



Docker

Containerization

- Containerization is the packaging together of software code with all its necessary components like libraries, frameworks, and other dependencies so that they are isolated in their own “container.”(redhat.com)

Containerization

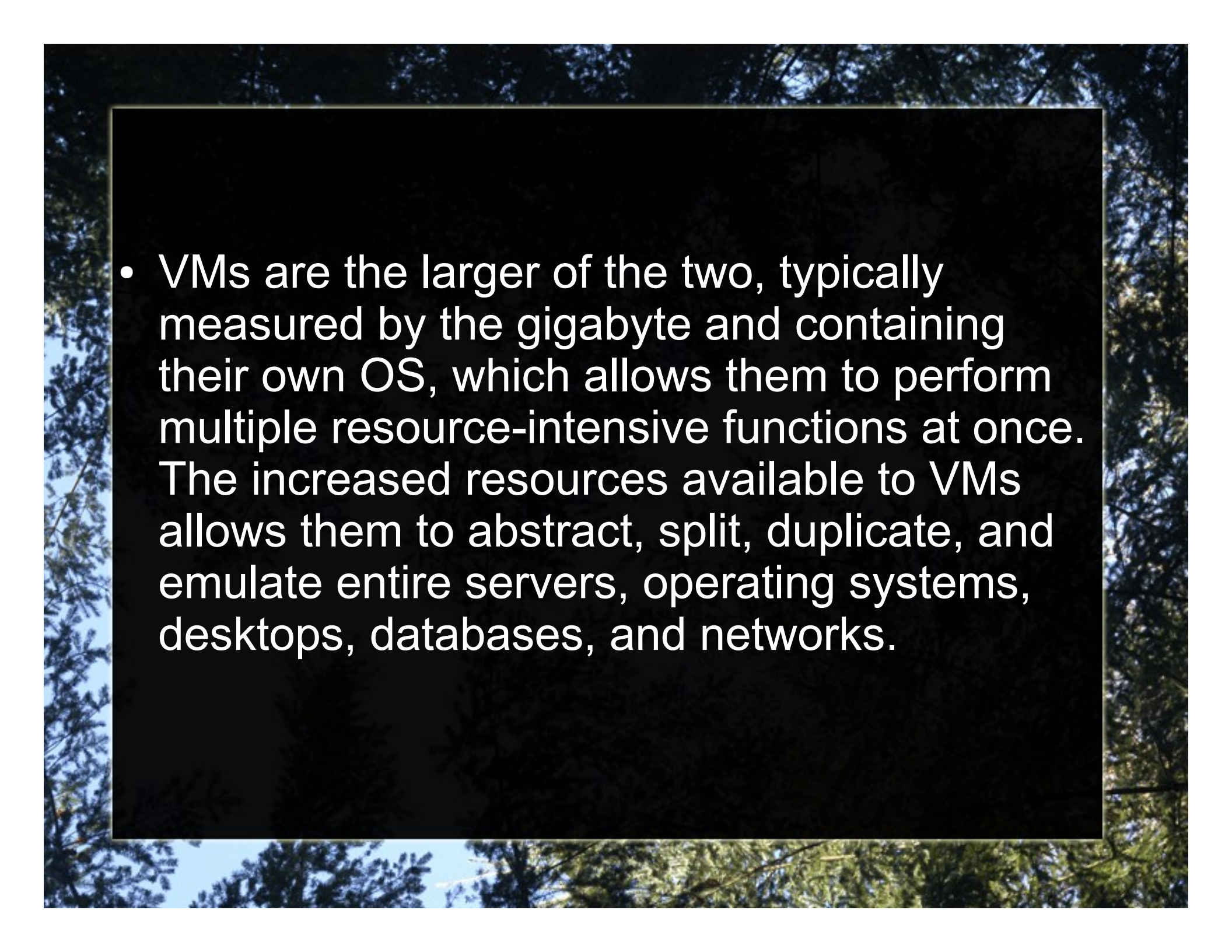
- The container acts as a kind of bubble or a computing environment surrounding the application and keeping it independent of its surroundings. It's basically a fully functional and portable computing environment.
- Containers are an alternative to coding on one platform or operating system, which made moving their application difficult since the code might not then be compatible with the new environment. This could result in bugs, errors, and glitches that needed fixing (meaning more time, less productivity, and a lot of frustration).


Containerization

- By packaging up an application in a container that can be moved across platforms and infrastructures, that application can be used wherever you move it because it has everything it needs to run successfully within it.

