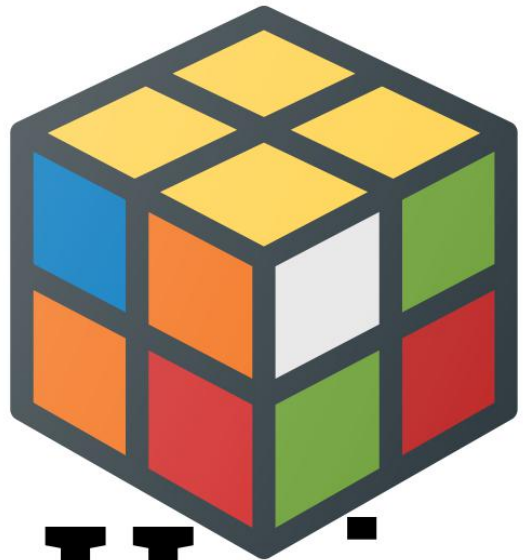


Geek University

Evolua seu lado geek!

www.geekuniversity.com.br

Conhecendo o Replication Controller e ReplicaSets

Conhecendo o Replication Controller e ReplicaSets

Quando estudamos sobre a arquiterua e componentes do Kubernetes vimos que o Controller Manager é o “cérebro” por traz de toda automação do Kubernetes.

Este componente monitora os objetos presentes no cluster e age de acordo com as necessidades.

Uma das ferramentas que faz parte do Controller Manager é o Replication Controller.

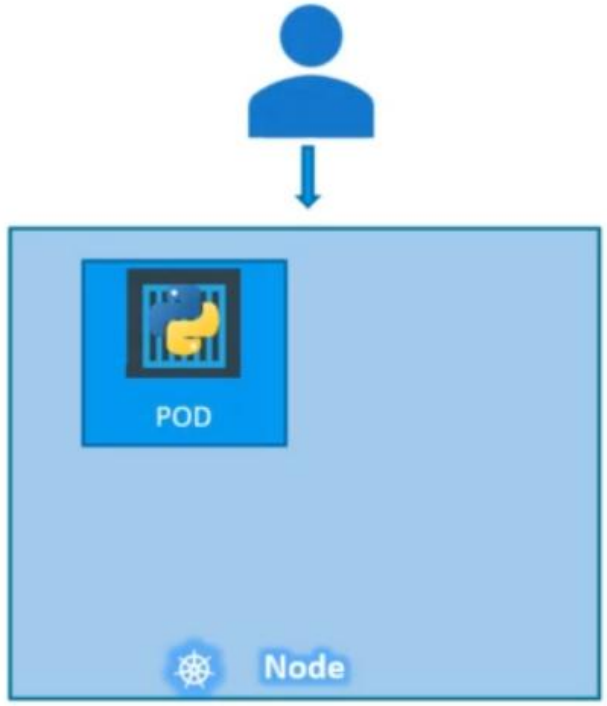
Mas o que seria uma Replica? E por quê precisamos do Replication Controller?

Conhecendo o Replication Controller e ReplicaSets

Replication Controller

Para entender o que é uma replica vamos imaginar o seguinte cenário: Imagine que você tenha desenvolvido uma aplicação e publicou ela em um cluster contendo 1 pod/container.

Os usuários começam a acessar sua aplicação.



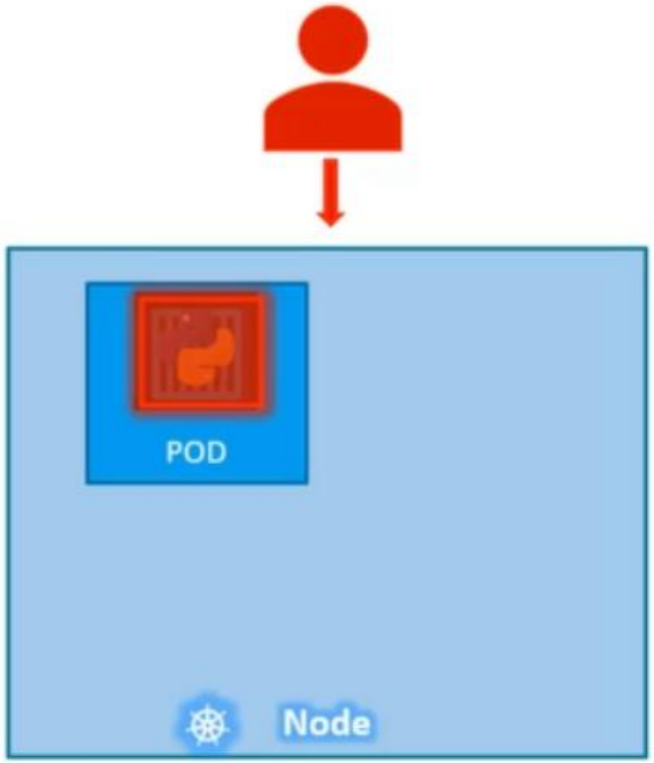
Conhecendo o Replication Controller e ReplicaSets

Replication Controller

Para entender o que é uma replica vamos imaginar o seguinte cenário: Imagine que você tenha desenvolvido uma aplicação e publicou ela em um cluster contendo 1 pod/container.

Os usuários começam a acessar sua aplicação.

Mas então o pod/container falha e seus usuários ficam impedidos de acessar a aplicação.



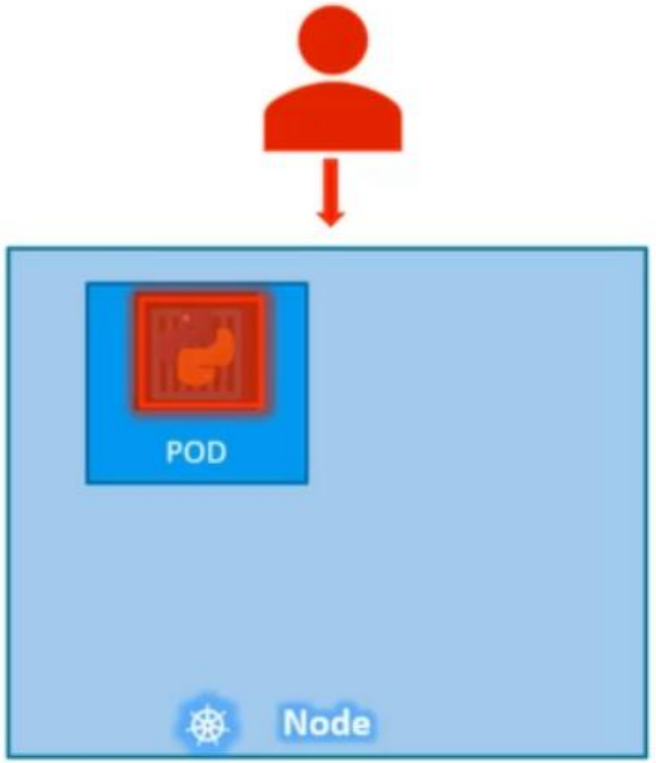
Conhecendo o Replication Controller e ReplicaSets

Replication Controller

Para entender o que é uma replica vamos imaginar o seguinte cenário: Imagine que você tenha desenvolvido uma aplicação e publicou ela em um cluster contendo 1 pod/container.

Os usuários começam a acessar sua aplicação.

Mas então o pod/container falha e seus usuários ficam impedidos de acessar a aplicação.



