Docker, Kubernetes, Helm

into DevOps methodologies

devops book

Passionate thinkers with the tiny little obsession of making business scalable and profitable, in energy and beyond

Contacts: feel free contact us via LinkedIn

Mariano Alesci

DevOps Architect presso BaxEnergy

Lorenzo Vetrano

Scrum Master and Agile Coach

Passionate thinkers with the tiny little obsession of making business scalable and profitable, in energy and beyond

Introduction to Virtualization Technologies

Docker: virtualization concepts

Docker technology

is one implementation of container based virtualization technologies

Docker: virtualization concepts

Pre-Virtualization World

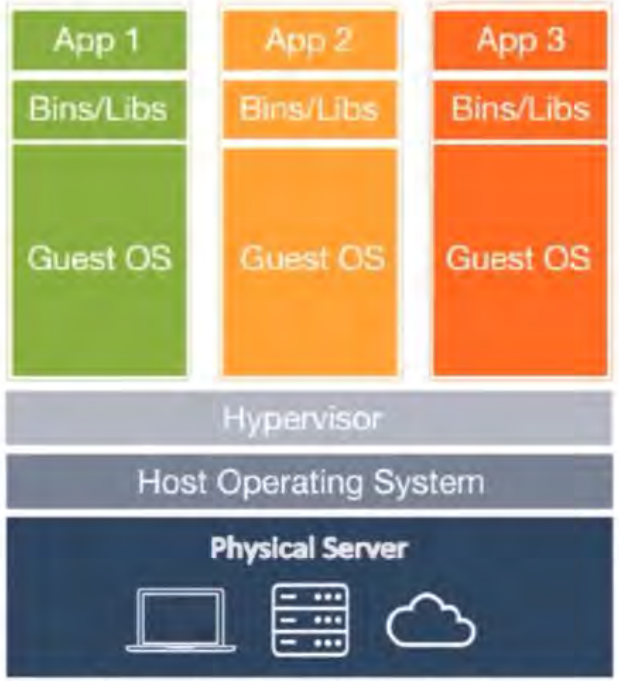
Pre-Virtualization

Problems:

- Huge Cost
- Slow Deployment
- Hard to Migrate

Docker: virtualization concepts

Hypervisor-based Virtualization



Hypervisor-based Virtualization

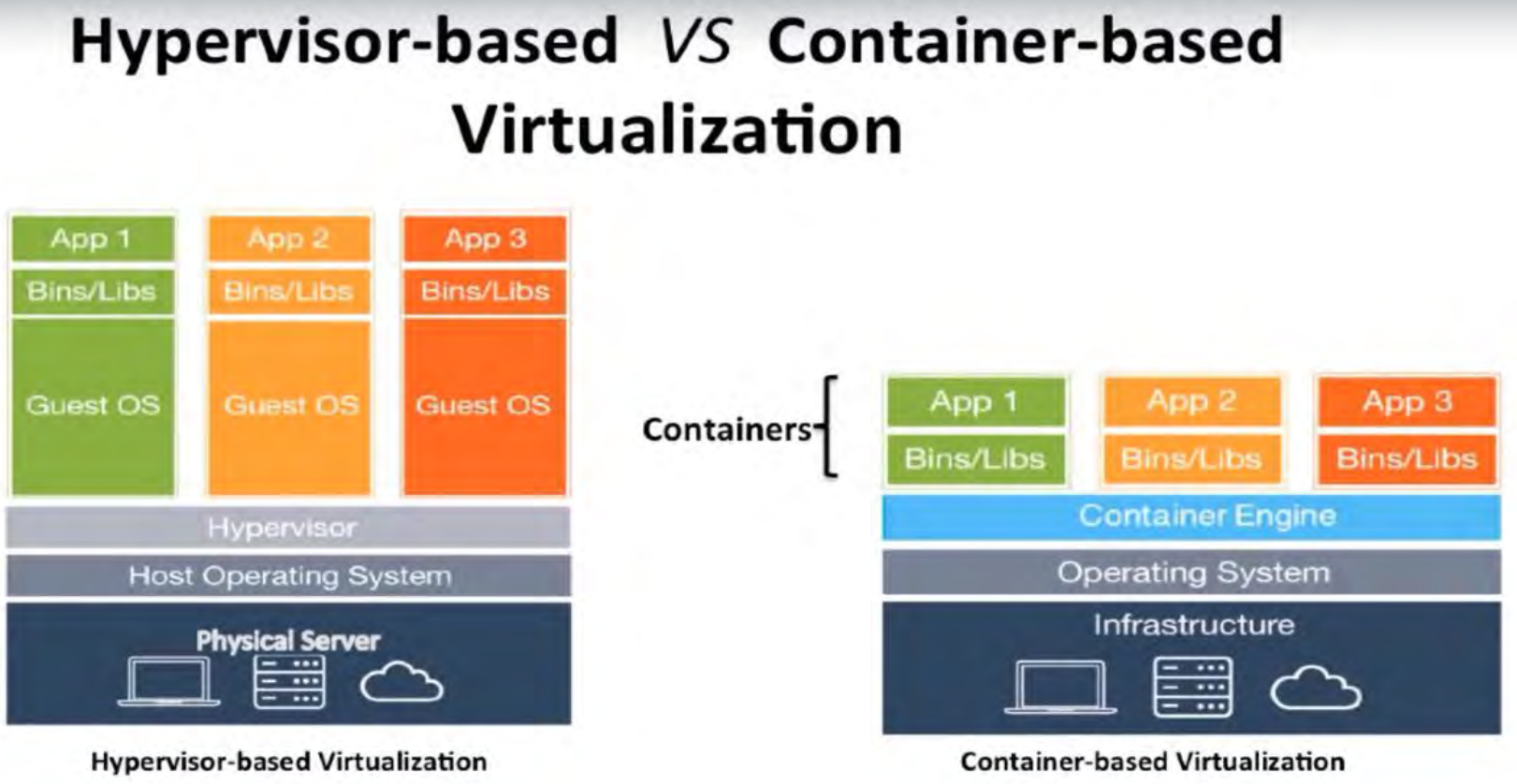
Benefits:

- Cost-Efficient
- Easy to Scale

Limitations:

- Kernel Resource Duplication
- Application Portability Issue

Docker: virtualization concepts

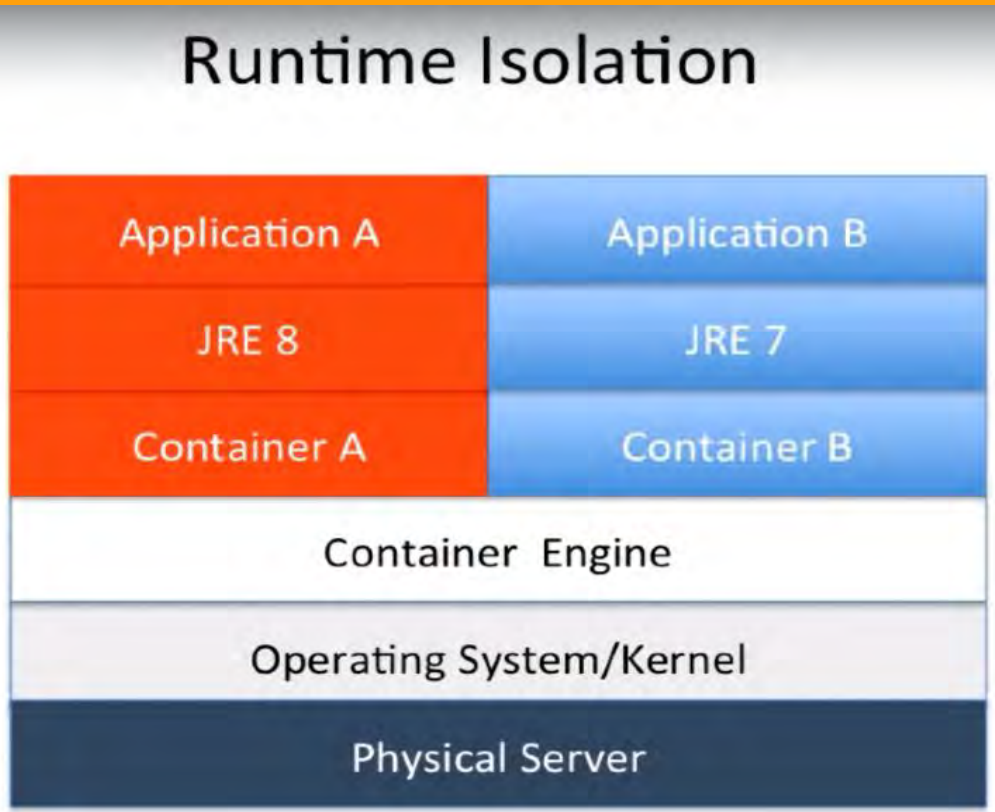


Docker: virtualization concepts

Runtime Isolation

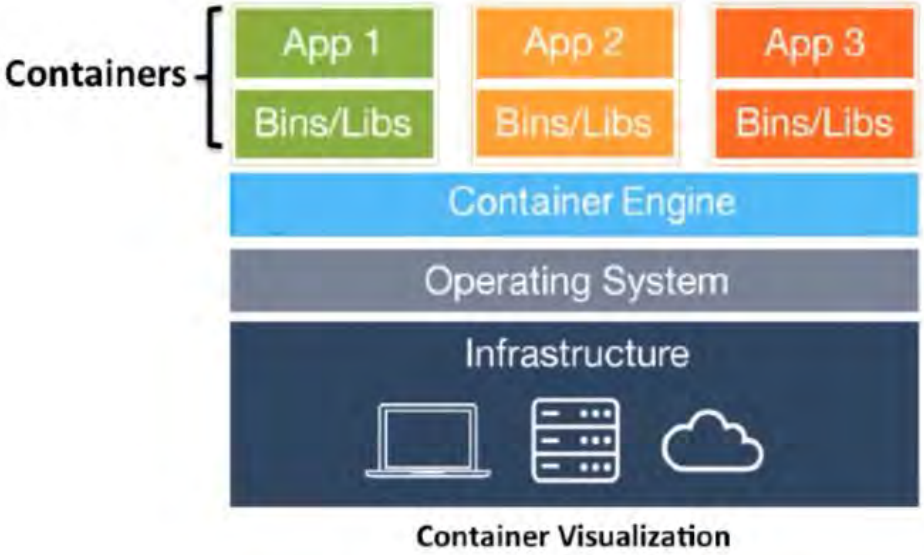
How to run two different Java applications with two different JREs on the same VM?

Docker: virtualization concepts



Docker: virtualization concepts

Container Virtualization

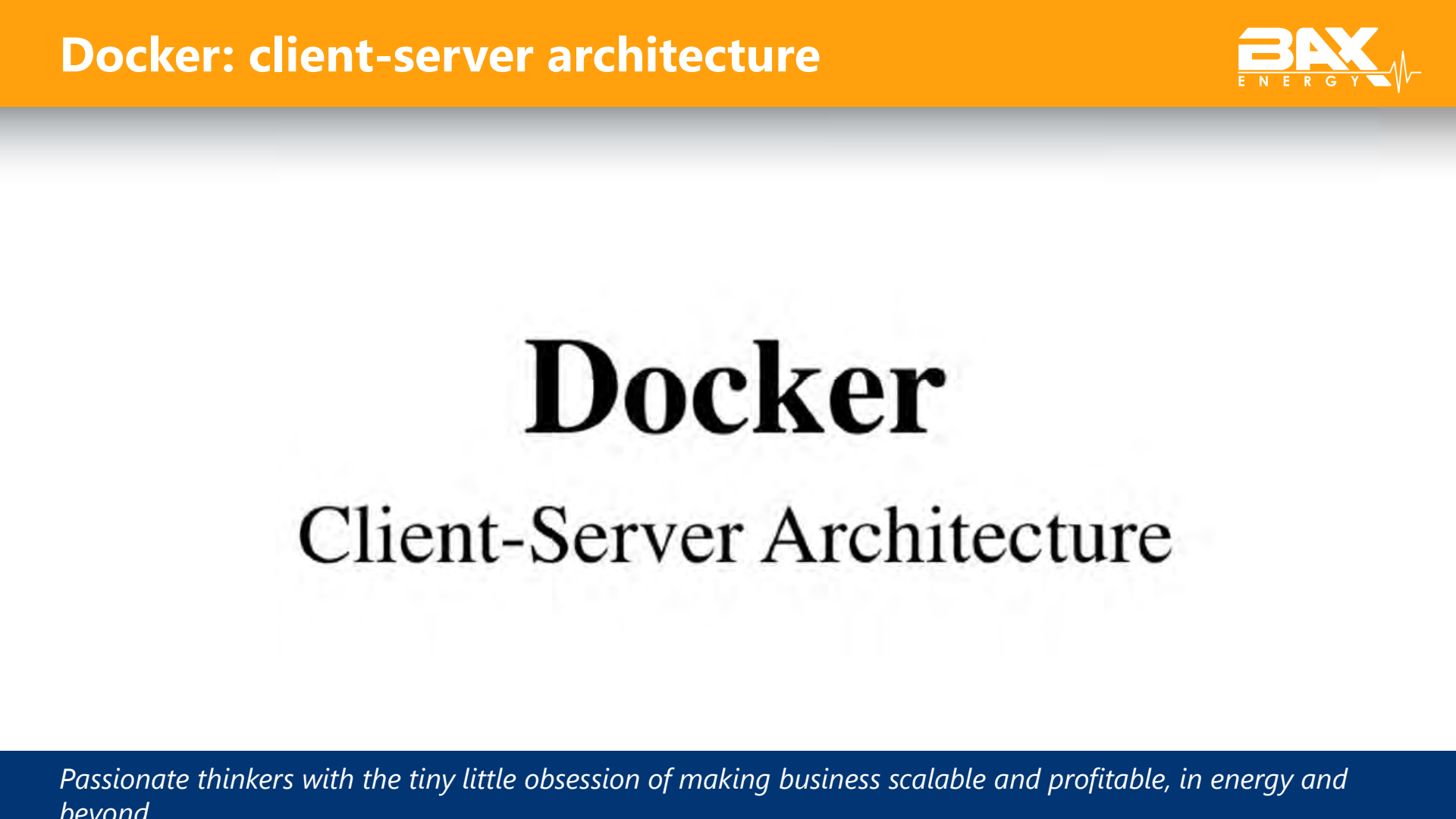


Container Visualization

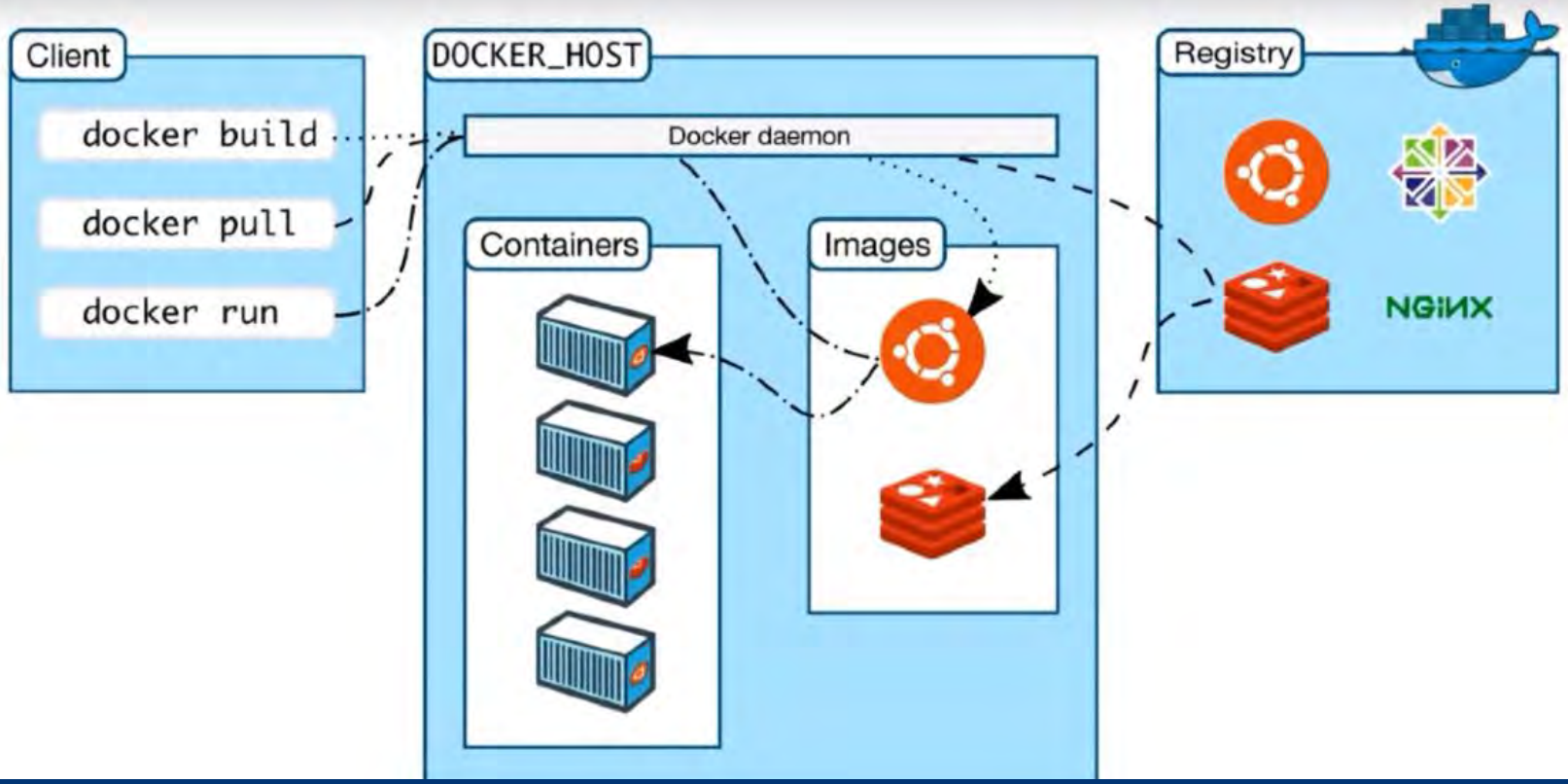
Benefits:

