



Cloud Computing (**LAB**)

HANDS ON  **kubernetes**

Giuseppe Bisicchia
giuseppe.bisicchia@phd.unipi.it

Dipartimento di Informatica, Università di Pisa

Cosa faremo?

- Sperimenteremo con *Kubernetes* scoprendo come **creare, lanciare e gestire** un Pod.
 - Impareremo a gestire **multiple repliche** di un'applicazione.
 - Lavoreremo con un **cluster** con uno o più nodi.
 - Extra: Sperimenteremo con i Servizi in Kubernetes.
-
- **Suggerimento:** durante il laboratorio usate «*minikube dashboard*» per capire visualmente cosa sta accadendo.

Minikube?

- Minikube è uno strumento che consente di eseguire Kubernetes **localmente**.
- Esegue un **cluster** Kubernetes locale **all-in-one** (cioé su un solo nodo) o **multi-nodo** sul vostro computer, in modo da poter provare Kubernetes o per lo sviluppo quotidiano.
- Minikube ha un ambiente docker interno, le immagini e i container presenti nel vostro sistema sono diversi da quelli al suo interno.

Minkube Useful commands

- `minikube start --nodes <number_of_nodes>`
it starts a minikube cluster with n_o_n nodes
- `minikube stop`
it stops the minikube cluster
- `minikube delete`
it delete the minkube cluster
- `minikube image load <image> [--daemon]`
it load a docker image from your local docker to minikube's docker
- `minikube service list`
it shows the list of services running on the minikube cluster
- `minikube kubectl -- <command>`
it allows to run a Kubernets (kubectl) command



More on: <https://minikube.sigs.k8s.io/docs/commands/>

Cluster

- Per iniziare abbiamo bisogno di un **cluster** su cui lavorare.
- Per i nostri scopi andrà benissimo un **cluster locale** formato da un **solo nodo** che creeremo con «*minikube start*».
- Per installare minikube seguire questa guida fino al **punto 3**: <https://minikube.sigs.k8s.io/docs/start/>

Cluster

- Possiamo verificare la presenza del nodo tramite la «minikube dashboard»

The screenshot shows the Kubernetes Dashboard interface. The top navigation bar includes the Kubernetes logo, a dropdown for the cluster ('default'), a search bar, and a '+' button. The main menu on the left lists 'Cluster > Nodes' and other resources like Ingresses, Services, Config and Storage, Config Maps, Persistent Volume Claims, Secrets, and Storage Classes. The 'Nodes' section displays a table with columns: Name, Labels, Ready, CPU requests (cores), CPU limits (cores), CPU capacity (cores), Memory requests (bytes), Memory limits (bytes), Memory capacity (bytes), Pods, and Created. One node, 'minikube', is listed with the following details:

Name	Labels	Ready	CPU requests (cores)	CPU limits (cores)	CPU capacity (cores)	Memory requests (bytes)	Memory limits (bytes)	Memory capacity (bytes)	Pods	Created
minikube	beta.kubernetes.io/arch: amd64 beta.kubernetes.io/os: linux kubernetes.io/arch: amd64	True	750.00m (18.75%)	0.00m (0.00%)	4.00	170.00Mi (2.16%)	170.00Mi (2.16%)	7.70Gi	11 (10.00%)	an hour ago

Esercizio: Primo Pod

- Create un Manifesto Kubernetes per lanciare un Pod contenente un solo container con immagine «*mhausenblas/simpleservice:0.5.0*» e che non sia in ascolto su alcuna porta.

Group of the
object &
what's it's
version?

What are the
things that
uniquely
identify your
Kubernetes
object?