Para evitar que nossos usuários/clientes tenham seu acesso à nossa aplicação prejudicado é ideal que tenhamos mais de 1 instância do pod/container sendo executada ao mesmo tempo, pois desta forma se um deles parar o outro continua o serviço.

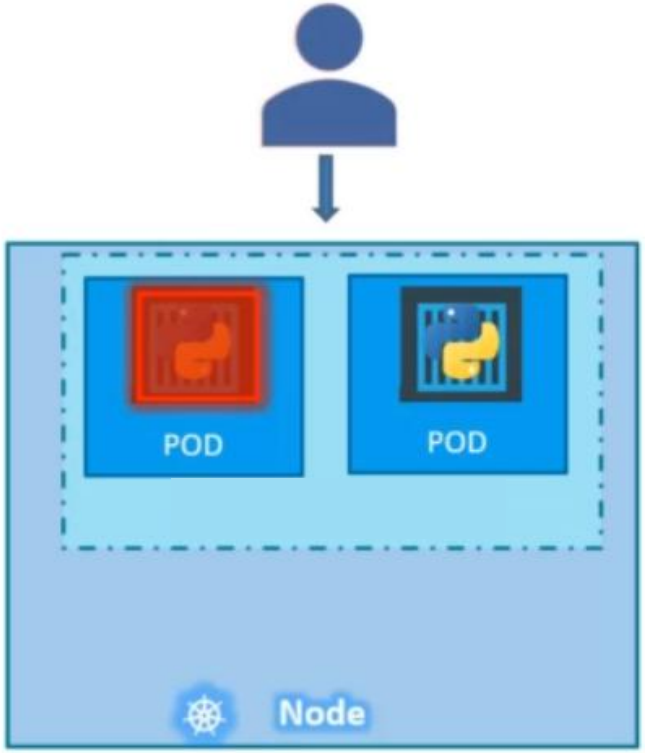
Conhecendo o Replication Controller e ReplicaSets

Replication Controller

Para entender o que é uma replica vamos imaginar o seguinte cenário: Imagine que você tenha desenvolvido uma aplicação e publicou ela em um cluster contendo 1 pod/container.

Os usuários começam a acessar sua aplicação.

Mas então o pod/container falha e seus usuários ficam impedidos de acessar a aplicação.



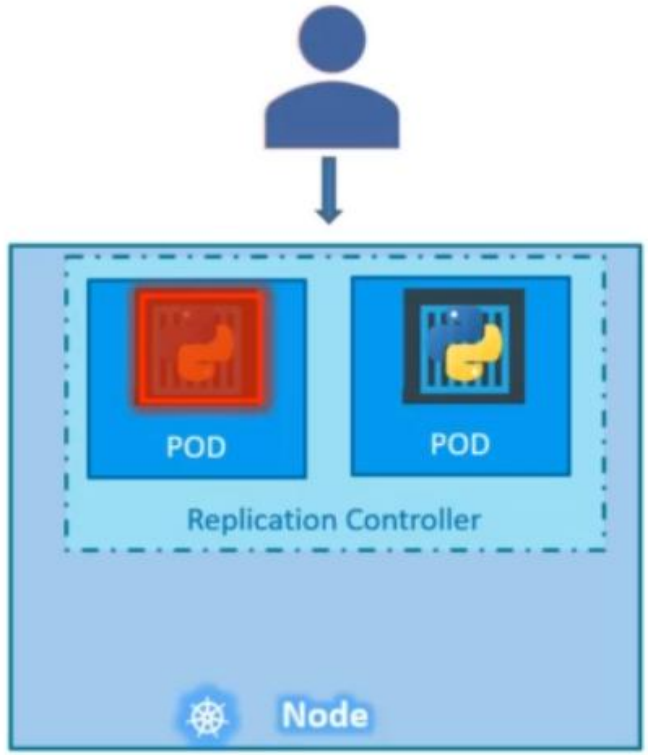
Para evitar que nossos usuários/clientes tenham seu acesso à nossa aplicação prejudicado é ideal que tenhamos mais de 1 instância do pod/container sendo executada ao mesmo tempo, pois desta forma se um deles parar o outro continua o serviço.

Conhecendo o Replication Controller e ReplicaSets

Replication Controller

É aqui que entra o Replication Controller, que nos ajuda a executar múltiplas instâncias de um pod/container no nosso cluster Kubernetes. Ele faz isso replicando (duplicando) o pod/container existente fazendo com que tenhamos uma cópia exata.

Isso provê para nossa aplicação
Alta Disponibilidade!



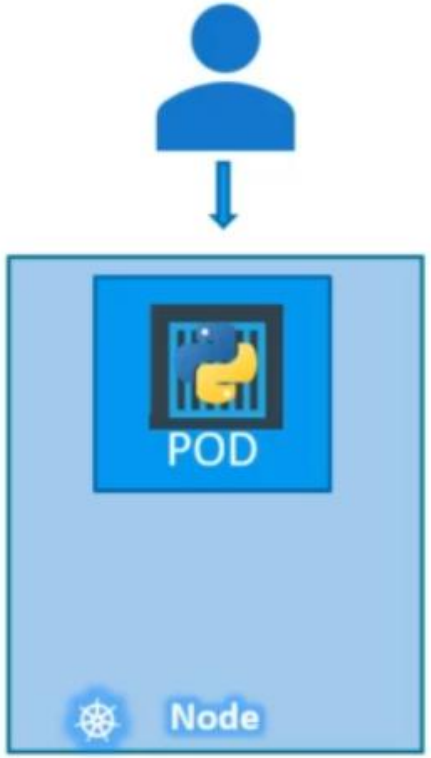
Conhecendo o Replication Controller e ReplicaSets

Replication Controller

Isso significa que o Replication Controller só nos ajuda se tivermos múltiplas instâncias do pod/container sendo executadas?

Ou seja, se nossa aplicação estiver sendo executada em um único pod/container não podemos usar o Replication Controller?

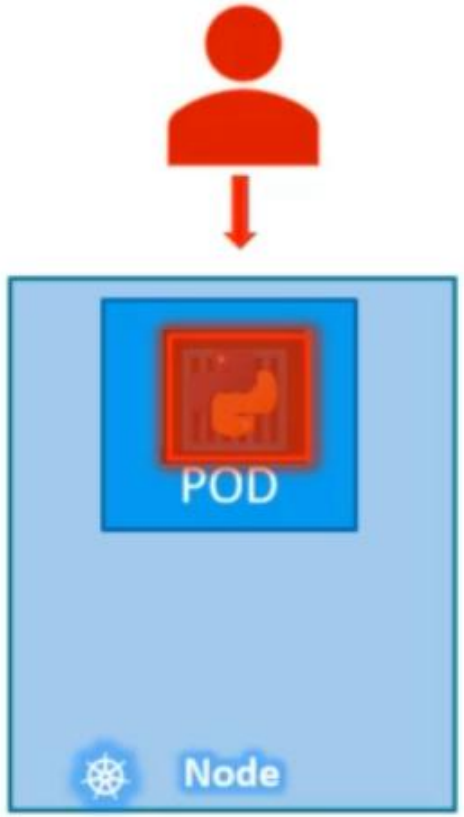
Podemos sim!



Conhecendo o Replication Controller e ReplicaSets

Replication Controller

Fazendo uso do Replication Controller, caso nosso pod/container falhe, uma replica do nosso pod é criada automaticamente.