- Cost-Efficient
- Fast Deployment
- Guaranteed Portability

Docker: client-server architecture

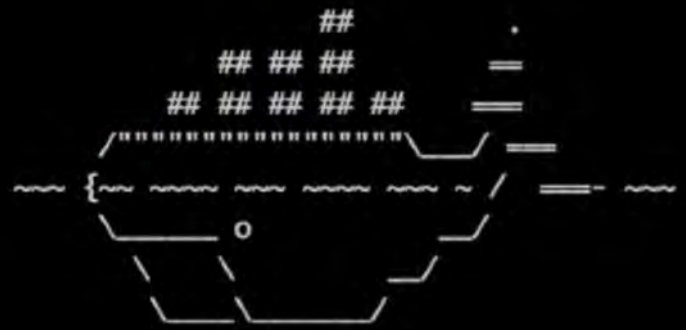


Docker: client-server architecture



Docker: client-server architecture

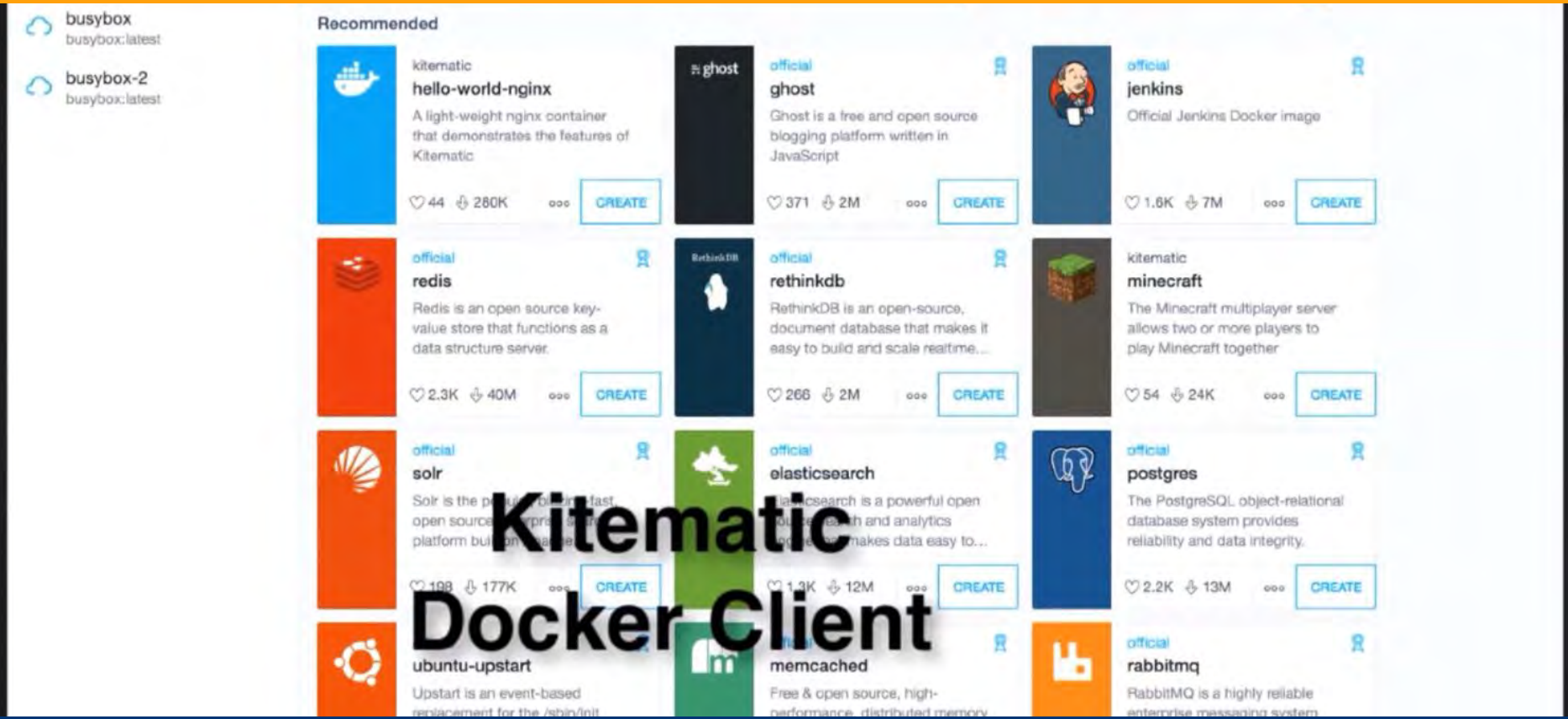
```
bash --login '/Applications/Docker/Docker Quickstart Terminal.app/Contents/Resources/Scripts/start.sh'
cwei69-mac:~ cwei$ bash --login '/Applications/Docker/Docker Quickstart Terminal.app/Contents/Resources/
```



Command Line
Docker Client

```
docker is configured to use the default machine with IP 192.168.99.100
For help getting started, check out the docs at https://docs.docker.com

cwei69-mac:~ cwei$
```

Docker: client-server architecture



Passionate thinkers with the tiny little obsession of making business scalable and profitable, in energy and beyond

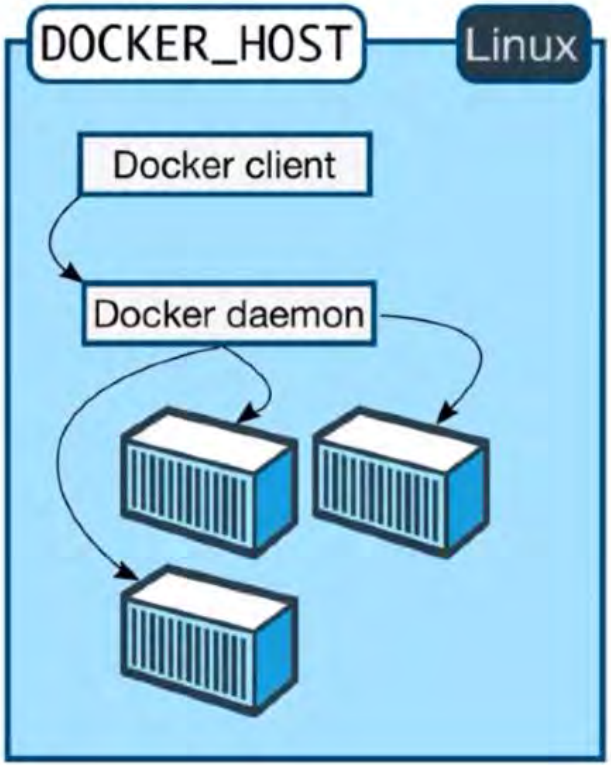
Docker: client-server architecture

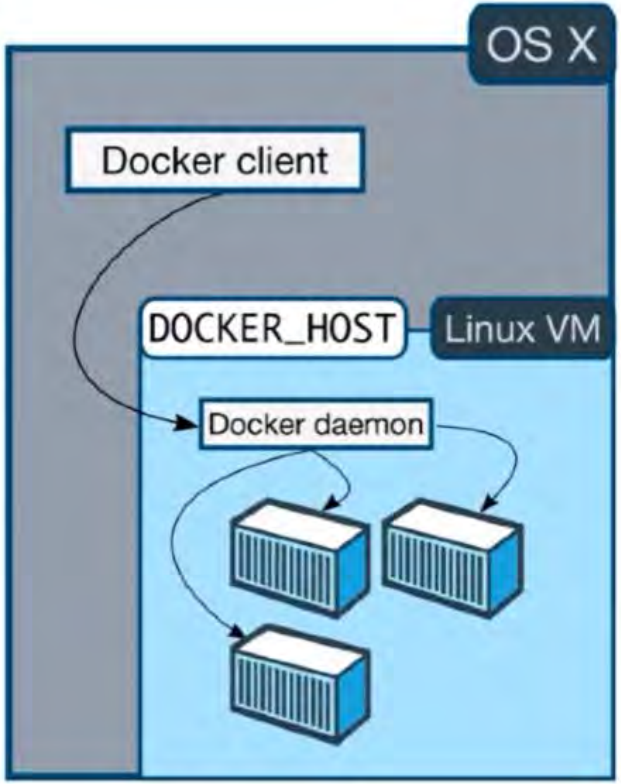
Docker Daemon

Docker Engine

Docker Server

Docker: client-server architecture





Docker: installation

Install Docker for Mac/Windows

Docker: installation

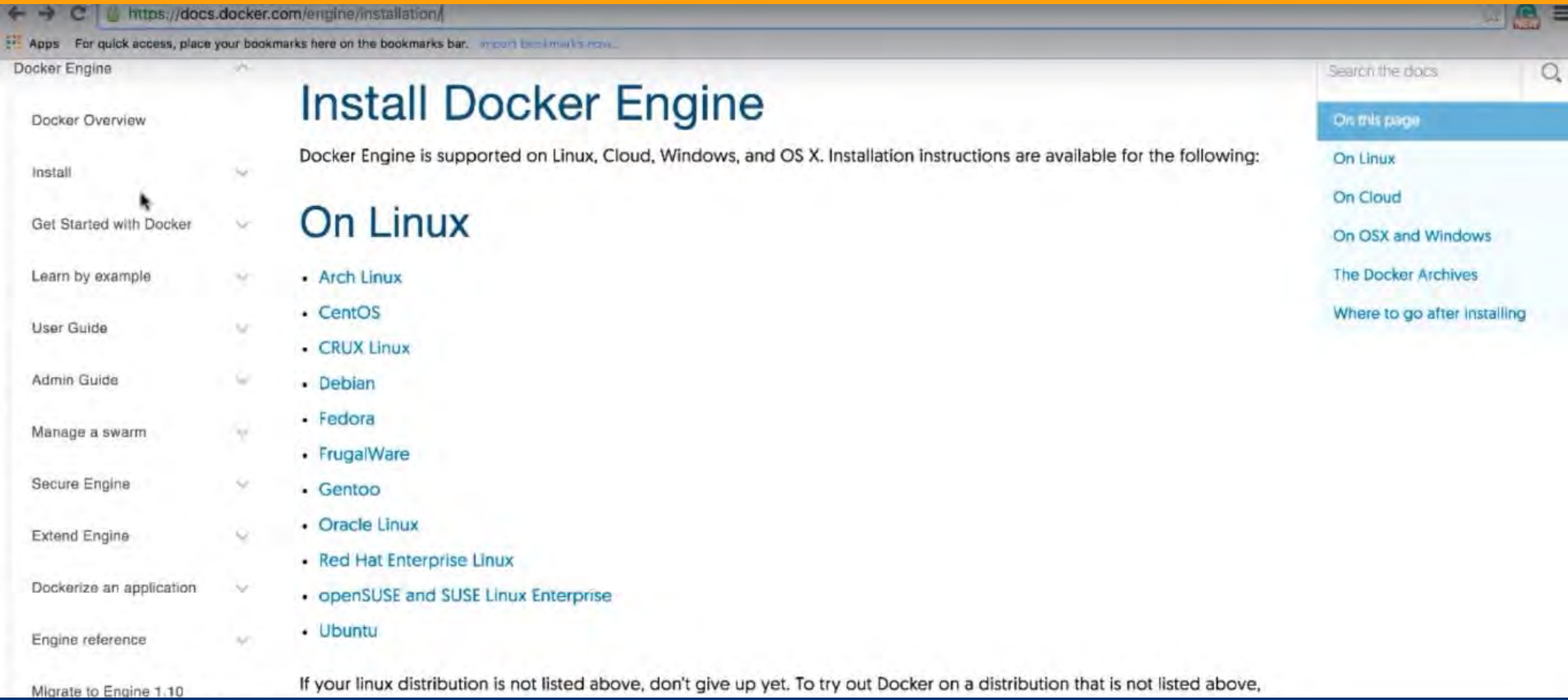
Install Docker

This lecture applies to you if:

- You are using Linux
- Or you are using Mac and your Mac version is OS X 10.10.3 or newer
- Or you are using Windows and your Windows version is Windows 10 or newer

Otherwise, you can skip this lecture and follow the installation guide of the next lecture.

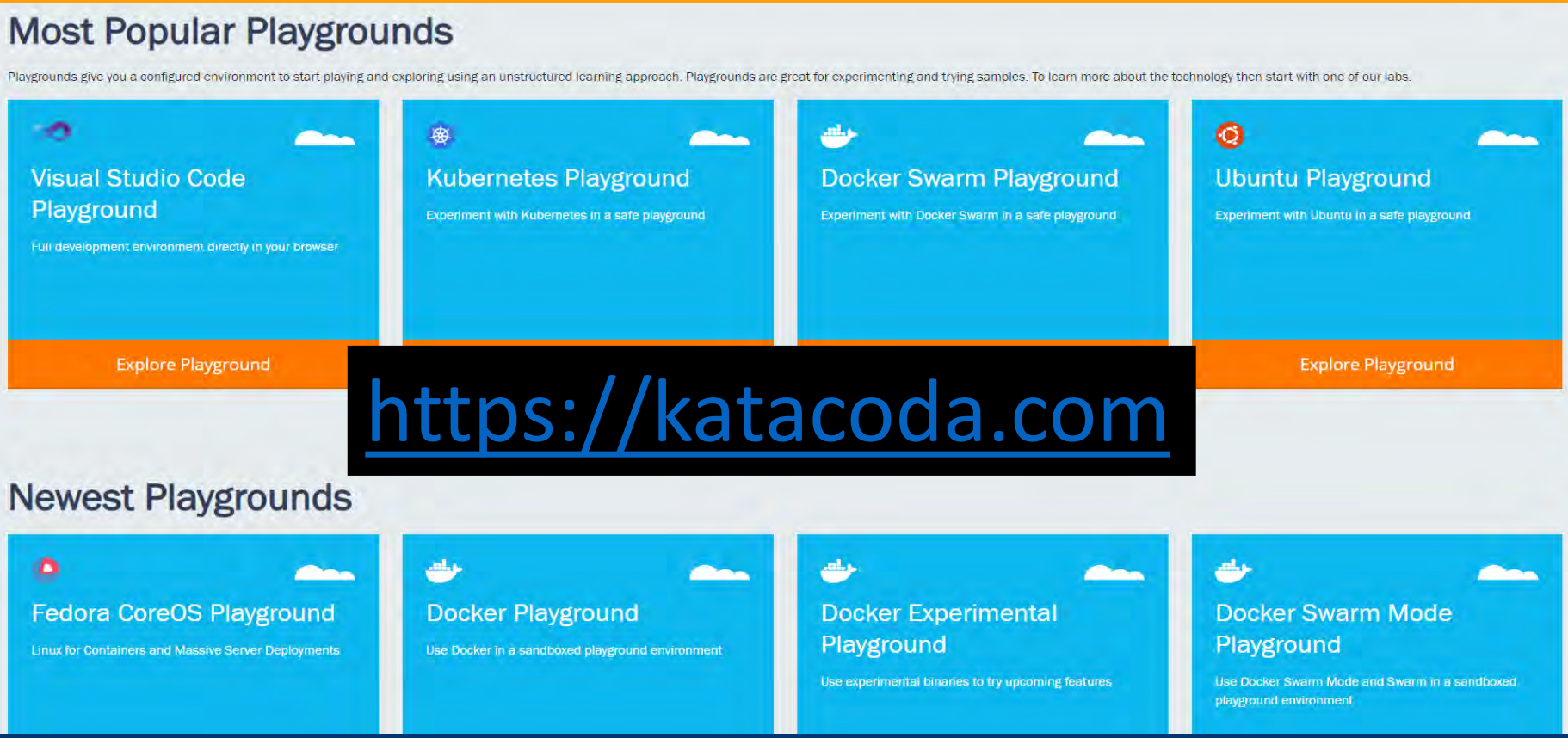
Docker: installation



Docker: installation

Docker info command

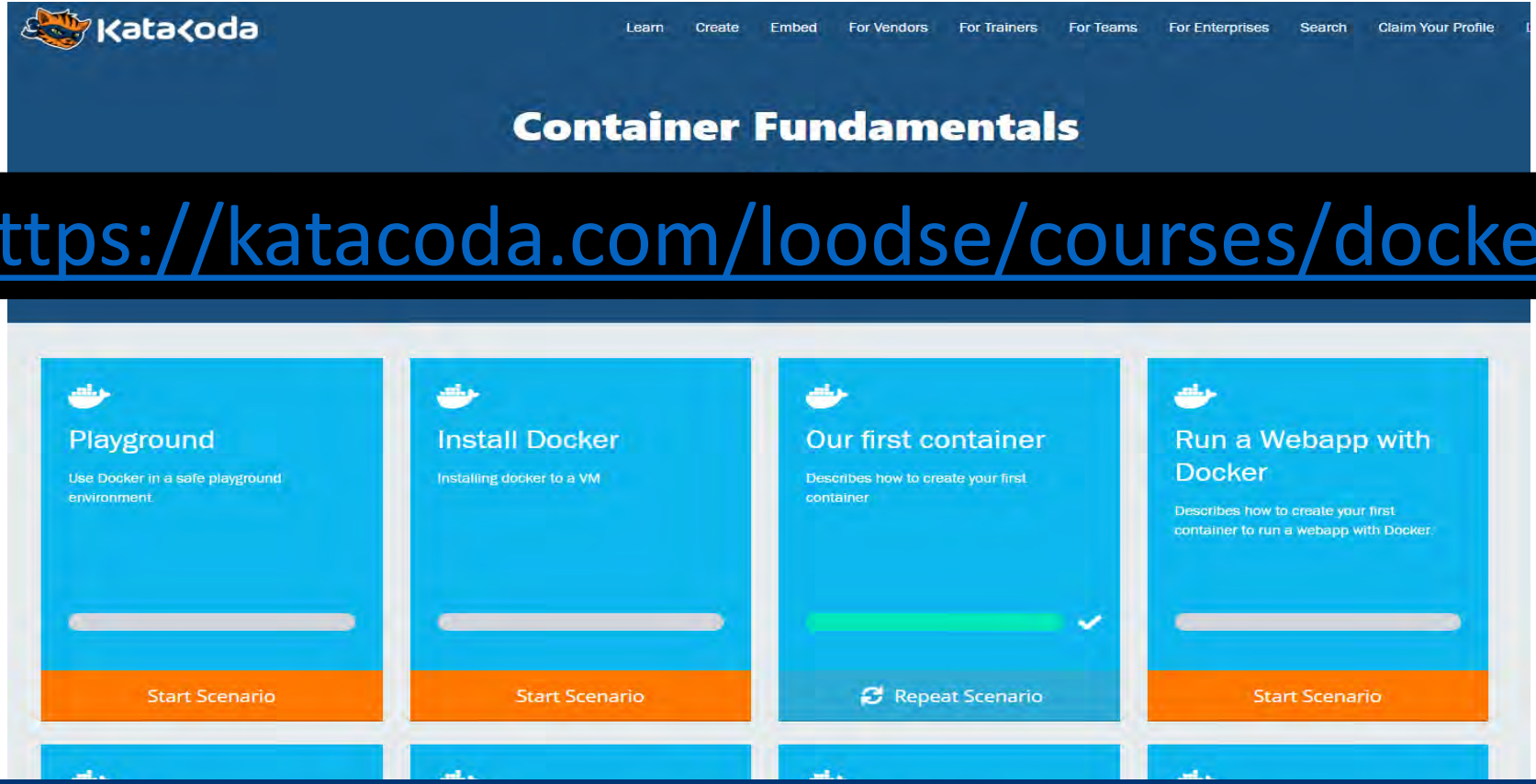
Docker: installation



Docker: first container

Run our First Hello World Docker Container

Docker: our first container



Docker: concepts

Important

Docker Concepts

Images

- Images are read only templates used to create containers.
- Images are created with the docker build command, either by us or by other docker users.
- Images are composed of layers of other images.
- Images are stored in a Docker registry.

Containers

- If an image is a class, then a container is an instance of a class - a runtime object.
- Containers are lightweight and portable encapsulations of an environment in which to run applications.
- Containers are created from images. Inside a container, it has all the binaries and dependencies needed to run the application.