THE MANIFEST FILE

API VERSION

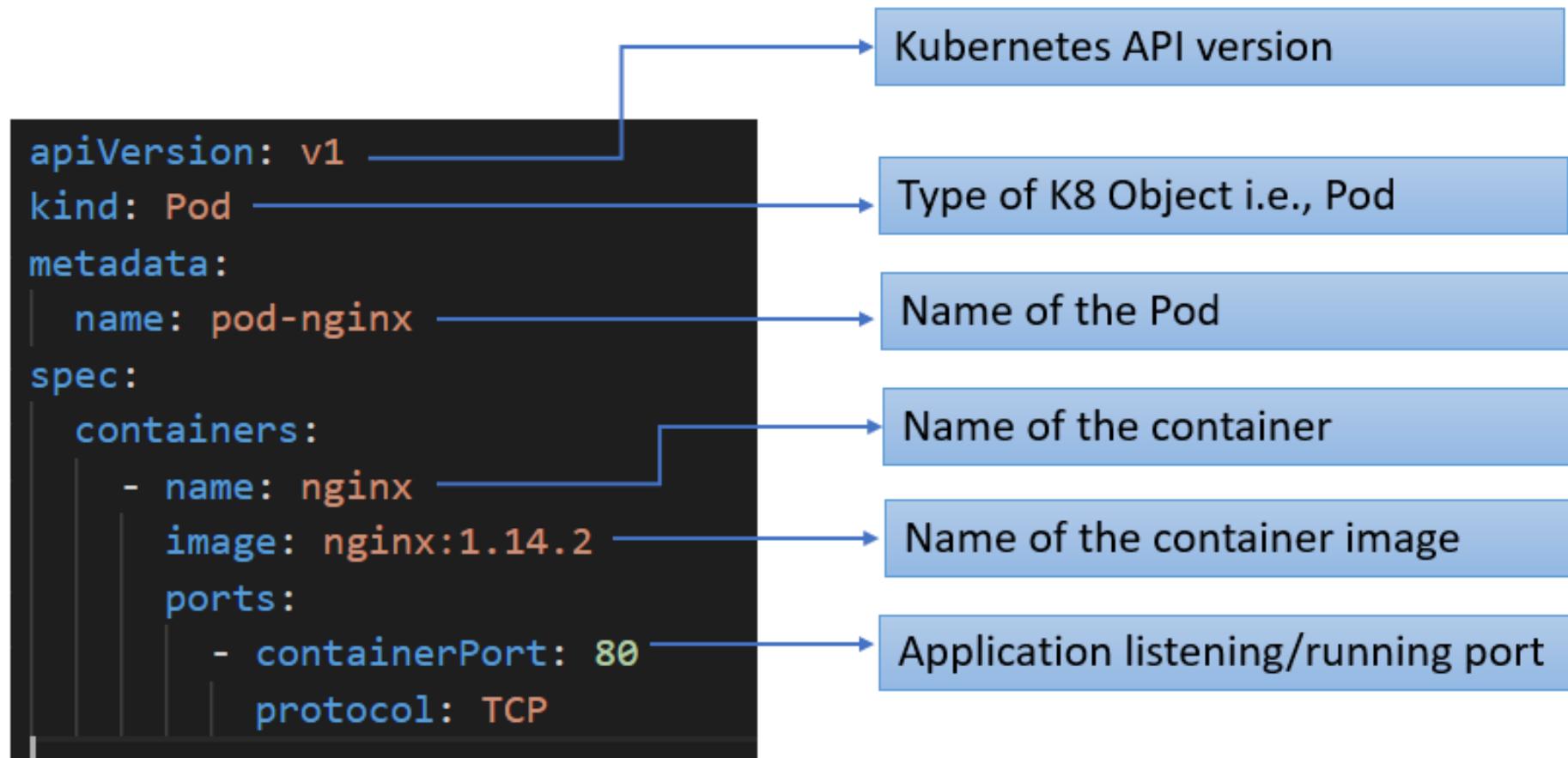
KIND

METADATA

SPEC

What are you
creating? Pod?
Service? or
something else

What should be
the state of
your object?
Which image to
use? Any other



Pod manifest: pod-nginx.yaml

Soluzione: Primo Pod

```
apiVersion: v1
kind: Pod
metadata:
  name: simple-pod
spec:
  containers:
    - name: simple-pod
      image: mhausenblas/simpleservice:0.5.0
      ports:
        - containerPort: 9876
```

Esercizio: Interagiamo con il Pod

- Lanciate il Pod
- Trovate come visualizzare la lista dei Pod in esecuzione tramite linea di comando.
- Visualizzate i log del Pod creato precedentemente.
- Cercate come è possibile ottenere l'IP interno del Pod tramite linea di comando

K8s Useful commands

- `kubectl get <resource>`
it shows the status of <resource> (e.g. pods, nodes, services).
- `kubectl apply -f <manifest.yaml>`
it creates or update a resource from a .yaml file.
- `kubectl drain <node_name> [--ignore-daemonsets]`
it remove everything from the node <node_name>.
- `kubectl uncordon <node_name>`
it marks <node_name> as schedulable.
- `kubectl delete <resource>`
it delete the <resource>.



