SUSE

12 PAŹDZIERNIKA 2021

Rancher Rodeo w Polsce

Warsztaty on-line i stacjonarne

Introduction

Jan Bruder
Field Engineer,

Rancher/SUSE

Jarosław Biniek
Presales Engineer

SUSE

Objects for the day

1. Explain Docker and Kubernetes concepts
2. Provision HobbyFarm user accounts
3. Install and Demo Rancher Server
4. Deploy a Kubernetes cluster
5. Deploy a hello-world application into the cluster
6. Deploy WordPress into the cluster
7. Technical Discussion, Questions

Forewarning

- This Rodeo Presentation is designed for those that are new to Kubernetes and Rancher
- We will cover some slightly advanced techniques (if we have time)
- What is presented in here is not necessarily how you should run something in production, but should help give you a basis to put together the right tools to be able to run production workloads

Prerequisites

- Up-to-date browser
  - Google Chrome
  - Mozilla Firefox
  - Other Chromium based browsers

Have comments or questions?

Talk to the team by:

- Use the Questions Feature in the GTM Panel
- Joining the Teams meeting during the break

Learning Platform URL

https://learn.eu.hobbyfarm.io

Containers & Kubernetes

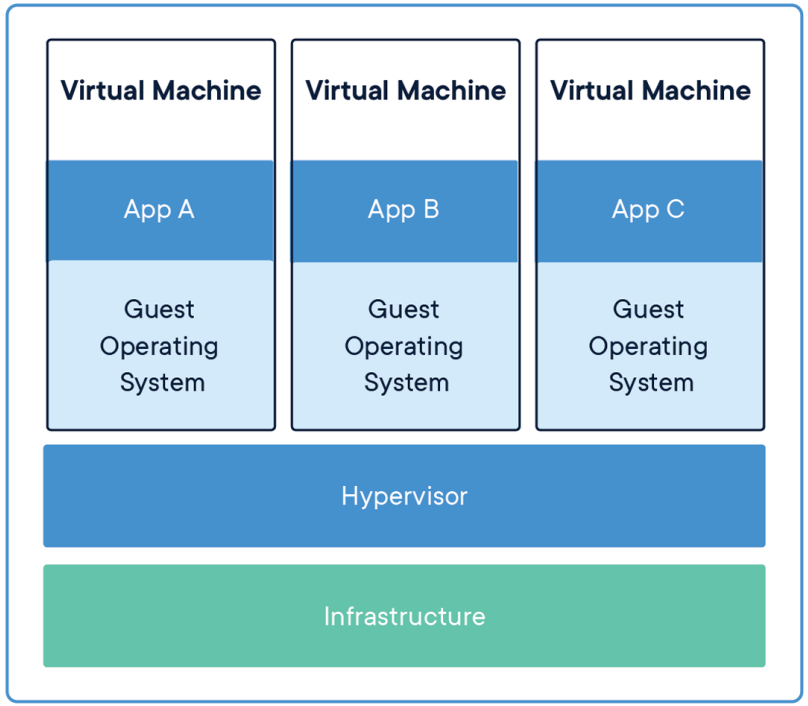
The Basics

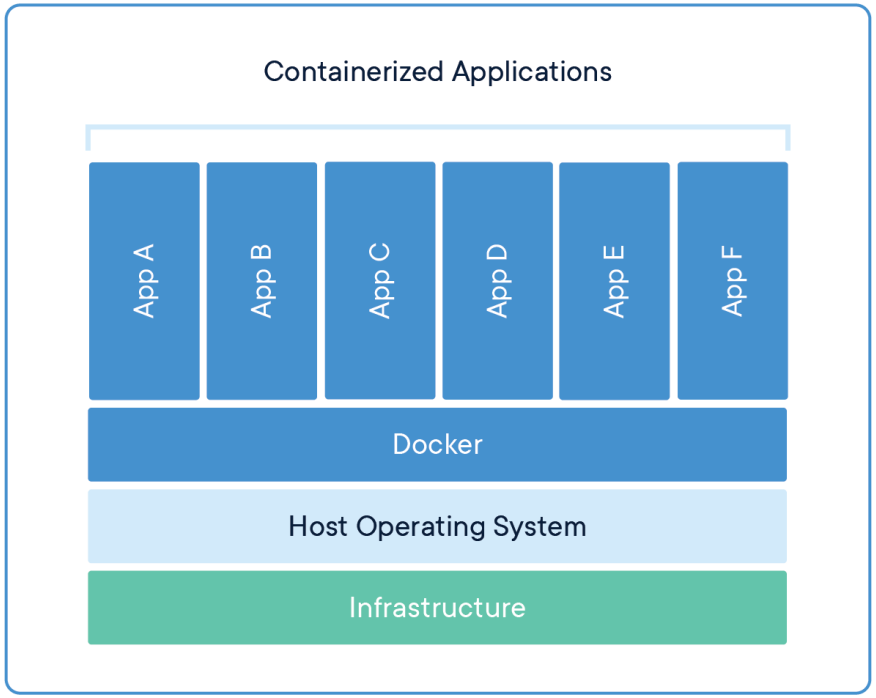
Containers are great!

Containers

- A running instance of a container image
- Portable - You build your application once, and you run it anywhere.
- You have a clean, isolated runtime environment for your application.
- You can scale your application horizontally by spawning more containers. Each container is guaranteed to be identical.

Containers





Source : https://www.docker.com/resources/what-container

Container Images

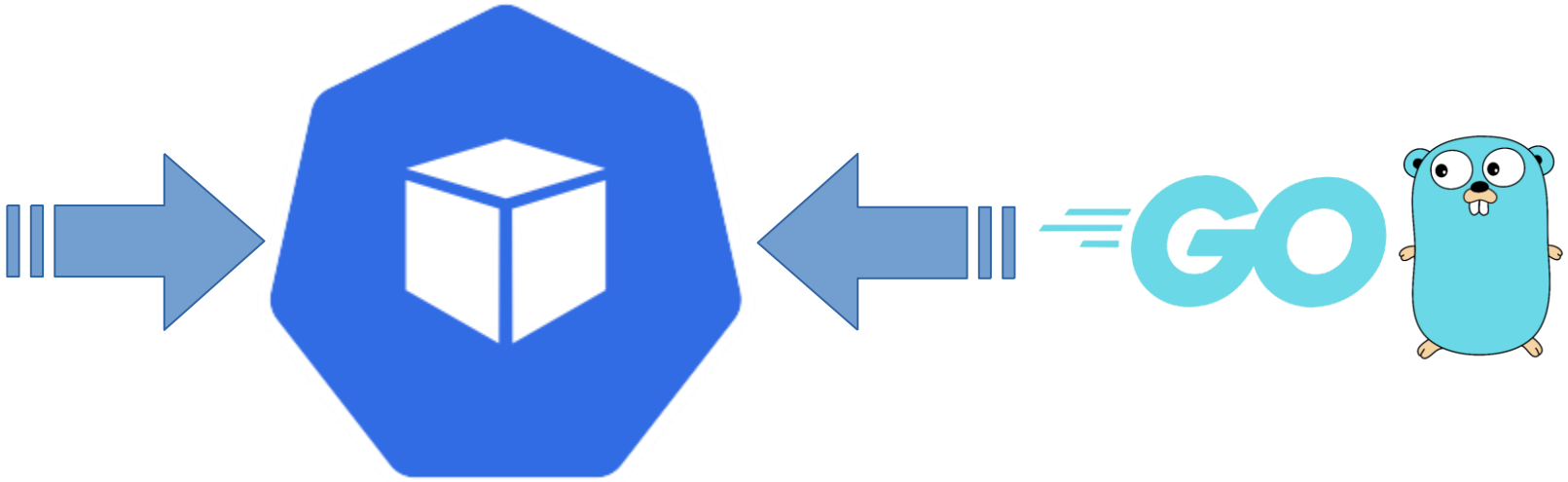
```go
package main

import (
    "fmt"
    "net/http"
)

func hello(w http.ResponseWriter, req *http.Request) {
    fmt.Fprintf(w, "Hello, world!\n")
}

func headers(w http.ResponseWriter, req *http.Request) {
    for name, headers := range req.Header {
        for _, h := range headers {
            fmt.Fprintf(w, "%v: %v\n", name, h)
        }
    }
}

func main() {
    http.HandleFunc("/hello", hello)
    http.HandleFunc("/headers", headers)
    http.ListenAndServe(":80", nil)
}
```