Registries and Repositories

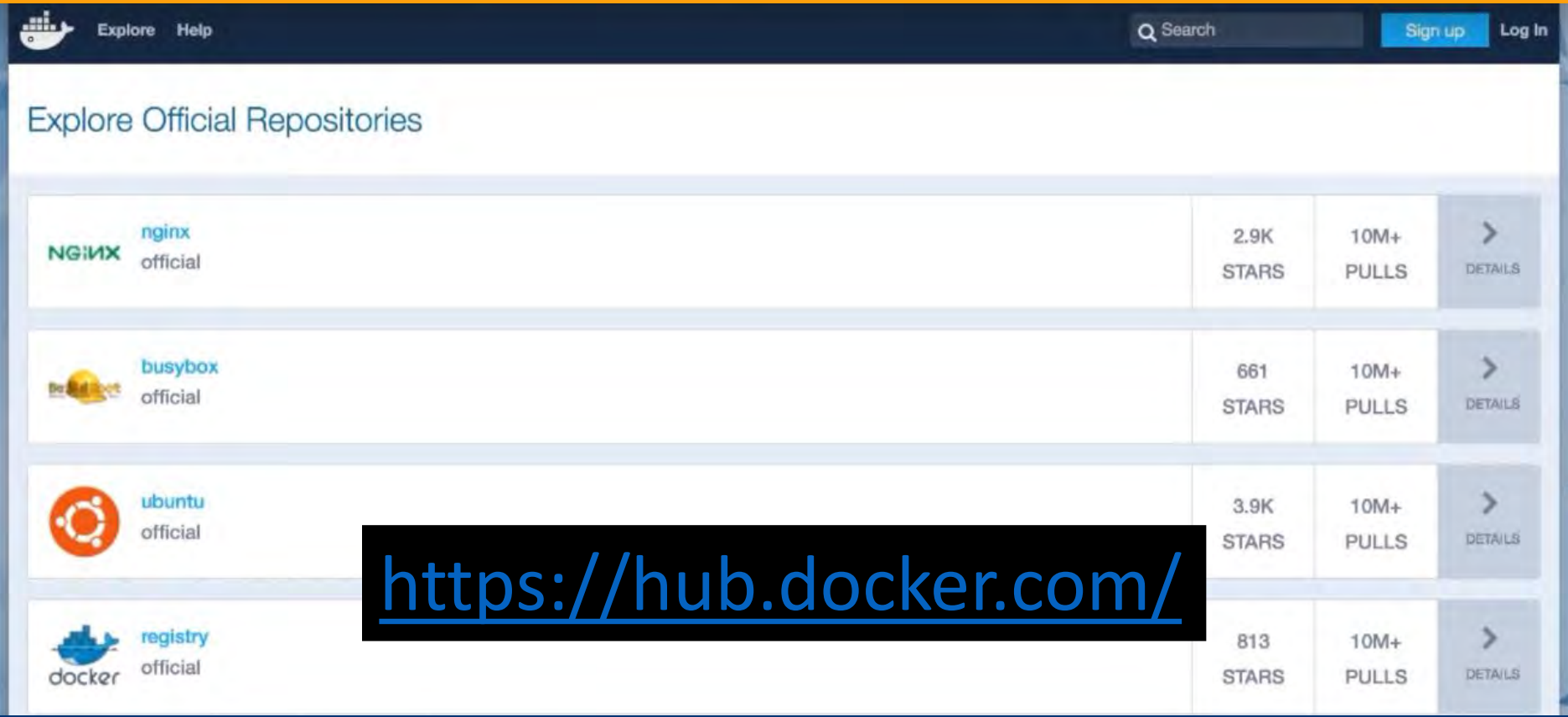
- A registry is where we store our images.
- You can host your own registry, or you can use Docker's public registry which is called DockerHub.
- Inside a registry, images are stored in repositories.
- Docker repository is a collection of different docker images with the same name, that have different tags, each tag usually represents a different version of the image.

Docker: concepts

Docker Hub

A Public Docker Repository

Docker: concepts



Deep Dive into Docker Containers

- running containers in detached mode
- docker ps command
- docker container name
- docker inspect command

Foreground vs Detached

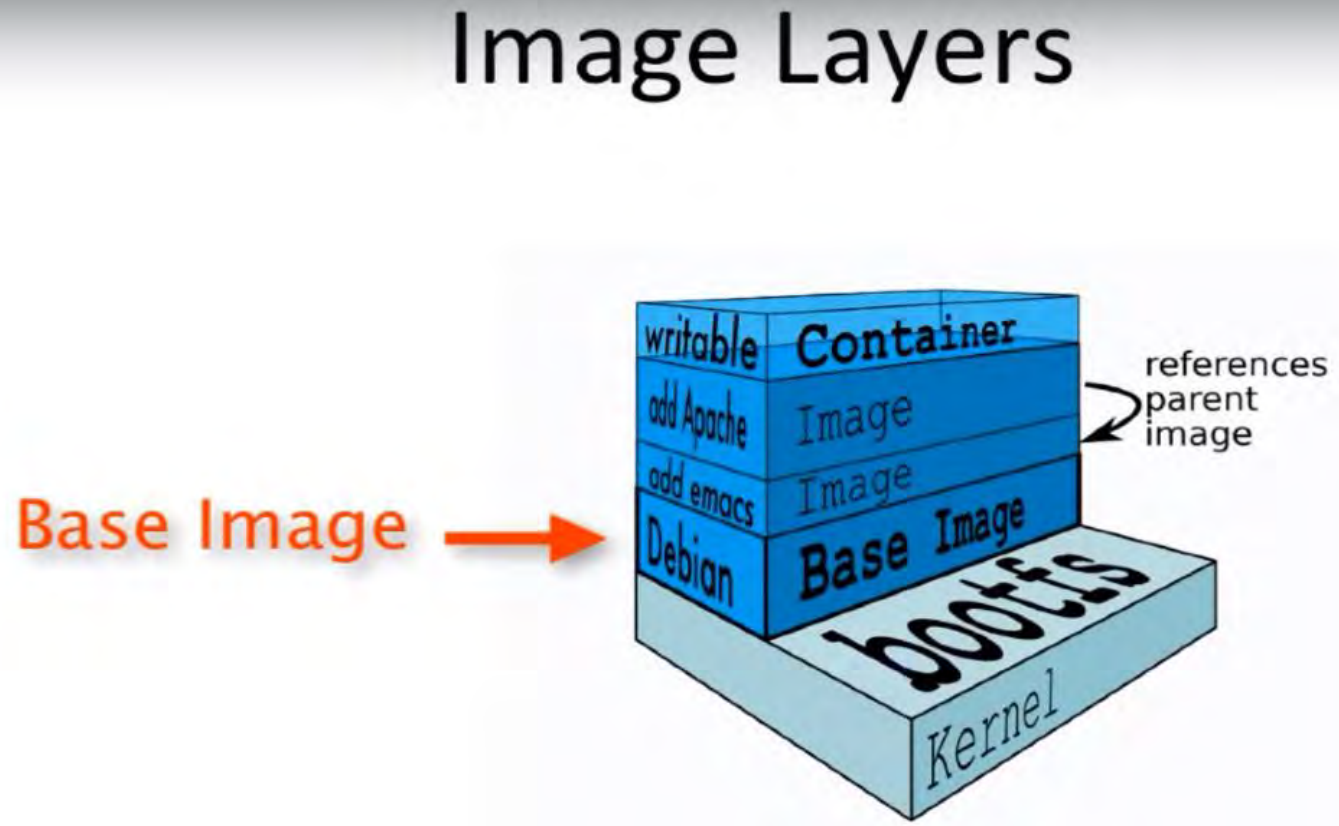
	Run Container in Foreground	Run Container in Background
Description	Docker run starts the process in the container and attaches the console to the process's standard input, output, and standard error.	Containers started in detached mode and exit when the root process used to run the container exits.
How to specify?	default mode	-d option
Can the console be used for other commands after the container is started up?	No	Yes

Docker: interacting with containers

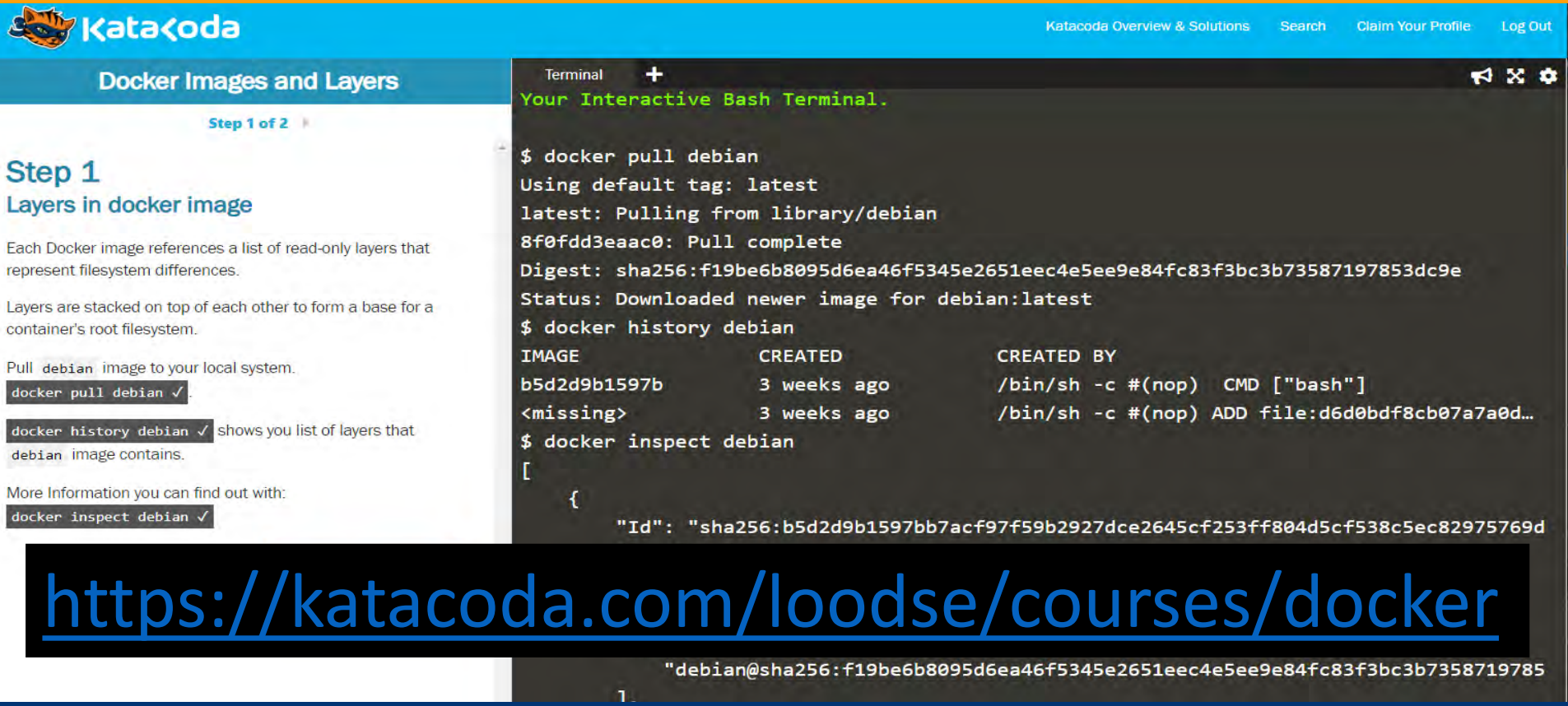
Docker: image layers

Docker Image Layers

Docker: image layers



Docker: image layers

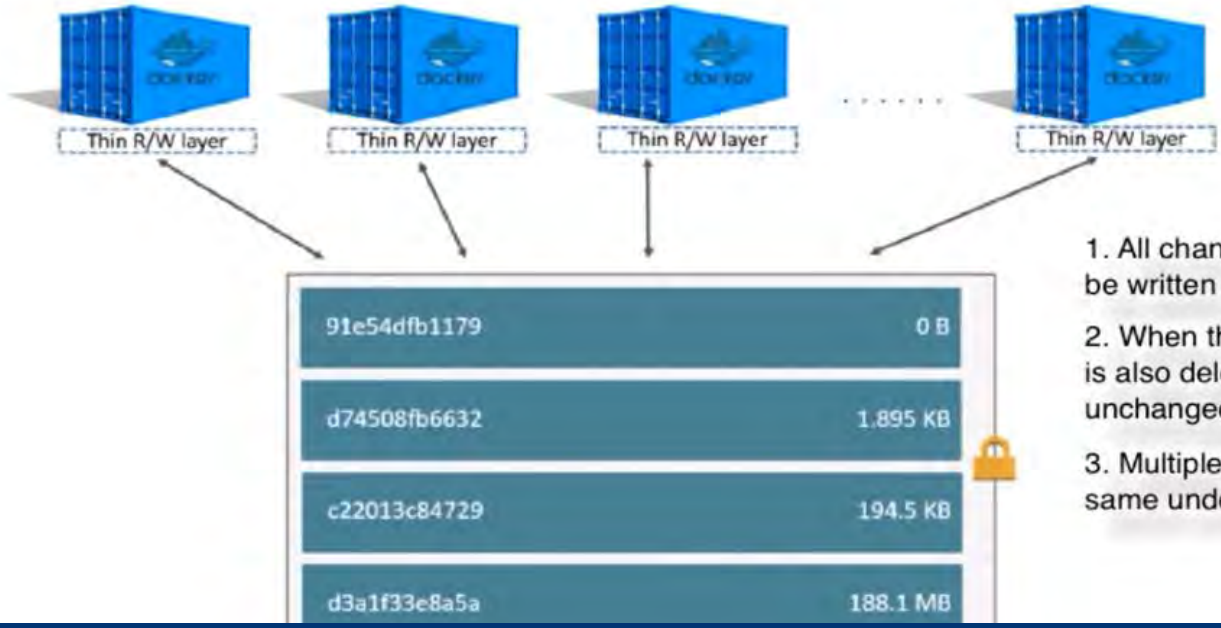


https://katacoda.com/loodse/courses/docker

Passionate thinkers with the tiny little obsession of making business scalable and profitable, in energy and beyond

Docker: image layers

Image Layers



1. All changes made into the running containers will be written into the writable layer.
2. When the container is deleted, the writable layer is also deleted, but the underlying image remains unchanged.
3. Multiple containers can share access to the same underlying image.

Docker: build image

Ways to Build a Docker Image

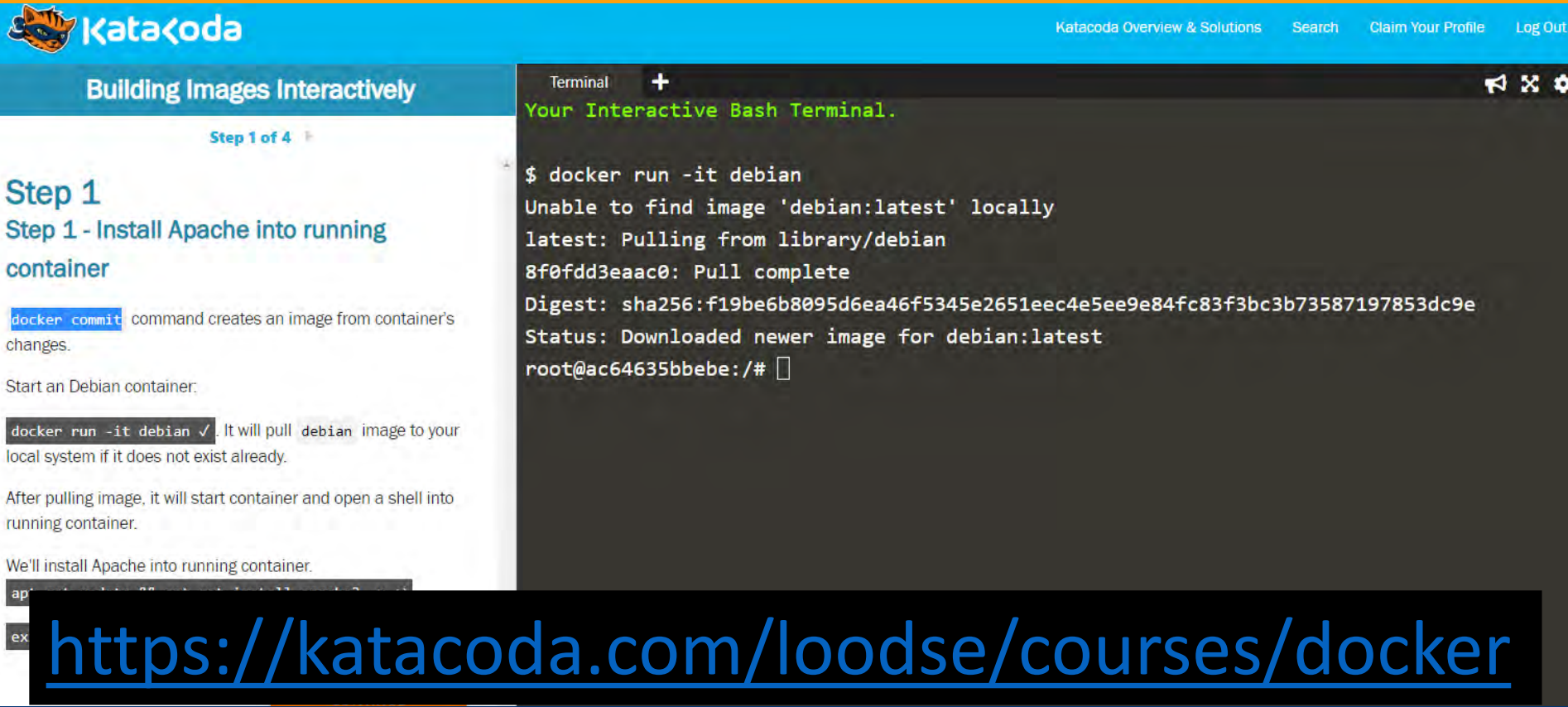
- Commit changes made in a Docker container.
- Write a Dockerfile.

Docker: build image

Build Docker Images

Approach 1: committing changes made in a container

Docker: build image interactively



https://katacoda.com/loodse/courses/docker

Passionate thinkers with the tiny little obsession of making business scalable and profitable, in energy and beyond

Docker: build image interactively

Steps

1. Spin up a container from a base image.
2. Install a package in the container.
3. Commit changes made in the container.

