DESCOMPLICANDO O KUBERNETES

O LIVRO

Deployments

Quando você utiliza o *kubectl run*, você está realizando o deploy de um objeto chamado Deployment. Como outros objetos, o Deployment também pode ser criado através de um arquivo YAML ou de um JSON, os spec files.

Se você deseja alterar alguma configuração de seus objetos, como o pod, você pode utilizar o kubectl apply, através de um spec file, ou ainda através do kubectl edit.

As versões anteriores dos ReplicaSets são mantidas, possibilitando o rollback em caso de falhas.

As labels são importantes para o gerenciamento do cluster, pois com ela é possível buscar ou selecionar recursos em seu cluster, fazendo com que você consiga organizar em pequenas categorias, facilitando assim a sua busca e organizando seus pods e seus recursos do cluster. As labels não são recursos do API server, eles são armazenados no metadata em formato chave-valor.

Antes nos tinhas somente o RC, Replication Controller, que era um controle sobre o número de réplicas que determinado pod estava executando, o problema que todo esse gerenciamento era feito do lado do client. Para solucionar esse problema, foi adicionado o objeto Deployment, que permite a atualização pelo lado do server. Deployments geram ReplicaSets, que oferecerem melhores opções do que o ReplicationController, e por esse motivo está sendo substituído.

Vamos criar os nossos primeiros Deployments:

```
# vim primeiro-deployment.yaml
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  labels:
    run: nginx
    app: giropops
  name: primeiro-deployment
  namespace: default
spec:
  replicas: 1
  selector:
    matchLabels:
      run: nginx
  template:
    metadata:
```

```
      labels:
        run: nginx
        dc: UK
    spec:
      containers:
      - image: nginx
        imagePullPolicy: Always
        name: nginx2
        ports:
        - containerPort: 80
          protocol: TCP
        resources: {}
        terminationMessagePath: /dev/termination-log
        terminationMessagePolicy: File
      dnsPolicy: ClusterFirst
      restartPolicy: Always
      schedulerName: default-scheduler
      securityContext: {}
      terminationGracePeriodSeconds: 30
```

```
# kubectl create -f primeiro-deployment.yaml
deployment.extensions/primeiro-deployment created
```

```
# vim segundo-deployment.yaml
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  labels:
    run: nginx
  name: segundo-deployment
  namespace: default
spec:
  replicas: 1
  selector:
    matchLabels:
      run: nginx
  template:
    metadata:
      labels:
        run: nginx
        dc: Netherlands
    spec:
      containers:
      - image: nginx
```

```
        imagePullPolicy: Always
        name: nginx2
        ports:
        - containerPort: 80
          protocol: TCP
        resources: {}
        terminationMessagePath: /dev/termination-log
        terminationMessagePolicy: File
      dnsPolicy: ClusterFirst
      restartPolicy: Always
      schedulerName: default-scheduler
      securityContext: {}
      terminationGracePeriodSeconds: 30
```

 # kubectl create -f segundo-deployment.yaml

```
deployment.extensions/segundo-deployment created
```

 # kubectl get deployment

```
NAME                 DESIRED   CURRENT   UP-TO-DATE   AVAILABLE
AGE
primeiro-deployment  1         1         1            1
6m
segundo-deployment   1         1         1            1
1m
```

 # kubectl get pods

```
NAME                                  READY   STATUS    RESTARTS
AGE
primeiro-deployment-68c9dbf8b8-kjqpt 1/1     Running   0
19s
segundo-deployment-59db86c584-cf9pp  1/1     Running   0
15s
```

 # kubectl describe pod primeiro-deployment-68c9dbf8b8-kjqpt

```
Name:               primeiro-deployment-68c9dbf8b8-kjqpt
Namespace:          default
Priority:           0
PriorityClassName:  <none>
Node:               elliot-02/10.138.0.3
Start Time:         Sat, 04 Aug 2018 00:45:29 +0000
Labels:             dc=UK
                    pod-template-hash=2475869464
                    run=nginx
Annotations:        <none>
```

```
Status:             Running
IP:                 10.46.0.1
Controlled By:      ReplicaSet/primeiro-deployment-68c9dbf8b8
Containers:
  nginx2:
    Container ID:
docker://963ec997a0aa4aa3cecabdb3c59f67d80e7010c51eac23735524899f7
f2dd4f9
    Image:          nginx
    Image ID:
docker-pullable://nginx@sha256:d85914d547a6c92faa39ce7058bd7529baa
cab7e0cd4255442b04577c4d1f424
    Port:           80/TCP
    Host Port:      0/TCP
    State:          Running
      Started:      Sat, 04 Aug 2018 00:45:36 +0000
    Ready:          True
    Restart Count:  0
    Environment:    <none>
    Mounts:
      /var/run/secrets/kubernetes.io/serviceaccount from
default-token-np77m (ro)
Conditions:
  Type              Status
  Initialized       True
  Ready             True
  ContainersReady   True
  PodScheduled      True
Volumes:
  default-token-np77m:
    Type:        Secret (a volume populated by a Secret)
    SecretName:  default-token-np77m
    Optional:    false
QoS Class:       BestEffort
Node-Selectors:  <none>
Tolerations:     node.kubernetes.io/not-ready:NoExecute for 300s
                 node.kubernetes.io/unreachable:NoExecute for 300s
Events:
  Type    Reason     Age   From                Message
  ----    ------     ----  ----                -------
  Normal  Scheduled  51s   default-scheduler   Successfully
assigned default/primeiro-deployment-68c9dbf8b8-kjqpt to elliot-02
  Normal  Pulling    50s   kubelet, elliot-02  pulling image
"nginx"
  Normal  Pulled     44s   kubelet, elliot-02  Successfully pulled
image "nginx"
```

```
  Normal  Created    44s   kubelet, elliot-02  Created container
  Normal  Started    44s   kubelet, elliot-02  Started container
```

