UNIVERSIDADE
BEIRA INTERIOR

Kubernetes

Tecnologias de Virtualização e Centros de Dados
Mestrado em Engenharia Informática

Alexandre Fonte
alexandre.fonte@ubi.pt
(Professor Convidado)
Departamento de Informática

Ano Letivo 2023/2024

Sumário

- Containers
- Kubernetes
- Kubernetes Architecture
- Pods, Services, Replicasets, and Deployments
- Kubernetes Cluster
  - Minikube
  - AKS and others

Versão: 29 abril de 2022,
Pequenas revisões/atualizações 9 abril de 2023 e 6 abril de 2024.

Containers

- A tecnologia de containers é cada vez mais popular para o desenvolvimento de sistemas e aplicações
  - Não é necessário configurar e gerir hardware
  - Não é necessário configurar e gerir sistemas operativos
  - Permite poupar tempo
  - Permite poupar recursos computacionais
  - Permite poupar custos
- Os sistemas de gestão de containers normais focam-se em containers individuais. Assim, em grandes projetos, alguns aspetos tornam-se difíceis de gerir:
  - Configuração e gestão de load balancing
  - Conectividade Internet
  - Orquestração do processo de lançamento de containers

Plataformas de gestão de containers

- De modo a facilitar a gestão de containers em grandes projetos, foram criadas plataformas de gestão de containers
- A mais popular chama-se kubernetes ou K8s
  - Por vezes é abreviada para **K8s**

kubernetes

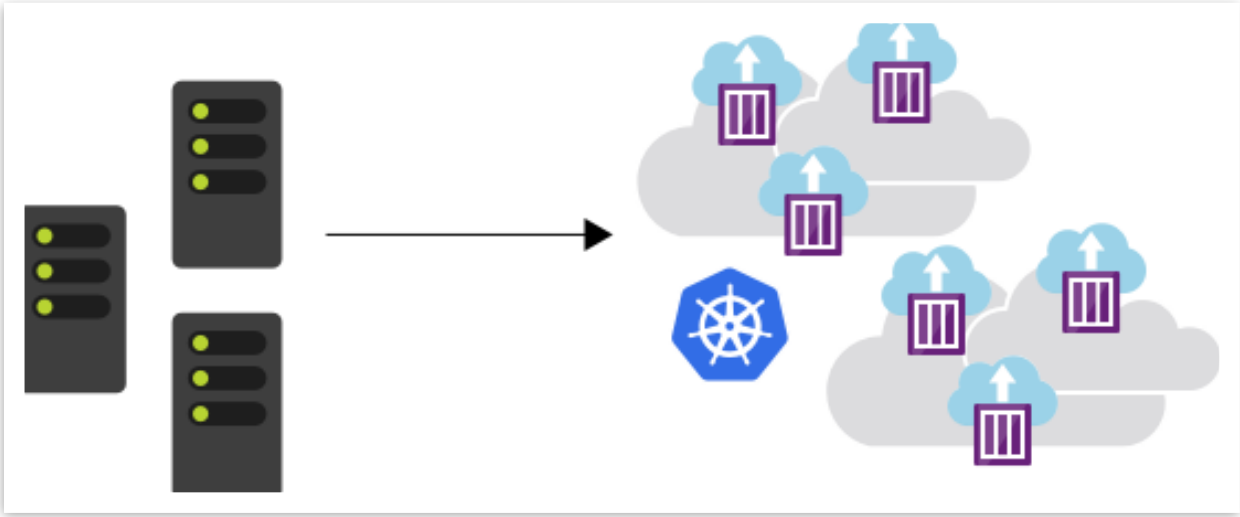
Kubernetes

- "Kubernetes é um sistema de código aberto para a automatização da implantação, dimensionamento e gestão de aplicações *containerizadas*".
- Kubernetes provém da palavra grega κυβερνήτης, que significa timoneiro ou piloto de navio. Com esta analogia em mente, podemos pensar em Kubernetes como o piloto de um navio de contentores.
- Kubernetes é também referido como k8s (pronuncia-se Kate's), uma vez que existem 8 caracteres entre k e s.
- Kubernetes é altamente inspirado pelo sistema Google Borg, um orquestrador de contentores e carga de trabalho para as suas operações globais.
- É um projecto de código aberto escrito na língua Go e licenciado sob a Licença Apache, Versão 2.0.

- Kubernetes foi iniciado pela Google e, com o seu lançamento v1.0 em Julho de 2015, doou-o à Cloud Native Computing Foundation (CNCF), uma das maiores subfundações da Linux Foundation.
- As novas versões Kubernetes são lançadas em ciclos de 4 meses. A versão estável actual é 1.23 (a partir de Abril de 2022).

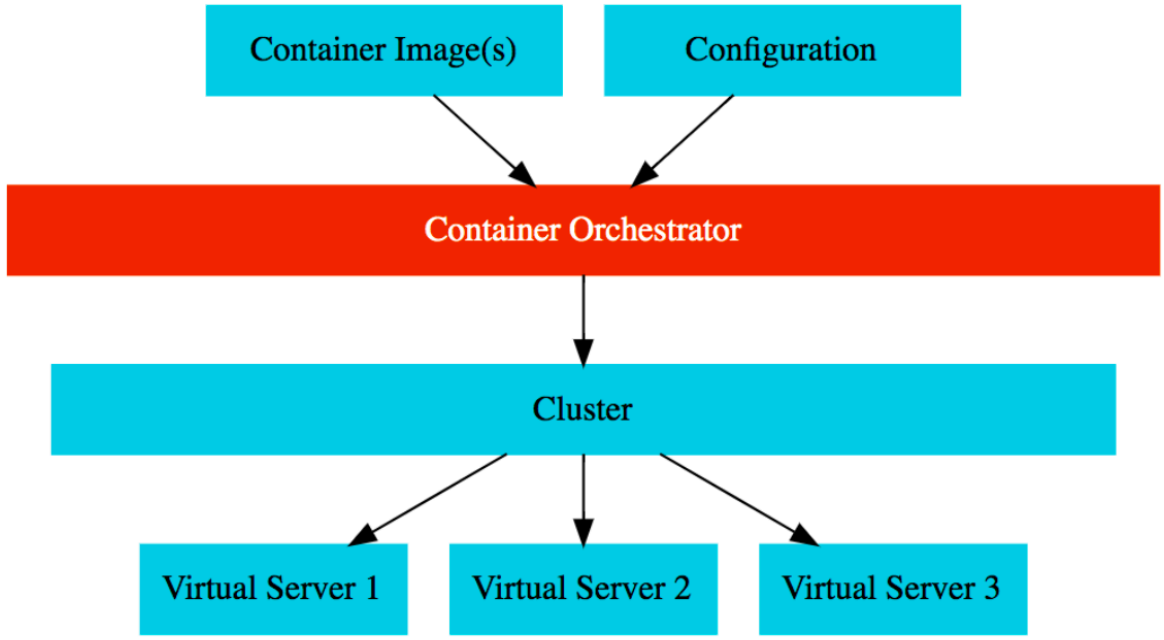
Kubernetes

- Kubernetes é uma plataforma portável, extensível e de código aberto para gerir e orquestrar aplicações baseadas em containers
- Kubernetes cria uma camada de abstração sobre tarefas complexas de gestão de containers



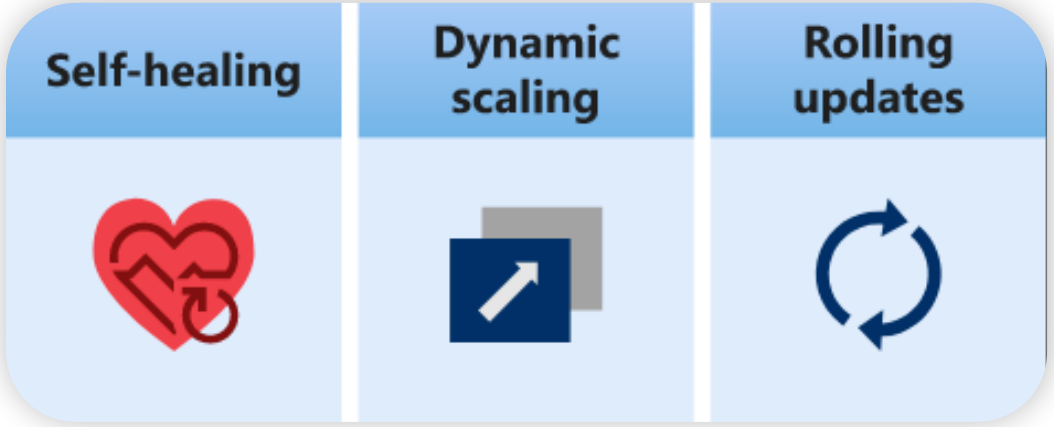
Kubernetes - Orquestração de *Containers*

- **Requisito:** Preciso de 10 instâncias do Contentor do Microserviço A, 15 instancias do Contentor do Microserviço B e …
- Também preciso de:
  - **-Auto Scaling** - Escalar containers com base na procura
  - **-Load Balancing** - Distribuir a carga entre múltiplas instancias do mesmo microserviço
  - **-Self Healing** - Verificação da saúde e substituição das instâncias em falha
  - **-Zero Downtime Deployments** - Atualizações com as novas versões sem downtime



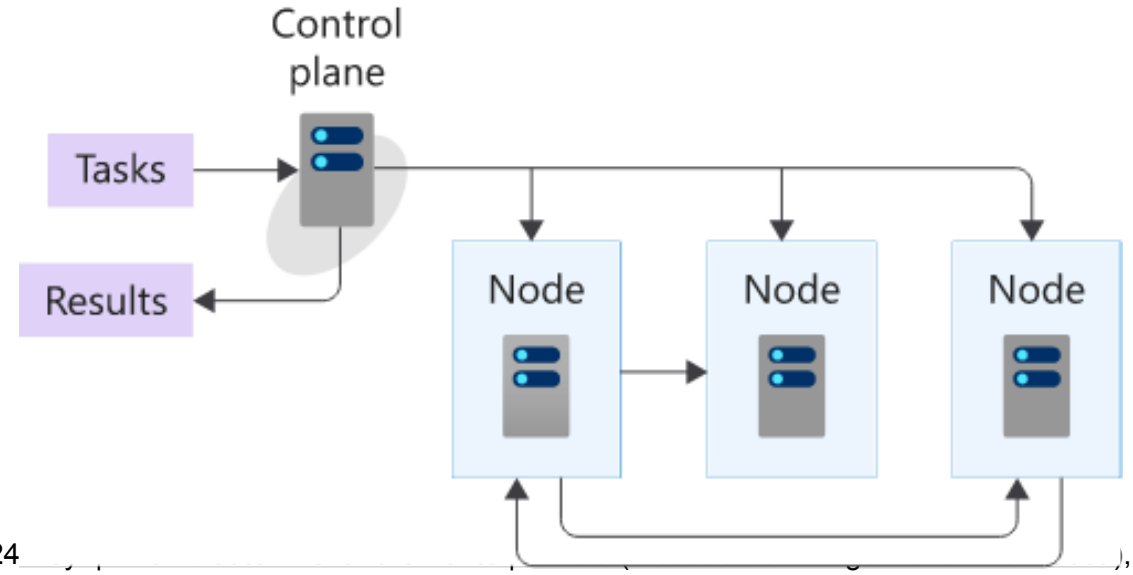
Vantagens de usar Kubernetes

- Faz a cura automática (restart ou replace) de containers
- Faz escalonamento automático em função da procura
- Automatiza as atualizações de containers
- Faz a gestão de storage
- Faz a gestão de tráfego de rede



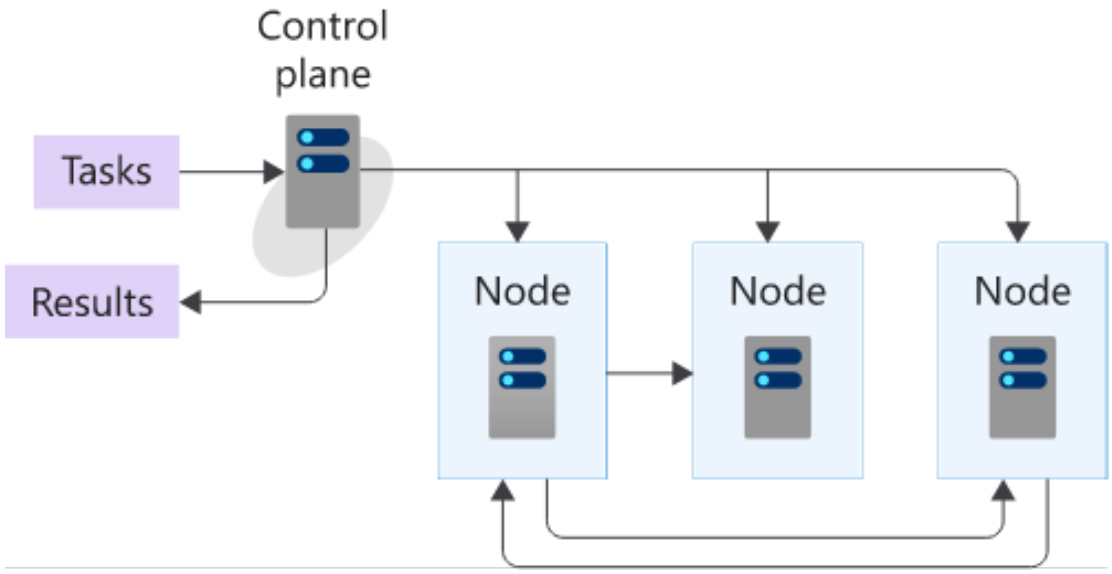
Cluster

- Um cluster é um conjunto de computadores que trabalham em conjunto e que podem ser vistos como um sistema único
- Cada computador de um cluster chama-se um nó (node)
- Os nós são tipicamente configurados para executar o mesmo tipo de tarefas, por exemplo alojar websites, correr containers (nosso foco), etc



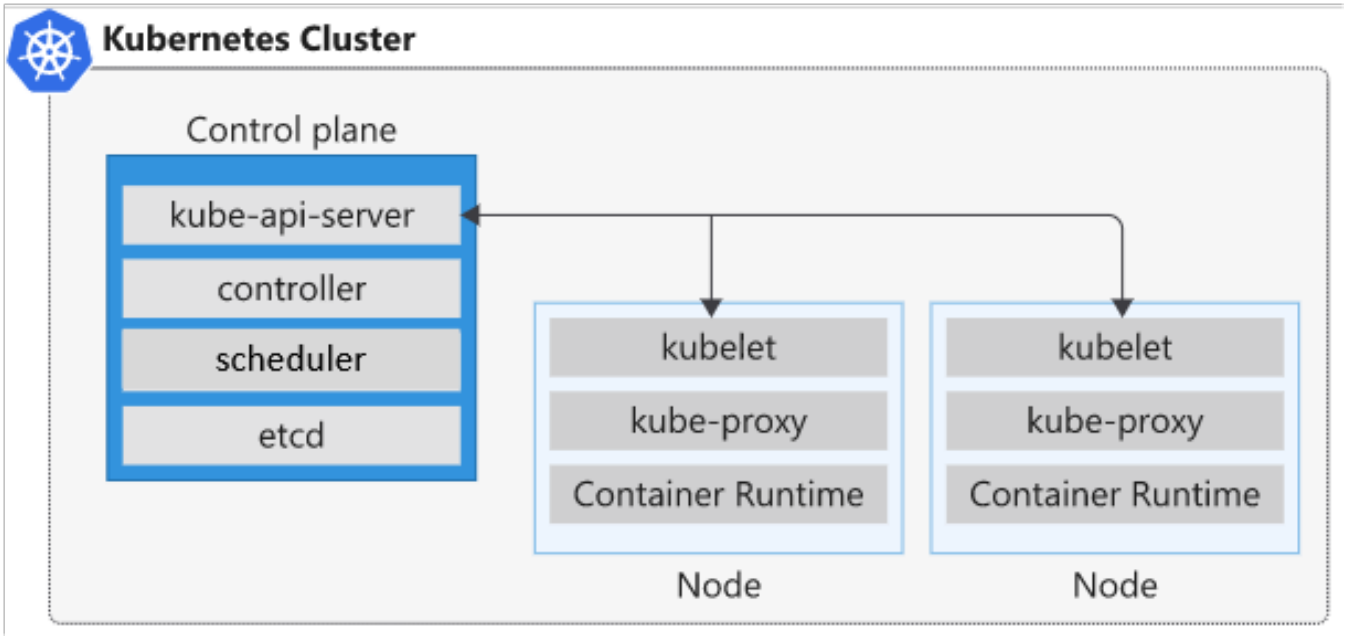
Cluster

- Um cluster usa software centralizado para agendar e controlar as tarefas que são atribuídas aos nós
- Este software centralizado é executado em computadores chamados control planes