Container vs VM

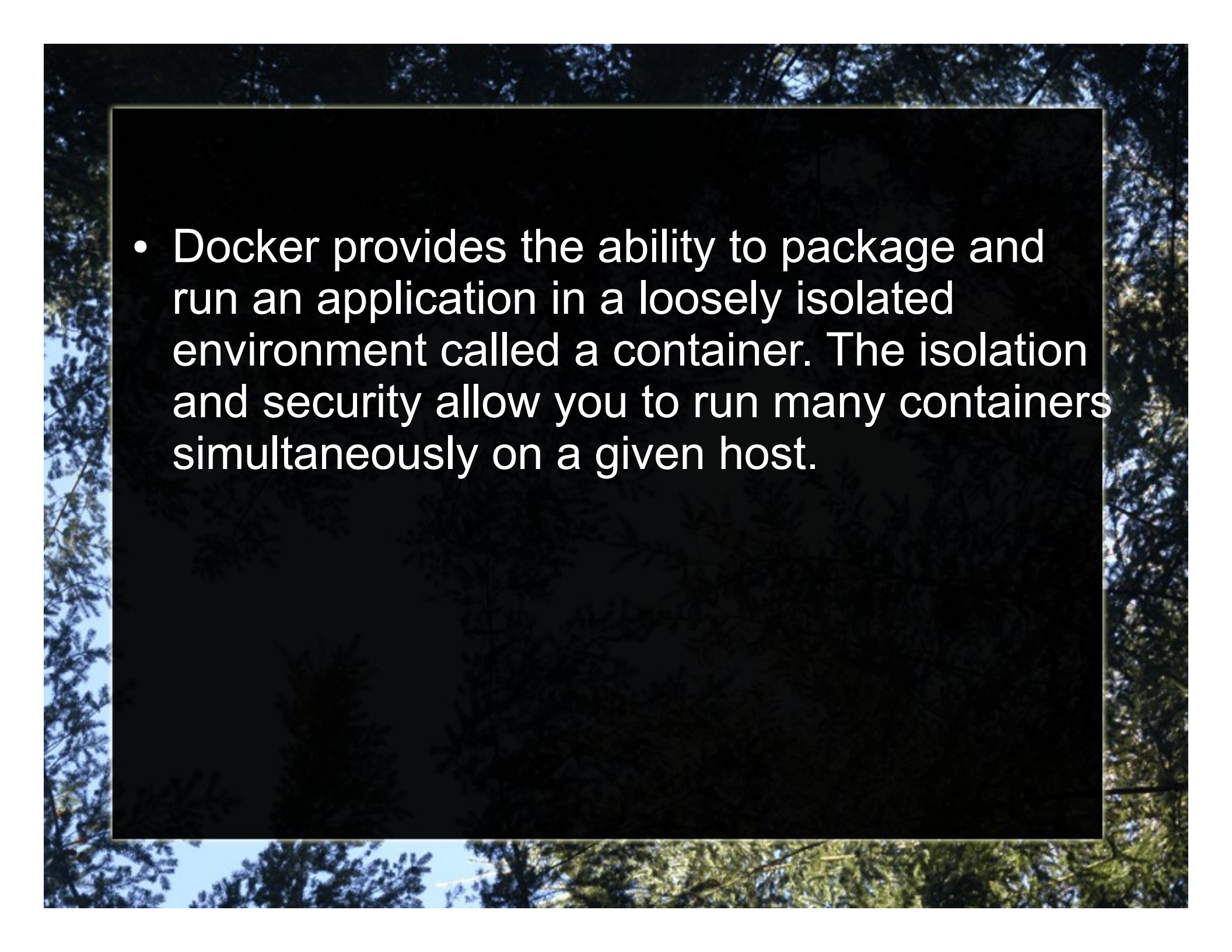
- A virtual machine (VM) is a virtual environment that functions as a virtual computer system with its own CPU, memory, network interface, and storage, created on a physical hardware system (located off- or on-premises).
- Containerization and virtualization are similar in that they both allow for full isolation of applications so that they can be operational in multiple environments. Where the main differences lie are in size and portability.

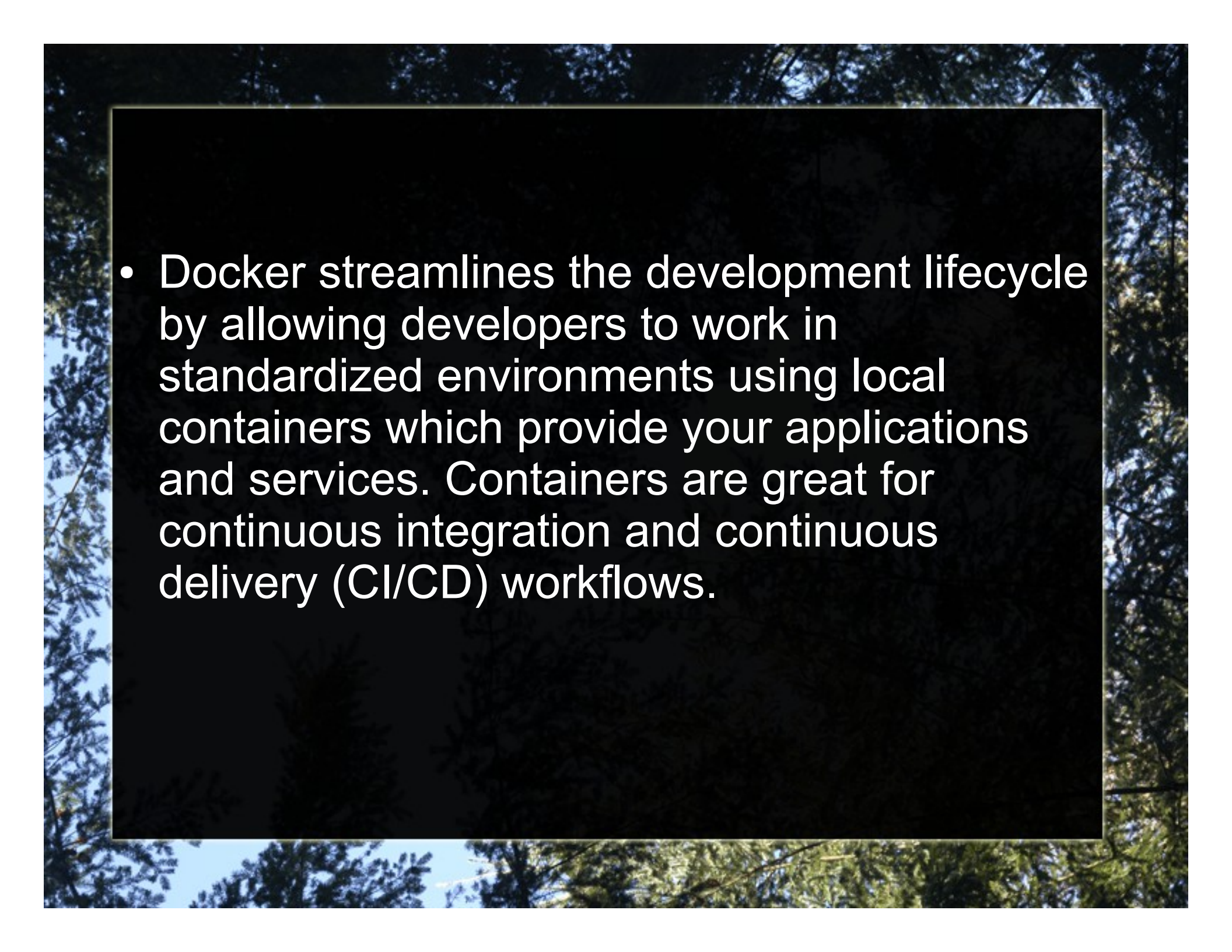
- 
- VMs are the larger of the two, typically measured by the gigabyte and containing their own OS, which allows them to perform multiple resource-intensive functions at once. The increased resources available to VMs allows them to abstract, split, duplicate, and emulate entire servers, operating systems, desktops, databases, and networks.

