APPUiO CLOUD

Making of a Swiss PaaS

APPUiO SWISS CONTAINER PLATFORM

Adrian Kosmaczewski, Developer Relations

Speaker notes

Welcome to this presentation about APPUiO Cloud.

Agenda

1. Introduction to Cloud Native & Devops
2. Containerization and deployment on Kubernetes
3. Kubernetes distributions
4. APPUiO Cloud
5. Break
6. Workshop

Speaker notes

The agenda for today is the following. First we're going to learn how Cloud Native and DevOps are shaping the world of software development in the cloud.

Then we will see how to containerize applications, what it means, and how to make them run in Kubernetes.

About Kubernetes, we will see the different flavors of it, how to relate with one another, and then we'll talk a bit about Red Hat OpenShift.

This will lead us to learn more about APPUiO Cloud, and of course I will try to reply to any question you might have.

Then we're going to have a 15 minute break, and when we return we'll put all of this knowledge in action, using APPUiO Cloud to deploy an application.

What is APPUiO Cloud?

APPUiO CLOUD

Speaker notes

So the first question that comes to your mind is, what is APPUiO Cloud? Well, it’s a "Platform as a Service" environment, or actually better yet, "OpenShift Project as a Service".

What is OpenShift?

OPENSHIFT

Speaker notes

Which bears the question, what is OpenShift? Well, it's a container platform created by Red Hat and based on Kubernetes.

What is Kubernetes?

Speaker notes

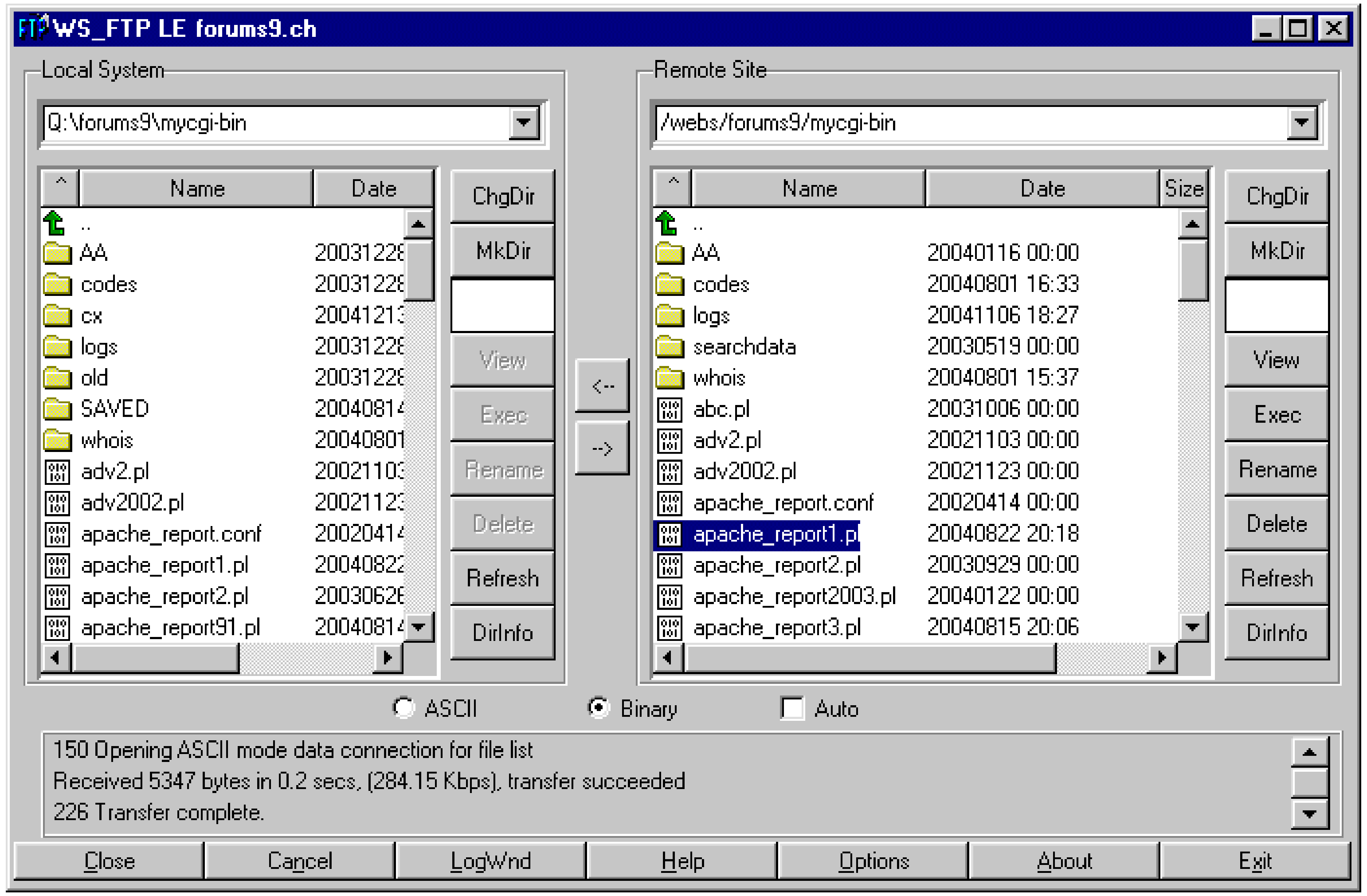
Which makes us ask, what is Kubernetes? It's a container orchestration platform created by Google.

What is a Container?

Speaker notes

So… what is a Container? Ah, well, I guess there’s a few things to talk about first!

Deploying Applications in 1997



Q:\forums9\mycgi-bin

Name	Date
..	
AA	2003122
codes	2003122
cx	2004121
logs	2003122
old	2003122
SAVED	2004081
whois	20040801
adv2.pl	2002110
adv2002.pl	2002112
apache_report.conf	2002041
apache_report1.pl	2004082
apache_report2.pl	2003062
apache_report91.pl	2004081

/webs/forums9/mycgi-bin

Name	Date	Size
..		
AA	20040116 00:00	
codes	20040801 16:33	
logs	20041106 18:27	
searchdata	20030519 00:00	
whois	20040801 15:37	
abc.pl	20031006 00:00	
adv2.pl	20021103 00:00	
adv2002.pl	20021123 00:00	
apache_report.conf	20020414 00:00	
apache_report1.pl	20040822 20:18	
apache_report2.pl	20030929 00:00	
apache_report2003.pl	20040122 00:00	
apache_report3.pl	20040815 20:06	

```
150 Opening ASCII mode data connection for file list
Received 5347 bytes in 0.2 secs, (284.15 Kbps), transfer succeeded
226 Transfer complete.
```

Speaker notes

When I started my career as a software developer in 1997, deploying an application on the "cloud" (we didn’t call it that way back then) involved literally uploading files to a server, and refreshing a browser. Of course all of this was very prone to errors and issues, like for example two developers uploading the same file and overwriting each other’s code.

Did I mention we did not have source control? There was no Git, no GitHub, no GitLab, no Visual Studio Code, no nothing. PHP was in its infancy, so we used whatever Microsoft gave us to build stuff.

"The Cloud"

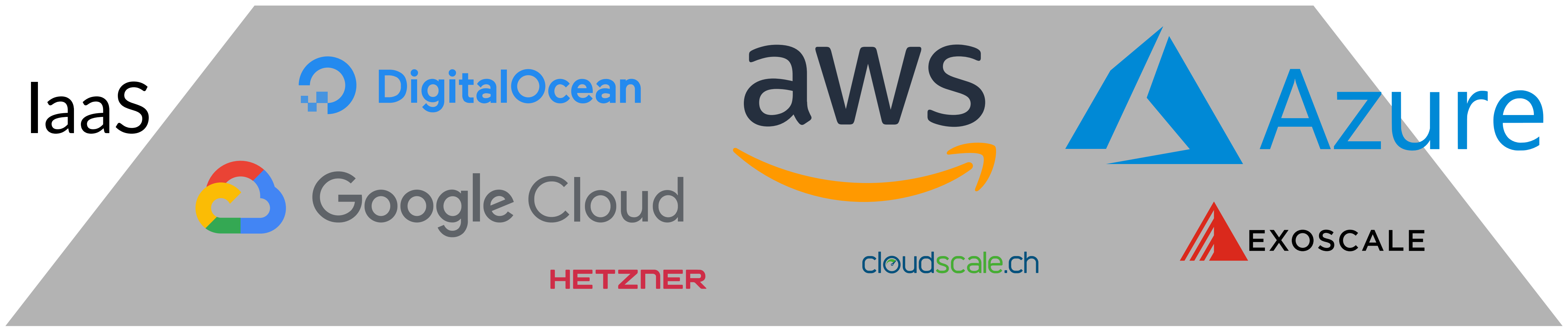
- IaaS
- PaaS
- SaaS

Speaker notes

These days we call "Cloud Native Applications" those apps created specifically to run in the Cloud. That could be a public cloud provider, a hyperscaler, or some other platform.

There are three basic words that appear all the time when you talk about Cloud Native Apps: IaaS, PaaS, SaaS. Let’s see each of them in detail to understand where we are, and where APPUiO Cloud fits.

IaaS



Speaker notes

Let’s organize all of the services on the Cloud in a nice pyramid. At the lowest level we have IaaS, or "Infrastructure as a Service". In this level we find all of the big hyperscalers like Amazon, Google, and Microsoft. Their primary customer is a system administrator, system architect, or app operator, who needs resources to build new services in the cloud.

IaaS Resources

- Storage
- Compute
- Networking
- Security

Speaker notes

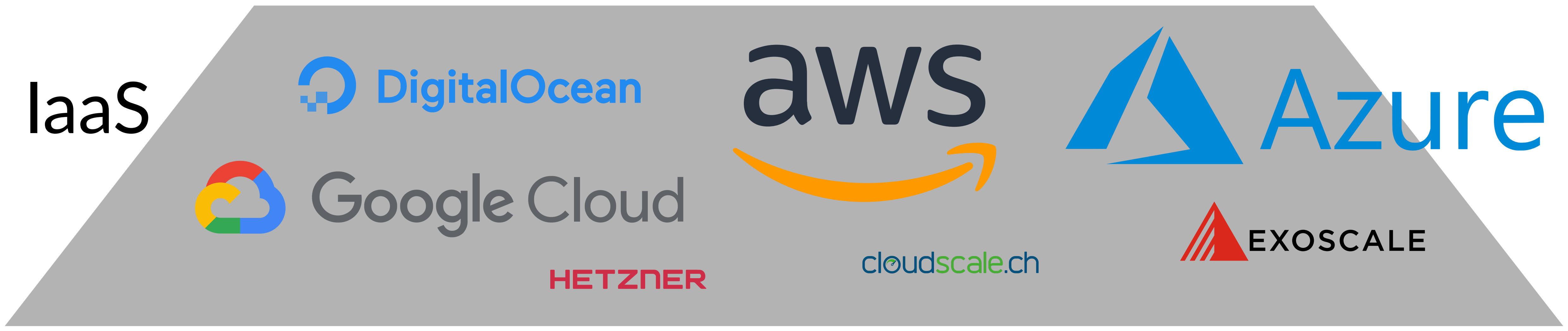
All IaaS providers always, always, always offer these four basic building blocks to create new systems:

Storage resources, to be able to keep large datasets (petabytes!) online with convenience and securely.