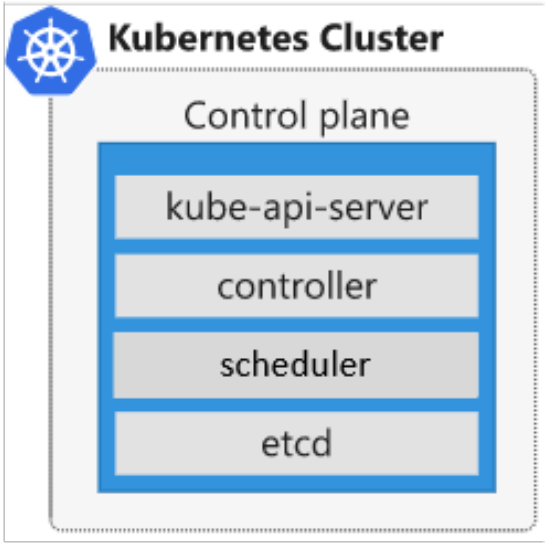
Arquitetura Kubernetes

- Um cluster Kubernetes tem pelo menos um *control plane* e um nó (que executa as aplicações *containerizadas*)
- O plano de controlo gere os nós e os pods no cluster
- Estes componentes podem correr em máquinas físicas, VM, ou instâncias na Cloud
- O OS predefinido em Kubernetes é o Linux



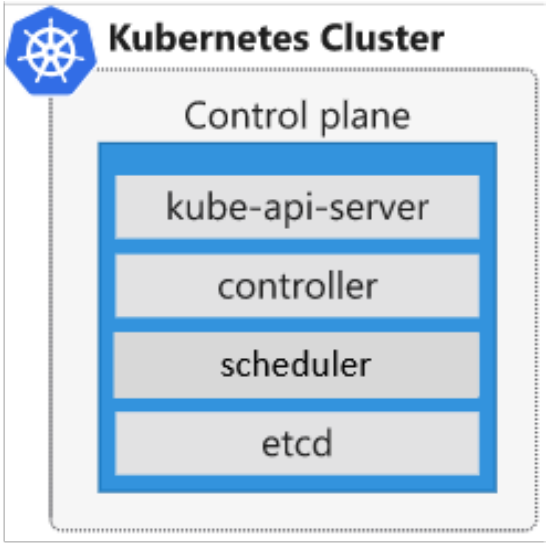
Componentes do control plane

- API server: é o front end de comunicação entre todos os componentes do cluster, através de uma API RESTful
- Scheduler: componente responsável pela atribuição de tarefas (Pods) aos nós. Para cada Pod, procura descobrir numa operação a 2-passos o melhor nó para executar esse Pod, de entre os nós admissíveis (i.e, 1.º-**Filtering**: filtra os nós com recursos suficientes; 2.º **Scoring**: ordena os nós admissíveis com base num score)
- Backing store (etcd): sistema de armazenamento de dados principal do cluster (configuração e estado atual do cluster). É um armazenamento do tipo chave/valor.
  - Tudo o que podemos consultar com comando `kubectl get` é armazenado no Etcd.
  - O comando `kubectl create` cria uma nova entrada no etcd



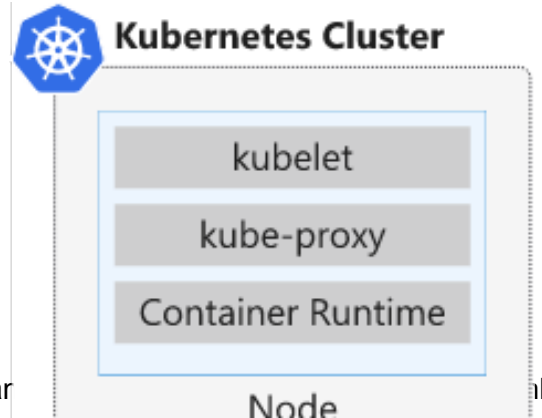
Componentes do control plane

- Controller manager: este componente lança controllers e vigia o estado dos controllers do cluster
- Controller process: um controlador é um processo que vigia continuamente o estado dos objetos no cluster, procurando que estes e cluster fiquem no estado desejado (spec field). Existem vários tipos, alguns são:
  - Node controller: responsável por notificar e dar uma resposta quando um nó fica em baixo
  - Job controller: cria Pods para correr as tarefas
- Cloud controller manager (opcional): integra kubernetes com as eventuais tecnologias cloud no cluster, que podem ser load balancers, filas ou storage



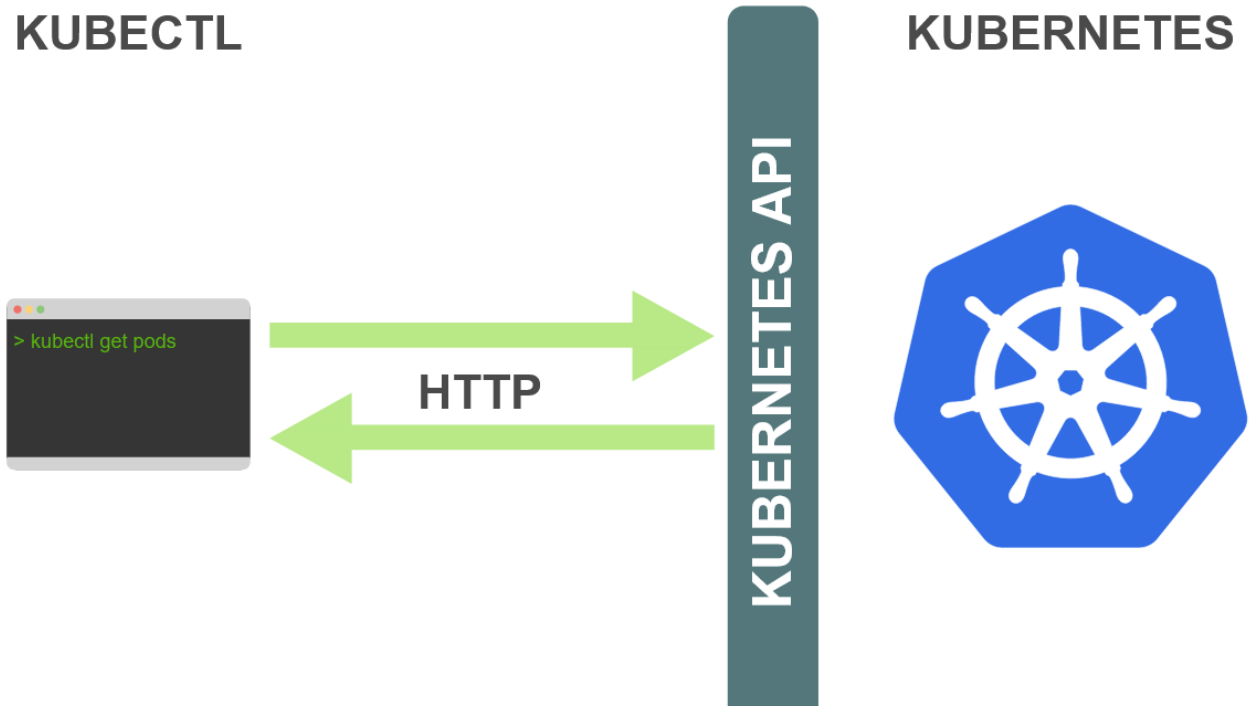
Componentes do nó

- Kubelet: é o agente que corre em cada nó. Este regista cada nó com o API server (pelo hostname; flag; ou outro identificador). Recebe pedidos de tarefas do control plane (através da API) e verifica se as tarefas (os Pods) estão a correr e saudáveis. Mais especificamente, recebe as especificações do Pod (PodSpec) para vigiar os containers descritos no PodSpec.
- Kube-proxy: responsável pela configuração de rede do nó, permitindo a comunicação com os Pods através de sessões de rede internas ou externas ao cluster. Também implementa regras para encaminhamento e load balancing. Em súmula implementa parte do conceito de Service.
- Container runtime: é o componente responsável por correr os containers. As suas funções incluem obter, iniciar e parar as imagens. Kubernetes suporta vários engines, incluindo Docker, rkt, CRI-O, containerd e frakti.



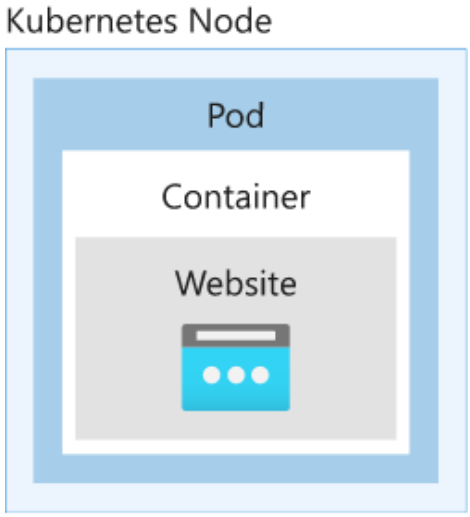
Usando CLI para interagir com kubernetes

- A interação com um cluster kubernetes pode ser feita através de uma ferramenta de linha de comando chamada **kubectl**
- **kubectl** permite fazer o *deploy* de aplicações, inspeccionar e gerir os recursos do cluster e visualizar *logs*.



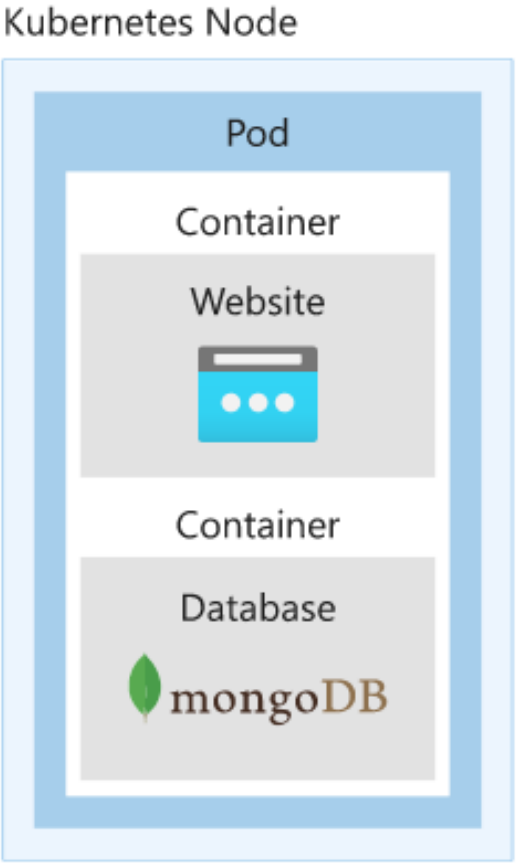
Kubernetes pod

- Um cluster kubernetes não consegue correr diretamente containers
- É preciso primeiro empacotar/encapsular os containers dentro de um objecto chamado **pod**
- O **pod** representa uma instância de uma aplicação a correr num cluster kubernetes
- O **pod** é o objeto mais pequeno que se pode criar em kubernetes

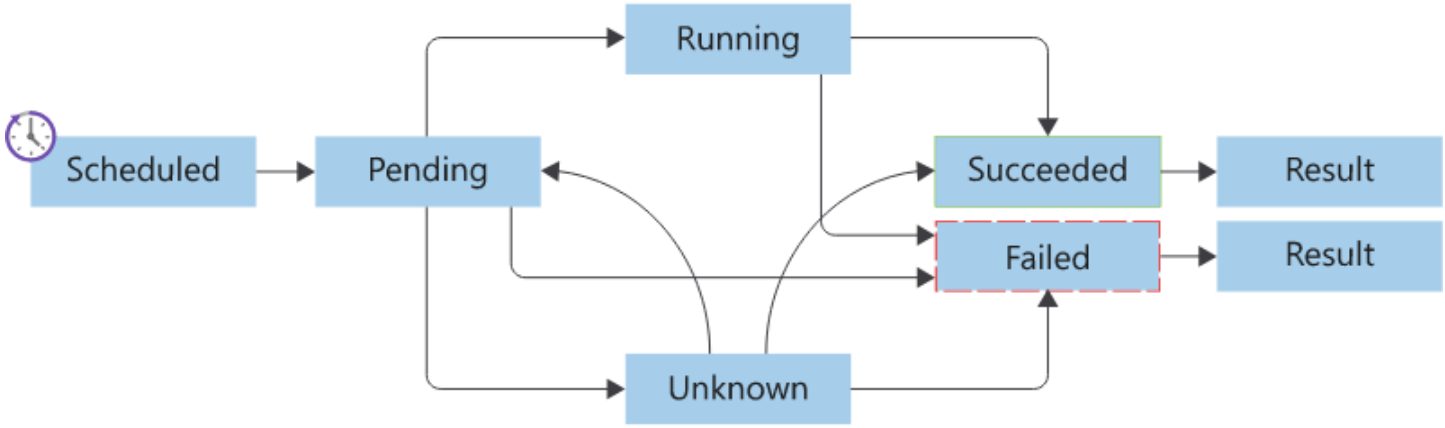


Kubernetes pod

- Um pod pode conter vários containers
- O pod contém também informação sobre:
  - -Storage partilhada
  - -Configuração de rede
  - -Instruções para correr os containers
- Contudo, se for necessário escalar uma aplicação não é possível criar um novo container no pod. É necessário criar um novo pod no mesmo nó ou noutro nó do cluster.



Pod lifecycle



Pending	After the pod run is scheduled, the container runtime downloads container images, and starts all containers for the pod.
Running	The pod transitions to a running state after all of the resources within the pod are ready.
Succeeded	The pod transitions to a succeeded state after the pod completes its intended task, and runs successfully.
Failed	Pods can fail for various reasons. A container in the pod may have failed, leading to the termination of all other containers. Or, maybe an image wasn't found during preparation of the pod containers. In these types of cases, the pod can go to a failed state. Pods can transition to a failed state from either a pending state or a running state. A specific failure can also place a pod back in the pending state.
Unknown	If the state of the pod can't be determined, the pod is an unknown state.

