Scalabilità e Resilienza in Kubernetes:

come le app Cloud-Native si auto-adattano al carico e rinascono dalle ceneri

...

Linux Day 2024
Francavilla Fontana

ing. Alessandro Argentieri

Mi presento

Ingegnere dell'Informazione presso il PoliBA

Lavoro da 10 anni in ambito IT

Appassionato in Architettura del Software e sistemi complessi

Tech-Lead @ Civo (civo.com), un Cloud Provider UK

linkedin.com/in/alessandroargentieri

github.com/alessandroargentieri

alessandro@civo.com

Indice

- piccola intro su Kubernetes (per chi è digiuno)
- i meccanismi di auto-adattamento di Kubernetes
- demo pratica

Kubernetes (k8s) spiegato a mia nonna

- è un progetto Open Source nato da Google
- donato alla CNCF (costola della Linux Foundation)
- è considerato il “Sistema Operativo” per il Cloud (CNCF annual survey 2022)
- **è un software distribuito che permette la gestione ed il rilascio centralizzato di app containerizzate su più macchine connesse**
- implementa una serie di pattern tipici per le app Cloud Native (load balancing, service-discovery, api-gateway, circuit-breaker, health-check, ...)
- curiosità: OpenAI usa un K8s di 7500 nodi

foto: **Kelsey Hightower**
pioniere dello sviluppo di Kubernetes in Google

App containerizzate ?!?

App avviate come processi isolati (linux) chiamati **container** .

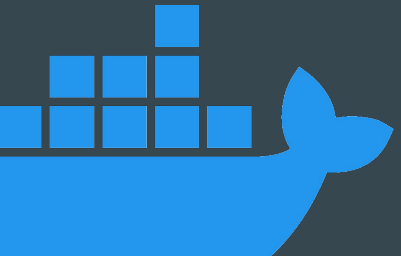
Simili a macchine virtuali ma più leggeri

I container condividono il sistema operativo linux (le chiamate al kernel)

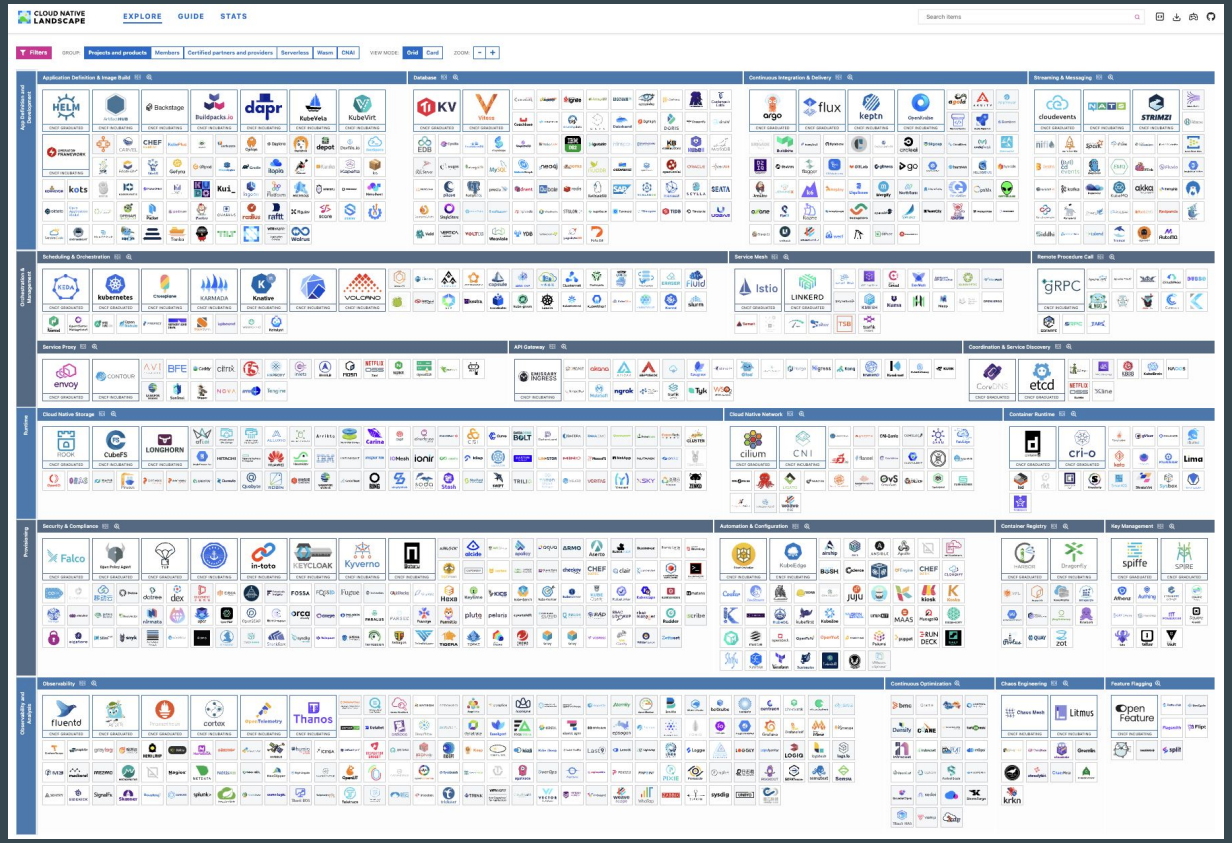
I processi nei container credono di avere una macchina propria (CPU/RAM/disco)