More on: <https://kubernetes.io/docs/reference/kubectl/>

Soluzione: Interagiamo con il Pod

- Per lanciare il Pod eseguite: «*kubectl apply -f NOME_FILE*»
- Eseguite «*kubectl get pod*» per avere la lista dei Pod in esecuzione.
- Con «*kubectl logs NOME_POD*» possiamo ottenere informazioni sui log del Pod.
- «*kubectl describe pod NOME_POD*» è un comando molto potente che ci consente di avere numerose informazioni sul Pod (e.g., il suo IP interno).

Interagiamo con il Pod

- Eseguite «*minikube ssh*» per entrare in una shell con cui interagire con i container. Provate il comando «*curl IP_POD:9876/info*», che accade?
- Con «*kubectl get all*» possiamo visualizzare tutte le risorse create da Kubernetes, è presente altro oltre al nostro Pod?
- Eliminiamo il Pod con il comando «*kubectl delete pod NOME_POD*»

Soluzione: Interagiamo con il Pod

- Attraverso il comando «*curl IP_POD:9876/info*» otteniamo in output un oggetto del tipo `{"host": "10.244.0.5:9876", "version": "0.5.0", "from": "10.244.0.1"}` con informazioni sull'IP dell'host e del richiedente
- Eseguendo «*kubectl get all*» notiamo che oltre al nostro Pod è presente un servizio «service/kubernetes» di tipo «ClusterIP» che consente ad altri componenti del cluster, come pod o servizi, di comunicare con il server API di Kubernetes.

Lanciamo più repliche del nostro Pod

- Per quanto sia possibile gestire direttamente i Pod, questa **non è mai l'opzione consigliata**.
- Infatti i Pod possono essere **terminati in qualsiasi momento** da Kubernetes ed esistono varie **astrazioni** per rendere più semplice ed efficace la gestione delle nostre applicazioni.

Esercizio: Più repliche

- Lanciate **due repliche** del precedente Pod.
- **Suggerimento:** per farlo create un nuovo Manifesto Kubernetes con «kind: Deployment».

```
kind: Deployment
metadata:
  name: ml-model-serving
  labels:
    app: ml-model
spec:
  replicas: 10
  selector:
    → matchLabels:
      app: ml-model
  template:
    metadata:
      → labels:
        app: ml-model
    spec:
      containers:
        - name: ml-rest-server
          image: ml-serving:1.0
          ports:
            - containerPort: 80
```

How many Pods should be running?

How do we find Pods that belong to this Deployment?

What should a Pod look like?

Add a label to the Pods so our Deployment can find the Pods to manage.

What containers should be running in the Pod?