Pod deployment options com CLI (kubectl)

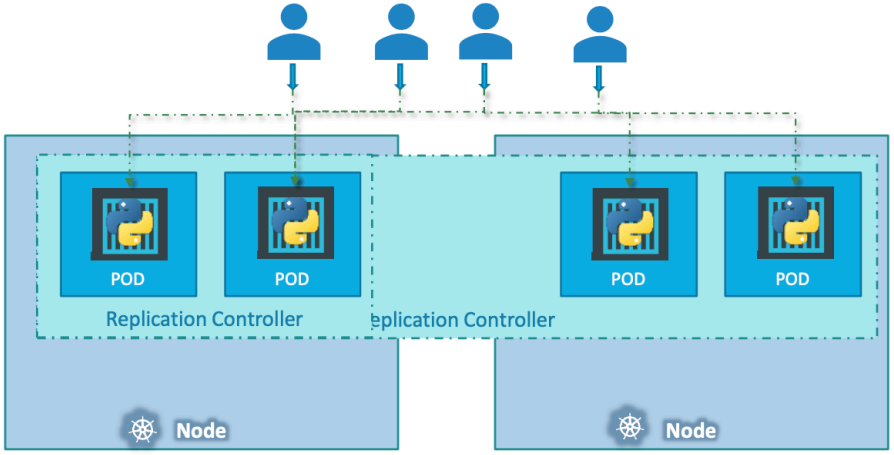
- POD templates: define a configuração do pod através de um ficheiro de configuração. O template tem informação dos nomes das imagens dos containers, o nome do registry a usar para descarregar as imagens, os ports a usar e outros pormenores de runtime.
- Replication controllers: usa pod templates e define um determinado número de pods que têm que correr. O replication controller assegura que esse número de pods está sempre a correr, substituindo dinamicamente os que eventualmente falhem
- Replica sets: semelhante aos replication controllers **(é uma versão moderna)**, mas adicionalmente consegue identificar pods através de uma etiqueta chamada selector value
- Deployments: cria um objeto de gestão acima de um replica set, útil para gerir os updates dos pods num cluster
  - rolling updates: lança pods de uma nova versão sem desativar antes os da versão anterior, o que permite zero downtime
  - rollback: reverte um update em caso de necessidade, mais uma vez com zero downtime

Replication Controller

- Replication controllers: usa pod templates e define um determinado número de pods que têm que correr. O replication controller assegura que esse número de pods está sempre a correr, substituindo dinamicamente os que eventualmente falhem
-

High Availability (HA)

Load-Balancing and Scaling

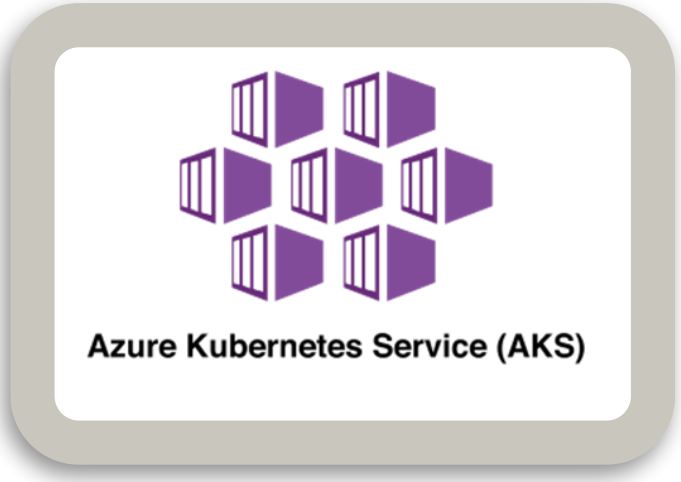


Replicação para assegurar tolerância a falhas. Mesmo existindo um único Pod em execução, se este falhar o replication controller recupera-o.

Como implementar kubernetes

Numa infraestrutura própria ou Através de uma plataforma cloud

- É necessário criar o cluster
- Implementar um ambiente de runtime (Docker ou outro)
- Arranjar soluções para load balancing, logging, monitoring
- É necessário gerir as redes virtuais, OS, memória, storage, etc



O serviço AKS providencia um ambiente kubernetes integrado no Azure, o que reduz a complexidade da sua implementação

Máquina Local usando o Minikube cluster

- minikube is local Kubernetes, focusing on making it easy to learn and develop for Kubernetes. (https://minikube.sigs.k8s.io/docs/start/)
- All you need is Docker (or similarly compatible) container or a Virtual Machine environment, and Kubernetes is a single command away:
  `minikube start`
- Download and install

Mac OS Systems (Intel x86-64)

```
curl -LO https://storage.googleapis.com/minikube/releases/latest/minikube-darwin-amd64
sudo install minikube-darwin-amd64 /usr/local/bin/minikube
```

Linux Systems (Intel x86-64)

```
curl -LO https://storage.googleapis.com/minikube/releases/latest/minikube-darwin-amd64
sudo install minikube-darwin-amd64 /usr/local/bin/minikube
```

Minikube cluster (https://minikube.sigs.k8s.io/docs/start/)

- Download and install (cont.)

Windows Systems via Powershell

```
New-Item -Path 'c:\' -Name 'minikube' -ItemType Directory -Force
Invoke-WebRequest -OutFile 'c:\minikube\minikube.exe' -Uri 'https://github.com/kubernetes/minikube/releases/latest/download/minikube-windows-amd64.exe' -UseBasicParsing

$oldPath = [Environment]::GetEnvironmentVariable('Path', [EnvironmentVariableTarget]::Machine)
if ($oldPath.Split(';') -inotcontains 'C:\minikube'){ `
  [Environment]::SetEnvironmentVariable('Path', $('{0};C:\minikube' -f $oldPath), [EnvironmentVariableTarget]::Machine) `
}
```

Interact with Minikube Cluster

- minikube start
- minikube stop
- minikube pause
  - Pause Kubernetes without impacting deployed applications
- minikube unpause
- minikube dashboard
- minikube dashboard —url
  - Getting just url of the dashboard
- minikube config set memory 16384
  - Increase the default memory limit (requires a restart)

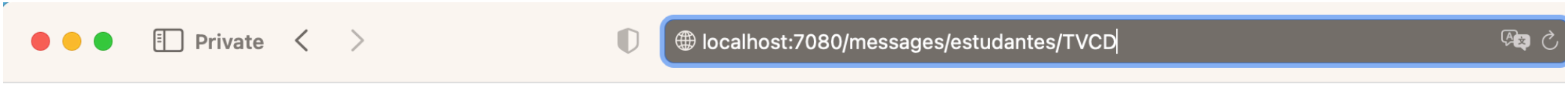
1. Deploy an Application into a Cluster

- Deploy and expose an App on Port 8080 using service type: NodeType
  - **kubectl create deployment** *hello-world-rest-api* --image=adfonte/hello-server
    - **Note:** Creates a new POD with a single container, the name of the deployment in this example is hello-world-rest-api
  - **kubectl expose deployment** *hello-world-rest-api* --port=8080 --type=NodePort --name=hello-word-rest-api-service
    - Exposes the service object for an application externally so that clients outside of the Kubernetes cluster can interface with the application
- **How to access this service in Minikube**
  - The easiest way to access this service is to let minikube launch a web browser for you (may not work when using minikube on Docker)
    - **minikube service *hello-world-rest-api***
  - Alternatively, use kubectl to forward the port:
    - **kubectl port-forward <*pod name*> 7080:8080**
    - Tada! Your application is now available at **`http://localhost:7080/`**
    - How to Get Pod name:
      - **kubectl get pods**
    - Go to the Browser and Enter the following URI

      `http://localhost:7080/messages/estudantes/TVCD`

1. Deploy an Application into a Cluster (Cont.)

```
afs-MacBook-Air:~ alexandrefonte$ kubectl get pods
NAME                                    READY   STATUS    RESTARTS   AGE
hello-world-rest-api-84bf8c5ddf-sftnj   1/1     Running   0          34m
afs-MacBook-Air:~ alexandrefonte$ kubectl port-forward hello-world-rest-api-84bf8c5ddf-sftnj 7080:8080
Forwarding from 127.0.0.1:7080 -> 8080
Forwarding from [::1]:7080 -> 8080
Handling connection for 7080
```



Good moorning Dear estudantes from our TVCD Docker Container!

It is 13:13, now.

Deploy an Application into a Cluster (Cont.) - Load Balancing

- Deploy, and Expose on Port 8080 using the LoadBalancer service type

  -kubectl create deployment *hello-world-rest-api* --image=adfonte/hello-server

  -kubectl expose deployment *hello-world-rest-api* --port=8080 --type=LoadBalancer --name=*hello-world-rest-service-lb*

  -minikube tunnel

    •connect to LB services (introduzir numa janela à parte) - just in the case of Minikube

  -kubectl get services

    •To check namely the allocated external IP

- Scale a pod deployment to 3 replicas

  -kubectl scale deployment *hello-world-rest-api* --replicas=3

Deploy an Application into a Cluster (Cont.) - Load Balancing

- Deploy, and Expose on Port 8080 using the LoadBalancer service type

```
afs-MacBook-Air:~ alexandrefonte$ kubectl expose deployment hello-world-rest-api --port=8080 --type=LoadBalancer --name=hello-world-rest-service-lb
service/hello-world-rest-service-lb exposed
afs-MacBook-Air:~ alexandrefonte$ kubectl get services
NAME                         TYPE           CLUSTER-IP      EXTERNAL-IP   PORT(S)          AGE
hello-world-rest-service-lb  LoadBalancer   10.99.181.183   <pending>     8080:31558/TCP   6s
kubernetes                   ClusterIP      10.96.0.1       <none>        443/TCP          12d
afs-MacBook-Air:~ alexandrefonte$ kubectl get services
NAME                         TYPE           CLUSTER-IP      EXTERNAL-IP   PORT(S)          AGE
hello-world-rest-service-lb  LoadBalancer   10.99.181.183   127.0.0.1     8080:31558/TCP   63s
kubernetes                   ClusterIP      10.96.0.1       <none>        443/TCP          12d
```

In case of Minikube running as a VM, appears a different IP.
In a "real" cluster, you should get a public external IP.

Deploy an Application into a Cluster (Cont.) - Load Balancing and Auto Scaling

- Deploy, and Expose on Port 8080 using the LoadBalancer service type and AutoScaling

  -kubectl create deployment *hello-world-rest-api* --image=adfonte/hello-server

  -kubectl expose deployment *hello-world-rest-api* --port=8080 --type=LoadBalancer --name=*hello-world-rest-service-lb*

- AutoScale a pod deployment to 10 replicas

  -kubectl autoscale deployment *hello-world-rest-api* --max=10 --cpu-percent=70

  -kubectl autoscale deployment *hello-world-rest-api* --min=3 --max=10 --cpu-percent=70

Deploy an Application into a Cluster (Cont.) - Load Balancing and Auto Scaling

- A **HorizontalPodAutoscaler** (HPA) automatically updates a workload resource (such as a Deployment or StatefulSet), with the aim of automatically scaling the workload to match demand.
- Horizontal scaling means that the response to increased load is to deploy more Pods. This is different from vertical scaling, which for Kubernetes would mean assigning more resources (for example: memory or CPU) to the Pods (…)
- If the load decreases, and the number of Pods is above the configured minimum, the HorizontalPodAutoscaler instructs the workload resource to scale back down.

```
afs-MacBook-Air:~ alexandrefonte$ kubectl get hpa
NAME                  REFERENCE                        TARGETS         MINPODS   MAXPODS   REPLICAS   AGE
hello-world-rest-api  Deployment/hello-world-rest-api  <unknown>/70%   3         10        3          11m
```

Precisamos de ativar um metrics-server!
Precisamos de alguma carga!

Deploy an Application into a Cluster (Cont.) - Load Balancing and Auto Scaling

- Ativação de um metrics-server no minikube k8s

-minikube addons enable metrics-server

```
afs-MacBook-Air:~ alexandrefonte$ minikube addons enable metrics-server
    ▪ Using image k8s.gcr.io/metrics-server/metrics-server:v0.4.2
🌟  The 'metrics-server' addon is enabled
```

```
afs-MacBook-Air:~ alexandrefonte$ minikube addons list
|-----------------------------|----------|--------------|--------------------------------|
|         ADDON NAME          | PROFILE  |    STATUS    |           MAINTAINER           |
|-----------------------------|----------|--------------|--------------------------------|
| ambassador                  | minikube | disabled     | third-party (ambassador)       |
| auto-pause                  | minikube | disabled     | google                         |
| csi-hostpath-driver         | minikube | disabled     | kubernetes                     |
| dashboard                   | minikube | enabled ✅   | kubernetes                     |
| default-storageclass        | minikube | enabled ✅   | kubernetes                     |
| efk                         | minikube | disabled     | third-party (elastic)          |
| freshpod                    | minikube | disabled     | google                         |
| gcp-auth                    | minikube | disabled     | google                         |
| gvisor                      | minikube | disabled     | google                         |
| helm-tiller                 | minikube | disabled     | third-party (helm)             |
| ingress                     | minikube | disabled     | unknown (third-party)          |
| ingress-dns                 | minikube | disabled     | google                         |
| istio                       | minikube | disabled     | third-party (istio)            |
| istio-provisioner           | minikube | disabled     | third-party (istio)            |
| kong                        | minikube | disabled     | third-party (Kong HQ)          |
| kubevirt                    | minikube | disabled     | third-party (kubevirt)         |
| logviewer                   | minikube | disabled     | unknown (third-party)          |
| metallb                     | minikube | disabled     | third-party (metallb)          |
| metrics-server              | minikube | disabled     | kubernetes                     |
| nvidia-driver-installer     | minikube | disabled     | google                         |
| nvidia-gpu-device-plugin    | minikube | disabled     | third-party (nvidia)           |
| olm                         | minikube | disabled     | third-party (operator          |
|                             |          |              | framework)                     |
| pod-security-policy         | minikube | disabled     | unknown (third-party)          |
| portainer                   | minikube | disabled     | portainer.io                   |
| registry                    | minikube | disabled     | google                         |
| registry-aliases            | minikube | disabled     | unknown (third-party)          |
| registry-creds              | minikube | disabled     | third-party (upmc enterprises) |
| storage-provisioner         | minikube | enabled ✅   | google                         |
| storage-provisioner-gluster | minikube | disabled     | unknown (third-party)          |
| volumesnapshots             | minikube | disabled     | kubernetes                     |
|-----------------------------|----------|--------------|--------------------------------|
```

```
afs-MacBook-Air:~ alexandrefonte$ minikube addons list
|-----------------------------|----------|--------------|--------------------------------|
|         ADDON NAME          | PROFILE  |    STATUS    |           MAINTAINER           |
|-----------------------------|----------|--------------|--------------------------------|
| ambassador                  | minikube | disabled     | third-party (ambassador)       |
| auto-pause                  | minikube | disabled     | google                         |
| csi-hostpath-driver         | minikube | disabled     | kubernetes                     |
| dashboard                   | minikube | enabled ✅   | kubernetes                     |
| default-storageclass        | minikube | enabled ✅   | kubernetes                     |
| efk                         | minikube | disabled     | third-party (elastic)          |
| freshpod                    | minikube | disabled     | google                         |
| gcp-auth                    | minikube | disabled     | google                         |
| gvisor                      | minikube | disabled     | google                         |
| helm-tiller                 | minikube | disabled     | third-party (helm)             |
| ingress                     | minikube | disabled     | unknown (third-party)          |
| ingress-dns                 | minikube | disabled     | google                         |
| istio                       | minikube | disabled     | third-party (istio)            |
| istio-provisioner           | minikube | disabled     | third-party (istio)            |
| kong                        | minikube | disabled     | third-party (Kong HQ)          |
| kubevirt                    | minikube | disabled     | third-party (kubevirt)         |
| logviewer                   | minikube | disabled     | unknown (third-party)          |
| metallb                     | minikube | disabled     | third-party (metallb)          |
| metrics-server              | minikube | enabled ✅   | kubernetes                     |
| nvidia-driver-installer     | minikube | disabled     | google                         |
| nvidia-gpu-device-plugin    | minikube | disabled     | third-party (nvidia)           |
| olm                         | minikube | disabled     | third-party (operator          |
|                             |          |              | framework)                     |
| pod-security-policy         | minikube | disabled     | unknown (third-party)          |
| portainer                   | minikube | disabled     | portainer.io                   |
| registry                    | minikube | disabled     | google                         |
| registry-aliases            | minikube | disabled     | unknown (third-party)          |
| registry-creds              | minikube | disabled     | third-party (upmc enterprises) |
| storage-provisioner         | minikube | enabled ✅   | google                         |
| storage-provisioner-gluster | minikube | disabled     | unknown (third-party)          |
| volumesnapshots             | minikube | disabled     | kubernetes                     |
|-----------------------------|----------|--------------|--------------------------------|
```