Docker: build image interactively

Docker commit

- Docker commit command would save the changes we made to the Docker container's file system to a new image.

docker commit container_ID repository_name:tag

Docker: build image

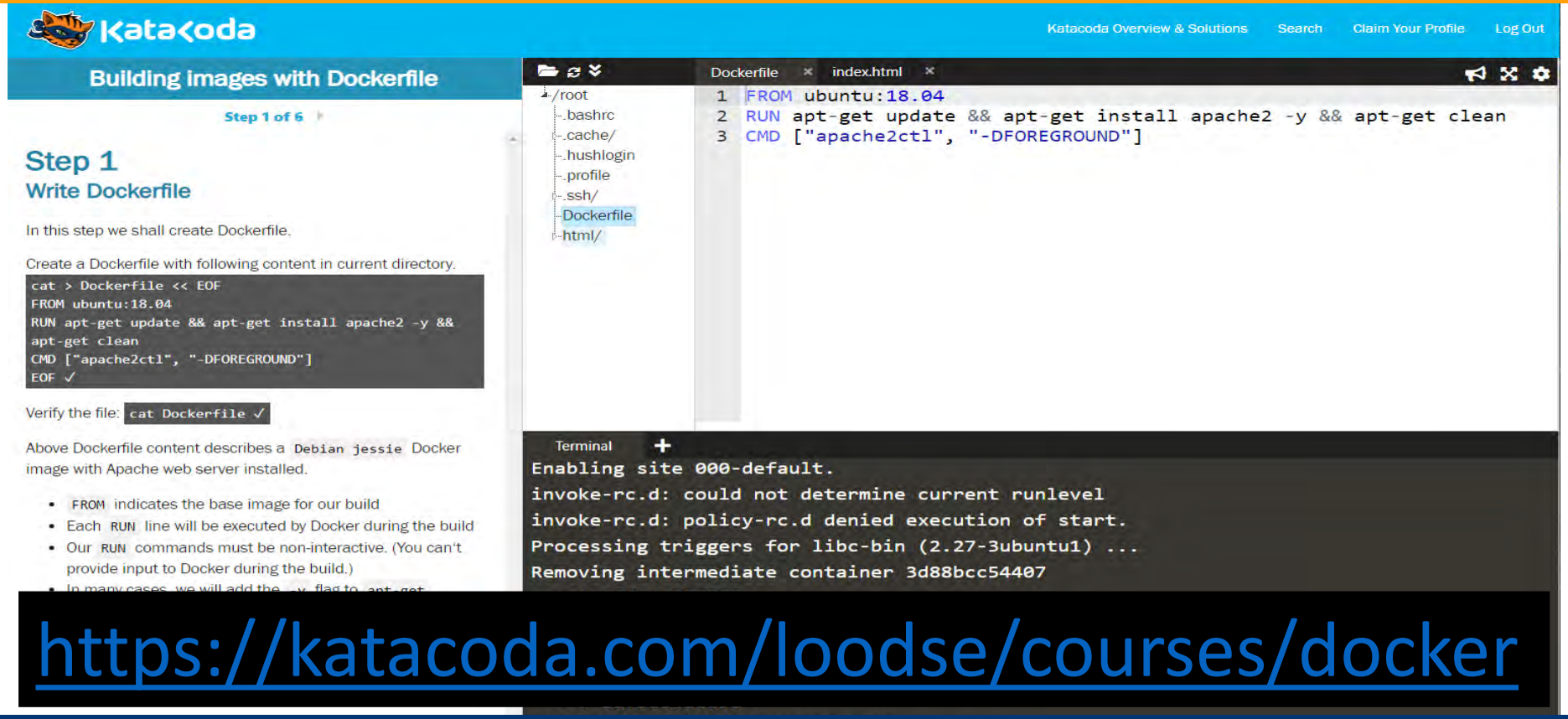
Build Docker Images

Approach 2: Writing a Dockerfile

Dockerfile and Instructions

- A Dockerfile is a text document that contains all the instructions users provide to assemble an image.
- Each instruction will create a new image layer to the image.
- Instructions specify what to do when building the image.

Docker: building images with Dockerfile



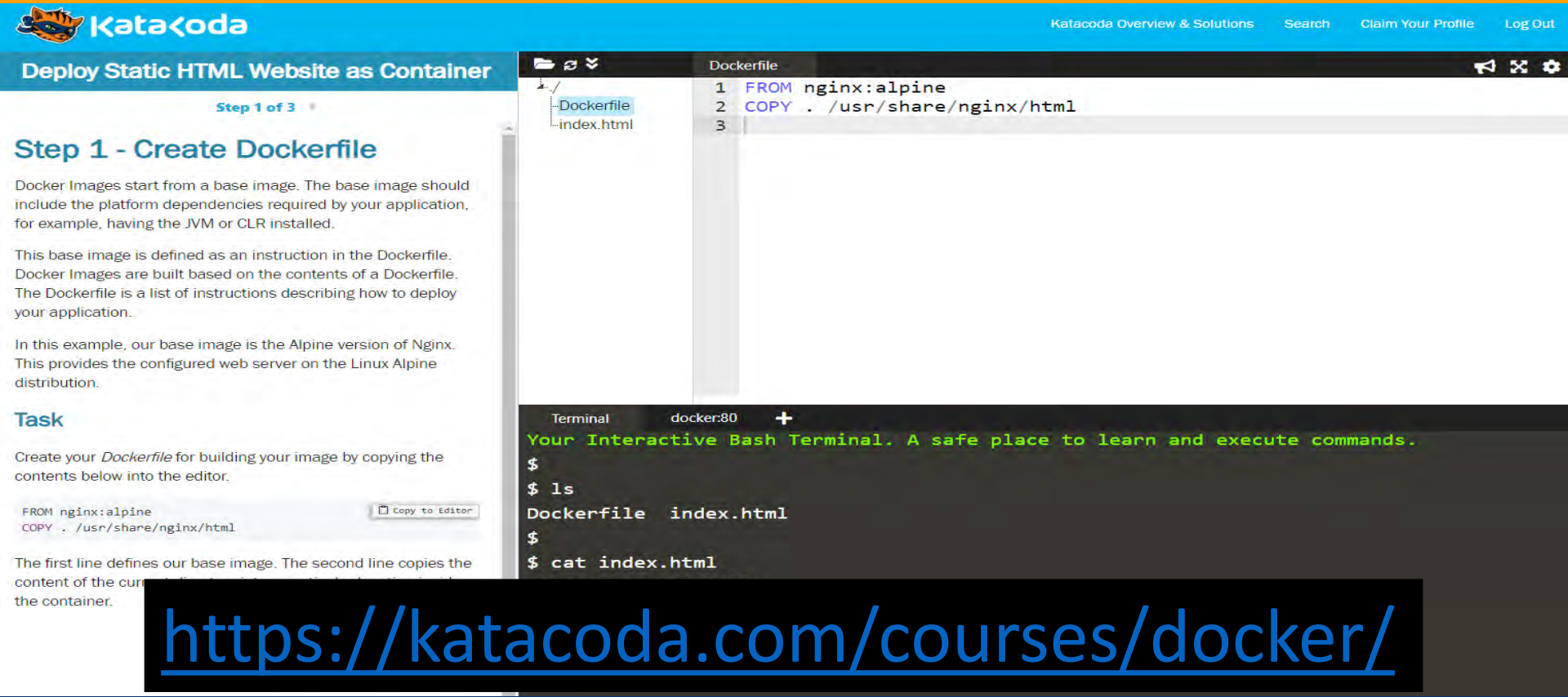
https://katacoda.com/loodse/courses/docker

Passionate thinkers with the tiny little obsession of making business scalable and profitable, in energy and beyond

Docker: run a webapp with docker

Dockerize a Hello World Web Application

Docker: dockerize HTML website



https://katacoda.com/courses/docker/

Passionate thinkers with the tiny little obsession of making business scalable and profitable, in energy and beyond

K8S: introduction

MASTER KUBERNETES

INTRODUCTION

Passionate thinkers with the tiny little obsession of making business scalable and profitable, in energy and beyond

K8S: introduction

Passionate thinkers with the tiny little obsession of making business scalable and profitable, in energy and beyond

K8S: introduction

Passionate thinkers with the tiny little obsession of making business scalable and profitable, in energy and beyond

K8S: introduction

WHAT & WHY - CONTAINERS, ORCHESTRATION

	CONTAINERS	CONTAINER ORCHESTRATION
FUNCTION	KEEP SOFTWARE SEPARATED INTO ITS OWN "CLEAN" VIEW OF AN OPERATING SYSTEM	DEFINE RELATIONSHIPS BETWEEN CONTAINERS, WHERE THEY COME FROM, HOW THEY SCALE, AND HOW THEY CONNECT TO THE WORLD AROUND THEM
PREDECESSORS/ ALTERNATIVES	•VIRTUAL MACHINES •DIRECT INSTALLATION	•HOMEGROWN SCRIPTS •MANUAL, BESPOKE STATIC CONFIGURATION BETWEEN CONTAINERS
PACKAGES/VENDORS	•DOCKER •RKT •GARDEN •LXC •MESOS	•KUBERNETES •DOCKER SWARM •AMAZON ECS •MESOS

K8S: introduction

THE CONTAINER "WAR" MAYBE COMING TO A CLOSE

- Docker is by-and-far the leader
- Defines:
  - A Container Format (Dockerfile)
  - Registry to host Containers (public and private)

K8S: introduction

THE ORCHESTRA(S) - ORCHESTRATION TECHNOLOGY

- A much newer space
- Kubernetes has quite a lead, and emerging as the clear leader & winner, it's closest competitor, Docker Swarm, recently announced support for Kubernetes-style configuration & compatibility
- Cooperation & formal standards bodies are more prevalent in the growth of orchestration

K8S: introduction

K8S: overview

Passionate thinkers with the tiny little obsession of making business scalable and profitable, in energy and beyond

K8S: overview

KUBERNETES

- Born from a Google internal project in mid-2014 (Google "Borg")
- 1.0 Release in July 2015
- Google partnered with the Linux Foundation to form the Cloud Native Computing Foundation (CNCF) to offer Kubernetes as an open standard
- Frequently abbreviated "k8s" - Greek for "helmsman" or "pilot"

K8S: overview

K8S: overview

MAKING KUBERNETES YOURS

- Kubernetes runs anywhere Linux does
  - Your Laptop
  - Globally distributed data centers
  - Major cloud providers
  - Anywhere in between & nearly any combination (not that you'd want to necessarily)

K8S: overview

Passionate thinkers with the tiny little obsession of making business scalable and profitable, in energy and beyond

K8S: overview

WHERE DOES K8S LIVE?

- Anywhere you need it to
- But, where would you *want* it to live?
- That depends on who you are & what you're using it for
  - Evaluation/Training - minikube (local version of k8s)
  - Development - minikube, dev cluster on a cloud provider
  - Deployment - cloud provider or bare metal

K8S: overview

K8S ON YOUR WORKSTATION (MINIKUBE)

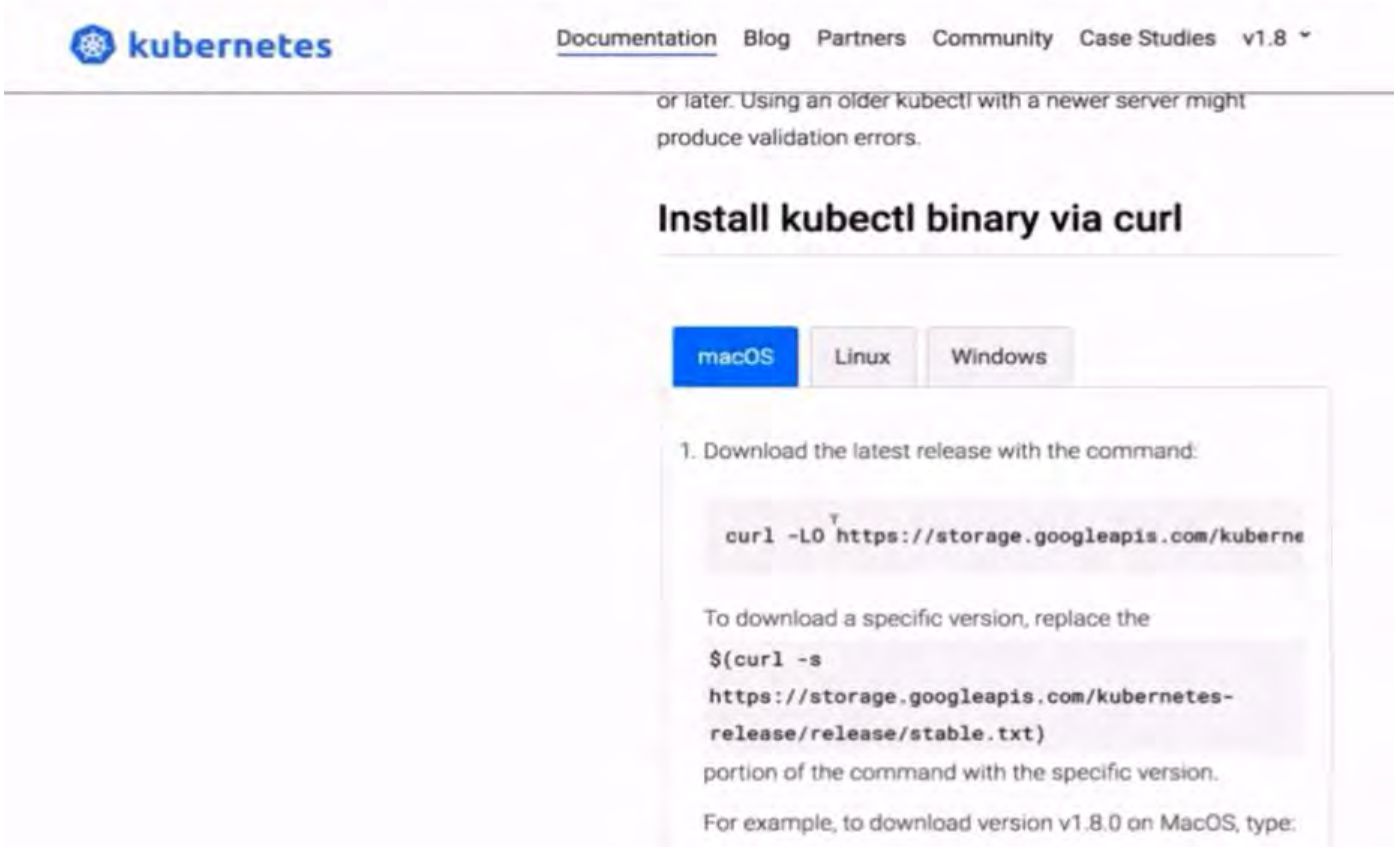
- minikube allows you to run the actual kubernetes code locally on your machine.
  - Avoid the complexity, expense, and slower response of a remote cluster
  - Well supported for development and testing
  - Not a production technology, defeats the purpose of container orchestration in many ways and cannot be relied upon for production workloads.

K8S: overview

A FEW CONCEPTS TO NOTE

- Kubernetes "deployments" are the high-level construct that define an application
- "Pods" are instances of a container in a deployment
- "Services" are endpoints that export ports to the outside world
- You can create, delete, modify, and retrieve information about any of these using the *kubectl* command (as well as the Kubernetes local UI we will show later)

K8S: overview



K8S: overview

TESTING KUBECTL

kubectl version

```
overpriced-strip-of-glass:Downloads basitmustafa$ kubectl version
Client Version: version.Info{Major:"1", Minor:"8", GitVersion:"v1.8.2", GitCommit:"bdaeafa71f6c7c04636251031f93464384d54963", GitTreeState:"clean", BuildDate:"2017-10-24T21:08:42Z", GoVersion:"go1.9.1", Compiler:"gc", Platform:"darwin/amd64"}
Server Version: version.Info{Major:"1", Minor:"8+", GitVersion:"v1.8.1-gke.1", GitCommit:"aba494e68a76583d2d7d1b9c97e4a97a19c3a920", GitTreeState:"clean", BuildDate:"2017-10-27T23:54:39Z", GoVersion:"go1.8.3b4", Compiler:"gc", Platform:"linux/amd64"}
```

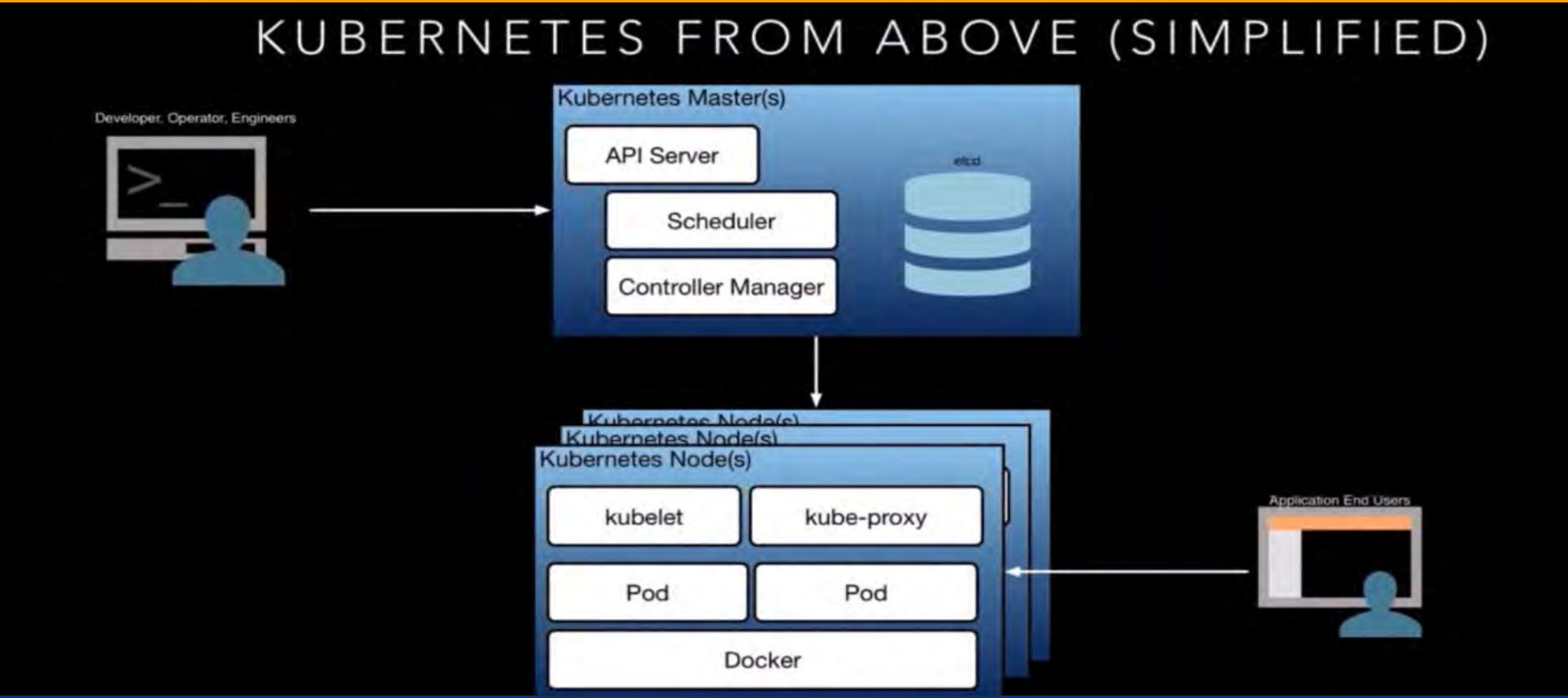
https://katacoda.com/courses/kubernetes/

Passionate thinkers with the tiny little obsession of making business scalable and profitable, in energy and beyond