 # kubectl describe pod segundo-deployment-59db86c584-cf9pp

```
Name:               segundo-deployment-59db86c584-cf9pp
Namespace:          default
Priority:           0
PriorityClassName:  <none>
Node:               elliot-02/10.138.0.3
Start Time:         Sat, 04 Aug 2018 00:45:49 +0000
Labels:             dc=Netherlands
                    pod-template-hash=1586427140
                    run=nginx
Annotations:        <none>
Status:             Running
IP:                 10.46.0.2
Controlled By:      ReplicaSet/segundo-deployment-59db86c584
Containers:
  nginx2:
    Container ID:
docker://a9e6b5463341e62eff9e45c8c0aace14195f35e41be088ca386949500
a1f2bb0
    Image:          nginx
    Image ID:
docker-pullable://nginx@sha256:d85914d547a6c92faa39ce7058bd7529baa
cab7e0cd4255442b04577c4d1f424
    Port:           80/TCP
    Host Port:      0/TCP
    State:          Running
      Started:      Sat, 04 Aug 2018 00:45:51 +0000
    Ready:          True
    Restart Count:  0
    Environment:    <none>
    Mounts:
      /var/run/secrets/kubernetes.io/serviceaccount from
default-token-np77m (ro)
Conditions:
  Type              Status
  Initialized       True
  Ready             True
  ContainersReady   True
  PodScheduled      True
Volumes:
  default-token-np77m:
    Type:        Secret (a volume populated by a Secret)
```

```
    SecretName:  default-token-np77m
    Optional:    false
QoS Class:       BestEffort
Node-Selectors:  <none>
Tolerations:     node.kubernetes.io/not-ready:NoExecute for 300s
                 node.kubernetes.io/unreachable:NoExecute for 300s
Events:
  Type    Reason     Age   From                Message
  ----    ------     ----  ----                -------
  Normal  Scheduled  2m    default-scheduler   Successfully
assigned default/segundo-deployment-59db86c584-cf9pp to elliot-02
  Normal  Pulling    2m    kubelet, elliot-02  pulling image
"nginx"
  Normal  Pulled     2m    kubelet, elliot-02  Successfully pulled
image "nginx"
  Normal  Created    2m    kubelet, elliot-02  Created container
  Normal  Started    2m    kubelet, elliot-02  Started container
```

 # kubectl describe deployment primeiro-deployment

```
Name:                   primeiro-deployment
Namespace:              default
CreationTimestamp:      Sat, 04 Aug 2018 00:45:29 +0000
Labels:                 app=giropops
                        run=nginx
Annotations:            deployment.kubernetes.io/revision=1
Selector:               run=nginx
Replicas:               1 desired | 1 updated | 1 total | 1
available | 0 unavailable
StrategyType:           RollingUpdate
MinReadySeconds:        0
RollingUpdateStrategy:  1 max unavailable, 1 max surge
Pod Template:
  Labels:  dc=UK
           run=nginx
  Containers:
   nginx2:
    Image:        nginx
    Port:         80/TCP
    Host Port:    0/TCP
    Environment:  <none>
    Mounts:       <none>
  Volumes:        <none>
Conditions:
  Type           Status  Reason
  ----           ------  ------
  Available      True    MinimumReplicasAvailable
```

```
  Progressing    True    NewReplicaSetAvailable
OldReplicaSets:  <none>
NewReplicaSet:   primeiro-deployment-68c9dbf8b8 (1/1 replicas
created)
Events:
  Type    Reason             Age   From                   Message
  ----    ------             ----  ----                   -------
  Normal  ScalingReplicaSet  3m    deployment-controller  Scaled
up replica set primeiro-deployment-68c9dbf8b8 to 1
```

 # kubectl describe deployment segundo-deployment

```
Name:                   segundo-deployment
Namespace:              default
CreationTimestamp:      Sat, 04 Aug 2018 00:45:49 +0000
Labels:                 run=nginx
Annotations:            deployment.kubernetes.io/revision=1
Selector:               run=nginx
Replicas:               1 desired | 1 updated | 1 total | 1
available | 0 unavailable
StrategyType:           RollingUpdate
MinReadySeconds:        0
RollingUpdateStrategy:  1 max unavailable, 1 max surge
Pod Template:
  Labels:  dc=Netherlands
           run=nginx
  Containers:
   nginx2:
    Image:        nginx
    Port:         80/TCP
    Host Port:    0/TCP
    Environment:  <none>
    Mounts:       <none>
  Volumes:        <none>
Conditions:
  Type           Status  Reason
  ----           ------  ------
  Available      True    MinimumReplicasAvailable
  Progressing    True    NewReplicaSetAvailable
OldReplicaSets:  <none>
NewReplicaSet:   segundo-deployment-59db86c584 (1/1 replicas
created)
Events:
  Type    Reason             Age   From                   Message
  ----    ------             ----  ----                   -------
```

```
  Normal  ScalingReplicaSet  3m    deployment-controller  Scaled
up replica set segundo-deployment-59db86c584 to 1
```

Filtrando por Labels

Quando criamos nossos Deployments adicionamos os Labels abaixo :

```
  labels:
        run: nginx
        dc: UK
---
  labels:
        run: nginx
        dc: Netherlands
```

Os Labels são utilizados para a organização do cluster, vamos listar nossos Pods procurando pelas Labels.

