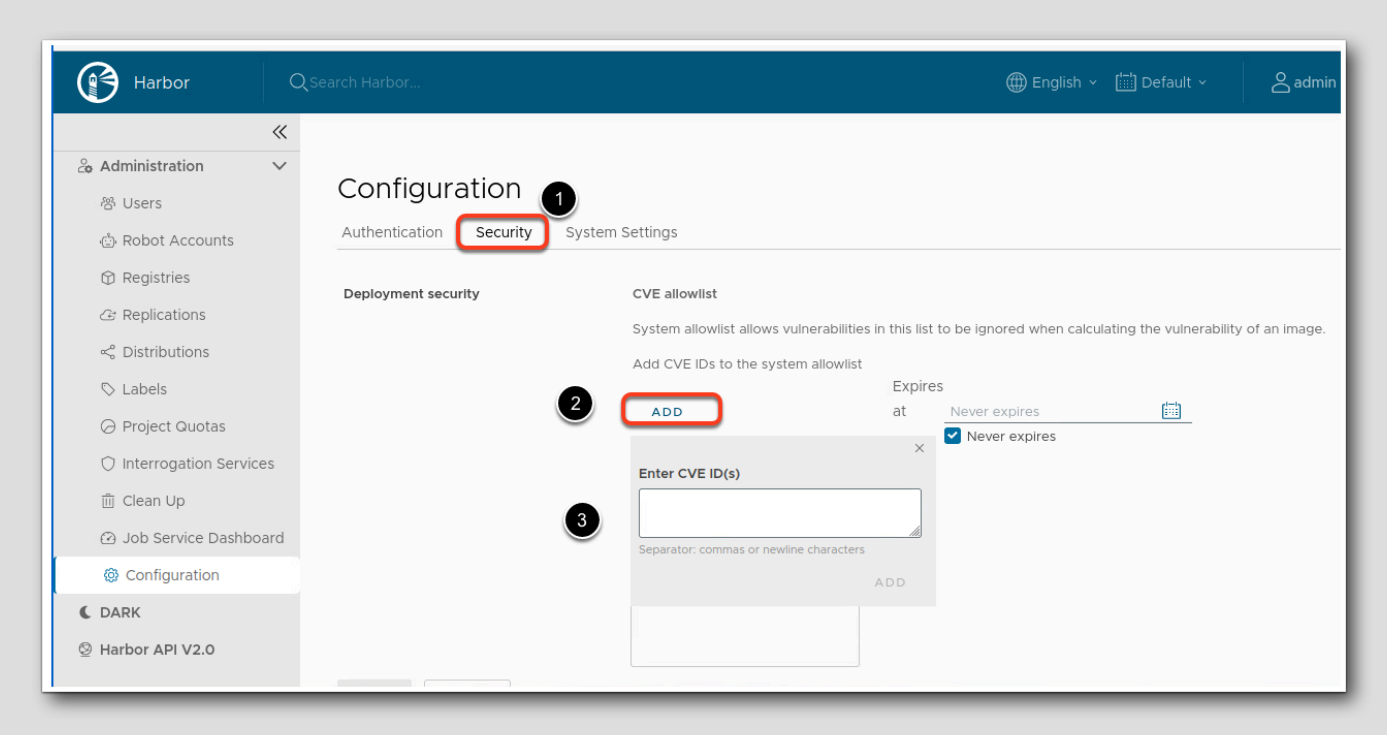


1. Click System Settings
2. You can limit the ability to create a Project to only the Administrator by selecting Admin Only (more on projects later)
3. You can limit Harbor to being a read only repository for non-administrators by selecting Repository Read Only
4. You may also want to include Harbor as part of a CI-CD pipeline, to enable callbacks at designed endpoints. To enable this feature select "Webhooks enabled"

Set Whitelists

[166]

As mentioned previously one of the key features of Harbor is the ability to perform vulnerability scans on objects and to limit the ability to pull objects from Harbor based on the results of the vulnerability scans. There may be times when you want to be able to allow certain detected vulnerabilities to be ignored. You can add them in the whitelist here.



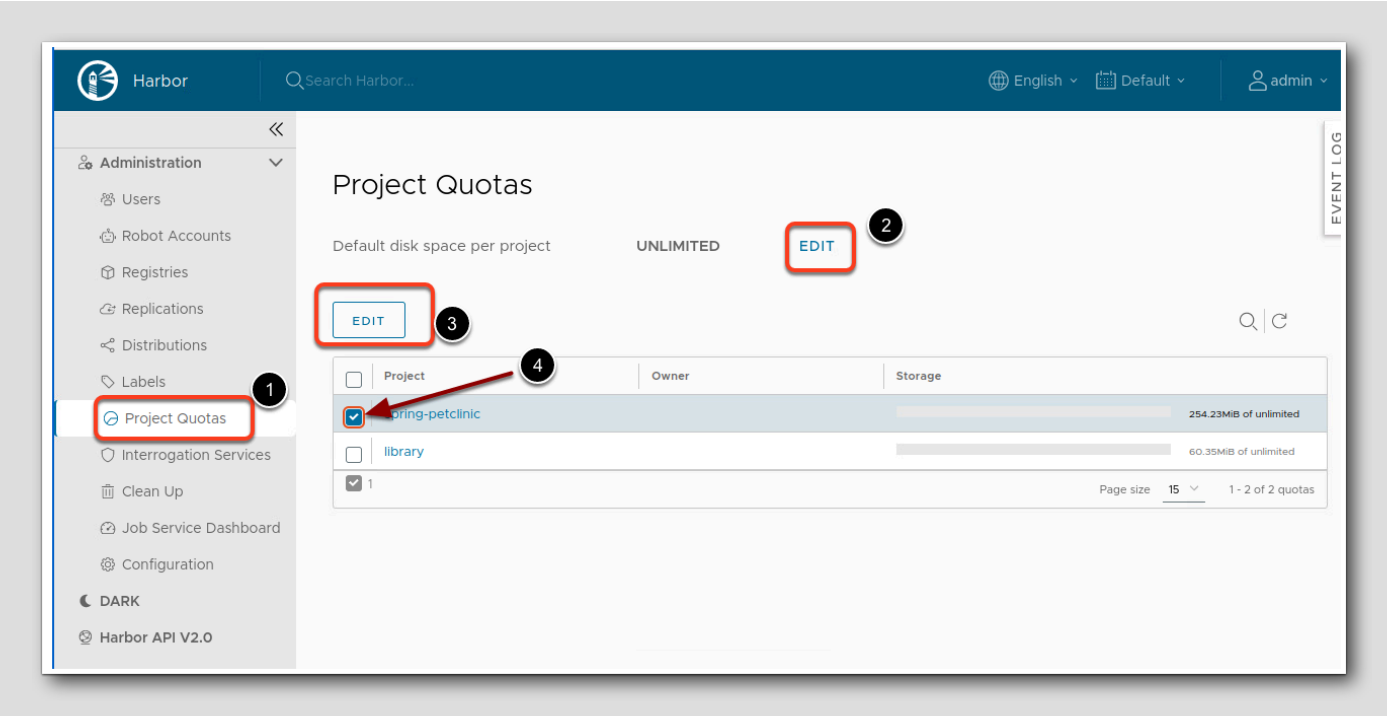
1. Select "Security" from the top menu.
2. Click "ADD"
3. Add a list of any known vulnerabilities that you do not want Harbor to ignore.

Configure Project Quotas

[167]

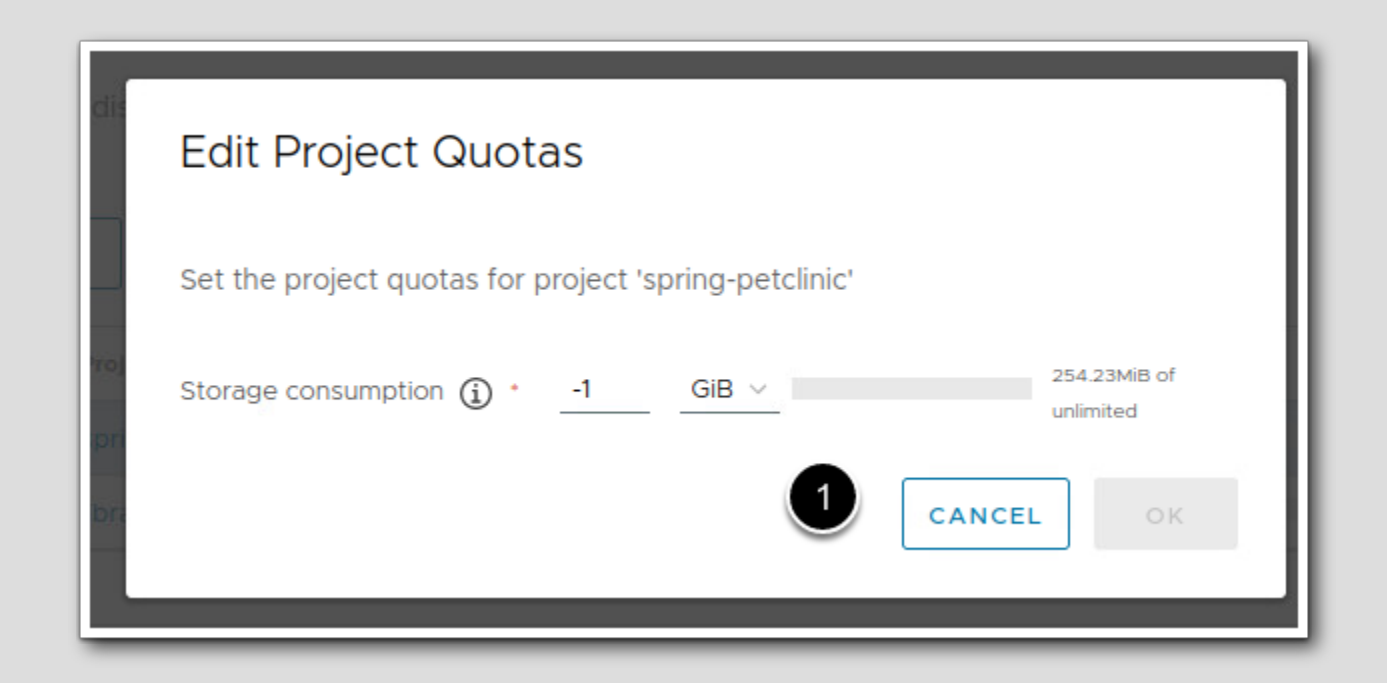
To exercise control over resource usage, as a Harbor system administrator you can set quotas on projects to limit the amount of storage capacity that a project can consume.

You can set default quotas that apply to all projects globally. You can also set quotas on individual projects. If you set a global default quota and you set different quotas on individual projects, the per-project quotas are applied.



By default, all projects have unlimited quotas for storage use.

1. Click **Project Quotas** on the LEFT menu.
2. **To set a default quota policy for ALL projects click EDIT.**
3. To set a default for a specific project select the project to set the quote for, in this example click the box next to the spring-petclinic project name to allow editing.
4. Click **EDIT** to set/modify individual Project quotas.



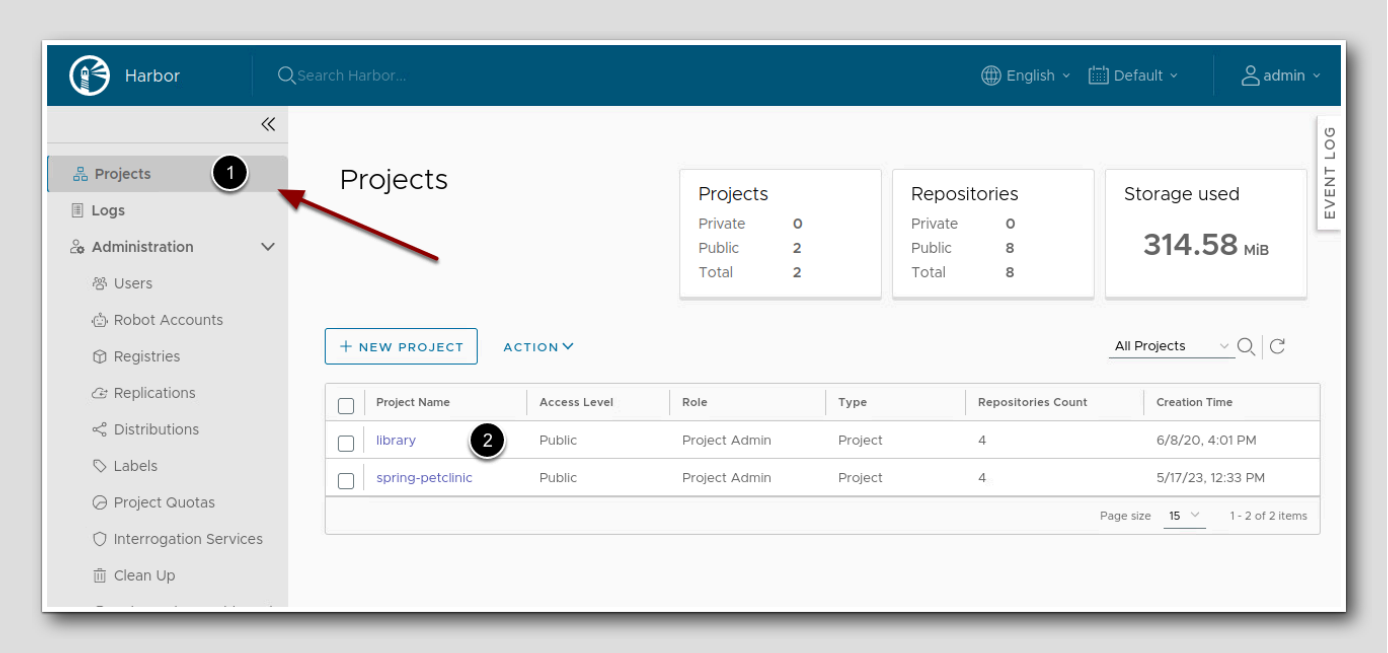
Here you can set limits on the amount of storage that can be used by a project.

1. Click **CANCEL** to exit the Edit project Quotas screen.

View Projects

[168]

Harbor organizes images into a set of projects and repositories within those projects. Repositories can have one or more images associated with them. Each of the images are tagged. Projects can have RBAC (Role Based Access Control) and replication policies associated with them so that administrators can regulate access to images and create image distribution pipelines across registries that might be geographically dispersed. You should now be at a summary screen that shows all of the projects in this registry. Lets start with the project called library.

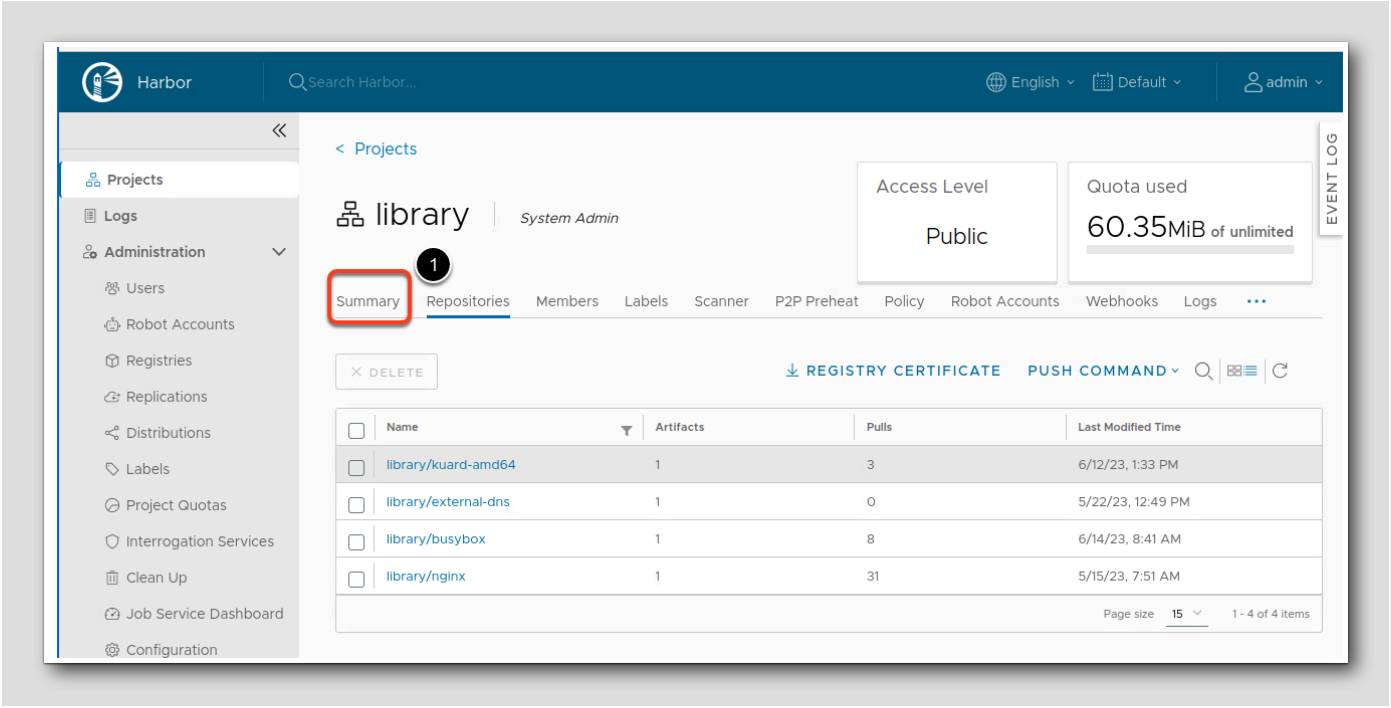


The library project contains eight repositories, two projects and is available to the public.

1. Click on **Projects**
2. Click on **library** to see the repositories that live under the library project

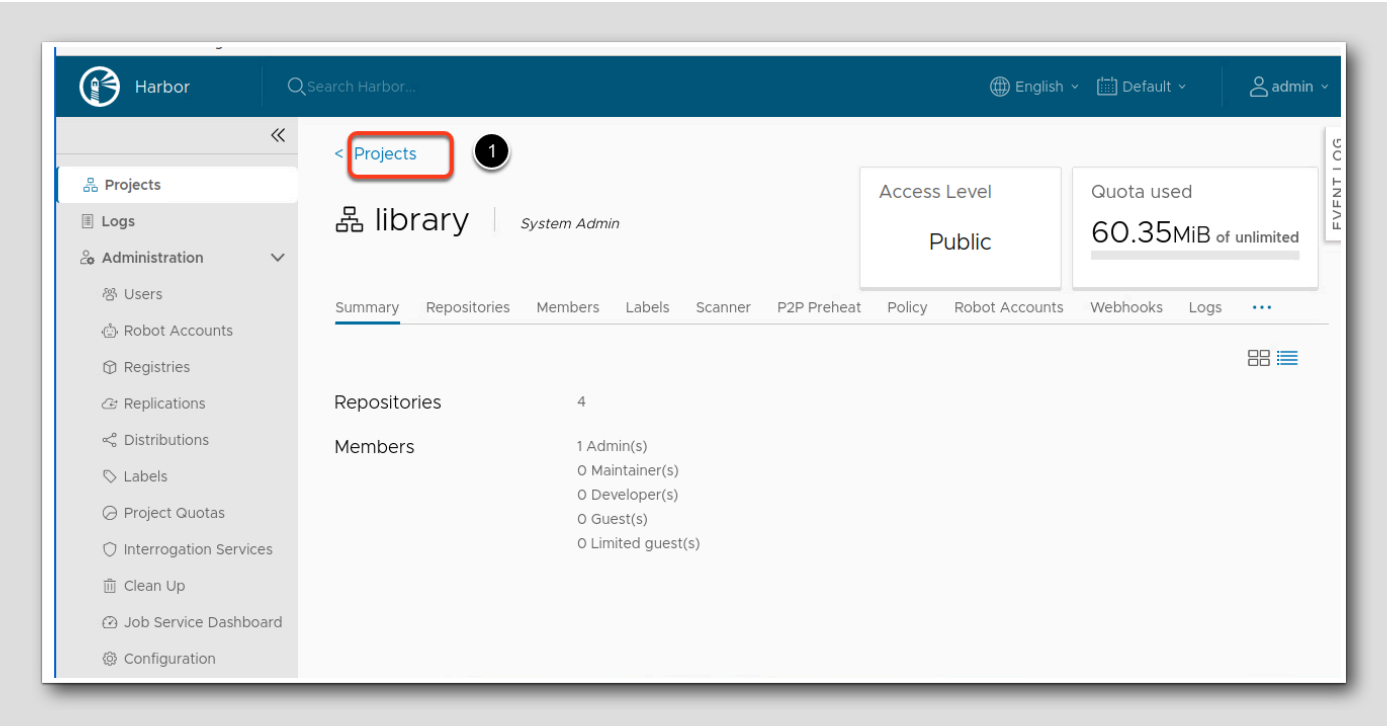
There are two types of project in Harbor:**Public: Any user can pull images from this project.This is a convenient way for you to share repositories with others.Private: Only users who are members of the project can pull images.**

View Repositories



You will now now see the repositories under "library".

1. Click on **Summary**



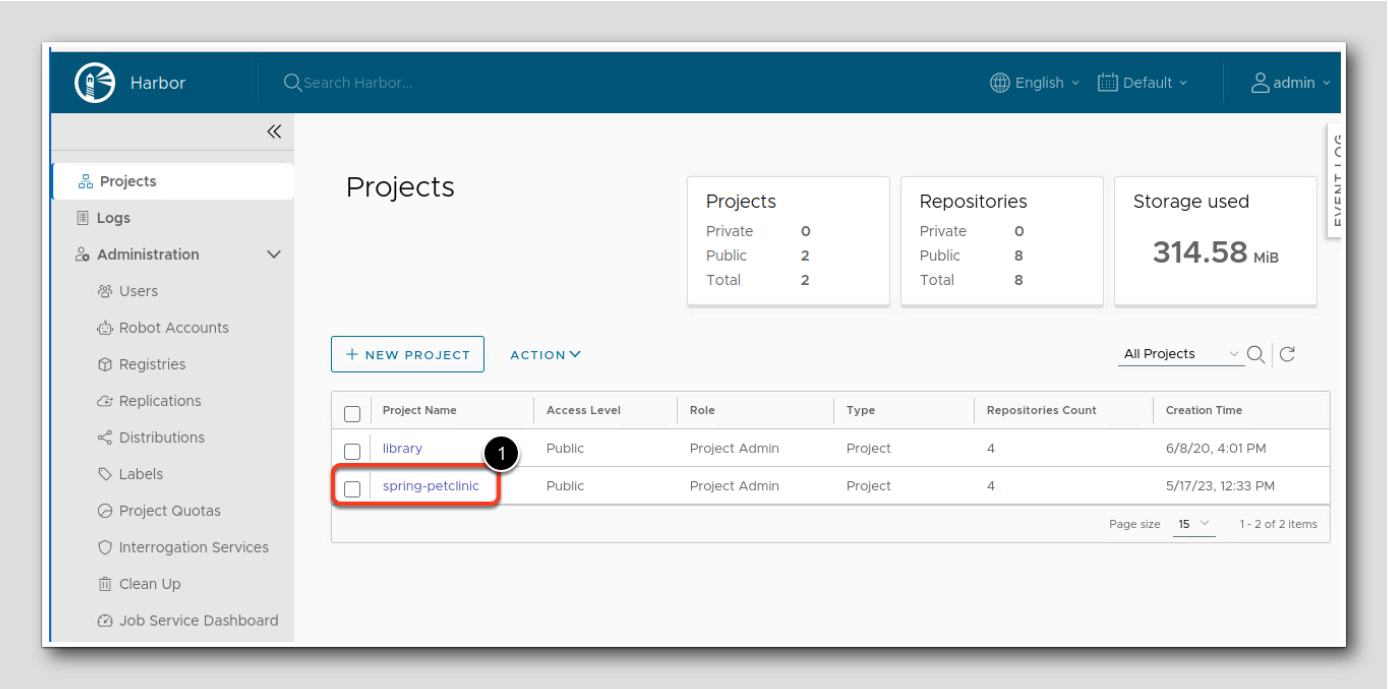
You see a summary of the library project folder, 4 repositories and 1 admin.

1. Click on "Projects" to return to the Projects view.

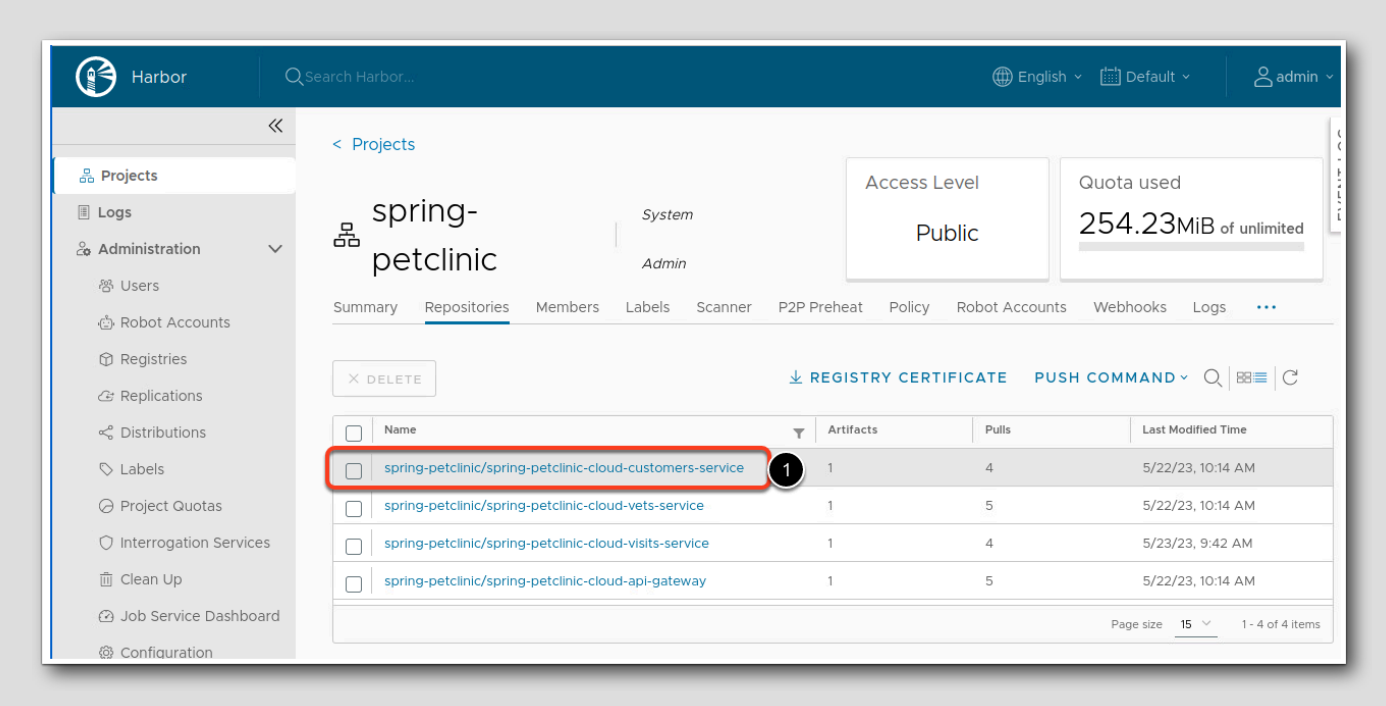
[170]

View the Spring Petclinic Image

[171]



1. From the Projects page click on "spring-petclinic".



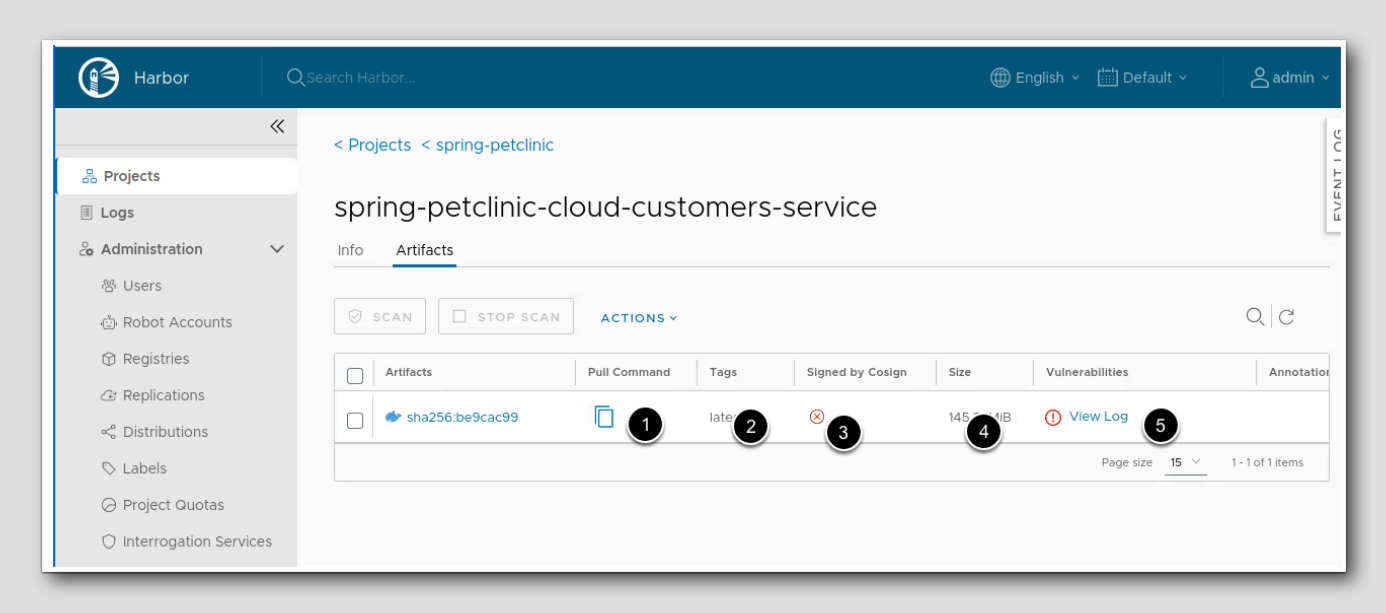
Examine The spring-petclinic Repository

[172]

Here you will see the repositories for the components which make up the spring-petclinic application. Since this is a micro-services application there are multiple components to spring-petclinic.