K8S: architecture overview

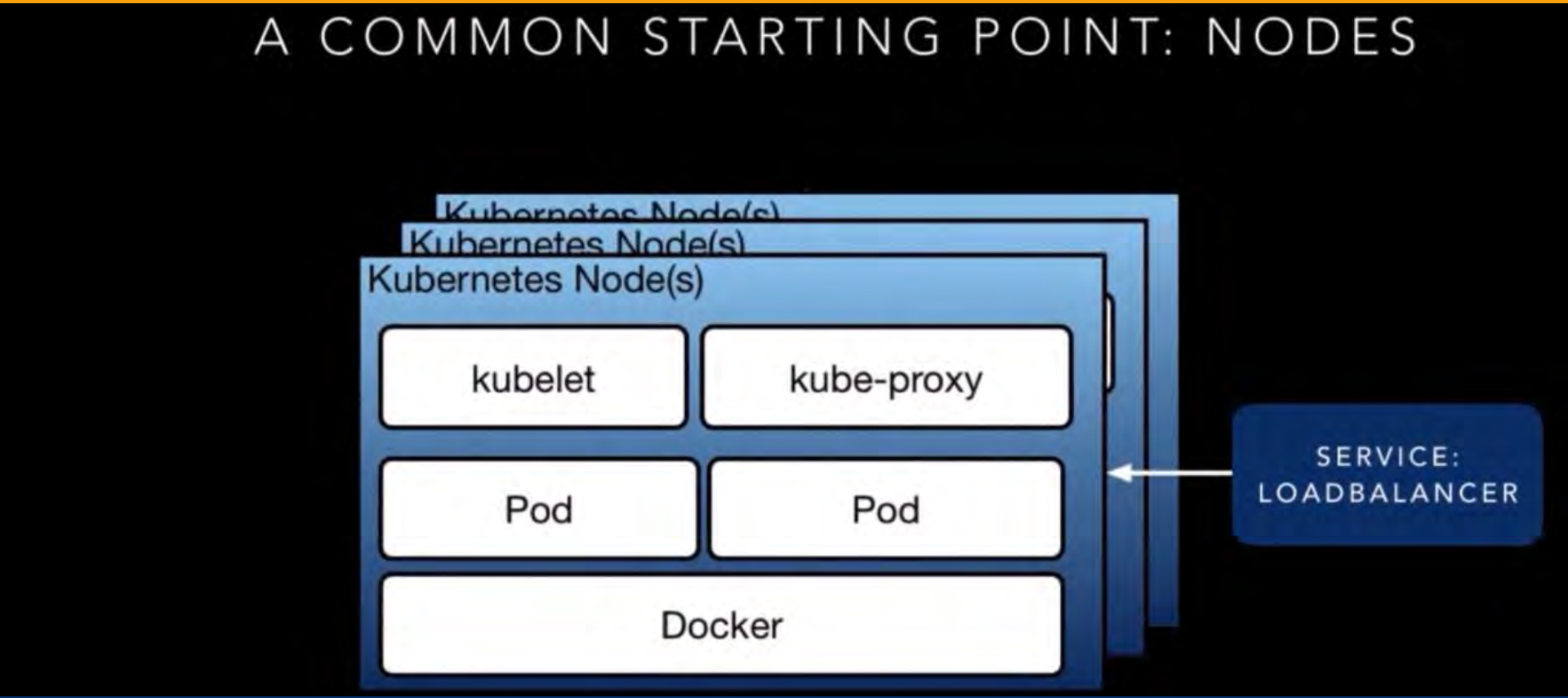
Passionate thinkers with the tiny little obsession of making business scalable and profitable, in energy and beyond

K8S: architecture overview



Passionate thinkers with the tiny little obsession of making business scalable and profitable, in energy and beyond

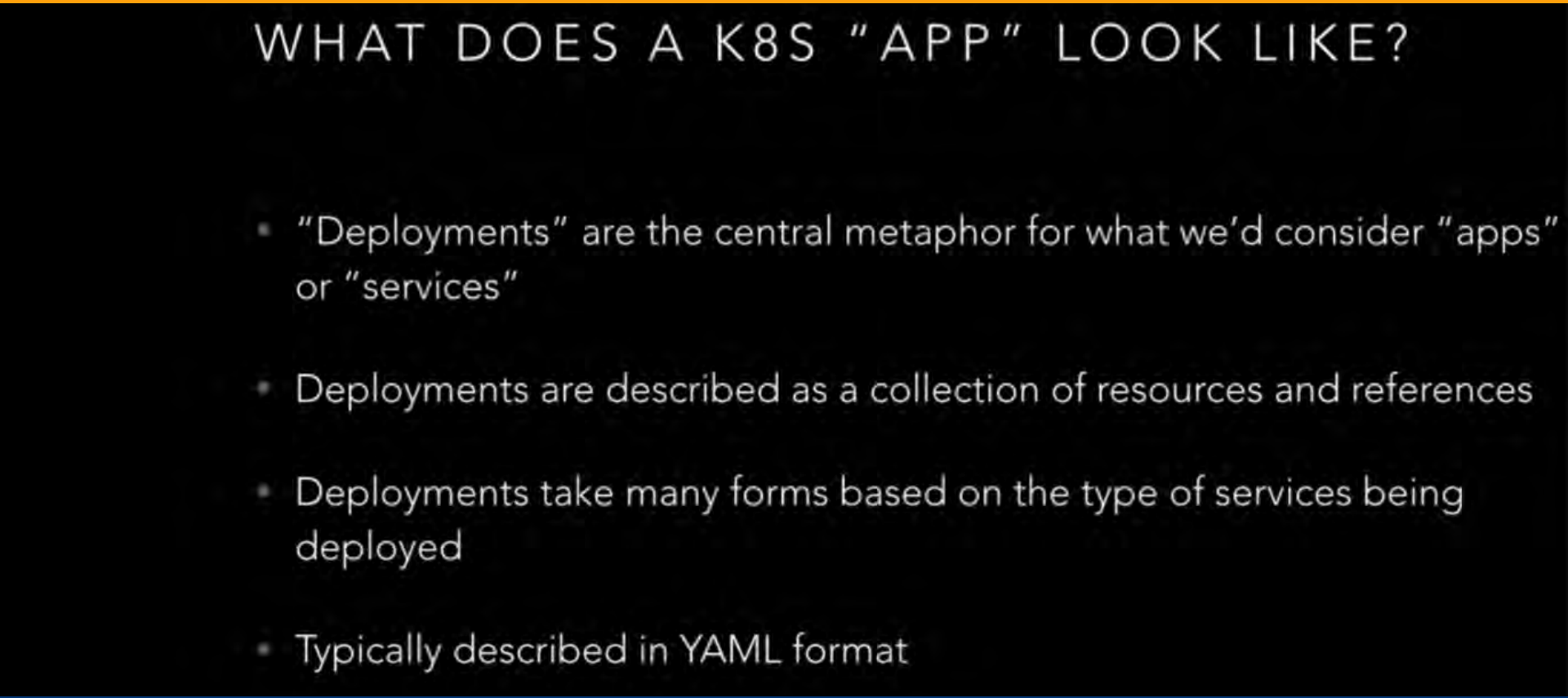
K8S: architecture overview



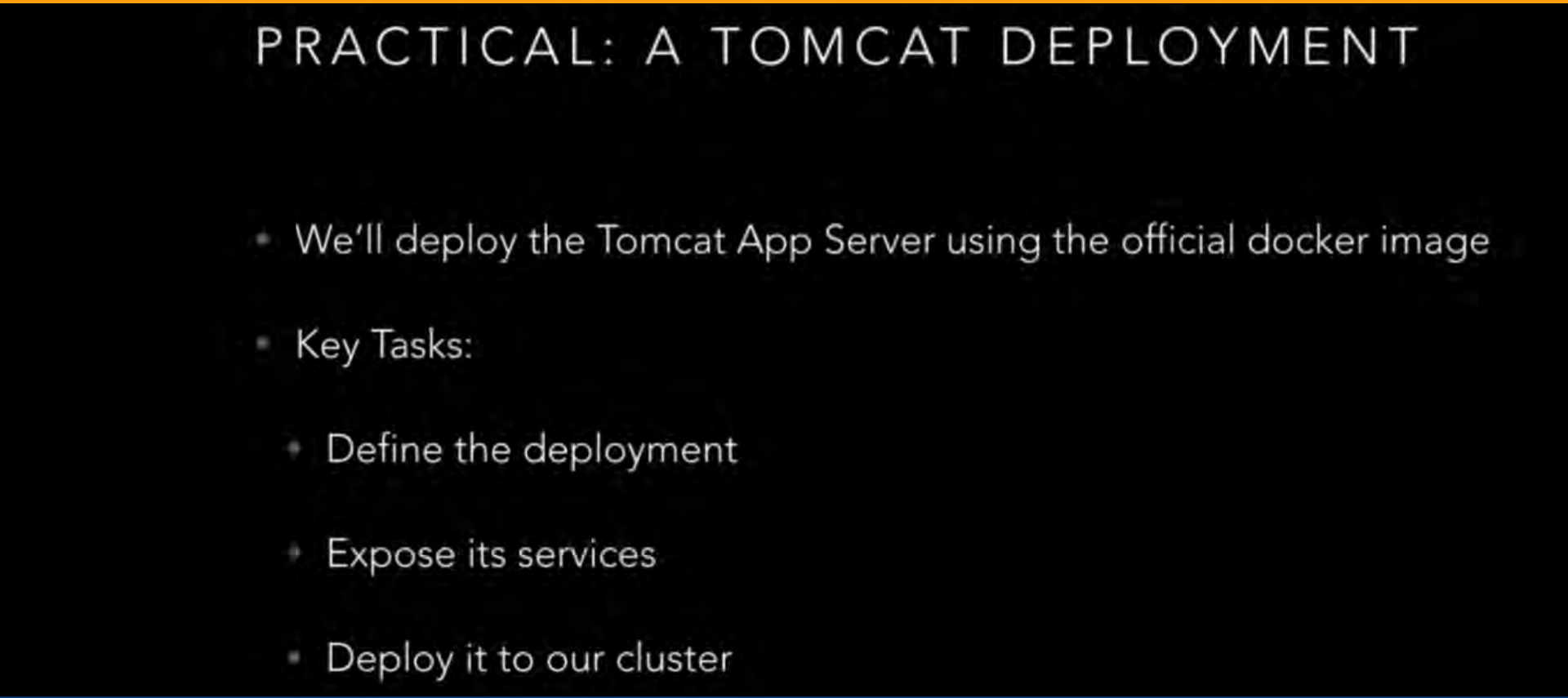
K8S: first app

Passionate thinkers with the tiny little obsession of making business scalable and profitable, in energy and beyond

K8S: first app



K8S: first app



K8S: first app

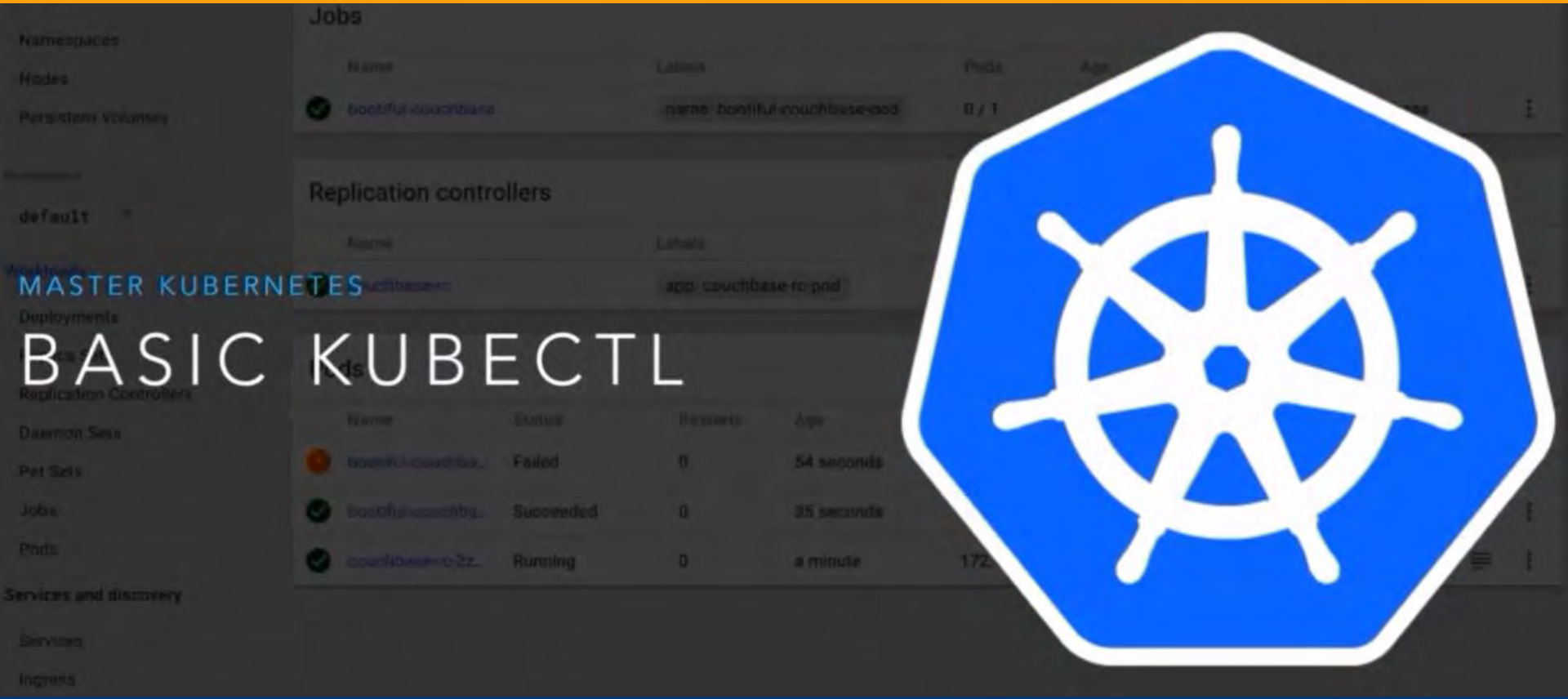
K8S: first app

PRACTICAL: A TOMCAT DEPLOYMENT

deployment.yaml

```
apiVersion: apps/v1beta2
kind: Deployment
metadata:
  name: tomcat-deployment
spec:
  selector:
    matchLabels:
      app: tomcat
  replicas: 1
  template:
    metadata:
      labels:
        app: tomcat
    spec:
      containers:
      - name: tomcat
        image: tomcat:9.0
        ports:
        - containerPort: 8080
```

K8S: basic kubectl



Passionate thinkers with the tiny little obsession of making business scalable and profitable, in energy and beyond

K8S: basic kubectl

KUBECTL GET PODS

```
$ kubectl get pods
```

- Lists all pods in all namespaces
- Provides the pod name, how many instances of the pod are running & ready, its status, how many times they have restarted, and their age

Sample Output

```
root@vt-es-1:~# kubectl get pods
NAME                                READY     STATUS    RESTARTS   AGE
tomcat-deployment-7bd7889564-717n2  1/1       Running   0          15m
root@vt-es-1:~#
```

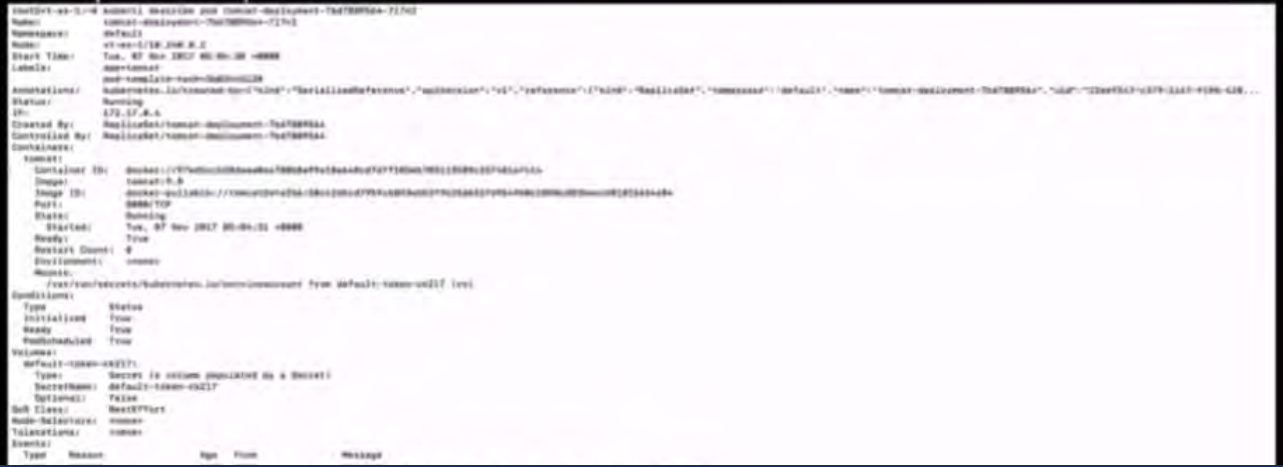
K8S: basic kubectl

KUBECTL DESCRIBE POD

```
$ kubectl get pods [pod name]
```

- Describes detailed information about all pods or a specified pod (optional pod name argument)

Sample Output



K8S: basic kubectl

KUBECTL EXPOSE PORT

```
$ kubectl expose <type name> <identifier/name> [—port=external port] [—target-port=container-port [—type=service-type]
```

- Exposes a port (TCP or UDP) for a given deployment, pod, or other resource

Sample Output

```
root@vt-es-1:~# kubectl expose deployment tomcat-deployment --type=NodePort
service "tomcat-deployment" exposed
```

K8S: basic kubectl

KUBECTL EXEC

```
$ kubectl exec  [-it] <pod name> [-c CONTAINER] — COMMAND [args...]
```

- Execute a command in a container
- -i option will pass stdin to the container
- -t option will specify stdin is a TTY

Sample Output

```
root@vt-es-1:~# kubectl exec -it tomcat-deployment-7bd7889564-717n2 bash
root@tomcat-deployment-7bd7889564-717n2:/usr/local/tomcat# whoami
root
root@tomcat-deployment-7bd7889564-717n2:/usr/local/tomcat#
```

K8S: scaling & replication

Passionate thinkers with the tiny little obsession of making business scalable and profitable, in energy and beyond

K8S: scaling & replication

SCALING APPLICATIONS

- Stateful vs Stateless Applications
- How state is handled can either enable or interfere with scaling, it most certainly dictates how you'll scale

K8S: scaling & replication

REPLICAS

- Kubernetes allows you to define replicas when you deploy an application - you have a few options
  - Setting "replica" in your Deployment (recommended)
  - Defining a ReplicaSet
  - Bare Pods
  - Job
  - DaemonSet