Primeiro vamos realizar uma pesquisa utilizando as labels dc=UK e dc=Netherlands:

```
# kubectl get pods -l dc=UK
NAME                                  READY   STATUS    RESTARTS
AGE
primeiro-deployment-68c9dbf8b8-kjqpt 1/1    Running   0          3m

# kubectl get pods -l dc=Netherlands
NAME                                 READY STATUS    RESTARTS   AGE
segundo-deployment-59db86c584-cf9pp 1/1   Running   0          4m
```

Caso queira uma saída mais personalizada podemos listar da seguinte forma, veja.

```
# kubectl get pod -L dc
NAME                          READY STATUS   RESTARTS AGE DC
primeiro-deployment-68c9...   1/1   Running  0        5m  UK
segundo-deployment-59db ...   1/1   Running  0        5m
Netherlands
```

Removendo o label:

```
# kubectl label nodes elliot-02 dc-
```

Removendo um determinado label de todos os nodes:

```
# kubectl label nodes --all dc-
```

Node Selector

O Node Selector é uma forma de classificar nosso nodes como por exemplo nosso node elliot-02 que possui disco SSD e está localizado no DataCenter UK, e o node elliot-03 que possui disco HDD e está localizado no DataCenter Netherlands.
Agora que temos essas informações vamos criar esses labels em nossos nodes, para utilizar os nodeSelector.

```
# kubectl label node elliot-02 disk=SSD
node/elliot-02 labeled

# kubectl label node elliot-02 dc=UK
node/elliot-02 labeled

# kubectl label node elliot-03 dc=Netherlands
node/elliot-03 labeled

# kubectl label nodes elliot-03 disk=hdd
node/elliot-03 labeled
```

Opa! Acabamos declarando o disk=hdd em letra minúscula , como arrumos isso ? sobre escrevendo o label como no comando abaixo:

```
# kubectl label nodes elliot-03 disk=HDD --overwrite
node/elliot-03 labeled
```

Para saber os labels configurado em cada node basta executar o comando:

```
# kubectl label nodes elliot-02 --list
dc=UK
disk=SSD
kubernetes.io/hostname=elliot-02
beta.kubernetes.io/arch=amd64
beta.kubernetes.io/os=linux

# kubectl label nodes elliot-03 --list
beta.kubernetes.io/os=linux
dc=Netherlands
disk=HDD
kubernetes.io/hostname=elliot-03
beta.kubernetes.io/arch=amd64
```

Agora, basta realizar o deploy novamente, porém antes vamos adicionar duas novas opções ao yaml e vamos ver a mágica acontecer. O nosso pod irá ser criado no node elliot-02, onde possui a label disk=SSD.

```
# vim terceiro-deployment.yaml
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  labels:
    run: nginx
  name: terceiro-deployment
  namespace: default
spec:
  replicas: 1
  selector:
    matchLabels:
      run: nginx
  template:
    metadata:
      creationTimestamp: null
      labels:
        run: nginx
        dc: Netherlands
    spec:
      containers:
      - image: nginx
        imagePullPolicy: Always
        name: nginx2
        ports:
        - containerPort: 80
          protocol: TCP
        resources: {}
        terminationMessagePath: /dev/termination-log
        terminationMessagePolicy: File
      dnsPolicy: ClusterFirst
      restartPolicy: Always
      schedulerName: default-scheduler
      securityContext: {}
      terminationGracePeriodSeconds: 30
      nodeSelector:
        disk: SSD


# kubectl create -f terceiro-deployment.yaml
deployment.extensions/terceiro-deployment created
```

```
# kubectl get pods -o wide
NAME                          READY STATUS  RESTARTS  AGE  IP           NODE
primeiro-deployment-56d9...   1/1   Running  0       14m  172.17.0.4 elliot-03
segundo-deployment-869f...    1/1   Running  0       14m  172.17.0.5 elliot-03
terceiro-deployment-59cd... 1/1   Running  0       22s  172.17.0.6 elliot-02
```

Agora imagina as infinitas possibilidades que isso poderá lhe proporcionar… Já estou pensando em várias, como por exemplo se é produção ou não, se consome muita CPU ou muita RAM, se precisa estar em determinado rack e por ae vai. 😃

Simples como voar, não?

kubectl Edit

Agora vamos fazer o seguinte, vamos utilizar o comando Edit para editar nosso primeiro Deployment, digamos que a "quente" com o Pod ainda em execução.

```
# kubectl edit deployment primeiro-deployment
```

Abriu um editor correto?. Vamos alterar o DC, vamos imaginar que esse Deployment agora rodará no DC de Netherlands, precisamos adicionar o Label e o nodeSelector:

```
spec:
  replicas: 1
  selector:
    matchLabels:
      run: nginx
  strategy:
    rollingUpdate:
      maxSurge: 1
      maxUnavailable: 1
    type: RollingUpdate
  template:
    metadata:
      creationTimestamp: null
      labels:
        dc: Netherlands
        app: giropops
        run: nginx
spec:
      containers:
      - image: nginx
```

```
        imagePullPolicy: Always
        name: nginx2
        ports:
        - containerPort: 80
          protocol: TCP
        resources: {}
        terminationMessagePath: /dev/termination-log
        terminationMessagePolicy: File
      dnsPolicy: ClusterFirst
      nodeSelector:
        dc: Netherlands
...

deployment.extensions/primeiro-deployment edited
```

Como podemos ver mudamos o valor do label dc e também modificamos o nodeSelector, onde ele agora subirá no node que tiver a label dc com o valor Netherlands, fácil! 😀

Veja se o resultado foi conforme esperado:

```
# kubectl get pods -l dc=Netherlands -o wide
NAME                      READY   STATUS    RESTARTS   AGE  ..NODE
primeiro-deployment-7..   1/1     Running   0          3m     elliot-03
segundo-deployment-5..    1/1     Running   0          49m
elliot-02
terceiro-deployment-5..   1/1     Running   0          14m
elliot-02
```

Com certeza, esse pod foi criado no node elliot-03, pois havíamos dito que ele possua essa label anteriormente.