Let's pick one to examine.

1. Click the first image, spring-petclinic-cloud-customers-service.



Here you will see there is one copy of the image.

1. The **Pull Command** if clicked will copy the git command required to pull the image from Harbor. Try hovering your mouse over the icon.
2. The **Tags** header will show any tags associated with the image, we'll discuss Tags in the next section but briefly they are a way of distinguishing different versions of an image.
3. An indication if the image was "signed" Signed images **provide a way to verify both the source of an image and its integrity**. Container Registry enables users or systems to push images to the registry and then sign them using a master encryption key.
4. The **Size** of the image.
5. Any **Vulnerabilities** reported for the image. Harbor is a secure image repository and among it's security fearures is the ability to install a image scanner such as Carbon Black. We'll talk more about that later.

Tagging an Image

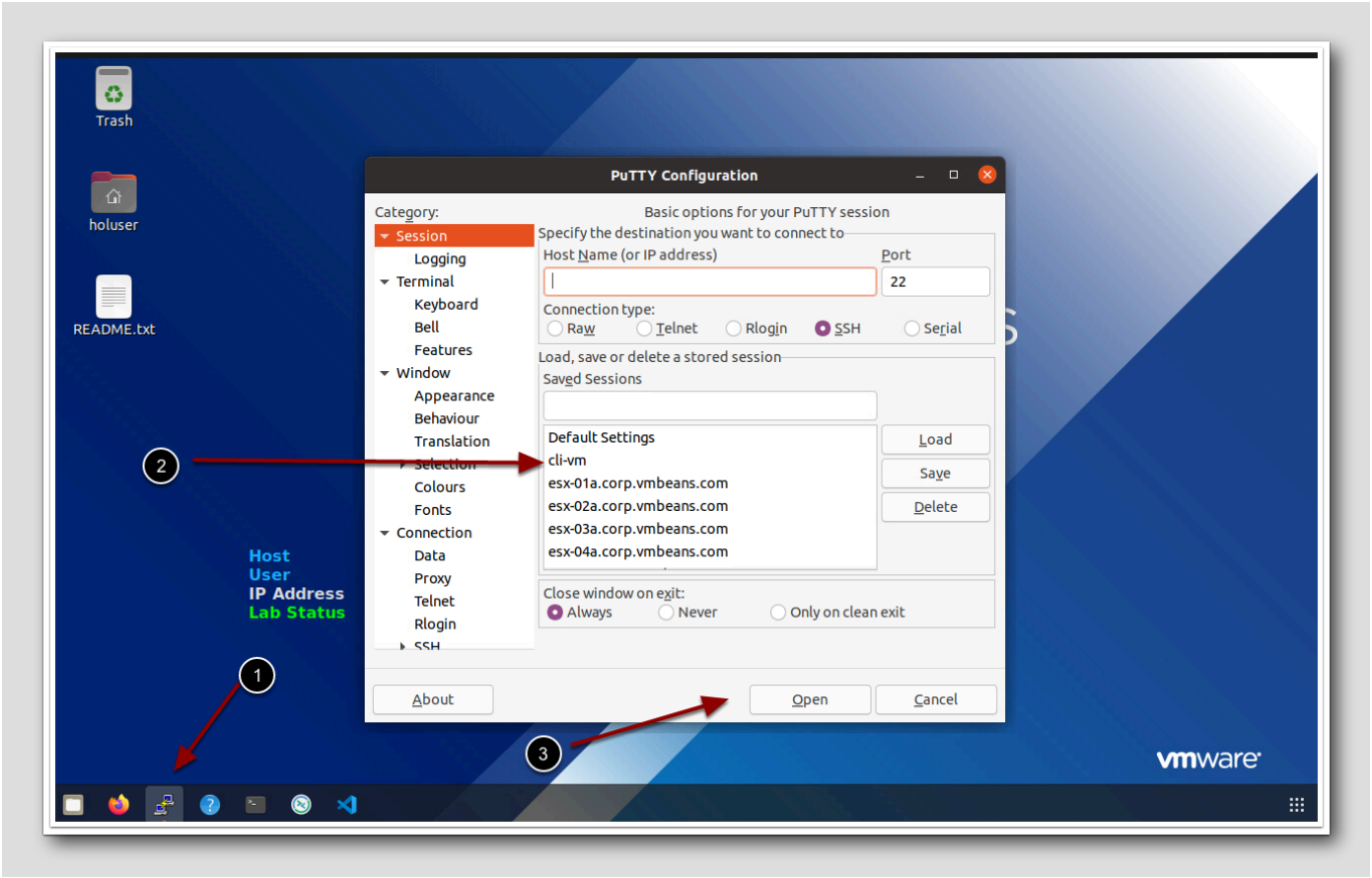
[173]

Previously we introduced the terms image "push" and "pull". What is a "push" and a "pull"?

Quite simply a "push" refers to the action of taking an image from your local instance and adding that to an image repository - you are "pushing" the image up to an image catalog. Conversely when you "pull" an image you are requesting that a copy of that image get "pulled" from an image repository and placed in your local image catalog.

Let's start with looking at the image catalog which exists on our docker repository.

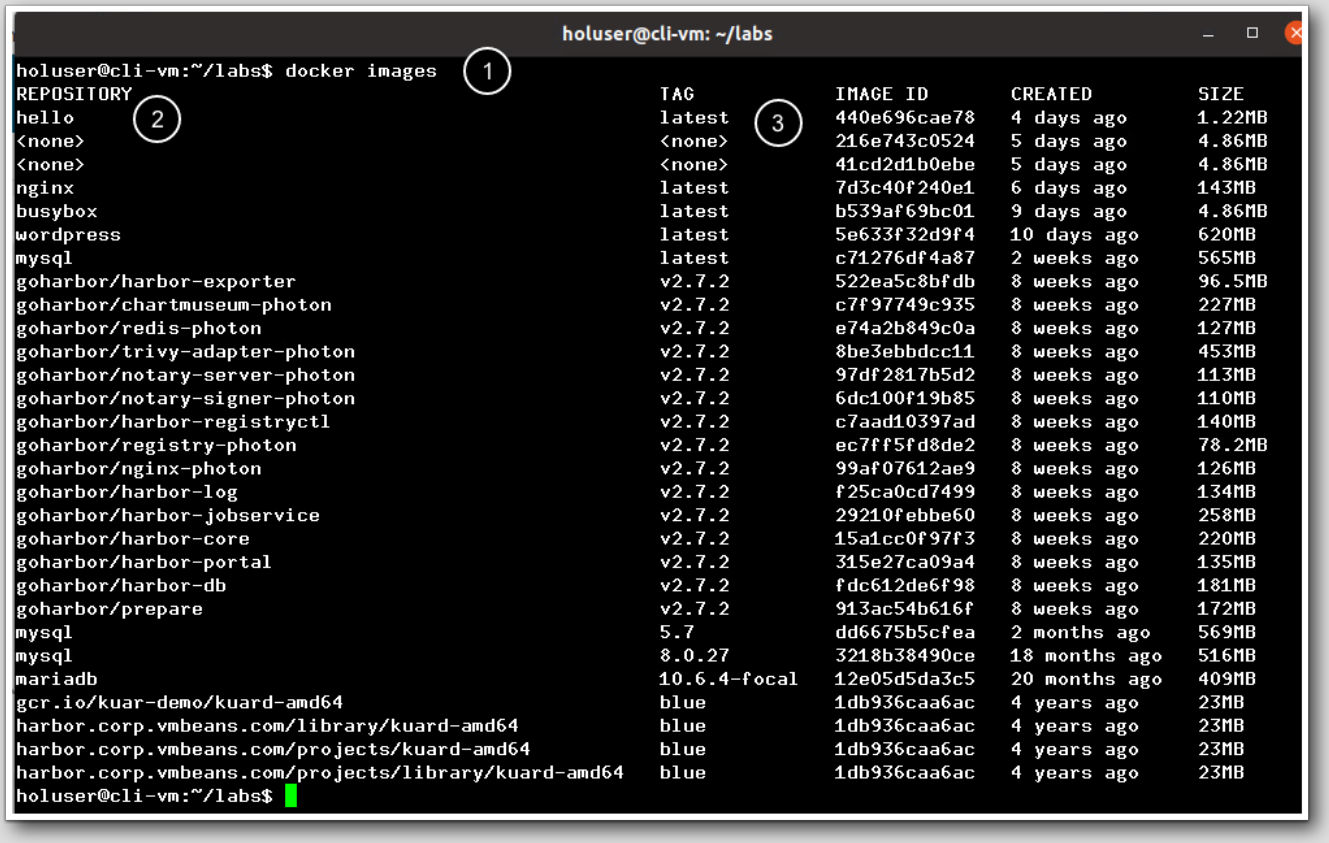
Return to the **cli-vm**. if the cli-vm is already open from a previous module skip the next step.



1. On the taskbar - double click on the **Putty** icon if it is not already open.
2. Select 'cli-vm' from saved sessions.
3. Click Open.

Listing The Docker Images

[174]



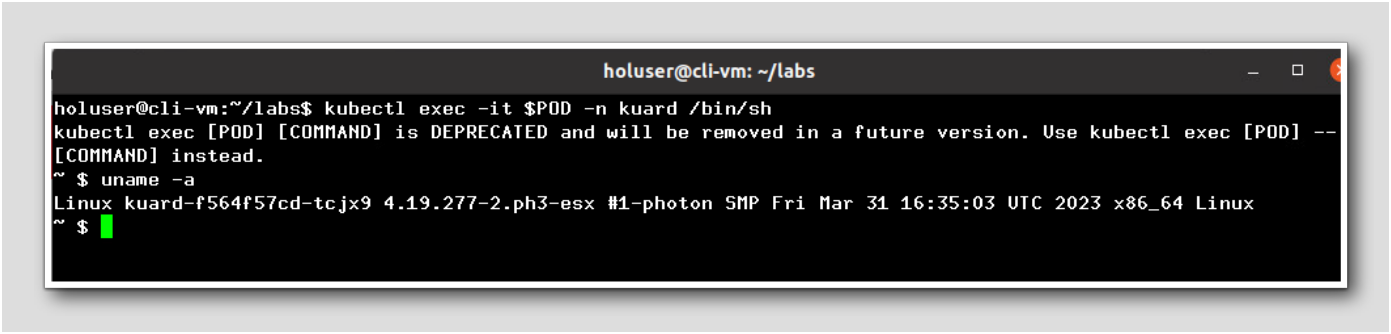
Return to the cli-vm.

1. Type: `docker images`. This lists all the images that are local, eg. in your current docker image catalog. Typically in a CI/CD pipeline, when done with an application you would want to add it to a image repository so it can be shared, tested, etc by other people. Remember this image catalog is local to your instance.
2. Note that the images in my local docker image store show the repository where the image was pulled from. For example harbor-01a.corp.local, projects.registry.vmware.com, gcr.io.
3. It also shows "tags". A "tag" is a way to reference the image. Since there may be multiple iterations of an image, v01, v02, latest, etc … use a tag to help identify the image. If no tag is specified, "latest" is used by default.

Typically a tag is used to distinguish multiple copies of an image, it could be a release number (r1/r2) or a tag like "production".

Let's try to tag and push an image to our harbor repository.

We will start by using an image we already have in our local docker image catalog. Let's use busybox.



First we need to tag the image so we'll tag the busybox image and push it to harbor.

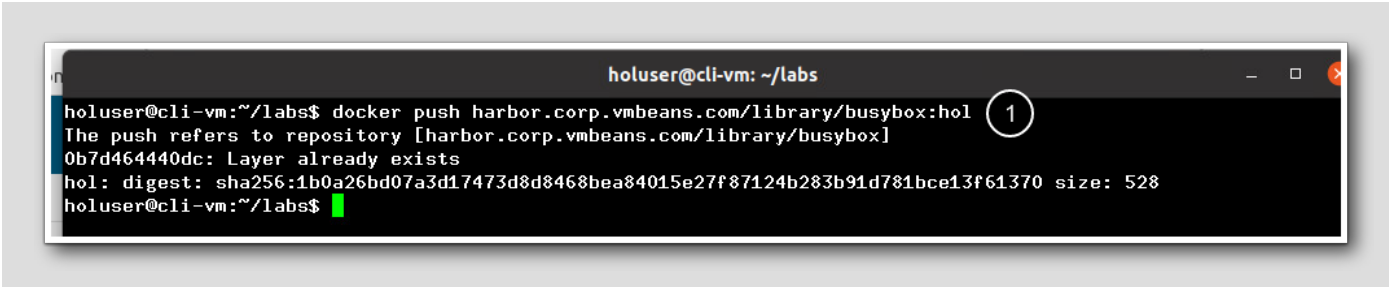
Note with "busybox:hol" we are telling docker to tag the image "busybox" with a tag "hol". The tag allows us to version our application. If we applied no tag the default would be "latest". Typically an image tag is used for identification or versioning, eg. V01, v02, v02.1....

1. `docker tag busybox harbor.corp.vmbeans.com/library/busybox:hol`
2. `docker images | grep busybox`
3. View the tags for "hol" and "latest"

Note we have a few instances of "busybox" all with different tags.

Pushing an Image

[175]

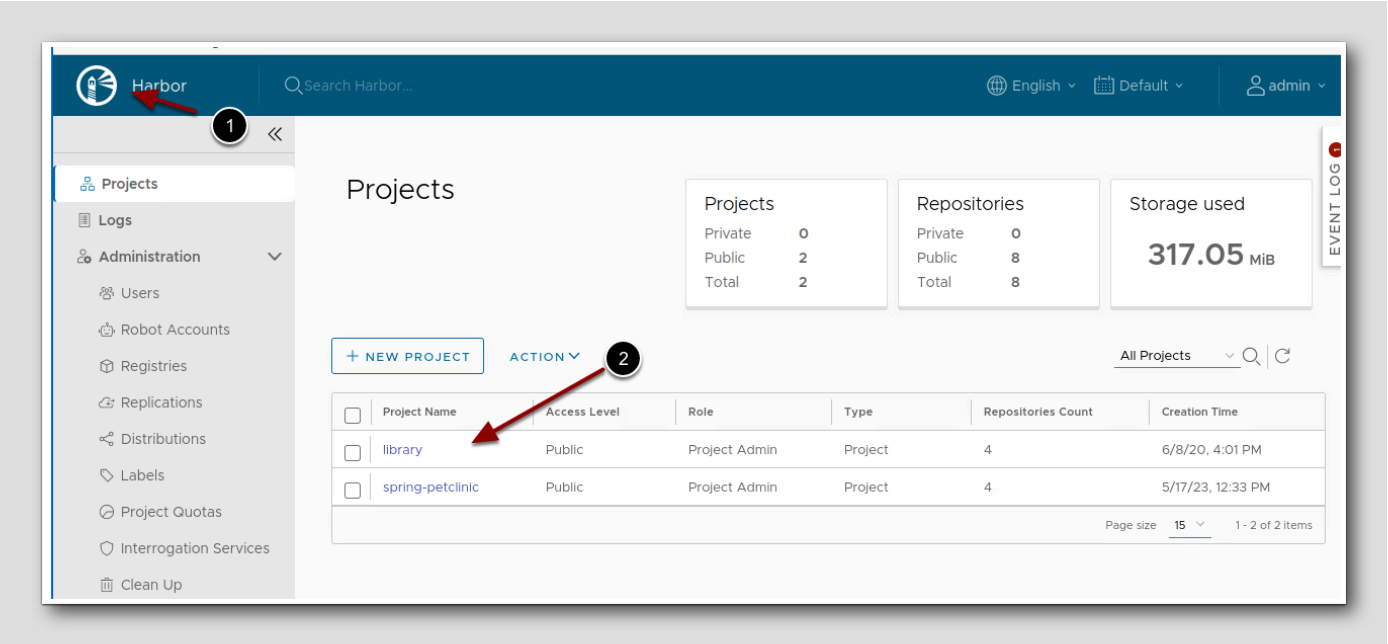


Now let's take the image that we applied the tag to and push it to our harbor image repository. Note that the tagging operation only assigned a value to the image in our local docker image cache, now we are going to add that image to the harbor image repository. The docker push operation will make a copy of the image that exists in our local image cache in our harbor image repository.

1. Type `docker pushharbor.corp.vmbeans.com/library/busybox:hol`

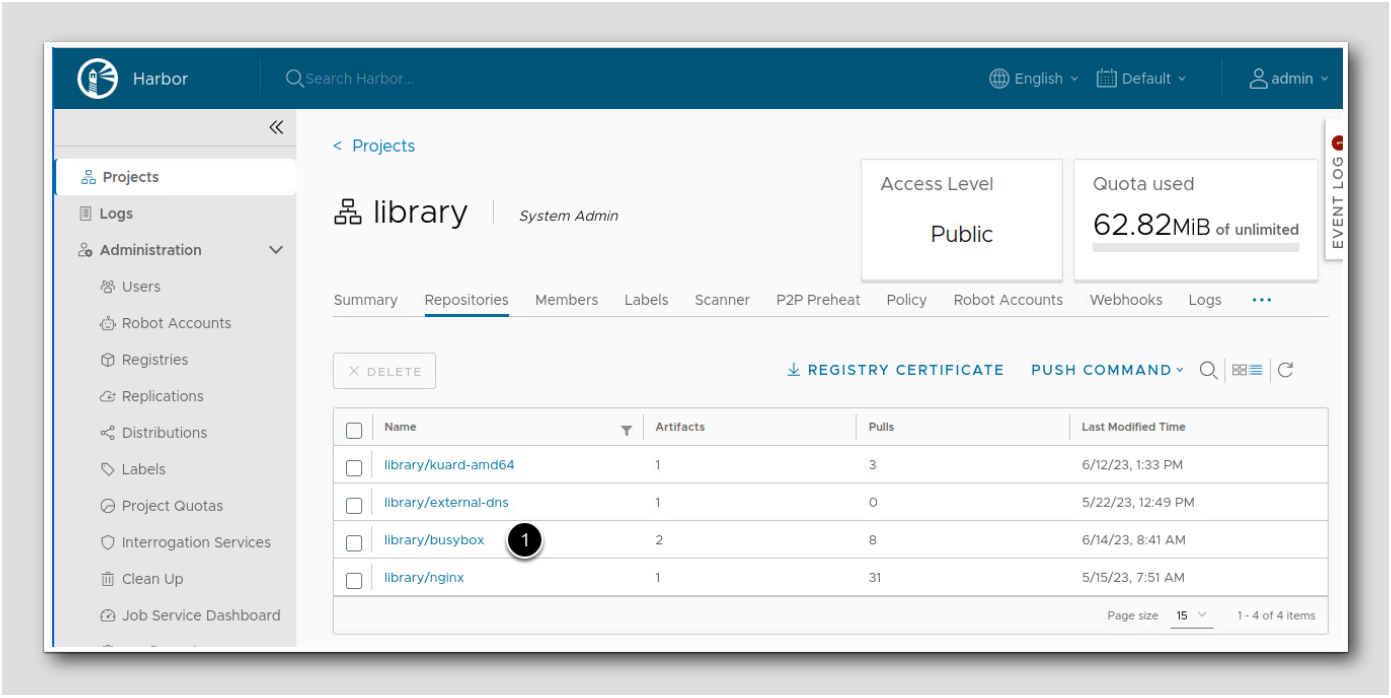
View the Busybox Image in Harbor

[176]



Go back to Firefox.

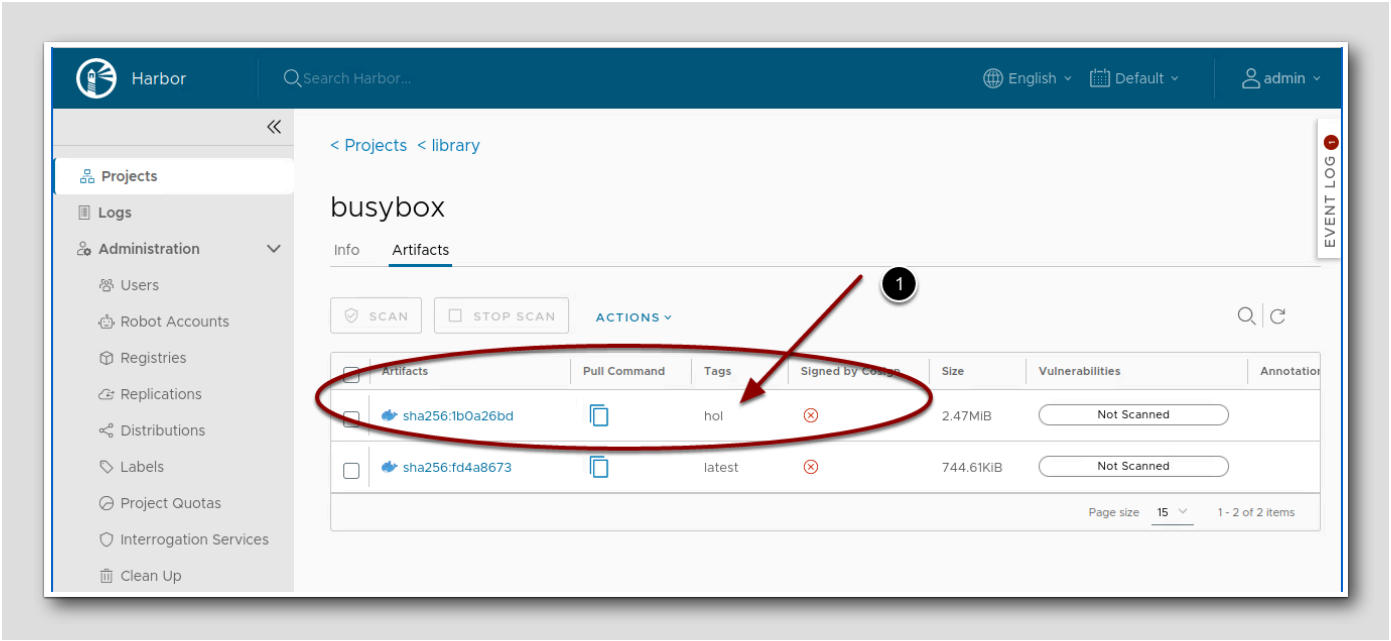
1. Click on the **Harbor lighthouse** to return to the landing page.
2. Click on the **library** project.



Examine The Busybox Image

[177]

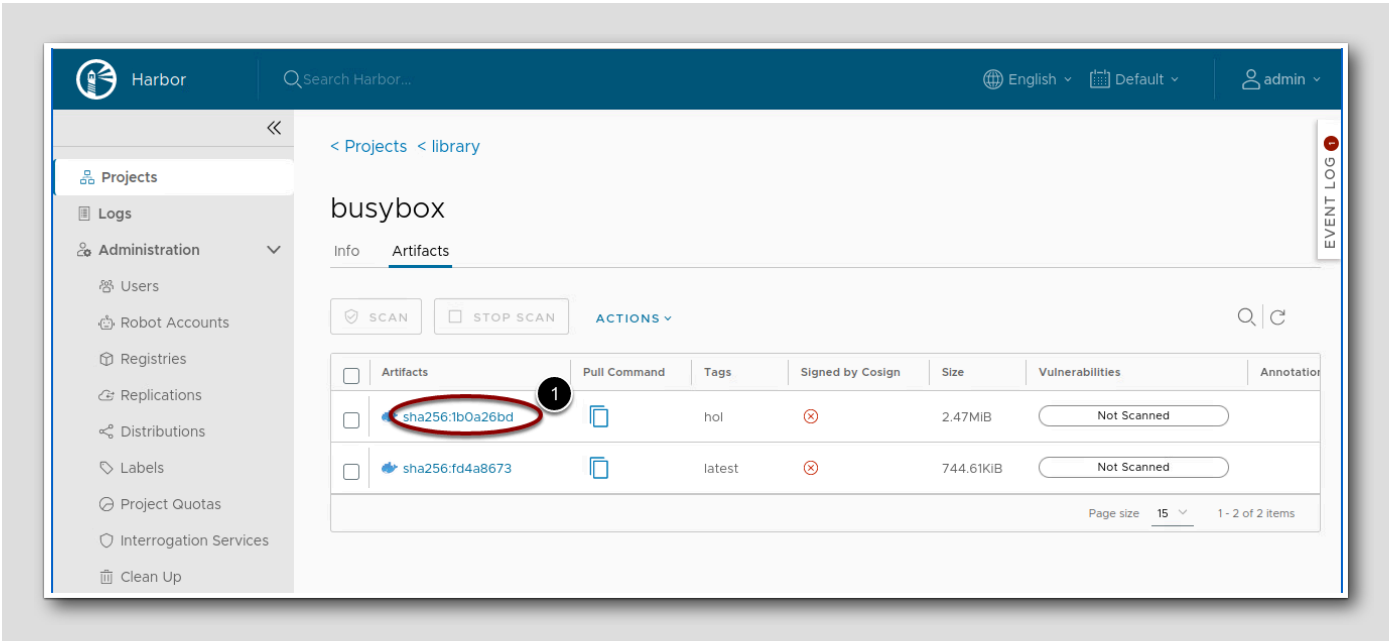
1. Click on the **library/busybox** hyperlink.