K8S: scaling & replication

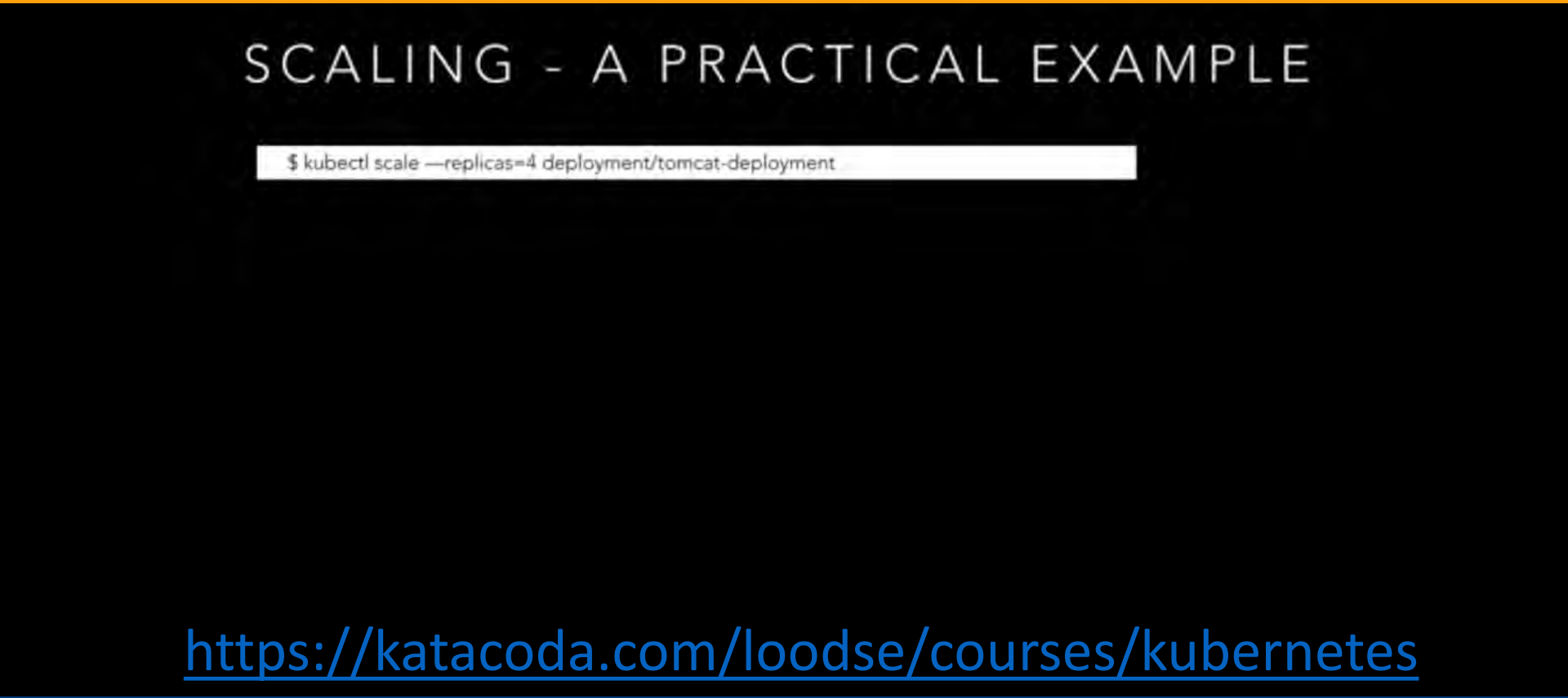
SCALING - A PRACTICAL EXAMPLE

deployment.yaml

```
apiVersion: apps/v1beta2
kind: Deployment
metadata:
  name: tomcat-deployment
spec:
  selector:
    matchLabels:
      app: tomcat
  replicas: 4
  template:
    metadata:
      labels:
        app: tomcat
    spec:
      containers:
      - name: tomcat
```

https://katacoda.com/loodse/courses/kubernetes

K8S: scaling & replication

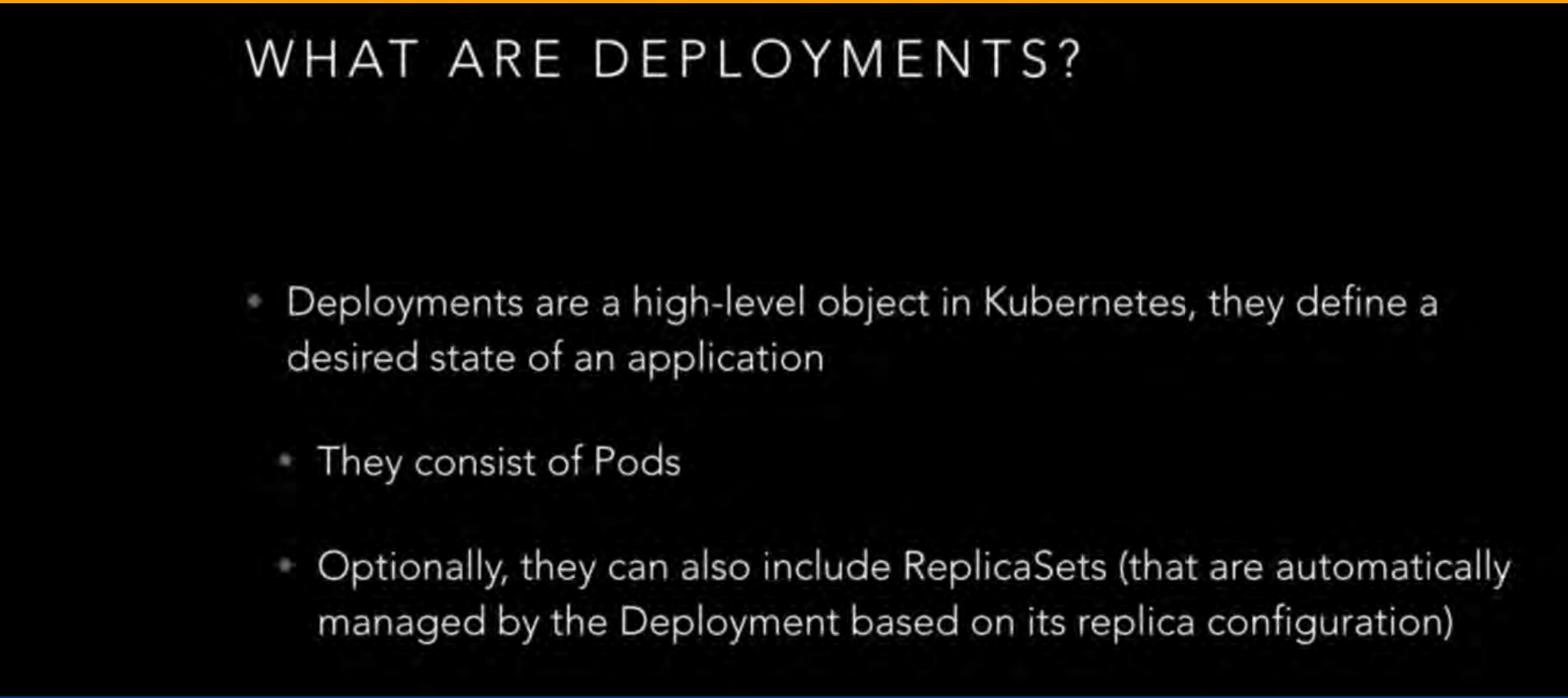


https://katacoda.com/loodse/courses/kubernetes

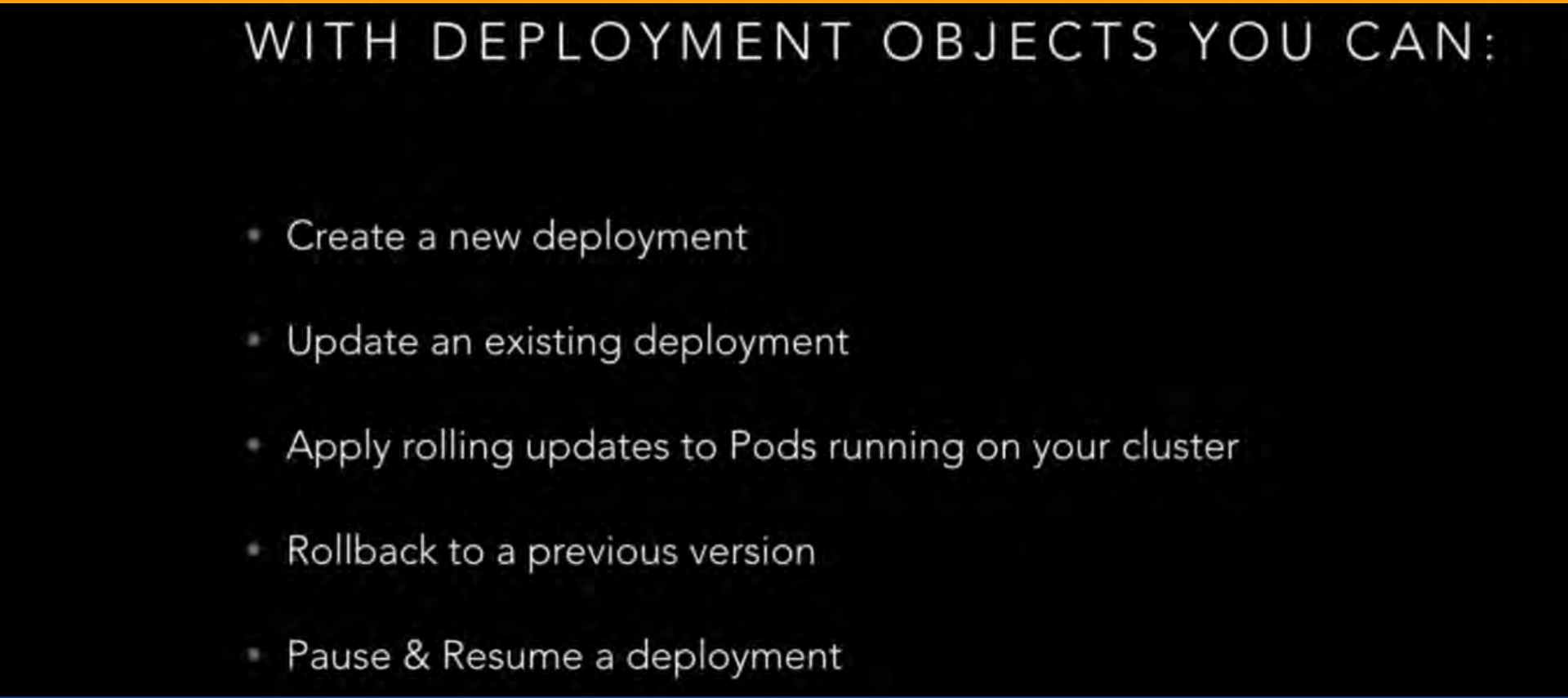
K8S: deployments

Passionate thinkers with the tiny little obsession of making business scalable and profitable, in energy and beyond

K8S: deployments



K8S: deployments



K8S: web interface

Passionate thinkers with the tiny little obsession of making business scalable and profitable, in energy and beyond

K8S: web interface

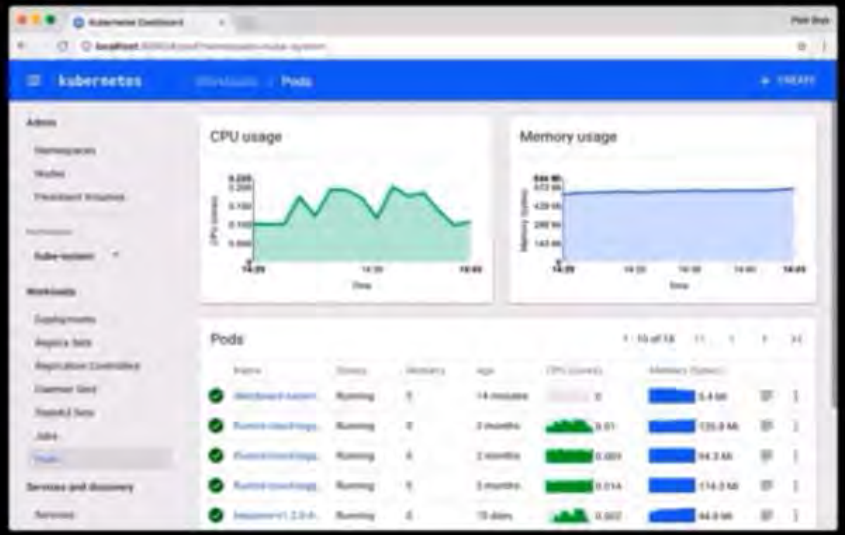
KUBERNETES WEB UI

- kubectl is great, but sometimes a web UI is useful, too
  - Called the "Dashboard UI"
- Runs on your Kubernetes master(s)
- Accessible directly if you have direct network connectivity directly to your cluster/master(s) (unlikely in production situations)
  - kubectl can create a proxy/tunnel for you in situations you do not

```
$ kubectl proxy
```

K8S: web interface

KUBERNETES WEB UI



- Provides a variety of views for nearly anything in your Kubernetes cluster
- Allows update, deletion, and creation of nearly anything in your Kubernetes cluster
- Accesses the same APIs as kubectl

K8S: web interface

USING THE WEB UI

- On some Kubernetes clusters the Dashboard UI is pre-installed, on some it is not (many cloud providers' Kubernetes services include it)
- Installing the Dashboard

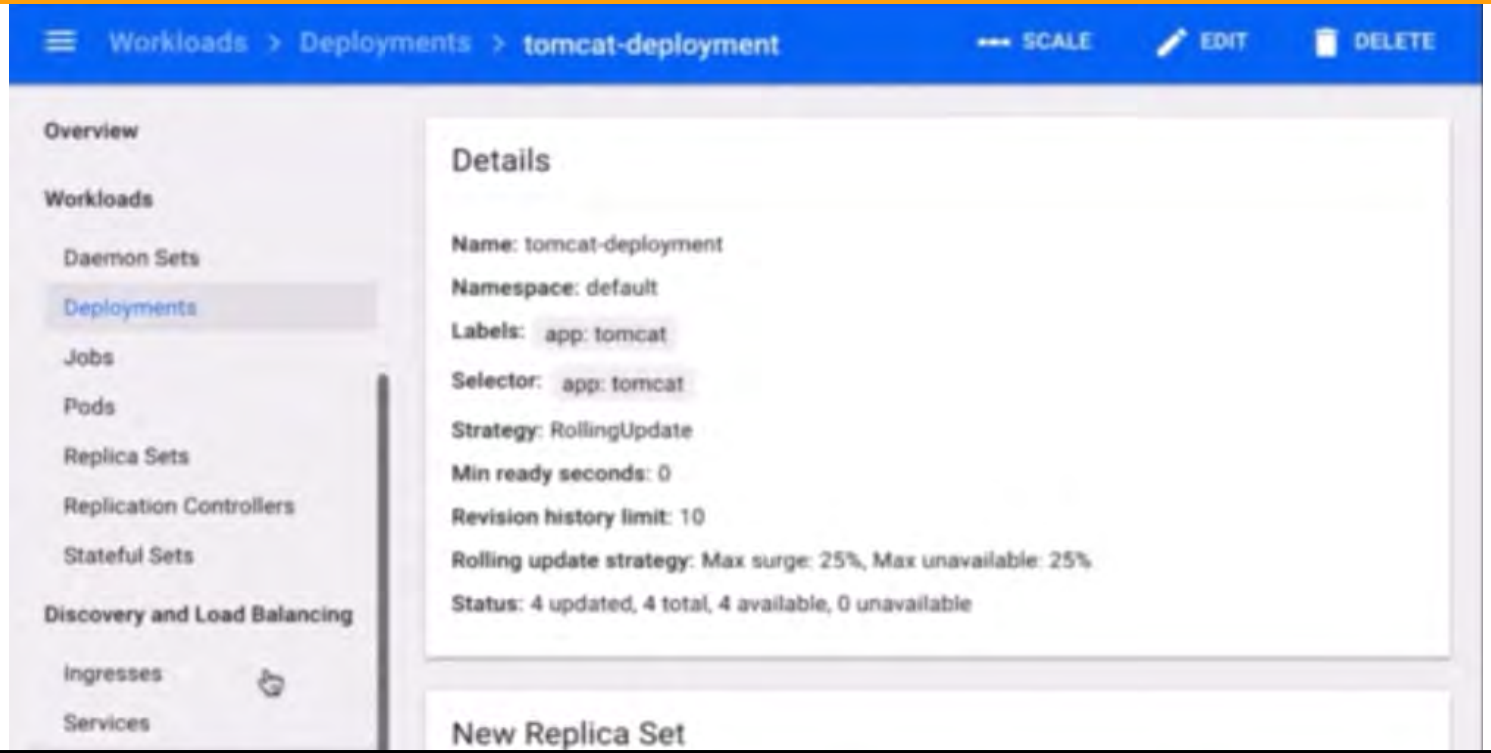
```
$ kubectl create -f https://raw.githubusercontent.com/kubernetes/dashboard/master/src/deploy/recommended/kubernetes-dashboard.yaml
```

- Accessing the dashboard (the most fool-proof way)

```
$ kubectl proxy
```

- Navigate to "http://localhost:8001/ui" in your web browser

K8S: web interface



https://www.katacoda.com/mjboxboat/courses/kubernetes-basic/

K8S: exercize

Passionate thinkers with the tiny little obsession of making business scalable and profitable, in energy and beyond

K8S: exercize

K8S: exercize

YOUR TASK

- Use what you've learned to deploy and scale MongoDB (a popular NoSQL datastore)
- Success criteria:
  - The current version of MongoDB is running on your Kubernetes cluster with four replicas
- A few hints:
  - MongoDB listens on port 27017
  - Officially supported MongoDB Docker images are available on Docker Hub

K8S: exercize

K8S: exercize

OTHER POSSIBLE SOLUTIONS

- Write a deployment.yaml file, use kubectl apply, & use kubectl expose to expose a service
- Write a deployment.yaml file & a service.yaml file and use kubectl apply on both
- Use a Kubernetes package manager like helm to handle the work for you

Helm: fundamentals

Passionate thinkers with the tiny little obsession of making business scalable and profitable, in energy and beyond

Helm: fundamentals

HELM FUNDAMENTALS

- What Is Helm?
- Helm Installation Guides for all Operating Systems
- First Helm Deployment
- Helm Charts and First Chart Creation
- Using Helm Template Calls
- Helm Value Files
- Common Helm Commands

Helm: fundamentals

WHAT IS HELM?

- Kubernetes can become incredibly complex with all the objects you need to manage
- Helm provides a simple way to package and configure exactly what you need by combining everything into one easy deployment
- Helm fills the need to be able to quickly and reliably provision container applications through easy installation, update, and removal

Helm: fundamentals

WHAT IS HELM?

- With Helm and its ‘charts’, you can define, run, and upgrade unique and complex Kubernetes apps
- The package manager helps developers streamline the installation and management of Kubernetes applications
- Helm Charts are deployment packages all bundled up

Helm: fundamentals

HOW DOES HELM WORK?

- The package manager is comprised of two parts:
  - The tool itself (the 'helm')
  - The server (the 'tiller') which runs inside Kubernetes clusters
  - When you input the Helm install command, a Tiller Server receives the request and installs the appropriate package

Helm: fundamentals

HELM CHARTS - OFFICIAL AND UNOFFICIAL

- Helm Charts gives you the ability to leverage Kubernetes packages through the click of a button or single CLI command
- Using the ‘helm’ tool, you can locate, customize, utilize, and install any of these charts
- All Helm Charts contain at least two things:
  - A package description (chart.yaml)
  - One or more templates, which contain Kubernetes manifest files

Helm: fundamentals

ACCESSING HELM CHARTS

- Helm offers a huge repository of both official and unofficial charts on GitHub
- Charts can be stored on a disk or just pull them from remote chart repositories as needed
- To install a chart, use ‘helm install stable/<chart>’

Helm: fundamentals

HELM REVIEW

- Helm provides a simple way to programmatically package everything into one simple deployment
- With Helm and its ‘charts’, developers can quickly define, run, and upgrade even really complex Kubernetes apps
- Helm offers a huge repository of both official and unofficial charts on GitHub
- Charts can be stored on a disk or just pull them from remote chart repositories as needed

Helm: installation

Passionate thinkers with the tiny little obsession of making business scalable and profitable, in energy and beyond

Helm: installation

HELM INSTALLATION GUIDES

- There are two parts to install for Helm: The ‘helm’ client and the ‘tiller’ server
- Options for ‘helm’ client installation include:
  - Binary releases
  - From Snap (for Linux)
  - From Homebrew (for macOS)
  - From Chocolatey (for Windows)

Helm: installation

- Fixed the `--tiller-namespace` flag for helm plugins
- Suppressed the 'unauthenticated users' warning when `--tiller-tls-verify` is present on `helm init`

There were so many fixes this release that we're probably missing a few noteworthy ones, so we suggest by having a look at the changelog for the full list of fixes and enhancements!