ReplicaSet

O ReplicaSet garante a quantidade solicitada de pods e os recursos necessários para um Deployment. Uma vez que o Deployment é criado, é o ReplicaSet que controla a quantidade de pods em execução, caso algum pod seja finalizado, ele que irá detectar e solicitar que outro pod seja executado em seu lugar, garantindo assim a quantidade de réplicas solicitadas.

Vamos criar nosso primeiro ReplicarSet:

```
# vim primeiro-replicaset.yaml
apiVersion: extensions/v1beta1
kind: ReplicaSet
metadata:
```

```
  name: replica-set-primeiro
spec:
  replicas: 3
  template:
    metadata:
      labels:
        system: Giropops
    spec:
      containers:
      - name: nginx
        image: nginx:1.7.9
        ports:
        - containerPort: 80
```

```
# kubectl create -f primeiro-replicaset.yaml
replicaset.extensions/replica-set-primeiro created
```

```
# kubectl get replicaset
NAME                   DESIRED   CURRENT   READY    AGE
replica-set-primeiro   3         3         1        2s
```

Podemos observar os pods em execução:

```
#kubectl get pods
NAME                         READY     STATUS    RESTARTS   AGE
replica-set-primeiro-6drmt   1/1       Running   0          12s
replica-set-primeiro-7j59w   1/1       Running   0          12s
replica-set-primeiro-mg8q9   1/1       Running   0          12s
```

Temos exatamente 3 pods de nginx rodando simultaneamente.
Podemos obter mais informações do nosso ReplicaSet utilizando o comando describe:

```
# kubectl describe rs replica-set-primeiro
Name:         replica-set-primeiro
Namespace:    default
Selector:     system=Giropops
Labels:       system=Giropops
Annotations:  <none>
Replicas:     3 current / 3 desired
Pods Status:  3 Running / 0 Waiting / 0 Succeeded / 0 Failed
Pod Template:
  Labels:  system=Giropops
  Containers:
   nginx:
    Image:        nginx:1.7.9
    Port:         80/TCP
    Host Port:    0/TCP
```

```
    Environment:  <none>
    Mounts:       <none>
  Volumes:        <none>
Events:
  Type    Reason            Age   From                   Message
  ----    ------            ----  ----                   -------
  Normal  SuccessfulCreate  31s   replicaset-controller  Created
pod: replica-set-primeiro-mg8q9
  Normal  SuccessfulCreate  31s   replicaset-controller  Created
pod: replica-set-primeiro-6drmt
  Normal  SuccessfulCreate  31s   replicaset-controller  Created
pod: replica-set-primeiro-7j59w
```

Assim podemos ver todos os pods associados ao ReplicaSet, e se excluirmos um desses Pods, o que será que acontece ?
Vamos testar:

```
# kubectl delete pod replica-set-primeiro-6drmt
pod "replica-set-primeiro-6drmt" deleted
```

Agora vamos verificar novamente os Pods em execução:

```
# kubectl get pods -l system=Giropops
NAME                         READY     STATUS    RESTARTS   AGE
replica-set-primeiro-7j59w   1/1       Running   0          1m
replica-set-primeiro-mg8q9   1/1       Running   0          1m
replica-set-primeiro-s5dz2   1/1       Running   0          15s
```

Percebeu que ele recriou outro Pod ? o ReplicaSet faz com que sempre tenha 3 pods disponíveis.

Vamos alterar para 4 réplicas e recriar o ReplicaSet, para isso vamos utilizar o kubectl edit visto anteriormente assim podemos alterar o ReplicaSet já em execução:

```
# kubectl edit rs replica-set-primeiro
apiVersion: extensions/v1beta1
kind: ReplicaSet
metadata:
  creationTimestamp: 2018-07-05T04:32:42Z
  generation: 2
  labels:
    system: Giropops
  name: replica-set-primeiro
  namespace: default
```

```
  resourceVersion: "471758"
  selfLink:
/apis/extensions/v1beta1/namespaces/default/replicasets/replica-se
t-primeiro
  uid: 753290c1-800c-11e8-b889-42010a8a0002
spec:
  replicas: 4
  selector:
    matchLabels:
      system: Giropops
  template:
    metadata:
      creationTimestamp: null
      labels:
        system: Giropops
...

replicaset.extensions/replica-set-primeiro edited

# kubectl get pods -l system=Giropops
NAME                          READY     STATUS    RESTARTS   AGE
replica-set-primeiro-7j59w    1/1       Running   0          2m
replica-set-primeiro-96hj7    1/1       Running   0          10s
replica-set-primeiro-mg8q9    1/1       Running   0          2m
replica-set-primeiro-s5dz2    1/1       Running   0          1m
```

Veja que ele não cria um deployment para esse replicaset:

```
# kubectl get deployment
No resources found.
```

Agora vamos editar um dos pods e modificar a versão da imagem do Nginx que estamos utilizando no exemplo, vamos alterar de " *image: nginx:1.7.9*" para "*image: nginx:1.15.0*" utilizando o "kubectl edit":

```
# kubectl edit pod replica-set-primeiro-7j59w
apiVersion: extensions/v1beta1
kind: ReplicaSet
metadata:
  name: replica-set-primeiro
...
    spec:
      containers:
      - name: nginx
        image: nginx:1.15.0
        ports:
```

```
        - containerPort: 80
...

pod/replica-set-primeiro-7j59w edited
```