Come si avvia una macchina virtuale da una ISO, un container parte da un'**immagine**

Le immagini sono salvate in **container registry** (es. Docker Hub)



CNCF: Cloud Native Computing Foundation

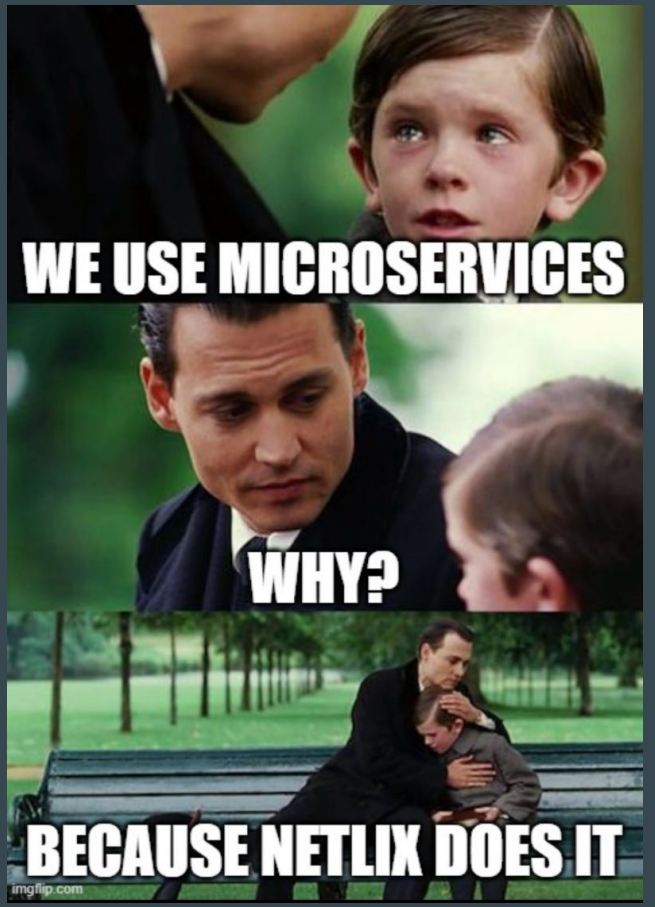


landscape.cncf.io

CLOUD NATIVE
COMPUTING FOUNDATION

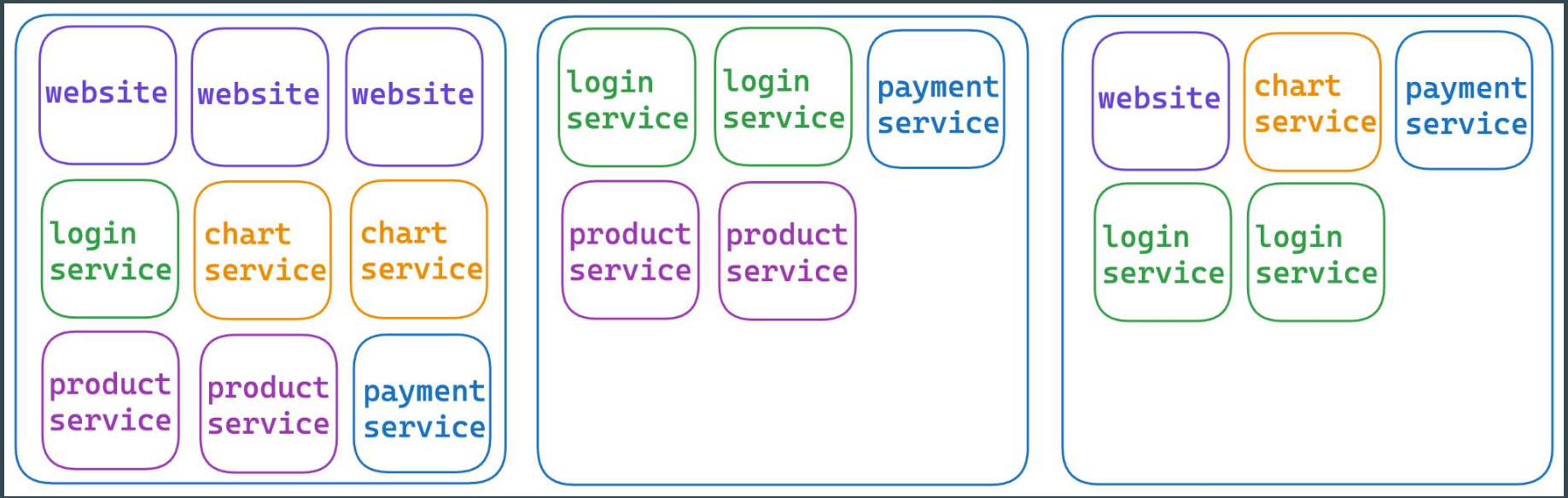
Le basi di Kubernetes

Immaginiamo un e-commerce composto da 5 microservizi che interagiscono tra loro



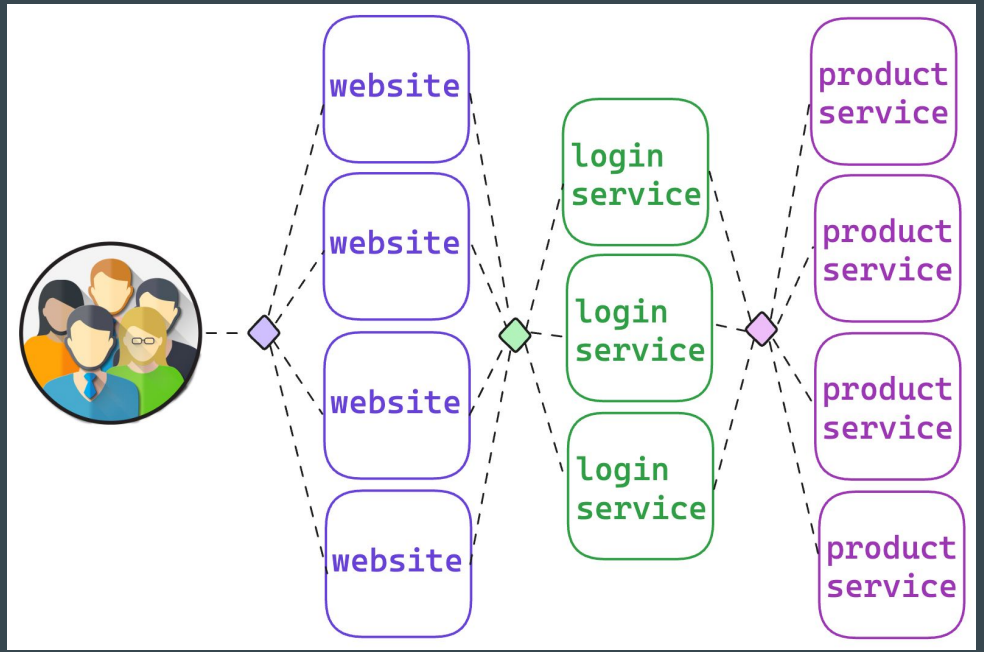
Le basi di Kubernetes

Il “cervello” del “cluster” Kubernetes distribuisce le app containerizzate nei vari nodi (macchine) disponibili con le repliche necessarie.