Compute resources, to run apps that perform calculations on those data.

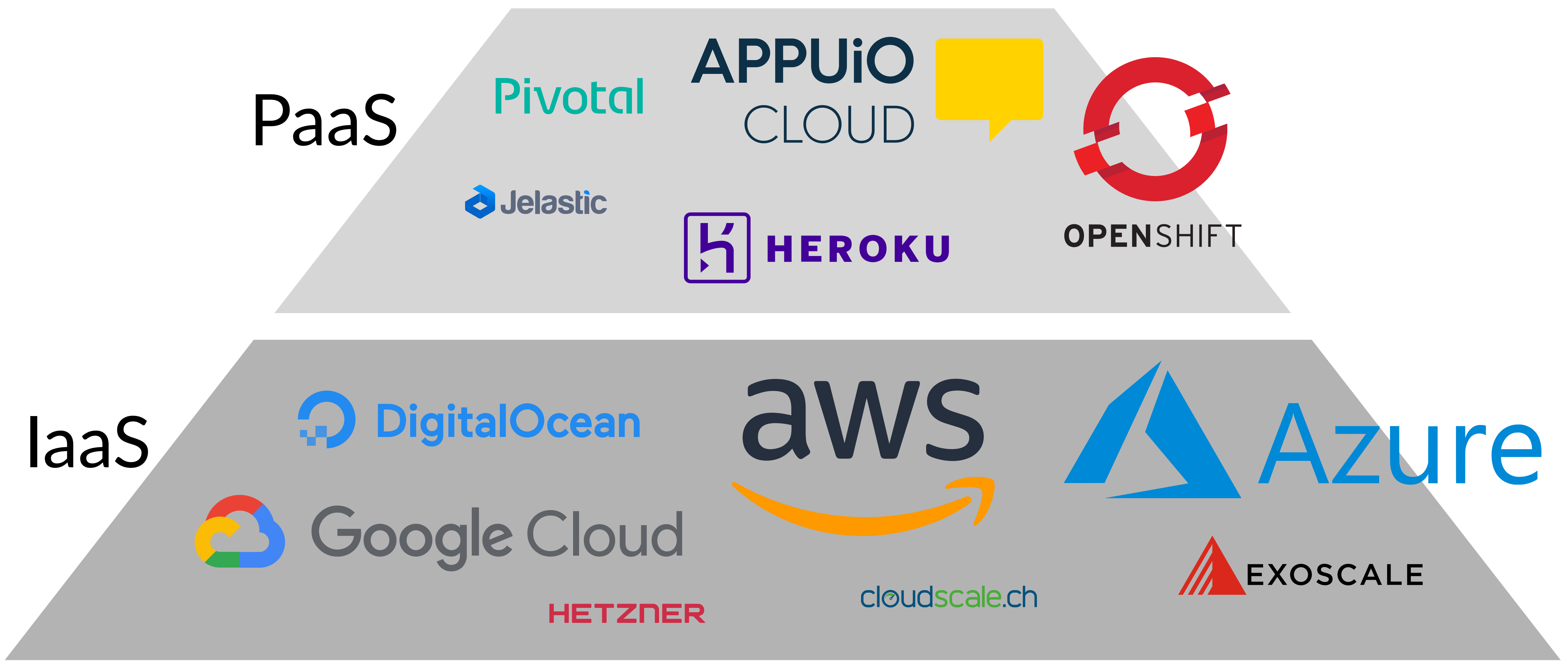
Networking resources, to make those apps available to the Internet and to other apps as APIs.

And security mechanisms, so that we can specify when, how, and by whom data is accessed.



Speaker notes

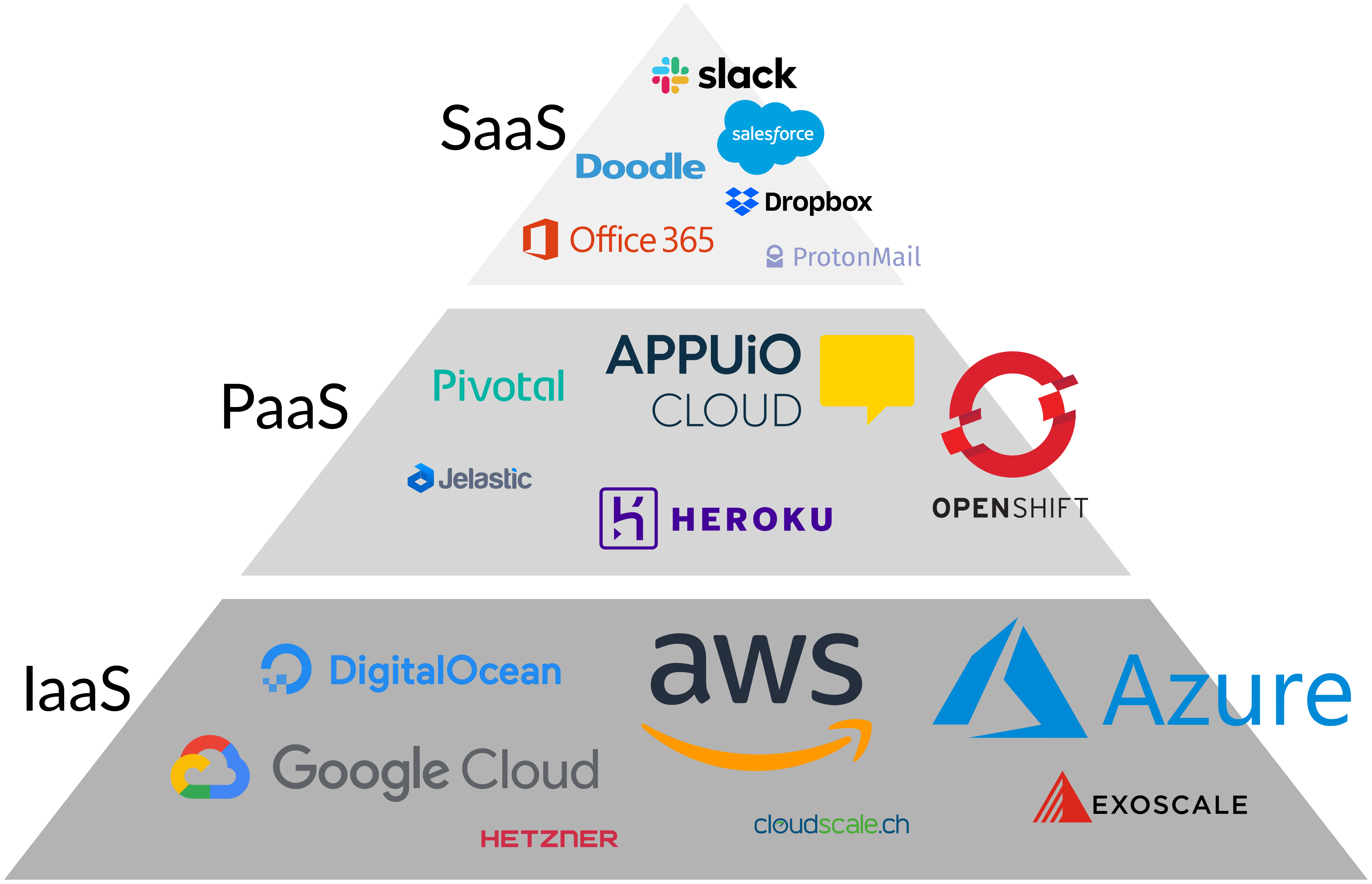
So again, this is the first level of cloud services.



Speaker notes

Let's go back to our original diagram. On top of IaaS we find PaaS, or "Platform as a Service". That's where we find APPUiO Cloud, see the logo over there. PaaS are built for developers, who do not want to spend time launching compute resources and wiring networking policies, they just want to write and run code!

PaaS offer command line tools, nice user interfaces, and even software development kits (SDKs) for developers to create applications with.



Speaker notes

And on top of the pyramid, we have SaaS, or "Software as a Service". These are geared to users; they do not want to build stuff, they want to get things done. They need to create and share documents, collaborate, exchange ideas, and for that they use more and more cloud services.

This is the final pyramid of cloud services in the public Internet.

What is a Cloud Native App?