Installation and Upgrading

Download Helm 2.10. The common platform binaries are here:

- MacOS amd64 (checksum)
- Linux amd64 (checksum)
- Linux arm (checksum)
- Linux arm64 (checksum)
- Linux i386 (checksum)
- Linux ppc64le (checksum)
- Windows amd64 (checksum)

Once you have the client installed, upgrade Tiller with `helm init --upgrade`.

https://github.com/helm/helm/releases

otes,

Sign in now to use ZenHub

Helm: installation

HELM INSTALLATION GUIDES - TILLER SERVER

- Connect to your kube cluster
- Install the 'tiller' server into the cluster by running `helm init`
  - This checks the local environment is set up right and connects to a default cluster
  - Once connected, it will install 'tiller' into the `kube-system` namespace
  - You should now be able to run `kubectl get pods --namespace kube-system` to see 'tiller' up and running

Helm: first deployment

Passionate thinkers with the tiny little obsession of making business scalable and profitable, in energy and beyond

Helm: first deployment

Katacoda

Katacoda Overview & Solutions | Search | Your Profile | Log Out

Helm Package Manager

Step 1 of 4

Install Helm

Helm is a single binary that manages deploying Charts to Kubernetes. A chart is a packaged unit of kubernetes software. It can be downloaded from https://github.com/kubernetes/helm/releases

```
curl -LO https://storage.googleapis.com/kubernetes-helm/helm-v2.8.2-linux-amd64.tar.gz
tar -xvf helm-v2.8.2-linux-amd64.tar.gz
mv linux-amd64/helm /usr/local/bin/ ↩
```

Once installed, initialise update the local cache to sync the latest available packages with the environment.

```
helm init
helm repo update ✓
```

CONTINUE

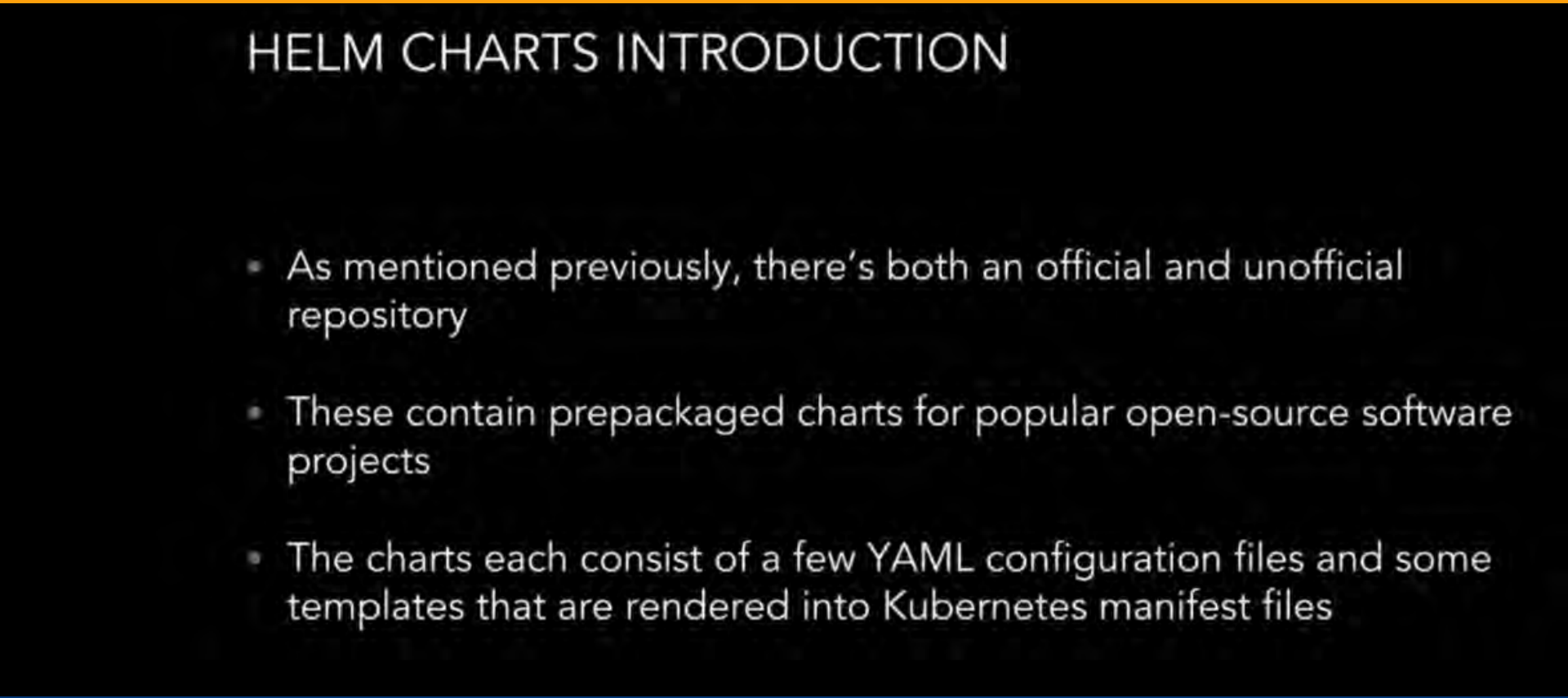
Terminal +

```
Your Interactive Hands On Lab Terminal

master $ launch.sh
Waiting for Kubernetes to start...
Kubernetes started
master $ helm init
Creating /root/.helm
Creating /root/.helm/repository
Creating /root/.helm/repository/cache
Creating /root/.helm/repository/local
Creating /root/.helm/plugins
Creating /root/.helm/starters
Creating /root/.helm/cache/archive
Creating /root/.helm/repository/repositories.yaml
Adding stable repo with URL: https://kubernetes-charts.storage.googleapis.com
Adding local repo with URL: http://127.0.0.1:8879/charts
$HELM_HOME has been configured at /root/.helm.

Tiller (the Helm server-side component) has been installed into your Kubernetes Cluster.

Please note: by default, Tiller is deployed with an insecure 'allow unauthenticated users' policy.
To prevent this, run `helm init` with the --tiller-tls-verify flag.
For more information on securing your installation see: https://docs.helm.sh/using_helm/#securing-your-helm-installation
```

https://katacoda.com/courses/kubernetes/helm-package-manager

Helm: charts

Helm: charts


HELM CHARTS INTRODUCTION
As mentioned previously, there's both an official and unofficial repository
These contain prepackaged charts for popular open-source software projects
The charts each consist of a few YAML configuration files and some templates that are rendered into Kubernetes manifest files

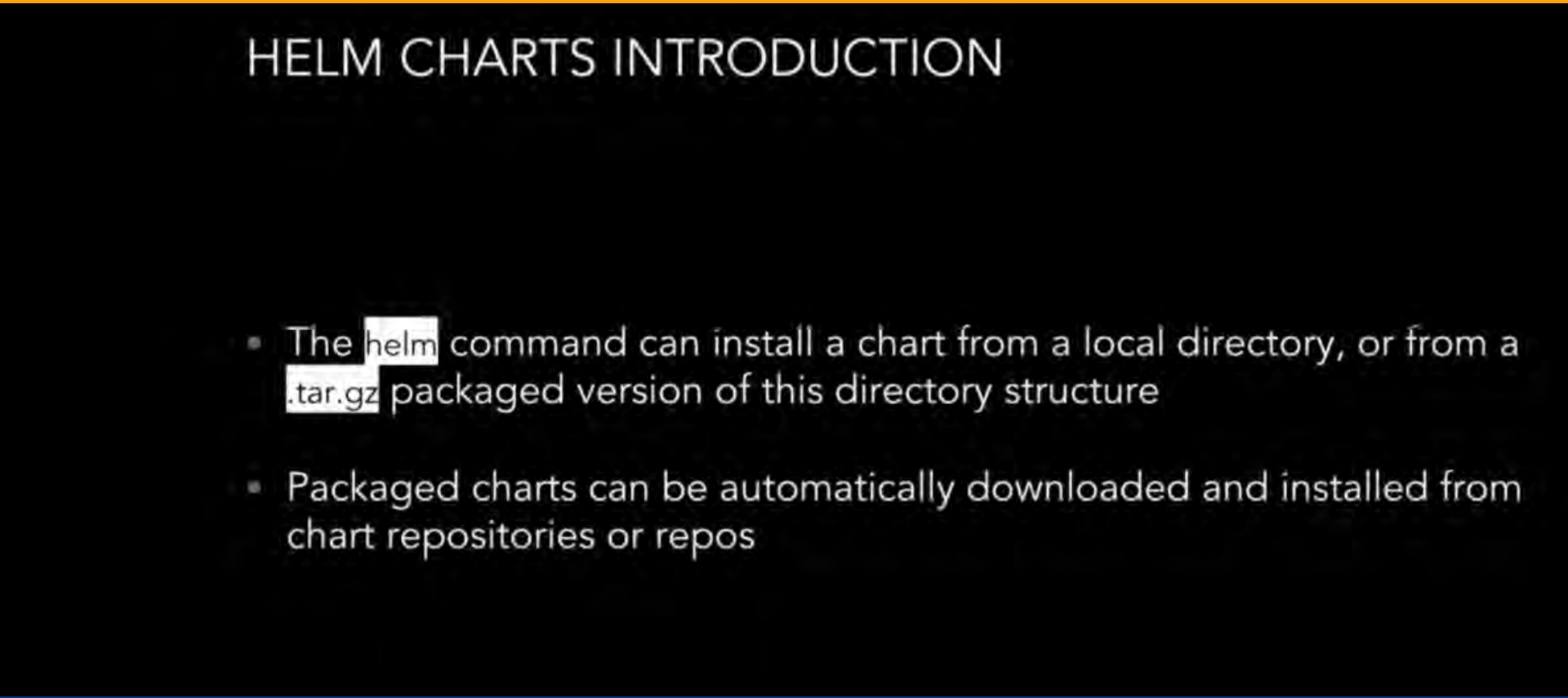
Helm: charts

BASIC HELM CHARTS DIRECTORY STRUCTURE

- Here is the basic directory structure of a chart:

```
package-name/
  charts/
  templates/
  Chart.yaml
  LICENSE
  README.md
  requirements.yaml
  values.yaml
```

Helm: charts


HELM CHARTS INTRODUCTION
The helm command can install a chart from a local directory, or from a .tar.gz packaged version of this directory structure
Packaged charts can be automatically downloaded and installed from chart repositories or repos

Helm: charts

HELM CHART CREATION

- Use the `helm create` command to scaffold out a chart example we can build on
- The following command will create a new chart named mychart in a new directory:

```
$ helm create mychart
```

Helm: charts

HELM CHART CREATION

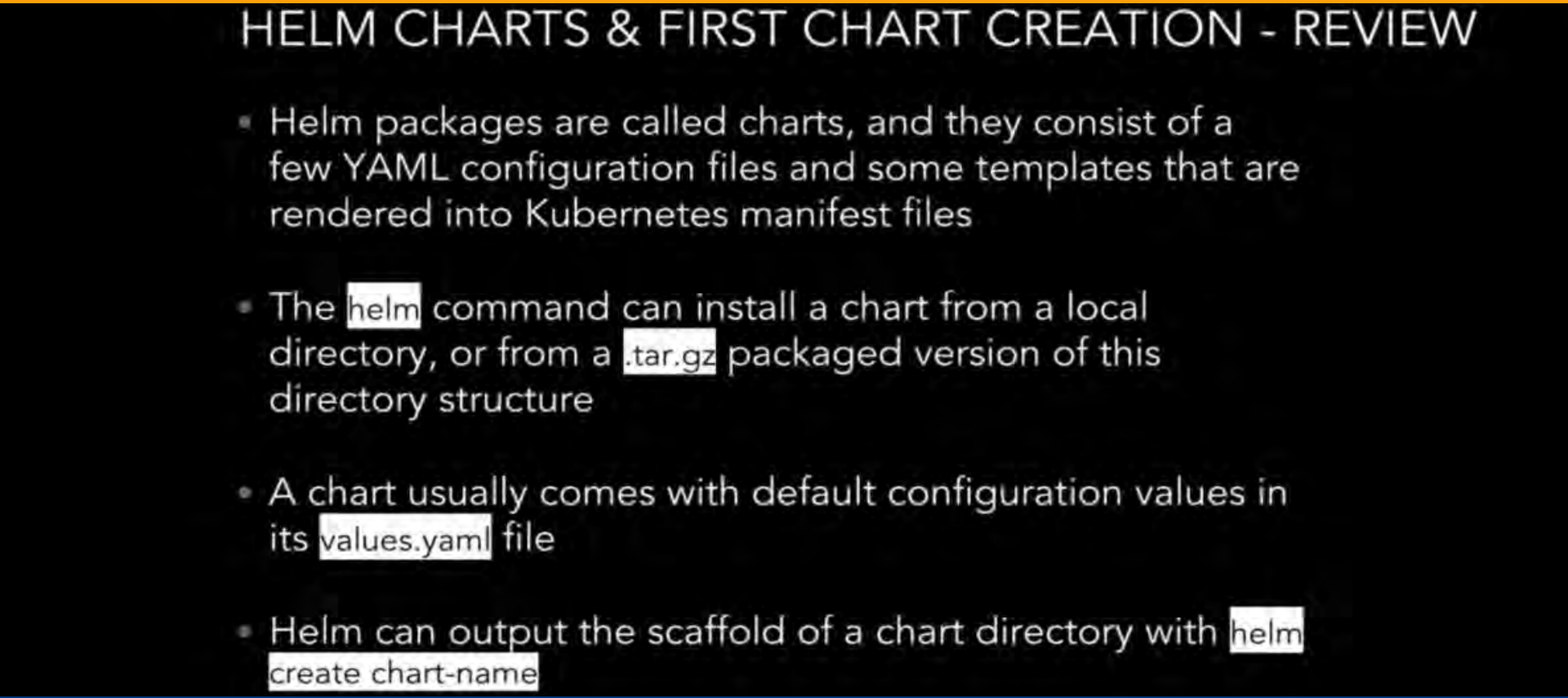
- The result will look like this:

```
mychart
|-- Chart.yaml
|-- charts
|-- templates
|   |-- NOTES.txt
|   |-- _helpers.tpl
|   |-- deployment.yaml
|   |-- ingress.yaml
|   `-- service.yaml
`-- values.yaml
```

- The most important element to note here is the templates/ directory

- This is where Helm finds the YAML definitions for your Services, Deployments, and other Kubernetes objects

Helm: charts


HELM CHARTS & FIRST CHART CREATION - REVIEW
Helm packages are called charts, and they consist of a few YAML configuration files and some templates that are rendered into Kubernetes manifest files
The helm command can install a chart from a local directory, or from a .tar.gz packaged version of this directory structure
A chart usually comes with default configuration values in its values.yaml file
Helm can output the scaffold of a chart directory with helm create chart-name

Helm: template calls

Passionate thinkers with the tiny little obsession of making business scalable and profitable, in energy and beyond

Helm: template calls

USING HELM TEMPLATE CALLS

- When 'tiller' evaluates a chart, it sends all files in the templates/ directory through the template rendering engine
- Templates, when combined with values, will generate valid Kubernetes manifest files
- The Chart.yaml file contains a chart description which is accessible from within a template
- A template directive is enclosed in {{ and }} blocks

Helm: template calls

SIMPLE TEMPLATE CALL

- The template directive `{{ .Release.Name }}` will insert a release name into the YAML template
- Think of values that are injected into a template as namespaced objects, where a dot (.) separates each element
- The leading dot before 'Release' shows that we start with the top-most namespace
- `{{ .Release.Name }}` reads then as "start at the top namespace, find the Release object, then look inside it for an object called Name"

Helm: template calls

TEST TEMPLATE RENDERING

- To test the template rendering, but not actually install anything, you can use `helm install --debug --dry-run ./mychart`
- This will send the chart to the 'tiller server,' which will create the templates
- Instead of installing the chart though, it will return the rendered template so you can see the output

Helm: value files

Passionate thinkers with the tiny little obsession of making business scalable and profitable, in energy and beyond

Helm: value files

HELM VALUE FILES

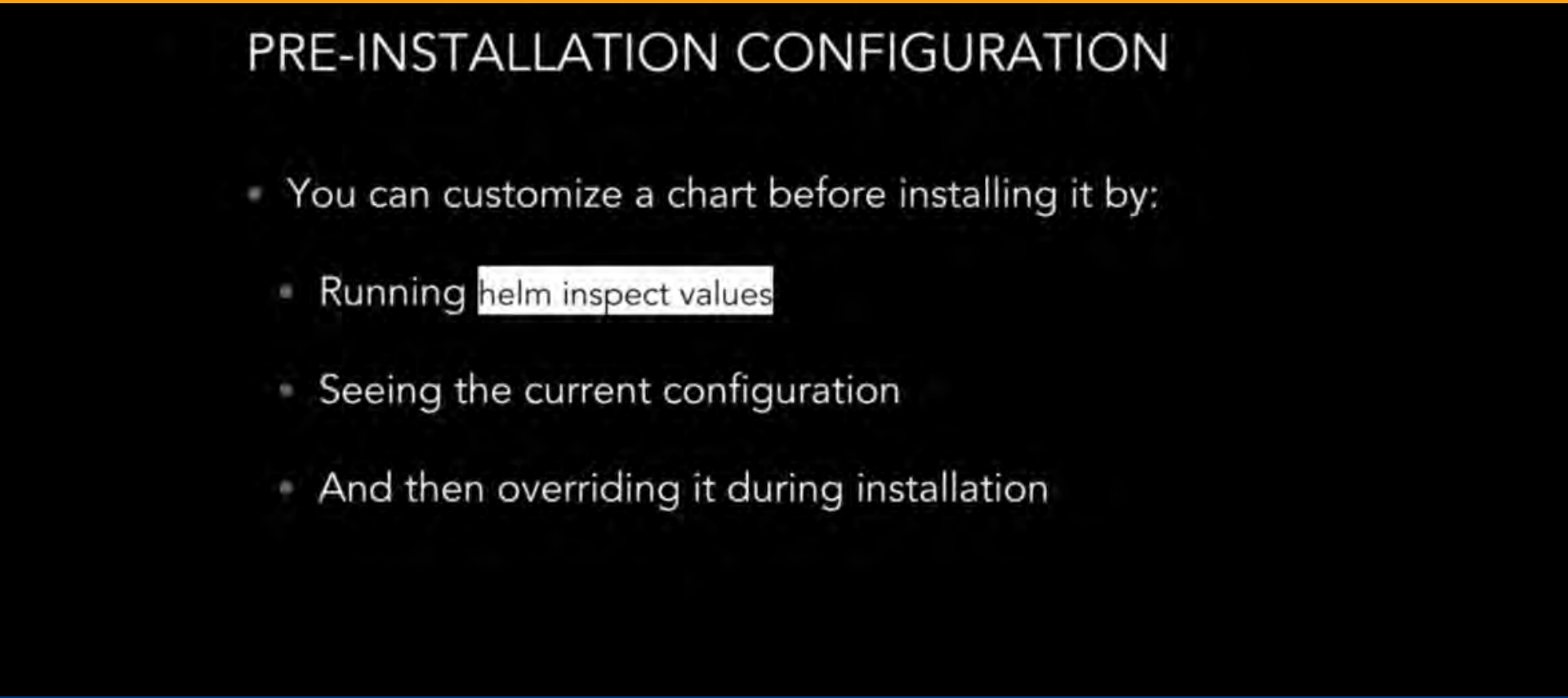
- Values is by far the most important built-in object in the templates directory
- It gives you access to all the values configured in the values.yaml of your chart, it's sub-charts, and any values files or values provided directly on the command line
- Values files are plain YAML files