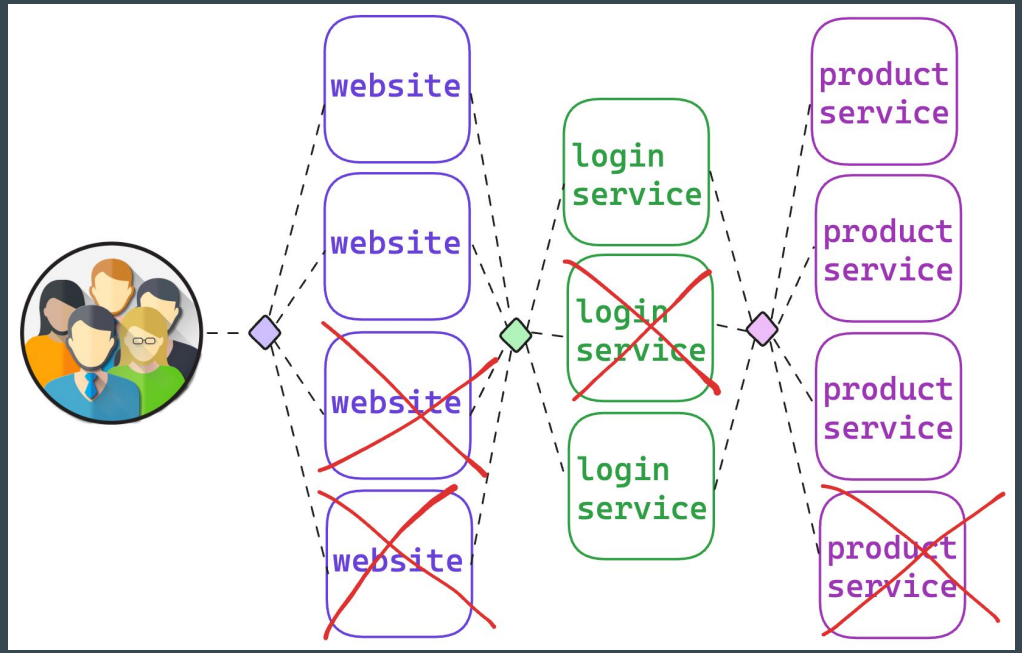
Le basi di Kubernetes

Gli utenti vedono l'app come un'entità unica, non curandosi di quanti servizi e quante repliche per servizio la costituiscono, né di quale macchina ospita quale replica



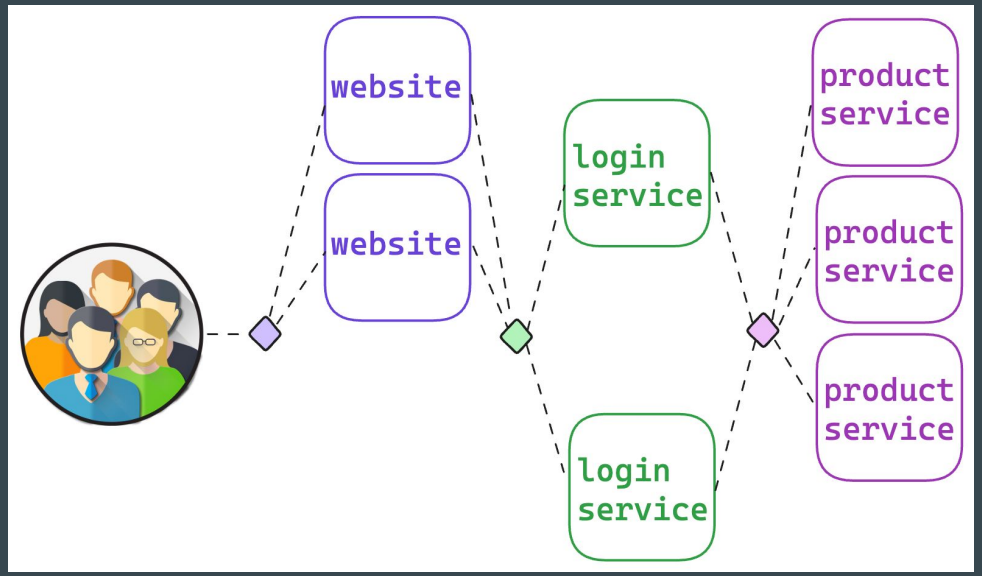
Le basi di Kubernetes

Se alcune repliche diventano irraggiungibili (crash del processo, problemi di rete, sovraccarico)



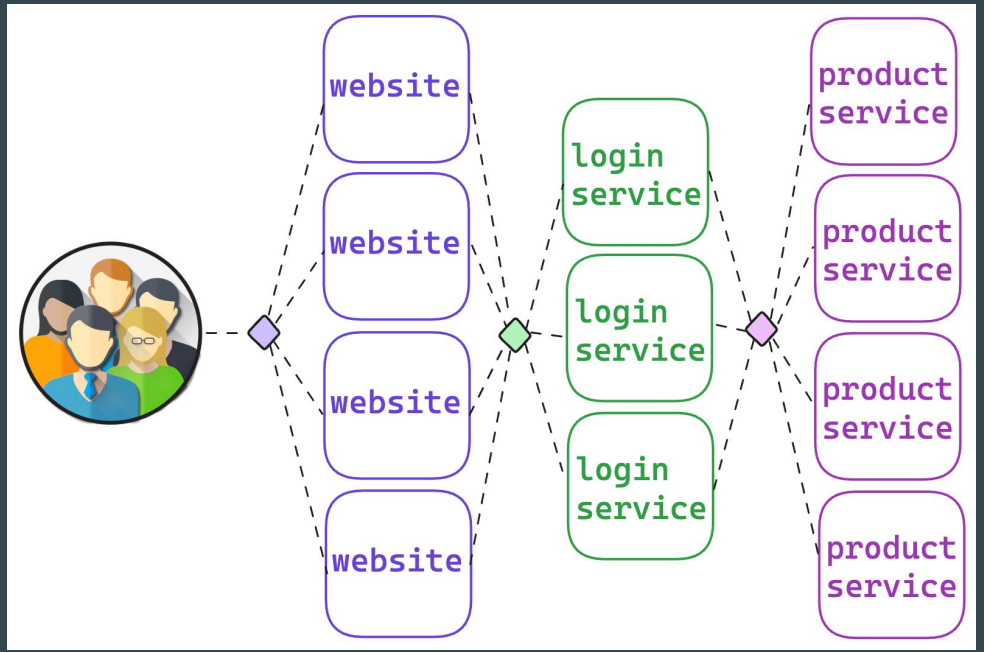
Le basi di Kubernetes

Kubernetes stacca i ponti per evitare che queste repliche siano utilizzate (producendo un errore ed interrompendo l'esperienza utente)