- Architecture to build applications built with cloud computing in mind
- Cloud-native ≠ cloud only ⇒ public cloud **and** on-premises
- Focus on interconnected (micro-)services
- Enabled by open source implementations and open standards

Speaker notes

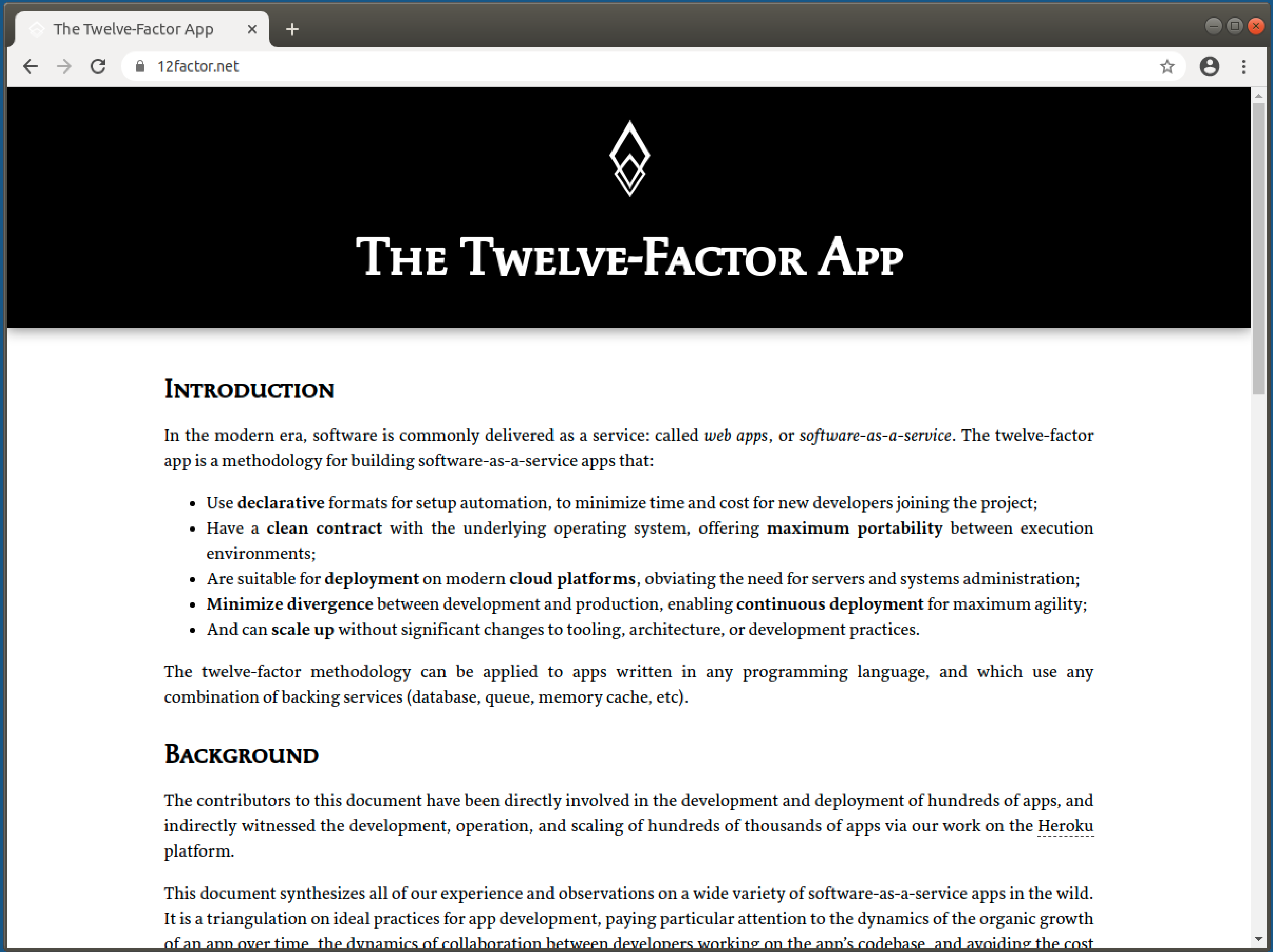
In this new Cloud paradigm, we need a new type of application, and hence was born the concept of Cloud Native applications.

They are apps that are built with those cloud concepts in mind.

However, that does not mean that they only run on AWS or Azure; they can run as well on "private clouds", that is, private datacenters that offer enterprise developers with similar tools and opportunities as a hyperscaler.

Usually, Cloud Native apps are built around microservices. Each component in a microservice architecture is small, focused, doing one and only one task.

Finally, Open Standards (and Open Source implementations thereof) are at the heart of this technology world.



The Twelve-Factor App

12factor.net

The Twelve-Factor App

Introduction

In the modern era, software is commonly delivered as a service: called *web apps*, or *software-as-a-service*. The twelve-factor app is a methodology for building software-as-a-service apps that:

- Use **declarative** formats for setup automation, to minimize time and cost for new developers joining the project;
- Have a **clean contract** with the underlying operating system, offering **maximum portability** between execution environments;
- Are suitable for **deployment** on modern **cloud platforms**, obviating the need for servers and systems administration;
- **Minimize divergence** between development and production, enabling **continuous deployment** for maximum agility;
- And can **scale up** without significant changes to tooling, architecture, or development practices.

The twelve-factor methodology can be applied to apps written in any programming language, and which use any combination of backing services (database, queue, memory cache, etc).

Background

The contributors to this document have been directly involved in the development and deployment of hundreds of apps, and indirectly witnessed the development, operation, and scaling of hundreds of thousands of apps via our work on the Heroku platform.

This document synthesizes all of our experience and observations on a wide variety of software-as-a-service apps in the wild. It is a triangulation on ideal practices for app development, paying particular attention to the dynamics of the organic growth of an app over time, the dynamics of collaboration between developers working on the app's codebase, and avoiding the cost

Speaker notes

To build Cloud Native applications, it would be hard to find a better guide than the 12 Factor website at 12factor.net.

The principles in this website provide a good practical starting point for developers looking forward to create modern Cloud Native applications.

Twelve-Factor Patterns

- Declarative formats ⇒ automation
- Portable across environments
- Suitable for deployment on modern cloud environments
- Minimize divergence between "dev" & "prod" ⇒ continuous deployment
- Built with scaling in mind

Speaker notes

For example, among the various principles, we find that Cloud Native apps should be described with textual formats, which leads to automation.

Cloud Native apps should be portable across environments; that is, ideally we should be able to avoid platform lock-in and be able to migrate to other cloud providers as required.

These applications should be built with those cloud environments in mind, in terms of security, connectivity, latency, and other factors.

There should be almost no differences between development and production environments, and ideally, there should be a continuous flow of applications from one environment to the other.

Finally, they should be scalable. Scalability should be a primary concern, so that apps can respond and adapt to traffic spikes.

DevOps

- Collaboration between development and operations
- Maximum automation through "infrastructure as code"
- Cost efficient ⇒ Lean
- Agile ⇒ react to changing requirements faster
- Continuous improvement built-in

Speaker notes

But what kind of teams are able to create Cloud Native apps? The "DevOps" movement, itself an evolution from the ideas of the Agile Manifesto in 2000, brings a social framework for such applications to exist.

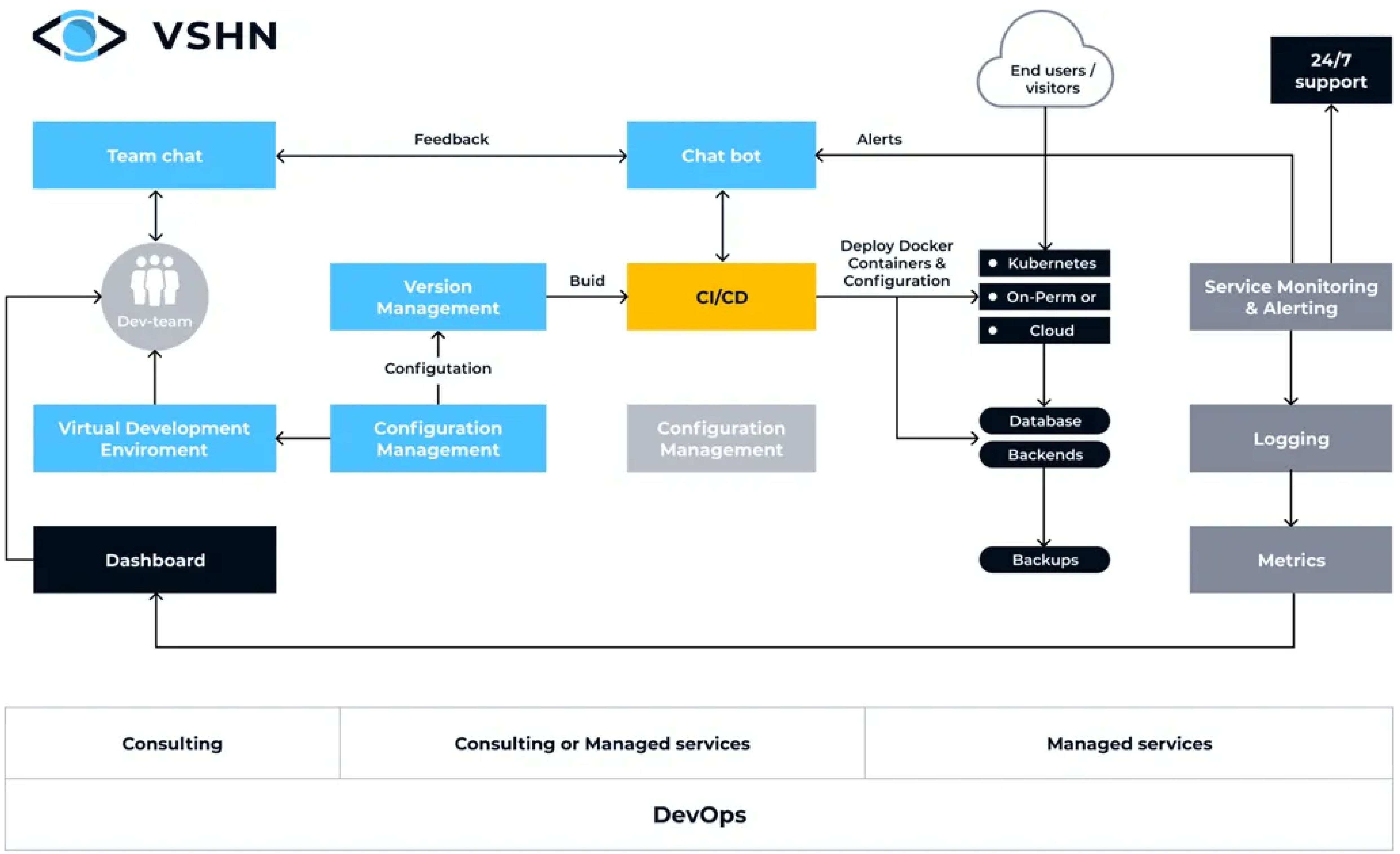
First of all, by ensuring communication between development and operations. There shouldn’t be barriers among those different groups.

The use of automation and "infrastructure as code" ensures that every element in the infrastructure is written down in text files, versioned in Git repositories, and handled with automated tools.

Being lean means being cost efficient, taking advantage of automation to lower costs.

Being agile means being more reactive to the demands of customers and stakeholders.