Agora vamos observar novamente os pods, como estão?

```
# kubectl get pods -l system=Giropops
NAME                          READY     STATUS    RESTARTS   AGE
replica-set-primeiro-7j59w    1/1       Running   1          8m
replica-set-primeiro-96hj7    1/1       Running   0          6m
replica-set-primeiro-mg8q9    1/1       Running   0          8m
replica-set-primeiro-s5dz2    1/1       Running   0          7m
```

Aparentemente nada aconteceu concordam? vamos detalhar melhor esse pod que acabamos de alterar.

```
#kubectl describe pod replica-set-primeiro-7j59w
Name:               replica-set-primeiro-7j59w
Namespace:          default
Priority:           0
PriorityClassName:  <none>
Node:               elliot-02/10.138.0.3
Start Time:         Sat, 04 Aug 2018 01:47:56 +0000
Labels:             system=Giropops
Annotations:        <none>
Status:             Running
IP:                 10.46.0.2
Controlled By:      ReplicaSet/replica-set-primeiro
Containers:
  nginx:
    Container ID:
docker://6991b627cf4d6daca039ab9d6336929c0de1fc279c55a451cf9c7304e
1c46504
    Image:          nginx:1.15.0
...
Successfully assigned default/replica-set-primeiro-7j59w to
elliot-02
  Normal  Pulled     9m               kubelet, elliot-02
Container image "nginx:1.7.9" already present on machine
  Normal  Killing    1m               kubelet, elliot-02  Killing
container with id docker://nginx:Container spec hash changed
(3238050430 vs 811632170).. Container will be killed and recreated.
  Normal  Pulling    1m               kubelet, elliot-02  pulling
image "nginx:1.15.0"
```

```
  Normal  Created    1m (x2 over 9m)  kubelet, elliot-02  Created
container
  Normal  Started    1m (x2 over 9m)  kubelet, elliot-02  Started
container
  Normal  Pulled     1m               kubelet, elliot-02
Successfully pulled image "nginx:1.15.0"
```

Como podemos observar ele alterou a imagem do nginx do 1.7.9 para 1.15.0, como o replicaset não tem um deployment ele apenas destruiu o container sem destruir o pod, então a configuração passada manualmente é uma configuração válida, mas caso o pod seja removido o ReplicaSet vai recriá-lo com as configurações originais.

Vamos apagar o pod e ver se realmente acontece isso:

```
# kubectl delete pod replica-set-primeiro-7j59w
pod "replica-set-primeiro-7j59w" delete

# kubectl get pods -l system=Giropops
NAME                         READY     STATUS    RESTARTS   AGE
replica-set-primeiro-96hj7   1/1       Running   0          12m
replica-set-primeiro-mg8q9   1/1       Running   0          14m
replica-set-primeiro-s5dz2   1/1       Running   0          13m
replica-set-primeiro-xzfvg   1/1       Running   0          5s

# kubectl describe pod replica-set-primeiro-xzfvg
Name:               replica-set-primeiro-xzfvg
Namespace:          default
Priority:           0
PriorityClassName:  <none>
Node:               elliot-02/10.138.0.3
Start Time:         Sat, 04 Aug 2018 02:02:35 +0000
Labels:             system=Giropops
Annotations:        <none>
Status:             Running
IP:                 10.46.0.2
Controlled By:      ReplicaSet/replica-set-primeiro
Containers:
  nginx:
    Container ID:
docker://e8b88065640ba3ea346c93bb368ae6b7fb7b1d9507a948d891ca632df
0dfc071
    Image:          nginx:1.7.9
...
```

Olha só , o novo pod foi criado com a imagem configurada no replicaset.
Agora vamos apagar nosso ReplicaSet.

```
# kubectl get rs
NAME                   DESIRED   CURRENT   READY     AGE
replica-set-primeiro   4         4         4         25m

# kubectl delete rs replica-set-primeiro
replicaset.extensions "replica-set-primeiro" deleted
```

DaemonSet

Basicamente a mesma coisa do que o ReplicaSet, com a diferença que quando você utiliza o DaemonSet você não especifica o número de réplicas, ele subirá um pod por node de seu cluster.

É sempre interessante quando criar usar e abusar dos labels, assim você conseguirá ter melhor flexibilidade na distribuição mais adequada de sua aplicação.

Ele é bem interessante para serviços que necessitem rodar em todos os nodes do cluster, como por exemplo, coletores de logs e agente de monitoração.

Vamos criar o nosso primeiro DaemonSet:

```
# vim primeiro-daemonset.yaml
apiVersion: extensions/v1beta1
kind: DaemonSet
metadata:
  name: daemon-set-primeiro
spec:
  template:
    metadata:
      labels:
        system: Strigus
    spec:
      containers:
      - name: nginx
        image: nginx:1.7.9
        ports:
        - containerPort: 80
```

Mas antes vamos permitir que todos os nossos nodes executem pods.

```
# kubectl taint nodes --all node-role.kubernetes.io/master-
```

```
node/elliot-01 untainted
taint "node-role.kubernetes.io/master:" not found
taint "node-role.kubernetes.io/master:" not found
```

Agora podemos criar nosso DaemonSet.

```
# kubectl create -f  primeiro-daemonset.yaml
daemonset.extensions/daemon-set-primeiro created
```