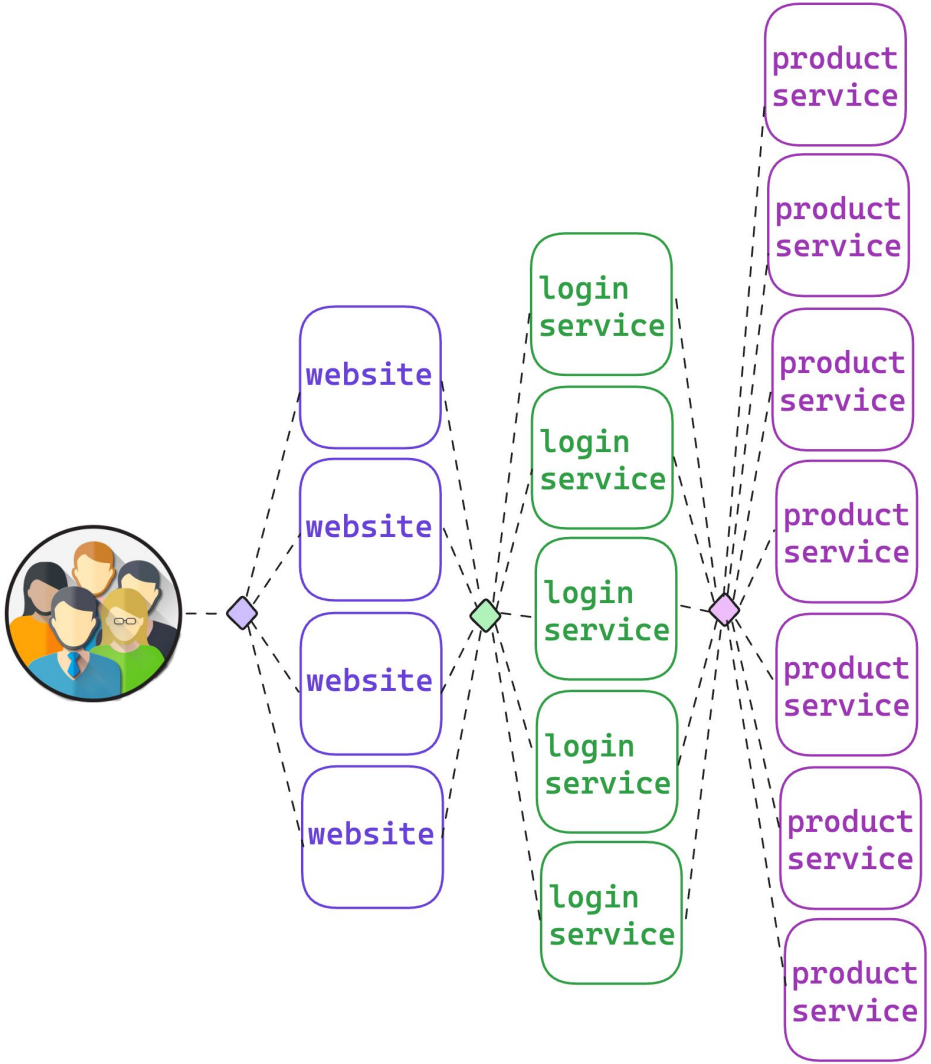
Le basi di Kubernetes

... e poco dopo la situazione viene ristabilita in automatico affinché lo stato reale raggiunga quello desiderato (programmazione dichiarativa - non imperativa)