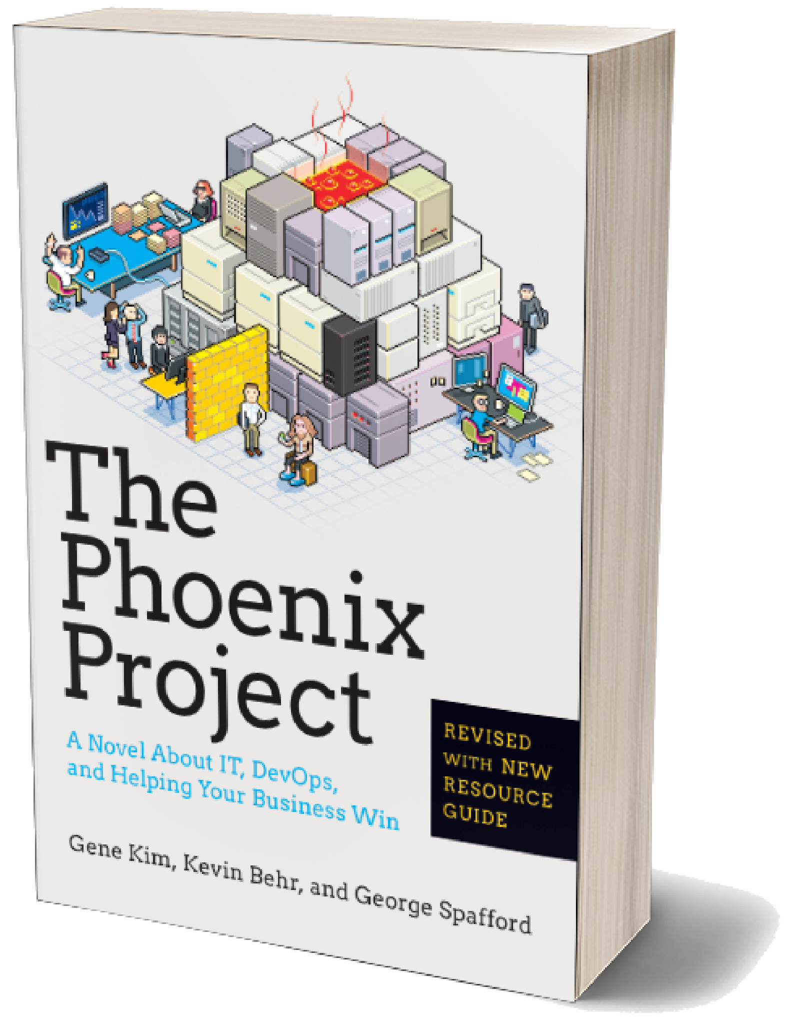
And finally, improving continuously with new features, bug fixes, in a continuous flow of work from development to production.



Speaker notes

At VSHN we use this diagram to represent our work based on the DevOps principles.

Book Recommendation


The Phoenix Project
A Novel About IT, DevOps, and Helping Your Business Win
REVISED WITH NEW RESOURCE GUIDE
Gene Kim, Kevin Behr, and George Spafford

Speaker notes

If you would like to learn more about DevOps, the most important book I can recommend is "The Phoenix Project", a novel about DevOps that has become a classic in the genre. It is very well written!

Best Practices for Cloud-Native

- Componentize apps in microservices ⇒ Horizontal scalability & team scaling
- Automate the release pipeline ⇒ Faster time-to-market
- Use infrastructure as code ⇒ Repeatability & testability
- Increase application observability ⇒ Resilience
- Automate app lifecycle ⇒ Increased security & availability

Speaker notes

In summary, when you want to create a Cloud Native application you should follow these best practices.

But regarding the first point in the list, how can you componentize your apps in microservices? Turns out, containers are the solution for that.

Docker and Application Containers

Speaker notes

Containers, popularized by the Docker tool, are the first and most important mechanism used to create microservices today.

Speaker notes

Of course the name "container" reminds us of these things, seen at harbours and train exchanges all over the world. Why do we use the word "containers" for software nowadays?

Shipping Containers

- Standard shipping container
- Built for "intermodal" freight transport
- Help reduce cost and times of transport
- Universal standard open to anyone
- Ecosystem of vehicles, training, accessories, tools…

Speaker notes

It turns out that the invention of standard shipping containers have been one of the major inventions in trade of the 20th century.

They are "intermodal", which means that you can put them on a train, on a boat, on a truck, and they will work; it does not matter what they contain, they can be moved from support to support.

This of course reduces the cost of transport, making it faster and more reliable.

And it’s a universal standard, open to anyone, which means that companies that support it can immediately join a large…

… ecosystem of vehicles, trainings, accessories, tools, etc.

Application Containers

- Standard (code) shipping containers
- Bundle: app + dependencies + libraries + runtimes + configuration + …
- Runtime isolation through OS-level virtualization
- Not a new idea: Version 7 Unix "chroot" (1979-1982), FreeBSD "jails" (2000), LXC (2008), Docker (2013)

Speaker notes

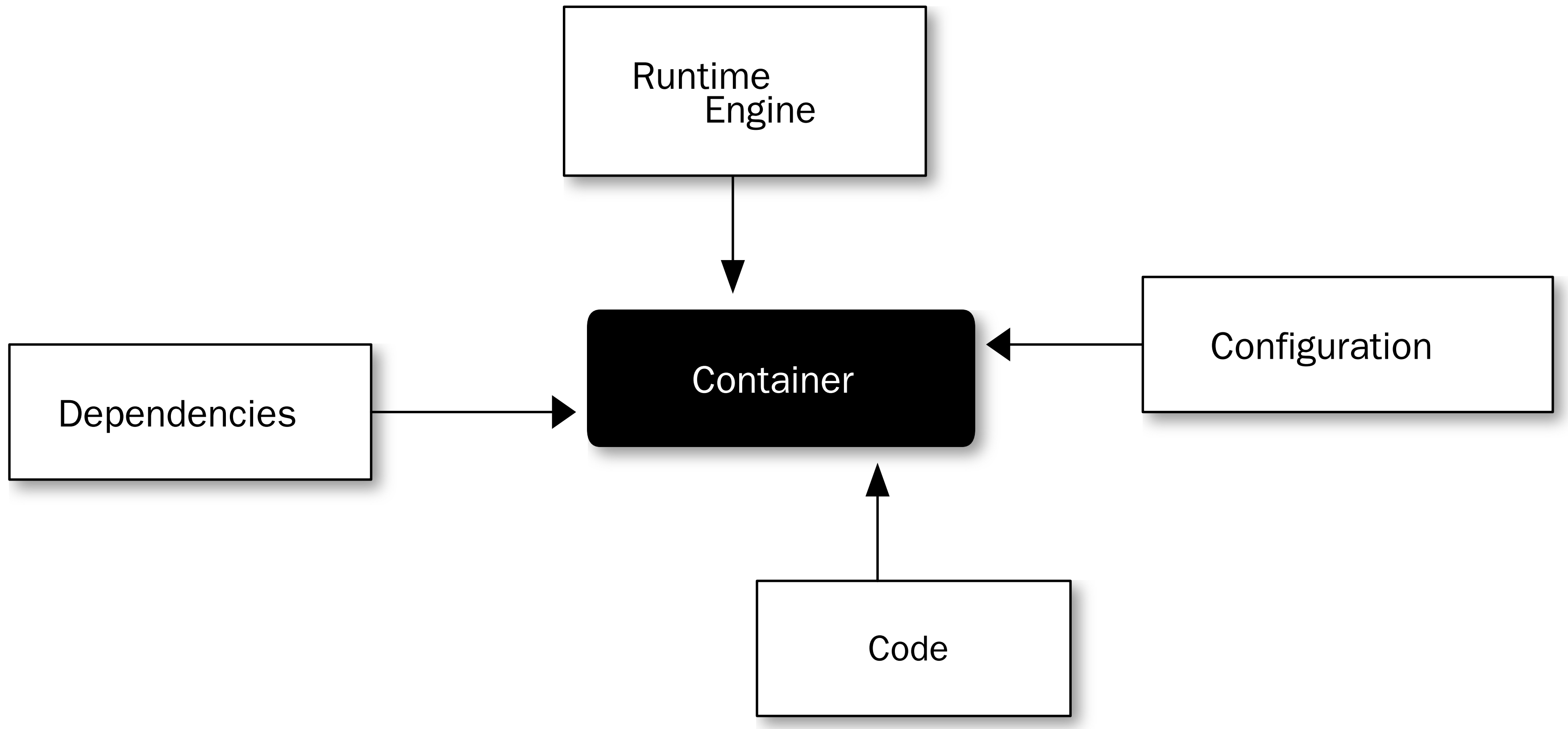
Software containers aim to reach the same kind of productivity for software as shipping containers did for international trade.

They are standardized by independent bodies, and their specs are open to anyone.

An application container is completely independent, and carries everything that it requires: libraries, runtimes, configurations, etc.

They provide some isolation, which means that their behaviour is guaranteed, and can be controlled from the outside.

And no, they aren’t really a good idea; Docker provided the final element required for them to become popular, but the `chroot` mechanism already existed in Unix Version 7, back in 1979!



Speaker notes

So sum it up: a container is an entity that encapsulates an application plus all of its dependencies and requirements.

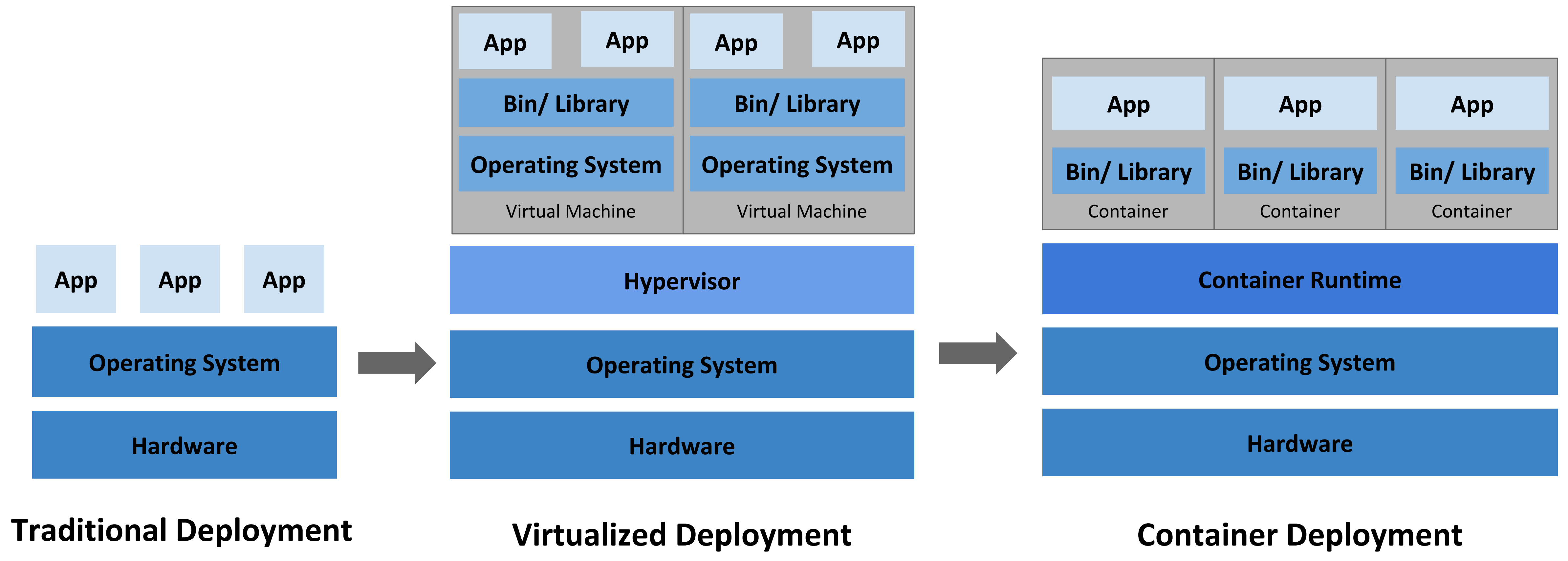
Tools

docker

podman

Speaker notes

The two most important tools used today by developers to create standard application containers are Docker and Podman, both free to use and very similar to one another.