Vamos listar nossos DaemonSet:

```
# kubectl get daemonset
NAME                  DESIRED  CURRENT  READY  UP-TO-DATE ... AGE
daemon-set-primeiro   3        3        3      3              30s

# kubectl describe ds daemon-set-primeiro
Name:           daemon-set-primeiro
Selector:       system=Strigus
Node-Selector:  <none>
Labels:         system=Strigus
Annotations:    <none>
Desired Number of Nodes Scheduled: 3
Current Number of Nodes Scheduled: 3
Number of Nodes Scheduled with Up-to-date Pods: 3
Number of Nodes Scheduled with Available Pods: 3
Number of Nodes Misscheduled: 0
Pods Status:  3 Running / 0 Waiting / 0 Succeeded / 0 Failed
Pod Template:
  Labels:  system=Strigus
  Containers:
   nginx:
    Image:        nginx:1.7.9
    Port:         80/TCP
    Host Port:    0/TCP
    Environment:  <none>
    Mounts:       <none>
  Volumes:        <none>
Events:
  Type    Reason            Age   From                  Message
  ----    ------            ----  ----                  -------
  Normal  SuccessfulCreate  41s    daemonset-controller  Created
pod: daemon-set-primeiro-jl6f5
  Normal  SuccessfulCreate  412    daemonset-controller  Created
pod: daemon-set-primeiro-jh2sp
  Normal  SuccessfulCreate  412    daemonset-controller  Created
pod: daemon-set-primeiro-t9rv9
```

```
# kubectl get pods -o wide
NAME                    READY   STATUS    RESTARTS  AGE  .. NODE
daemon-set-primeiro..  1/1     Running   0         1m
elliot-01
daemon-set-primeiro..  1/1     Running   0         1m
elliot-02
daemon-set-primeiro..  1/1     Running   0         1m
elliot-03
```

Como podemos observar temos um pod por nó rodando nosso daemon-set-primeiro.

Vamos alterar a imagem desse pod diretamente no DaemonSet, usando o comando kubectl set:

```
# kubectl set image ds daemon-set-primeiro nginx=nginx:1.15.0
daemonset.extensions/daemon-set-primeiro image updated
```

Vamos confirmar se a imagem foi realmente alterada:

```
# kubectl describe ds daemon-set-primeiro
Name:           daemon-set-primeiro
Selector:       system=Strigus
Node-Selector:  <none>
Labels:         system=Strigus
Annotations:    <none>
Desired Number of Nodes Scheduled: 3
Current Number of Nodes Scheduled: 3
Number of Nodes Scheduled with Up-to-date Pods: 0
Number of Nodes Scheduled with Available Pods: 3
Number of Nodes Misscheduled: 0
Pods Status:  3 Running / 0 Waiting / 0 Succeeded / 0 Failed
Pod Template:
  Labels:  system=Strigus
  Containers:
   nginx:
    Image:        nginx:1.15.0
    Port:         80/TCP
    Host Port:    0/TCP
    Environment:  <none>
    Mounts:       <none>
  Volumes:        <none>
Events:
  Type    Reason            Age   From                   Message
  ----    ------            ----  ----                   -------
  Normal  SuccessfulCreate  2m    daemonset-controller  Created
pod: daemon-set-primeiro-jl6f5
```

```
  Normal  SuccessfulCreate  2m    daemonset-controller  Created
pod: daemon-set-primeiro-jh2sp
  Normal  SuccessfulCreate  2m    daemonset-controller  Created
pod: daemon-set-primeiro-t9rv9
```

Agora vamos verificar se as imagens dos pods estão atualizadas:

```
# kubectl get pods
NAME                         READY     STATUS    RESTARTS   AGE
daemon-set-primeiro-jh2sp    1/1       Running   0          2m
daemon-set-primeiro-jl6f5    1/1       Running   0          2m
daemon-set-primeiro-t9rv9    1/1       Running   0          2m
```

Como podemos observar não tivemos nenhum restart nos pods.
Vamos verificar a imagem executando em um dos pods.

```
# kubectl describe pod daemon-set-primeiro-jh2sp | grep -i image:
     Image:          nginx:1.7.9
```

Exatamente, Não conseguimos alterar informações do DaemonSet em execução.
E se o pod for deletado?

```
# kubectl delete pod daemon-set-primeiro-jh2sp
pod "daemon-set-primeiro-jh2sp" deleted

# kubectl get pods
NAME                         READY     STATUS    RESTARTS   AGE
NAME                         READY     STATUS    RESTARTS   AGE
daemon-set-primeiro-hp4qc    1/1       Running   0          3s
daemon-set-primeiro-jl6f5    1/1       Running   0          10m
daemon-set-primeiro-t9rv9    1/1       Running   0          10m
```

Vamos listar o novo Pod que foi criado, após deletarmos o Pod antigo:

```
# kubectl describe pod daemon-set-primeiro-hp4qc | grep -i image:
    Image:          nginx:1.15.0
```

Agora um Pod que já estava em execução:

```
# kubectl describe pod daemon-set-primeiro-jl6f5 | grep -i image:
    Image:          nginx:1.7.9
```

Como podemos observar , para atualizar todos os pods do DaemonSet precisamos recriar-lo ou destruir todos os pods relacionado a ele, mas isso não é muito ruim ? sim é

bem ruim para melhorar nossas vidas temos a opção RollingUpdate que vamos ver no próximo capítulo.

Rollouts e Rollbacks

Agora vamos imaginar que essa nossa última edição utilizando o comando "kubectl set" no DaemonSet não foi correta e precisamos voltar a configuração anterior, onde a versão da imagem era outra, como faremos?