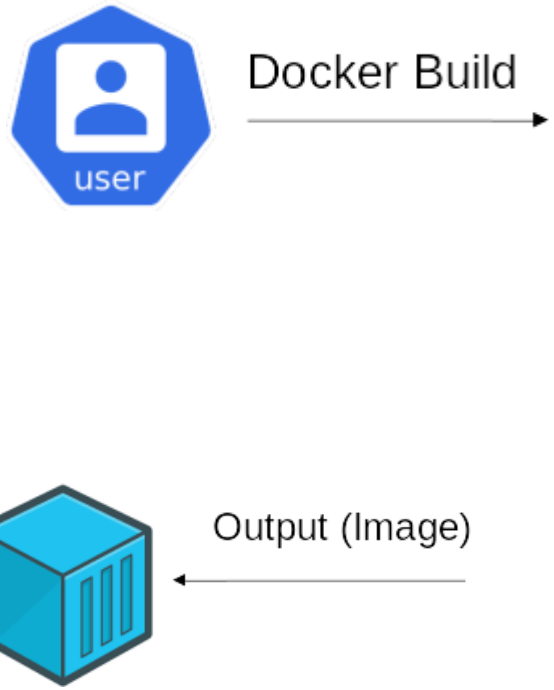


Application Code

Container Image

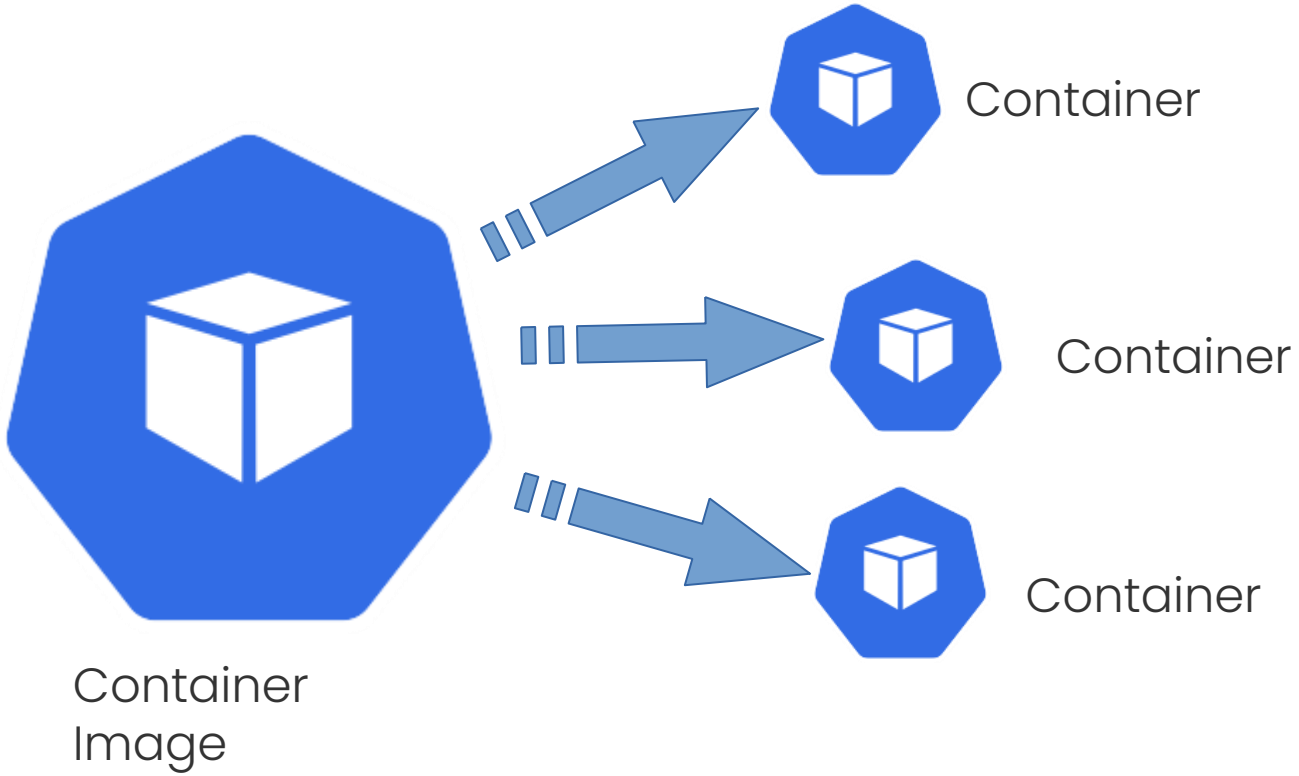
Application Dependencies

Building an image

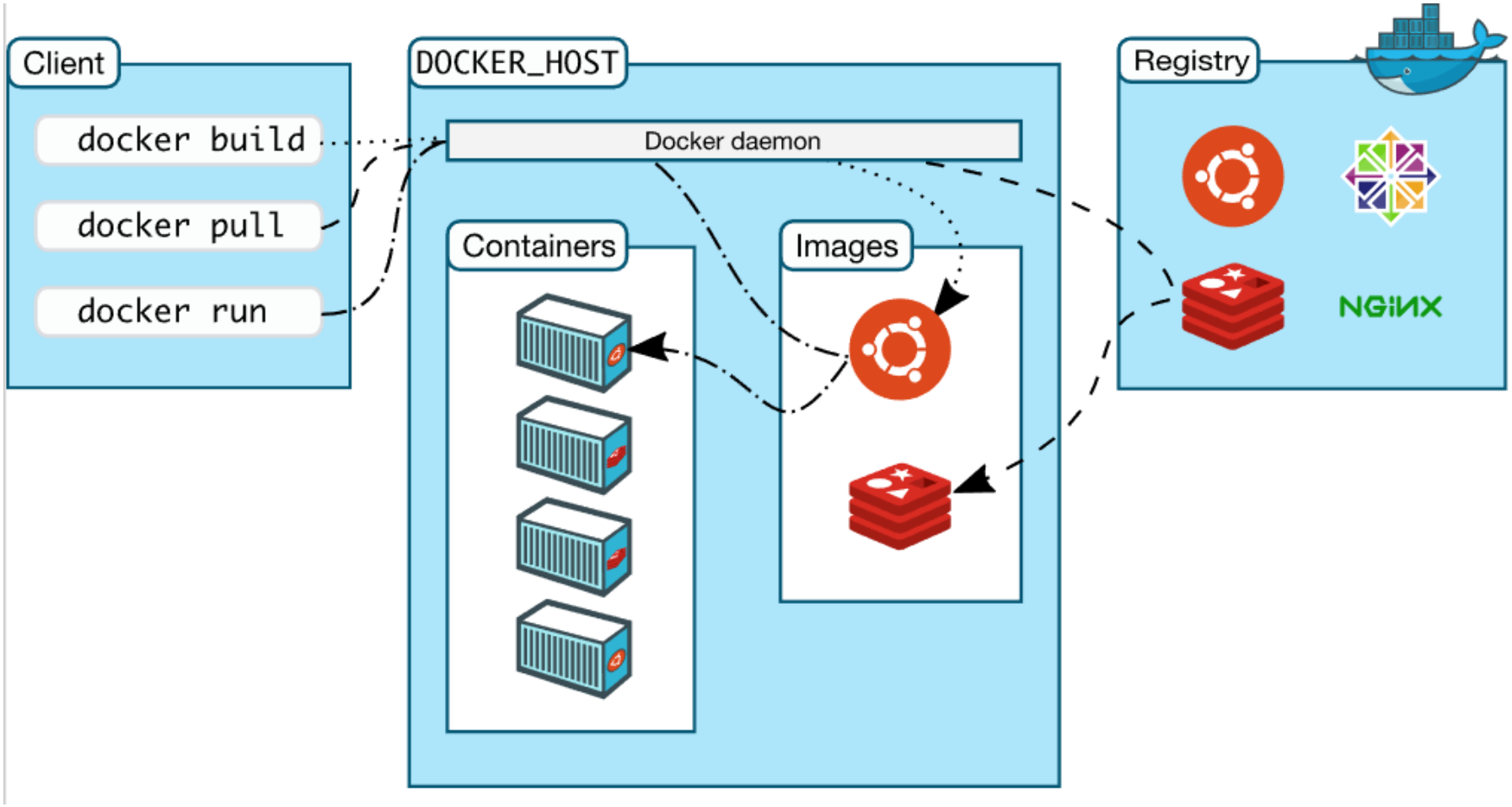



```
1   FROM ubuntu:latest
2
3   LABEL maintainer="someone@someorg.com"
4
5   RUN \
6           apt-get update && \
7           apt-get -y upgrade && \
8           apt-get install -y apache2
9
10  COPY /home/user/src/* /var/www/html
11
12  WORKDIR /root
13
14  CMD ["usr/sbin/apache2ctl", "-D", "FOREGROUND"]
15
16  EXPOSE 80
```