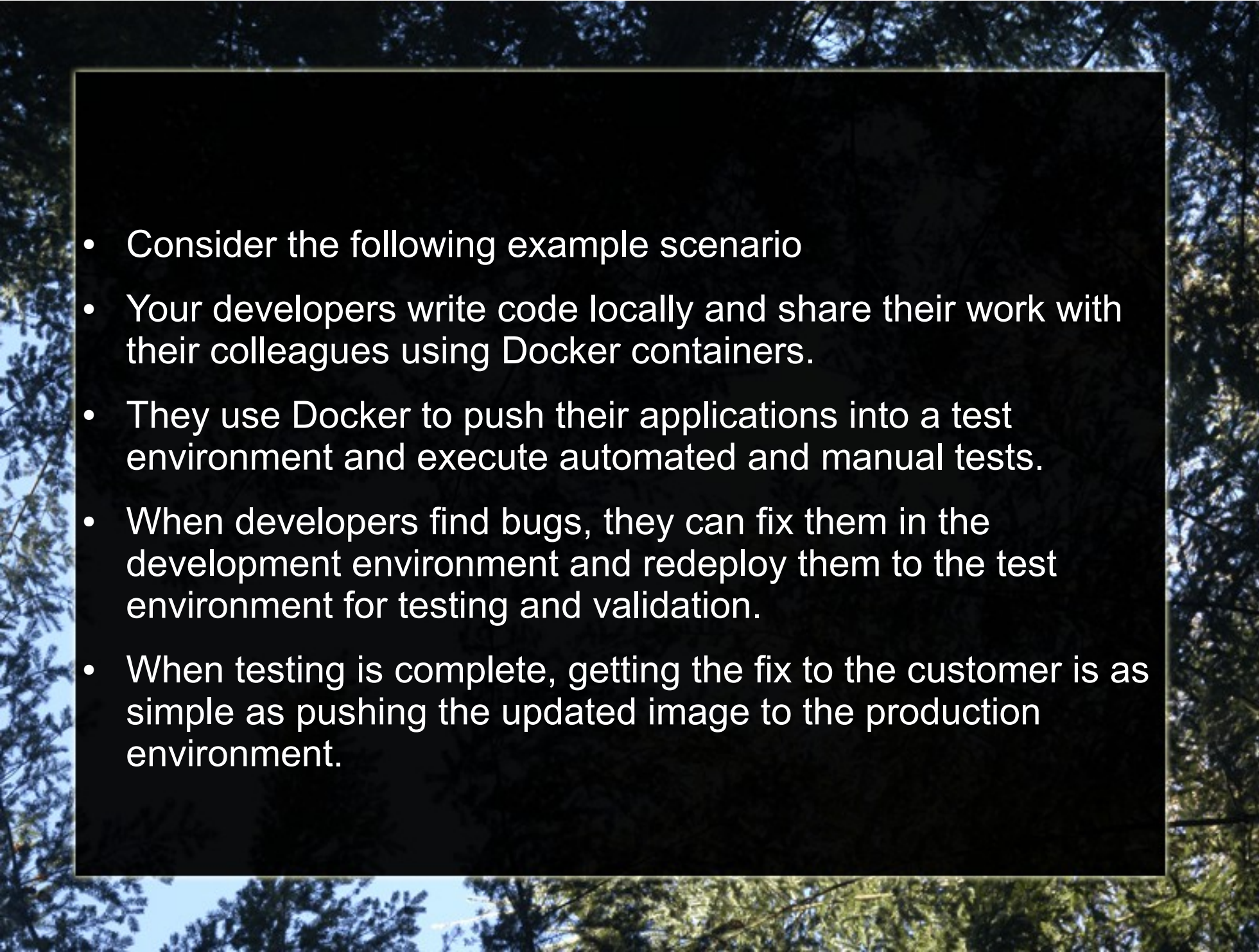
- 
- Containers are much smaller, typically measured by the megabyte and not packaging anything bigger than an app and its running environment.
 - Where VMs work well with traditional, monolithic IT architecture, containers were made to be compatible with newer and emerging technology like clouds, CI/CD, and DevOps.