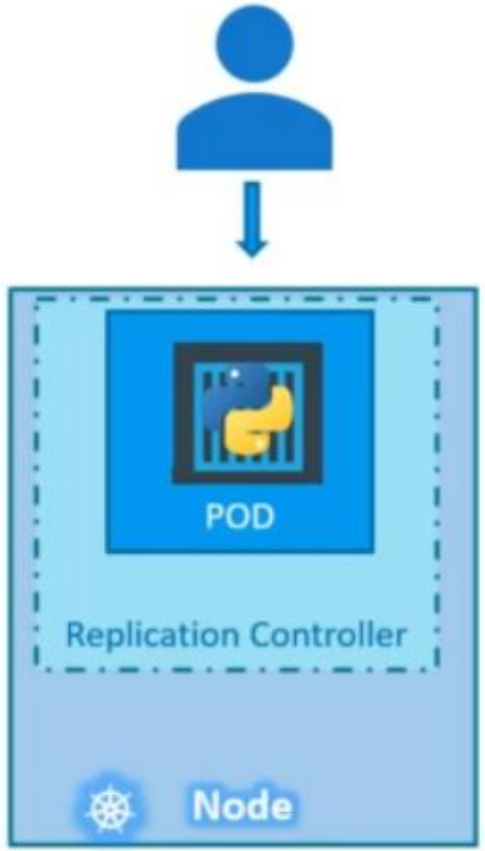
Conhecendo o Replication Controller e ReplicaSets

Replication Controller

Fazendo uso do Replication Controller, caso nosso pod/container falhe, uma replica do nosso pod é criada automaticamente.

O que o Replication Controller faz é se certificar que o número especificado de pods do cluster estejam sempre sendo executados.

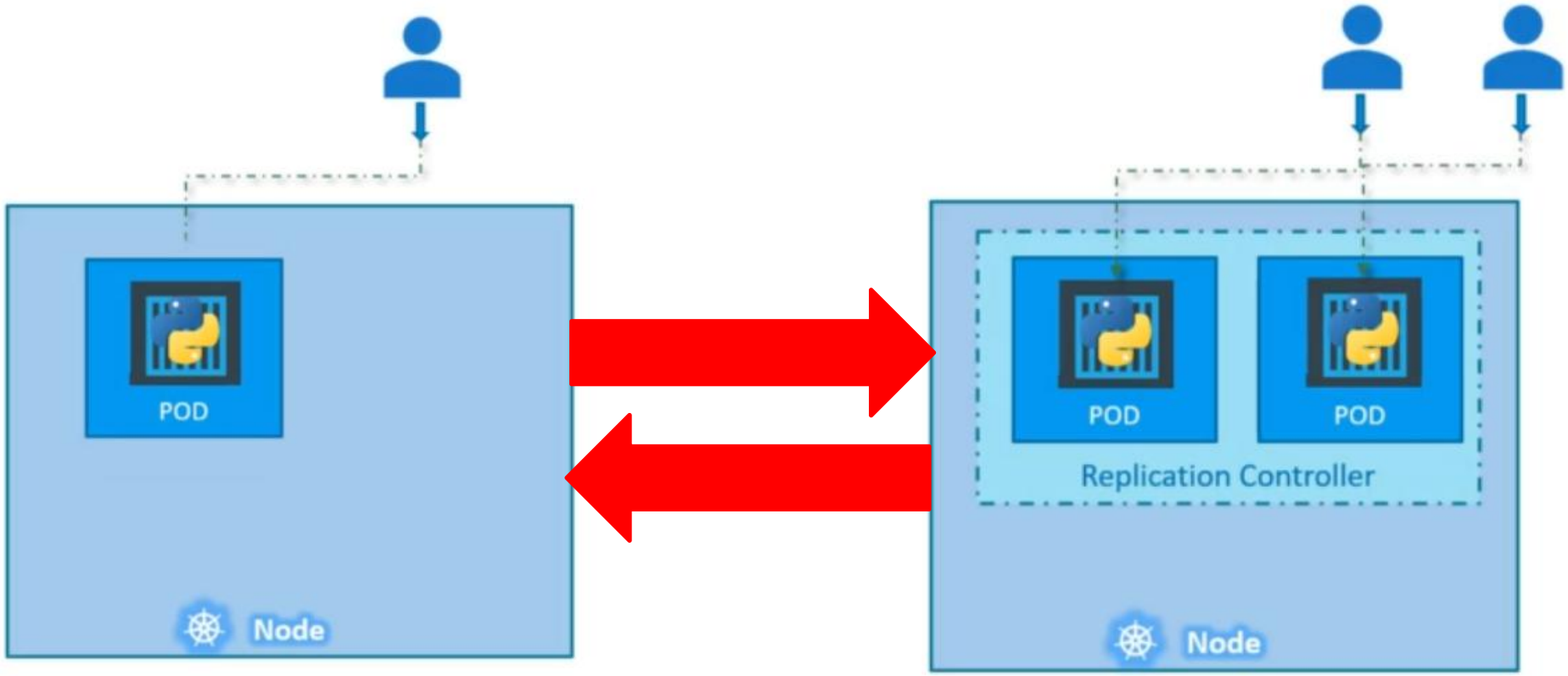
Não importa se temos um cluster com 1 pod ou 1000 pods.



Conhecendo o Replication Controller e ReplicaSets

Balanceamento de Carga e Escalamento da Aplicação

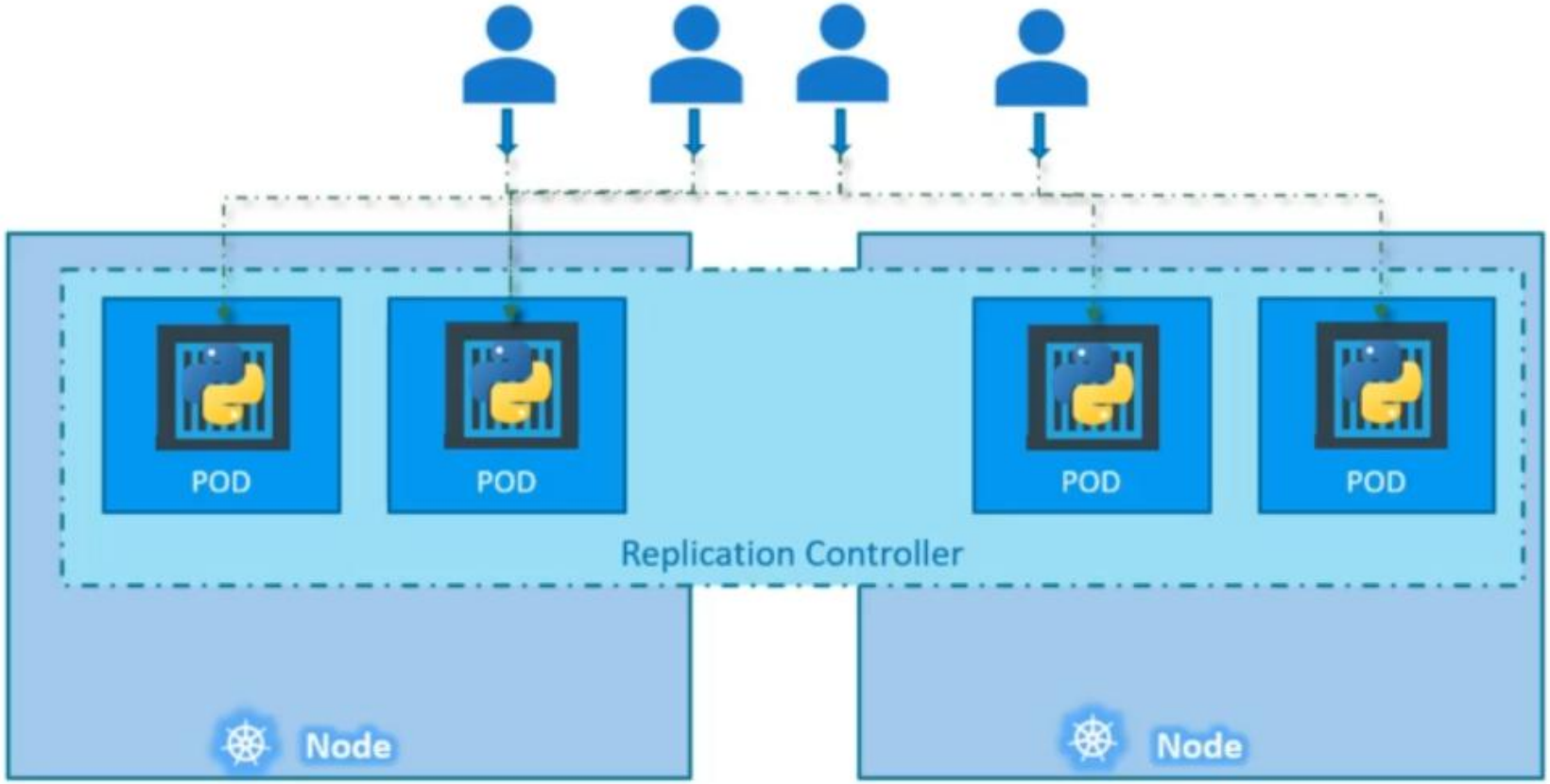
Outro fator importante para o uso do Replication Controller é a realização de balanceamento de carga e escalamento da nossa aplicação.