1. Note the busybox image you just pushed with the tag "hol" now appears in harbor.

Examining Busybox Further

[178]



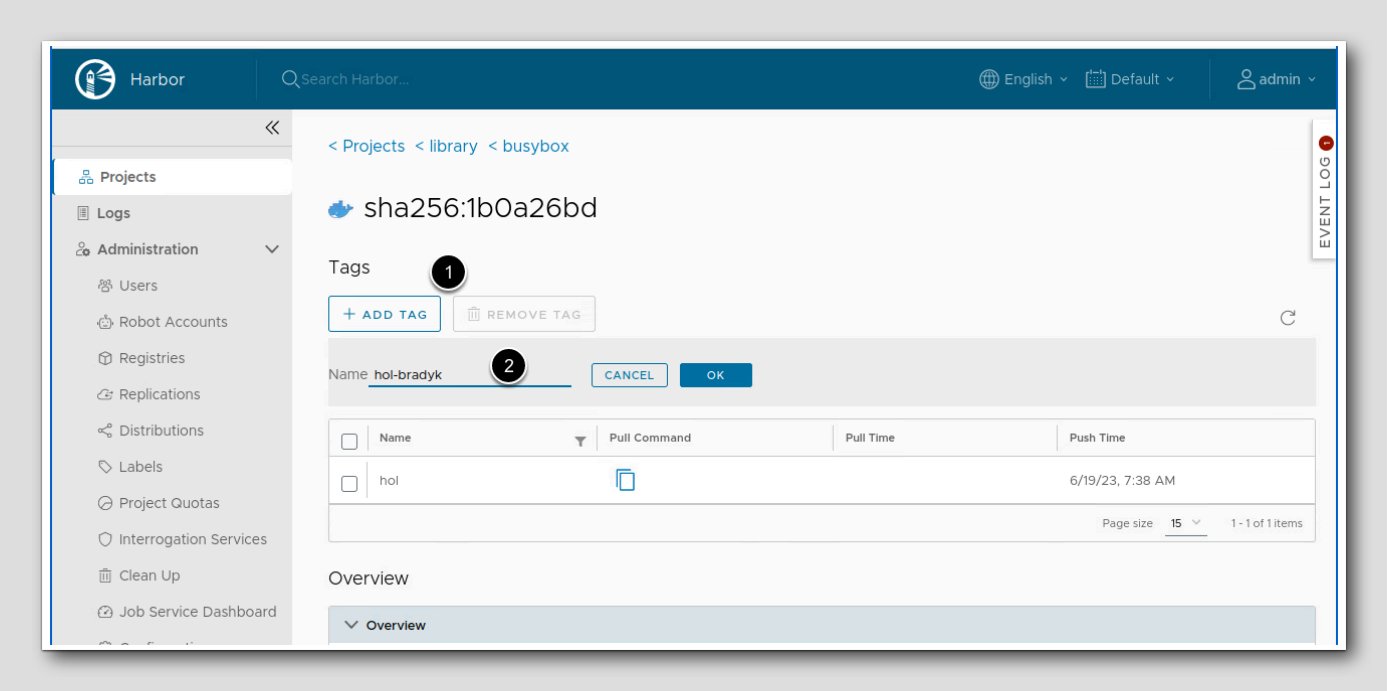
Click on the hyperink starting with "sha256".

You can now examine the details of the image.

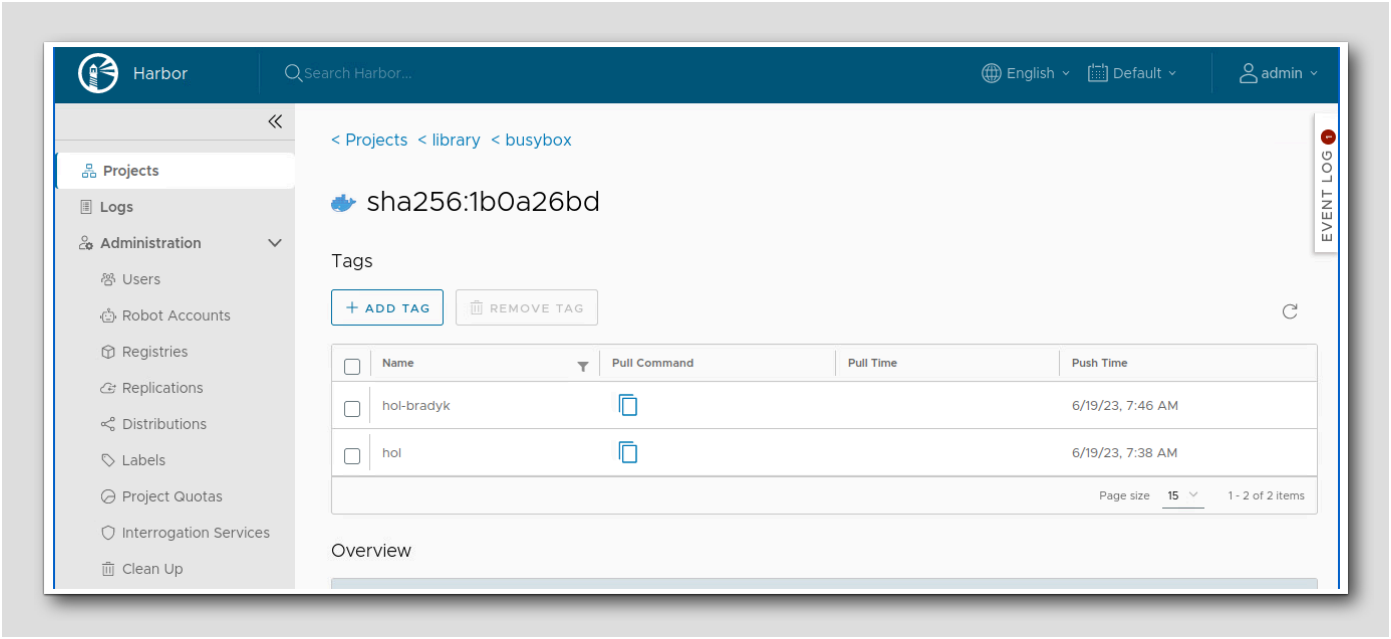
Adding A Tag Through Harbor

[179]

In addition to having the capability of tagging an image using the CLI we can also directly tag an image thru the Harbor GUI.



1. Click "Add Tag".
2. Enter a Tag name, we used "hol-<your name>" but the tag is just an identifier for the image so any contiguous text string will work.
3. Click "OK".



1. The tag we just created is now visible for the image in Harbor.

Examining the Security Features of Harbor

[180]

- Vulnerabilities Scan: it allows you to scan all the docker images registered in the different repositories to check if they have vulnerabilities. It also provides automation during that process to make sure that every time we push a new image it is scanned automatically. Also, it will enable defining policies to avoid pulling any image with vulnerabilities and set the level of vulnerabilities (low, medium, high, or critical) to tolerate. By default, it comes with Clair as the default scanner, but you can introduce others as well.
- Signed images: Harbor provides options to deploy notary as part of its components to be able to sign images during the push process to make sure that no modifications are done to that image
- Tag Immutability and Retention Rules: Harbor also provides the option to define tag immutability and retention rules to make sure that we don't have any attempt to replace images with others using the same tag.

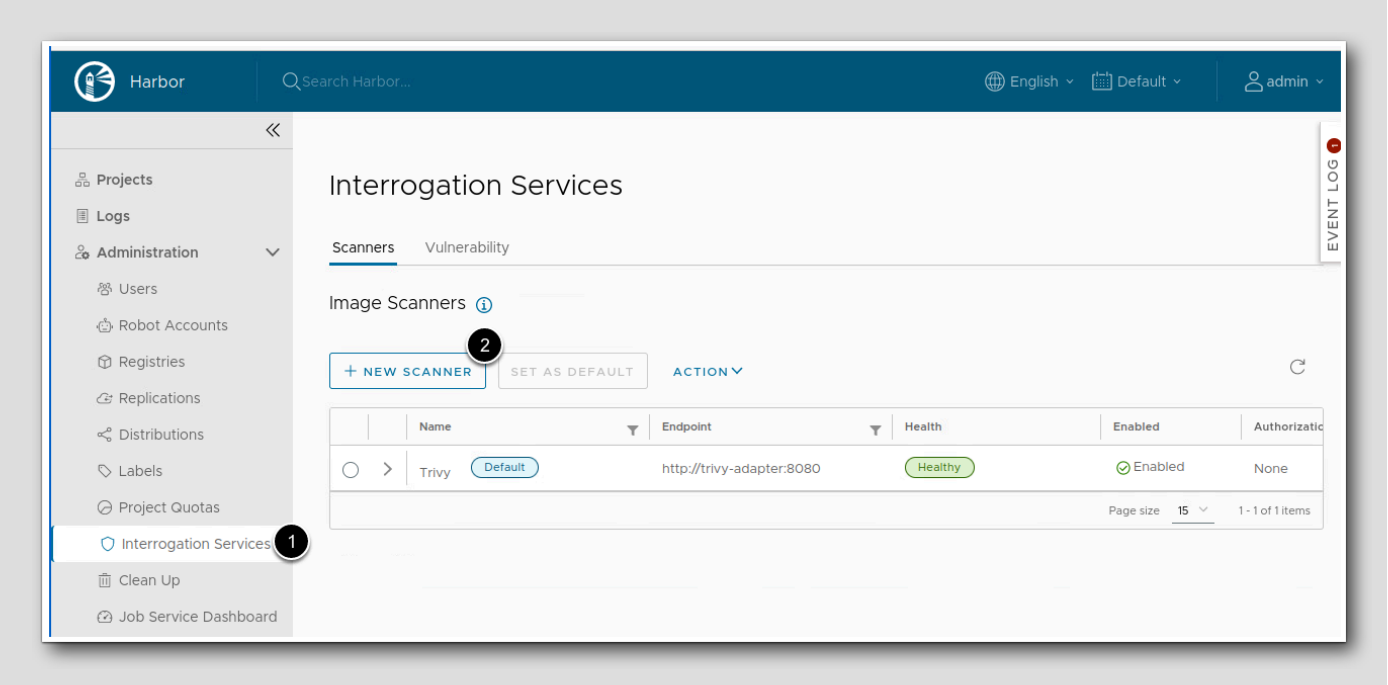
Harbor Vulnerabilities Scanning

[181]

To verify if images contain any known CVE vulnerabilities, Harbor supports several different vulnerability scanners, the default scanner is Trivy but other scanners such as Carbon Black can be used to scan the images at rest. The image scanners then maintain and update a vulnerabilities database by pulling from a configured set of sources. The repositories view provides a quick visual view of the results and a detailed report is available in the web interface.

If Harbor Deployment Security is enabled you can restrict users from downloading images which have known vulnerabilities.

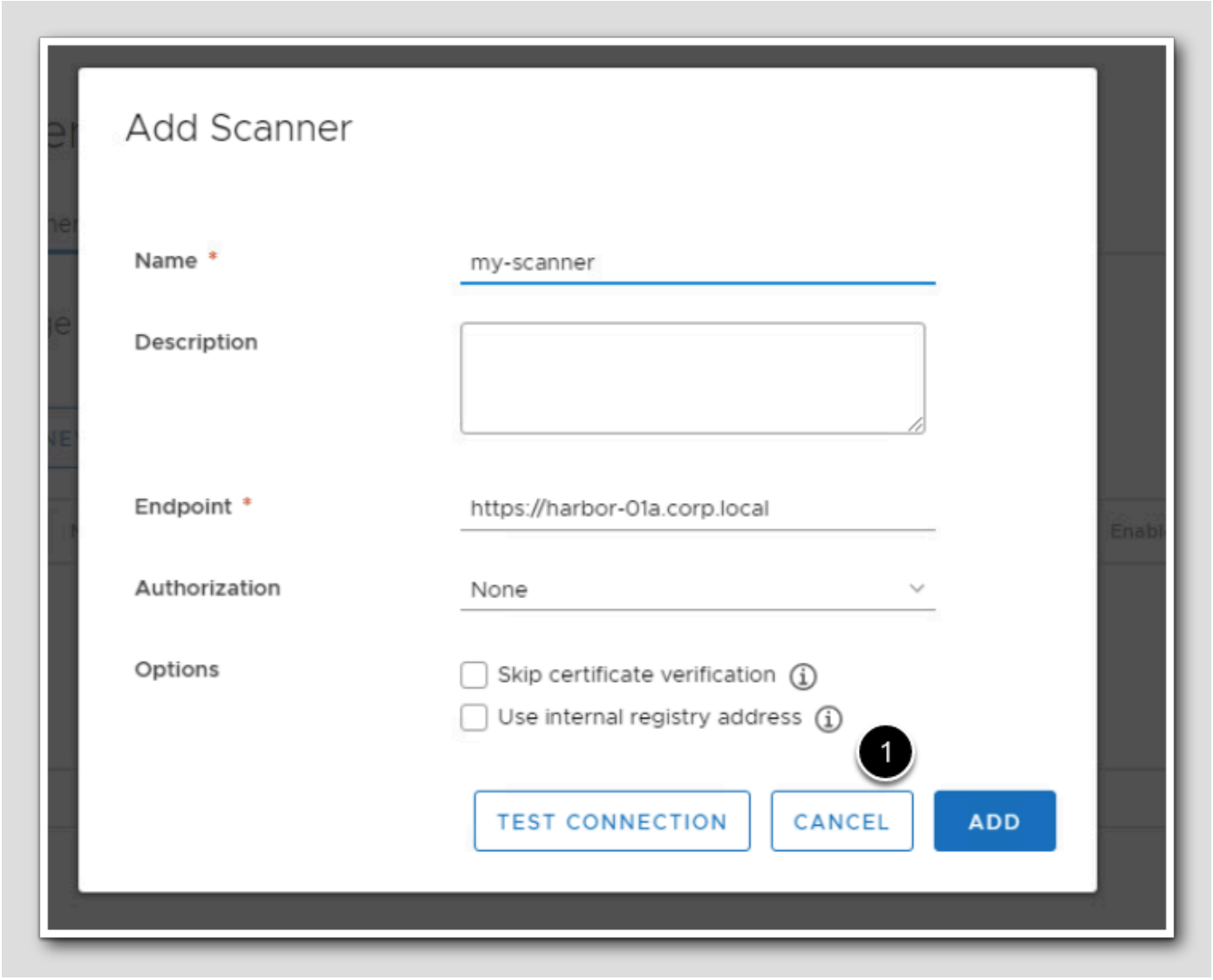
Harbor integrates with third-party vulnerability scanners such as Carbon Black or Trivy to scan your Docker images. In this lab we have installed the Trivy scanner for you.



1. Click Interrogation Services. Note there is a scanner installed, the Trivy scanner is installed for you and is set as the default image scanner.
2. However if you wanted to install a different or additional scanner - Click New Scanner.

Adding Additional Vulnerability Scanners

[182]



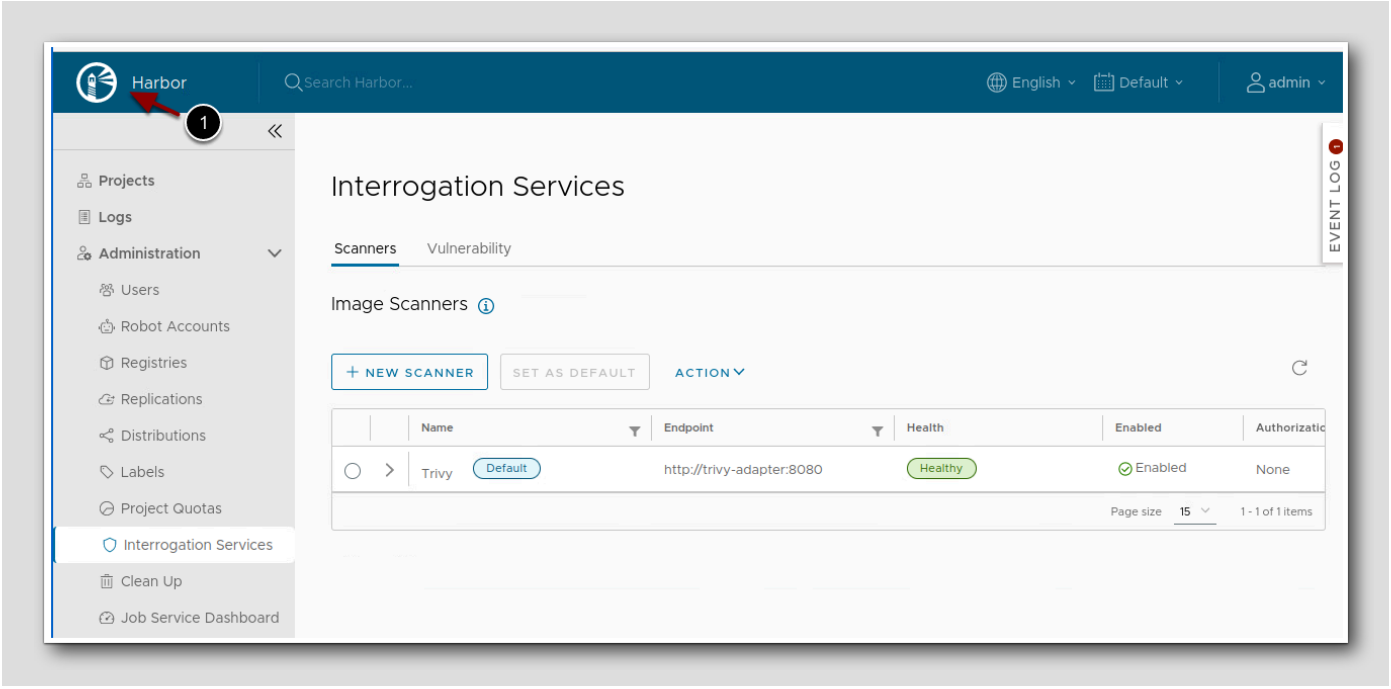
Note: Due to the restricted internet access for this lab you will not be able to add a scanner. The Endpoint adress would be the address of the scanner, in this case we are using Harbor.

The wizard would then walk thru the steps required to install a scanner.

Once installed you can then enable Harbor Content Trust and restrict users from downloading images with vulnerabilities.

1. Click CANCEL to exit the wizard.

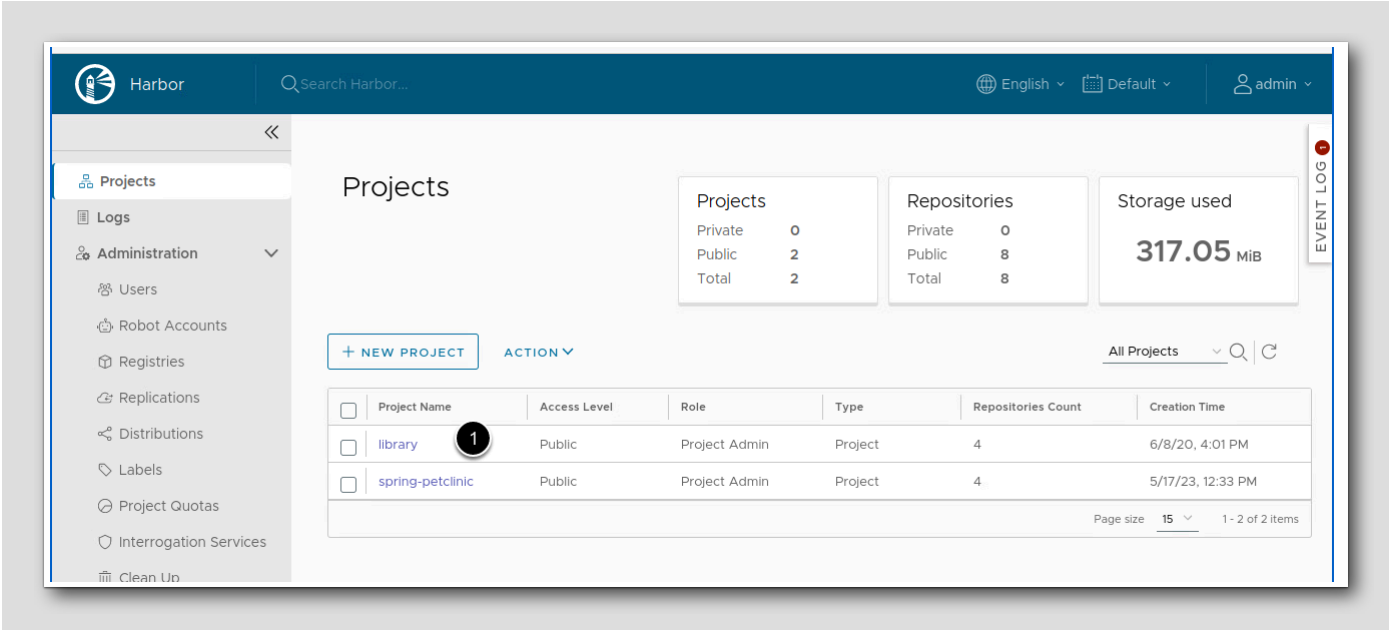
Scanning For Vulnerabilities - Step 1



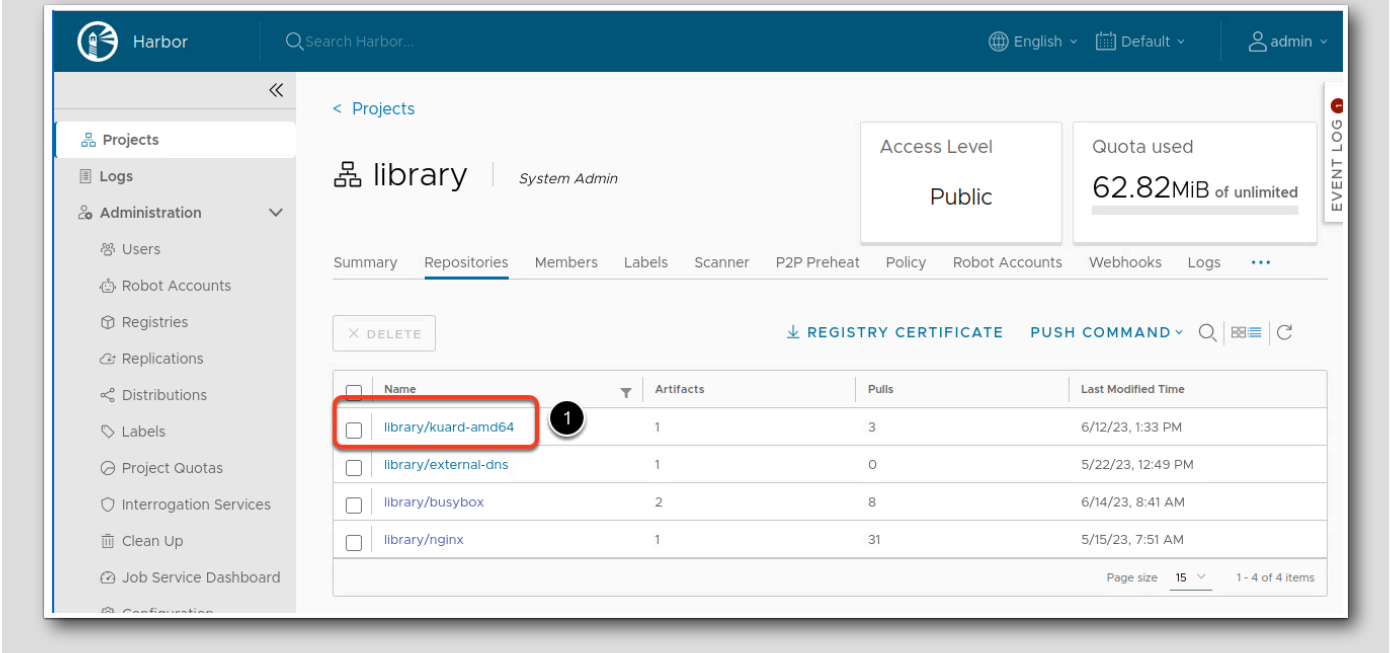
Let's examine the steps to scan an image for Vulnerabilities.

1. Click on the Harbor lighthouse at the left hand top of the screen.

Scanning For Vulnerabilities - Step 2



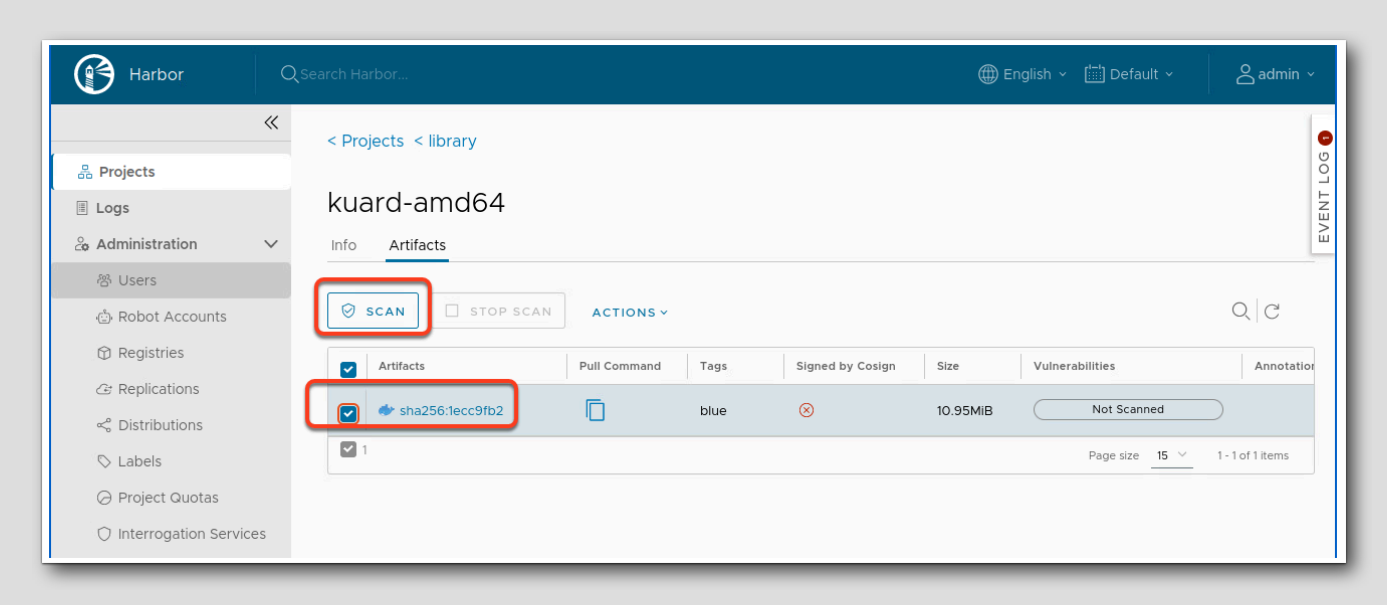
1. Click on library.



1. Click on the library/kuard-amd64 link.

Scanning For Vulnerabilities - Step 3

[185]



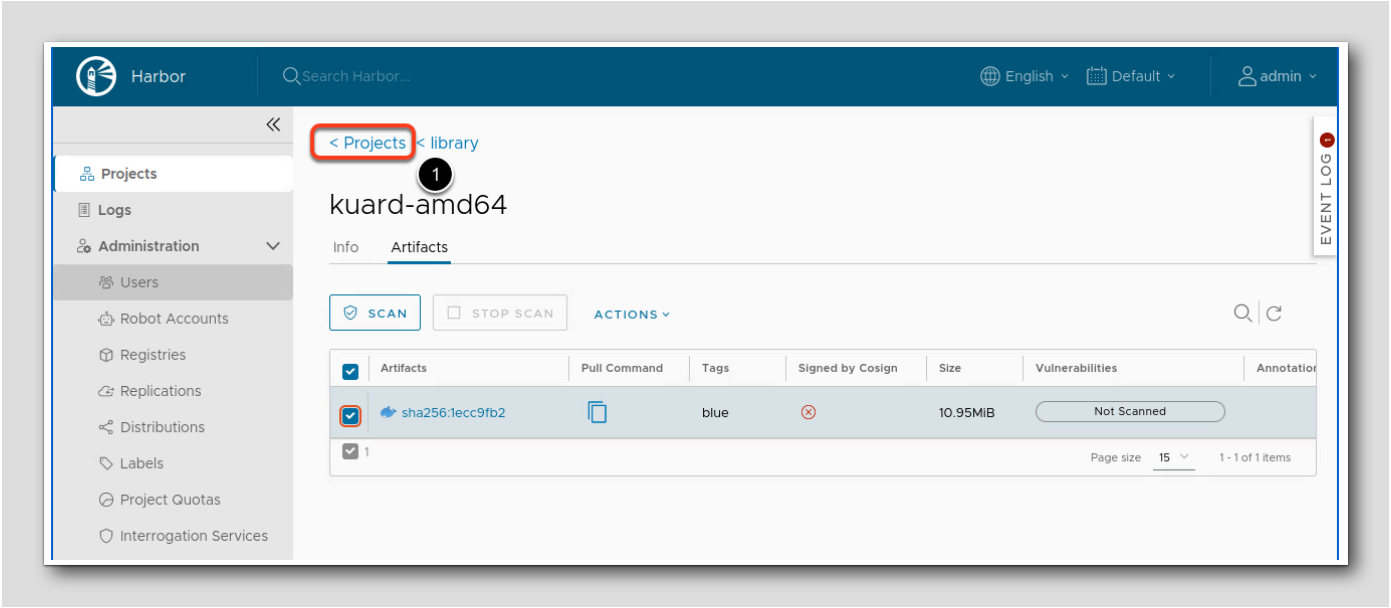
1. You can then scan a image for vulnerabilities by selecting the image
2. and clicking **Scan**

Note that the option to scan is greyed out if no image is selected or scanner is installed. Images can be scanned manually, automatically (daily), or scanned when they are pushed (through a project-level policy). You can also set policies on a per-project basis to automatically scan images when they are pushed to the registry. Similarly, you can prevent vulnerable images from being pulled from the registry based on the severity level of the vulnerability.