Speaker notes

This diagram depicts the evolution of application hosting. On the left we have the situation in 1997, where we would have a hardware server in a datacenter, and we used FTP to deploy code into it.

In the 2000s virtualization became very popular, with solutions like VMWare and Vagrant, which provide isolation and much more flexibility.

But how are containers different from virtual machines? It turns out that containers share the kernel with the host, which makes them faster to start and stop, and of course, less hungry in resources. Every VM must have its own copy of the operating system kernel, of course.

As a downside, of course, containers must run the same operating system as the host; that means that, for example, Windows containers cannot run in Linux hosts, and vice-versa, while Windows virtual machines can run on Linux hosts.

Benefits of Containers

- Key element of collaboration & communication
- Continuous Integration ⇒ Continuous Deployment
- Speed: fast to start, fast to stop
- Portable: across operating systems, hardware, cloud platforms, environments…
- Isolation: controlled resource utilization
- Security: each container runs isolated from others

Speaker notes

Containers have so many benefits that the industry has adopted completely. Here’s a list of the major advantages.

"Containers" and "Images"

Docker Images

Template to create containers; one image can be used to create many containers.

Docker Containers

Running instance. Each container is created from one image.

Speaker notes

A common confusion among engineers new to containers is the distinction between "Docker Images" and "Docker Containers"; here’s the difference.

Example Service

```python
import os
import subprocess
from flask import Flask, jsonify
from subprocess import run, PIPE
from random import randrange

app = Flask(__name__)
version = '1.2-python'
port = 8080
hostname = subprocess.check_output('hostname').decode('utf8')

@app.route("/")
def fortune():
    number = randrange(1000)
    fortune = run('fortune', stdout=PIPE, text=True).stdout
    return jsonify({ 'number': number,
                     'message': fortune,
                     'version': version,
                     'hostname': hostname })

if __name__ == "__main__":
    app.run(host='0.0.0.0', port=os.environ.get('listenport', port))
```

Speaker notes

Here we have a very simple web API written in Python. A perfect candidate to be containerized!

Requirements

```
click==8.0.3
Flask==2.0.2
itsdangerous==2.0.1
Jinja2==3.0.3
MarkupSafe==2.0.1
Werkzeug==2.0.2
```

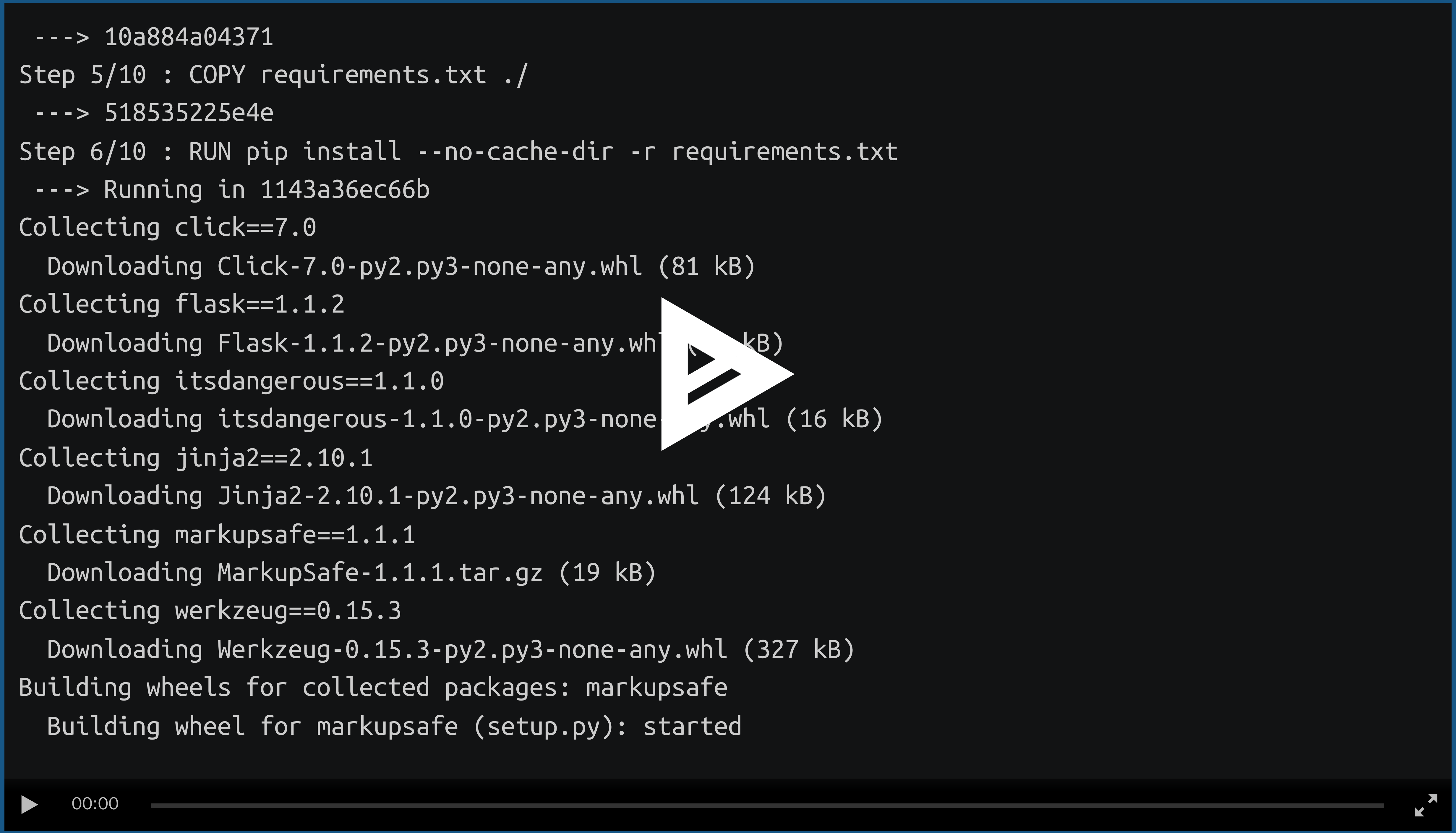
Speaker notes

As most software projects, this Python application has some dependencies, which are handled by `pip`, the standard package manager of the Python world.

```dockerfile
# Inherit from this "empty base image"
# https://hub.docker.com/_/python/
FROM python:3.7-alpine

# Install some required software on Alpine Linux
RUN apk add fortune

# Directory to install the app inside the container
WORKDIR /usr/src/app

# Install python dependencies
# (cached if requirements.txt does not change)
COPY requirements.txt ./
RUN pip install --no-cache-dir -r requirements.txt

# Copy application source code into container
COPY app.py .

# Expose this TCP-port, same as app.py
EXPOSE 9090

# Drop root privileges when running the application
USER 1001

# Run this command at run-time
CMD [ "python", "app.py" ]
```

Speaker notes

To create a new container image for this Python API application, we need a `Dockerfile` like the one shown here.



```
 ---> 10a884a04371
Step 5/10 : COPY requirements.txt ./
 ---> 518535225e4e
Step 6/10 : RUN pip install --no-cache-dir -r requirements.txt
 ---> Running in 1143a36ec66b
Collecting click==7.0
  Downloading Click-7.0-py2.py3-none-any.whl (81 kB)
Collecting flask==1.1.2
  Downloading Flask-1.1.2-py2.py3-none-any.wh[illegible] kB)
Collecting itsdangerous==1.1.0
  Downloading itsdangerous-1.1.0-py2.py3-none[illegible].whl (16 kB)
Collecting jinja2==2.10.1
  Downloading Jinja2-2.10.1-py2.py3-none-any.whl (124 kB)
Collecting markupsafe==1.1.1
  Downloading MarkupSafe-1.1.1.tar.gz (19 kB)
Collecting werkzeug==0.15.3
  Downloading Werkzeug-0.15.3-py2.py3-none-any.whl (327 kB)
Building wheels for collected packages: markupsafe
  Building wheel for markupsafe (setup.py): started
```

asciinema.org/a/333322

Speaker notes

This demo movie available online at asciinema.org/a/333322 shows how to use the `Dockerfile` to create a new container image, and then how to run a new instance of a container with the new image. We see how it can be also pushed to Docker Hub, so that other developers can run it and use it if needed.

Sharing Dockerfiles

- Part of the source code
- Versioned in repository

Speaker notes

How do you share your `Dockerfile` with your colleagues? Very easy; you add them to your projects, as one more file for your team to include in your Git repository.

Sharing Container Images

- Through "Image Repositories"
- Make images available to team members, outside contributors, or other deployment environments
- Public & Private
- Ready to run

Speaker notes

Of course you might want to only share the container, and not the `Dockerfile` with your colleagues or customers; in those cases, you can use what is called a "Container Image Repository", where you can make images available to others, either in public or private fashion, and where they are ready to run.