Deploy an Application into a Cluster (Cont.) - Load Balancing and Auto Scaling

- Verificação se o metrics-server está em execução no Cluster
  - **-kubectl get pods --all-namespaces | grep metrics-server**

```
alexandrefonte — -bash — 107×24
afs-MacBook-Air-2:~ alexandrefonte$ kubectl get pods --all-namespaces | grep metrics-server
kube-system            metrics-server-6588d95b98-rcdgk             1/1     Running   0             17m
afs-MacBook-Air-2:~ alexandrefonte$
```

- Consultar as métricas através da metrics API
  - **-# Get the metrics for all nodes**
  - **-kubectl get --raw /apis/**metrics.k8s.io/v1beta1/nodes
  - **-# Get the metrics for all pods**
  - **-kubectl get --raw /apis/**metrics.k8s.io/v1beta1/pods
- Consultar os pods “Once it is enabled, you can view the metrics of a running pod using the following command”:
  - **-kubectl top pods**

```
alexandrefonte — -bash — 127×24
afs-MacBook-Air-2:~ alexandrefonte$ kubectl top pod
NAME                                    CPU(cores)   MEMORY(bytes)
hello-world-rest-api-6fbb455794-9dpkd   4m           189Mi
hello-world-rest-api-6fbb455794-bg7mk   4m           206Mi
hello-world-rest-api-6fbb455794-lcv5g   4m           219Mi
afs-MacBook-Air-2:~ alexandrefonte$
```

Deploy an Application into a Cluster (Cont.) - Load Balancing and Auto Scaling

- Algoritmo:

desiredReplicas = ceil[currentReplicas * (currentMetricValue / desiredMetricValue)]

Exemplo:

desiredReplicas = 1 * 305 / 50 = 6.1
(Arrendondado para cima)

```
NAME          REFERENCE                          TARGET      MINPODS   MAXPODS   REPLICAS   AGE
php-apache    Deployment/php-apache/scale        305% / 50%  1         10        1          3m
```

```
NAME          REFERENCE                          TARGET      MINPODS   MAXPODS   REPLICAS   AGE
php-apache    Deployment/php-apache/scale        305% / 50%  1         10        7          3m
```

Deploy an Application into a Cluster (Cont.) - Load Balancing and Auto Scaling

```
afs-MacBook-Air:~ alexandrefonte$ kubectl autoscale deployment hello-world-rest-api --max=10 --cpu-percent=70
horizontalpodautoscaler.autoscaling/hello-world-rest-api autoscaled
afs-MacBook-Air:~ alexandrefonte$ kubectl autoscale deployment hello-world-rest-api --min=3 --max=10 --cpu-percent=70
Error from server (AlreadyExists): horizontalpodautoscalers.autoscaling "hello-world-rest-api" already exists
afs-MacBook-Air:~ alexandrefonte$ kubectl get all
NAME                                     READY   STATUS    RESTARTS   AGE
pod/hello-world-rest-api-84bf8c5ddf-hr7rz   1/1     Running   0          14m

NAME                                 TYPE           CLUSTER-IP      EXTERNAL-IP   PORT(S)          AGE
service/hello-world-rest-service-lb   LoadBalancer   10.99.181.183   127.0.0.1     8080:31558/TCP   13m
service/kubernetes                   ClusterIP      10.96.0.1       <none>        443/TCP          12d

NAME                                  READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/hello-world-rest-api   1/1     1            1           14m

NAME                                           DESIRED   CURRENT   READY   AGE
replicaset.apps/hello-world-rest-api-84bf8c5ddf   1         1         1       14m

NAME                                                     REFERENCE                         TARGETS         MINPODS   MAXPODS   REPLICAS   AGE
horizontalpodautoscaler.autoscaling/hello-world-rest-api   Deployment/hello-world-rest-api   <unknown>/70%   1         10        1          47s
afs-MacBook-Air:~ alexandrefonte$ kubectl delete horizontalpodautoscaler.autoscaling/hello-world-rest-api
horizontalpodautoscaler.autoscaling "hello-world-rest-api" deleted
afs-MacBook-Air:~ alexandrefonte$ kubectl autoscale deployment hello-world-rest-api --min=3 --max=10 --cpu-percent=70
horizontalpodautoscaler.autoscaling/hello-world-rest-api autoscaled
afs-MacBook-Air:~ alexandrefonte$ kubectl get all
NAME                                     READY   STATUS    RESTARTS   AGE
pod/hello-world-rest-api-84bf8c5ddf-422lg   1/1     Running   0          28s
pod/hello-world-rest-api-84bf8c5ddf-6zphf   1/1     Running   0          28s
pod/hello-world-rest-api-84bf8c5ddf-hr7rz   1/1     Running   0          16m

NAME                                 TYPE           CLUSTER-IP      EXTERNAL-IP   PORT(S)          AGE
service/hello-world-rest-service-lb   LoadBalancer   10.99.181.183   127.0.0.1     8080:31558/TCP   14m
service/kubernetes                   ClusterIP      10.96.0.1       <none>        443/TCP          12d

NAME                                  READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/hello-world-rest-api   3/3     3            3           16m

NAME                                           DESIRED   CURRENT   READY   AGE
replicaset.apps/hello-world-rest-api-84bf8c5ddf   3         3         3       16m

NAME                                                     REFERENCE                         TARGETS         MINPODS   MAXPODS   REPLICAS   AGE
horizontalpodautoscaler.autoscaling/hello-world-rest-api   Deployment/hello-world-rest-api   <unknown>/70%   3         10        3          34s
```

Get Pods, Services, Replicasets and Deployments

- kubectl get services *hello-world-rest-api*
- kubectl get pods
  - To check that an application deployment create one or more pods
- Kubectl get describe pods
  - To get more detailed info about pods
- kubectl get replicasets
- Kubectl get nodes
  - To get the list and status about nodes in a cluster
- Kubectl get describe nodes *<name of the node>*
  - To get more detailed information about a node

Complete reference commands:
https://kubernetes.io/docs/reference/kubectl/

Deleting Pods, Services or Deployments

- **kubectl delete pod** *hello-world-rest-api-pod*
- **kubectl delete service** *hello-world-rest-api*
- **kubectl delete deployment** *hello-world-rest-api*

Complete reference commands:
https://kubernetes.io/docs/reference/kubectl/

Create a Pod with Yaml

- pod-definition.yaml

```
apiVersion: v1

kind: Pod
metadata:
     name: myapp-pod
     labels:
       app: myapp
       type: front-end
spec:
     containers:
      - name: nginx-container
        image: nginx:1.7.0
        ports:
         - containerPort: 80
      - name: postgres
        image: postgres
        env:
         - name: POSTGRES_PASSWORD
           value: mysecretpassword
```

Pod Template

kind: can be a pod, service, replicaSet, deployment

metadata: is a dictionary with the data about the object

labels: is a dictionary with several data defined freely

containers: is a List/Array of the pod containers

spec: is a dictionary with the specification of the pod

```
kubectl create -f pod-definition.yml
Kubectl get pods
```

Atividade - Create a Pod with Yaml

- Create a Pod named **hello-word-rest-api-pod** based on adfonte/hello-server image. Expose Pod on Port 8080.

  - Manually using the **kubctl run** command

```
afs-MacBook-Air:kubernetes alexandrefonte$ kubectl run hello-world-rest-api-pod --image=adfonte/hello-server
pod/hello-world-rest-api-pod created
afs-MacBook-Air:kubernetes alexandrefonte$ kubectl get pods
NAME                        READY   STATUS    RESTARTS   AGE
hello-world-rest-api-pod    1/1     Running   0          19s
```

  - With a Yaml file

```
afs-MacBook-Air:kubernetes alexandrefonte$ kubectl create -f pod-definition.yaml
pod/hello-world-rest-api-pod created
afs-MacBook-Air:kubernetes alexandrefonte$ kubectl get pods
NAME                        READY   STATUS              RESTARTS   AGE
hello-world-rest-api-pod    0/1     ContainerCreating   0          3s
```

```yaml
apiVersion: v1
kind: Pod
metadata:
     name: hello-world-rest-api-pod
     labels:
       app: hello-world-rest-api
       type: front-end
spec:
     containers:
      - name: hello-world-rest-api
        image: adfonte/hello-server
        ports:
         - containerPort: 8080
```

Create a Pod with Yaml

- Result:

In case of
Minikube running as a VM,
appears a different IP

```
afs-MacBook-Air:kubernetes alexandrefonte$ kubectl expose pod hello-world-rest-api-pod --port=8080 --type=LoadBalancer --name=hello-world-service-lb
service/hello-world-service-lb exposed
afs-MacBook-Air:kubernetes alexandrefonte$ kubectl get services
NAME                    TYPE           CLUSTER-IP       EXTERNAL-IP   PORT(S)          AGE
hello-world-service-lb  LoadBalancer   10.108.105.231   127.0.0.1     8080:32168/TCP   2m1s
kubernetes              ClusterIP      10.96.0.1        <none>        443/TCP          7d5h
```

Create a Replication Controller with Yaml

- rc-definition.yaml

```yaml
apiVersion: v1
kind: ReplicationController
metadata:
  name: myapp-rc
  labels:
    app: myapp
    type: front-end
spec:
  template:
    metadata:
      name: myapp-pod
      labels:
        app: myapp
        type: front-end
    spec:
      containers:
       - name: nginx-container
         image: nginx:1.7.0
         ports:
          - containerPort: 80
       - name: postgres
         image: postgres
         env:
          - name: POSTGRES_PASSWORD
            value: mysecretpassword
  replicas: 3
```

Pod Template

spec: now it includes the specification of pods

template: the template section declares pods

replicas: declares the number of replicas of pods

```
kubectl create -f rc-definition.yaml
kubectl get replicationcontroller
Kubectl get pods
```

Atividade - Create a Replication Controller with Yaml

- Create a Replication Controller named **hello-word-rest-api-rc** based on adfonte/hello-server image. Create 3 replicas.
- Expose replication controller on Port 8080 using a LoadBalancer.

```yaml
apiVersion: v1
kind: ReplicationController
metadata:
     name: hello-world-rest-api-rc
     labels:
       app: hello-world-rest-api
       type: front-end
spec:
    template:
       metadata:
            name: hello-world-rest-api-pod
            labels:
              app: hello-world-rest-api
              type: front-end
       spec:
            containers:
             - name: hello-world-rest-api
               image: adfonte/hello-server
               ports:
                - containerPort: 8080
    replicas: 3
```

Create a Replication Controller with Yaml

- Result:

```
replicationcontroller/hello-world-rest-api-rc created
afs-MacBook-Air:kubernetes alexandrefonte$ kubectl get pods
NAME                          READY   STATUS    RESTARTS   AGE
hello-world-rest-api-rc-g5d5f   1/1     Running   0          16s
hello-world-rest-api-rc-l6p5c   1/1     Running   0          16s
hello-world-rest-api-rc-rhcrp   1/1     Running   0          16s
afs-MacBook-Air:kubernetes alexandrefonte$ kubectl get all
NAME                              READY   STATUS    RESTARTS   AGE
pod/hello-world-rest-api-rc-g5d5f   1/1     Running   0          33s
pod/hello-world-rest-api-rc-l6p5c   1/1     Running   0          33s
pod/hello-world-rest-api-rc-rhcrp   1/1     Running   0          33s

NAME                                        DESIRED   CURRENT   READY   AGE
replicationcontroller/hello-world-rest-api-rc   3         3         3       33s

NAME                 TYPE        CLUSTER-IP    EXTERNAL-IP   PORT(S)   AGE
service/kubernetes   ClusterIP   10.96.0.1     <none>        443/TCP   7d5h
afs-MacBook-Air:kubernetes alexandrefonte$ kubectl expose replicationcontroller  hello-world-rest-api-rc --port=8080 --type=LoadBalancer --name=hello-world-service-lb
service/hello-world-service-lb exposed
afs-MacBook-Air:kubernetes alexandrefonte$ kubectl get services
NAME                    TYPE           CLUSTER-IP       EXTERNAL-IP   PORT(S)          AGE
hello-world-service-lb   LoadBalancer   10.111.135.185   127.0.0.1     8080:31185/TCP   5m43s
kubernetes               ClusterIP      10.96.0.1        <none>        443/TCP          7d5h
```

In case of Minikube running as a VM, appears a different IP

Now Scale to 6 replicas:

```
afs-MacBook-Air:kubernetes alexandrefonte$ kubectl scale --replicas=6  -f rc-definition.yaml
replicationcontroller/hello-world-rest-api-rc scaled
afs-MacBook-Air:kubernetes alexandrefonte$ kubectl get replicationcontroller
NAME                      DESIRED   CURRENT   READY   AGE
hello-world-rest-api-rc   6         6         6       11m
```

Create a Replicaset with Yaml

- replicaset-definition.yaml

```yaml
apiVersion: apps/v1
kind: ReplicaSet
metadata:
  name: myapp-replicaset
  labels:
    app: myapp
    tier: front-end
spec:
  template:
    metadata:
      name: myapp-pod
      labels:
        app: myapp
        tier: front-end
    spec:
      containers:
      - name: nginx-container
        image: nginx:1.7.0
        ports:
         - containerPort: 80
      - name: postgres
        image: postgres
        env:
         - name: POSTGRES_PASSWORD
           value: mysecretpassword
  replicas: 3
  selector:
    matchLabels:
      tier: front-end
```

Pod Template

apiVersion: apps/v1 this is the api version that supports ReplicaSet

```
kubectl create -f replicaset-definition.yaml
kubectl get replicaset
Kubectl get pods
Kubectl delete replicaset myapp-replicaset
```

selector: enable replicaset to identify the pods to monitor by filtering them by their labels. Example: Filtering by tier label.