You can optionally click scan to initiate an image scan note that the scan takes quite a bit of time.

Selecting A Scanned Image To Review

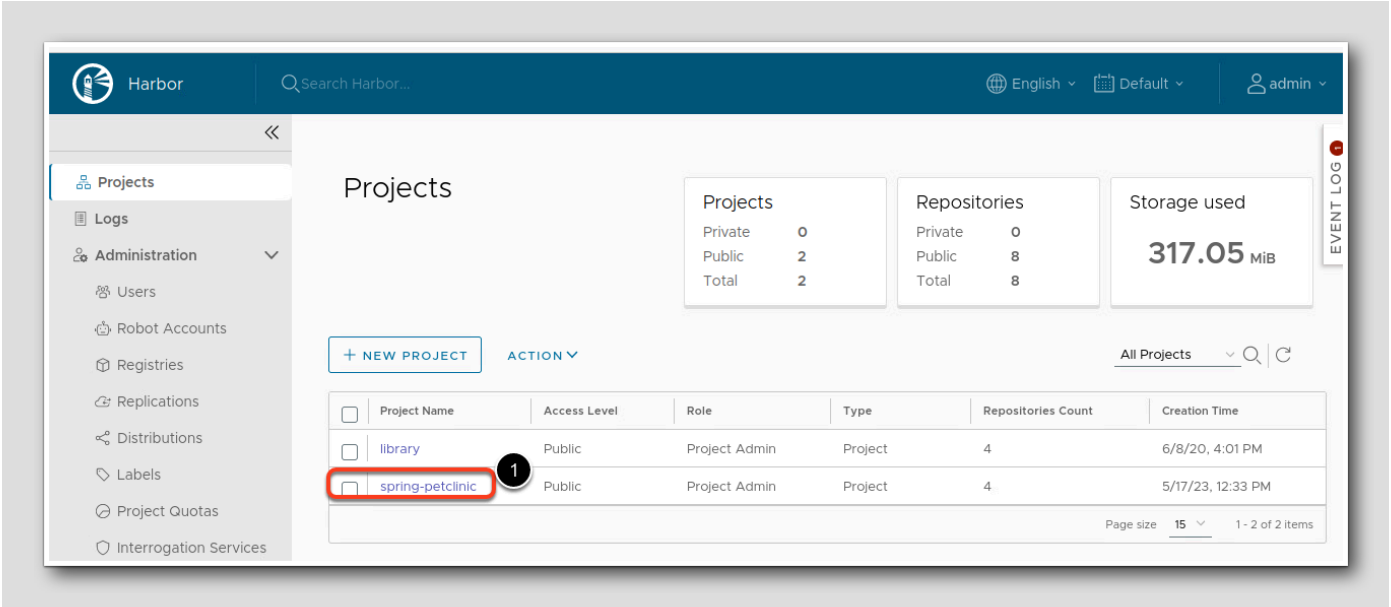
[186]



Let's examine the result from a previously scanned image.

1. Click the Projects link in Projects<library to return to the Projects screen.

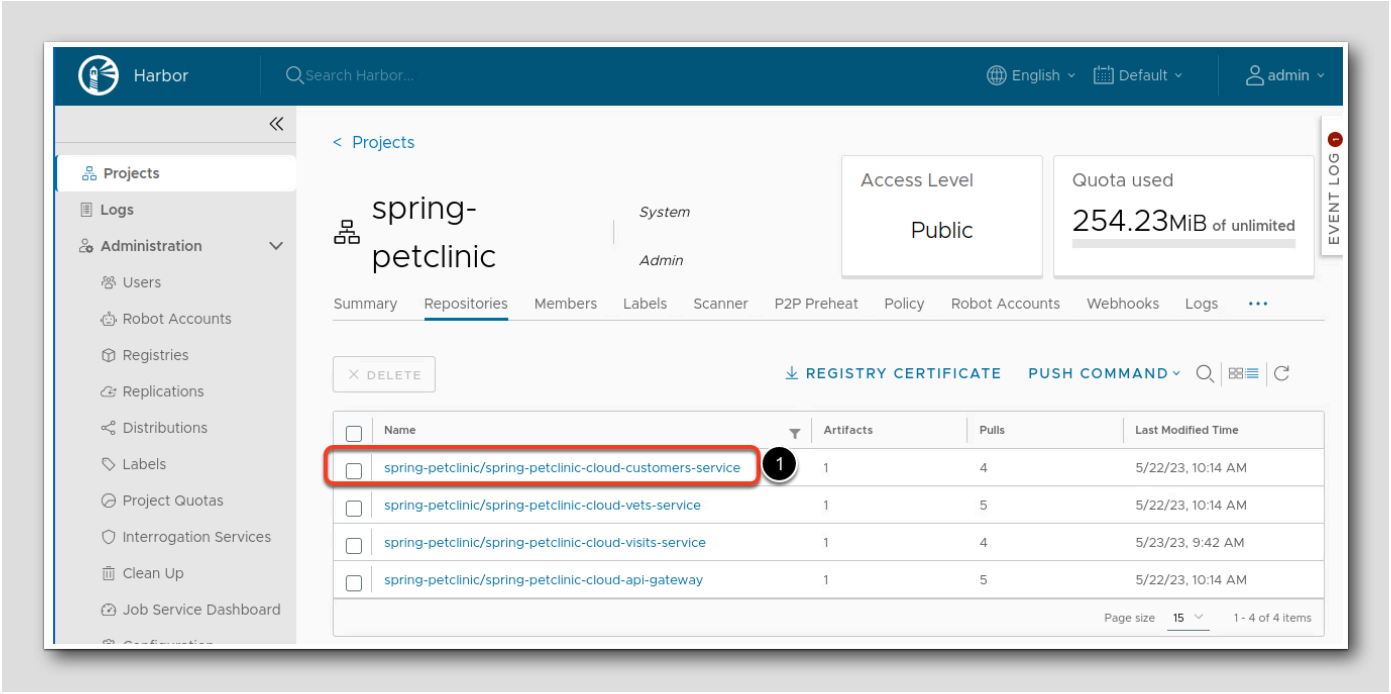
Do not click library, click the first Projects link.



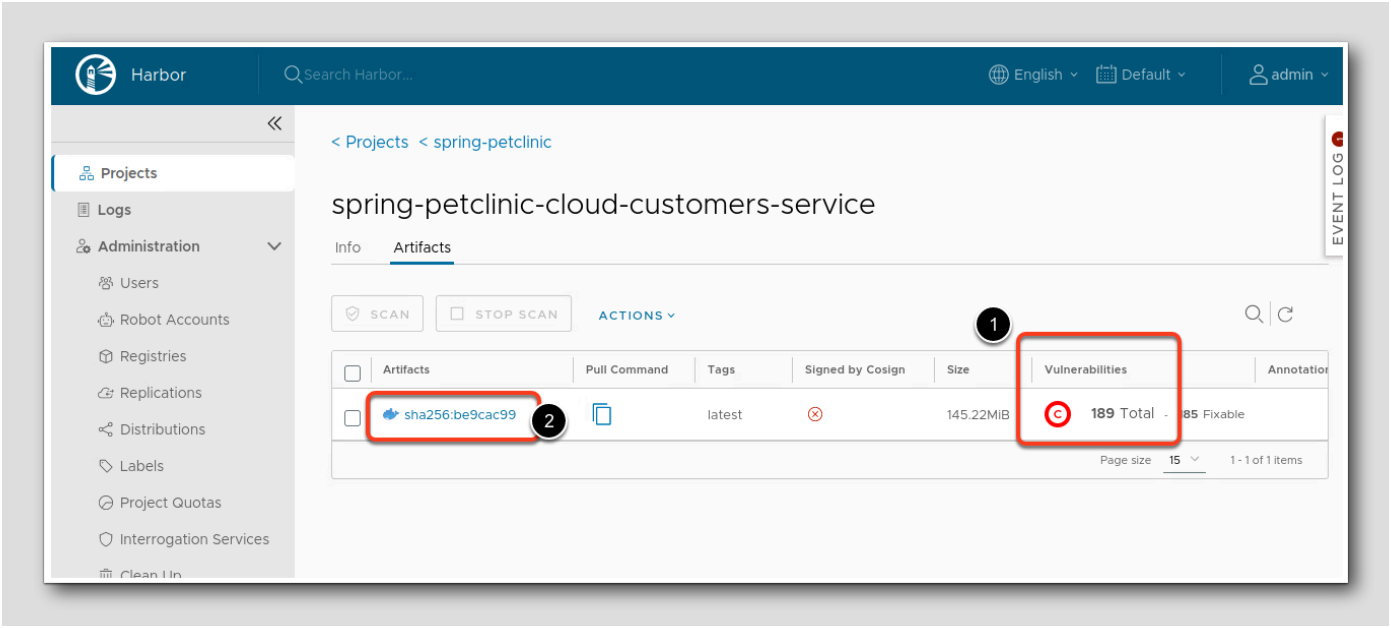
1. Click spring-petclinic.

Examining A Scanned Image

[187]

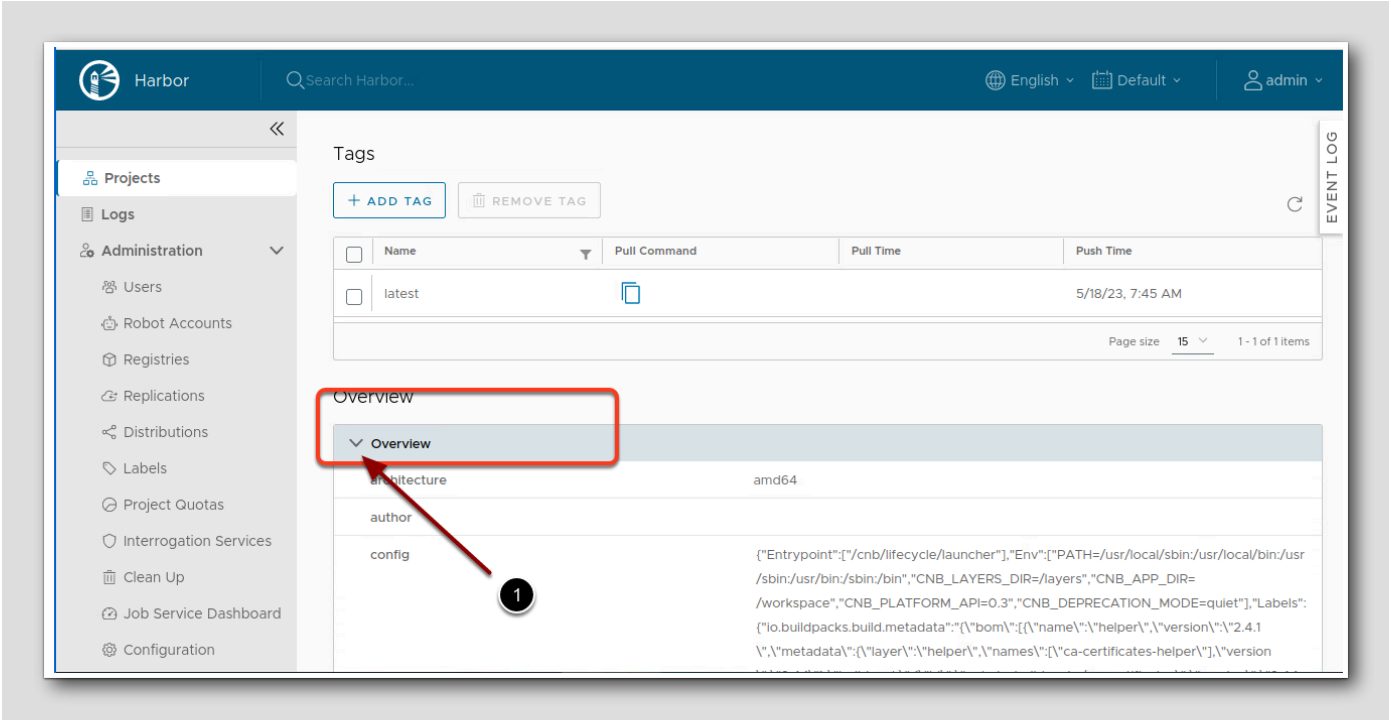


1. Select the spring-petclinic-cloud-customers-service image.

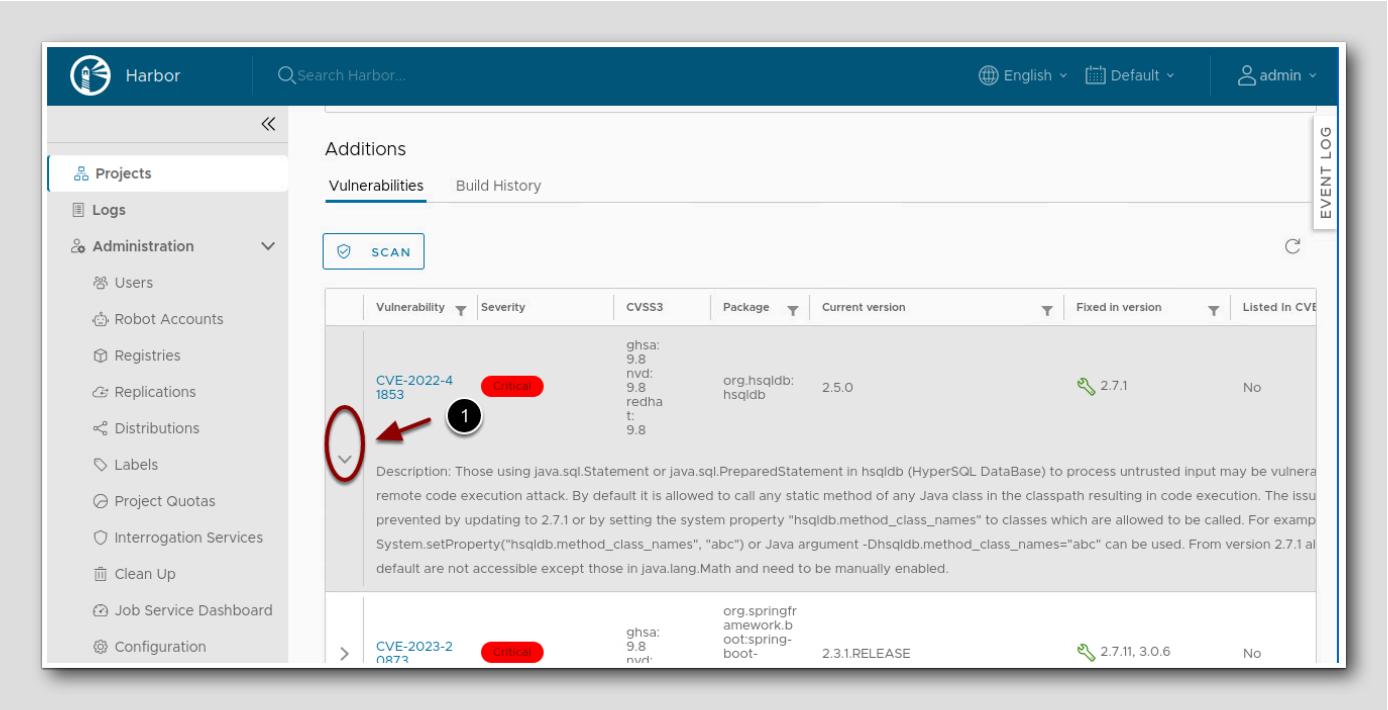


1. You will see that 189 Vulnerabilities were found. If you hover your mouse over the "C" in red you will see that some were "critical" . If you hover your mouse over the "189" you will see a snapshot of the list.
2. Click the artifact If you want to see a detailed list of the vulnerabilities.

Examining The Vulnerabilities In Detail



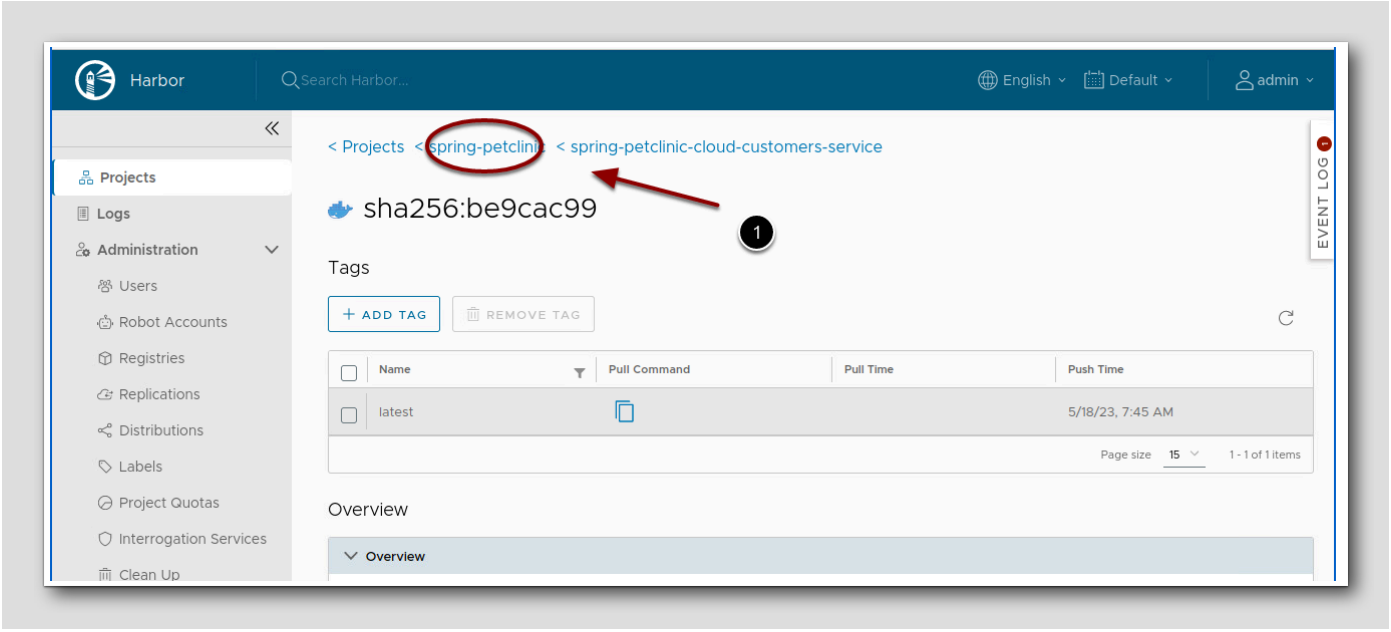
Collapse the Overview by clicking the down arrow. This will make it easier to view the vulnerabilities list.



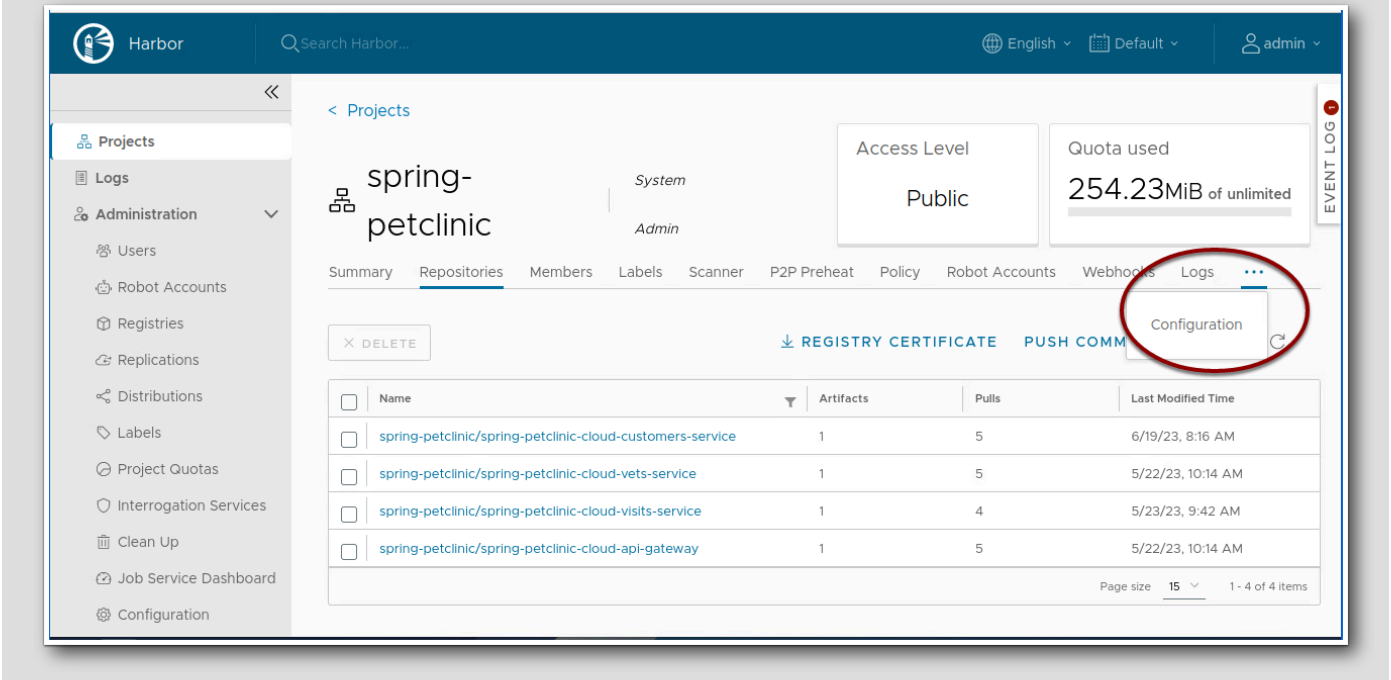
1. Click the ">" to show the specific details of the vulnerability found.

You can examine the vulnerabilities in detal.

How to Enable Harbor Deployment Security - Harbor Content Trust



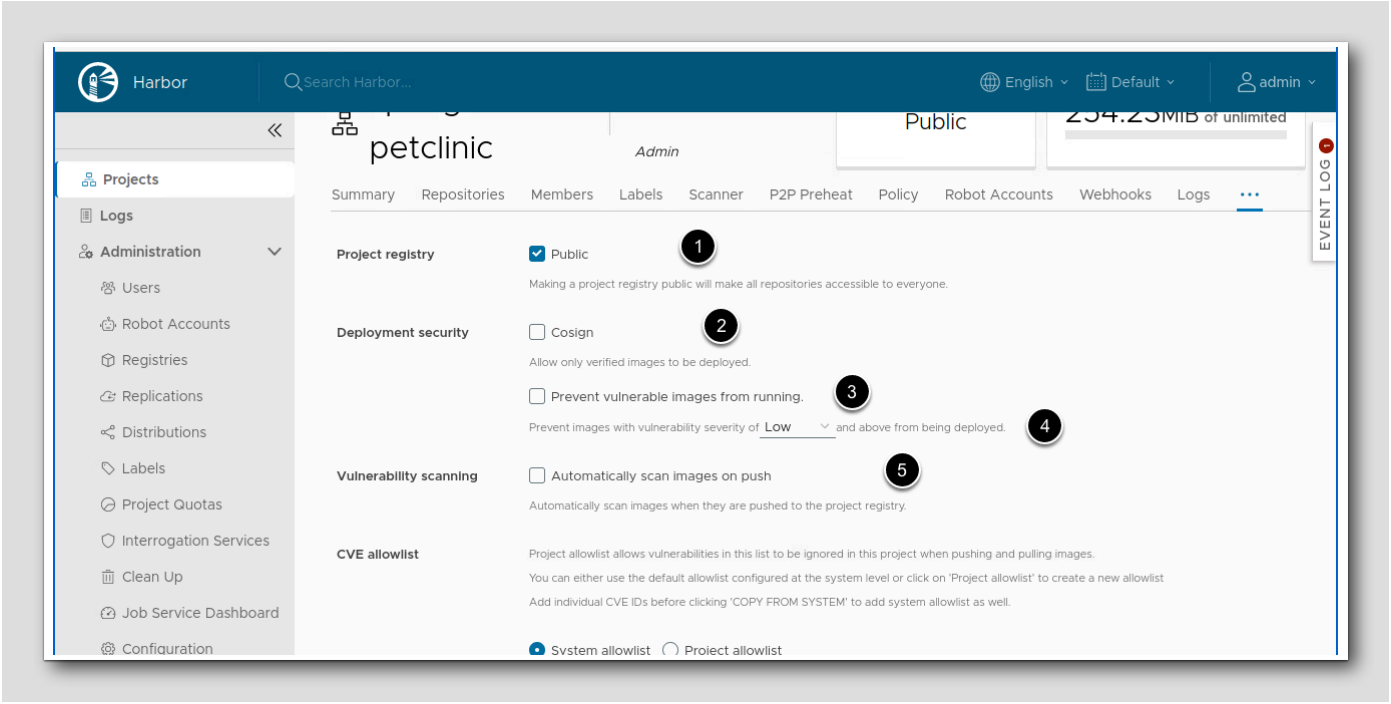
1. Click "spring-petclinic" to return to the spring-petclinic project page.



1. Click the 3 dots to the right, then Configuration.

Configure Harbor to Automatically Scan Images

[190]

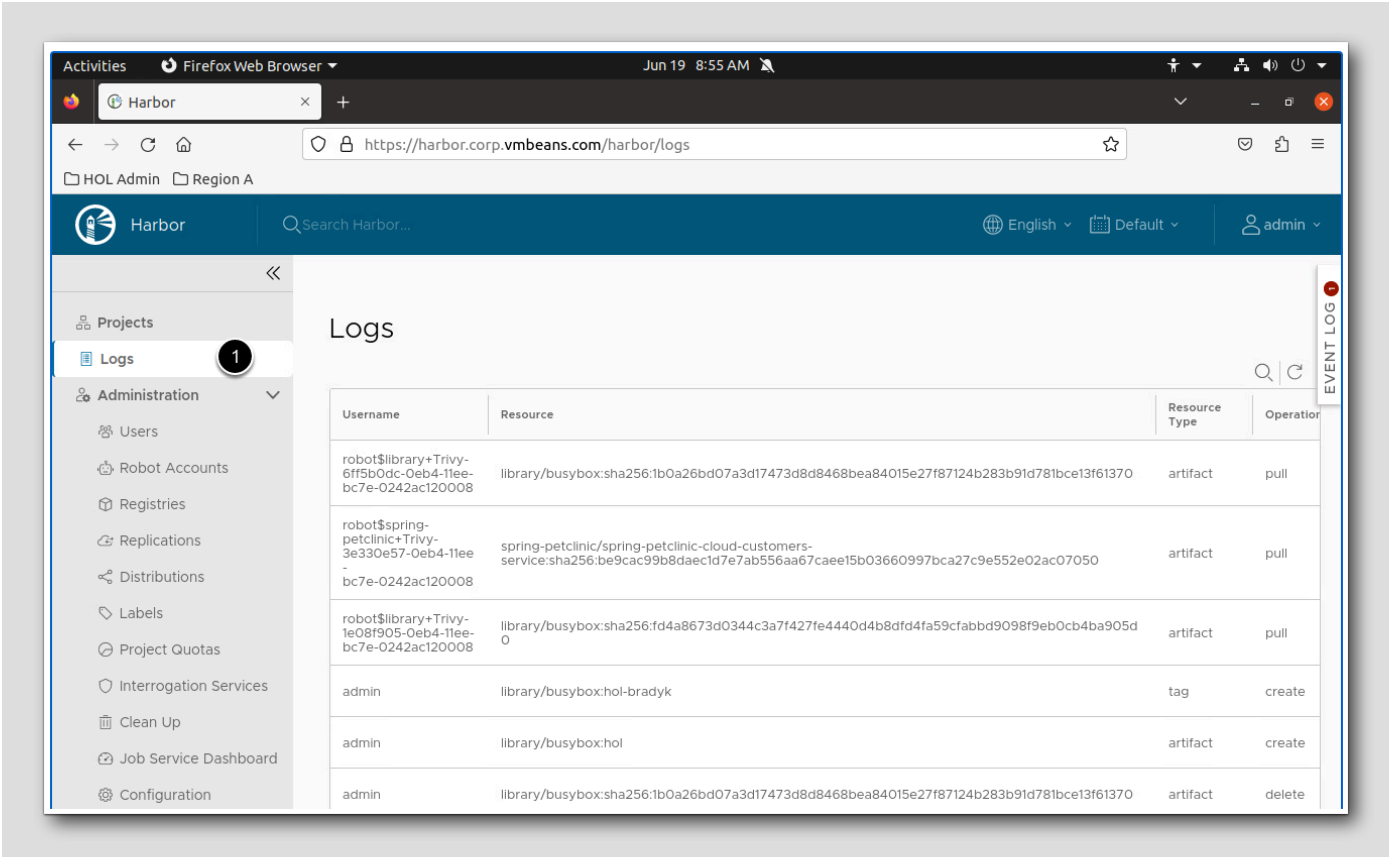


1. A "public" registry allows any user to pull the image from harbor.
2. Set "Cosign" to only allow signed images.
3. Set "Prevent vulnerable images from scanning" This prevents images with vulnerabilities from being pulled from Harbor.
4. You can select the threat level.
5. You can have Harbor scan images on push by default by clicking "Automatically scan images on push".