Soluzione: Più repliche

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: simple-deployment
spec:
  replicas: 2
  selector:
    matchLabels:
      app: simple-deployment
  template:
    metadata:
      labels:
        app: simple-deployment
    spec:
      containers:
        - name: simple-pod
          image: mhausenblas/simpleservice:0.4.0
          ports:
            - containerPort: 9876
```

Esercizio: Interagiamo con il Deployment

- Lanciate il deployment.
- Visualizzate il contenuto del deployment tramite linea di comandi.
- Visualizzate i Pod presenti.

Soluzione: Interagiamo con il Deployment

- Per lanciare il deployment basta «*kubectl apply -f NOME_FILE*»
- Con «*kubectl get deployment*» possiamo vedere il contenuto del deployment.
- Possiamo osservare i Pod presenti con «*kubectl get pod*»

Esercizio: Interagiamo con il Deployment

- Usate «*kubectl get all*», Cosa vi aspettate di vedere? E' stato creato altro? Perché?
- Per modificare il numero di repliche basta modificare il manifesto e ri-eseguire apply. Provate prima ad aumentare e poi diminuire il numero di repliche e osservata sulla dashboard cosa accade.
Alternativamente alla dashboard potete usare «*kubectl get pod --watch*» in un nuovo terminale.

Soluzione: Interagiamo con il Deployment

- Usando «*kubectl get all*» possiamo notare come sia stato creato anche un ReplicaSet per il nostro Deployment.
- Lo scopo di un ReplicaSet è mantenere un insieme stabile di repliche di un Pod in esecuzione in qualsiasi momento. Per questo motivo, viene spesso utilizzato per garantire la disponibilità di un numero specifico di Pod identici.
- Un Deployment è gestisce i ReplicaSet e fornisce aggiornamenti dichiarativi ai Pod insieme a molte altre utili funzionalità.

Esercizio: Pod e Deployment

- Lanciate di nuovo il Pod singolo dell'esercizio precedente.
- Aprite la dashboard o eseguite il comando `watch`.
- Una volta che tutti i container sono in esecuzione lanciate il comando «`kubectl delete pod --all`» per eliminare tutti i Pod, che accade? Perché?
- Per eliminare tutto basta lanciare «`kubectl delete deployment,rs,pod --all`»

Soluzione: Pod e Deployment

- Il Deployment tramite il ReplicaSet garantisce che anche in caso di terminazione di una o più repliche altrettante vengano create per rispettare i desiderata dell'utente. Il Pod invece non ci da queste garanzie e una volta terminato deve essere lanciato nuovamente manualmente.

Esercizio: Più nodi

- Stoppatate la versione corrente di minikube e lanciate un'altra con 3 nodi (2 se il vostro PC non ha abbastanza risorse).
- Sperimentate lanciando Pods e Deployments e utilizzando i comandi drain e uncordon per i nodi. Che differenza c'è?

Soluzione: Più nodi

Node Drain:

1. Quando un nodo viene "drained" (drenato), Kubernetes termina tutte le pod in esecuzione su quel nodo e ne pianifica il riavvio altrove nel cluster.
2. Questa operazione è utile quando è necessario mettere offline un nodo per manutenzione o per altre ragioni, garantendo che le applicazioni in esecuzione su di esso vengano trasferite altrove prima di interrompere il nodo.
3. Un nodo può essere drenato manualmente utilizzando il comando `kubectl drain`, specificando il nome del nodo come argomento.

Node Uncordon:

1. Quando un nodo è "uncordoned" (sbloccato), significa che è stato ripristinato alla piena capacità di accettare nuove pod.
2. Un nodo può essere messo in uno stato uncordoned manualmente utilizzando il comando `kubectl uncordon` seguito dal nome del nodo come argomento.
3. Dopo che un nodo è stato uncordoned, Kubernetes può quindi pianificare nuove pod su di esso.

"*node drain*" è utilizzato per spostare i pod da un nodo prima di metterlo offline, "*node uncordon*" è utilizzato per consentire a un nodo di accettare nuovi pod dopo essere stato messo offline o aver completato la manutenzione.

Extra: Servizi

Perché usare i servizi?

- Al momento per accedere alla nostra applicazione dobbiamo **conoscere l'IP** di uno **specifico Pod** ed interagire direttamente con esso, anche se sono presenti più repliche.
- Inoltre, nel caso che un Pod non sia più disponibile dobbiamo usare l'IP di un altro Pod.
- Infine, quell'IP **non è accessibile al di fuori** del cluster.

Creiamo un servizio

- Con un **servizio** possiamo accedere i nostri Pod sia **all'interno** che **all'esterno** di un servizio, inoltre ci permetterà di accedere ad un deployment senza dover specificare un particolare Pod.
- In questo esperimento utilizzeremo il deployment precedente ed un ulteriore Pod. Il Pod aggiuntivo ci servirà per poter accedere ai Pod del deployment.

Creiamo un servizio

- Per iniziare lanciamo il deployment precedente con apply.
- Ora lanciate il Pod presente su moodle e successivamente eseguite il comando «*kubectl exec -it bash -- /bin/bash*» che permetterà di aprire una shell bash all'interno del container.
- All'interno della bash eseguite «*apt update && apt install dnsutils curl*» per installare dei tool che ci serviranno a breve.
- Utilizzate ora curl come visto precedentemente per accedere al deployment all'indirizzo «*IP POD:9876/health*», come si deve fare?

Esercizio: Creiamo un Servizio

- Con i Servizi possiamo **disaccoppiare** l'accesso ad un deployment evitando di dover specificare un Pod specifico ed il relativo IP.
- Create quindi il manifesto di un servizio (i.e., «kind: Service») collegato al nostro deployment.