É muito simples, para isso existe o *Rollout*. Com ele você pode verificar quais foram as modificações que aconteceram em seu Deployment ou DaemonSet, como se fosse um versionamento. Vejaaaa! (Com a voz do Nelson Rubens)

```
# kubectl rollout history ds daemon-set-primeiro
daemonsets "daemon-set-primeiro"
REVISION  CHANGE-CAUSE
1         <none>
2         <none>
```

Ele irá mostrar duas linhas, a primeira que é a original, com a imagem do nginx:1.7.9 e a segunda já com a imagem nginx:1.15.0. As informações não estão muito detalhada s concordam ?
Veja como verificar os detalhes de cada uma dessas entradas, que são chamadas de revision:

```
# kubectl rollout history ds daemon-set-primeiro --revision=1
daemonsets "daemon-set-primeiro" with revision #1
Pod Template:
  Labels:  system=DaemonOne
  Containers:
   nginx:
    Image: nginx:1.7.9
    Port:  80/TCP
    Host Port:  0/TCP
    Environment:     <none>
    Mounts:     <none>
  Volumes: <none>

# kubectl rollout history ds daemon-set-primeiro --revision=2
daemonsets "daemon-set-primeiro" with revision #2
Pod Template:
  Labels:  system=DaemonOne
  Containers:
   nginx:
    Image: nginx:1.15.0
```

```
    Port:  80/TCP
    Host Port:  0/TCP
    Environment:     <none>
    Mounts:     <none>
  Volumes: <none>
```

Para voltar para a revision desejada, basta fazer o seguinte:

```
# kubectl rollout undo ds daemon-set-primeiro --to-revision=1
daemonset.extensions/daemon-set-primeiro rolled back
```

Perceba que trocamos o history por undo e o revision por to-revision, assim faremos o rollback em nosso DaemonSet, e voltamos a versão da imagem que desejamos. 😃

Para acompanhar o rollout, execute:

```
# kubectl rollout status ds daemon-set-primeiro
```

Vamos confirmar se já estamos executando a nova imagem e um dos nosso pods:

```
# kubectl describe daemon-set-primeiro-hp4qc | grep -i image:
Image:          nginx:1.15.0
```

Sim não funcionou, porque ? porque teremos que matar o Pod para ele ser recriado com as novas configuração.

Vamos afinar esse nosso DamonSet, vamos adicionar o RollingUpdate e esse cara vai atualizar automaticamente os Pods quando houver alguma alteração.
Vamos lá, primeiro vamos remover o DaemonSet adicionar duas novas informações em nosso manifesto yaml e em seguida criar outro DaemonSet em seu lugar.

```
# kubectl delete -f primeiro-daemonset.yaml
daemonset.extensions "daemon-set-primeiro" deleted
```

```
# vim primeiro-daemonset.yaml
apiVersion: extensions/v1beta1
kind: DaemonSet
metadata:
  name: daemon-set-primeiro
spec:
  template:
    metadata:
      labels:
        system: DaemonOne
    spec:
      containers:
```

```
      - name: nginx
        image: nginx:1.7.9
        ports:
        - containerPort: 80
  updateStrategy:
    type: RollingUpdate
```

```
# kubectl create -f primeiro-daemonset.yaml
daemonset.extensions/daemon-set-primeiro created
```

Sucesso, vamos verificar se nosso DaemonSet foi inicializado certinho.

```
# kubectl get daemonset
NAME                 DESIRED   CURRENT   READY   ...   AGE
daemon-set-primeiro  3         3         3       ...   5m
```

```
# kubectl describe ds daemon-set-primeiro
Name:           daemon-set-primeiro
Selector:       system=DaemonOne
Node-Selector:  <none>
Labels:         system=DaemonOne
Annotations:    <none>
Desired Number of Nodes Scheduled: 3
Current Number of Nodes Scheduled: 3
Number of Nodes Scheduled with Up-to-date Pods: 3
Number of Nodes Scheduled with Available Pods: 3
Number of Nodes Misscheduled: 0
Pods Status:  3 Running / 0 Waiting / 0 Succeeded / 0 Failed
Pod Template:
  Labels:  system=DaemonOne
  Containers:
   nginx:
    Image:        nginx:1.7.9
    Port:         80/TCP
    Host Port:    0/TCP
    Environment:  <none>
    Mounts:       <none>
  Volumes:        <none>
Events:
  Type    Reason            Age   From                  Message
  ----    ------            ----  ----                  -------
  Normal  SuccessfulCreate  5m    daemonset-controller  Created
pod: daemon-set-primeiro-52k8k
  Normal  SuccessfulCreate  5m    daemonset-controller  Created
pod: daemon-set-primeiro-6sln2
```

```
  Normal  SuccessfulCreate  5m    daemonset-controller  Created
pod: daemon-set-primeiro-9v2w9
    daemonset-controller  Created pod: daemon-set-primeiro-9dktj
```

Vamos verificar nossa recém adicionada configuração de RollingUpdate:

```
# kubectl get ds daemon-set-primeiro -o yaml | grep -A 2 Strategy
  updateStrategy:
    rollingUpdate:
      maxUnavailable: 1
```

Agora com nosso DaemonSet já configurado, vamos alterar aquela mesma imagem do nginx e ver o que acontece de fato:

```
# kubectl set image ds daemon-set-primeiro nginx=nginx:1.15.0
daemonset.extensions/daemon-set-primeiro image updated
```