Harbor Audit Logging

[191]

For auditing purposes, Harbor logs all operations to the repositories including the user who performed them. This can be especially useful for regulatory compliance where an audit trail of user operations is required.



The image above shows the logs collected on Harbor.

1. Click **Logs** to display the logs for the Harbor instance installed for this lab.

This concludes a brief overview of some of the capabilities of Harbor. To examine Harbor in more detail see: *https://goharbor.io/*

The Harbor image scanner is publicly available from GitHub.

The next chapter will dive into working with images in Harbor.

Working with Harbor And Images

[192]

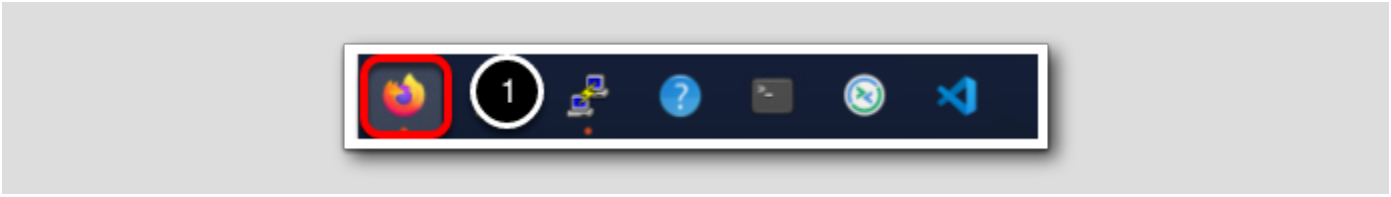
One of the most significant benefits of containers is that they allow you to package up an application with its dependencies and run it anywhere. This form of application packaging is called a container image. Docker images need to be stored in a secure and reliable location, a image registry. Harbor is an open source container image registry for Docker images.

In this lab, we'll explore how to use Harbor to store our Docker images and we'll update an application in Kubernetes.

Open Firefox

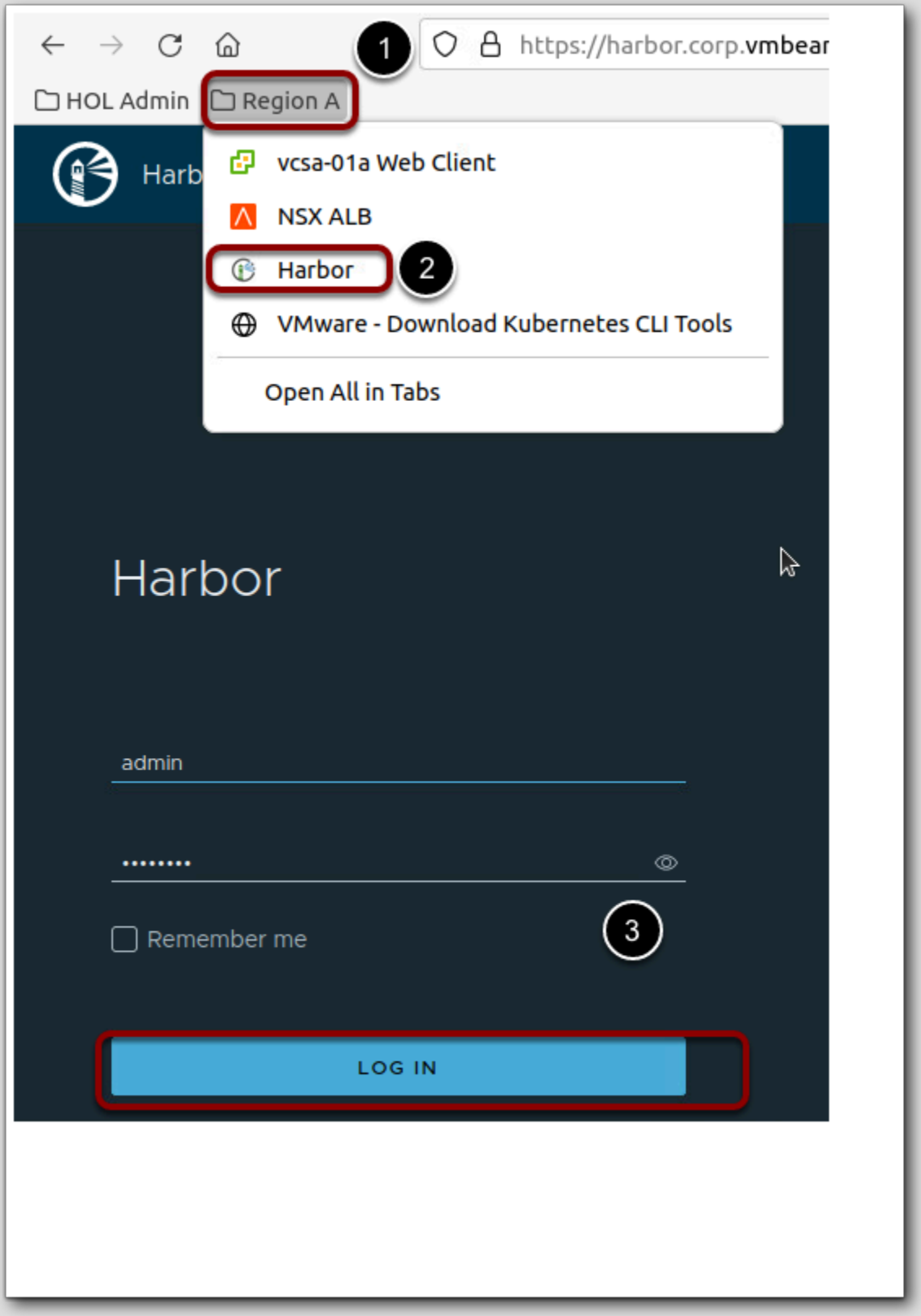
[193]

If you still have your web browser open and logged into Harbor you can skip the next few steps and proceed to Exploring Harbor.



1. Click on the Firefox Icon on the Quick Launch Task Bar.

Log in to Harbor


1
https://harbor.corp.vmbear
HOL Admin
Region A
vcsa-01a Web Client
NSX ALB
Harbor
2
VMware - Download Kubernetes CLI Tools
Open All in Tabs
Harbor
admin
Remember me
3
LOG IN

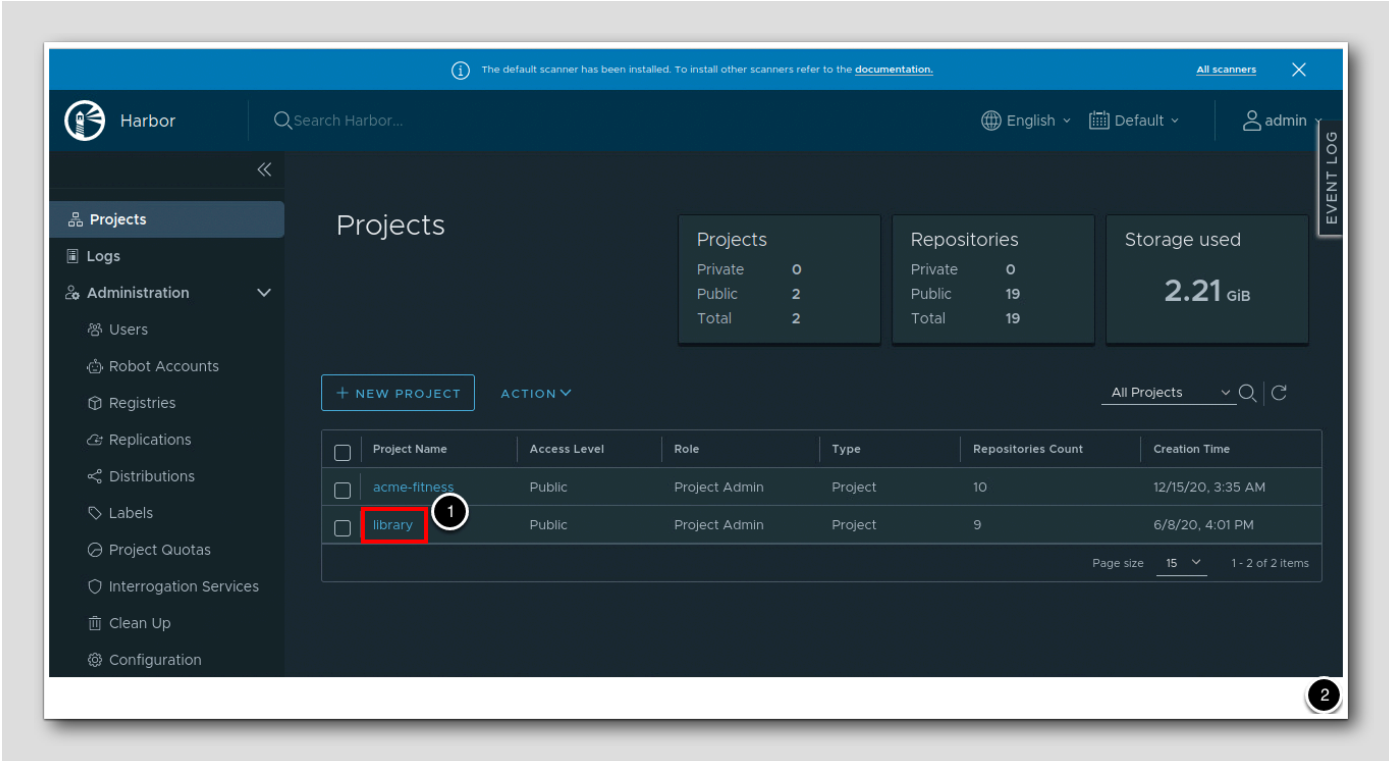
1. Click on the **Region A** Folder in the Bookmark toolbar.
2. Click on **Harbor** link in the Bookmark toolbar.
3. Click the **Login** Button. The credentials should be saved for you.

If credentials aren't saved, use the following:

- username: admin
- password: VMware1!

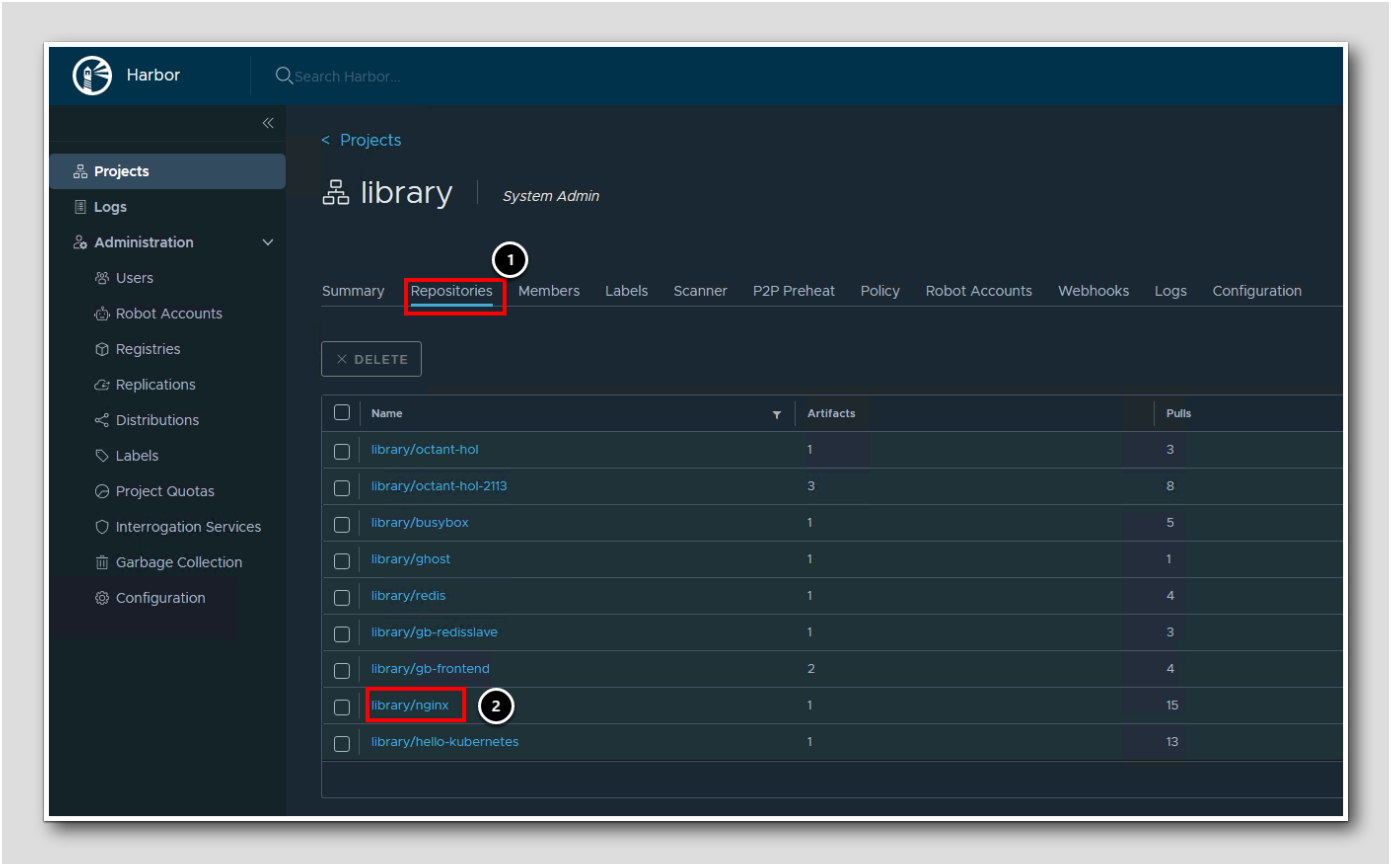
Exploring Harbor

[195]



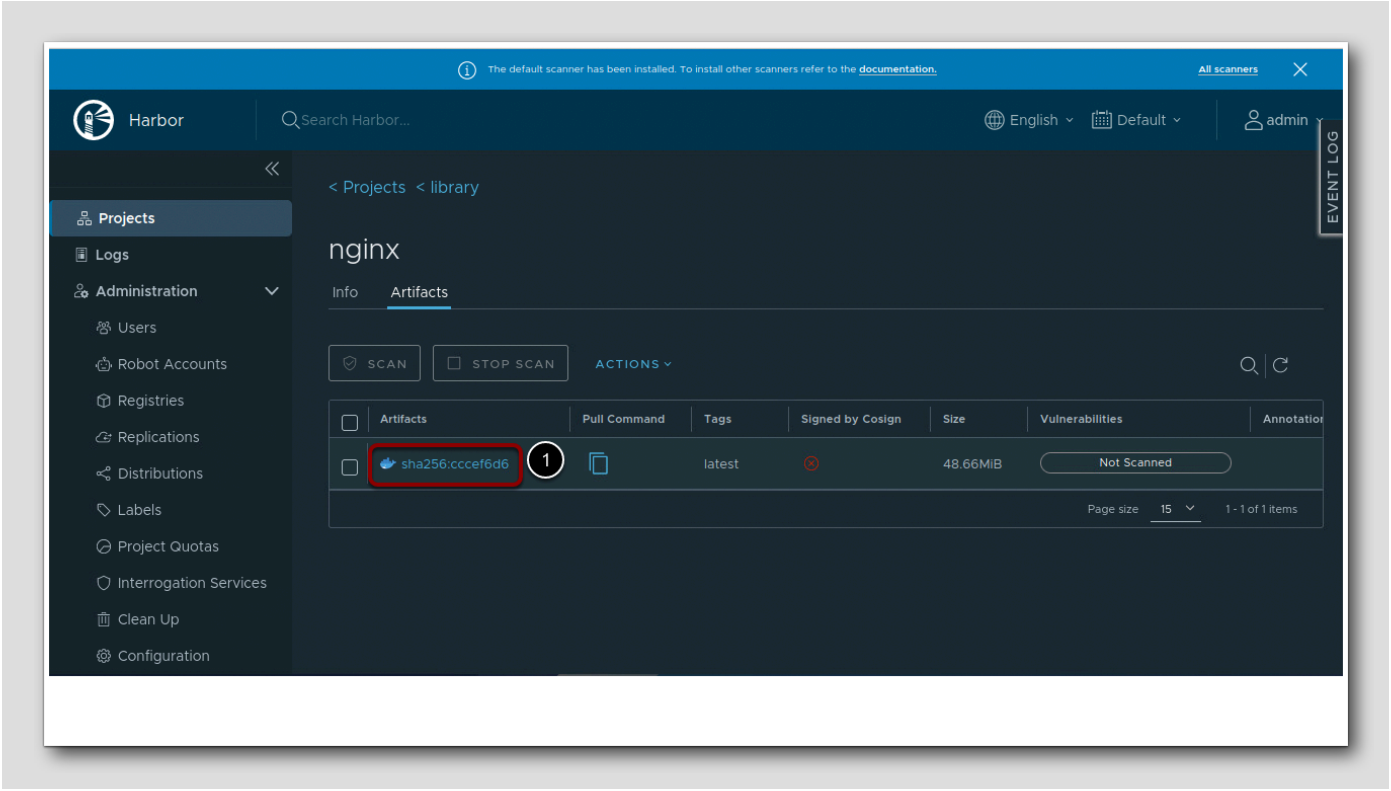
You'll notice there is a project called library:

1. Click on **library**

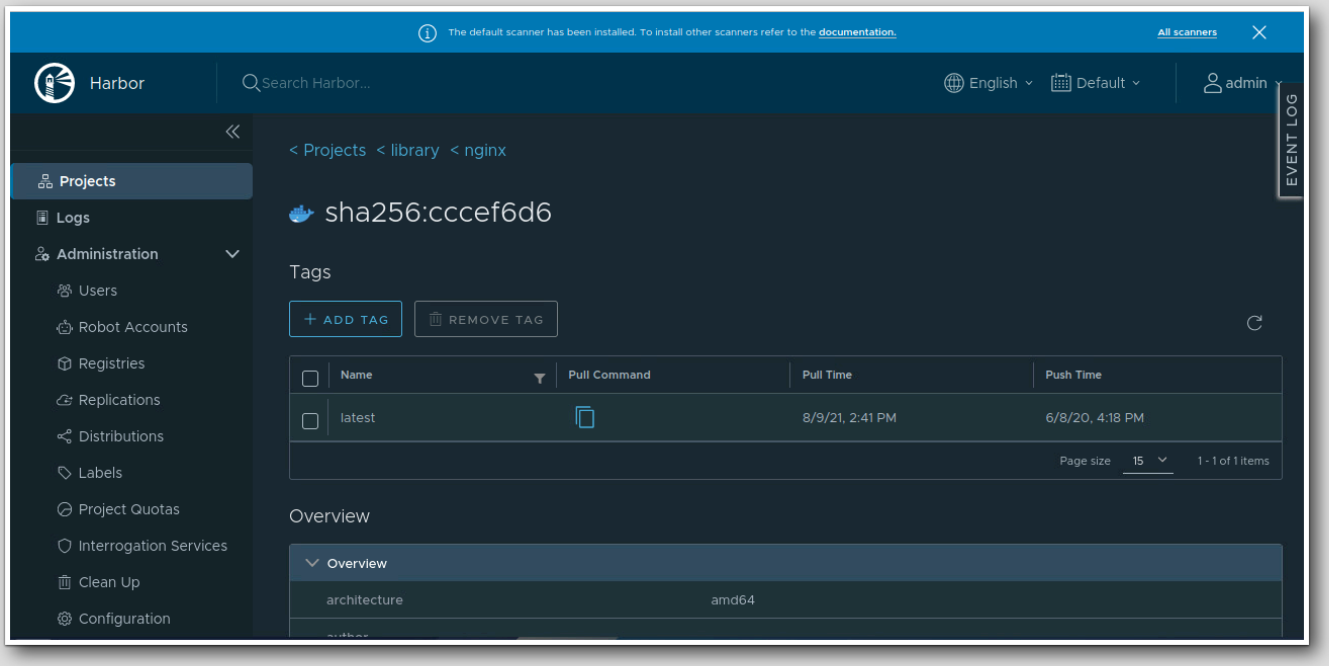


1. Click on **Repositories**
2. Click on **library/nginx**

Now, the tags for the NGINX image will be displayed. We only have one tag "latest".



1. Click on the image hash, beginning by sha256, to see more information about the image:



Creating a new Docker image from an existing one

[196]

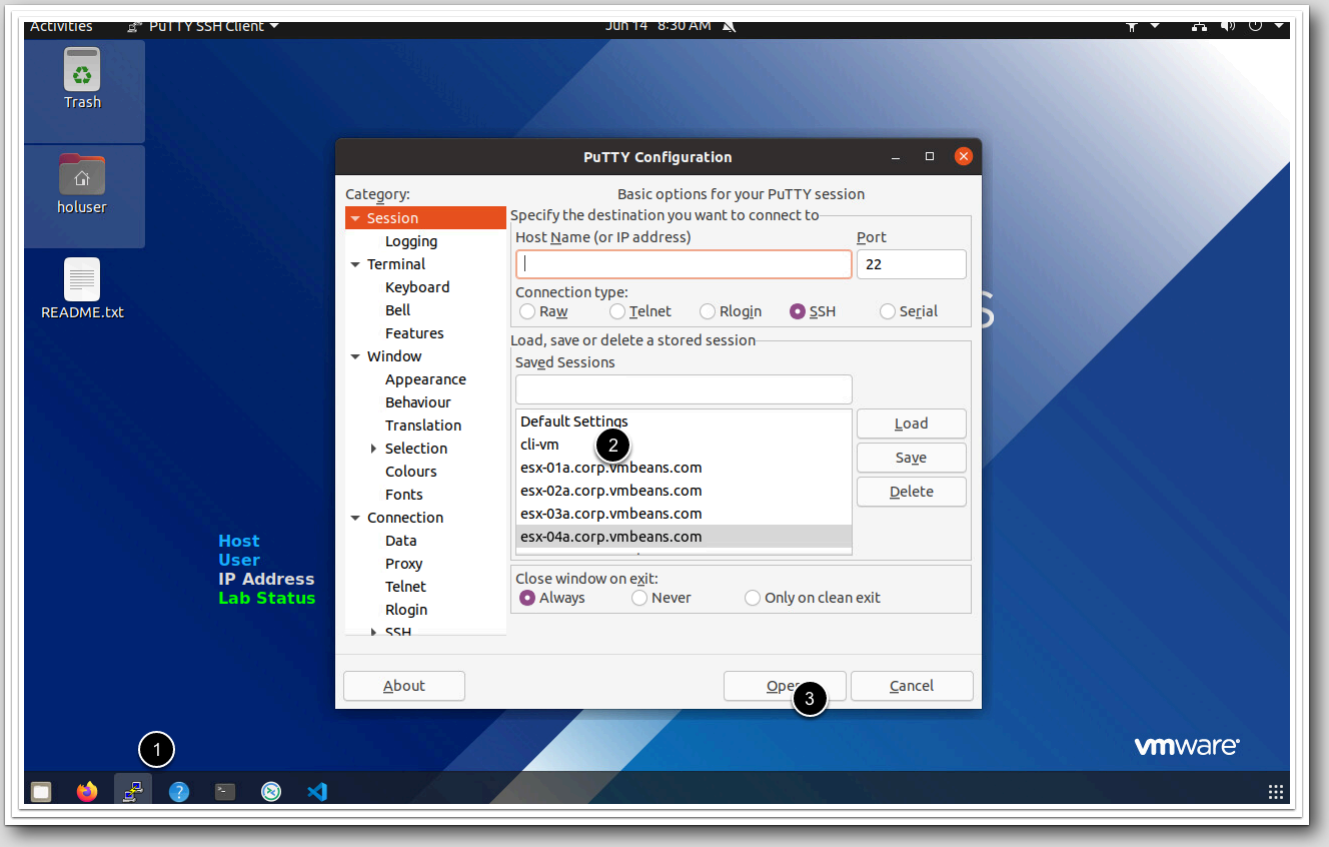
Now we will create a Docker image. All Docker images are created from a file called a Dockerfile. This Dockerfile defines how the image will be built.

Docker images are built using layers. This means that you can cache commonly-used layers so that you only need to download or upload the layers that you do not already have.

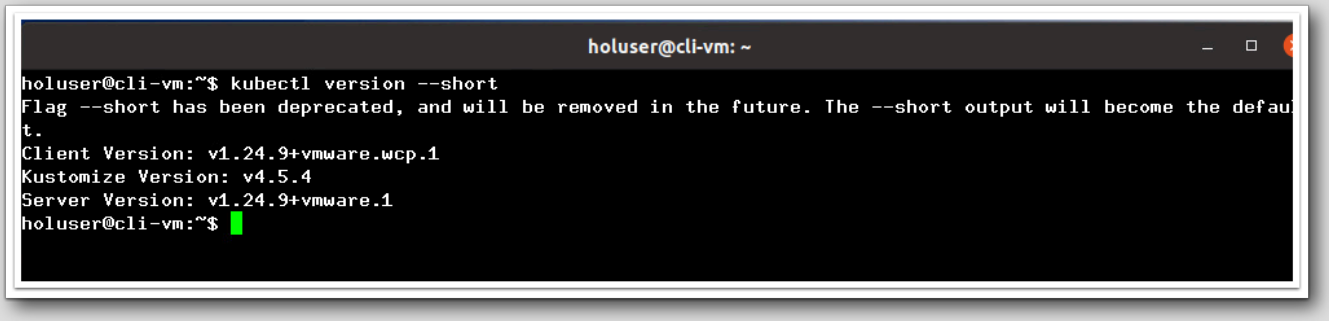
We'll use the existing NGINX image and we'll modify the index.html page to display a change.

If you still have your PuTTY window open you can skip the next steps.

1. Click on the **PuTTY** icon in the taskbar to open a new SSH session.



1. On the desktop - double click on the **Putty** icon.
2. Select 'cli-vm' from saved sessions
3. Click **Open**



This lab is deployed on vSphere 8 and is using Tanzu Kubernetes Grid (TKG)

Before we access our kubernetes environment we must login to the Tanzu Kubernetes cluster. Cluster authentication is integrated with vSphere Single Sign On through the vSphere Kubernetes Plugin. This plugin is already installed for you.

If you have already logged into the supervisor cluster previously in this lab you can skip this step.

Log in to Supervisor Cluster:

1. Type or copy/paste the following command into Putty: `kubectl vsphere login --server=https://192.168.130.11 -u administrator@vsphere.local --tanzu-kubernetes-cluster-namespace=ecom01-stage --tanzu-kubernetes-cluster-name tkg-cluster01 --insecure-skip-tls-verify`
2. You will be prompted for a password, type or copy/paste the password: VMware1!

Now navigate to ~/labs/nginx and create a new file called index.html. This will be our new front page in our NGINX web serve

If you examine it you'll see it's a very simple text file.

1. `cd ~/labs/nginx`

Now, copy/paste or drag and drop the entire command line below into your PuTTY window.

Note you need to copy the entire command line from the cat to the EOF ..