Helm: value files

HELM VALUE FILES

- Using incorrect values in config files or failing to roll out apps correctly from YAML templates can break deployments
- By preconfiguring certain values through Helm Charts and setting others to sensible defaults, you create a consistent interface for a changing configuration
- This dramatically reduces complexity, and eliminates deployment errors by locking out incorrect configurations

Helm: value files



Helm: value files

VALUE FILES CONFIGURATION

- To provide values for a template in a specific chart:
  - Use a file called `values.yaml` inside all created charts, which contains default values
  - OR supply a YAML file that contains values (this can be provided in the `helm install` command)
  - Using `-f, --values valueFiles` flags allows you to specify values in a YAML file or a URL (can specify multiple) (default [])

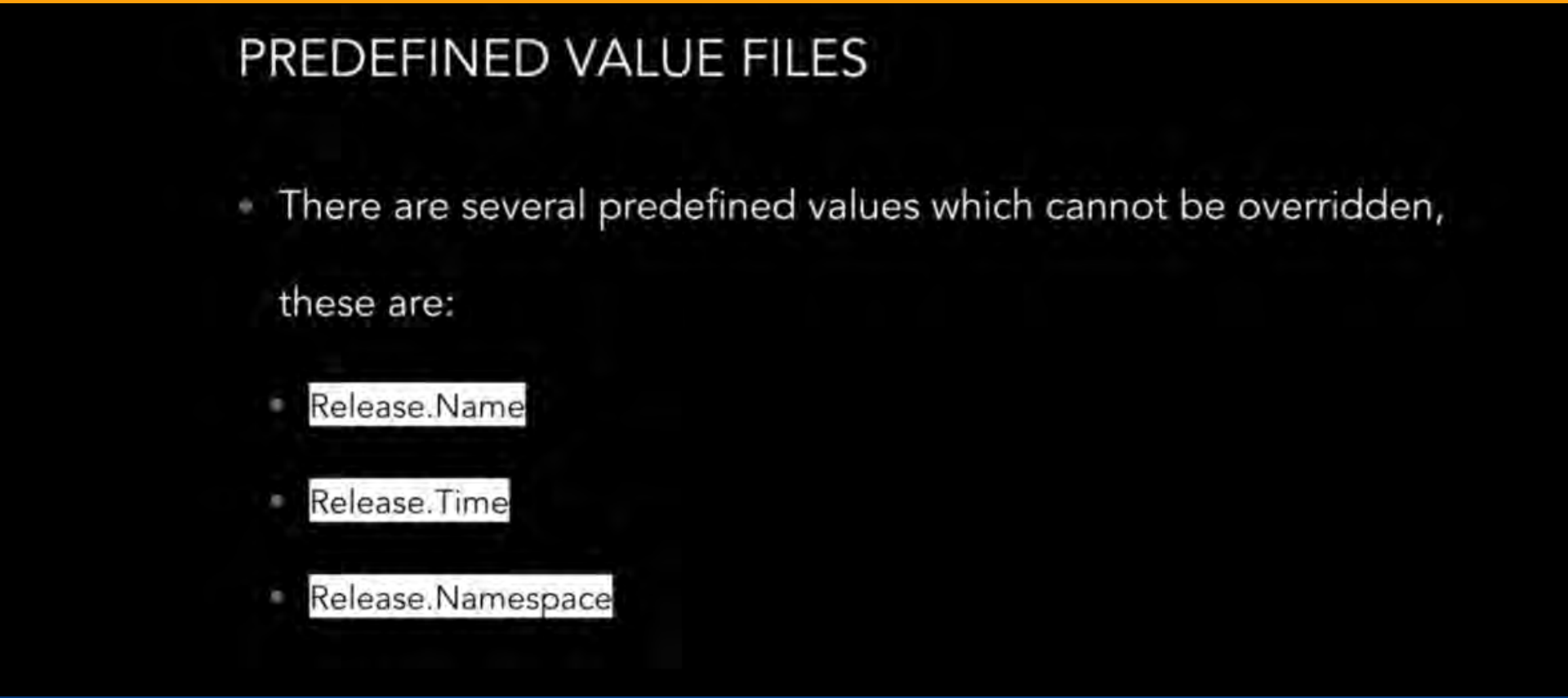
Helm: value files

ACCESSING VALUES

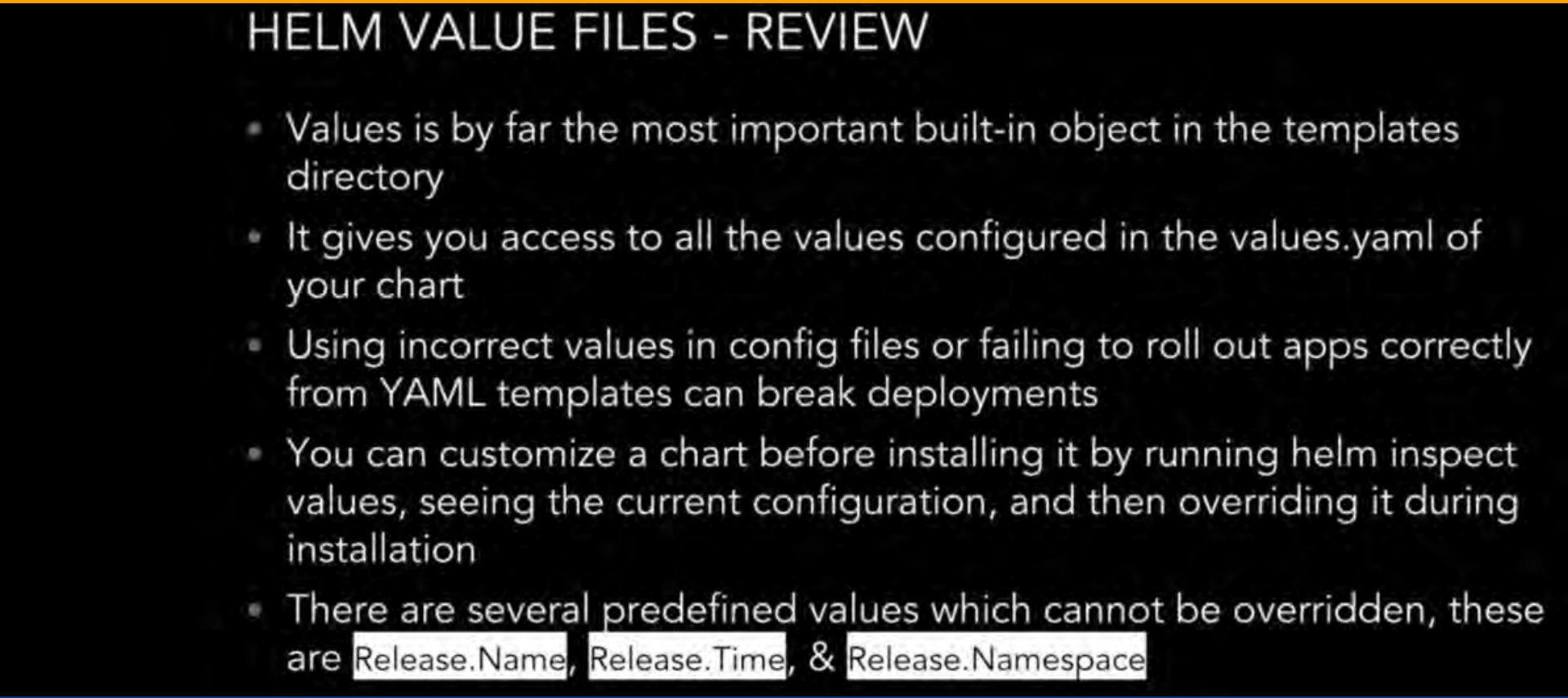
- Go templates provide the typical control structures you'll find in nearly all templating languages: if / else, and range (loop)
- Example:

```
{{- if .Values.deployment.volumes }}
volumes:
{{- range .Values.deployment.volumes }}
- name: {{ .name }}
  secret:
    secretName: {{ .secretName }}
{{- end }}
{{- end }}
```

Helm: value files



Helm: value files



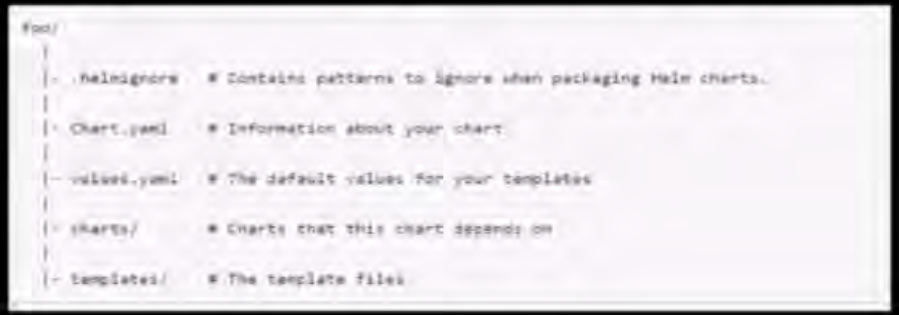
Helm: commands

Passionate thinkers with the tiny little obsession of making business scalable and profitable, in energy and beyond

Helm: commands

HELM CREATE

- `helm create` - Build and name a new chart
  - Use the command to create a chart directory as well as the necessary common files and directories the chart will use
  - E.g., `helm create foo` will create the following directory structure:



Helm: commands

HELM FETCH

- `helm fetch` - Download and unpack a chart from a repository into a local directory
  - Useful for fetching packages to examine, adjust, or repackage
  - Will also run a cryptographic verification of a chart without installing it
  - If a –verify flag is named, the requested chart MUST have a provenance file, and MUST pass the verification analysis
  - Use `helm fetch [flags] [chart URL | repo/chartname] [...]`

Helm: commands

HELM UPGRADE

- `helm upgrade` - Upgrade a requested release
  - Specify 'release' and 'chart' arguments
  - The chart argument can be one of:
    i. A chart reference('stable/mariadb')—use `--version` and `--devel` flags for older versions
    ii. A chart directory path
    iii. A packaged chart
    iv. A full URL
  - Use `helm upgrade [RELEASE] [CHART] [flags]`

Helm: commands

HELM STATUS

- `helm status` - Shows the status of the requested release
  - The status will include:
    i. Last deployment time
    ii. Kube namespace where the release lives
    iii. Release state (UNKNOWN, DEPLOYED, DELETED, SUPERSEDED, FAILED or DELETING)
    iv. List of resources that this release consists of, sorted by kind
    v. Details on last test suite run (if applicable)
    vi. Additional notes from the chart
  - Use `helm status [flags] RELEASE_NAME`

Helm: commands

HELM ROLLBACK

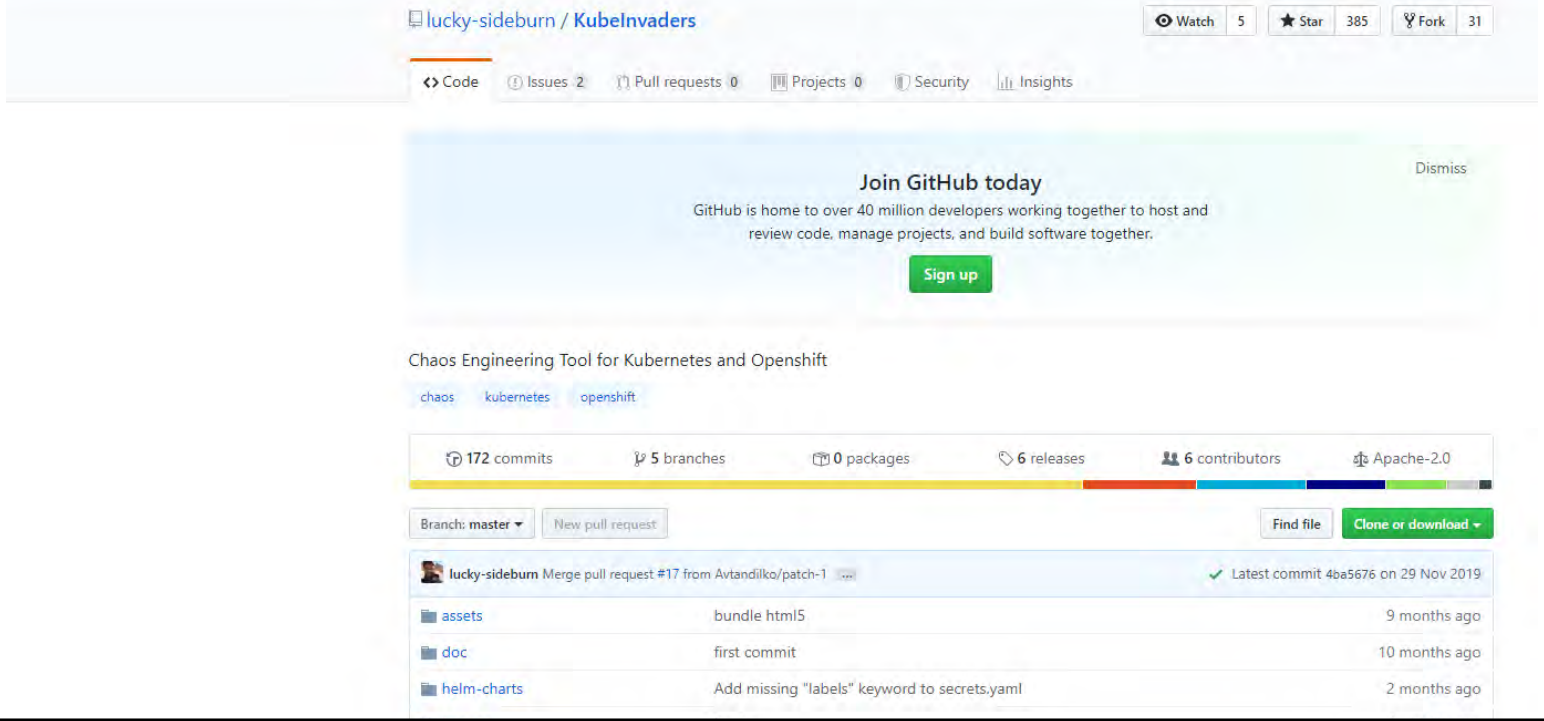
- `helm rollback` - Allows you to roll back a release to a previous version
  - The rollback command's first argument is the 'name' of a release
  - The second is the 'revision' (version) number
  - To display revision numbers, run `helm history RELEASE`
  - Use `helm rollback [flags] [RELEASE] [REVISION]`

KubeInvaders: a gamified chaos engineering tool

KubeInvaders is a gamified chaos engineering tool for Kubernetes. It is like Space Invaders but the aliens are PODs

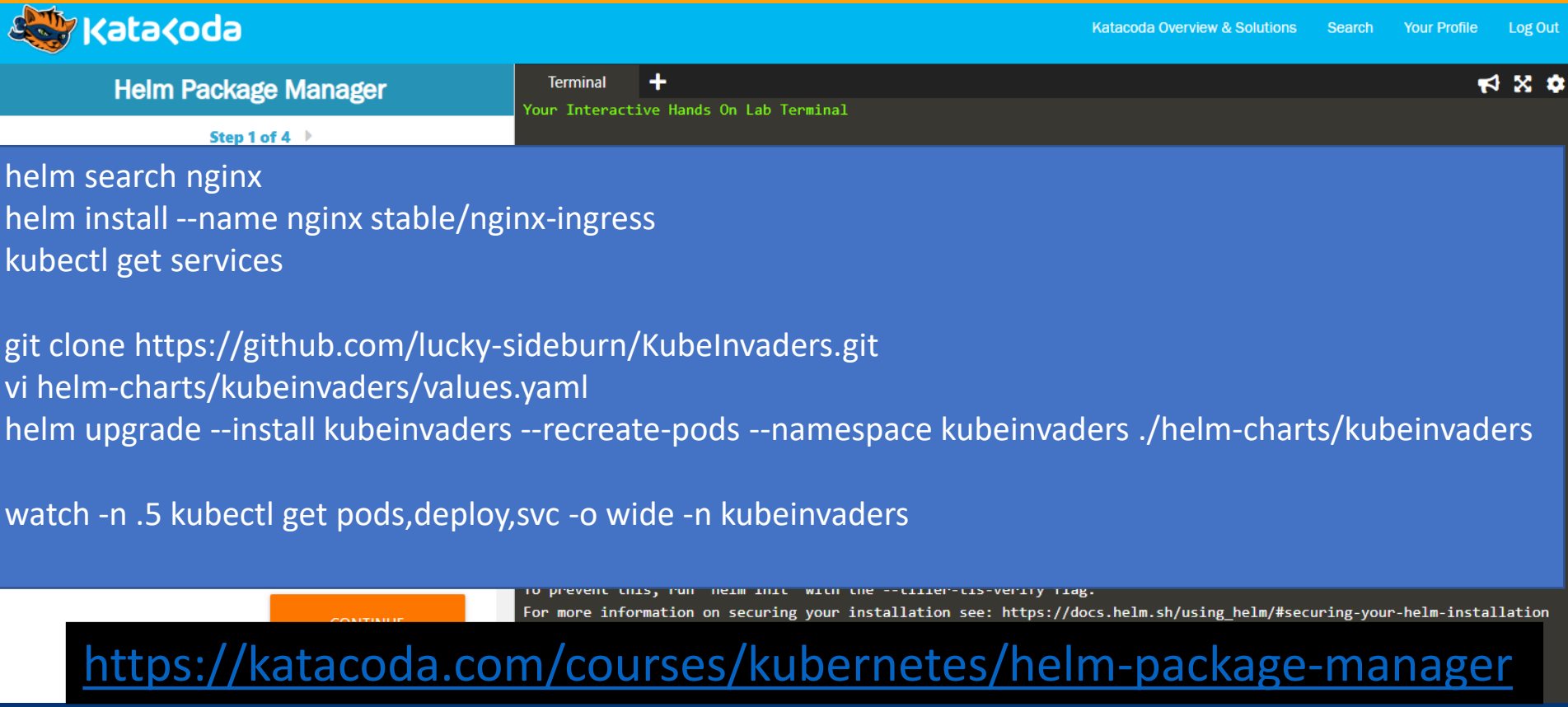
Passionate thinkers with the tiny little obsession of making business scalable and profitable, in energy and beyond

KubeInvaders: a gamified chaos engineering tool



https://github.com/lucky-sideburn/KubeInvaders

KubeInvaders: a gamified chaos engineering tool



Passionate thinkers with the tiny little obsession of making business scalable and profitable, in energy and beyond

Conclusion

Passionate thinkers with the tiny little obsession of making business scalable and profitable, in energy and beyond

Conclusion

Passionate thinkers with the tiny little obsession of making business scalable and profitable, in energy and beyond

Credits

- https://katacoda.com
- https://www.level-up.one
- https://github.com/lucky-sideburn/KubeInvaders

THANK YOU