Vamos listar o DaemonSet e os Pods para ter certeza de que nada se quebrou.

```
# kubectl get daemonset
NAME                 DESIRED   CURRENT   READY   ...   AGE
daemon-set-primeiro  3         3         3       ...   6m

# kubectl get pods -o wide
NAME                        READY  STATUS    RESTARTS  AGE  NODE
daemon-set-primeiro-7m...   1/1    Running   0         10s
elliot-02
daemon-set-primeiro-j7...   1/1    Running   0         10s
elliot-03
daemon-set-primeiro-v5...   1/1    Running   0         10s
elliot-01
```

Como podemos observar nosso DaemonSet se manteve o mesmo, porém os pods foram recriados, vamos detalhar o DaemonSet para visualizar as alterações realizadas.

```
# kubectl describe ds daemon-set-primeiro
Name:           daemon-set-primeiro
Selector:       system=DaemonOne
Node-Selector:  <none>
Labels:         system=DaemonOne
Annotations:    <none>
Desired Number of Nodes Scheduled: 3
Current Number of Nodes Scheduled: 3
Number of Nodes Scheduled with Up-to-date Pods: 3
Number of Nodes Scheduled with Available Pods: 3
Number of Nodes Misscheduled: 0
Pods Status:  3 Running / 0 Waiting / 0 Succeeded / 0 Failed
Pod Template:
```

```
  Labels:  system=DaemonOne
  Containers:
   nginx:
    Image:        nginx:1.15.0
    Port:         80/TCP
    Host Port:    0/TCP
    Environment:  <none>
    Mounts:       <none>
  Volumes:        <none>
Events:
  Type    Reason            Age   From                   Message
  ----    ------            ----  ----                   -------
  Normal  SuccessfulCreate  8m   daemonset-controller  Created
pod: daemon-set-primeiro-52k8k
  Normal  SuccessfulCreate  8m   daemonset-controller  Created
pod: daemon-set-primeiro-6sln2
  Normal  SuccessfulCreate  8m   daemonset-controller  Created
pod: daemon-set-primeiro-9v2w9
  Normal  SuccessfulDelete  10m   daemonset-controller  Deleted
pod: daemon-set-primeiro-6sln2
  Normal  SuccessfulCreate  1m   daemonset-controller  Created
pod: daemon-set-primeiro-j788v
  Normal  SuccessfulDelete  10m   daemonset-controller  Deleted
pod: daemon-set-primeiro-52k8k
  Normal  SuccessfulCreate  1m   daemonset-controller  Created
pod: daemon-set-primeiro-7mpwr
  Normal  SuccessfulDelete  10m   daemonset-controller  Deleted
pod: daemon-set-primeiro-9v2w9
  Normal  SuccessfulCreate  1m   daemonset-controller  Created
pod: daemon-set-primeiro-v5m47
```

Olha que Bacana , se observamos o campo Events podemos ver que o RollingUpdate matou os pods antigos e recriou com a nova imagem que alteramos utilizando o kubectl set.

Podemos também verificar em um dos Pod se essa alteração realmente aconteceu.

```
# kubectl describe pod daemon-set-primeiro-j788v | grep -i image:
Image:          nginx:1.15.0
```

Viram? Muito sensacional esse negócio de RollingUpdate.
Vamos verificar nosso histórico de modificações:

```
# kubectl rollout history ds daemon-set-primeiro
daemonsets "daemon-set-primeiro"
REVISION  CHANGE-CAUSE
1         <none>
2         <none>
```

Sim temos duas alterações , vamos detalhar para saber qual é qual:

```
# kubectl rollout history ds daemon-set-primeiro --revision=1
daemonsets "daemon-set-primeiro" with revision #1
Pod Template:
  Labels:  system=DaemonOne
  Containers:
   nginx:
    Image: nginx:1.7.9
    Port:  80/TCP
    Host Port:  0/TCP
    Environment:     <none>
    Mounts:     <none>
  Volumes: <none>

# kubectl rollout history ds daemon-set-primeiro --revision=2
daemonsets "daemon-set-primeiro" with revision #2
Pod Template:
  Labels:  system=DaemonOne
  Containers:
   nginx:
    Image: nginx:1.15.0
    Port:  80/TCP
    Host Port:  0/TCP
    Environment:     <none>
    Mounts:     <none>
  Volumes: <none>
```

Agora vamos realizar o RollBack do nosso DaemonSet para a revision 1:

```
# kubectl rollout undo ds daemon-set-primeiro --to-revision=1
daemonset.extensions/daemon-set-primeiro rolled back

# kubectl get pods
NAME                          READY     STATUS    RESTARTS   AGE
daemon-set-primeiro-c2jjk   1/1       Running   0          19s
daemon-set-primeiro-hrn48   1/1       Running   0          19s
daemon-set-primeiro-t6mr9   1/1       Running   0          19s

# kubectl describe pod daemon-set-primeiro-c2jjk | grep -i image:
Image:          nginx:1.7.9
```

Sensacional não?
Deu ruim?
Basta retornar para a outra configuração:

```
# kubectl rollout undo ds daemon-set-primeiro --to-revision=2
daemonset.extensions/daemon-set-primeiro rolled back

# kubectl rollout status ds daemon-set-primeiro
daemon set "daemon-set-primeiro" successfully rolled out

# kubectl get pods
NAME                          READY     STATUS    RESTARTS   AGE
daemon-set-primeiro-jzck9   1/1       Running   0          32s
daemon-set-primeiro-td7h5   1/1       Running   0          29s
daemon-set-primeiro-v5c86   1/1       Running   0          40s

# kubectl describe pod daemon-set-primeiro-jzck9 | grep  -i image:
Image:          nginx:1.15.0
```

Agora vamos deletar nosso DaemonSet.

```
# kubectl delete ds daemon-set-primeiro
daemonset.extensions "daemon-set-primeiro" deleted
```