Conhecendo o Replication Controller e ReplicaSets

Balanceamento de Carga e Escalamento da Aplicação

Entenda que não há limites para o balanceamento de carga proporcionado pelo Replication Controller. Se necessário ele vai criar novas replicas em outros nodes do cluster.



Conhecendo o Replication Controller e ReplicaSets

Replication Controller x ReplicaSet

Ambos possuem o mesmo propósito, mas não são a mesma coisa.

Replication Controller é uma ferramenta “mais antiga” que está sendo substituída pela ReplicaSet.

Isso significa que tudo que aprendemos até então sobre Replication Controller é válido e funciona exatamente da mesma forma com ReplicaSet.

Não se preocupe que iremos aprender o que difere em ambas ferramentas um pouco mais à frente.

Conhecendo o Replication Controller e ReplicaSets

Replication Controller com Kubernetes YAML

Já sabemos que um arquivo de definição do Kubernetes escrito em YAML possui 4 campos top level.

Conhecendo o Replication Controller e ReplicaSets

Replication Controller com Kubernetes YAML

```yaml
apiVersion: v1
kind: ReplicationController
metadata:
  name: aplicacao-rc
  labels:
    app: aplicacao
    type: frontend

spec:
```

Nos 3 primeiros campos, não temos muita diferença se comparado ao que fizemos para o POD anteriormente. Note contudo que o ‘kind’, ou seja, tipo, é ReplicationController.

É na parte de ‘spec’ (especificações do objeto) que ocorrem as maiores mudanças, pois é aqui que será especificado qual é o pod que deverá ser replicado se necessário.

Conhecendo o Replication Controller e ReplicaSets

Replication Controller com Kubernetes YAML

```yaml
apiVersion: v1
kind: ReplicationController
metadata:
  name: aplicacao-rc
  labels:
    app: aplicacao
    type: frontend

spec:
  template:
```

Em ‘spec’ devemos definir qual o template (modelo) do pod a ser replicado.

Conhecendo o Replication Controller e ReplicaSets

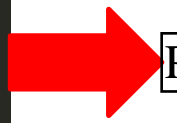
Replication Controller com Kubernetes YAML

```yaml
apiVersion: v1
kind: ReplicationController
metadata:
  name: aplicacao-rc
  labels:
    app: aplicacao
    type: frontend

spec:
  template:
```

Em ‘spec’ devemos definir qual o template (modelo) do pod a ser replicado.

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: aplicacao-pod
  labels:
    app: aplicacao
    type: frontend
spec:
  containers:
    - name: nginx-container
      image: nginx
```



POD

Conhecendo o Replication Controller e ReplicaSets

Replication Controller com Kubernetes YAML

```yaml
apiVersion: v1
kind: ReplicationController
metadata:
  name: aplicacao-rc
  labels:
    app: aplicacao
    type: frontend

spec:
  template:
```

Em ‘spec’ devemos definir qual o template (modelo) do pod a ser replicado.

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: aplicacao-pod
  labels:
    app: aplicacao
    type: frontend
spec:
  containers:
    - name: nginx-container
      image: nginx
```

POD

O template do pod é todo o conteúdo do pod existente excluindo-se as duas primeiras linhas.

Conhecendo o Replication Controller e ReplicaSets

Replication Controller com Kubernetes YAML

```yaml
apiVersion: v1
kind: ReplicationController
metadata:
  name: aplicacao-rc
  labels:
    app: aplicacao
    type: frontend

spec:
  template:
    metadata:
      name: aplicacao-pod
      labels:
        app: aplicacao
        type: frontend
    spec:
      containers:
        - name: nginx-container
          image: nginx
```

Atenção ao correto espaçamento hierárquico dos elementos YAML exatamente conforme aprendemos no curso de Docker.

Conhecendo o Replication Controller e ReplicaSets

Replication Controller com Kubernetes YAML

```yaml
apiVersion: v1
kind: ReplicationController
metadata:
  name: aplicacao-rc
  labels:
    app: aplicacao
    type: frontend

spec:
  template:
    metadata:
      name: aplicacao-pod
      labels:
        app: aplicacao
        type: frontend
    spec:
      containers:
        - name: nginx-container
          image: nginx
```

Replication Controller

POD

Note que agora temos 2 campos top level "metadata" e "spec", mas cada um do seu próprio objeto Kubernetes.

Conhecendo o Replication Controller e ReplicaSets

Replication Controller com Kubernetes YAML

```yaml
apiVersion: v1
kind: ReplicationController
metadata:
  name: aplicacao-rc
  labels:
    app: aplicacao
    type: frontend

spec:
  template:
    metadata:
      name: aplicacao-pod
      labels:
        app: aplicacao
        type: frontend
    spec:
      containers:
        - name: nginx-container
          image: nginx
```

Uma informação faltante ainda no arquivo de definição do Replication Controller é quantas réplicas devemos ter?

Ou seja, você tem sua aplicação sendo executada em 2 pods. Este é o número de réplicas que devem estar sendo executados simultaneamente.

Caso um destes pods falhe, outro será replicado.

Conhecendo o Replication Controller e ReplicaSets

Replication Controller com Kubernetes YAML

```yaml
apiVersion: v1
kind: ReplicationController
metadata:
  name: aplicacao-rc
  labels:
    app: aplicacao
    type: frontend

spec:
  template:
    metadata:
      name: aplicacao-pod
      labels:
        app: aplicacao
        type: frontend
    spec:
      containers:
        - name: nginx-container
          image: nginx
  replicas: 2
```

O número de replicas é definido pelo campo ‘replicas’ que deve estar no mesmo nível hierárquico de ‘template.