After you copy the line you must press the enter key to return to the command prompt.

```
cat << 'EOF' >> ~/labs/nginx/index.html

Welcome to VMware Hands-on-Labs!  If you see this page, you have modified correctly the base NGINX i
Congratulations!

EOF
```

Add screenshot

Next we will create a very simple Dockerfile. If you examine it you will see it only has 2 lines. The "FROM" line is telling docker to pull the nginx image with the label "latest" from our local Harbor repository. The "COPY" line is telling docker to take the file "index.html" and copy that file to the "/usr/share/nginx/html" directory for the image it is pulling.

Docker will then run that image.

1. Type or copy/paste the following command into PuTTY:

Note you need to copy the entire command line from the cat to the EOF ..

After you copy the line you must press the enter key to return to the command prompt.

```
cat << 'EOF' >> ~/labs/nginx/Dockerfile
FROM harbor.corp.vmbeans.com/library/nginx:latest
COPY index.html /usr/share/nginx/html
EOF
```

add screenshot

In the first line, we are telling Docker to start building our new image from the existing nginx image in the Harbor registry.

In the second line, we are telling Docker to copy the index.html file we just generated in the NGINX www folder.

Build the image

[197]

First of all, we need to log in to Harbor from the CLI:

1. Type or copy/paste the following command into PuTTY: `docker login harbor.corp.vmbeans.com`
2. Use **admin** as your username
3. Use **VMware1!** as your password

We can now build and tag our image. The Docker images need to be tagged with the following format to be stored in Harbor registry_url/project/image:tag

1. Type or copy/paste the following command into PuTTY: `docker build . -t harbor.corp.vmbeans.com/library/nginx:new`

We can build and tag our image with a single command using the -t flag.

```
root@linux-01a:~/labs/nginx# docker build . -t harbor.corp.vmbeans.com/library/nginx:new
Sending build context to Docker daemon  4.096kB
Step 1/2 : FROM harbor.corp.vmbeans.com/library/nginx:latest
latest: Pulling from library/nginx
Digest: sha256:cccef6d6bdea671c394956e24b0d0c44cd82dbe83f543a47fdc790fadea48422
Status: Downloaded newer image for harbor.corp.vmbeans.com/library/nginx:latest
 ---> 602e111c06b6
Step 2/2 : COPY index.html /usr/share/nginx/html
 ---> c36032e2f112
Successfully built c36032e2f112
Successfully tagged harbor.corp.vmbeans.com/library/nginx:new
```

We are now ready to upload the new image to Harbor. You can do so with this command:

1. Type or copy/paste the following command into PuTTY: `docker push harbor.corp.vmbeans.com/library/nginx:new`

```
root@linux-01a:~/labs/nginx# docker push harbor.corp.vmbeans.com/library/nginx:new
The push refers to repository [harbor.corp.vmbeans.com/library/nginx]
e37f6e03e586: Pushed
b3003aac411c: Layer already exists
216cf33c0a28: Layer already exists
c2adabaecedb: Layer already exists
new: digest: sha256:94c4e58ce5ecef570d1c16c76de2f34eff4957d56875171f325e77630f8f7080 size: 1155
```

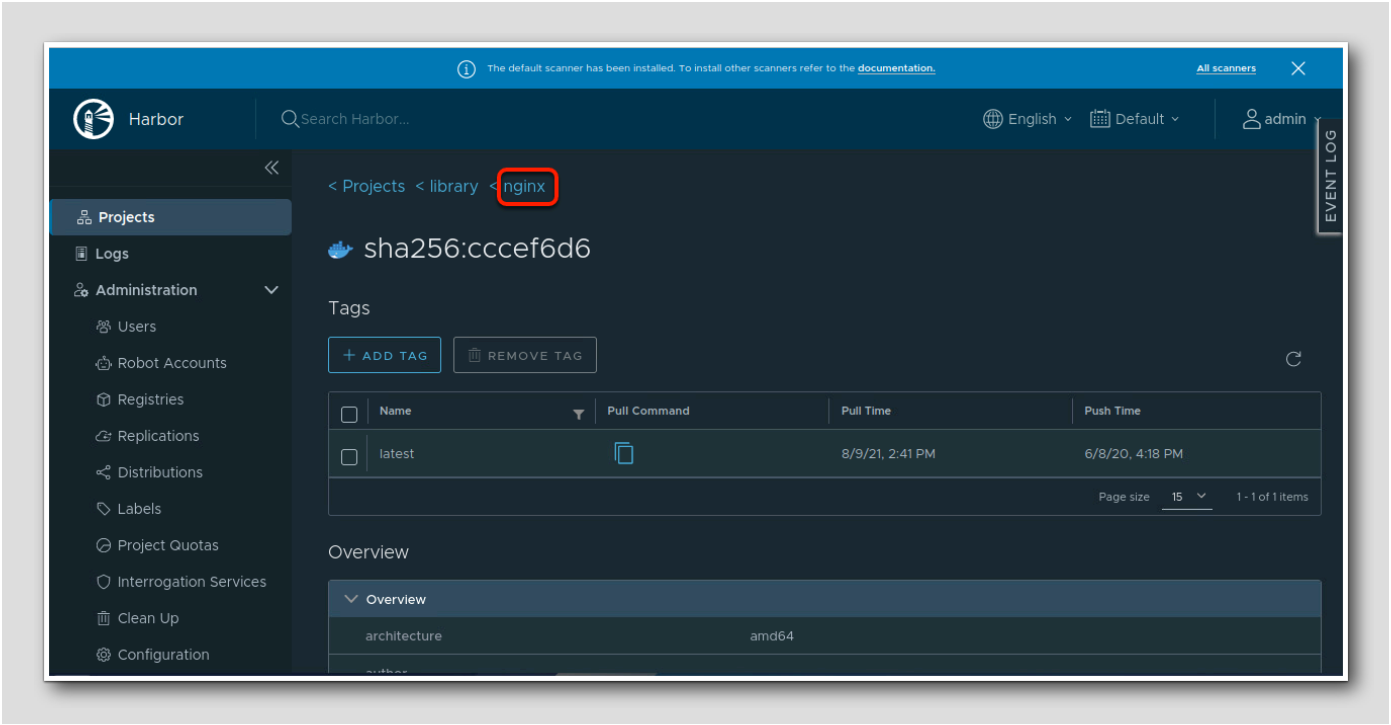
Docker images are built using layers. Therefore, Docker didn't need to upload the full image again to Harbor. Docker just pushed the layer with the change we made to the base image

View image in Harbor Web UI

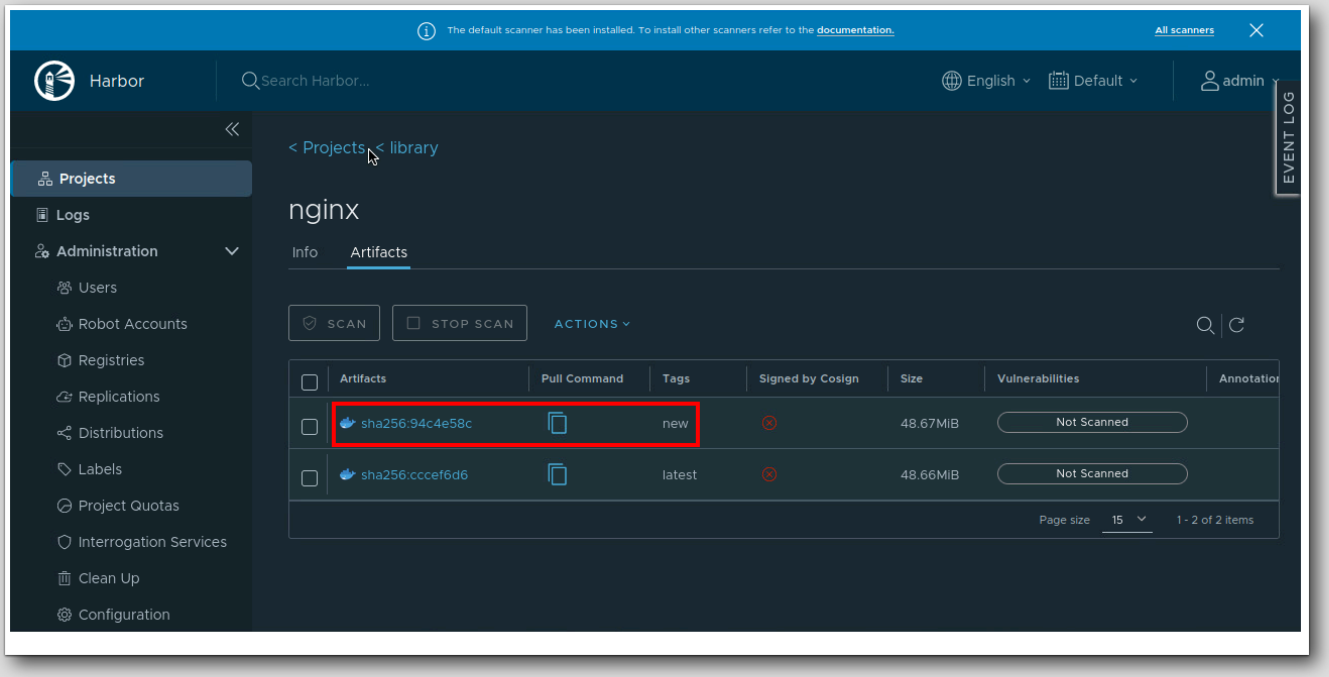
[198]

If you go back to the Harbor Web UI and back to the NGINX project, you'll notice the new tag:

1. Click on nginx to return to the project view.



2. A new image with the tag **"new"** has been created.



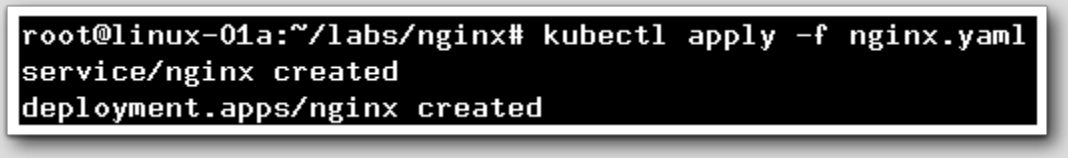
Deploying the NGINX application with the base image

[199]

We can now proceed and deploy the NGINX server to the cluster. We'll deploy the nginx.yaml file located in **~/labs/nginx** .

1. **Type or copy/paste** the following command into PuTTY: `kubectl apply -f nginx.yaml`

The application will now be deployed:



You can verify that the application is running:

1. **Type or copy/paste** the following command into PuTTY: `kubectl get pods`

```
root@linux-01a:~/labs/nginx# kubectl get pods
NAME                     READY   STATUS    RESTARTS   AGE
nginx-cfb48bc5-28cn5     1/1     Running   0          28s
nginx-cfb48bc5-fdpvf     1/1     Running   0          26s
nginx-cfb48bc5-nbwkw     1/1     Running   0          30s
```

You can get the application endpoint by using:

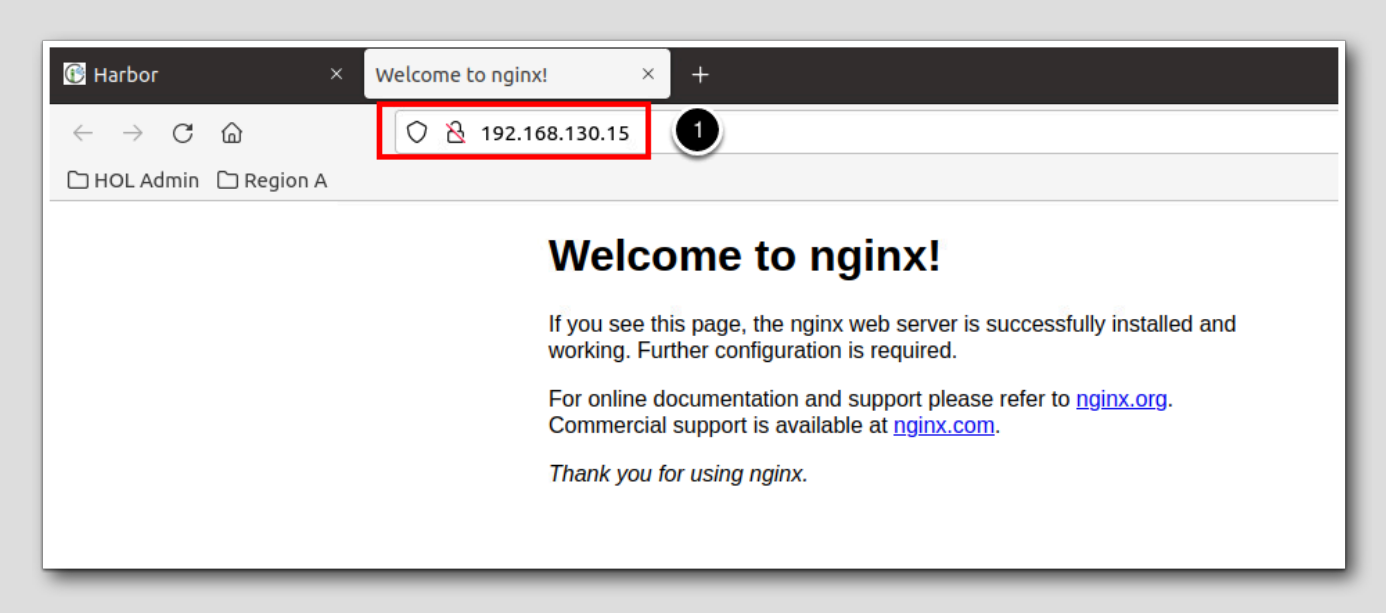
1. Type or copy/paste the following command into PuTTY: `kubectl get services`

```
root@linux-01a:~/labs/nginx# kubectl get services
NAME         TYPE           CLUSTER-IP       EXTERNAL-IP      PORT(S)        AGE
kubernetes   ClusterIP      198.48.0.1       <none>           443/TCP        9d
nginx        LoadBalancer   198.51.186.109   192.168.130.15   80:32515/TCP   22s
supervisor   ClusterIP      None             <none>           6443/TCP       9d
```

Access application using the web browser

[200]

You can now use your web browser to navigate to that address and see the NGINX welcome page. Please note that your External-IP Address may be different:



1. Open a new tab on your browser and type the External-IP endpoint from the previous step. In this case, it is: **192.168.130.15**

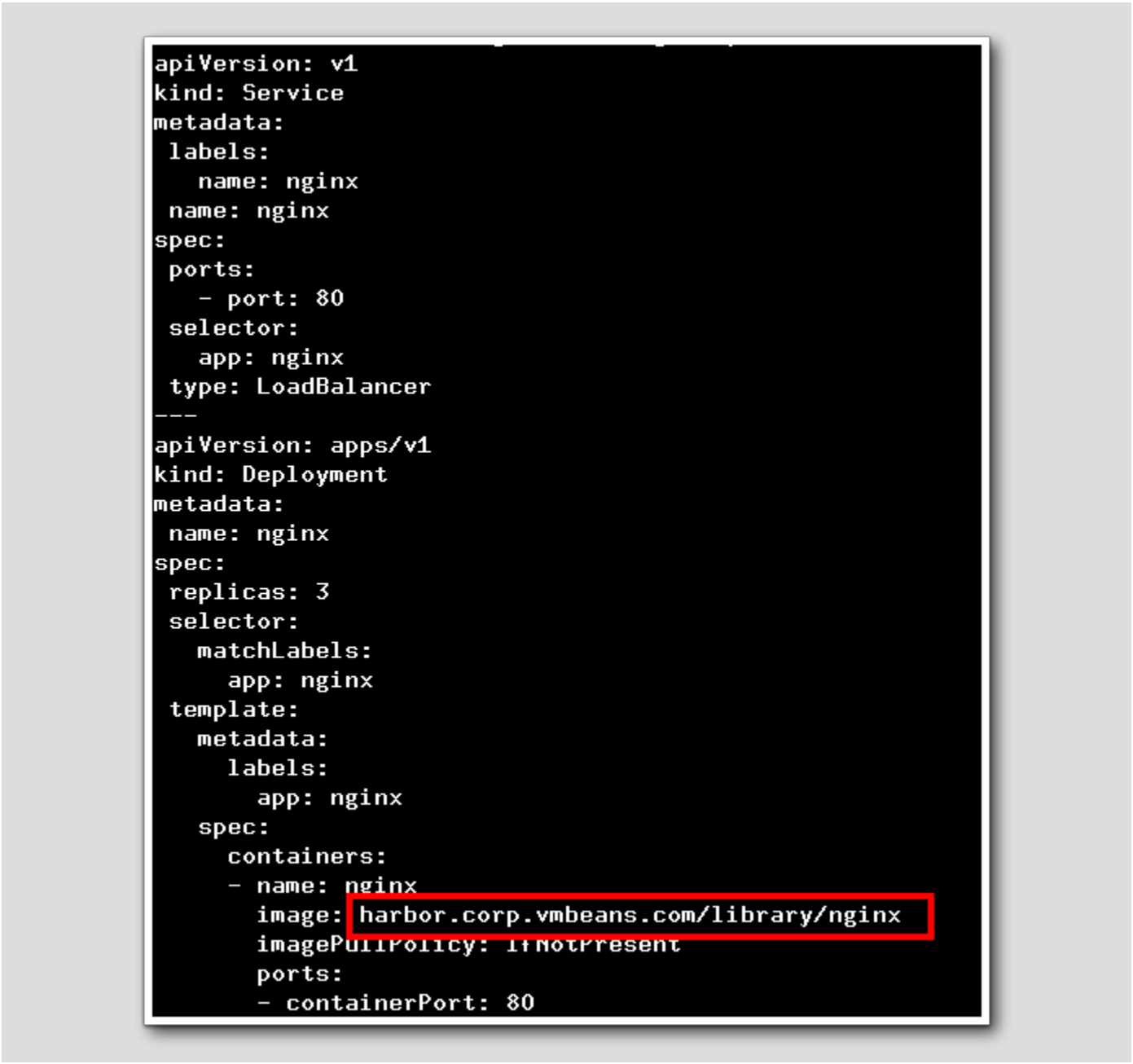
Changing the NGINX deployment with a new Docker image [201]

We are going to modify the existing NGINX deployment with our new image with its custom index.html page.

To do so, open the **nginx.yaml** file. You'll notice that the existing Deployment has harbor.corp.vmbeans.com/library/nginx as an image.

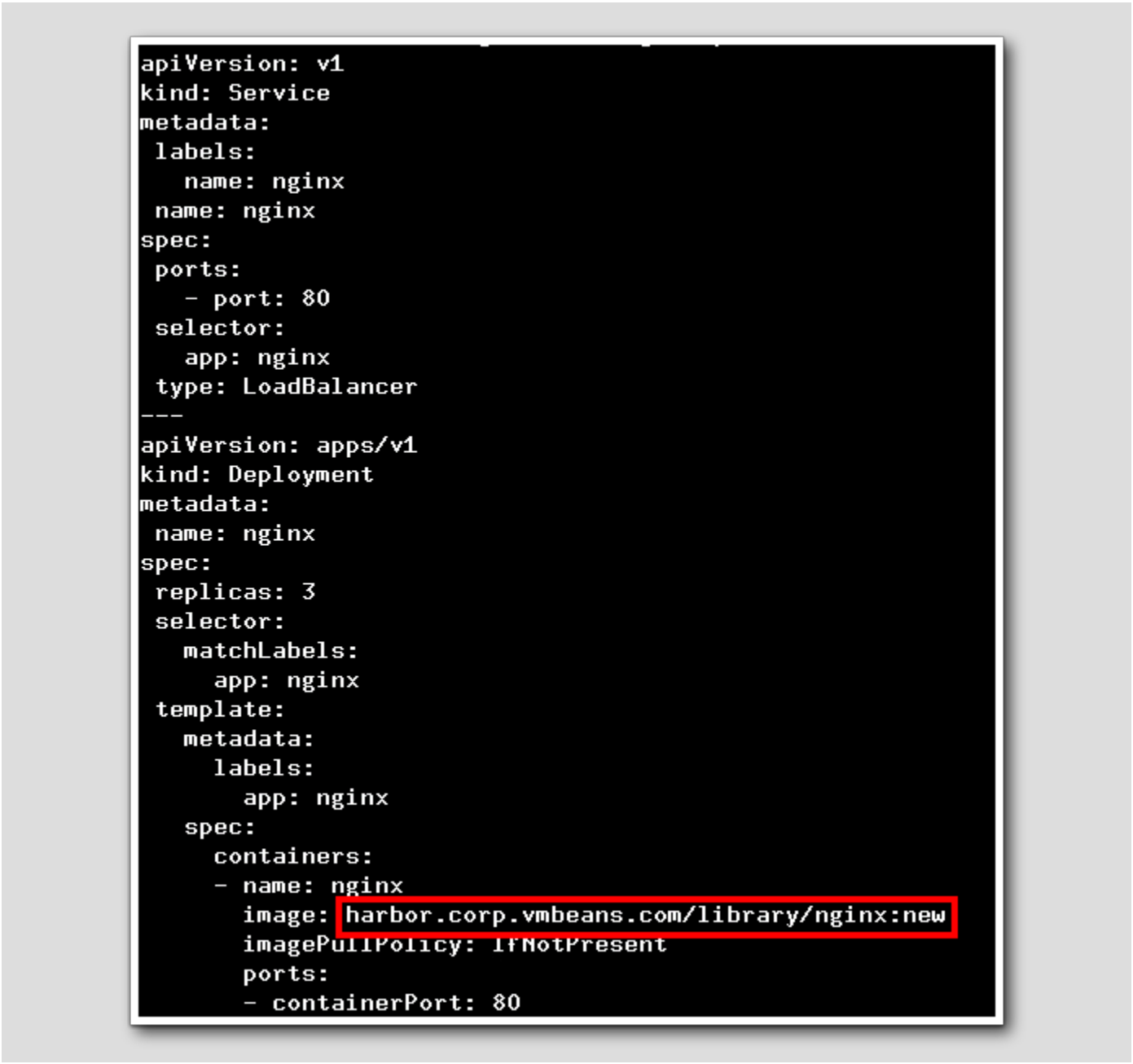
1. **Type or copy/paste** the following command into PuTTY: `cat ~/labs/nginx/nginx.yaml`

When no tag is specified, Docker will assume that it is the tag "latest"



Now, we'll modify the file to have the new tag.

1. Type or copy/paste the following command into PuTTY: `sed -i s@harbor.corp.vmbeans.com/library/nginx@harbor.corp.vmbeans.com/library/nginx:new@g ~/labs/nginx/nginx.yaml`
2. Type or copy/paste the following command into PuTTY to verify that the change is now applied: `cat ~/labs/nginx/nginx.yaml`



```
apiVersion: v1
kind: Service
metadata:
 labels:
   name: nginx
 name: nginx
spec:
 ports:
   - port: 80
 selector:
   app: nginx
 type: LoadBalancer
---
apiVersion: apps/v1
kind: Deployment
metadata:
 name: nginx
spec:
 replicas: 3
 selector:
   matchLabels:
     app: nginx
 template:
   metadata:
     labels:
       app: nginx
   spec:
     containers:
     - name: nginx
       image: harbor.corp.vmbeans.com/library/nginx:new
       imagePullPolicy: IfNotPresent
       ports:
       - containerPort: 80
```

This command added the label "new" to the nginx yaml file. Now we are able to update the Kubernetes deployment by running:

1. Type or copy/paste the following command into PuTTY: `kubectl apply -f nginx.yaml`

```
root@linux-01a:~/labs/nginx# kubectl apply -f nginx.yaml
service/nginx unchanged
deployment.apps/nginx configured
```

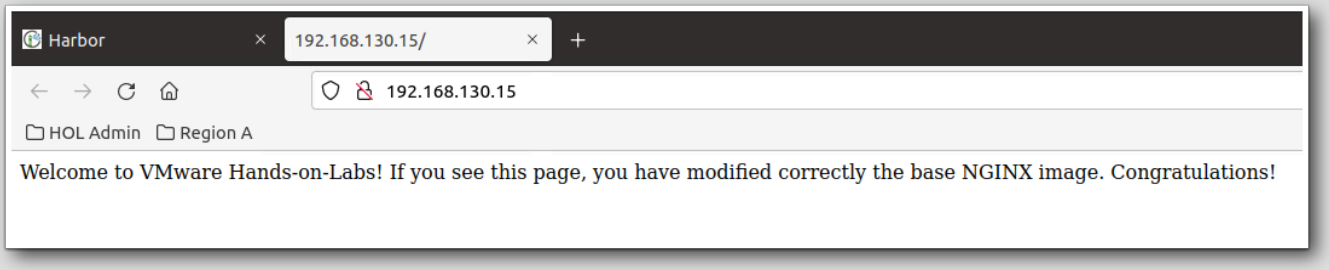
Kubernetes will notice that the only thing that changed was the Docker image. As such, it will only change the Deployment and not the Service.

If you get all the Pods, you'll notice that Kubernetes launched new Pods with the updated Docker image and removed the old ones (rolling update):

1. Type or copy/paste the following command into PuTTY: `kubectl get pods`

```
root@linux-01a:~/labs/nginx# kubectl get pods
NAME                     READY   STATUS    RESTARTS   AGE
nginx-cfb48bc5-28cn5     1/1     Running   0          28s
nginx-cfb48bc5-fdpvf     1/1     Running   0          26s
nginx-cfb48bc5-nbwkw     1/1     Running   0          30s
```

We can head back to our web browser and refresh the page. We should now have the updated index.html page.



Conclusion

[202]

Congratulations on completing Module 4. You should now have a solid foundational understanding of the Harbor container image registry.

VMware provides a host of Modern Apps FREE learning resources hosted on our *Tanzu website* as well as the *Modern Apps*

Community.

Please use the QR code for more information.

Congratulations you have finished Module 4

[203]

Based on your interests, please proceed to any module below:

- Module 1 - Introduction to Containers (15 minutes) (Basic) - In this module, we will explain containers and how they enable 3rd Platform application architectures to be run efficiently in distributed environments. We will also delve into the basics of Docker as a container platform. This is a reading module only without any interactive labs.
- Module 2 - A Quick Tour of Docker - (15 minutes) (Basic) In this module, we're getting hands on with docker and explore how to create simple containers and how to leverage docker container networking. Finally, we'll look at how you can build your own images using a Dockerfile.
- Module 3 - Introduction to Kubernetes - (30 minutes) (Basic) In this module we will explain the place where container management fits and take a high level walk thru the Kubernetes architecture. We will walk through some worked examples using a kubernetes instance deployed on Tanzu Kubernetes Grid (TKG).
- Module 5 - Introduction to Tech Zone VMware Learning Platform & Kube Academy - (15 minutes) (Basic) In this module we will introduce some of the many FREE learning resources which VMware provides for users looking to start their journey into modern apps and kubernetes or looking to build on the knowledge and experience they already have.

From here you can:

1. Click to advance to the next page and continue with the next lab module
2. Open the TABLE OF CONTENTS to jump to any module or lesson in this lab manual
3. End your lab and come back and start it again in the future

Module 5 - Introduction to Tanzu Tech Zone, VMware Learning Platform & Kube Academy (15 minutes) Basic

Introduction to TechZone

[205]

The new *Tanzu Tech Zone* is the fastest path to understanding, evaluating, and deploying VMware Tanzu solutions. Tech Zone's mission is to bring you insightful and accurate guidance on our current, released products.

The program is a initiative designed to address the unique needs of the IT and developer workforce as the market transitions towards increasingly automated and cloud native technologies.

Note: To appreciate the full experience of the VMware Tanzu Tech Zone you will need access the websites highlighted in this session from browser on a second device with access to the internet as some of the links in this Module and the various courses may not work from within the Hands On Labs (HoL) environment. For security the HoL environment has internet filtering enabled and therefore not all of the links in this module will work from within the Hands On Labs Environment.

What is VMware Tanzu Tech Zone

[206]

Tanzu Tech Zone is a technical content portal designed to make it easier than ever to find practical technical guidance for evaluating, implementing and operating VMware Tanzu solutions.

VMware Tanzu Tech Zone Overview

[207]

Practical, hands on experience and regular practice is key to becoming an effective practitioner in cloud native technologies. Tanzu Tech Zone focuses on hands on providing a fast path education, centered around real world workflows, tools and processes, and made real through actual contribution in open & open source community projects

The new Tanzu Tech Zone is the fastest path to understanding, evaluating, and deploying VMware Tanzu solutions. Tech Zone's mission is to bring you insightful and accurate guidance on our current, released products.