Actividade - Create a Replicaset with Yaml

- Create a ReplicationSet named **hello-word-rest-api-rs** based on adfonte/hello-server image. Create 3 replicas. Filter pods by tier label front-end
- Expose **replicationset** on Port 8080 using a LoadBalancer.

```yaml
apiVersion: apps/v1
kind: ReplicaSet
metadata:
     name: hello-world-rest-api-rs
     labels:
       app: hello-world-rest-api
       tier: front-end
spec:
    replicas: 3
    selector:
       matchLabels:
        tier: front-end
    template:
       metadata:
        name: hello-world-rest-api-pod
        labels:
              app: hello-world-rest-api
              tier: front-end
       spec:
            containers:
             - name: hello-world-rest-api
               image: adfonte/hello-server
               ports:
                - containerPort: 8080
```

Create a Replicaset with Yaml

- Result:

```
afs-MacBook-Air:kubernetes alexandrefonte$ kubectl create -f replicationset-definition.yaml
replicaset.apps/hello-world-rest-api-rs created
afs-MacBook-Air:kubernetes alexandrefonte$ kubectl create -f replicationset-definition.yaml
Error from server (AlreadyExists): error when creating "replicationset-definition.yaml": replicasets.apps "hello-world-rest-api-rs" already exists
afs-MacBook-Air:kubernetes alexandrefonte$ kubectl get all
NAME                                 READY   STATUS    RESTARTS   AGE
pod/hello-world-rest-api-rs-5drvn    1/1     Running   0          4m13s
pod/hello-world-rest-api-rs-bzqnk    1/1     Running   0          4m13s
pod/hello-world-rest-api-rs-fgzmj    1/1     Running   0          4m13s

NAME                 TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)   AGE
service/kubernetes   ClusterIP   10.96.0.1    <none>        443/TCP   7d6h

NAME                                       DESIRED   CURRENT   READY   AGE
replicaset.apps/hello-world-rest-api-rs    3         3         3       4m13s
```

```
afs-MacBook-Air:kubernetes alexandrefonte$ kubectl create -f replicationset-definition.yaml
replicaset.apps/hello-world-rest-api-rs created
afs-MacBook-Air:kubernetes alexandrefonte$ kubectl get replicasets
NAME                     DESIRED   CURRENT   READY   AGE
hello-world-rest-api-rs   3         3         3       9s
afs-MacBook-Air:kubernetes alexandrefonte$ kubectl expose replicaset  hello-world-rest-api-rs --port=8080 --type=LoadBalancer --name=hello-world-service-lb
service/hello-world-service-lb exposed
```

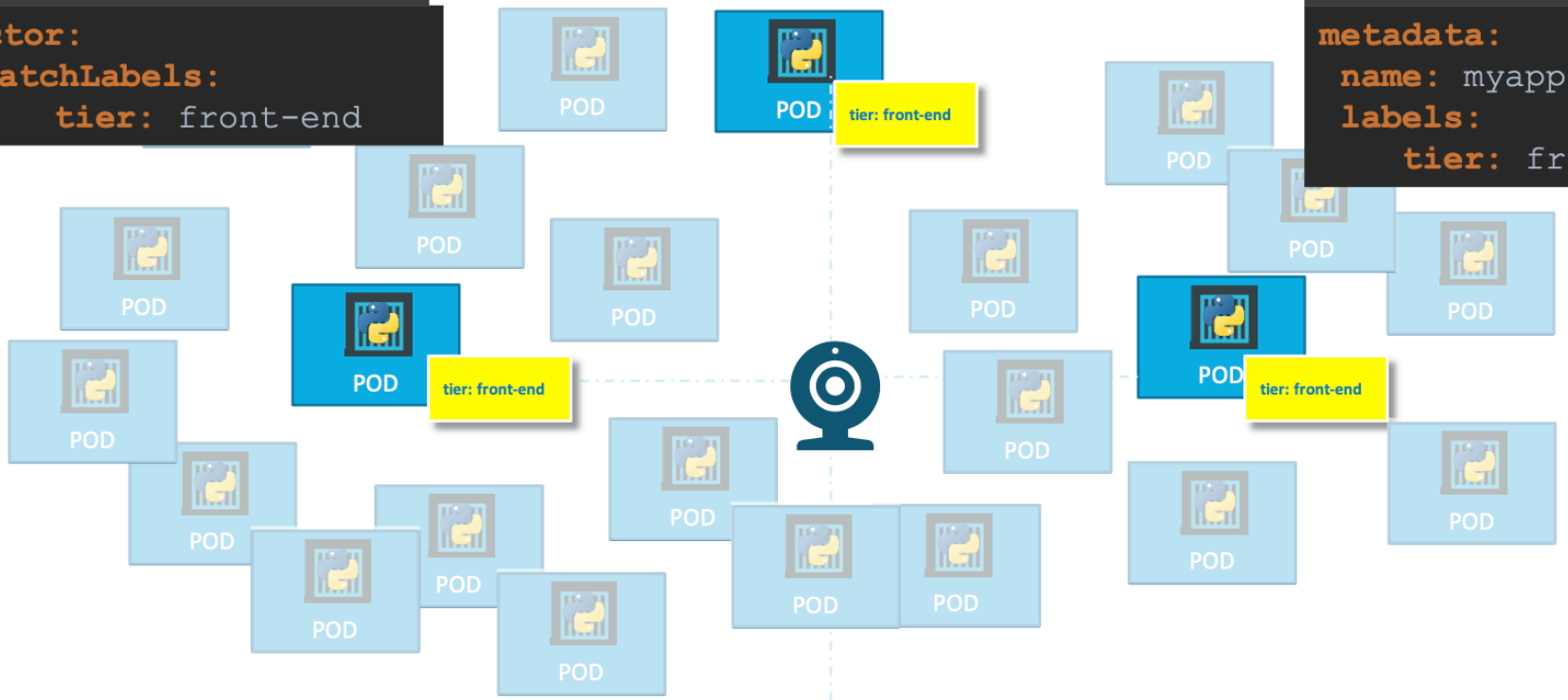
Create a Replicaset with Yaml

replicaset-definition.yml

```
selector:
    matchLabels:
        tier: front-end
```

pod-definition.yml

```
metadata:
  name: myapp-pod
  labels:
    tier: front-end
```



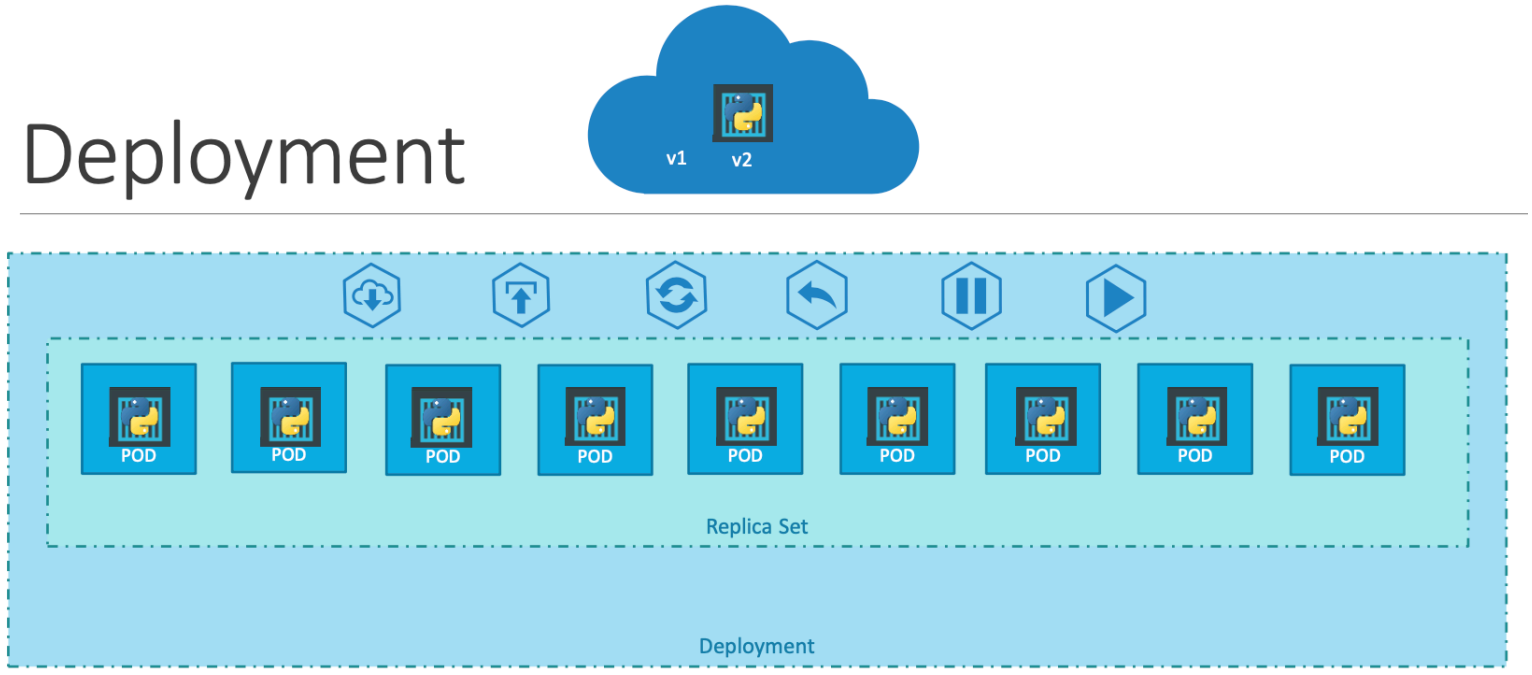
How to Scale

- **Way 1 - By editing the replicaset-definition.yaml**
  - -Update in the file the section **replicas: 6**
  - **-Kubectl replace** -f replicaset-definition.yaml

- **Way 2 - By using the kubectl scale command**
  - **-Kubectl scale** --replicas=6 -f replicaset-definition.yaml
  - -Nota: Mas não atualiza o conteúdo do ficheiro.
  - -OR
  - **-Kubectl scale** --replicas=6 replicaset.apps/*hello-world-rest-api-rs*
- **Way 3 - Automatically based on load (autoscale command)**

Kubernetes Deployments

- PODs deploy single instances of our application such as the web application. Each container of our is encapsulated in PODs.
- Multiple such PODs are deployed using Replication Controllers or Replica Sets
- Deployment is a kubernetes object that comes higher in the hierarchy.
- The deployment provides us with capabilities to upgrade the underlying instances seamlessly using rolling updates, undo changes (in case of unexpected error during an update), and pause and resume changes to deployments



Create a Deployment with Yaml

- deployment-definition.yaml

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: myapp-deployment
  labels:
    app: myapp
    tier: front-end
spec:
  template:
    metadata:
      name: mysql:5.7
      labels:
        app: myapp
        tier: front-end
    spec:
      containers:
      - name: nginx-container
        image: nginx:1.7.0
        ports:
         - containerPort: 80
      - name: postgres
        image: postgres
        env:
         - name: POSTGRES_PASSWORD
           value: mysecretpassword
  replicas: 3
  selector:
    matchLabels:
      tier: front-end
```

kind: deployment.
The rest is identically to a replicaset

```
kubectl create -f deployment-definition.yaml
kubectl get deployments
kubectl get replicasets
Kubectl get pods
```

Atividade - Create a Deployment with Yaml

- deployment-definition.yaml

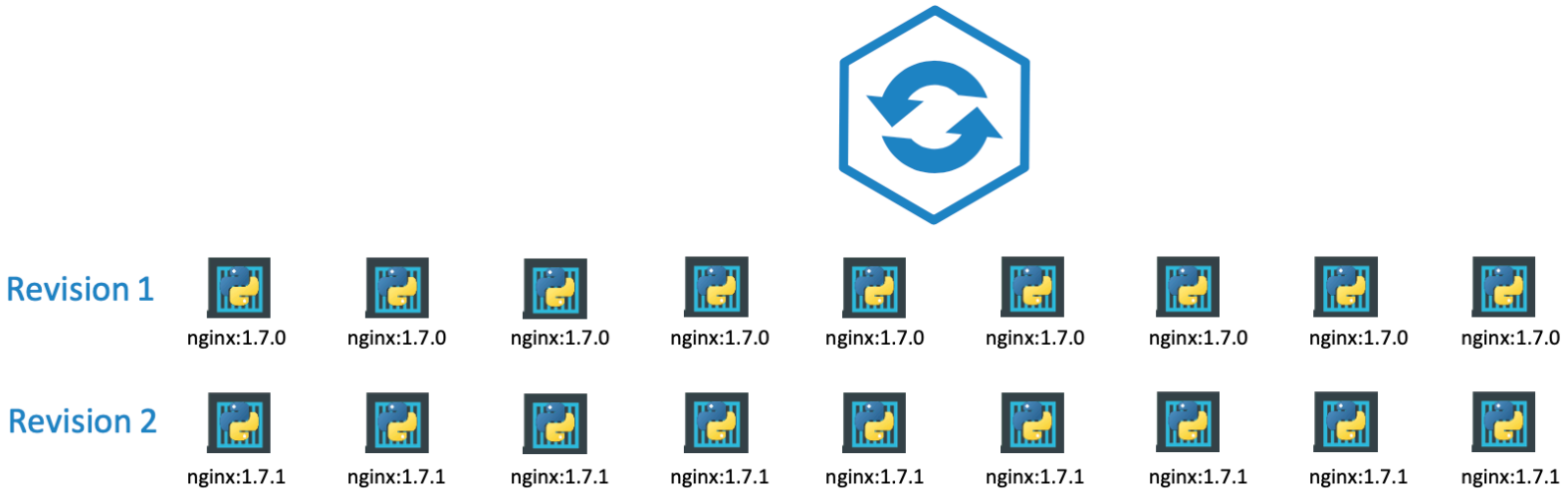
- Create a Deployment named **hello-word-rest-api-deployment** based previous replicaset. Define below replicas section a label with the strategy type: RollingUpdate
- Expose **deployment** on Port 8080 using a LoadBalancer.

```
afs-MacBook-Air:kubernetes alexandrefonte$ kubectl create -f deployment-definition.yaml
deployment.apps/hello-world-rest-api-deployment created
afs-MacBook-Air:kubernetes alexandrefonte$ kubectl get all
NAME                                                  READY   STATUS              RESTARTS   AGE
pod/hello-world-rest-api-deployment-7d645f687f-9fr2q   0/1     ContainerCreating   0          4s
pod/hello-world-rest-api-deployment-7d645f687f-jb7tz   0/1     ContainerCreating   0          4s
pod/hello-world-rest-api-deployment-7d645f687f-nrsxr   1/1     Running             0          4s

NAME                 TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)   AGE
service/kubernetes   ClusterIP   10.96.0.1    <none>        443/TCP   7d6h

NAME                                             READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/hello-world-rest-api-deployment   1/3     3            1           7s

NAME                                                        DESIRED   CURRENT   READY   AGE
replicaset.apps/hello-world-rest-api-deployment-7d645f687f   3         3         1       6s
```

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
     name: hello-world-rest-api-deployment
     labels:
       app: hello-world-rest-api
       tier: front-end
spec:
    replicas: 3
     strategy:
        type: RollingUpdate
    selector:
       matchLabels:
        tier: front-end
    template:
       metadata:
        name: hello-world-rest-api-pod
        labels:
              app: hello-world-rest-api
              tier: front-end
       spec:
            containers:
             - name: hello-world-rest-api
               image: adfonte/hello-server
               ports:
                - containerPort: 8080
```

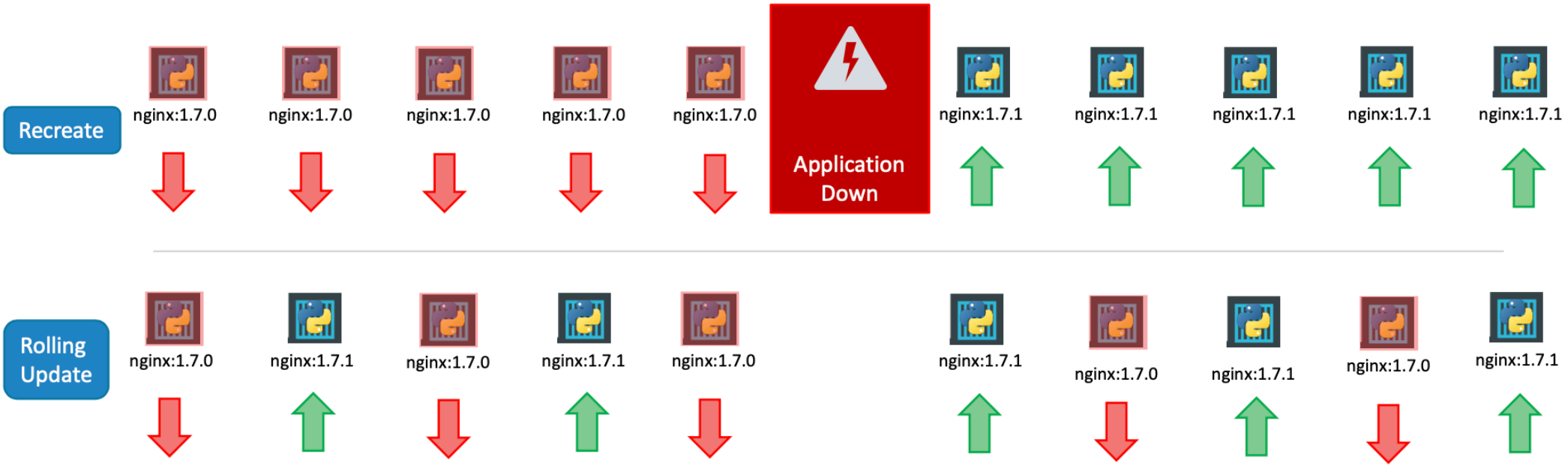
Rollout and Versioning

- A rollout is the process of gradually deploying or upgrading your application containers.
- When you first create a deployment, it triggers a rollout. A new rollout creates a new Deployment revision. Let's call it revision 1.
- In the future when the application is upgraded – meaning when the container version is updated to a new one – a new rollout is triggered and a new deployment revision is created named Revision 2.
- This helps us keep track of the changes made to our deployment and enables us to rollback to a previous version of deployment if necessary.