Docker for Containerization

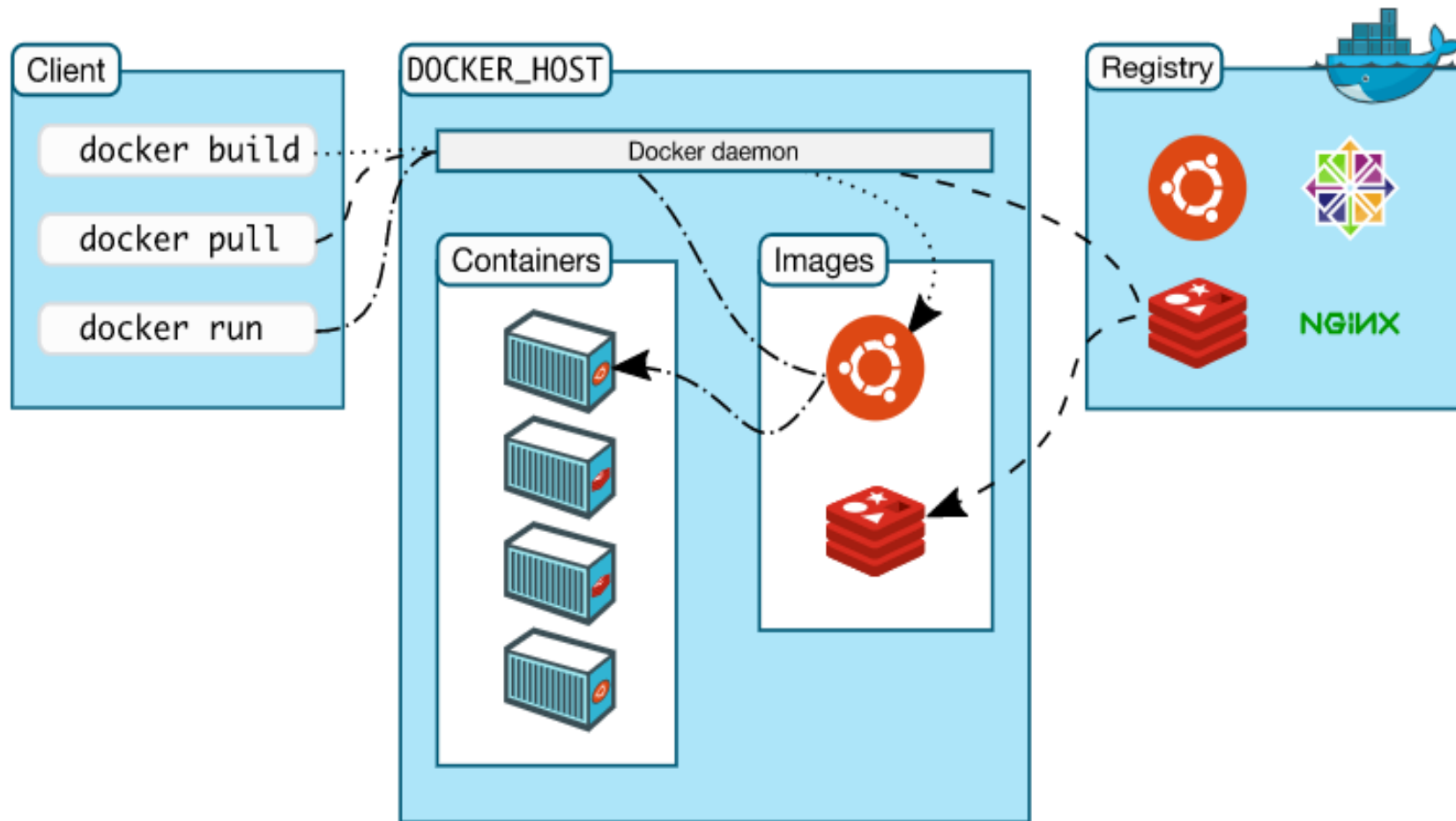
- Docker is an open platform for developing, shipping, and running applications. Docker enables you to separate your applications from your infrastructure so you can deliver software quickly. With Docker, you can manage your infrastructure in the same ways you manage your applications. By taking advantage of Docker's methodologies for shipping, testing, and deploying code quickly, you can significantly reduce the delay between writing code and running it in production.

- 
- Docker provides the ability to package and run an application in a loosely isolated environment called a container. The isolation and security allow you to run many containers simultaneously on a given host.

- 
- Docker streamlines the development lifecycle by allowing developers to work in standardized environments using local containers which provide your applications and services. Containers are great for continuous integration and continuous delivery (CI/CD) workflows.

- 
- Consider the following example scenario
 - Your developers write code locally and share their work with their colleagues using Docker containers.
 - They use Docker to push their applications into a test environment and execute automated and manual tests.
 - When developers find bugs, they can fix them in the development environment and redeploy them to the test environment for testing and validation.
 - When testing is complete, getting the fix to the customer is as simple as pushing the updated image to the production environment.

Docker Architecture



Docker Deamon

- The Docker daemon (dockerd) listens for Docker API requests and manages Docker objects such as images, containers, networks, and volumes. A daemon can also communicate with other daemons to manage Docker services.

Docker Client

- The Docker client (docker) is the primary way that many Docker users interact with Docker. When you use commands such as `docker run`, the client sends these commands to `dockerd`, which carries them out. The `docker` command uses the Docker API. The Docker client can communicate with more than one daemon.

Docker Registry

- A Docker registry stores Docker images. Docker Hub is a public registry that anyone can use, and Docker is configured to look for images on Docker Hub by default. You can even run your own private registry.
- When you use the `docker pull` or `docker run` commands, the required images are pulled from your configured registry. When you use the `docker push` command, your image is pushed to your configured registry.

Docker Objects

- When you use Docker, you are creating and using images, containers, networks, volumes, plugins, and other objects. This section is a brief overview of some of those objects.

Images

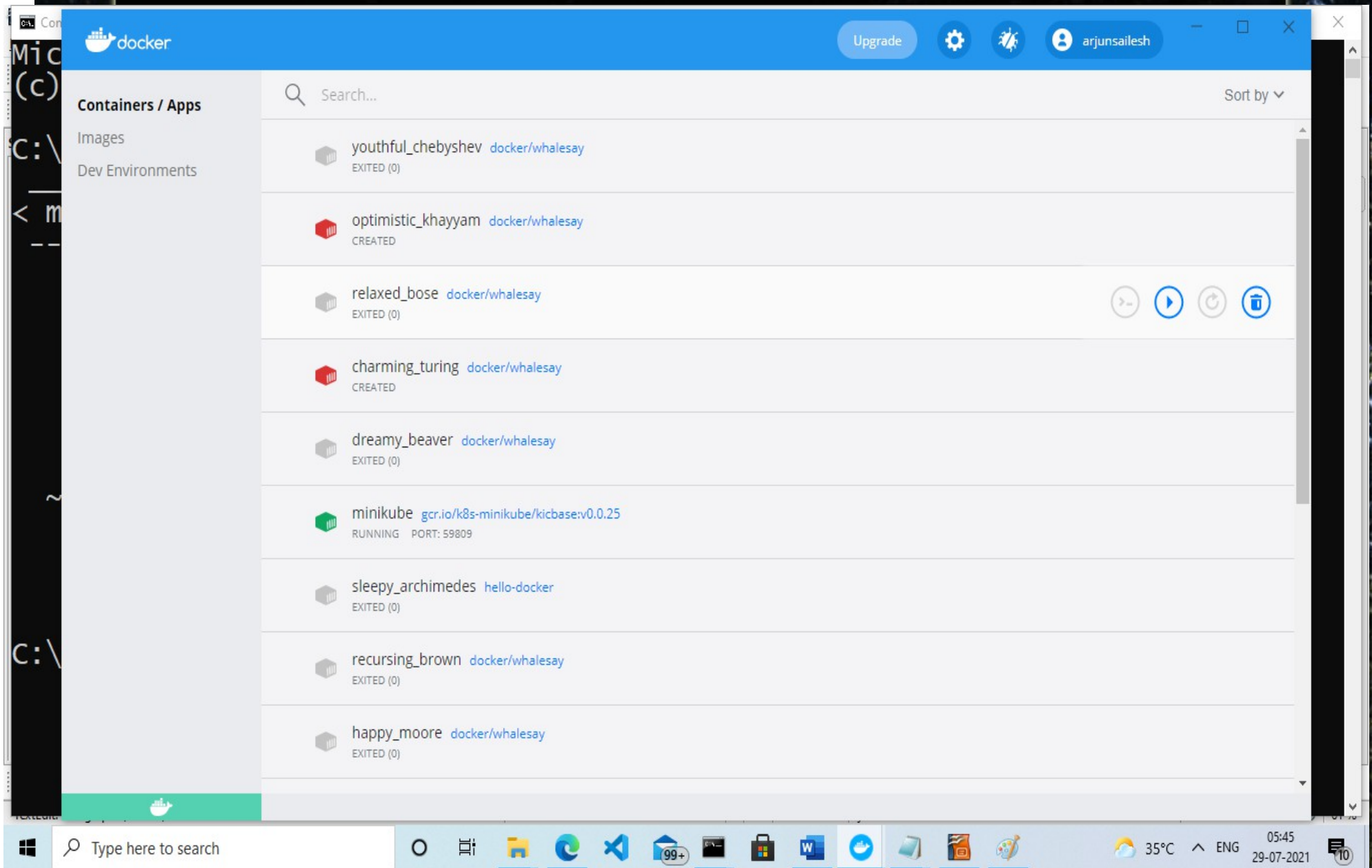
- An image is a read-only template with instructions for creating a Docker container. Often, an image is based on another image, with some additional customization. For example, you may build an image which is based on the ubuntu image, but installs the Apache web server and your application, as well as the configuration details needed to make your application run.