Running an image in a container



Distributing a Container Image



Container Registries

- Central place to store images
  - Public
  - Private
  - On-prem / Cloud
- Examples:
  - Amazon Elastic Container Registry
  - Azure Container Registry
  - Docker Hub
- Some are extended with other capabilities:
  - Vulnerability scanning
  - LDAP / AD Support
  - Auditing
  - Replication
  - …

Containers are great.....but

Managing a couple – no problem

Containers are great.....but

How about managing many?

How do we address:

Networking, Security, Scheduling, Automation, etc?

kubernetes

Kubernetes is a powerful open-source platform for managing containerized workloads and services

Declarative Configuration

Everything in Kubernetes is a an API Resource

You can create/retrieve/update/delete resources

Pods

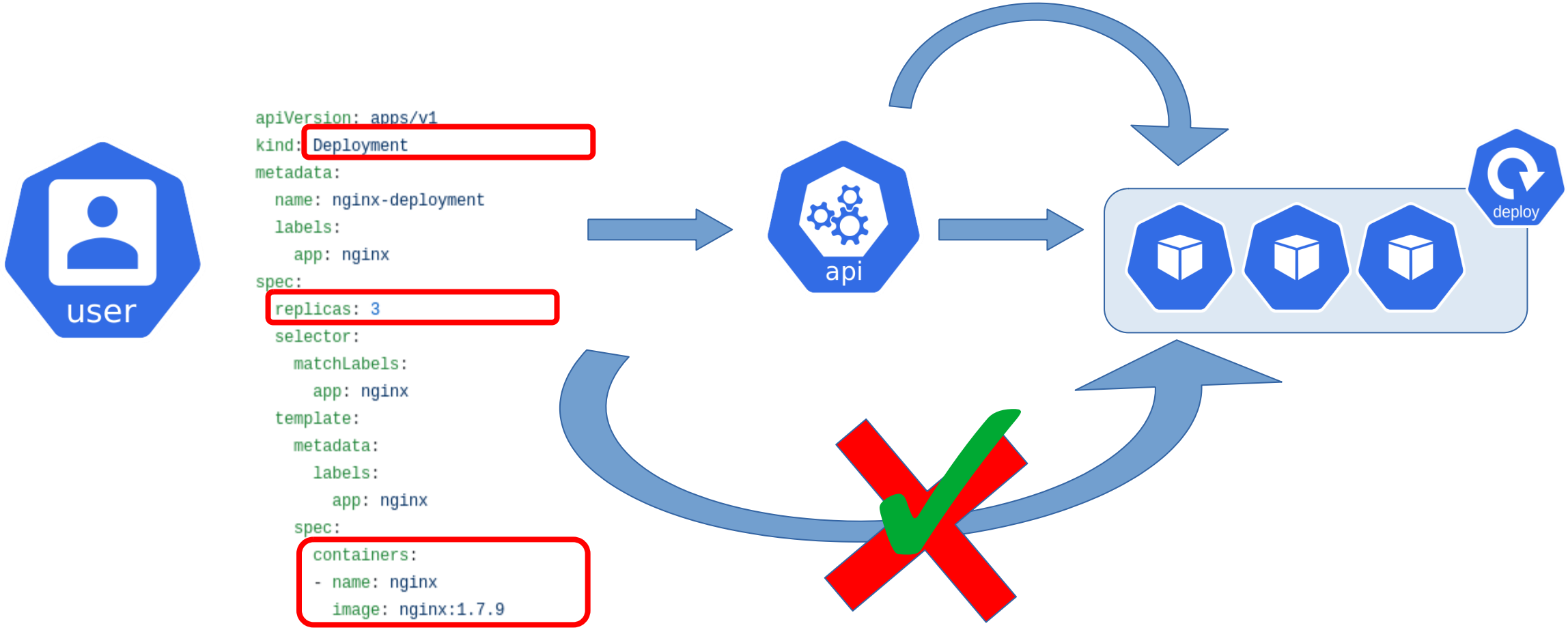
- Smallest unit of deployment
  - Similar to what a VM is in the hypervisor world
- Can contain one or more containers
- Has its own resources:
  - Storage (i.e., shared volumes)
  - Networking
- Scheduled on to a Worker node

Deployments

- Abstraction for managing a number of identical pods
- Manages these pods as a group
  - Updating
  - Scaling
  - Rollback
- Manages desired state vs actual state