Types of Deployment Strategies

- **Recreate** - First all instances are destroyed, a then are deployed all new instances of the new application version
- **Rolling Update (Default)**- The instances are destroyed and then deployed new instances, one by one.



How to update a Deployment

- deployment-definition.yml

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: myapp-deployment
  labels:
    app: myapp
    type: front-end
spec:
  template:
    metadata:
      name: mysql:5.7
      labels:
        app: myapp
        type: front-end
    spec:
      containers:
      - name: nginx-container
        image: nginx:latest
        ports:
         - containerPort: 80
      - name: postgres
        image: postgres
        env:
         - name: POSTGRES_PASSWORD
           value: mysecretpassword
  replicas: 3
  selector:
    matchLabels:
      Tier: front-end
```

kind: deployment.
The rest is identically to a replicaset

Update with:

```
kubectl apply -f deployment-definition.yml
```

Or

Warning: The deployment-definition File must be then updated with the new configuration

```
kubectl set image deployment/myapp-deployment nginx=nginx:latest
```

To check status or history:

```
Kubectl describe deployment deployment/myapp-deployment

Kubectl rollout status deployment deployment/myapp-deployment

Kubectl rollout history deployment deployment/myapp-deployment
```

If something goes wrong, we can rollback to previous version:

```
Kubectl rollout undo deployment/myapp-deployment
```

How to update a Deployment

- Update the container image to **adfonte/hello-server:newversion**.

  -First create the new image using the docker tag command based on adfonte/hello-server:latest

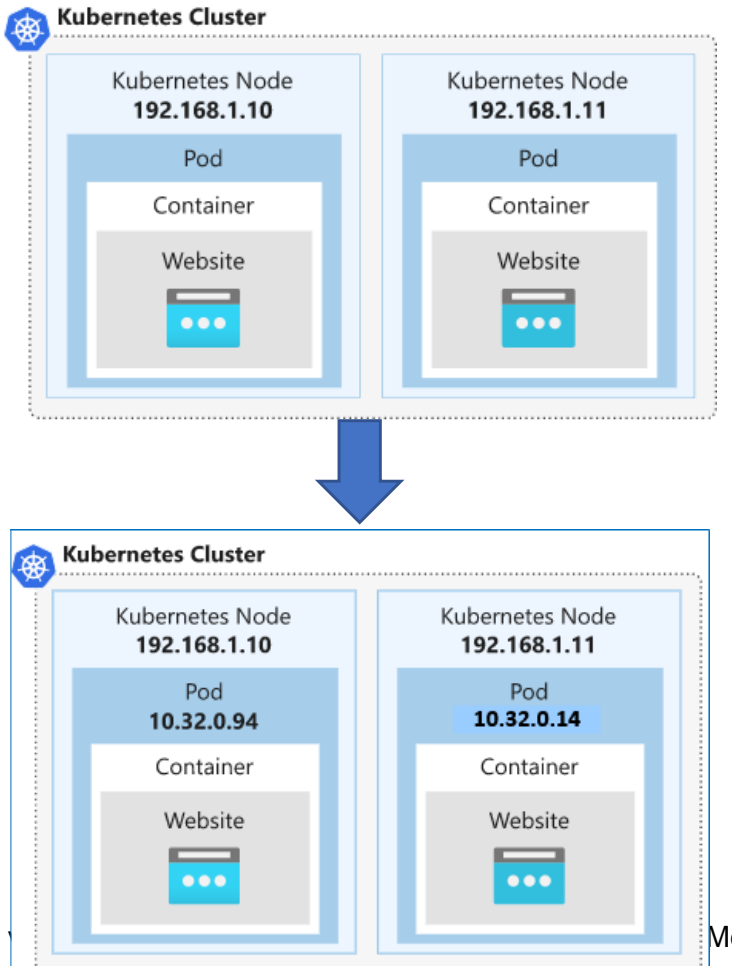
```
afs-MacBook-Air:kubernetes alexandrefonte$ docker tag adfonte/hello-server:latest adfonte/hello-server:version
afs-MacBook-Air:kubernetes alexandrefonte$ docker image ls -a
REPOSITORY              TAG          IMAGE ID       CREATED         SIZE
adfonte/hello-server    latest       d09ee31d0506   29 hours ago    544MB
adfonte/hello-server    newversion   d09ee31d0506   29 hours ago    544MB
adfonte/hello-server    version      d09ee31d0506   29 hours ago    544MB
product-server          latest       93310c05cbd6   6 days ago      491MB
message-server          latest       60ce7822e7c8   6 days ago      544MB
httpd                   latest       c30a46771695   9 days ago      144MB
kicbase/stable          v0.0.30      1312ccd2422d   2 months ago    1.14GB
```

  -Update the deployment

Result:

```
afs-MacBook-Air:kubernetes alexandrefonte$ kubectl set image deployment/hello-world-rest-api-deployment hello-world-rest-api=hello-world:newversion
deployment.apps/hello-world-rest-api-deployment image updated
```

Kubernetes networking

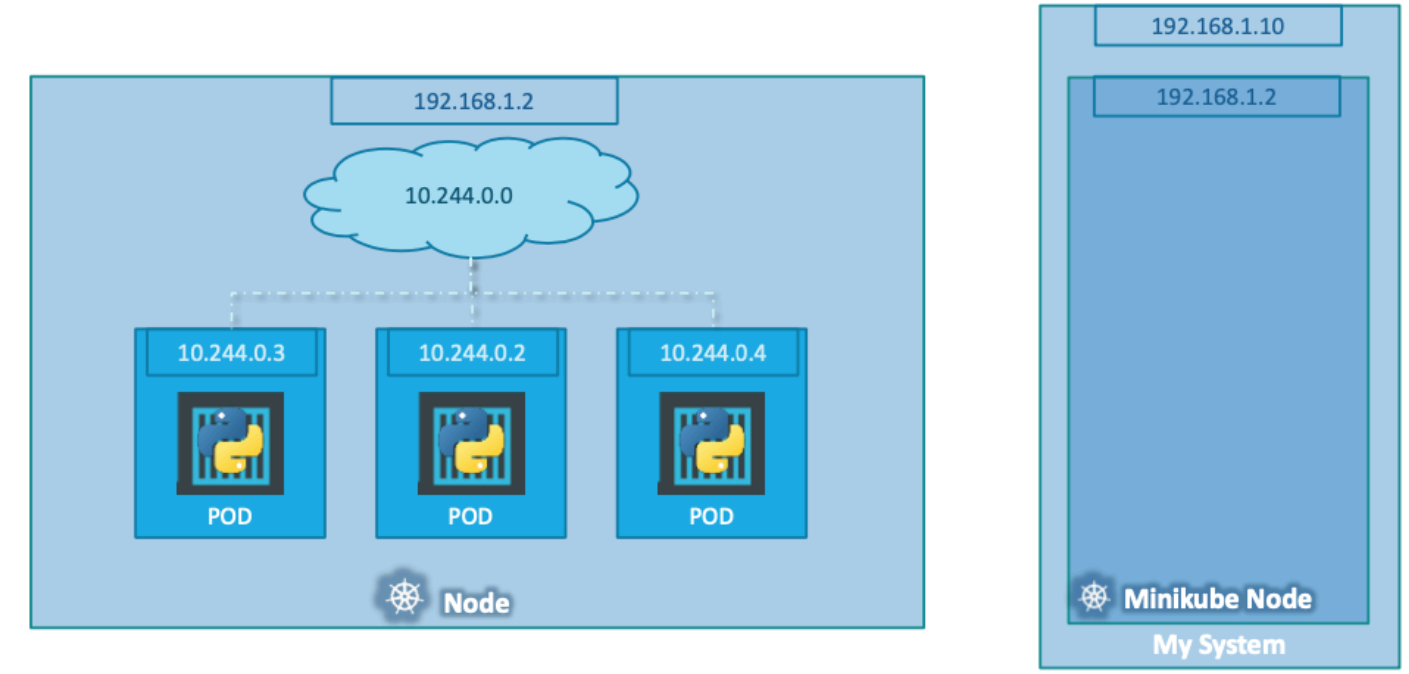
- Os nós têm endereços IP da rede onde estão ligados. Este endereço IP permite-nos o acesso ao nó via rest ou SSH.
- A cada pod é atribuído um endereço IP dinâmico (interno) de uma rede virtual separada da rede dos nós criada para ligar os pods
- 

Although each Pod has a unique IP address, those IPs are not exposed outside the cluster without a Service.

Services allow your applications to receive traffic. Services can be exposed in different ways by specifying a `type` in the ServiceSpec

Kubernetes networking with Minikube

- Host: 192.168.1.10
- Minikube Kubernetes node: 192.168.1.2 (virtualization)
- Pods: 10.244.0.(2-4)



Service Types

- Although each Pod has a unique IP address, those IPs are not exposed outside the cluster without a Service. Services allow your applications to receive traffic. Services can be exposed in different ways by specifying a type in the ServiceSpec:

  -ClusterIP (default) - Exposes the Service on an internal IP in the cluster. This type makes the Service only reachable from within the cluster.

  -NodePort - Exposes the Service on the same port of each selected Node in the cluster using NAT. Makes a Service accessible from outside the cluster using <NodeIP>:<NodePort>. Superset of ClusterIP.

  -LoadBalancer - Creates an external load balancer in the current cloud (if supported) and assigns a fixed, external IP to the Service. Superset of NodePort.

  -ExternalName - Maps the Service to the contents of the externalName field (e.g. foo.bar.example.com), by returning a CNAME record with its value. No proxying of any kind is set up. This type requires v1.7 or higher of kube-dns, or CoreDNS version 0.0.8 or higher.

Service Types - Cluster IP

- Service is like a virtual service inside the node …

  -ClusterIP: Exposes a service which is only accessible from within the cluster.

service-definition.yml

```yaml
apiVersion: v1
kind: Service
metadata:
    name: back-end

spec:
    type: ClusterIP
    ports:
     - targetPort: 80
       port: 80

    selector:
      app: myapp
      type: back-end
```

pod-definition.yml

```yaml
spec:
  containers:
  - name: nginx-container
    image: nginx
```

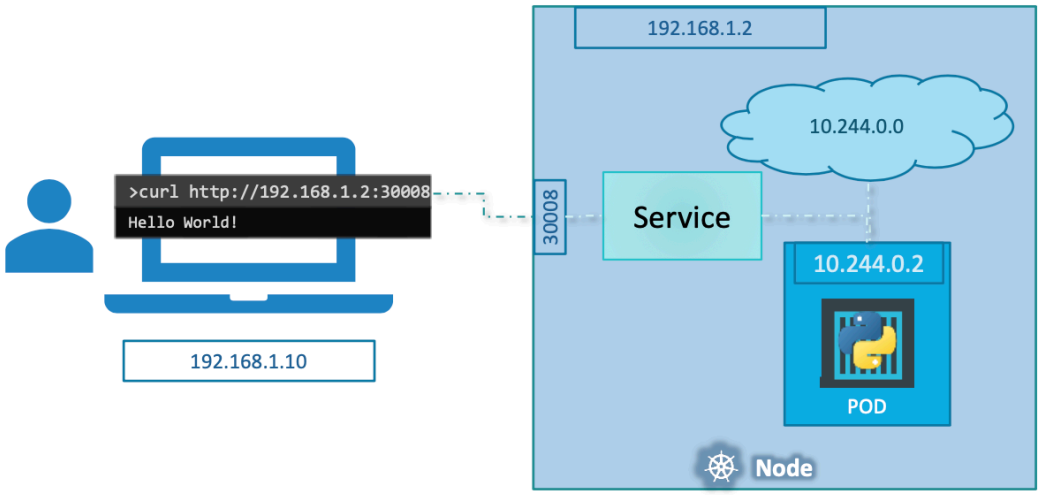
```
> kubectl create -f service-definition.yml
service "back-end" created

> kubectl get services
NAME         TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)    AGE
kubernetes   ClusterIP   10.96.0.1        <none>        443/TCP    16d
back-end     ClusterIP   10.106.127.123   <none>        80/TCP     2m
```

NodePort vs LoadBalancer

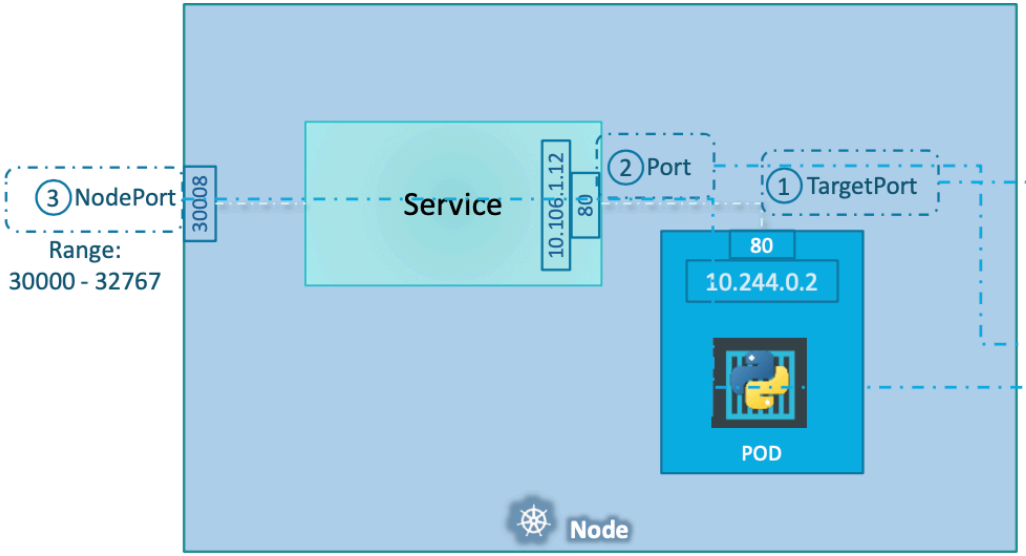
- A **NodePort** service is the most basic way to get external traffic directly to your service. NodePort, as the name implies, opens a specific port, and any traffic that is sent to this port is forwarded to the service.
- A **LoadBalancer** service is the standard way to expose a service to the internet. With this method, each service gets its own IP address.
- Using minikube tunnel
  - Services of type LoadBalancer can be exposed via the minikube tunnel command. It must be run in a separate terminal window to keep the LoadBalancer running. Ctrl-C in the terminal can be used to terminate the process at which time the network routes will be cleaned up.

Service Types - NodePort (External Communication)

- Service is like a virtual service inside the node …
  - **NodePort:** makes a internal pod accessible on a Port of the Node



There are
3 ports involved.
- Pod port (targetPort) where the container is running.
- Port of service itself
- Port of the node (range 30000-32767)



service-definition.yml