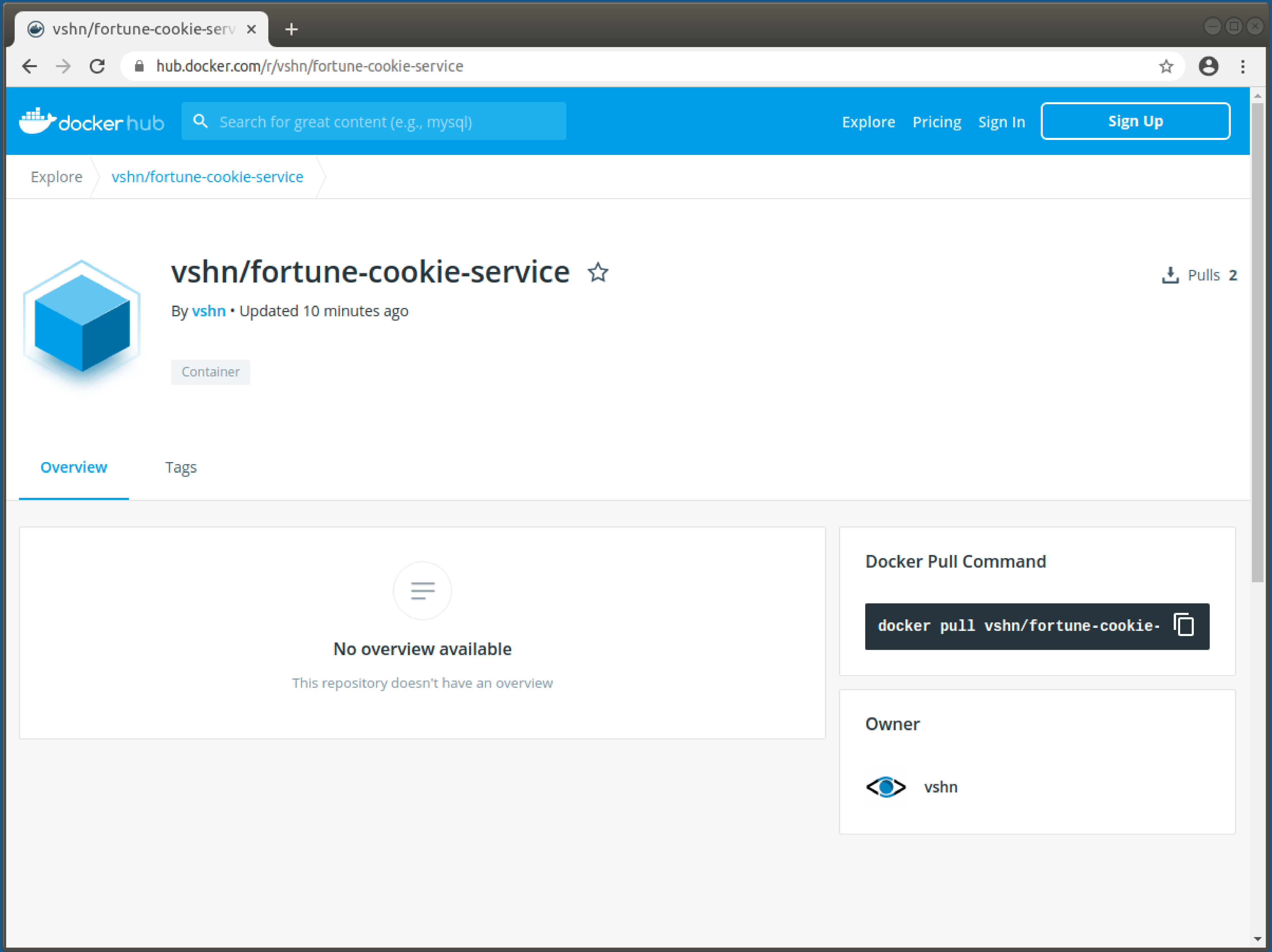
Container Repositories

- Public Cloud Services
  - Docker Hub
  - Red Hat Quay
  - Amazon ECR
  - GitHub
- On-Premises
  - GitLab
  - OpenShift
  - Harbor

Speaker notes

There are many container repositories available; some are public and free, available online; like Docker Hub, GitHub, or Red Hat Quay.

Others are installable on-premises, like for example Harbor or the ones included in GitLab and OpenShift. These ones are private, so that developers of a same organization can share containers with one another.


vshn/fortune-cookie-serv
hub.docker.com/r/vshn/fortune-cookie-service
docker hub
Search for great content (e.g., mysql)
Explore
Pricing
Sign In
Sign Up
Explore
vshn/fortune-cookie-service

vshn/fortune-cookie-service

By vshn • Updated 10 minutes ago

Pulls 2

Container

Overview Tags

No overview available

This repository doesn't have an overview

Docker Pull Command

```
docker pull vshn/fortune-cookie-
```

Owner

vshn

Speaker notes

This is for example a screenshot of Docker Hub, where a container is made available for others to use.

Kubernetes

Speaker notes

Once we have lots of containers, we must run them.
And this is where Kubernetes comes in.

Running Many Containers

- Most cloud applications have different components:
  - Databases
  - Web servers
  - Message queues
  - Monitoring systems
- ... all deployed in various environments: "dev", "test"...
- ... all connected to each other...
- ... and with redundancy!

Speaker notes

Complex Cloud Native applications are made of multiple containers: a database, web servers, application servers, message queues, monitoring systems, and all of these apps must be installed in various environments, they must all be connected to each other in similar ways, and they must all support redundancy and scalability! Clearly, it becames very complex to run all of this by hand.

How to Coordinate those Components?

- If each component is inside a container…
- … each with its own IP or port…
- … each loosely connected to one another…
- … we need a "Container Orchestrator"

Speaker notes

To run many containers together at once, in a repeatable fashion, and with automation, we need something that acts like a "container orchestrator."

What is Kubernetes?

- Usually referred to as "K8s"
- Container orchestrator platform originally created by Google
- Open Source and cross-platform
- Part of the CNCF ecosystem
- Latest version: 1.23.1 (December 16, 2021)

Speaker notes

And precisely, Kubernetes is a container orchestrator. It is not the only one (there are others like Docker Swarm and Apache Mesos) but it has become a standard in the industry. Pretty much every major application you use in the Internet these days is running in a Kubernetes cluster.

How do you pronounce it?

- Greek word: Κυβερνήτης or "Kivernitis"
  - Meaning: governor, commander, or captain
- Root of "government" and "cybernetics"

Speaker notes

Here’s the funny part; you’re going to hear lots of various pronounciations of the word, but this is the true one in Greek.

Kubernetes Terminology 1/3

Cluster

Group of machines working together.

Node

A machine (virtual or not) in a cluster.

Pod

Minimum unit of code execution, with one or many containers inside, and which can be scaled up and down. Ephemeral!

Speaker notes

Kubernetes manages a cluster of compute resources, usually virtual machines provided by some IaaS. Each of those resources is called a Node. One (or two) of the nodes in a Kubernetes clusters are called "Master Nodes", and their job is to work as the director of the orchestra (after all, it’s a container orchestration system, get it?). Master Nodes also expose the Kubernetes API, used by DevOps engineers to "talk" to the cluster and to manage it. The other nodes are "Worker Nodes", and their job is to run containers following the instructions of the Master Node(s), and that’s it.

Inside of those nodes, Kubernetes runs containers inside of small units called Pods. A pod can contain one or many containers, running together. Pods can be killed, scheduled, shuffled around from node to node, as required and as possible, depending on the conditions of the cluster. Kubernetes can for example restart pods if they crash, automatically. Containers inside of a pod can talk to each other using localhost and specifying the port number of each application exposed inside of a container.

Kubernetes Terminology 2/3

Deployment

A set of "Pods", "Services" and "Persistent Volumes" running in the nodes of a cluster.

Service

Used to expose a deployment to the internal network of the cluster.

Ingress

Exposes services to the wider Internet.

Speaker notes

Pods are grouped together into Deployments. A deployment specifies the identity and number of containers inside of each pod, and provides information to Kubernetes about replication, scheduling, and various types of metadata.

By default, deployments are not exposed to other deployments in the cluster, or to the outer world of the Internet; for that, you must create a Kubernetes Service. Services expose a network port and address for deployments to talk to one another. They can also provide load balancing, for applications to have a higher level of availability.

Services only work at the level of a cluster; to expose things to the outer world of the Internet, one must use an Ingress. Ingresses are Kubernetes objects that expose and create a route for applications to be reachable from the outer world.

Kubernetes Terminology 3/3

Persistent Volume

A unit of disk storage available for pods to save data.

Persistent Volume Claim

A declaration from a deployment about storage requirements, that Kubernetes tries to fulfill as best as possible.

Namespace

Mechanism for isolating groups of resources within a single cluster.

Speaker notes

In order to store data, deployments require Persistant Volume Claims or PVCs. They are requests for Persistant Volumes or PVs, that the cluster tries to fulfill if possible. DevOps engineers setting up a Kubernetes cluster can configure storage providers, so that PVCs are fulfilled appropriately. This level of abstraction clearly separates applications from the storage they require to save data.

Last but not least, namespaces provide a way to group objects within the same cluster. Object names must be unique in a namespace, but not across namespaces; we’re going to use this capability in the workshop today.

Kubernetes as an IaaS

- Compute services using Deployments and Pods.
- Storage with PVs and PVCs.
- Network with Services and Ingress.
- Security with… well, nothing really. Kubernetes does not provide any user management features!

Speaker notes

Remember what I said in the beginning; IaaS providers always provide basic building blocks, but Kubernetes only provides part of the solution.

User management is one of the features that the Kubernetes project creators decided to leave to implementers to add to their own Kubernetes distributions.

This makes the Kubernetes project very similar to the Linux Kernel project. The Linux kernel provides useful features like process scheduling and support for file systems, but if you want to edit documents, you’re better off installing a distribution that includes LibreOffice; and that’s what Red Hat Enterprise Linux or Ubuntu provide on top of the kernel.