Deployments

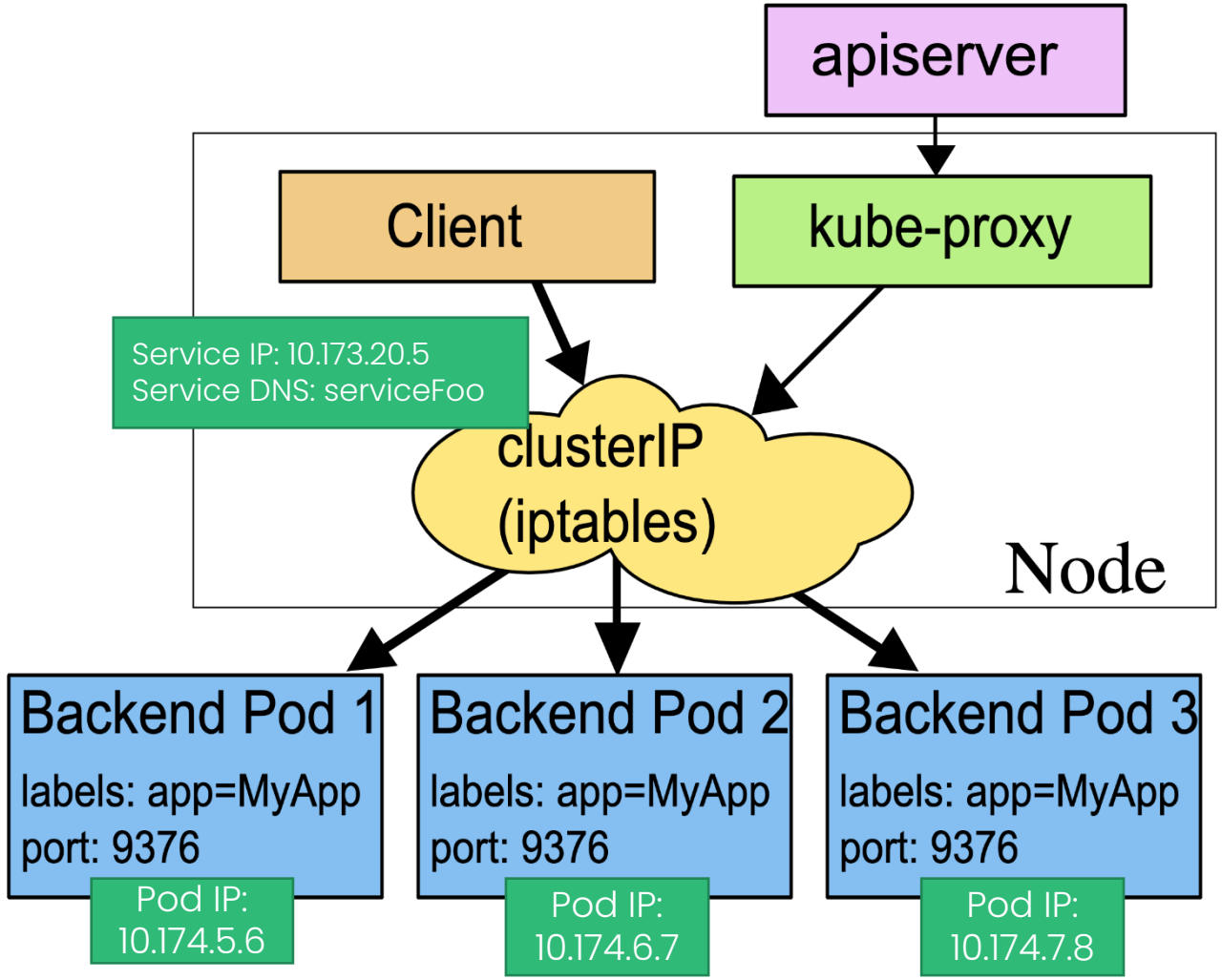
```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
  labels:
    app: nginx
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.7.9
```



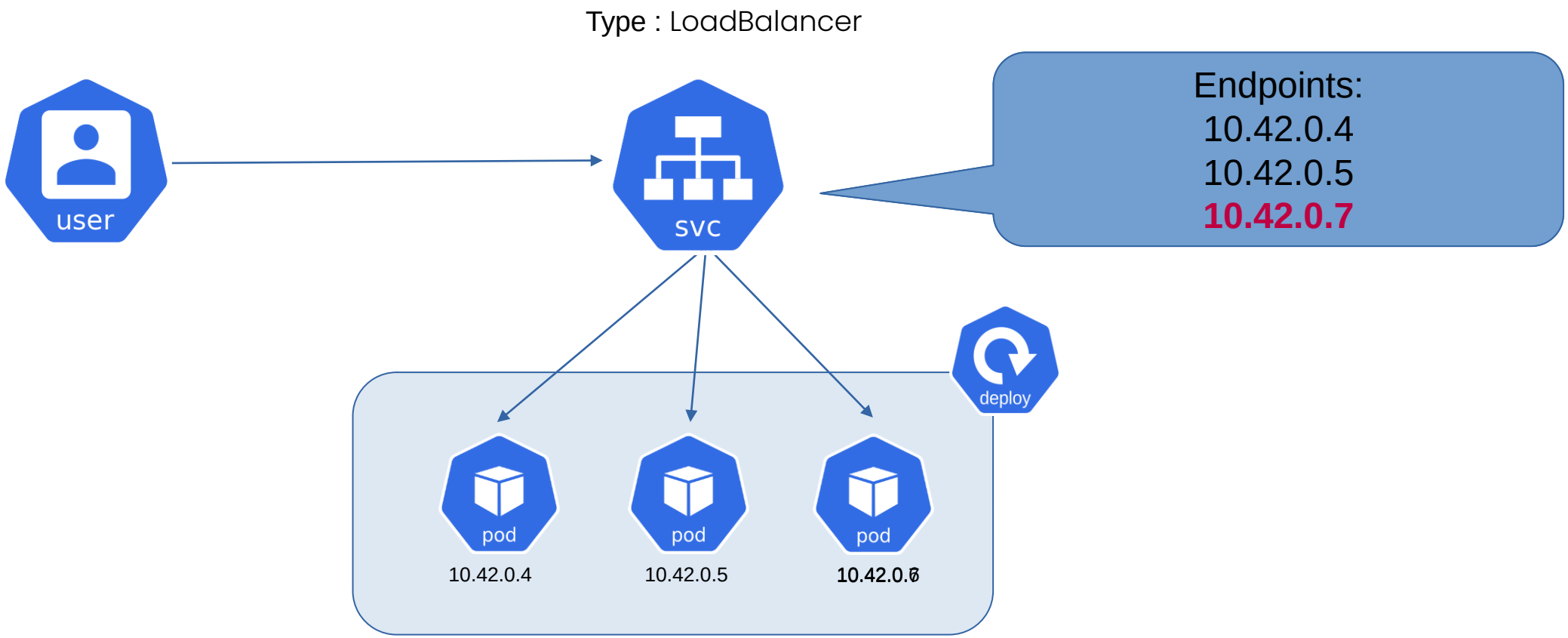
Services

- Exposes an application running on a set of pods
- Reliable, static reference for objects such as deployments
- Has its own IP address / DNS record
- Pods are **mortal** – they have their own life-cycle
  - Don’t address them directly
- Services can be Internal (ClusterIP) or External facing (NodePort & Loadbalancer)

Service Type ClusterIP (VIP)



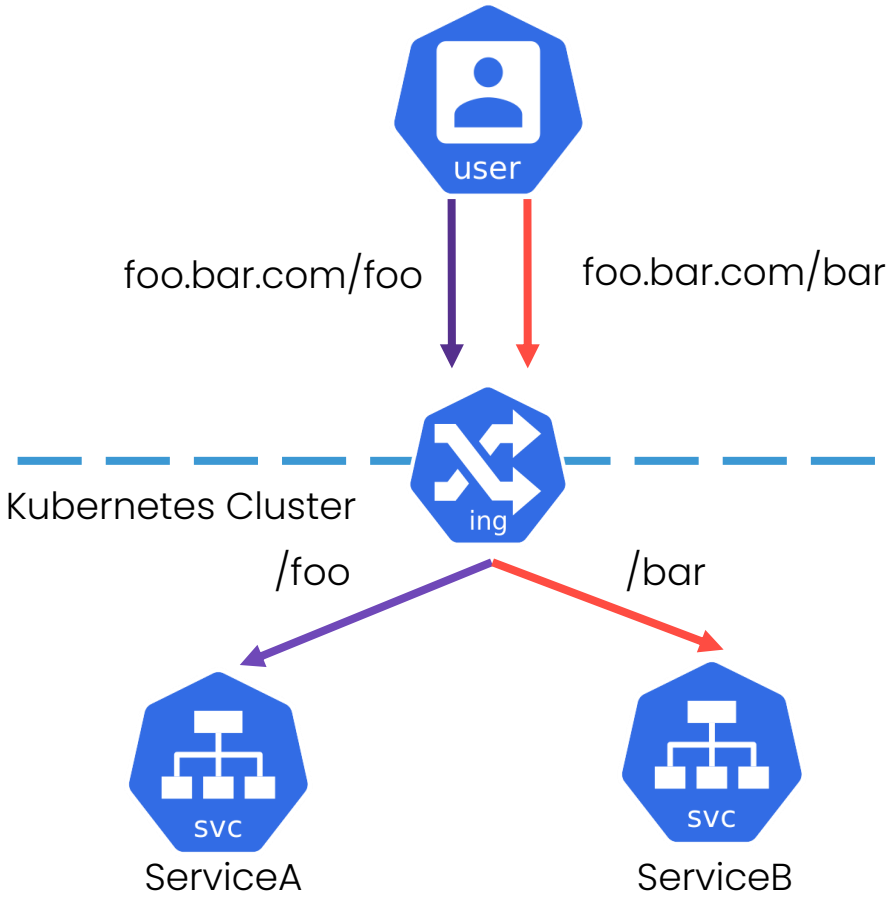
Service Type LoadBalancer



Ingresses

- Manages external access to services in a cluster
  - Commonly HTTP
- Gives services externally-reachable URLs
- Load Balance traffic
- Terminate SSL/TLS
- Facilitate name-based virtual hosting