Conhecendo o Replication Controller e ReplicaSets

Kubectl

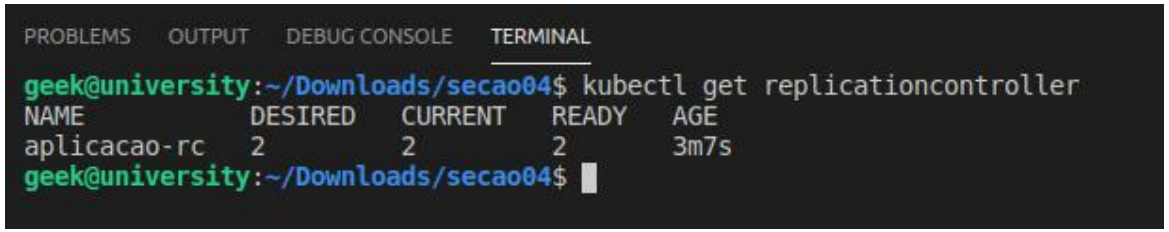
O comando para criação do objeto Replica Controller através de um arquivo de definição do Kubernetes é o mesmo que usamos para criação do pod também através do arquivo:

Ao executar o comando de criação, o replication controller irá criar primeiro os pods especificados na quantidade de replicas que devem existir, neste nosso exemplo 2.

Conhecendo o Replication Controller e ReplicaSets

Kubectl

Podemos consultar a quantidade de replicas existentes com o comando:

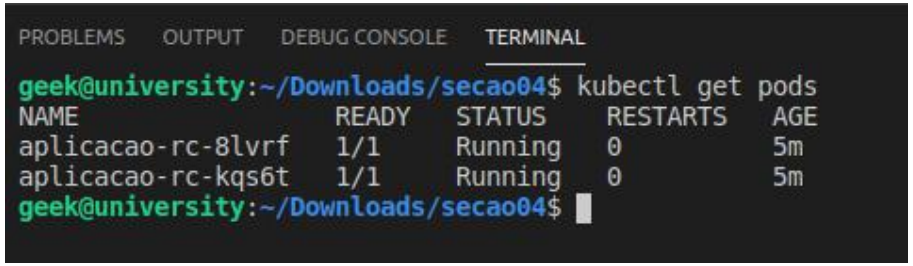


Note que ene nos mostra o nome do Replication Controller criado, a quantidade de réplicas “desejadas”, a quantidade de réplicas existente atualmente, quantos estão prontos e o tempo de vida.

Conhecendo o Replication Controller e ReplicaSets

Kubectl

E nada mudou em relação ao que aprendemos antes, então desta forma, se quisermos verificar os pods existentes basta executar o comando já conhecido:



Conhecendo o Replication Controller e ReplicaSets

ReplicaSet

Mas e a diferença entre o Replication Controller e o ReplicaSet?

Conhecendo o Replication Controller e ReplicaSets

ReplicaSet

Mas e a diferença entre o Replication Controller e o ReplicaSet?

```
apiVersion: apps/v1
kind:
metadata:

spec:
```

Enquanto no Replication Controller usamos como apiVersion o valor “v1” com ReplicaSet este valor deve ser no formado apps/v1 e caso você não utilize este formato terá como erro:

```
error: unable to recognize "rs.yaml":
 no matches for /, Kind=ReplicaSet
```

Conhecendo o Replication Controller e ReplicaSets

ReplicaSet

Mas e a diferença entre o Replication Controller e o ReplicaSet?

O tipo do objeto é especificado no campo Kind como sendo ReplicaSet

Em metadata não difere do que usamos em Replication Controller.

Conhecendo o Replication Controller e ReplicaSets

ReplicaSet

Mas e a diferença entre o Replication Controller e o ReplicaSet?

```yaml
apiVersion: apps/v1
kind: ReplicaSet
metadata:
  name: aplicacao-rs
  labels:
    app: aplicacao
    type: frontend
spec:
  template:
    metadata:
      name: aplicacao-pod
      labels:
        app: aplicacao
        type: frontend
    spec:
      containers:
        - name: nginx-container
          image: nginx
  replicas: 2
```

O campo ‘spec’ é praticamente igual ao que temos com o Replication Controller tendo apenas 1 diferença...

Conhecendo o Replication Controller e ReplicaSets

ReplicaSet

Mas e a diferença entre o Replication Controller e o ReplicaSet?

```yaml
apiVersion: apps/v1
kind: ReplicaSet
metadata:
  name: aplicacao-rs
  labels:
    app: aplicacao
    type: frontend
spec:
  template:
    metadata:
      name: aplicacao-pod
      labels:
        app: aplicacao
        type: frontend
    spec:
      containers:
        - name: nginx-container
          image: nginx
  replicas: 2
  selector:
```

O campo ‘selector’ é a principal diferença entre o ReplicaSet e o Replication Controller.

Conhecendo o Replication Controller e ReplicaSets

Selector

O campo ‘selector’ está disponível tanto em Replication Controller quanto no ReplicaSet, mas enquanto no Replication Controller ele é opcional, aqui no ReplicaSet ele é obrigatório.

Mas qual a finalidade do campo ‘selector’?

Um cluster pode ter múltiplos pods, inclusive pods que já estejam sendo usados antes da criação ou uso de um Replication Controller ou ReplicaSets.

O campo Selector indica, dentre os pods existentes, qual deve ser usado para ser replicado no cluster.

Conhecendo o Replication Controller e ReplicaSets

ReplicaSet

```yaml
apiVersion: apps/v1
kind: ReplicaSet
metadata:
  name: aplicacao-rs
  labels:
    app: aplicacao
    type: frontend
spec:
  template:
    metadata:
      name: aplicacao-pod
      labels:
        app: aplicacao
        type: frontend
    spec:
      containers:
        - name: nginx-container
          image: nginx
  replicas: 2
  selector:
    matchLabels:
      type: frontend
```

Note que no template onde informamos o metadata do pod usado no ReplicaSet temos em “labels” o campo “type” onde o valor é “frontend”.

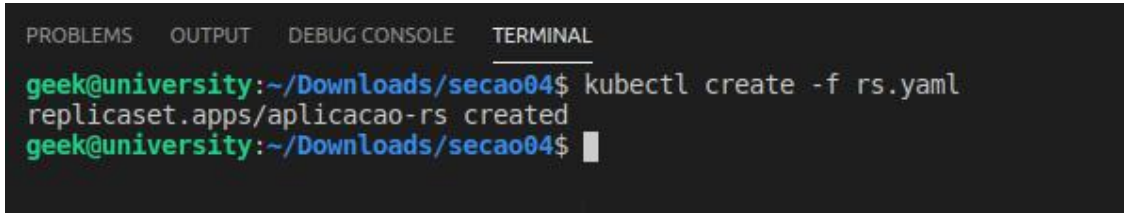
No selector, informamos exatamente este valor para correspondência do labels como sendo o “type” com valor “frontend”.

Usando ReplicaSet temos vários outros campos que podemos usar para “corresponder” ao pod a ser usado, enquanto no Replication Controller outros campos não estão disponíveis.