Container

- A container is a runnable instance of an image. You can create, start, stop, move, or delete a container using the Docker API or CLI. You can connect a container to one or more networks, attach storage to it, or even create a new image based on its current state.
- By default, a container is relatively well isolated from other containers and its host machine. You can control how isolated a container's network, storage, or other underlying subsystems are from other containers or from the host machine.
- A container is defined by its image as well as any configuration options you provide to it when you create or start it. When a container is removed, any changes to its state that are not stored in persistent storage disappear.

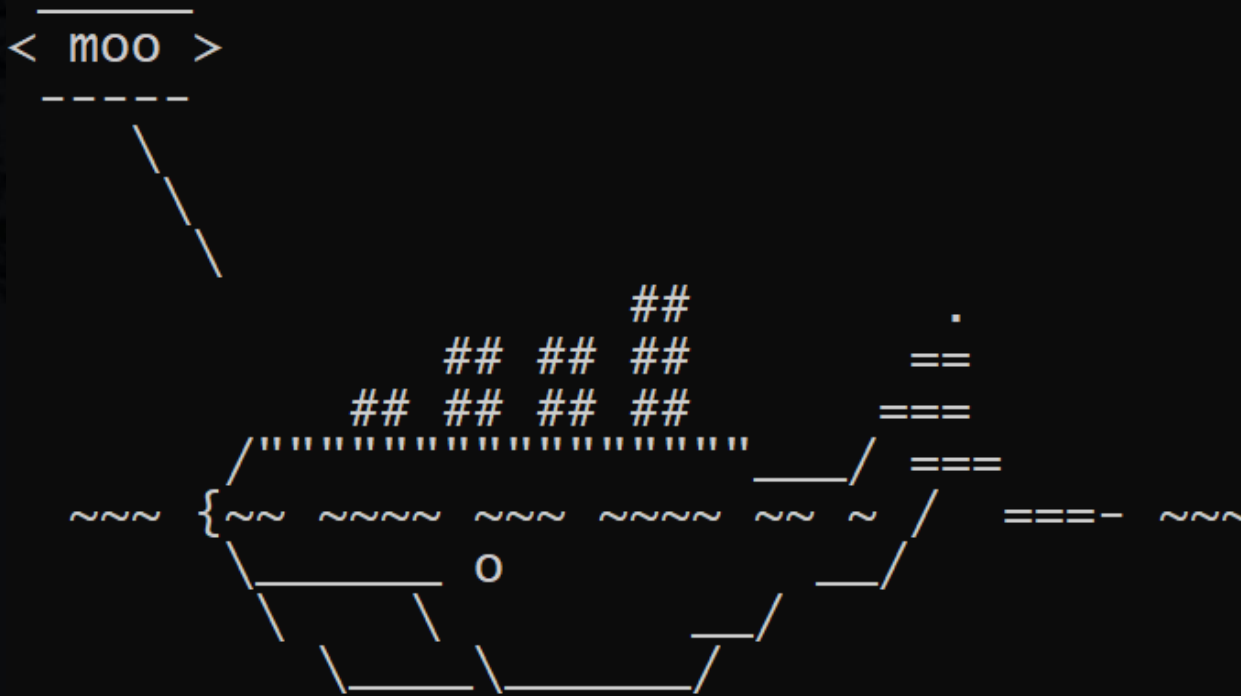
Getting Started With Docker

- Starting docker desktop



Docker run

- Docker run docker/whalesay cowsay moo



Docker Build Command

- Docker can build images automatically by reading the instructions from a Dockerfile. A Dockerfile is a text document that contains all the commands a user could call on the command line to assemble an image. Using docker build users can create an automated build that executes several command-line instructions in succession.
- <https://docs.docker.com/engine/reference/builder/>

Dockerfile

- FROM node:alpine
- COPY . /app
- WORKDIR /app
- CMD node hello.js

Docker Images

- Docker Images displays images available in docker image repository

Docker push

- `docker push arjunsailash/mydocker:hello-docker`

```
C:\WINDOWS\system32>docker login
```

Login with your Docker ID to push and pull images from Docker Hub. If you don't have a Docker ID, head over to <https://hub.docker.com> to create one.

Username: arjunsailsh

Password:

Login Succeeded

```
C:\WINDOWS\system32>docker push arjunsailsh/mydocker:hello-docker
```

The push refers to repository [docker.io/arjunsailsh/mydocker]

5f70bf18a086: Mounted from docker/whalesay

a2c9479aba2d: Pushed

908ecb461e7f: Mounted from library/node

77fc6ca46abe: Mounted from library/node

0605eb5303bb: Mounted from library/node

b2d5eeeaba3a: Mounted from library/node

hello-docker: digest: sha256:b97082ac0a3b2bc58d6471577b2fa090930ad453efd3080c530a704856bbf20f size: 1571

```
C:\WINDOWS\system32>docker login
```

Authenticating with existing credentials...