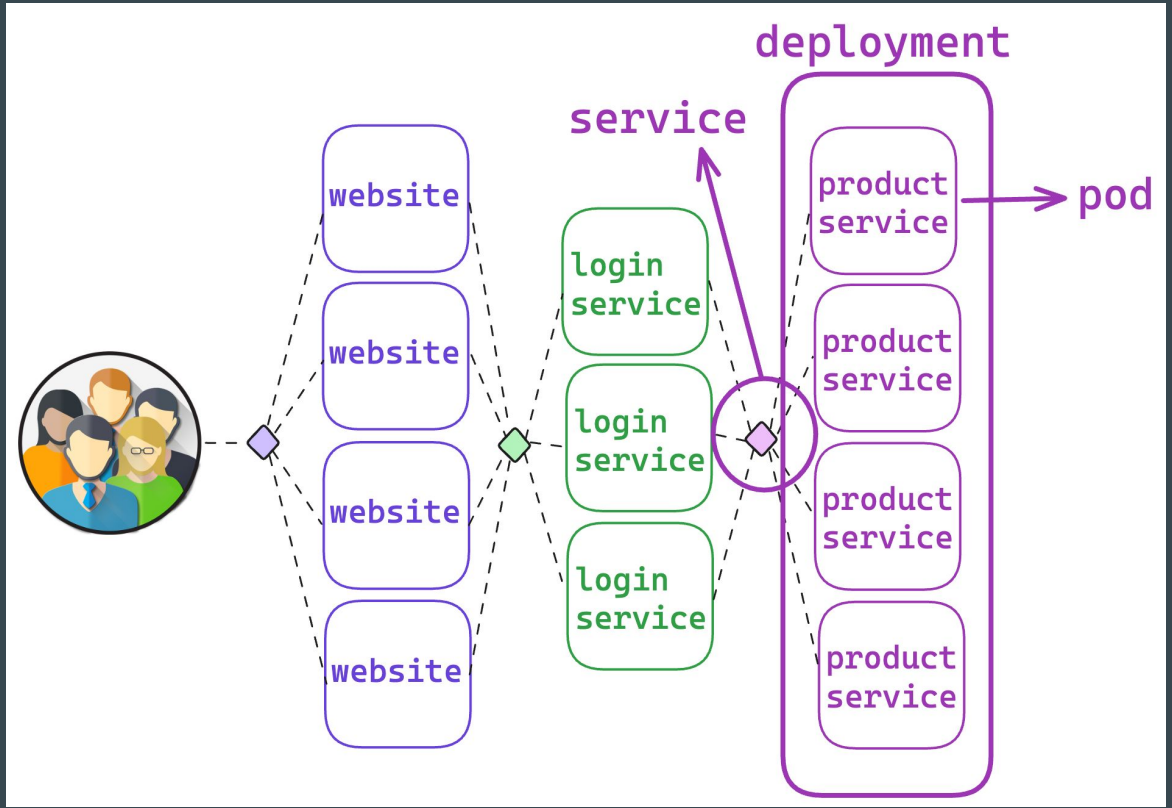


Le basi di Kubernetes

Se il traffico di rete aumenta e sono necessarie più repliche, il sistema reagisce e ne crea di nuove, adattandosi alle nuove necessità