```
apiVersion: v1
kind: Service
metadata:
  name: ml-model-svc
  labels:
    app: ml-model
spec:
  type: ClusterIP
  selector:
    app: ml-model
  ports:
  - protocol: TCP
    port: 80
```

How do we want to expose our endpoint?

How do we find Pods to direct traffic to?

How will clients talk to our Service?

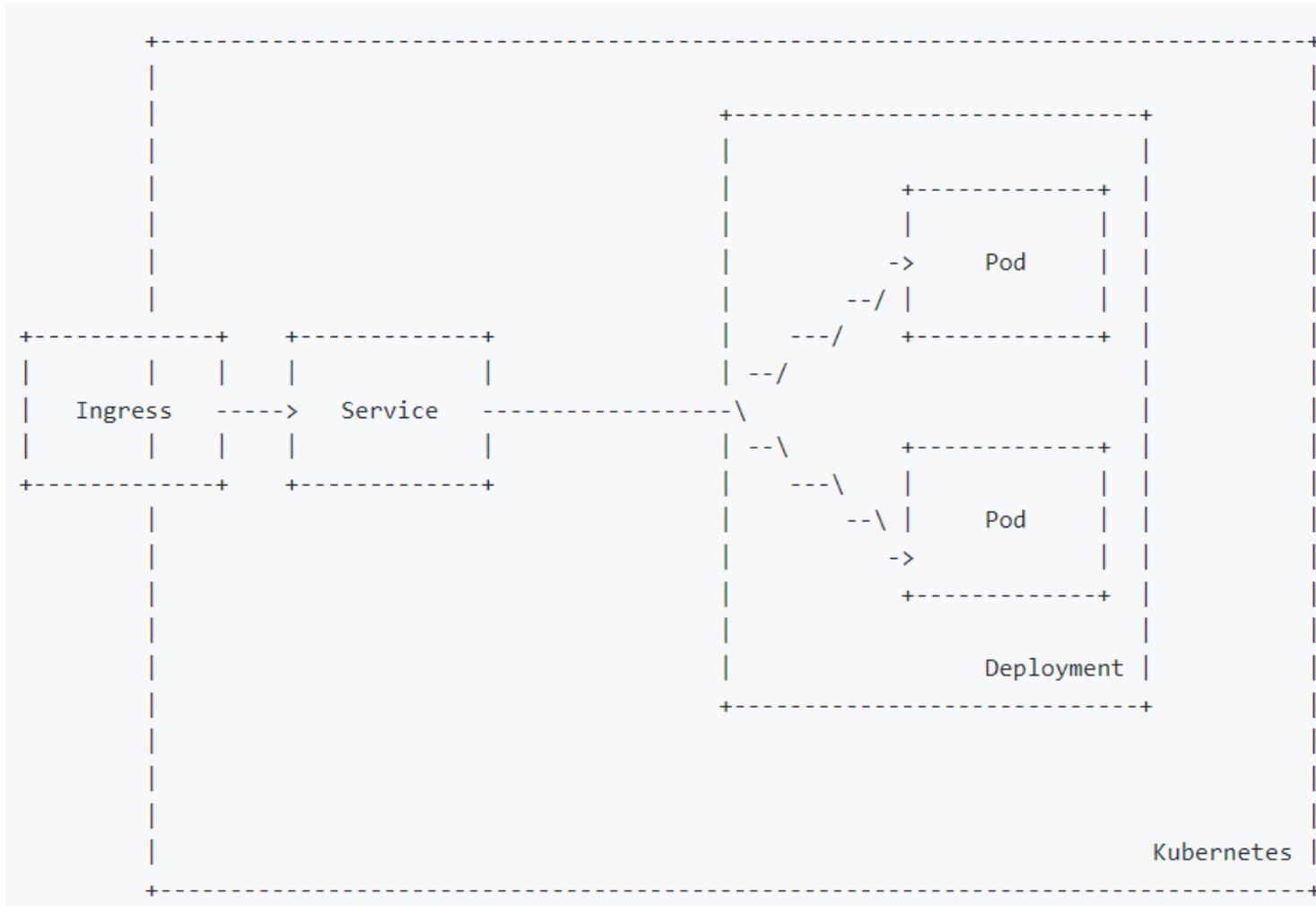
Soluzione: Creiamo un Servizio