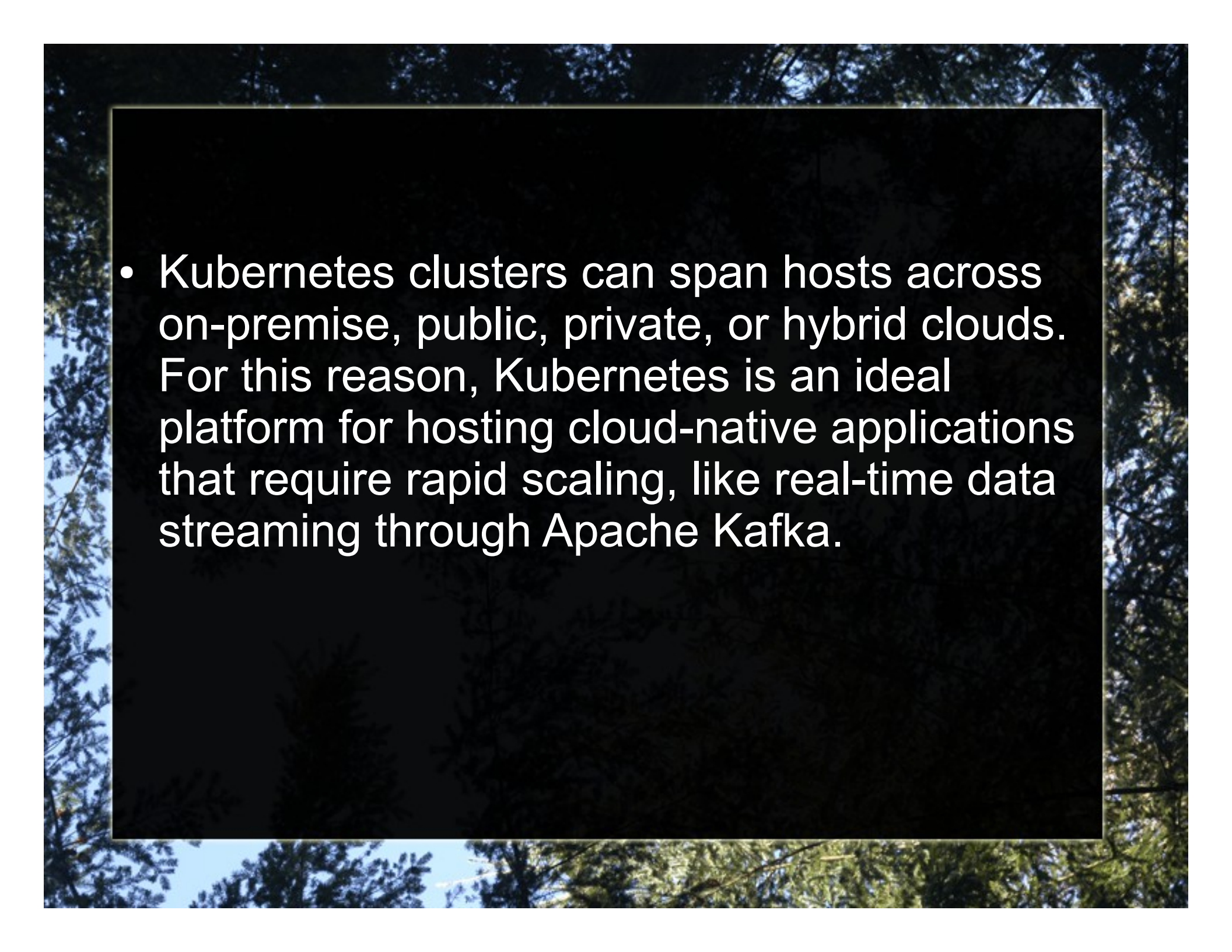
Login Succeeded

```
C:\WINDOWS\system32>docker images
```