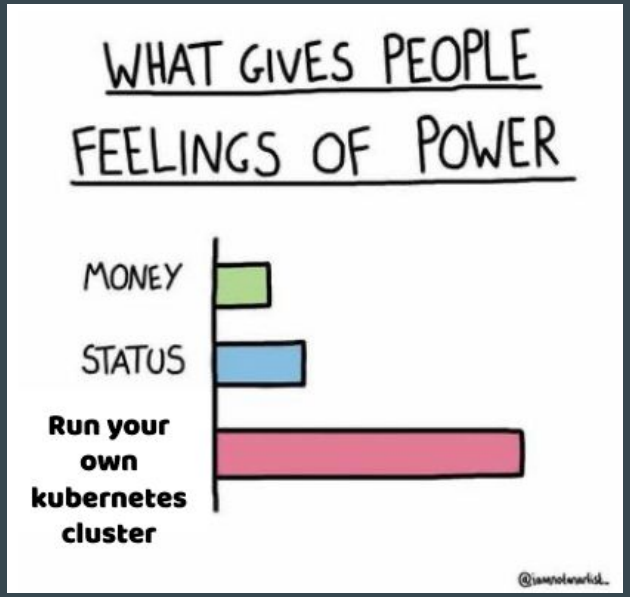


Le basi di Kubernetes

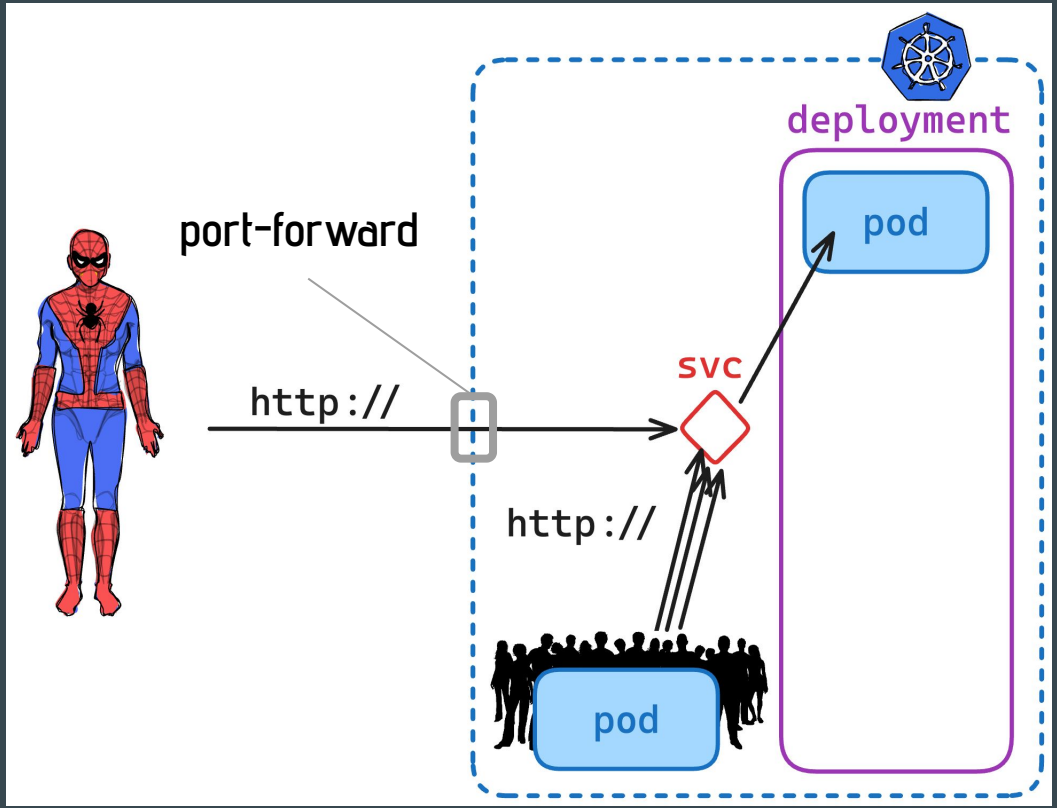


Piccola demo

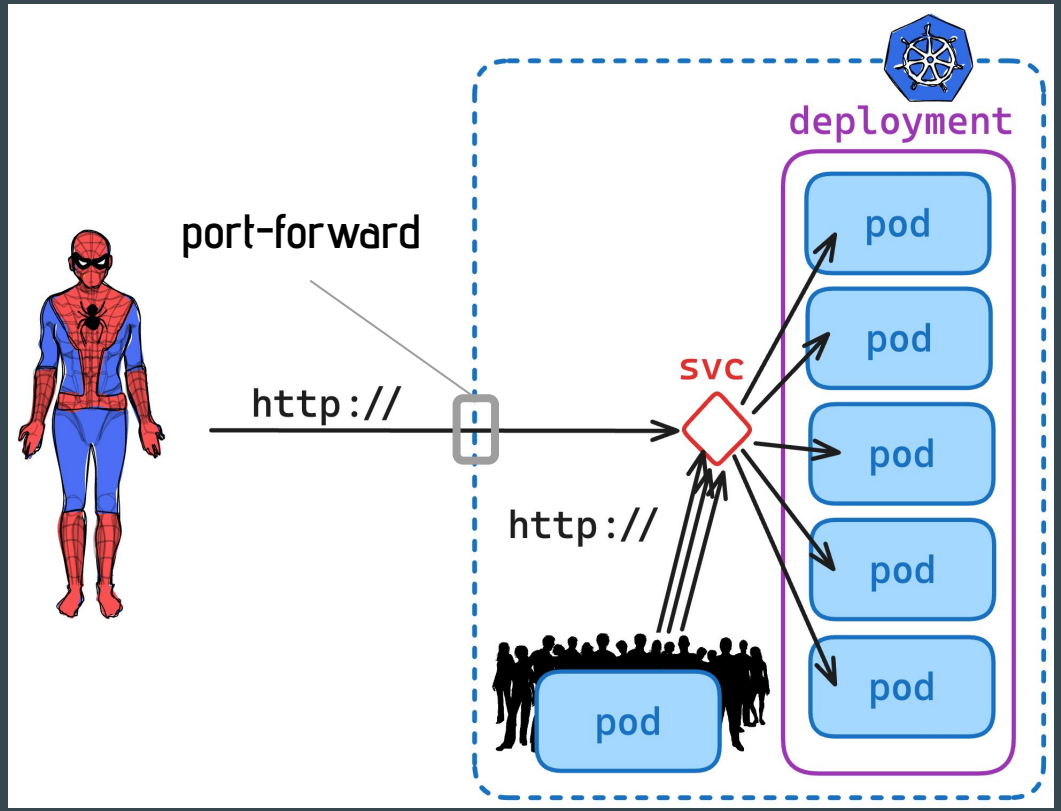
https://github.com/f4rezer0/linux-day-2024-k8s-autoscaling



Piccola demo



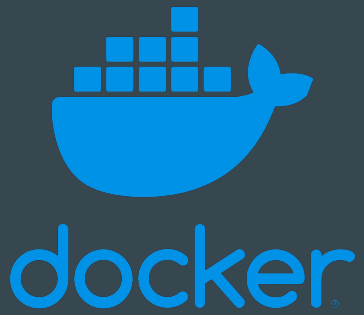
Piccola demo



Prerequisiti

Software installati sulla tua macchina:

- curl
- git
- docker
- kubectl



Prepariamo un cluster locale con k3d (k3s in docker)

install k3d

```
~  $ curl -s https://raw.githubusercontent.com/k3d-io/k3d/main/install.sh | bash
```

install k3d

```
~  $ k3d cluster create linuxday-k3d \
      --agents 2 \
      --port '80:80@loadbalancer' \
      --port '443:443@loadbalancer'
```

Installiamo i componenti nel cluster

install k3d

```
~  $ git clone https://github.com/f4rezer0/linux-day-2024-k8s-autoscaling.git
~  $ cd linux-day-2024-k8s-autoscaling
```

install k3d

```
~/linux-day-2024-k8s-autoscaling (main)  $ kubectl apply -f metrics-server.yaml
~/linux-day-2024-k8s-autoscaling (main)  $ kubectl apply -f php-apache.yaml
```

```
deployment.apps/php-apache-deployment created
horizontalpodautoscaler.autoscaling/php-apache-hpa created
service/php-apache-svc created
```

Installiamo i componenti nel cluster

list of the services

```
$ kubectl get services
NAME             TYPE           CLUSTER-IP      EXTERNAL-IP   PORT(S)        AGE
kubernetes       ClusterIP      10.43.0.1       <none>        443/TCP        4h42m
php-apache-svc   LoadBalancer   10.43.218.244   <pending>     80:31319/TCP   4h38m
```

list of the deployments

```
$ kubectl get deployments
NAME                     READY   UP-TO-DATE   AVAILABLE   AGE
php-apache-deployment    1/1     1            1           4h35m
```

list of the pods

```
$ kubectl get pods
NAME                                     READY   STATUS    RESTARTS   AGE
php-apache-deployment-5bdbb8dbf8-wrtxl   1/1     Running   0          4h10m
```

Test del deployment

port-forward the service on localhost:8888

```
$ kubectl port-forward svc/php-apache-svc 8888:80
Forwarding from 127.0.0.1:8888 -> 80
Forwarding from [::1]:8888 -> 80
```

port-forwarding

```
~  $ curl http://localhost:8888
OK!
```

Simuliamo il carico

create a load-generator pod

```
$ kubectl run -i --tty load-generator --rm --image=busybox:1.28 --restart=Never -- /bin/sh -c
"while sleep 0.01; do wget -q -O- http://php-apache-svc; done"
```

list the pods

```
$ kubectl get pods
NAME                                READY   STATUS    RESTARTS   AGE
php-apache-deployment-5bdbb8dbf8-wrtxl   1/1     Running   0          4h46m
load-generator                      1/1     Running   0          3m26s
php-apache-deployment-5bdbb8dbf8-s2nj4   1/1     Running   0          3m6s
php-apache-deployment-5bdbb8dbf8-n5n4v   1/1     Running   0          3m6s
php-apache-deployment-5bdbb8dbf8-vpvzc   1/1     Running   0          3m6s
php-apache-deployment-5bdbb8dbf8-nj4nc   1/1     Running   0          2m21s
php-apache-deployment-5bdbb8dbf8-fpcgt   1/1     Running   0          2m21s
```

Approfondimento: ispezioniamo k3d con Docker (1)

inspect the kubernetes node

```
$ kubectl get nodes
NAME                       STATUS   ROLES                  VERSION
k3d-linuxday-k3d-server-0   Ready    control-plane,master   v1.27.4+k3s1
k3d-linuxday-k3d-agent-1    Ready    <none>                 v1.27.4+k3s1
k3d-linuxday-k3d-agent-0    Ready    <none>                 v1.27.4+k3s1
```

inspect the pods

```
$ kubectl get pods
NAME                                     READY   STATUS
php-apache-deployment-5bdbb8dbf8-28ld9   1/1     Running
```

Approfondimento: ispezioniamo k3d con Docker (1)

inspect the kubernetes node

```
$ docker ps | grep k3s1
6d921f65e059   rancher/k3s:v1.27.4-k3s1   k3d-linuxday-k3d-agent-1
aead2f2e57a1   rancher/k3s:v1.27.4-k3s1   k3d-linuxday-k3d-agent-0
1e7c713453b6   rancher/k3s:v1.27.4-k3s1   k3d-linuxday-k3d-server-0
```

inspect the pods

```
$ docker exec k3d-lday1-k3d-agent-0 crictl ps | grep php
CONTAINER           STATE               NAME            POD
5f70e8fdab916       Running             php-apache      php-apache-deployment-5bdbb8dbf8-28ld9
```

Grazie dell'attenzione

farezero.org

linkedin.com/in/alessandroargentieri

github.com/alessandroargentieri

alessandro@civo.com