Ingresses

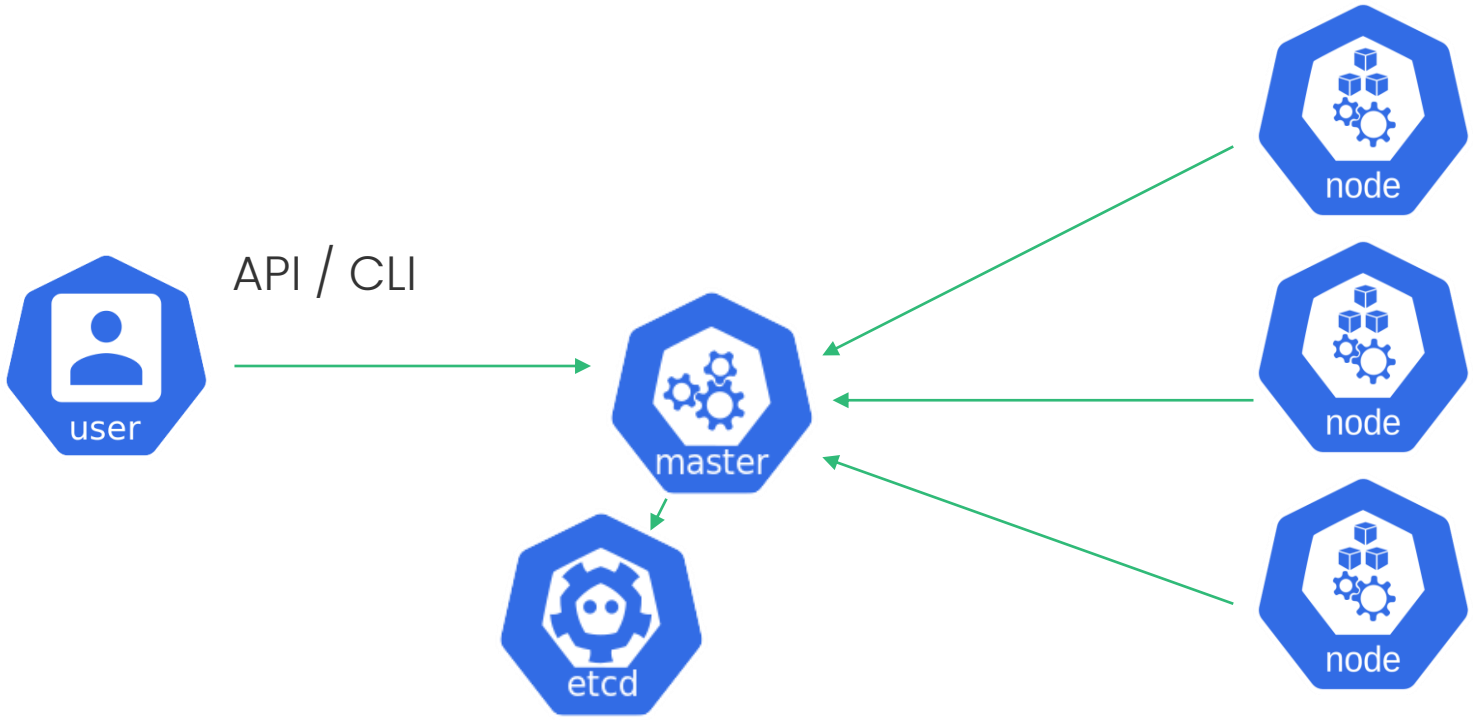



```yaml
apiVersion: networking.k8s.io/v1beta1
kind: Ingress
metadata:
  name: simple-fanout-example
spec:
  rules:
  - host: foo.bar.com
    http:
      paths:
      - path: /foo
        backend:
          serviceName: serviceA
          servicePort: 4200
      - path: /bar
        backend:
          serviceName: serviceB
          servicePort: 8080
```

Many more resources …

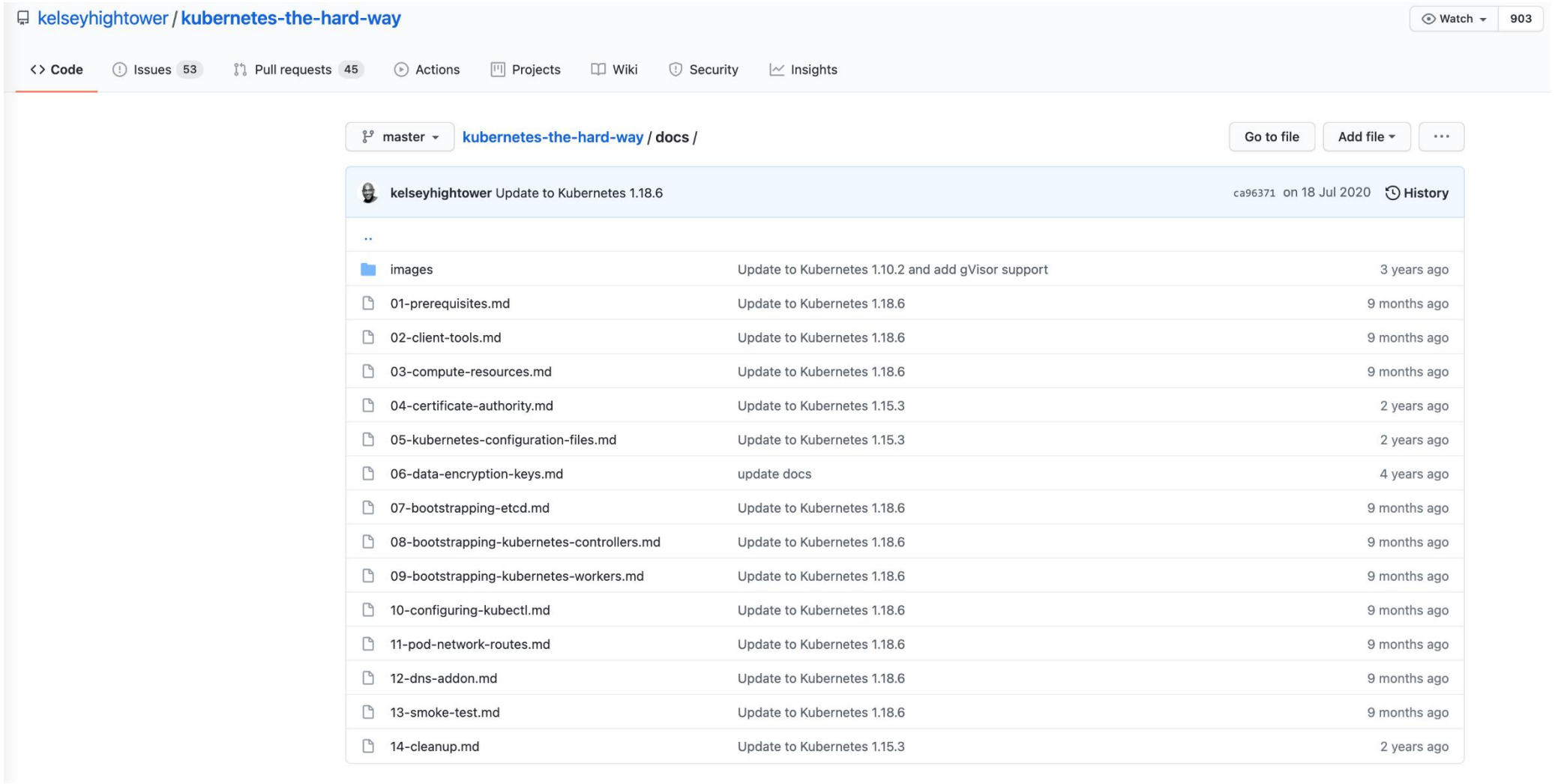
- ConfigMaps
- Secrets
- PersistentVolumes
- HorizontalPodAutoscalers
- CronJobs
- DaemonSets
- StatefulSets
- ….