What kind of content is on the Tanzu Tech Zone?

[208]

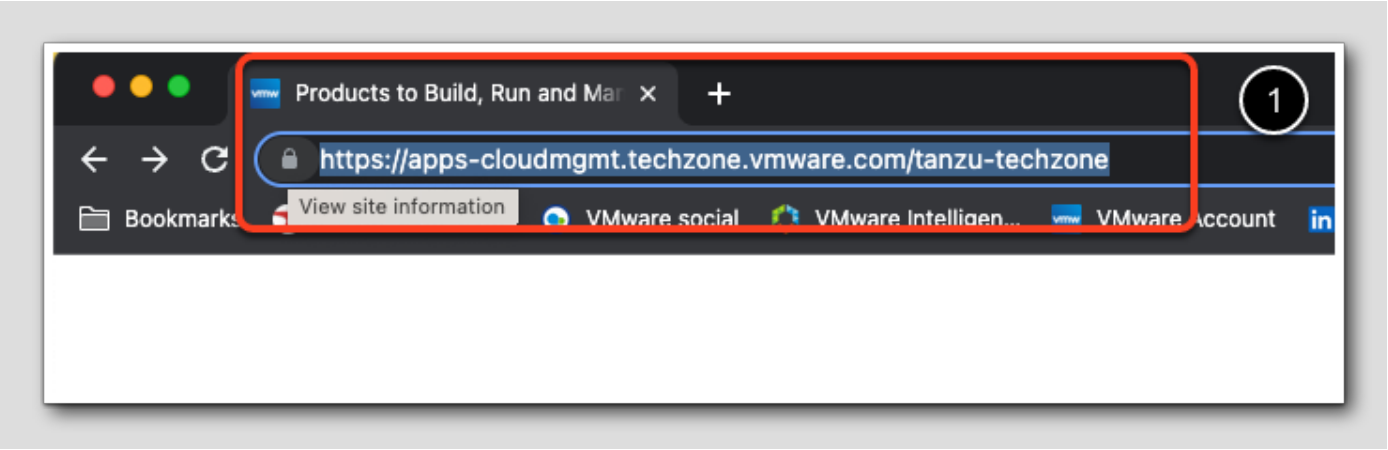
On the *Tanzu Tech Zone*, we've made it our mission to provide you with the resources you need, wherever you are in your modernization journey. Tech Zone content is, as the name implies, technical. The platform is managed by engineers and is delivered with a "by engineers, for engineers" approach to ensure content is practical and effective.

We start by providing introductory level, accelerated technical content, and continue with demos, presentations, videos, guides, and a full suite of content to help you accelerate the process of understanding and evaluating Tanzu solutions.

Beyond resources to understand and evaluate VMware Tanzu products, Tech Zone provides practical guidance and consolidated resources for product implementation and optimization to fully prepare DevOps teams with all the best practices and resources needed to deliver and manage Tanzu solutions throughout their complete operational lifecycle.

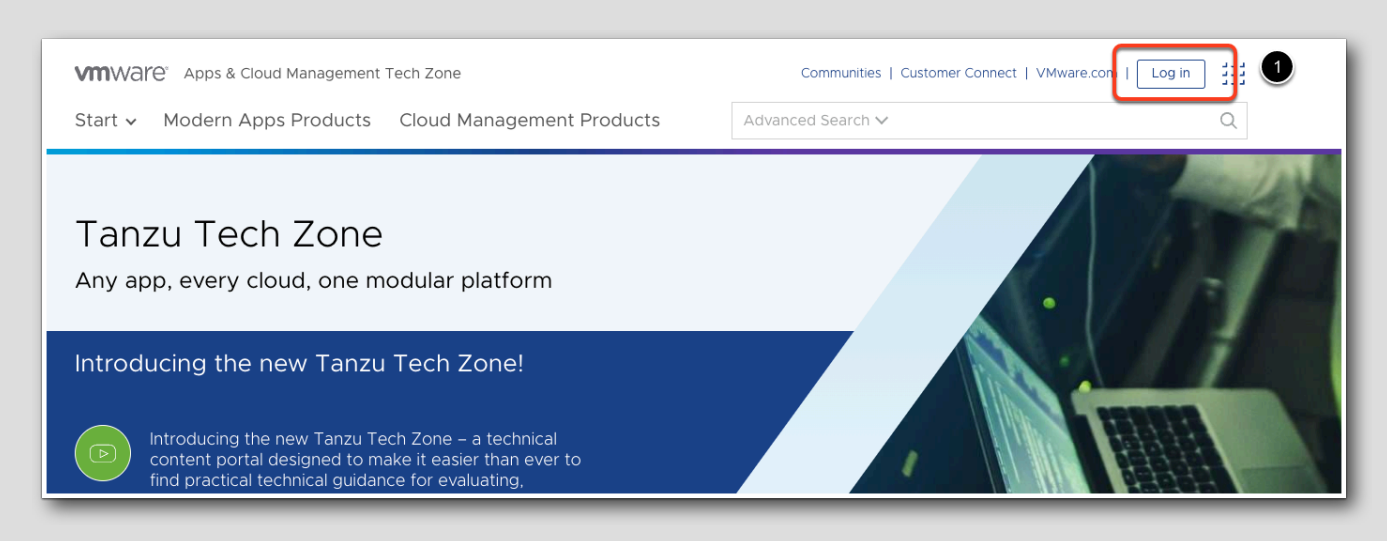
https://apps-cloudmgmt.techzone.vmware.com/tanzu-techzone

Getting Started



Getting started with VMware Tanzu Tech Zone is simple.

1. Go to :https://apps-cloudmgmt.techzone.vmware.com/tanzu-techzone



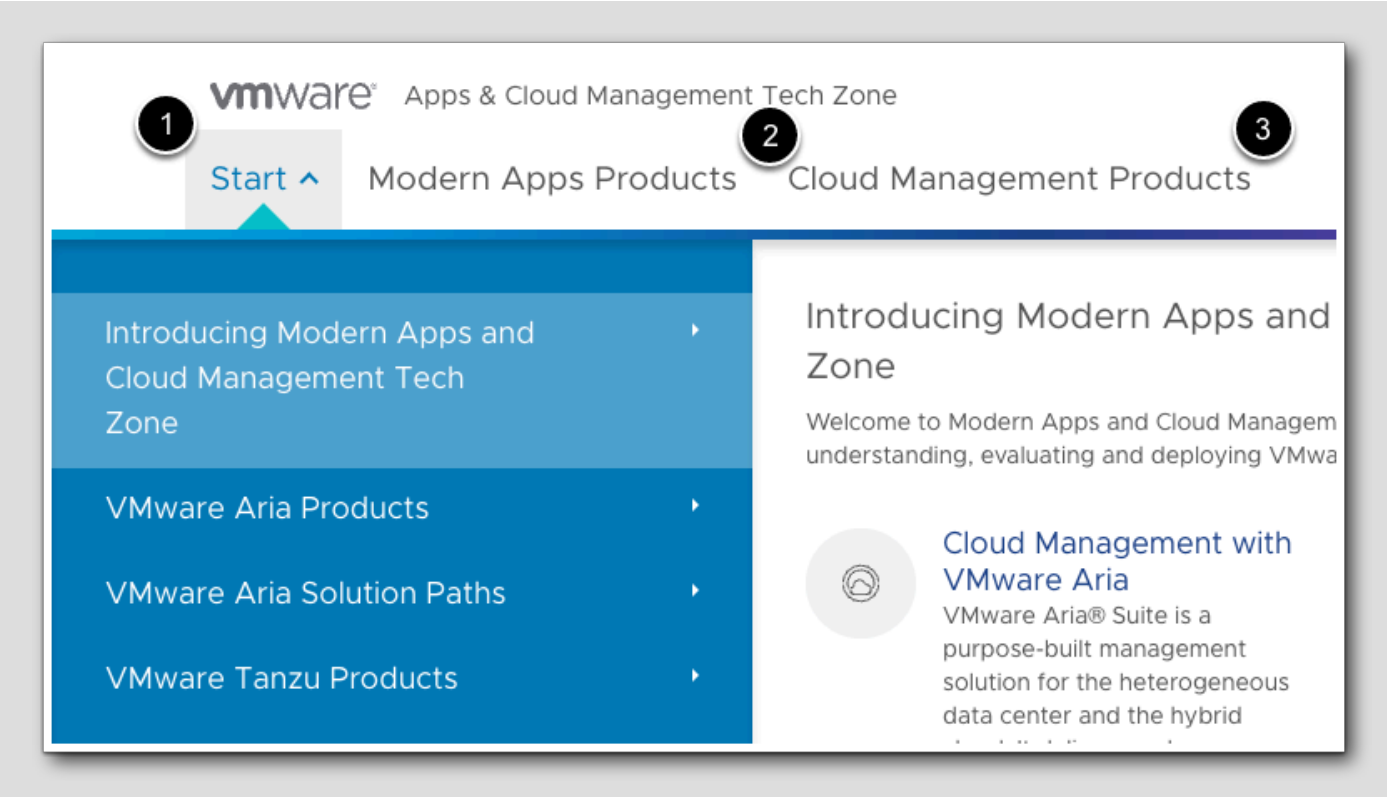
1. Click Log In.

You can use an existing Customer Connect / Partner Connect / Customer Connect ID or you can create a new log in. There is no charge for using Tech Zone.

Click any of the buttons to access content.

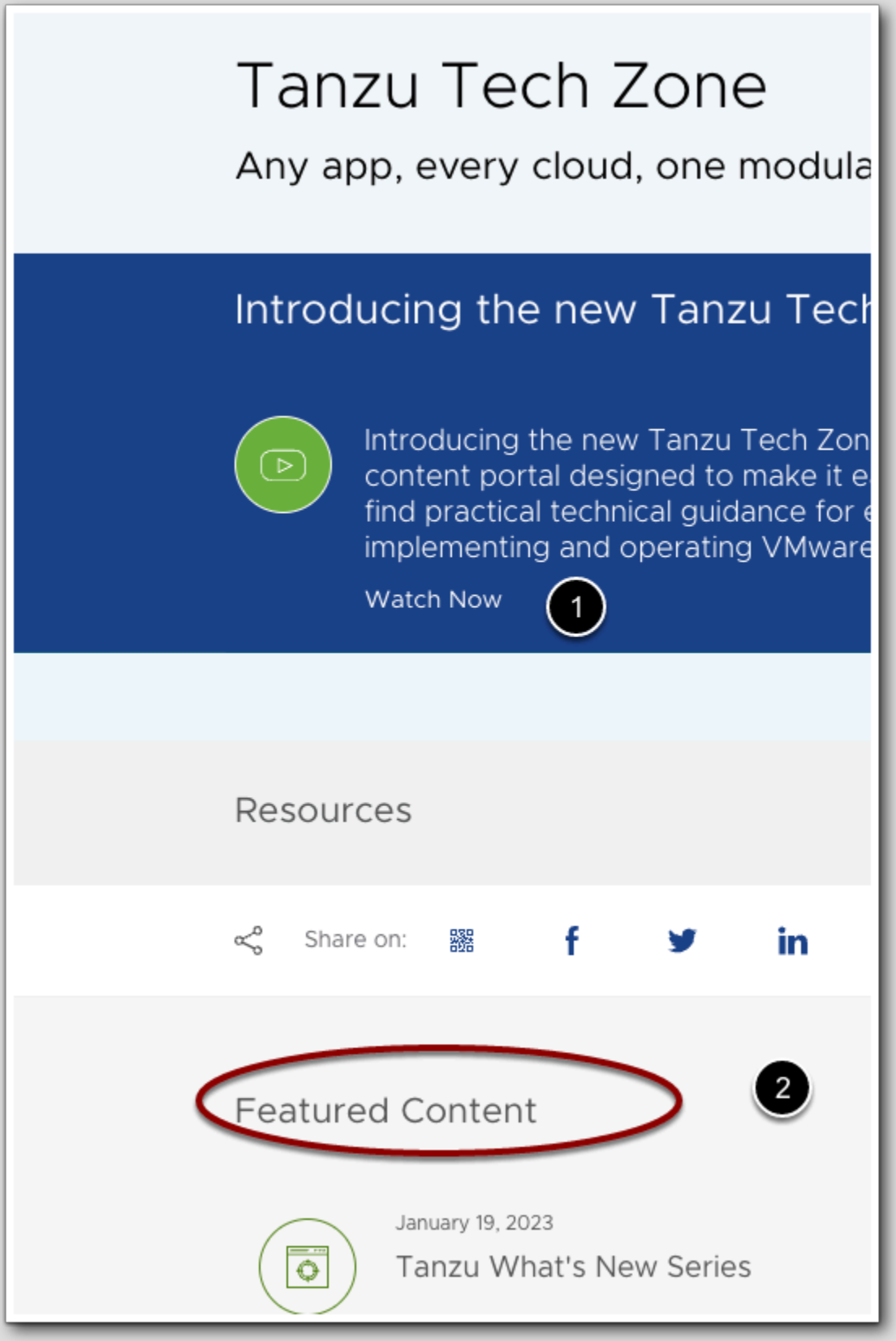
How do I navigate Tanzu Tech Zone?

[210]



The screenshot above gives an overview of the key pages and navigation elements of the Tanzu Tech Zone:

1. Click Start
2. The Tanzu Products drop-down menu displays a list of the primary Tanzu product families that link to product pages that feature both product-specific content as well as content for related solutions and use cases.
3. The Cloud Management Products drop-down menu displays a list of cloud management products that link to product pages that provide associated content.


Tanzu Tech Zone
Any app, every cloud, one modula
Introducing the new Tanzu Tech
Introducing the new Tanzu Tech Zon
content portal designed to make it e
find practical technical guidance for
implementing and operating VMware
Watch Now
1
Resources
Share on:
Featured Content
2
January 19, 2023
Tanzu What's New Series

1. Click Watch Now to see a video introduction to the Tanzu family of solutions.
2. The Featured Content section is regularly updated with the most popular and in-demand Tanzu content.

If you continue to scroll down the page, you will see additional sections that list the Tanzu products and a Latest Content feed that displays all of the most recent content that has been uploaded to the Tanzu Tech Zone.

As you explore further into Tech Zone, you will find both featured-content and latest-content feeds filtered for each product. Each product also has a dedicated Activity Path page that displays available content in key categories, including demos, labs, blogs, training, and more.

Additional Free Resources

[211]

VMware provides a host of Modern Apps FREE learning resources, check out:

https://tanzu.vmware.com/developer/workshops/

https://tanzu.vmware.com/

https://modernapps.ninja/

Introduction to Kube Academy & VMware Connect Learning

[212]

The new *Tanzu Tech Zone* is the fastest path to understanding, evaluating, and deploying VMware Tanzu solutions. Tech Zone's mission is to bring you insightful and accurate guidance on our current, released products.

The program is a initiative designed to address the unique needs of the IT and developer workforce as the market transitions towards increasingly automated and cloud native technologies.

Note: To appreciate the full experience of the VMware Tanzu Tech Zone you will need access the websites highlighted in this session from browser on a second device with access to the internet as some of the links in this Module and the various courses may not work from within the Hands On Labs (HoL) environment. For security the HoL environment has internet filtering enabled and therefore not all of the links in this module will work from within the Hands On Labs Environment.

Introduction to Kube Academy

Practical, hands on experience and regular practice is key to becoming an effective practitioner in cloud native technologies. VMware Kube Academy focuses on hands on experience providing a fast path education, centered around real world workflows, tools and processes, and made real through actual contribution in open & open source community projects

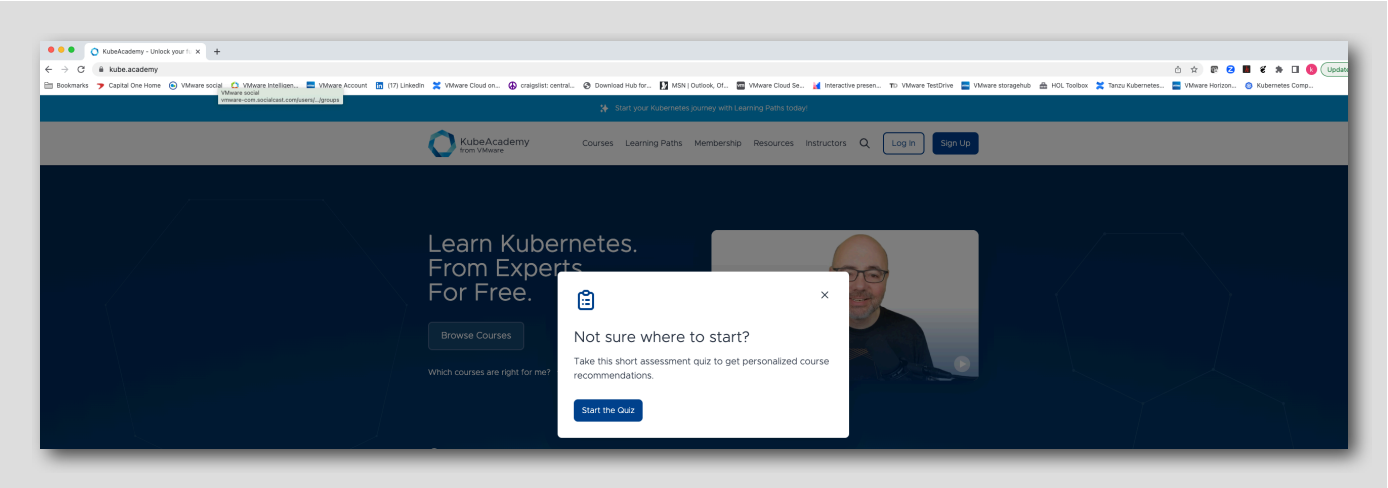
Kubernetes Academy Brought to You by VMware—is a free, product-agnostic Kubernetes and cloud native technology education platform. *As Kubernetes adoption continues to soar*, Kubernetes Academy provides an accessible learning path to advance your skill set, regardless of where you are on your Kubernetes journey. Learning a new technology can be challenging, especially one as young as Kubernetes. If you find yourself eager to learn more about Kubernetes but you' re short on time, Kubernetes Academy was made with you in mind.

Inspired by Khan Academy—we admire how Sal Khan broke down complex topics into component parts that are easy to understand. Since its inception in 2008, Khan Academy has taken on a life of its own—hosting content from a variety of instructors on a multitude of topics. That' s our goal with Kubernetes Academy—to build a wealth of knowledge from instructors throughout the open source community to provide well-rounded, unbiased training.

Kubernetes Academy courses are composed of a series of video lessons—each five-to-eight minutes long. The courses dive into topics for skill levels ranging from beginner to intermediate, with advanced topics being introduced soon. A breadth of courses are available for various roles, from system administrators new to Kubernetes to operators ready to manage clusters

Getting Started With Kube Academy

[214]

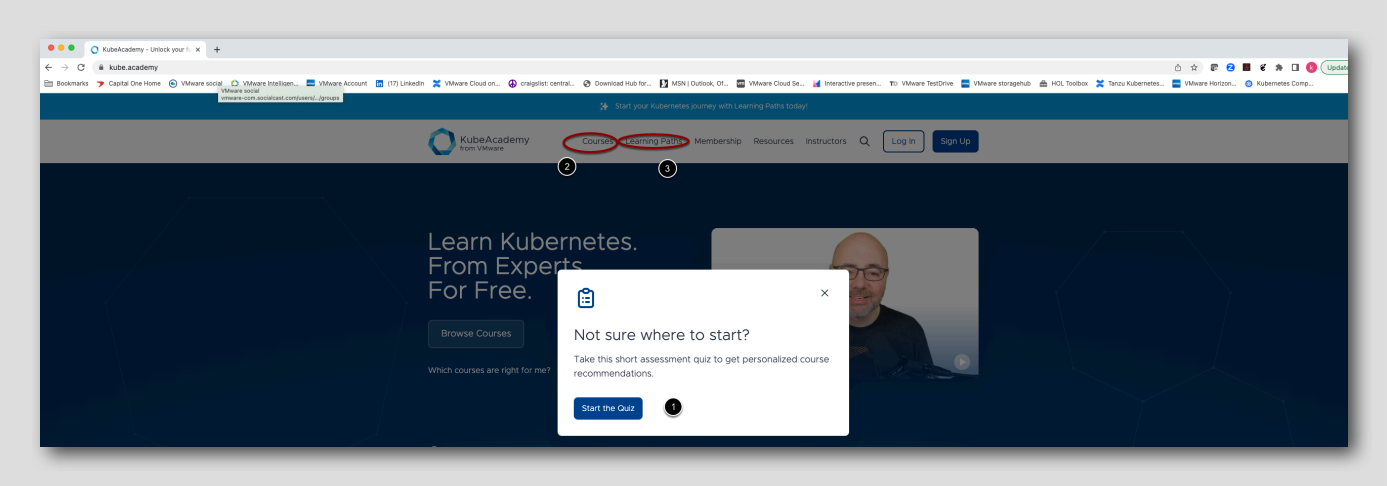


Getting started is easy: https://kube.academy

if you have never used Kube Academy before click the Sign Up button to create a Kube Academy profile so you can track your progress, save your favorite courses and lessons, earn achievements, partake in discussions, and more.

Navigating The Content On Kube Academy

[215]



Navigation on Kube Academy is simple, either take the "Not sure where to start ?" quiz (1) or jump right in by examining courses (2) or learning paths (3)

Introducing Kube Academy Pro: In-Depth Kubernetes Training, Totally Free

[216]

At VMware, we want our knowledge, and the knowledge of Kubernetes experts, to be your gain. Last August, when *we launched Kube Academy* our free, product-agnostic Kubernetes and cloud native technology education program the reception was overwhelmingly positive, so we decided to take Kube Academy one step further.

Thats why were excited to introduce *Kube Academy Pro* a free education program that grants members access to technical, in-depth, product-agnostic Kubernetes training, exclusive workshops, virtual events with community leaders, instructor-led webinars, and more.

Kube Academy Pro connects you with industry leaders for free so you can advance your skill set regardless of where you are on your Kubernetes journey. Whether youre a systems administrator new to Kubernetes or an operator ready to manage clusters, KubeAcademy Pro includes a breadth of courses for a wide range of roles.

Kube Academy Pro instructors are made up of VMware Kubernetes architects, field engineers, trainers, and developer advocates who bring their experience helping customers adopt Kubernetes, as well as their own experience learning Kubernetes.

Kube Academy's lessons run just 5-15 minutes, as they are designed for those who simply want to kick the tires on Kubernetes. But some topics cannot be done justice in under 15 minutes.

Kube Academy Pro courses are designed for those who are ready to take their Kubernetes knowledge to the next level. Each lesson runs 15-40 minutes long, allowing instructors to dive deeply into each topic.

In addition to in-depth courses, Kube Academy Pro members get access to interactive virtual events. Kube Academy instructors will be hosting a series of webinars that will dive even deeper into certain course topics for which they will be taking Pro members' questions.

Kube Academy Pro extends Kube Academy with lengthier courses to extend your knowledge for free. While Kube Academy courses are 5 to 15 minutes in length, Kube Academy Pro courses run 15 to 40 minutes explaining the concepts in-depths, targeting the people who already have some basic understanding of the technology.

If you already have a Kube Academy account you do not need to create an account for Kube Academy Pro as they use the same account.

Kube Academy Pro can be accessed at : https://kube.academy

if you have never used Kube Academy or Kube Academy Pro click the Sign Up button to create a Kube Academy profile so you can track your progress, save your favorite courses and lessons, earn achievements, partake in discussions, and more.

This QR code will connect you to the Kube Academy content.

Introduction to Spring Academy

[217]

Start your learning journey for free with development projects that get you using Spring in real-world development scenarios. Our browser-based lab environment gives you access to the tools you need to get hands-on experience with ease. And with Spring Academy Pro, unlock access to exclusive content, video lessons, and hands-on labs, plus a voucher for the Spring Certified Professional exam.

Create your free account *HERE*.

This QR code will link you to the Spring Academy content.

Introduction to VMware Connect Learning

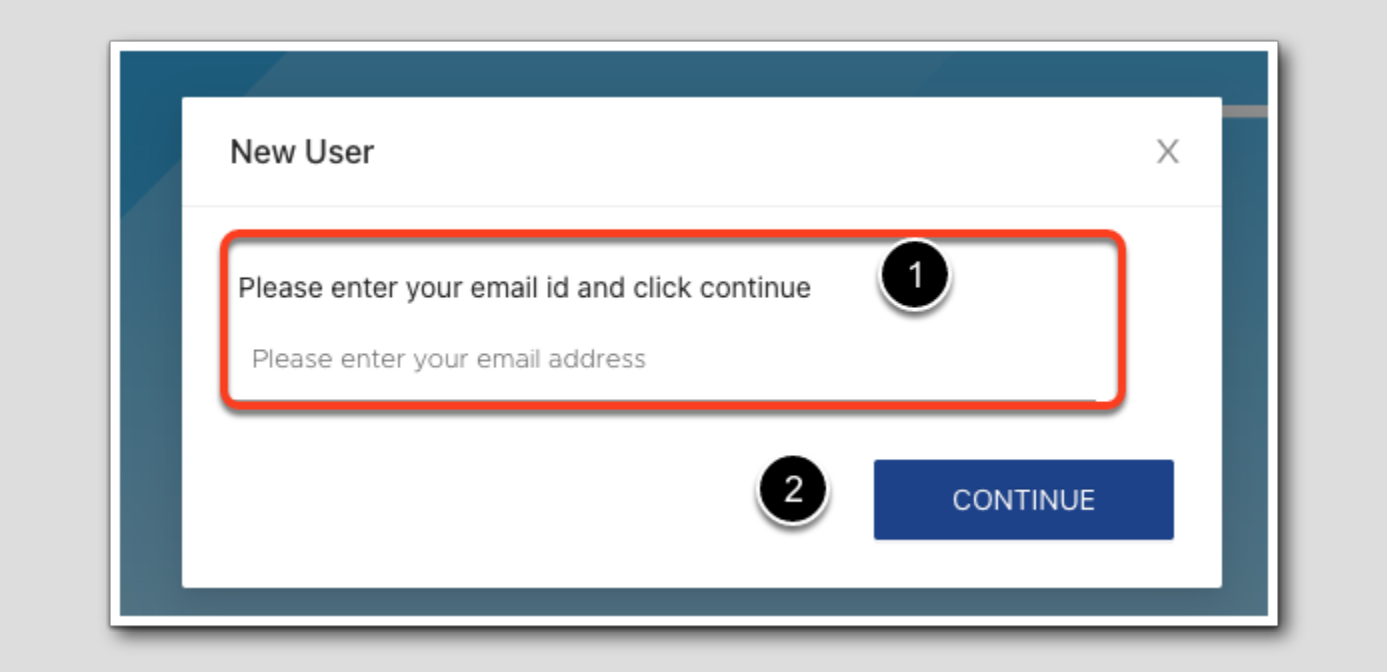
Connect Learning provides you with 24/7 access to a comprehensive digital learning library, created by VMware experts, including access to self-paced eLearning, relevant webinars, and live community events.