Kubernetes Deployment

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: fortune-deployment
  labels:
    app: fortune-app
spec:
  template:
    spec:
      containers:
      - image: vshn/fortune-cookie-service:1.0
        name: fortune-container
        ports:
        - containerPort: 9090
          name: fortune-port
    metadata:
      labels:
        app: fortune-app
  selector:
    matchLabels:
      app: fortune-app
  strategy:
    type: Recreate
```

Speaker notes

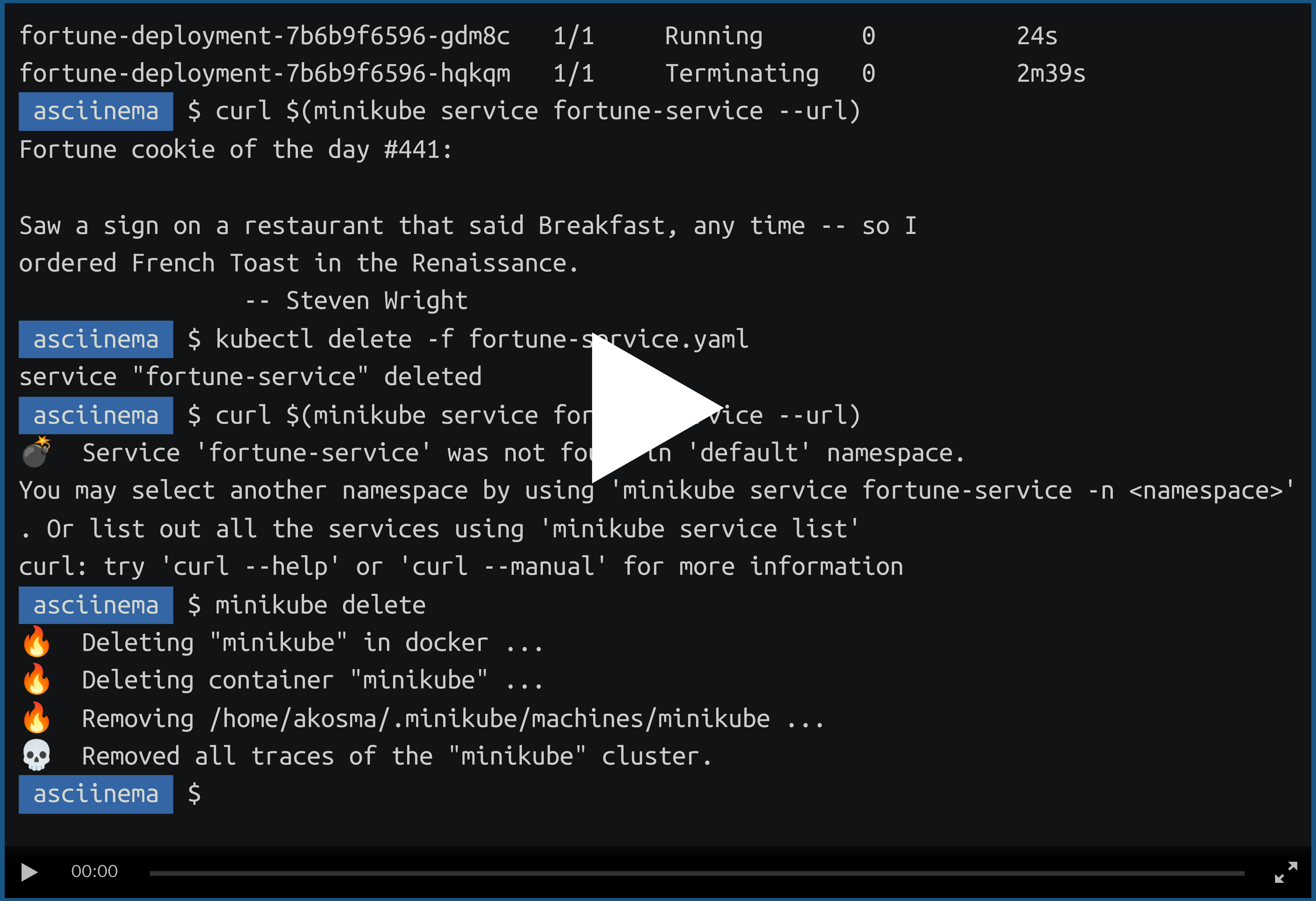
This is what a Kubernetes deployment looks like. This configuration language is called "YAML" and specifies a hierarchy of key-value pairs.

Kubernetes Service

```yaml
apiVersion: v1
kind: Service
metadata:
  name: fortune-service
  labels:
    app: fortune-app
spec:
  ports:
    - port: 3000
      targetPort: fortune-port
  selector:
    app: fortune-app
  type: LoadBalancer
```

Speaker notes

This is a Kubernetes service, exposing the previously shown deployment.



```
fortune-deployment-7b6b9f6596-gdm8c   1/1     Running       0          24s
fortune-deployment-7b6b9f6596-hqkqm   1/1     Terminating   0          2m39s
asciinema  $ curl $(minikube service fortune-service --url)
Fortune cookie of the day #441:

Saw a sign on a restaurant that said Breakfast, any time -- so I
ordered French Toast in the Renaissance.
                -- Steven Wright
asciinema  $ kubectl delete -f fortune-service.yaml
service "fortune-service" deleted
asciinema  $ curl $(minikube service for          vice --url)
💣   Service 'fortune-service' was not fou    in 'default' namespace.
You may select another namespace by using 'minikube service fortune-service -n <namespace>'
. Or list out all the services using 'minikube service list'
curl: try 'curl --help' or 'curl --manual' for more information
asciinema  $ minikube delete
🔥   Deleting "minikube" in docker ...
🔥   Deleting container "minikube" ...
🔥   Removing /home/akosma/.minikube/machines/minikube ...
💀   Removed all traces of the "minikube" cluster.
asciinema  $
```

asciinema.org/a/333331

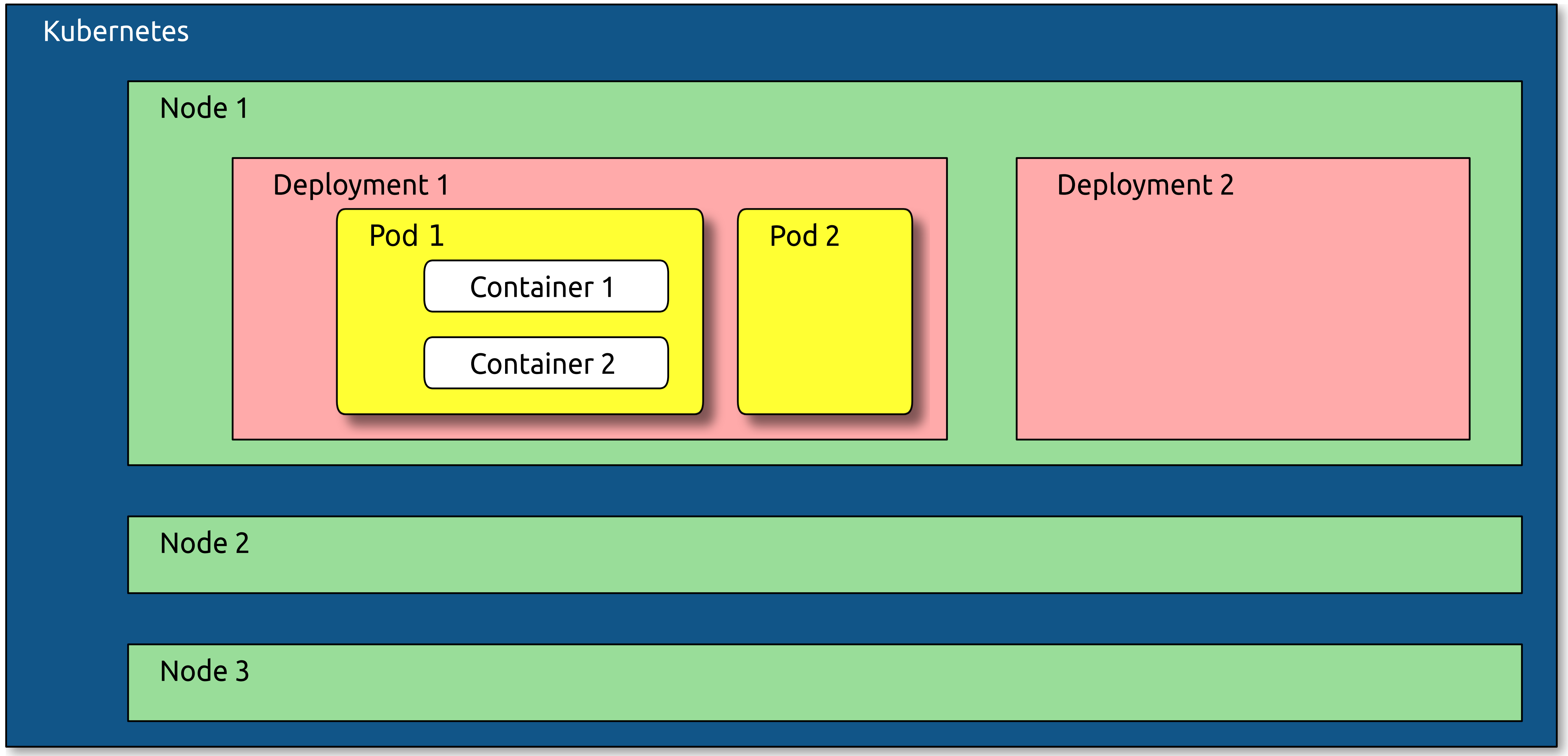
Speaker notes

This video, available at asciinema.org/a/333331, shows the deployment of an application to a Kubernetes cluster. The application shown is called K9s, it's a TUI (Text User Interface) tool that is very popular to manage Kubernetes clusters.

- Kubernetes is a container orchestrator that manages clusters
- Clusters consist of Nodes
- Kubernetes runs Deployments in Nodes
- Deployments usually consist of Pods, Services and Storage
- Services expose network ports to the outside world
- Pods consist of Containers
- Containers are built from Images
- Images are built with a Dockerfile

Speaker notes

So, a short summary about Kubernetes here.



Speaker notes

This is a diagram putting all of the concepts we’ve seen in context.

Kubernetes Distributions

- Rancher RKE
- Red Hat OpenShift
  - APPUiO Cloud ("OpenShift Namespace as a Service")
- Amazon Web Services: Elastic Kubernetes Service (EKS)
- Microsoft Azure: Azure Kubernetes Service (AKS)
- Google Cloud: Google Kubernetes Engine (GKE)

Speaker notes

There are many flavors of Kubernetes, just like there are various flavors of Linux.

Amazon, Azure, and Google Cloud all offer a "managed Kubernetes" solution, which is a great way to have a Kubernetes cluster in a short amount of time.

Kubernetes in your Laptop

- Docker Desktop
- Rancher Desktop
- Minikube
- kind
- SUSE K3s
- Canonical Microk8s
- Red Hat OpenShift CodeReady Containers (CRC)

Speaker notes

If you would like to try Kubernetes, no need to pay for it at any of the big hyperscalers; you can use any of these services to run a small Kubernetes cluster within your laptop.

Red Hat Openshift

OPENSHIFT

Speaker notes

Among all of the Kubernetes distributions above, one of the most popular among big enterprises is Red Hat OpenShift. It offers lots of very interesting features, and this is the reason why we have chosen it to be the basis of APPUiO Cloud.

Openshift Features

- User management
- Graphical user interface
- One-click app deployment
- CI/CD
- Observability
- Integrated container registry
- Enhanced security settings
- `kubectl` ⇒ `oc`

Speaker notes

Let's take a look at some of the things that OpenShift adds to standard Kubernetes off-the-box.

It even provides its own tool, called `oc` instead of `kubectl`. We will use this tool during the workshop this afternoon.

APPUiO
SWISS CONTAINER PLATFORM

APPUiO
CLOUD

Speaker notes

And finally! We reach the gist of this presentation. Now we're going to talk about APPUiO Cloud, and the various challenges we had to face to put the system into production.

VSHN + PUZZLE ITC changing IT for the better = APPUiO SWISS CONTAINER PLATFORM

Speaker notes

APPUiO is a brand and a series of products offered by VSHN together with our partner Puzzle ITC, an IT service provider from Bern. This collaboration started in 2016.

And what does the word "APPUiO" mean? It’s a word in Esperanto meaning "Support".

APPUiO "Flavors"

- ~~Public~~
  - OpenShift 3, pay-per-project, discontinued
- Cloud
  - OpenShift 4, pay-per-use
- Managed
- Self-Managed

Speaker notes

There are several types of APPUiO, but they are all based on OpenShift. APPUiO Cloud is replacing the previous "APPUiO Public", which was based on OpenShift 3.

Why did we change the name? Simply because the business model is different; even though they are both based on shared infrastructure, APPUiO Public is billed per-namespace, while APPUiO Cloud is billed per usage; that is, you only pay for what you use.

APPUiO Managed consists of offering OpenShift clusters to their customers, entirely dedicated, so that nobody else can use them. (Of course this is more expensive!) VSHN and Puzzle ITC take care of the management of the cluster (updates, backups, security, etc.)

APPUiO Self-Managed is like APPUiO Managed, but we train the customer to be able to manage the clusters by themselves, which can be interesting for bigger customers with a dedicated IT team.

APPUiO Cloud Features

- Instant On
- Pay-per-use
- User Management
- Backup
- Pre-Installed and Configured Operators
- Community Support
- Support packages available at extra cost

Speaker notes

APPUiO Cloud has various interesting features that set it apart in the market.

Target Audience

- Cloud Native App Development
- MVP
- DevOps & CI/CD Pipelines
- Machine Learning
- Production App Hosting
- Mobile App Backends
- Education!

Speaker notes

Who have we built APPUiO Cloud for? These are typical customers who need quick access to a managed OpenShift 4 cluster, without the hassle of installation and management.

Shared Platform Restrictions

- Maintenance Policies
- Status information: status.appuio.cloud
- No guarantees of resource availability
- SLA: Best-effort.
- Fair-Use Policy
- Privileged containers can’t run on APPUiO Cloud
- Log retention: 72 hours max, then deleted

Speaker notes

APPUiO Cloud is a shared platform; the same way you would buy "shared PHP + MySQL hosting" for your hobby website in Hostpoint or Infomaniak, you can get a "shared OpenShift cluster" for your enterprise in APPUiO Cloud. This means that there are a few limitations and restrictions in place.

Speaker notes

Where is APPUiO Cloud located? We have two zones at the moment, and more are coming soon.

Zones

- `cloudscale-lpg-2` in Kanton Aargau
- `exoscale-ch-gva-2-0` in Canton de Genève
- … more soon!

Challenges

- Security
- Storage
- Management
- Policies
- Documentation

Speaker notes

The development of APPUiO Cloud needed to solve various problems. Let’s see some of them in detail.

Security & Policies

KEYCLOAK

Speaker notes

All of the APPUiO Cloud zones are managed through a single Identity Provider (IdP); we use Keycloak www.keycloak.org for that.

Keycloak is an open source software product under the stewardship of Red Hat, written in Java, to allow single sign-on with Identity and Access Management aimed at modern applications and services.

Whenever a new user requests an APPUiO Cloud account, they get an account at our Keycloak, and that allows them to access any of the APPUiO Cloud zones without distinction.

The other logo belongs to Kyverno kyverno.io, a policy engine for Kubernetes which we use to enforce policies and quotas in APPUiO Cloud.

Storage

ceph

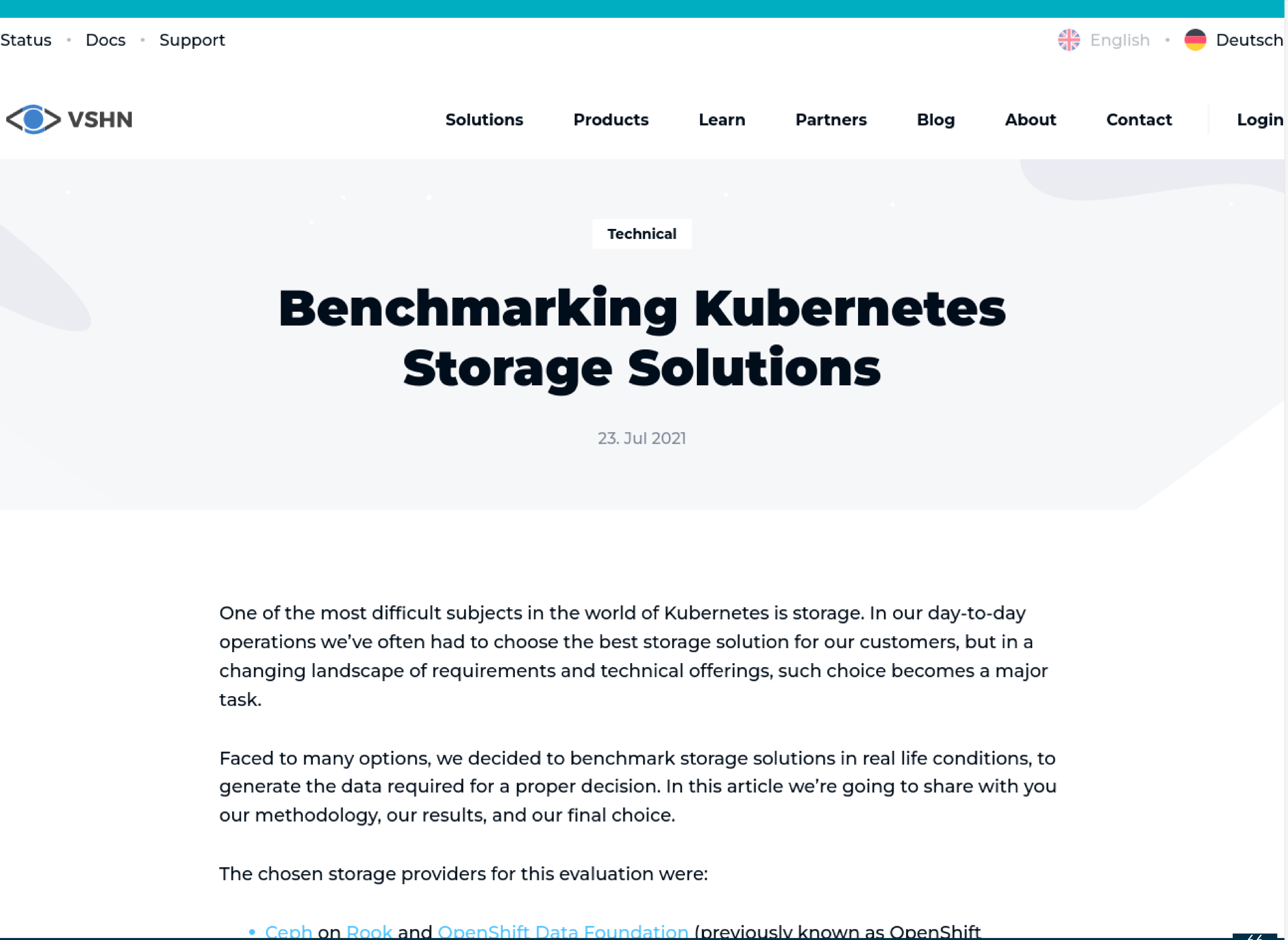
ROOK

Speaker notes

Our engineers benchmarked various storage solutions before deciding on the one to be used by APPUiO Cloud:

- Ceph
- Longhorn
- Gluster

Ceph on Rook was finally chosen.



Status • Docs • Support

English • Deutsch

VSHN

Solutions Products Learn Partners Blog About Contact Login

Technical

Benchmarking Kubernetes Storage Solutions

23. Jul 2021

One of the most difficult subjects in the world of Kubernetes is storage. In our day-to-day operations we've often had to choose the best storage solution for our customers, but in a changing landscape of requirements and technical offerings, such choice becomes a major task.

Faced to many options, we decided to benchmark storage solutions in real life conditions, to generate the data required for a proper decision. In this article we're going to share with you our methodology, our results, and our final choice.

The chosen storage providers for this evaluation were:

- Ceph on Rook and OpenShift Data Foundation (previously known as OpenShift

Speaker notes

You can find the full storage benchmark done by our VSHNeer Simon Gerber in our blog: www.vshn.ch/en/blog/benchmarking-kubernetes-storage-solutions/

Documentation

- Product description at products.docs.vshn.ch
- System Engineering documentation at kb.vshn.ch
- Documentation for end users at docs.appuio.cloud

Speaker notes

VSHN is a document-driven company. We document everything we do, at every step, and this attitude has helped us a lot during the pandemic; new hires could find all the information they need to start working, and whenever somebody goes on holidays, we don’t need to call them to ask how to do stuff on their behalf.

So as expected, we have created lots of documentation for APPUiO Cloud; in fact, we have documented in written every single step of the creation of the platform. And even better, all of this documentation is freely available online:

These three websites are completely open, free, and ready to read; actually, this very presentation is in a way a summary of the information contained in those three websites!

Learn More!

Speaker notes

This was a quick overview of APPUiO Cloud before we get to use it in a real-world scenario. There’s a lot of information out there for you to learn more about it!

For Engineers

kb.vshn.ch/appuio-cloud/

For Users

docs.appuio.cloud/user/

Product Description

products.docs.vshn.ch/products/appuio/cloud/

Status

status.appuio.cloud

Portal

portal.appuio.cloud

Speaker notes

Here are some links where you can find all the information you need about APPUiO Cloud.

Thanks!

APPUiO CLOUD

Adrian Kosmaczewski, Developer Relations – adrian@vshn.ch

VSHN AG – Neugasse 10 – CH-8005 Zürich – +41 44 545 53 00 – vshn.ch – info@vshn.ch

Speaker notes

Thanks a lot for your attention!