Kubernetes architecture



- Controlplane: Manages the cluster and exposes an API for control
- Etcd: a key value store used as Kubernetes' backing store for all cluster data.
- Worker: Runs workloads and all of the supporting components.

Setting up Kubernetes is hard



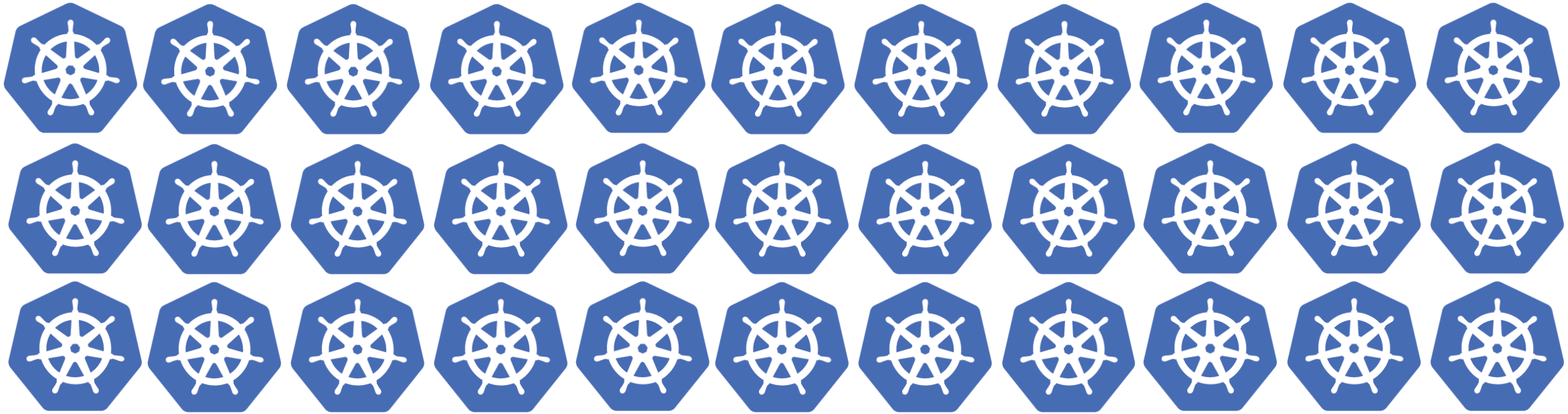
You don’t compile Linux from scratch, you use a distribution

Rancher Kubernetes Engine

- 100% Upstream Kubernetes
- CNCF certified
- Easy installation
- Zero-downtime upgrades
- Backup & Disaster Recovery
- Air gapped installation support

RKE

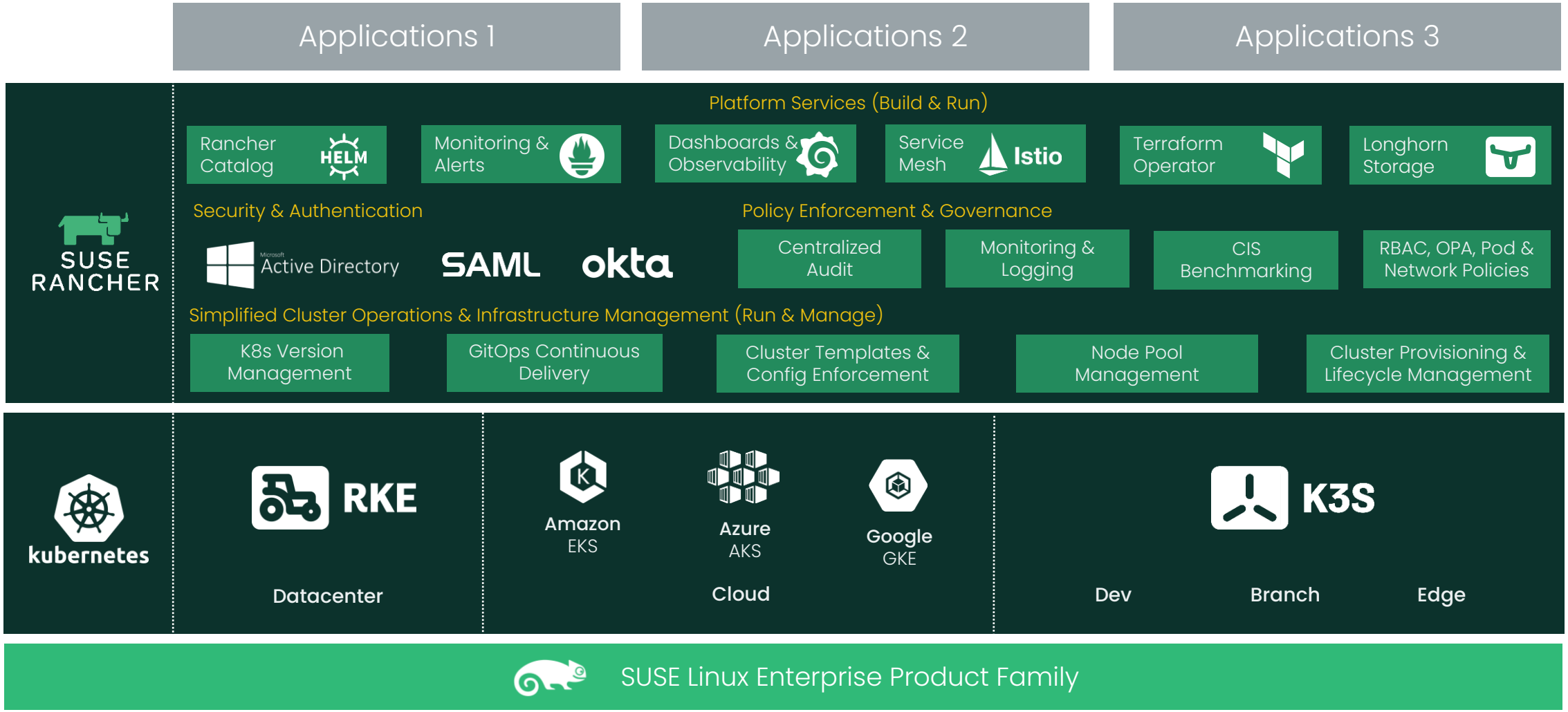
Kubernetes clusters are great……but



How about managing many?