```yaml
apiVersion: v1
kind: Service
metadata:
    name: myapp-service

spec:
    type: NodePort
    ports:
      - targetPort: 80
        port: 80
        nodePort: 30008
```

Service Types - NodePort (External Communication)

- Exemplo

```
apiVersion: apps/v1
kind: Deployment
metadata:
    name: my-nginx-deploy
    labels:
        app: nginx
spec:
    template:
        metadata:
            labels:
                app: nginx
        spec:
            containers:
            - name: test-nginx
              image: nginx:alpine
              ports:
              - containerPort: 80
    replicas: 2
    selector:
        matchLabels:
            app: nginx
```

```
kubectl apply -f nginx-deploy.yml
```

```
apiVersion: v1
kind: Service
metadata:
  name: my-service
spec:
  type: NodePort
  selector:
    app: nginx
  ports:
    - port: 80
      targetPort: 80
      nodePort: 30007
```

```
kubectl apply -f nodeport.yml
```

Service Types - NodePort (External Communication)

- Create a service named **my-service** of NodePort type
- Create a port-forward from host computer port 7080 to 8080

nodeport.yaml

```yaml
apiVersion: v1
kind: Service
metadata:
  name: my-service
spec:
  type: NodePort
  selector:
    app: hello-world-rest-api
  ports:
    - port: 8080
      targetPort: 8080
      nodePort: 30000
```

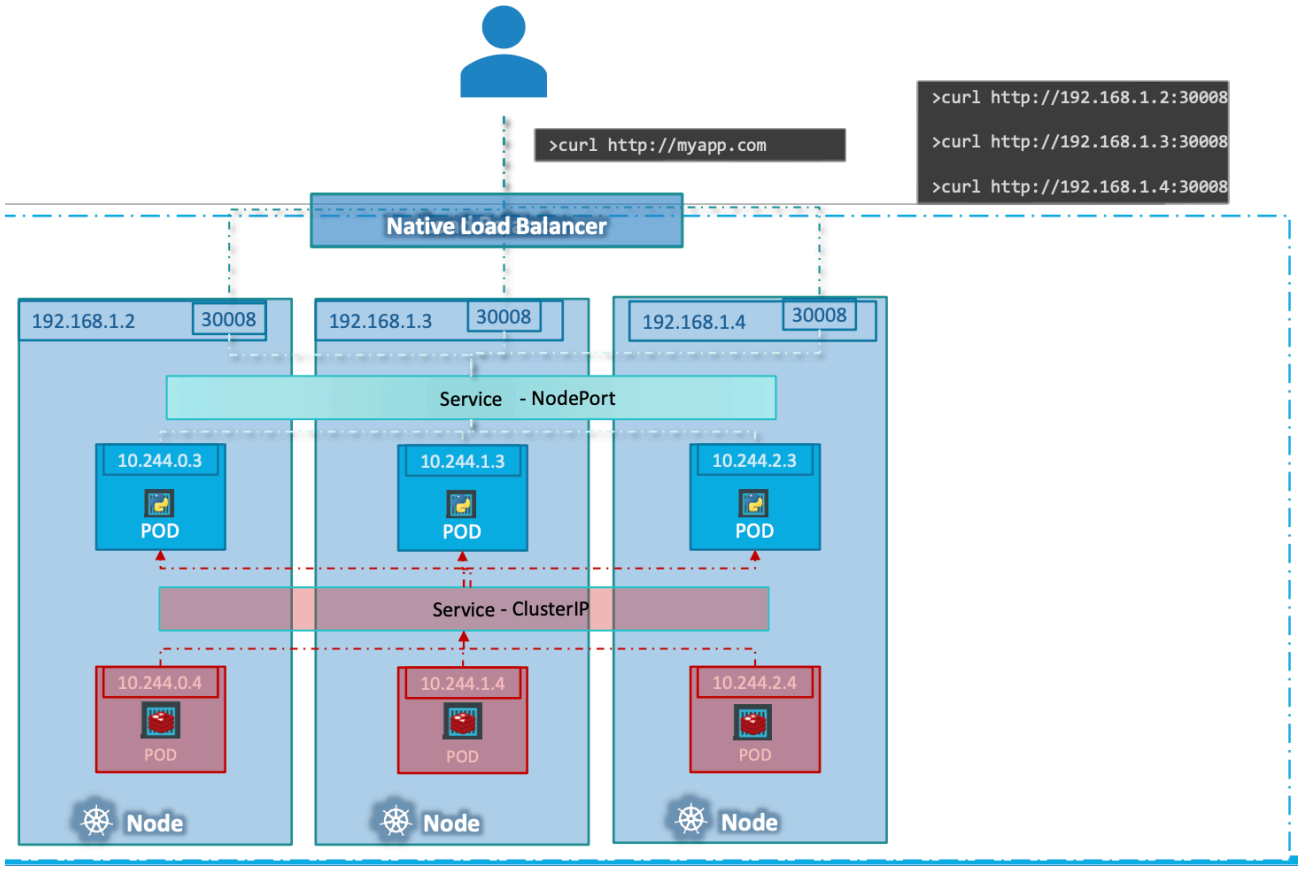
```
afs-MacBook-Air:kubernetes alexandrefonte$ kubectl apply -f nodeport.yaml
The Service "my-service" is invalid: spec.ports[0].nodePort: Invalid value: 7080: provided port is
not in the valid range. The range of valid ports is 30000-32767
afs-MacBook-Air:kubernetes alexandrefonte$ kubectl apply -f nodeport.yaml
service/my-service created
afs-MacBook-Air:kubernetes alexandrefonte$ kubectl get all
NAME                                                READY   STATUS    RESTARTS   AGE
pod/hello-world-rest-api-deployment-7d645f687f-bsq67   1/1     Running   0          3m23s
pod/hello-world-rest-api-deployment-7d645f687f-dwkwg   1/1     Running   0          3m23s
pod/hello-world-rest-api-deployment-7d645f687f-twsss   1/1     Running   0          3m23s

NAME                 TYPE        CLUSTER-IP    EXTERNAL-IP   PORT(S)          AGE
service/kubernetes   ClusterIP   10.96.0.1     <none>        443/TCP          8d
service/my-service   NodePort    10.110.2.30   <none>        8080:30000/TCP   5s

NAME                                           READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/hello-world-rest-api-deployment   3/3     3            3           3m24s

NAME                                                      DESIRED   CURRENT   READY   AGE
replicaset.apps/hello-world-rest-api-deployment-7d645f687f   3         3         3       3m24s
```

```
afs-MacBook-Air:kubernetes alexandrefonte$ kubectl port-forward deployment/hello-world-rest-api-deployment 7080:8080
Forwarding from 127.0.0.1:7080 -> 8080
Forwarding from [::1]:7080 -> 8080
```

In a browser enter:
Localhost:7080/messages/students/aid

Service Types - LoadBalancer (External Communication)

- Service is like a virtual service inside the node …
  - **LoadBalancer** service type is built on top of NodePort service types by provisioning and configuring external load balancers. It exposes services that are running in the cluster by forwarding layer 4 traffic to worker nodes.



Service Types - LoadBalancer (External Communication)

- Service is like a virtual service inside the node …
  - **LoadBalancer** (Cont.)

service-definition.yml

```yaml
apiVersion: v1
kind: Service
metadata:
    name: front-end

spec:
    type: LoadBalancer
    ports:
     - targetPort: 80
       port: 80

    selector:
        app: myapp
        type: front-end
```

```
> kubectl create -f service-definition.yml
service "front-end" created
```

```
> kubectl get services
NAME          TYPE          CLUSTER-IP       EXTERNAL-IP   PORT(S)   AGE
kubernetes    ClusterIP     10.96.0.1        <none>        443/TCP   16d
front-end     LoaBalancer   10.106.127.123   <Pending>     80/TCP    2m
```

Service Types - LoadBalancer (External Communication)

- Create a service named **my-service-lb** of LoadBalancer type

  -nodePort: 30001

```
afs-MacBook-Air:kubernetes alexandrefonte$ kubectl get service
NAME          TYPE        CLUSTER-IP     EXTERNAL-IP   PORT(S)          AGE
kubernetes    ClusterIP   10.96.0.1      <none>        443/TCP          8d
my-service    NodePort    10.110.2.30    <none>        8080:30000/TCP   22m
afs-MacBook-Air:kubernetes alexandrefonte$ kubectl apply -f loadbalancer.yaml
service/my-service-lb created
afs-MacBook-Air:kubernetes alexandrefonte$ kubectl get service
NAME            TYPE           CLUSTER-IP       EXTERNAL-IP   PORT(S)          AGE
kubernetes      ClusterIP      10.96.0.1        <none>        443/TCP          8d
my-service      NodePort       10.110.2.30      <none>        8080:30000/TCP   22m
my-service-lb   LoadBalancer   10.102.128.100   <pending>     8080:30001/TCP   2s
```

```
afs-MacBook-Air:kubernetes alexandrefonte$ minikube tunnel
✅  Tunnel successfully started

📌  NOTE: Please do not close this terminal as this process must stay alive for the tunnel to be accessible ...

🏃  Starting tunnel for service my-service-lb.
```

```
afs-MacBook-Air:kubernetes alexandrefonte$ kubectl get service
NAME            TYPE           CLUSTER-IP       EXTERNAL-IP   PORT(S)          AGE
kubernetes      ClusterIP      10.96.0.1        <none>        443/TCP          8d
my-service      NodePort       10.110.2.30      <none>        8080:30000/TCP   24m
my-service-lb   LoadBalancer   10.102.128.100   127.0.0.1     8080:30001/TCP   103s
afs-MacBook-Air:kubernetes alexandrefonte$
```

loadbalancer.yaml