```
apiVersion: v1
kind: Service
metadata:
  name: simple-internal-service
spec:
  ports:
    - port: 80
      targetPort: 9876
  selector:
    app: simple-deployment
```

Accediamo al Servizio

- Per lanciare il servizio basta usare il comando `apply`.
- Una volta fatto tramite la bash creata precedentemente eseguiamo il comando «`nslookup NOME_SERVIZIO`» per ottenere l'IP.
- Eseguiamo ora `curl` all'URL «`IP_SERVIZIO/health`» (ricordatevi la porta specificata nel manifesto).
- Cosa accade se proviamo ad accedere al servizio al di fuori del cluster?

Global Overview



Esercizio: Accediamo al Servizio dall'esterno

- Un servizio permette di disaccoppiare l'accesso ad un deployment rispetto alle repliche specifiche. Per rendere però il servizio accessibile esternamente abbiamo bisogno di un «Ingress» da connettere al servizio.
- Prima di tutto dobbiamo installare alcuni addons per minikube con «*minikube addons enable ingress*» e «*minikube addons enable ingress-dns*».
- Esercizio: create un manifesto «*kind:Ingress*» associato al nostro servizio.

```
apiVersion: networking.k8s.io/v1beta1
kind: Ingress
metadata:
  name: ml-product-ingress
  annotations:
    kubernetes.io/ingress.class: "nginx"
    nginx.ingress.kubernetes.io/rewrite-target: /
spec:
  rules:
  - http:
      paths:
      - path: /app
        backend:
          serviceName: user-interface-svc
          servicePort: 80
```

Configure options for the Ingress controller.

How should external traffic access the service?

What Service should we direct traffic to?

Soluzione: Accediamo al Servizio dall'esterno

```
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: simple-ingress
  annotations:
    nginx.ingress.kubernetes.io/ssl-redirect: "false"
spec:
  backend:
    serviceName: simple-internal-service
    servicePort: 80
```

Esercizio: Accediamo al Servizio dall'esterno

- Ora non resta che lanciare l’Ingress con apply e ottenere l’IP del nostro cluster con «minikube ip».
- Adesso possiamo accedere al nostro servizio e quindi al nostro deployment (che gestisce le repliche del nostro Pod) dall’esterno all’IP del cluster e con la porta che abbiamo specificato nel manifesto.

Esercizio

Partendo dalle due immagini

- <https://hub.docker.com/r/yeasy/simple-web> (listens on port **80**)
- <https://hub.docker.com/r/scottyc/webapp> (listens on port **3000**)

scrivere

- un file **pod.yaml** per configurare il pod che gestisce le due immagini
- un file **service.yaml** per esporre il pod come servizio raggiungibile

Per eliminare tutto digitare «kubectl delete ingress,service,deployment,rs,pod --all» e «minikube delete --all» per eliminare il cluster minikube.

Documentazione

- <https://kubernetes.io/docs/tasks/configure-pod-container/static-pod/>
- <https://kubernetes.io/docs/concepts/workloads/pods/>
- <https://kubernetes.io/docs/concepts/services-networking/service/>

More details

<https://github.com/algolia/kubernetes-hands-on>