Conhecendo o Replication Controller e ReplicaSets

kubectl

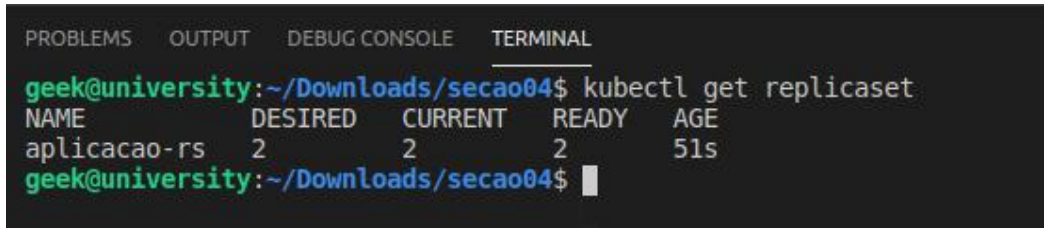
Como sempre, o comando para criação do objeto ReplicaSet é o mesmo usado para outros objetos:



Conhecendo o Replication Controller e ReplicaSets

kubectl

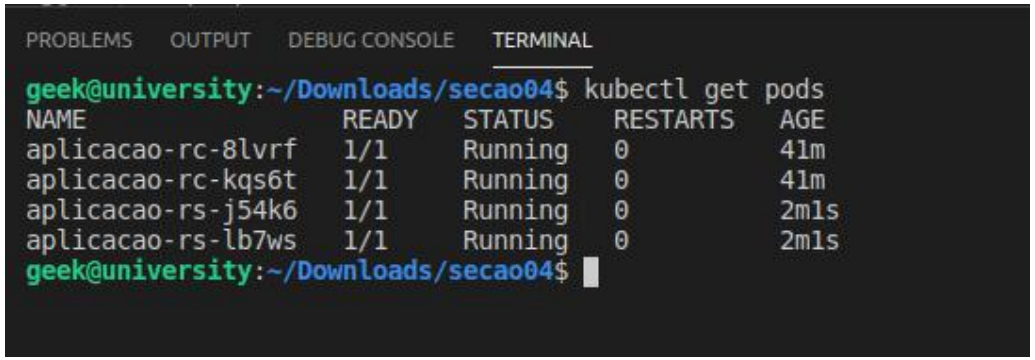
E para checar informações sobre o replicaset podemos executar:



Conhecendo o Replication Controller e ReplicaSets

kubectl

E para verificar os pods existentes no cluster, nada mudou:



OBS: Note que temos pods tanto do Replication Controller quanto do ReplicaSet

Conhecendo o Replication Controller e ReplicaSets

Labels e Selectors

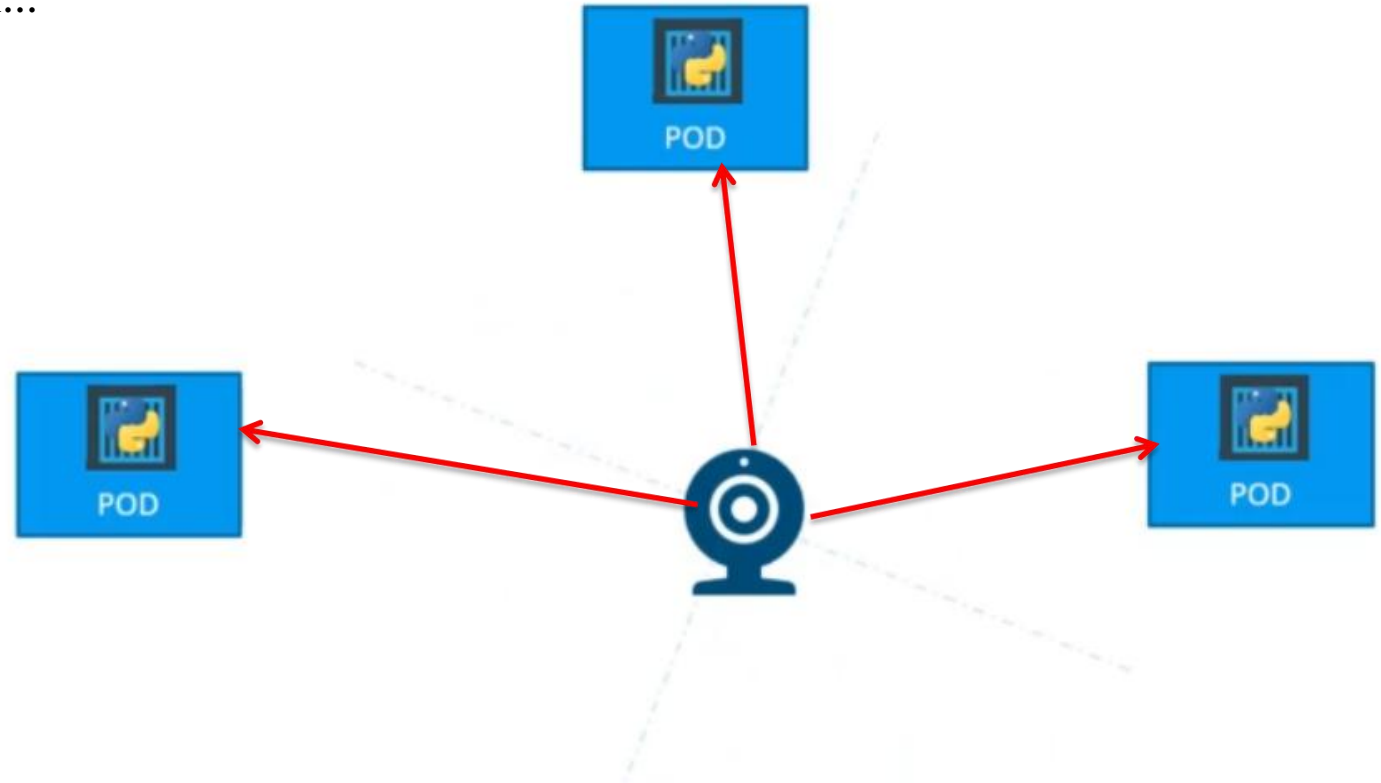
Qual a utilidade de espeficiar labens nos nossos pods? E qual a relação destes labels com o Selector?

Conhecendo o Replication Controller e ReplicaSets

Labels e Selectors

O Replication Controller/ReplicaSet está monirodando os pods existentes no cluster para se certificar que eles estão sendo executados corretamente, e então criar novas instâncias (replicas) se necessário.

Identificar 2 ou 3 pods é fácil...