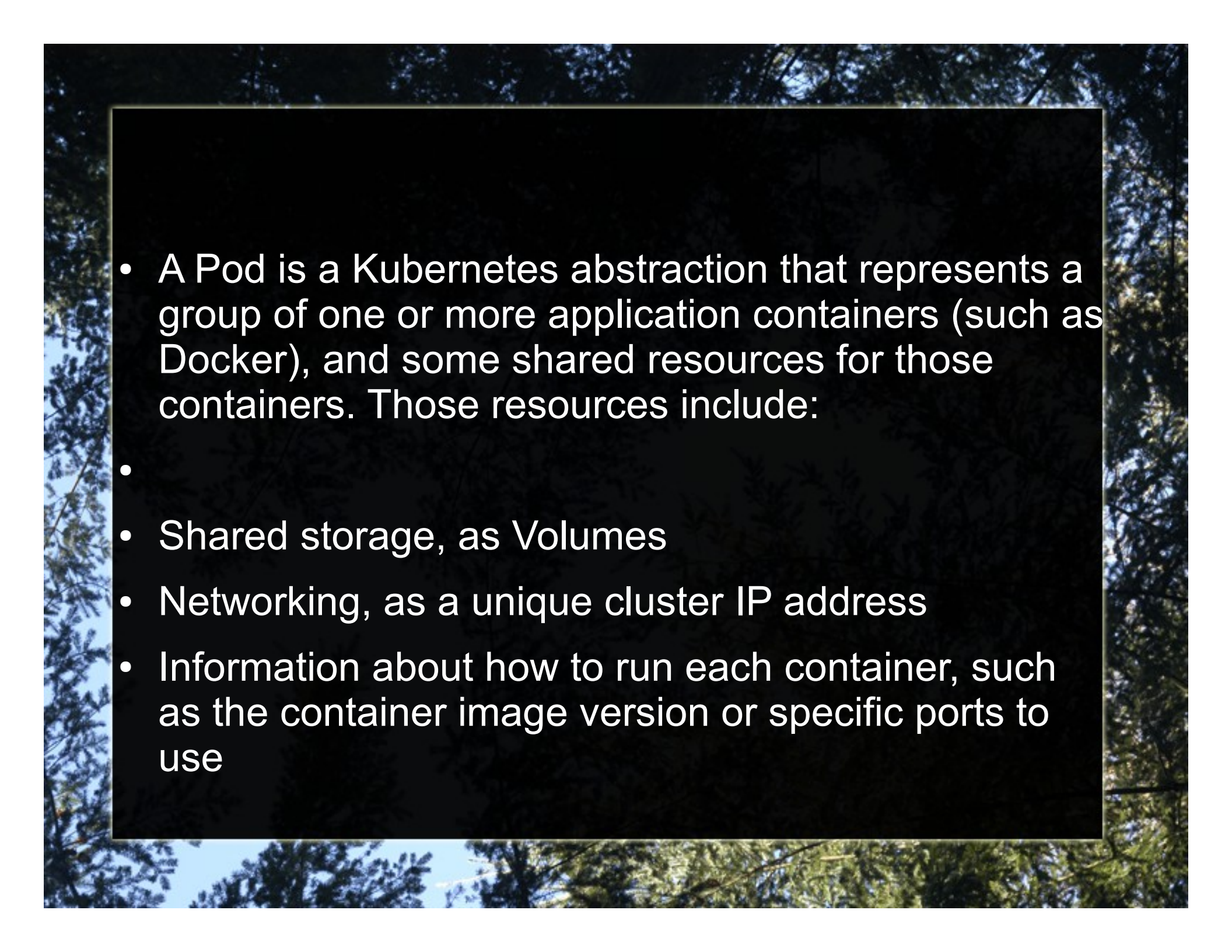
Kubernetes

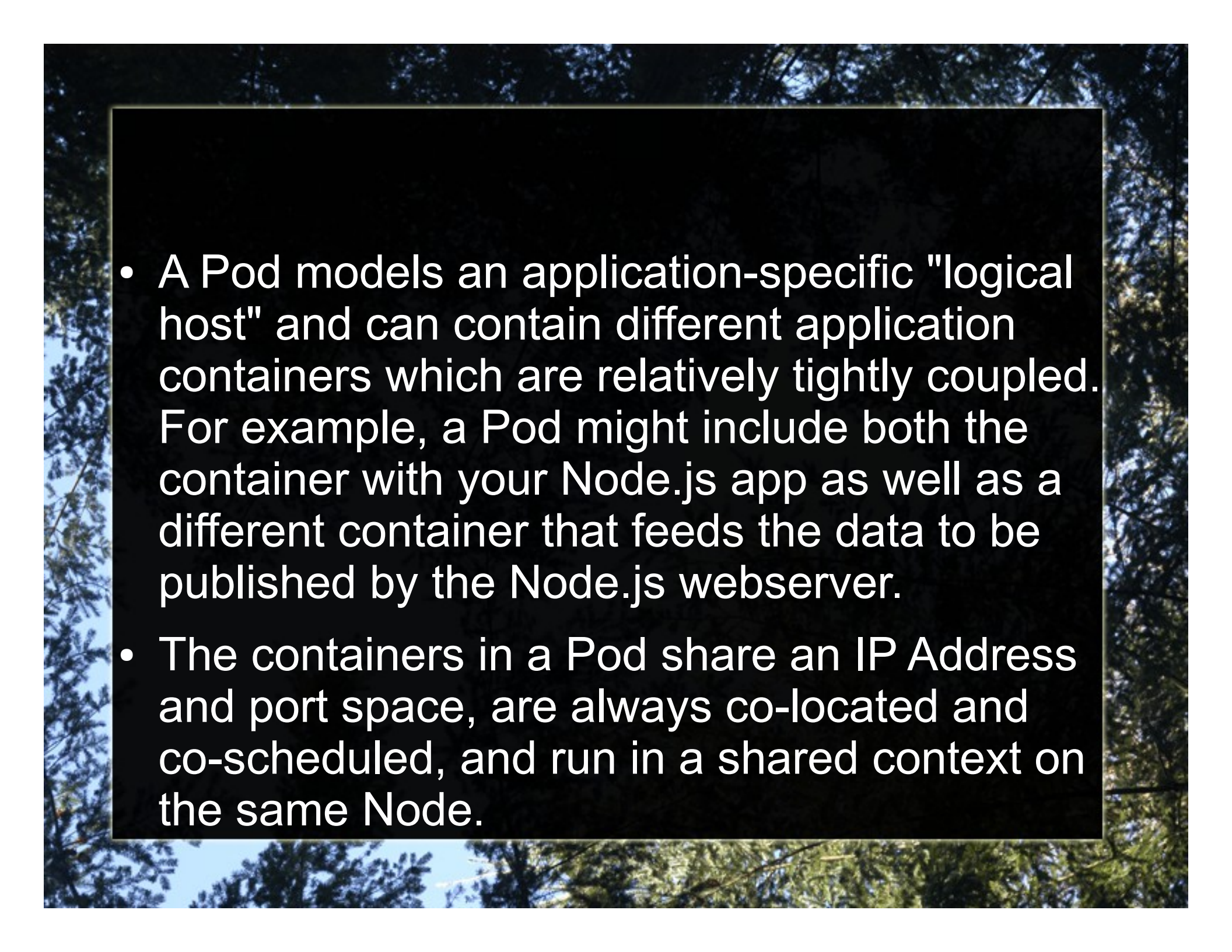
- Kubernetes is a portable, extensible, open-source platform for managing containerized workloads and services, that facilitates both declarative configuration and automation. It has a large, rapidly growing ecosystem. Kubernetes services, support, and tools are widely available.
- The name Kubernetes originates from Greek, meaning helmsman or pilot. K8s as an abbreviation results from counting the eight letters between the "K" and the "s". Google open-sourced the Kubernetes project in 2014. Kubernetes combines over 15 years of Google's experience running production workloads at scale with best-of-breed ideas and practices from the community
- <https://kubernetes.io/docs/concepts/overview/what-is-kubernetes/>

- 
- Kubernetes clusters can span hosts across on-premise, public, private, or hybrid clouds. For this reason, Kubernetes is an ideal platform for hosting cloud-native applications that require rapid scaling, like real-time data streaming through Apache Kafka.