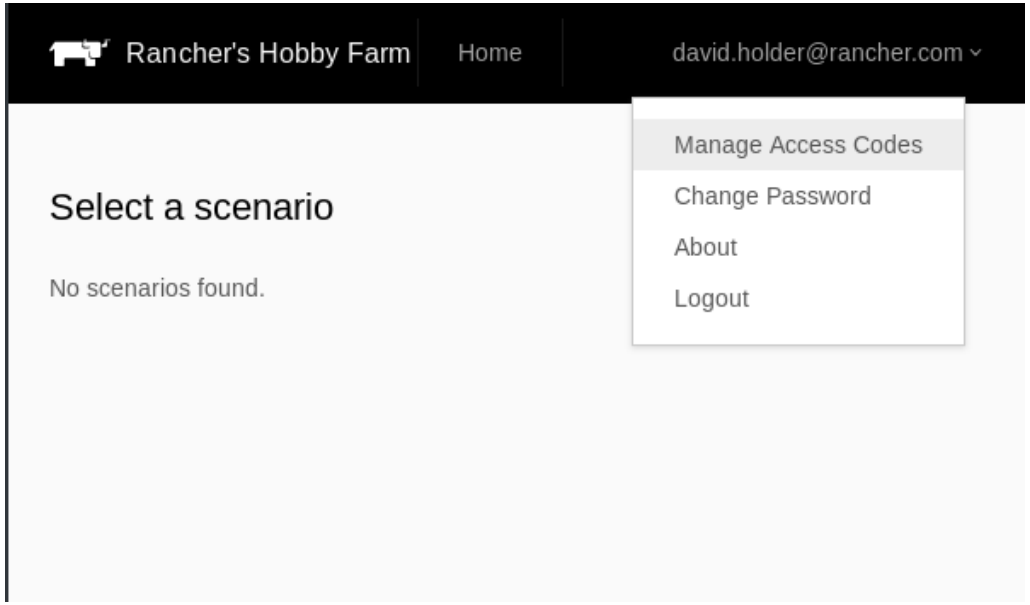
- Different environments
- Different teams
- Different hardware
- Different locations
- Edge devices

SUSE Rancher – Manage Kubernetes Everywhere



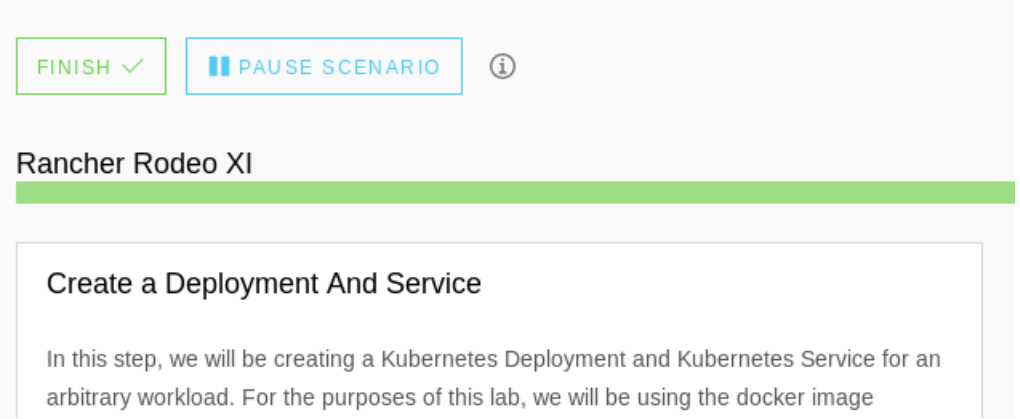
Let’s try it out

HobbyFarm



- Navigate to learn.eu.hobbyfarm.io
- Register with your e-mail and password
- Access Code: rodeo1012
- Note: Browser refresh may be required
- If no scenario shows up, check access code

HobbyFarm Scenario Agenda



Part 1: Rancher Installation

- Generating SSH Keys
- Installing required tools (RKE, Kubectl, Helm)
- Creating the management cluster
- Deploying Cert-Manager and Rancher