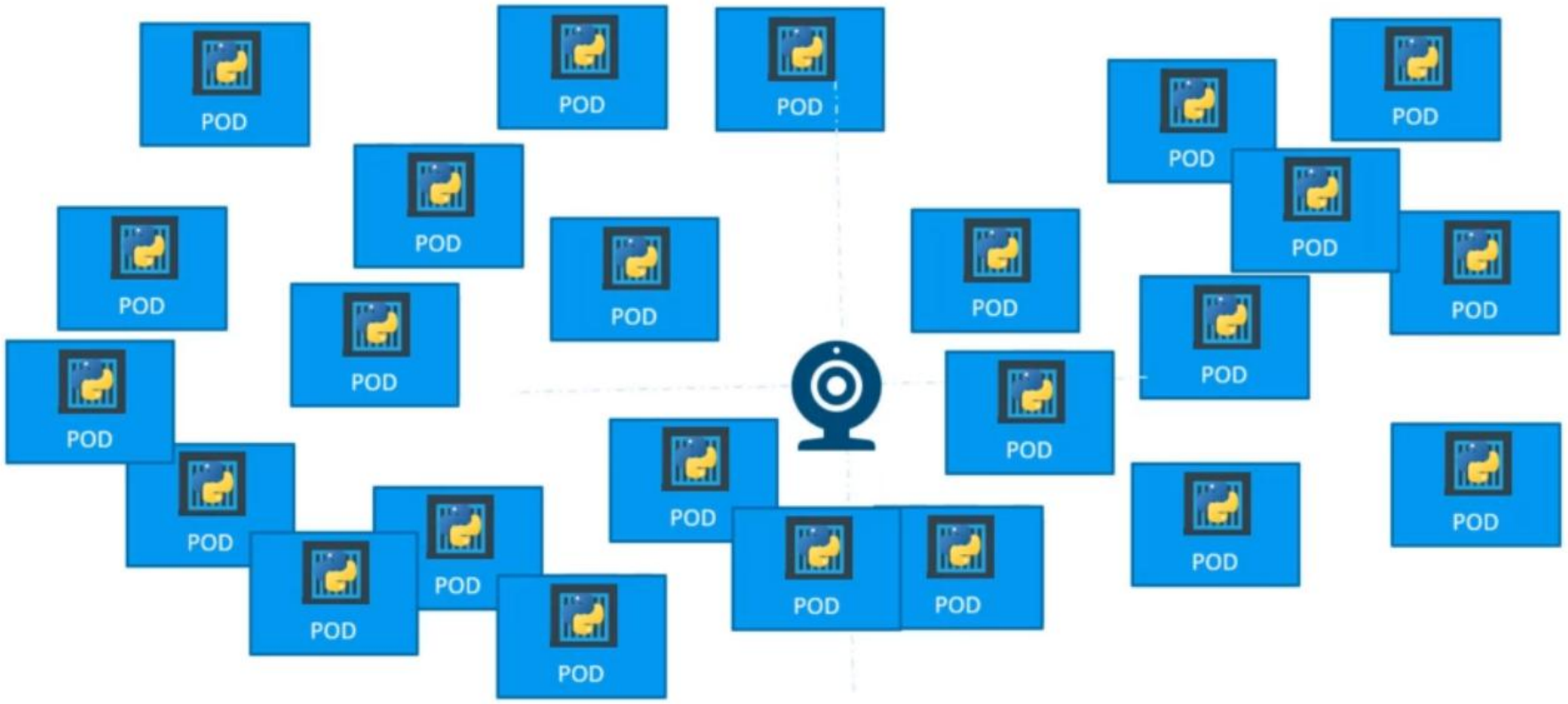


Conhecendo o Replication Controller e ReplicaSets

Labels e Selectors

Mas como seria se no nosso cluster tivesse dezenas, centenas ou até mesmo milhares de pods sendo executados?

É aqui que a utilização de labels para identificação dos pods entra em jogo.



Conhecendo o Replication Controller e ReplicaSets

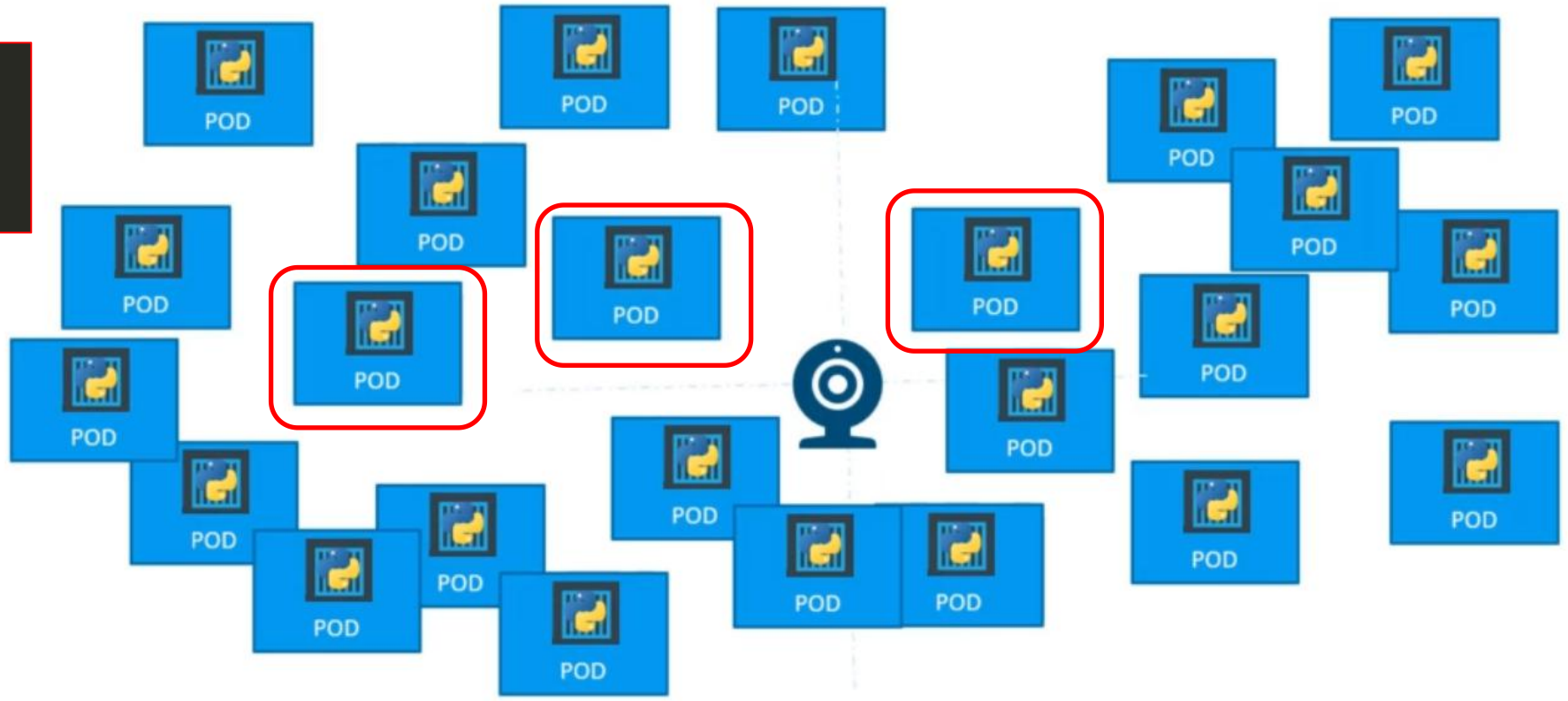
Labels e Selectors

Mas como seria se no nosso cluster tivesse dezenas, centenas ou até mesmo milhares de pods sendo executados?