```yaml
apiVersion: v1
kind: Service
metadata:
  name: my-service-lb
spec:
  type: LoadBalancer
  selector:
    app: hello-world-rest-api
  ports:
# Port accessible inside cluster
    - port: 8080
# Port to forward to inside the pod
      targetPort: 8080
# Port accessible outside cluster
      nodePort: 30001
```

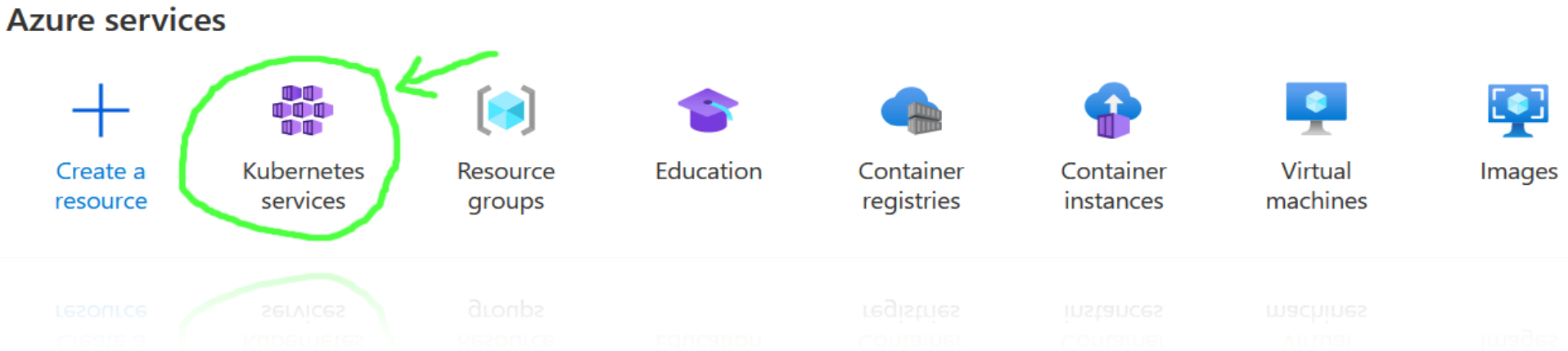
In a different terminal window

In case of Minikube running as a VM, appears a different IP

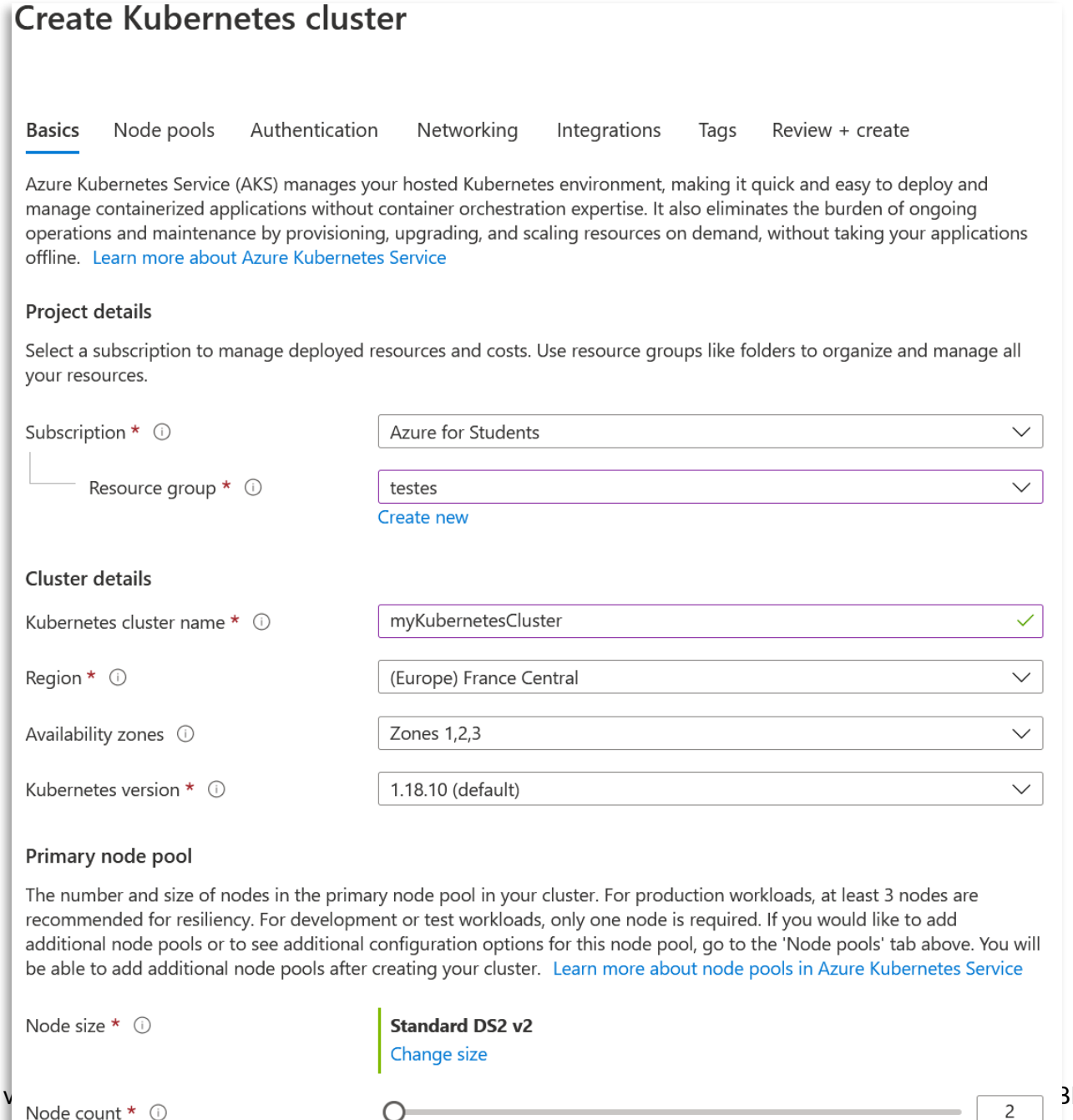
In a browser enter: 127.0.0.1:8080/messages/students/aid

Azure Kubernetes Service (AKS)

- O serviço AKS providencia um ambiente kubernetes integrado na cloud Azure
- Desta forma evita-se todo o trabalho adicional associado à implementação de kubernetes em ambientes físicos
- O control plane (master node) é gratuito, só se paga as VM onde vão ficar os outros nós (worker nodes), storage e networking

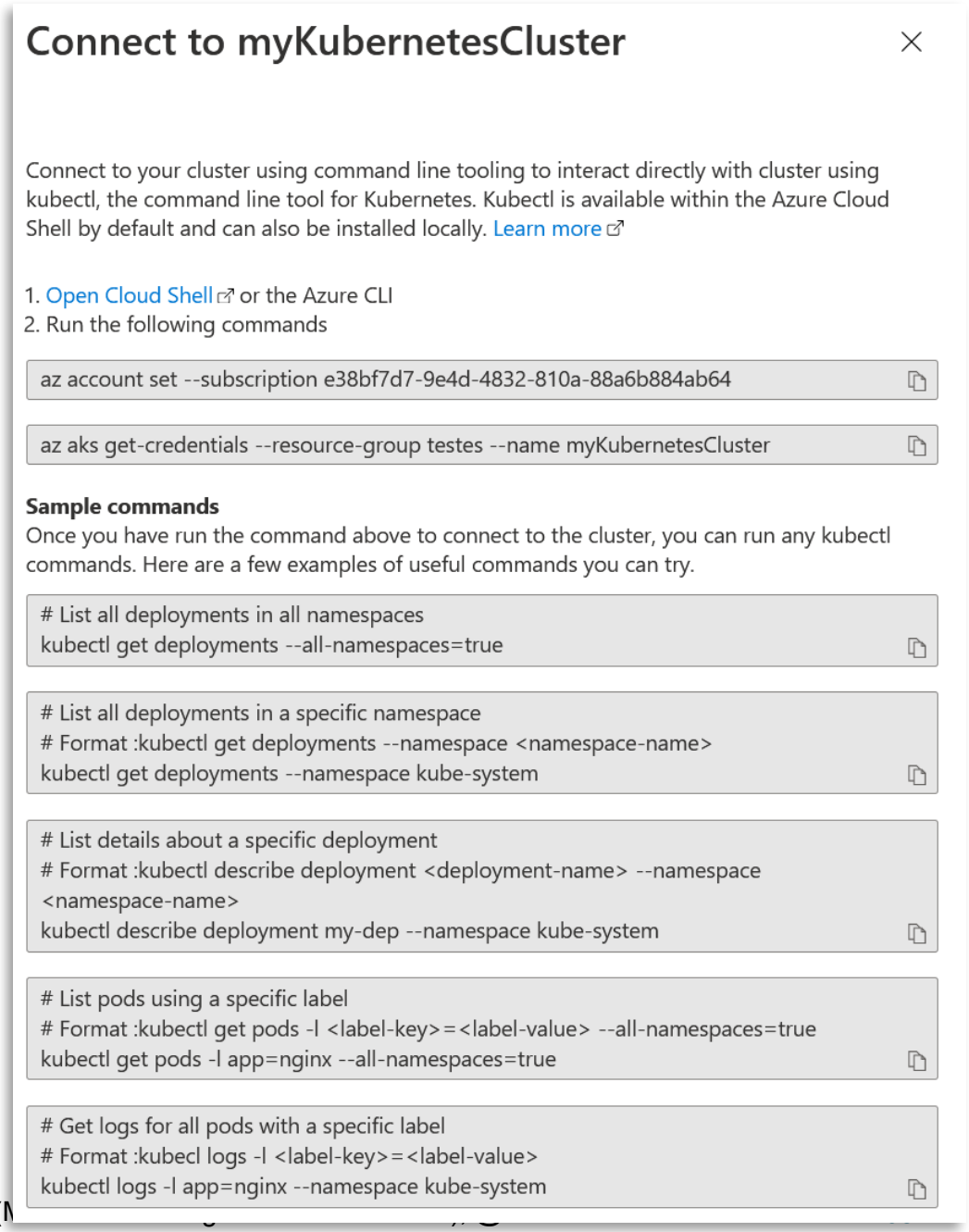


Criação de um cluster K8s no AKS



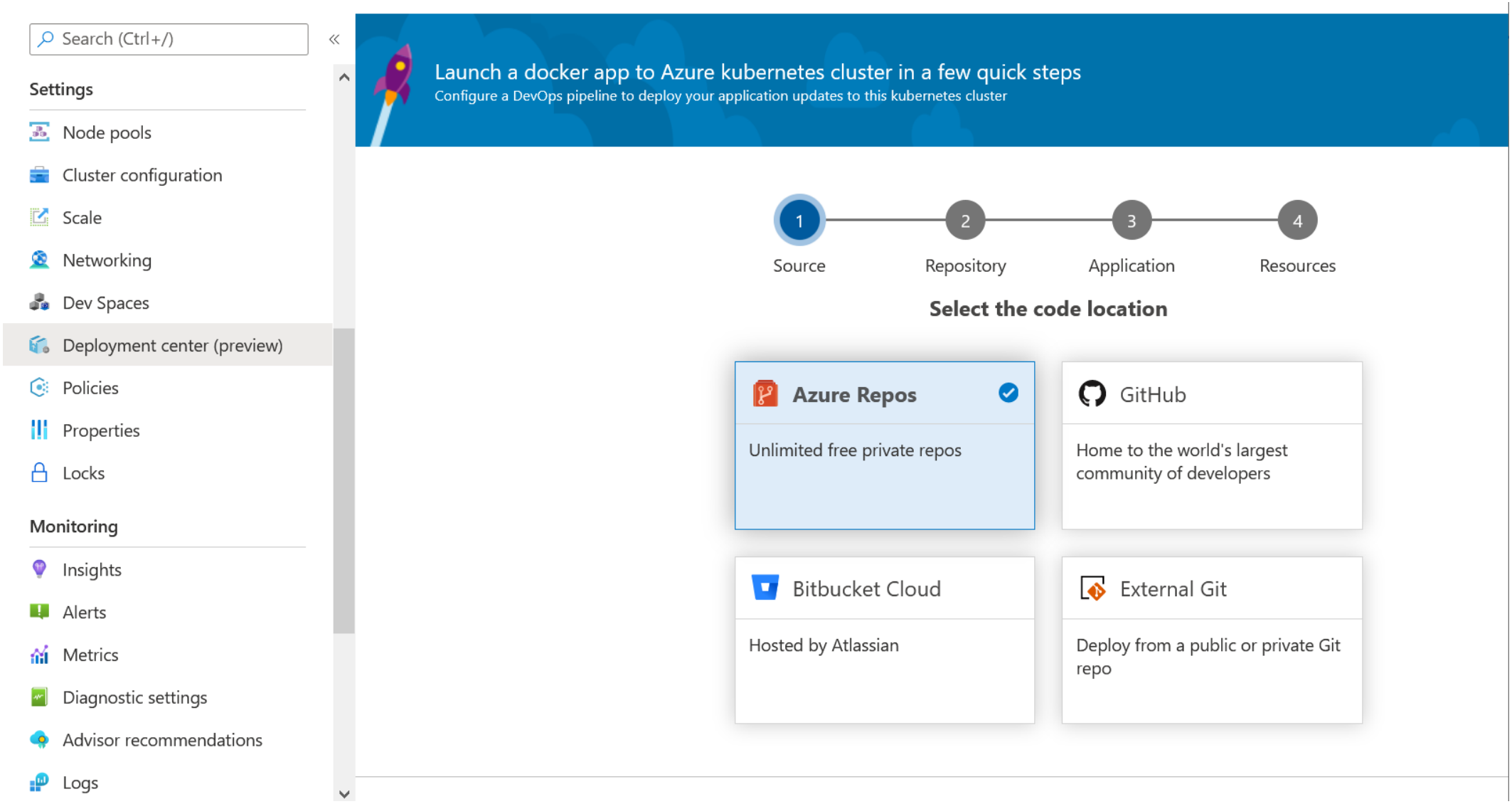
Acesso ao cluster criado por AKS

- A interação com o cluster criado no AKS pode ser feita da forma habitual - usando o comando CLI kubectl
- O kubectl já está integrado no Azure Cloud Shell
- Mas também pode ser instalado localmente

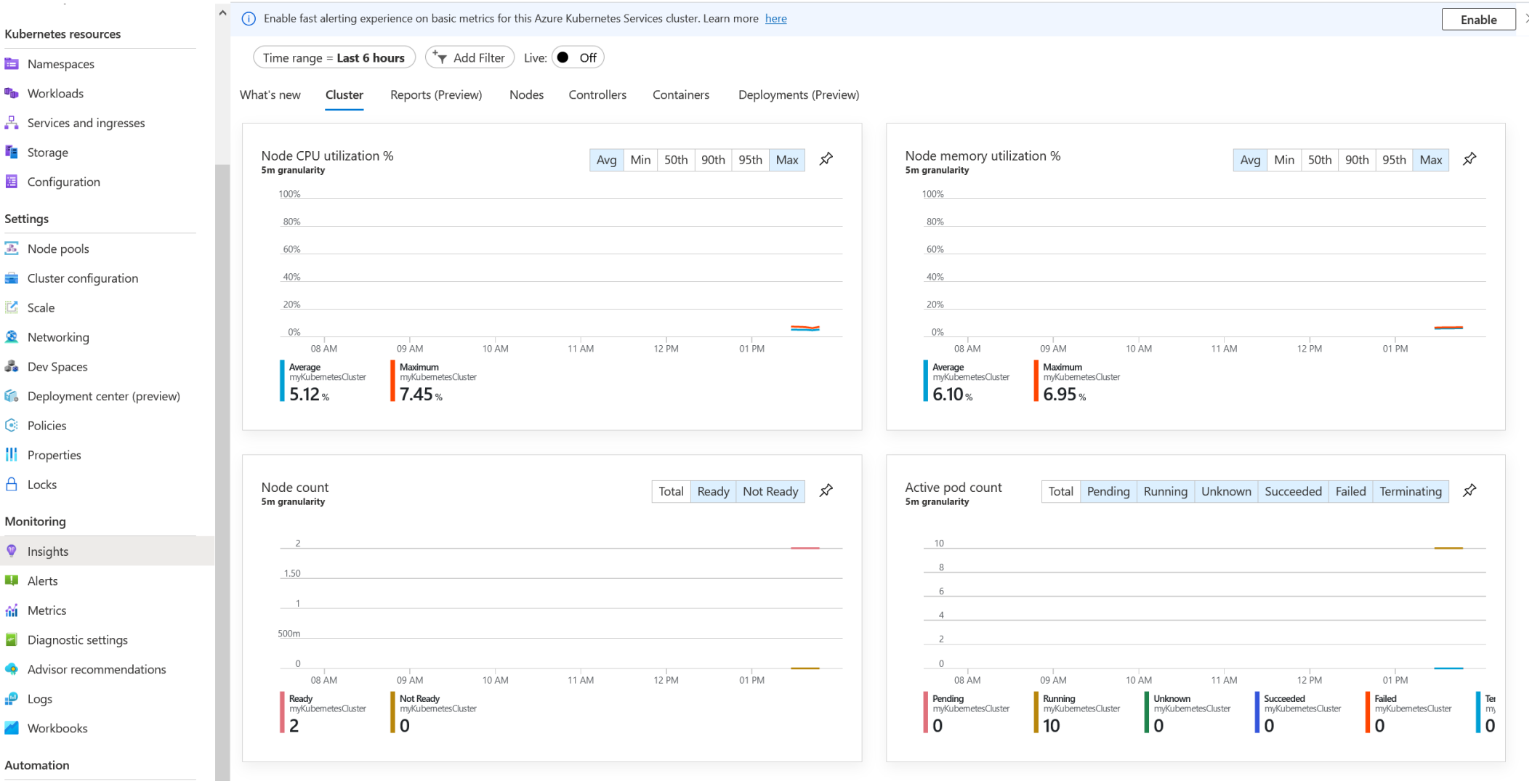


Também pode ser usado o portal Azure...

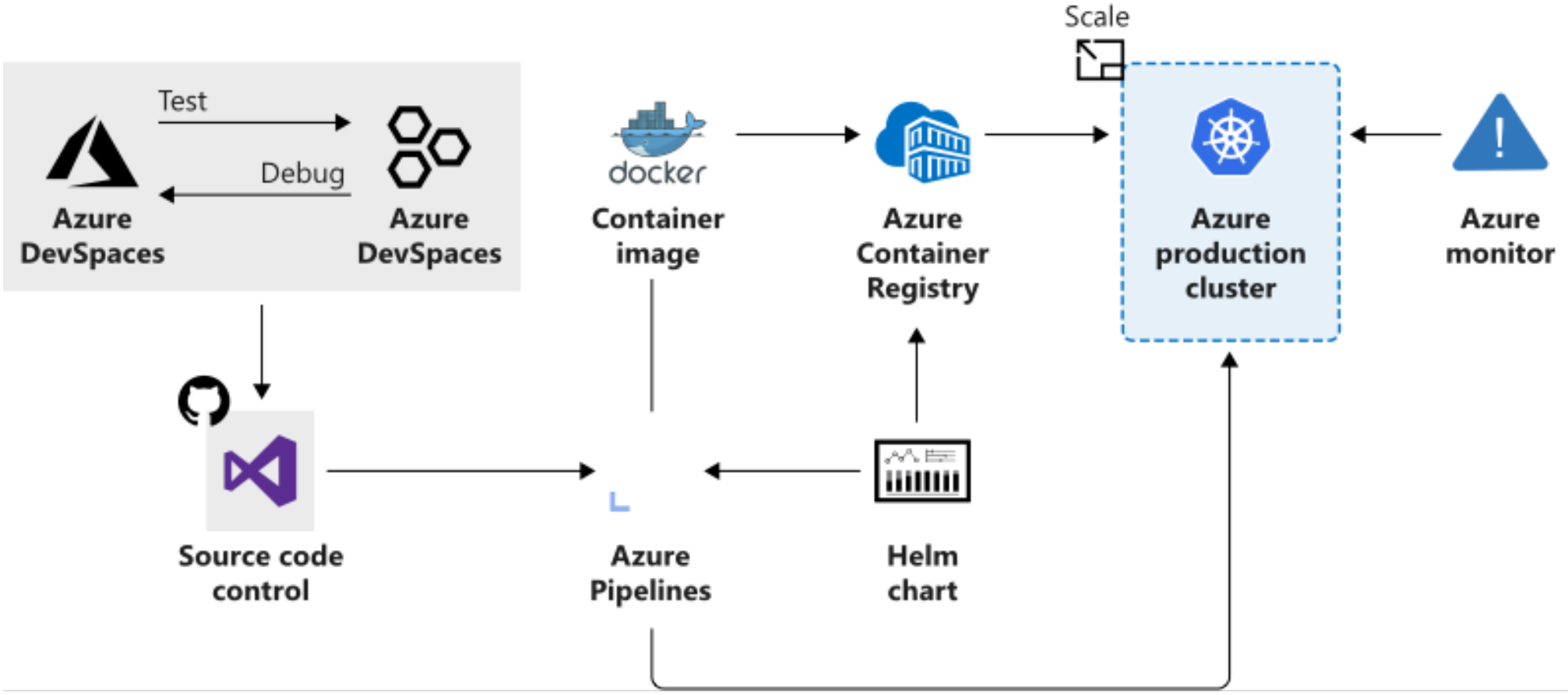
- Nesta imagem pode ser visto o deployment center



Monitorização do cluster kubernetes



Desenvolvimento e implementação de workloads



Questões