POD

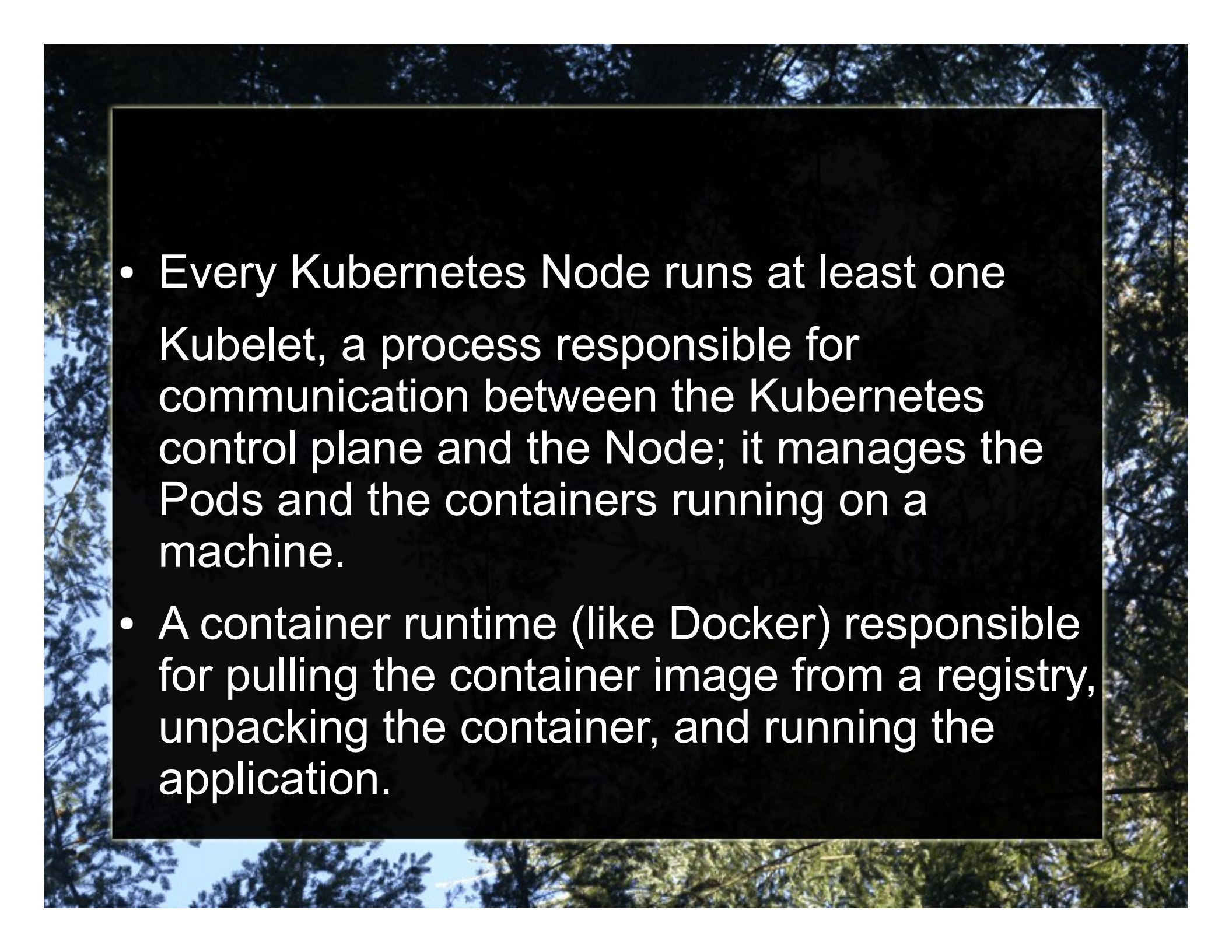
- A pod is the smallest execution unit in Kubernetes. A pod encapsulates one or more applications. Pods are ephemeral by nature, if a pod (or the node it executes on) fails, Kubernetes can automatically create a new replica of that pod to continue operations. Pods include one or more containers (such as Docker containers).

- 
- A Pod is a Kubernetes abstraction that represents a group of one or more application containers (such as Docker), and some shared resources for those containers. Those resources include:
 -
 - Shared storage, as Volumes
 - Networking, as a unique cluster IP address
 - Information about how to run each container, such as the container image version or specific ports to use

- 
- The background of the slide is a photograph of a dense forest with tall trees and green foliage. A large, dark, semi-transparent rectangular box is centered over the image, containing the text. The text is white and uses a sans-serif font.
- A Pod models an application-specific "logical host" and can contain different application containers which are relatively tightly coupled. For example, a Pod might include both the container with your Node.js app as well as a different container that feeds the data to be published by the Node.js webserver.
 - The containers in a Pod share an IP Address and port space, are always co-located and co-scheduled, and run in a shared context on the same Node.

Nodes

- A Pod always runs on a Node. A Node is a worker machine in Kubernetes and may be either a virtual or a physical machine, depending on the cluster. Each Node is managed by the control plane. A Node can have multiple pods, and the Kubernetes control plane automatically handles scheduling the pods across the Nodes in the cluster. The control plane's automatic scheduling takes into account the available resources on each Node.

- 
- Every Kubernetes Node runs at least one Kubelet, a process responsible for communication between the Kubernetes control plane and the Node; it manages the Pods and the containers running on a machine.
 - A container runtime (like Docker) responsible for pulling the container image from a registry, unpacking the container, and running the application.

Using MiniKube

- We can use MiniKube and kubectl for creating kubernetes cluster
- If you already have kubectl installed, you can now use it to access your cluster
- Minikube can be started using minikube start command
- A dashboard for Kubernetes is available after using minikube dashboard command

View nodes

- Nodes active in Kubernetes nodes can be viewed using
- Kubectl get nodes
-

Deploying an image to kubernetes

- `kubectl create deployment hello-minikube --image=k8s.gcr.io/echoserver:1.4`
-

Exposing a deployment

- `kubectl expose deployment hello-minikube --type=NodePort --port=8080`

Accessing app through port forwarding

- `kubectrl port-forward service/hello-minikube 7080:8080`

-