- Digital training options include online courses, labs, and videos that help build your VMware knowledge and skills
- 24/7 access to training that spans the VMware product portfolio including VMware NSX, vRealize Suite, vSphere, and more
- Expert-level instruction helps you configure, deploy, and troubleshoot your VMware solutions
- Basic and Premium Subscriptions for individual learners and Enterprise-tier Subscriptions for enterprises who need to train global teams

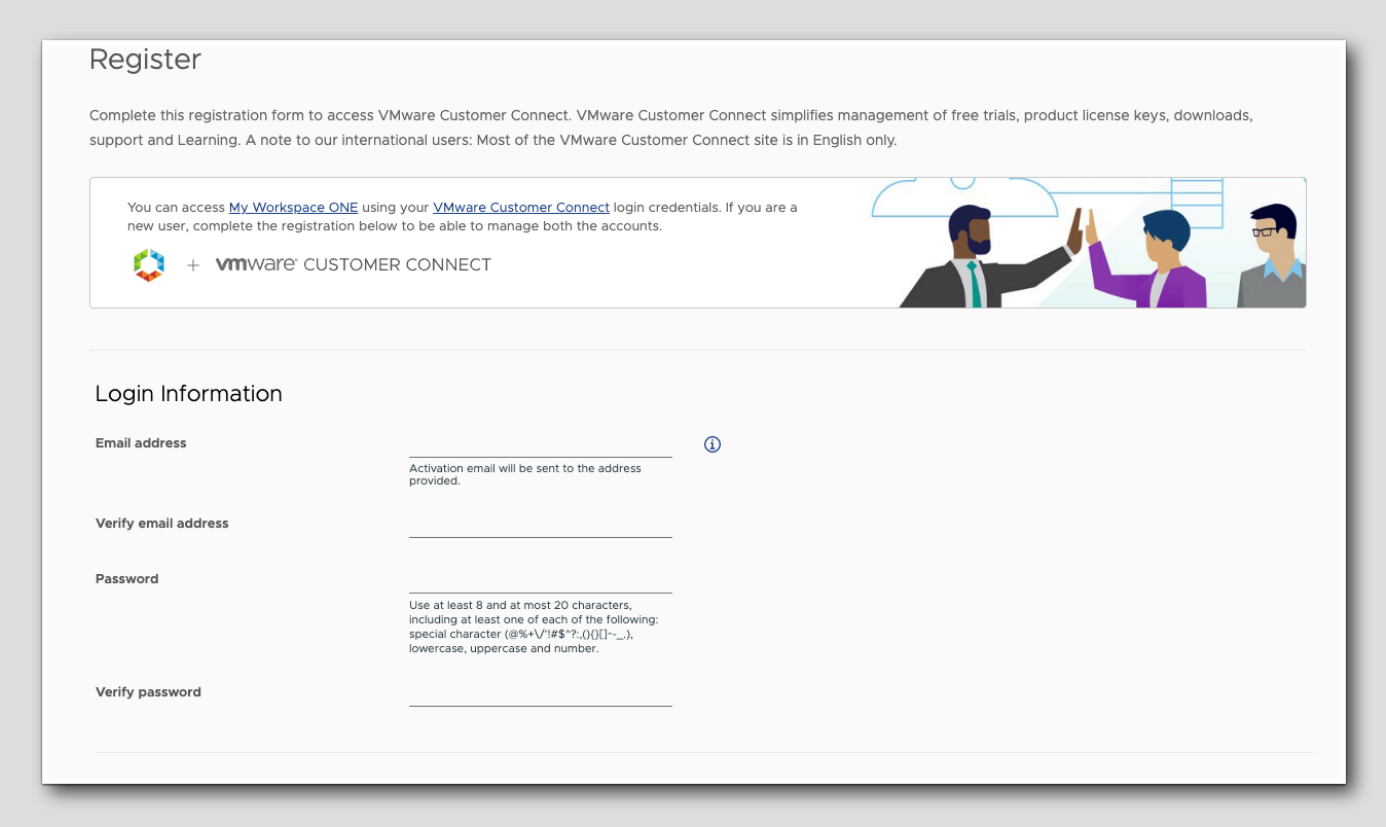
Getting Started With VMware Connect Learning

Go to : https://learning.customerconnect.vmware.com

1. Click "Sign Up"



1. Enter the email address you wish to use.
2. Click CONTINUE,



Then complete your registration on the Register page.

Additional Free Resources

[220]

VMware provides a host of Modern Apps FREE learning resources, check out:

https://tanzu.vmware.com/developer/workshops/

https://tanzu.vmware.com/

https://modernapps.ninja/

Introduction to Tanzu Academy [221]

Get the training you need to succeed with VMware Tanzu Academy [222]

Tanzu Academy is an on-demand, comprehensive learning hub for platform and app operators to become experts at achieving outcomes with Tanzu products. Built by Tanzu experts, members gain access to highly curated material such as expert tutorials, guides, and hands-on labs infrastructure. Tanzu Academy includes in-depth learning for products like VMware Tanzu Application Platform, VMware Tanzu for Kubernetes Operations, and VMware Tanzu Application Service. Members will learn how to install, maintain, and get the most value out of each product.

- Improve your time to value with Tanzu products
- Grow skills for platform operations teams
- Improve ability to address security vulnerabilities
- Get flexible access to content and labs anytime, anywhere for global teams

Tanzu Academy has two learning tiers, the Free edition and the Pro edition.

Free Edition:

- View course catalog
- On-demand access to video lectures
- Video and article lessons
- Progress tracking

Pro Edition:

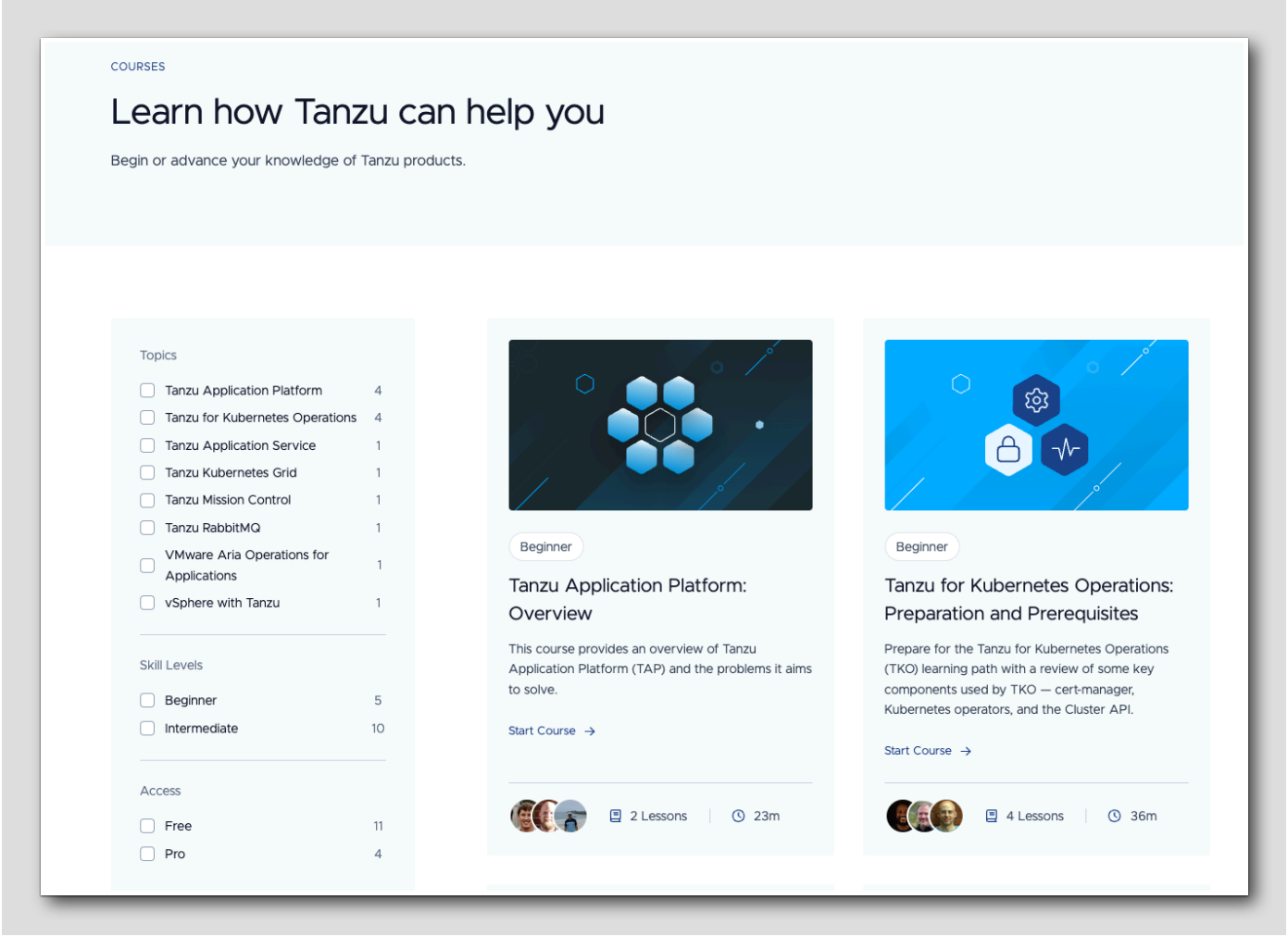
- Hands-on labs
- Manually provisioned IaaS infra
- Certification prep courses
- Certified professional exam voucher

Navigating Tanzu Academy

[223]

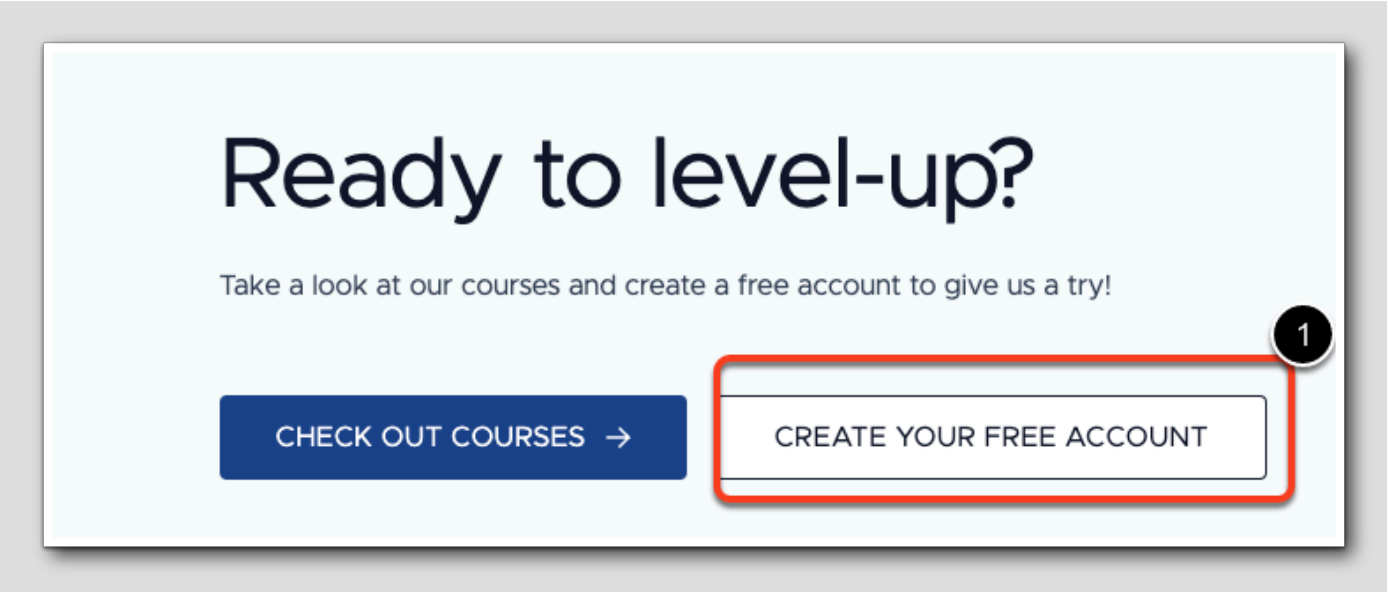
Start by visiting the Tanzu Academy landing page *HERE*.

From the landing page you can examine the courses which are available as well as register for a free account.



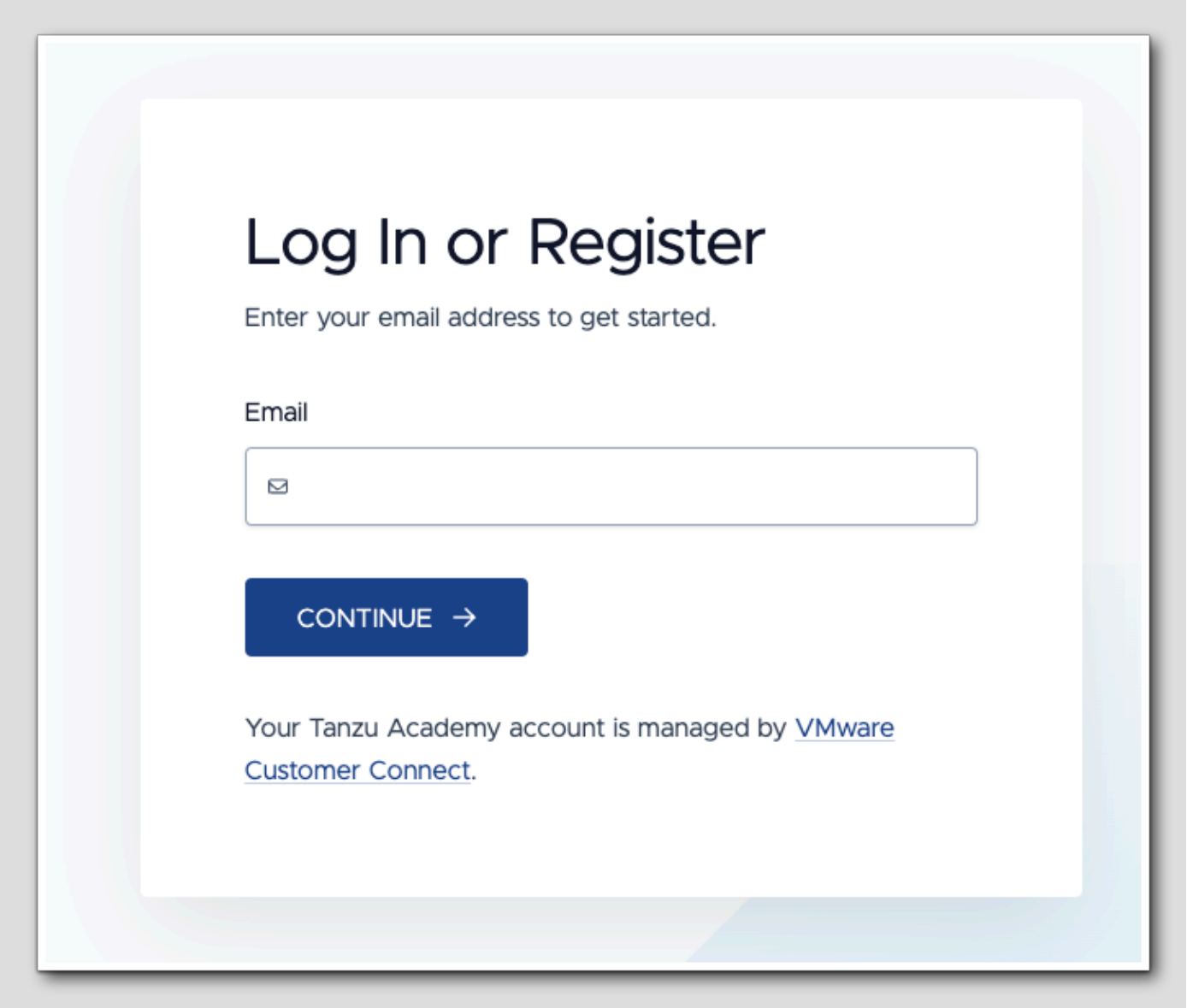
Creating a Free Tanzu Academy Account

[224]



Scroll down on the landing page.

1. Click CREATE YOUR FREE ACCOUNT

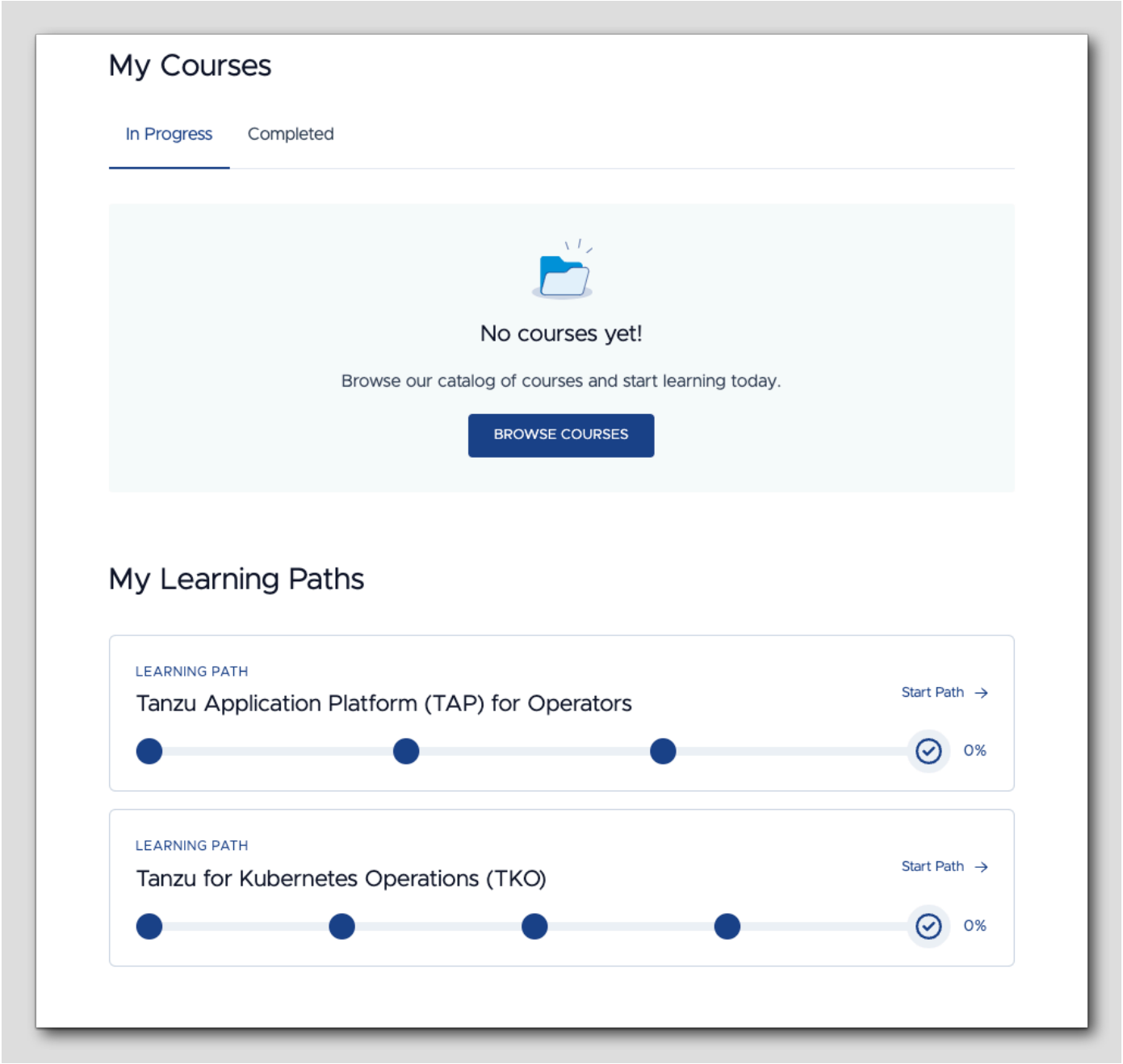


Enter your email account.

You will be taken to a registration page to create your free account and start your learning adventure with Tanzu Academy.

Choosing a Learning Path

[225]



After registering you can begin a learning path.

Tanzu Application Platform (TAP) for Operators [226]

LEARNING PATH

Tanzu Application Platform (TAP) for Operators

VMware Tanzu Application Platform is a modular, application-aware platform that provides a rich set of developer tooling and a pre-paved path to production to build and deploy software quickly and securely on any compliant public cloud or on-premises Kubernetes cluster. The material in this learning path will help platform operators prepare, set up, configure, and use Tanzu Application Platform.

3 Courses | 12h 13m

START PATH →

Tanzu Application Platform: Overview

This course provides an overview of Tanzu Application Platform (TAP) and the problems it aims to solve.

2 Lessons | 23m

Tanzu Application Platform: Installation

Get hands-on practice with configuring a basic installation of VMware Tanzu Application Platform.

11 Lessons | 2h 42m

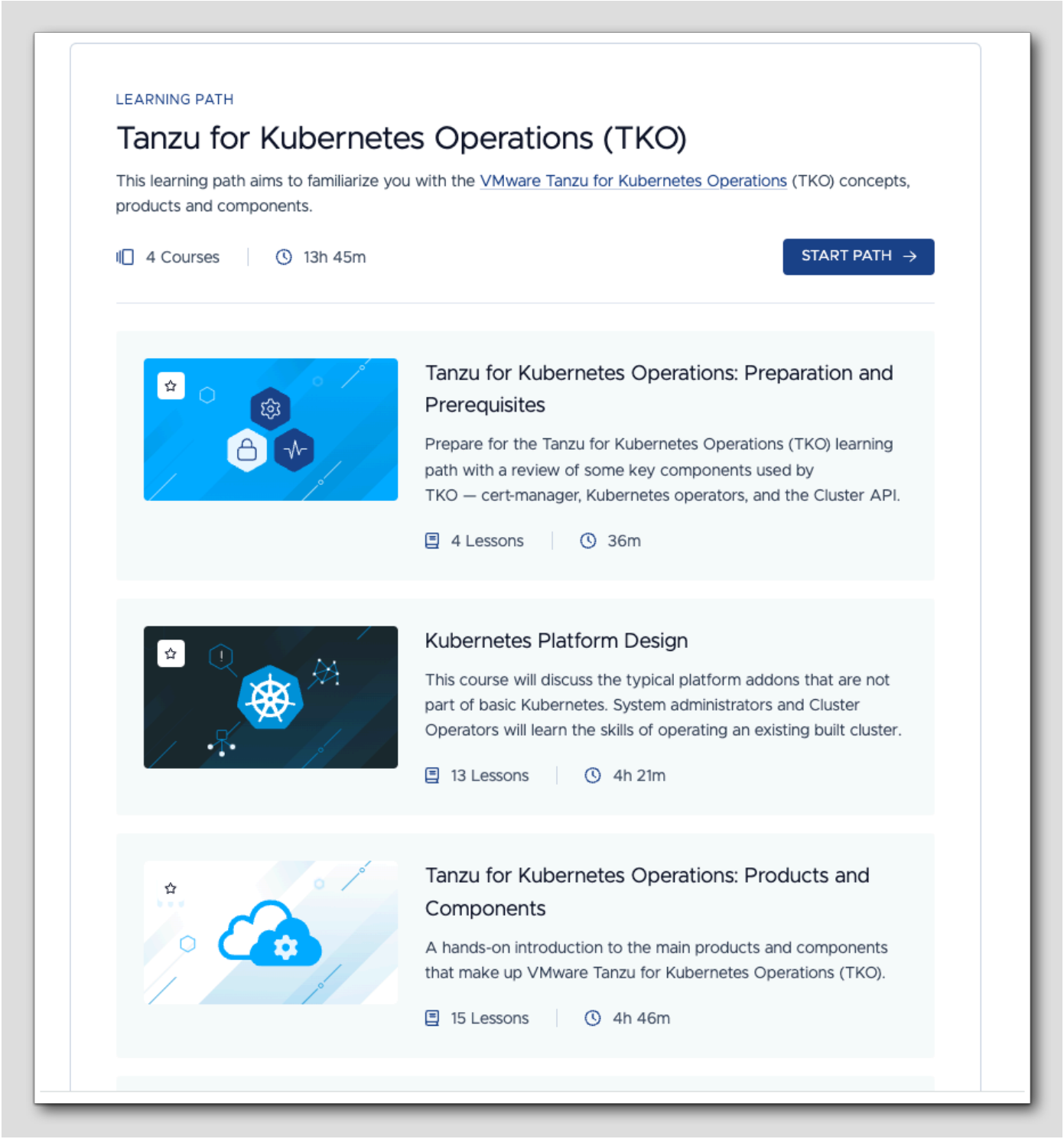
Tanzu Application Platform: Deploying a Production Environment

Make your Tanzu Application Platform more production-ready

12 Lessons | 9h 7m

VMware Tanzu Application Platform is a modular, application-aware platform that provides a rich set of developer tooling and a pre-paved path to production to build and deploy software quickly and securely on any compliant public cloud or on-premises Kubernetes cluster. The material in this learning path will help platform operators prepare, set up, configure, and use Tanzu Application Platform.

Tanzu for Kubernetes Operations (TKO) Learning Path [227]

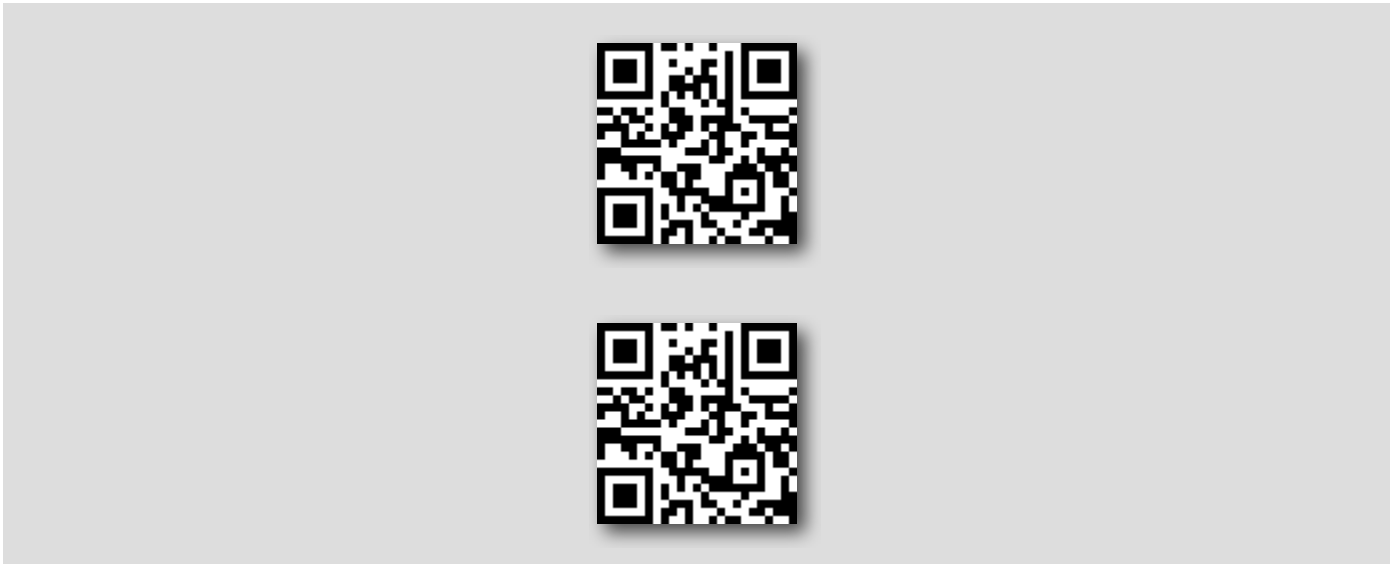


The material in this learning path aims to help you understand the VMware Tanzu for Kubernetes Operations(TKO) concepts, products, and components. It will prepare you to use theTKO reference architecture solutions to build and deploy TKO instances.

Conclusion

[228]

Congratulations on completing Module 5 You should now have an understanding of the VMware Tanzu Learning Products and Kube Academy products. Please use the QR code for more information on VMware TechZone.



Congratulations you have finished Module 5

To continue with this lab, proceed to any module below which interests you most.

Based on your interests, please proceed to any module below:

1. Module 1 - Introduction to Containers (15 minutes) (Basic) - In this module, we will explain containers and how they enable 3rd Platform application architectures to be run efficiently in distributed environments. We will also delve into the basics of Docker as a container platform. This is a reading module only without any interactive labs.
2. Module 2 - A Quick Tour of Docker (15 minutes) (Basic) In this module, we're getting hands on with Docker and explore how to create simple containers and how to leverage Docker container networking. Finally, we'll look at how you can build your own images using a Dockerfile.
3. Module 3 - Introduction to Kubernetes - (30 minutes) (Basic) In this module we will explain the place where container management fits and take a high level walk thru the Kubernetes architecture. We will walk through some worked examples using a kubernetes instance deployed on Tanzu Kubernetes Grid (TKG).
4. Module 4 - Introduction to Harbor - (30 minutes(Advanced) In this module we will walk through some of the capabilities of the Harbor secure image repository and explore some worked examples using images pre-populated into this labs Harbor instance.

From here you can:

1. Click to advance to the next page and continue with the next lab module
2. Open the TABLE OF CONTENTS to jump to any module or lesson in this lab manual
3. End your lab and come back and start it again in the future

VMware, Inc. 3401 Hillview Avenue Palo Alto CA 94304 USA Tel 877-486-9273 Fax 650-427-5001 vmware.com.
 Lab SKU: HOL-2433-01-MAP Version: 20230907-165422