Pause for the coffee break at the “Accessing Rancher” step

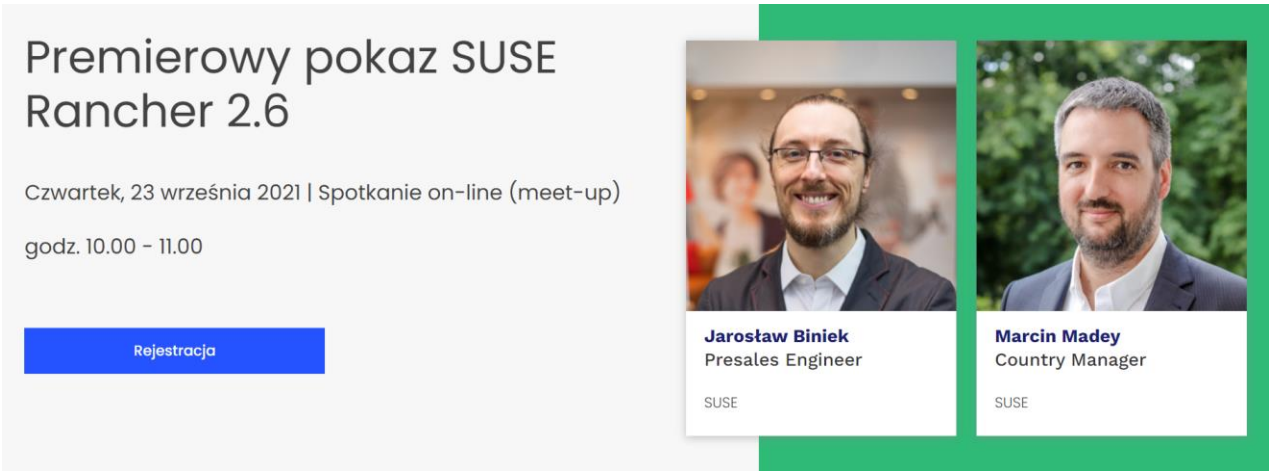
Coffee break

Until 11:15

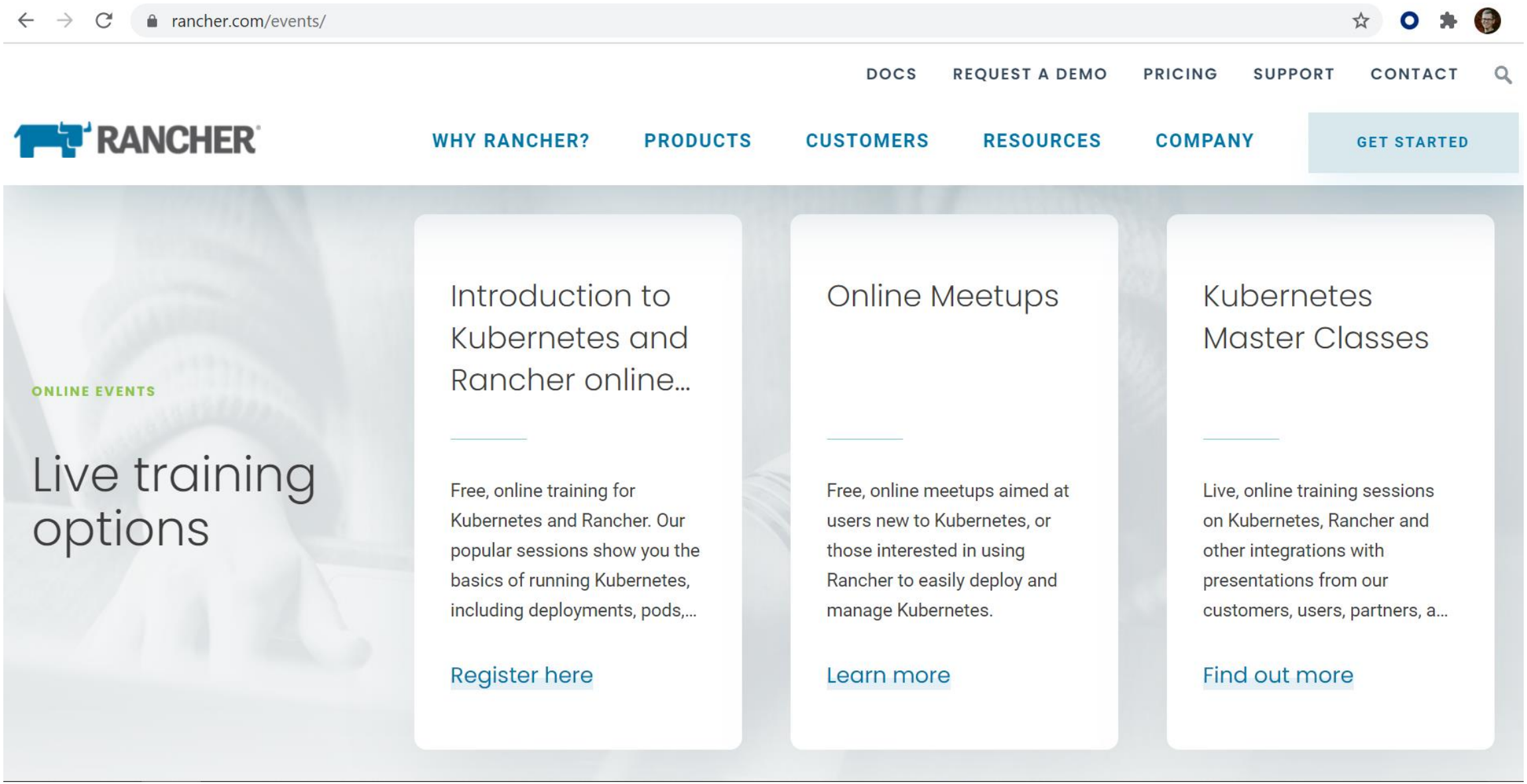
Watch on-demand

Online Meetup

SUSE Rancher 2.6
Launch in Poland

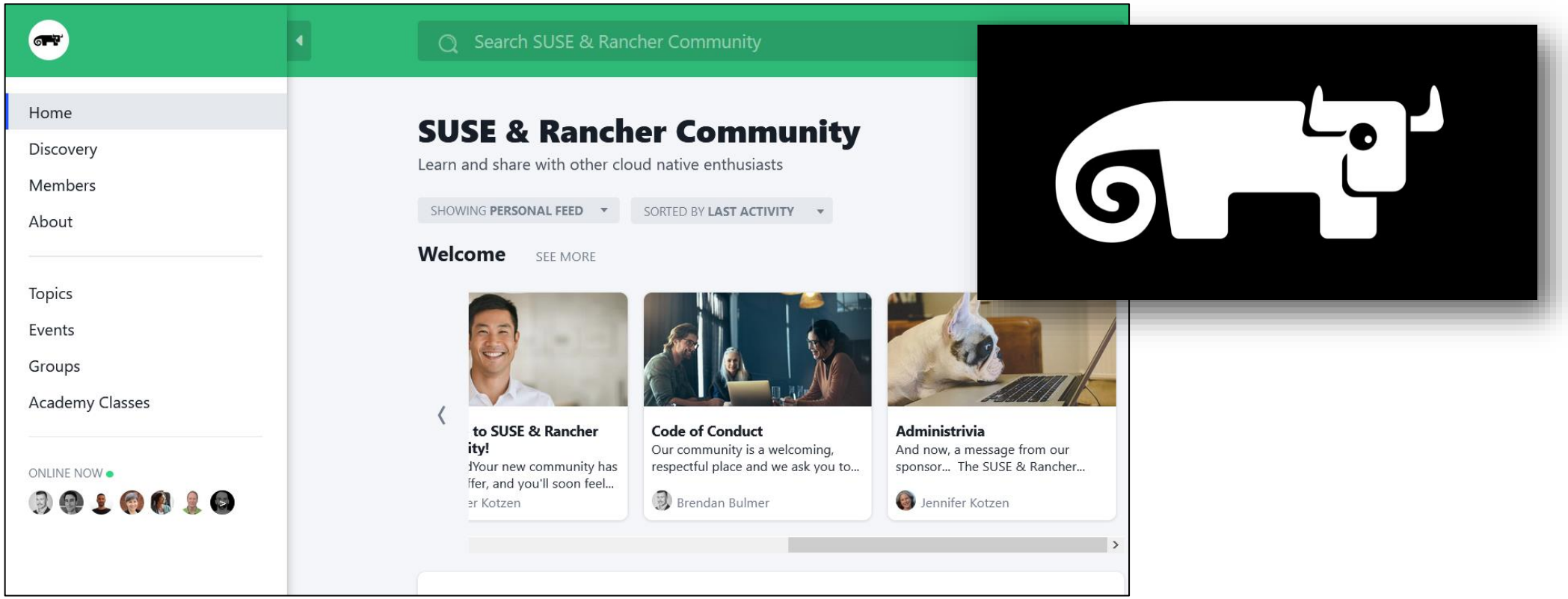


Available on-demand: https://linkd.pl/paedc



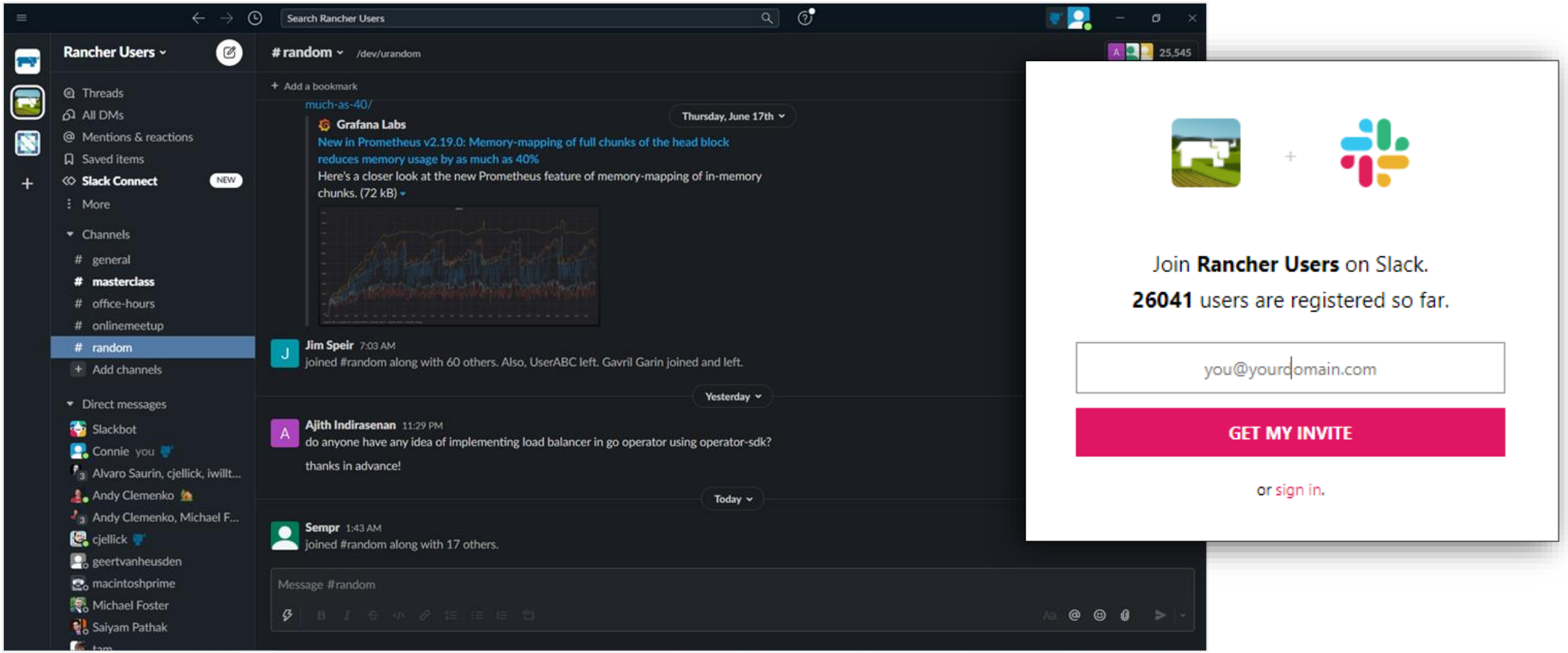
Join the SUSE & Rancher Community

https://community.suse.com



Join the Rancher Slack Network

http://slack.rancher.io



Thank you

For more information, contact SUSE at:

+48 22 537 5020 (Poland)

+49 (0)911-740 53-0 (Worldwide)

SUSE Polska Sp. z o.o.

Postępu 21, 02-676 Warszawa

www.suse.com