É aqui que a utilização de labels para identificação dos pods entra em jogo.

```yaml
name: frontend-pod
labels:
  app: frontend-app
  type: frontend
```

```yaml
selector:
  matchLabels:
    type: frontend
```



Conhecendo o Replication Controller e ReplicaSets

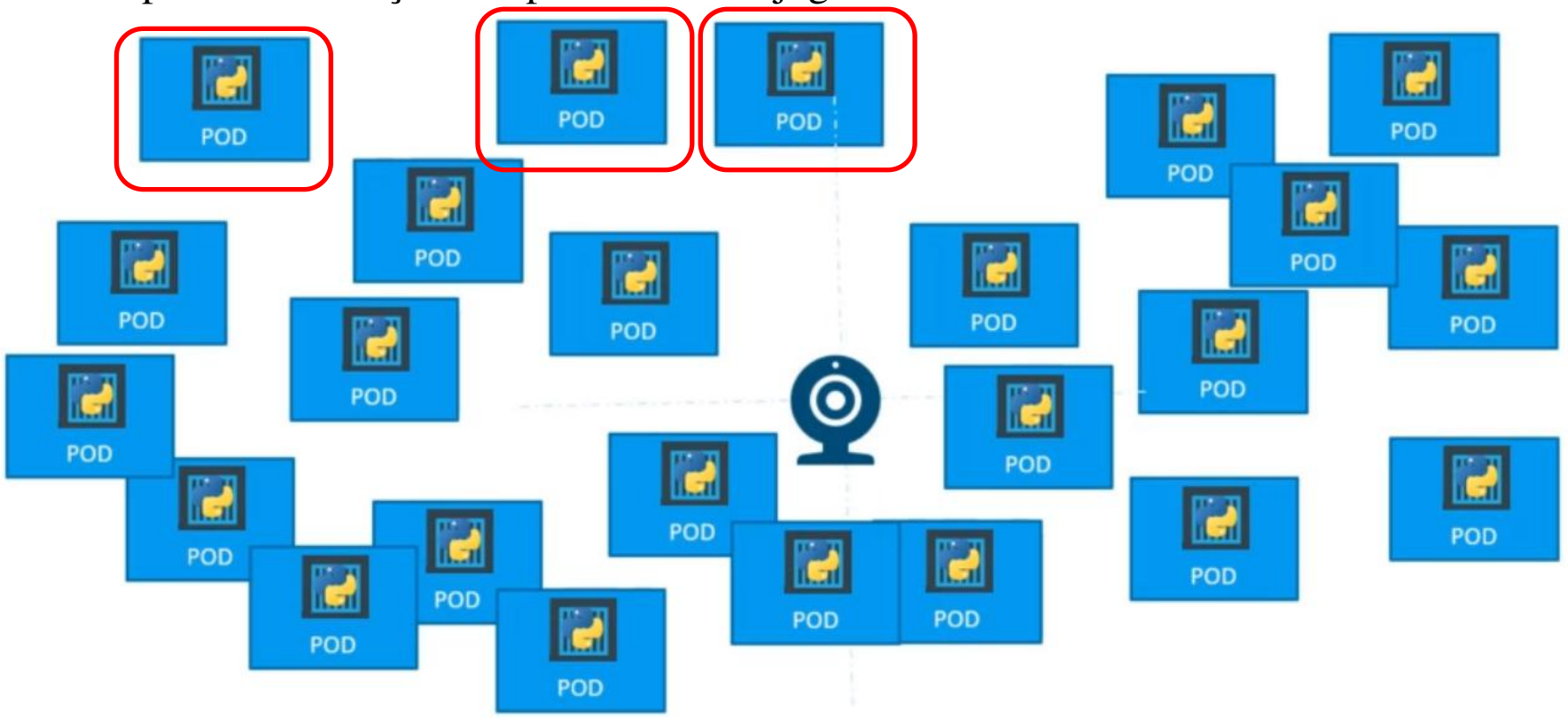
Labels e Selectors

Mas como seria se no nosso cluster tivesse dezenas, centenas ou até mesmo milhares de pods sendo executados?

É aqui que a utilização de labels para identificação dos pods entra em jogo.

```yaml
name: backend-pod
labels:
  app: backend-app
  type: backend
```

```yaml
selector:
  matchLabels:
    type: backend
```



Conhecendo o Replication Controller e ReplicaSets

Labels e Selectors

Os labels são usados a todo momento nos objetos Kubernetes como identificação nas mais diferentes funções e componentes.

Conhecendo o Replication Controller e ReplicaSets

Escalando a aplicação

Pudemos compreender como fazer com que tenhamos um mínimo de instâncias de pods sempre em execução mantendo nossa aplicação com alta disponibilidade fazendo uso do Replication Controller/ReplicaSet.

Mas isso não irá escalar nossa aplicação caso a carga de trabalho esteja muito alta.

Como podemos resolver isso?

Conhecendo o Replication Controller e ReplicaSets

Escalando a aplicação

Existem várias formas de se resolver isso.

Uma delas é alterar o número de replicadas no arquivo de definição do ReplicationController/ReplicaSet

E então usar o kubectl replace para atualizar as definições do objeto conforme:

Conhecendo o Replication Controller e ReplicaSets

Escalando a aplicação

Outra forma é usar o kubectl scale conforme:

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL
geek@university:~/Downloads/secao04$ kubectl scale --replicas=6 -f rs.yaml
replicaset.apps/aplicacao-rs scaled
geek@university:~/Downloads/secao04$
```

Note que informamos a nova quantidade de replicas através do parâmetro e ainda informamos qual é o arquivo de definição nos quais a definição está sendo alterada.

Conhecendo o Replication Controller e ReplicaSets

Escalando a aplicação

Podemos ainda não especificar o arquivo, mas sim o tipo de objeto alterado e o nome do objeto conforme:

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL
geek@university:~/Downloads/secao04$ kubectl scale --replicas=8 replicaset aplicacao-rs
replicaset.apps/aplicacao-rs scaled
geek@university:~/Downloads/secao04$
```

Note que o nome do objeto replicaset deve ser o mesmo do objeto declarado.

```yaml
apiVersion: apps/v1
kind: ReplicaSet
metadata:
  name: aplicacao-rs
  labels:
    app: aplicacao
    type: frontend
spec:
  template:
    metadata:
      name: aplicacao-pod
      labels:
        app: aplicacao
        type: frontend
    spec:
      containers:
        - name: nginx-container
```

OBS: Estes comandos não alteram os dados do arquivo, mas apenas as configurações dos objetos no cluster.

Conhecendo o Replication Controller e ReplicaSets

Não se preocupe que em aulas futuras iremos entrar em tópicos mais avançados e aprender como escalar a aplicação de forma automática de acordo com a carga de trabalho.

Recomendo revisar e anotar os comandos para consulta rápida quando necessário.

Comandos úteis:

kubectl create -f arquivo.yaml

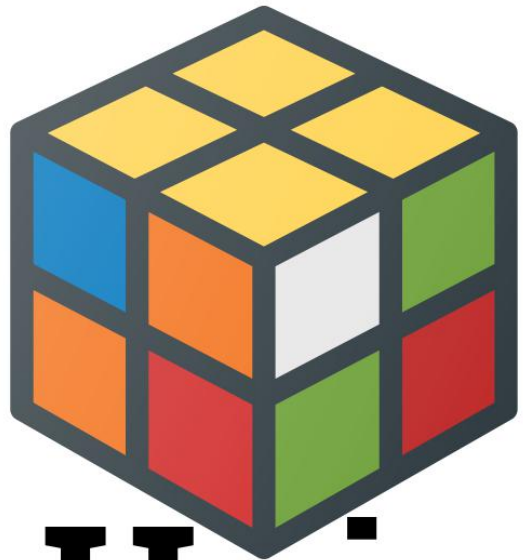
kubectl get replicaset

kubectl delete replicaset nome-do-replicaset

kubectl replace -f arquivo.yaml

kubectl scale --replicas=x -f arquivo.yaml

OBS: Não se preocupe pois a próxima aula iremos realizar um exercício prático em conjunto.



Geek University

Evolua seu lado geek!

www.geekuniversity.com.br