44Kubernetes 系列（四十一）Kubernetes deployment 一些控制策略

Kubernetes 系列（四十一）Kubernetes deployment 一些控制策略

概述

开发运维实践将经常使用多个部署来管理应用程序部署场景，如“持续部署”、“蓝绿部署”、“金丝雀部署”等。本文提供了扩展和管理容器的实践，这样您就可以完成这些使用多个异构部署的常见场景。

你将会做些什么

- 通过 kubectl 命令进行练习
- 创建 deployment yaml 文件
- 创建, 更新, 以及控制 deployments 副本
- 练习更新 Deployment 以及部署风格

deployments 相关介绍

异构部署通常涉及连接两个或多个不同的基础设施环境或区域，以解决特定的技术或操作需求。根据部署的具体情况，异构部署被称为“混合”、“多云”或“公共-私有”。对于这个实验，异构部署包括跨单个云环境、多个公共云环境(多云)或本地和公共云环境的组合(混合或公共-私有)区域的部署。

在局限于单一环境或区域的部署中，可能会出现各种业务和技术挑战:

- **资源透支** :在任何单一环境中，特别是在企业内部环境中，您可能没有足够的计算、网络和存储资源来满足您的生产需求。
- **有限的地理覆盖** :在单一环境中的部署需要地理上彼此相距很远的人员访问一个部署。他们的交通可能会穿越世界到达一个中心位置。
- **有限的可用性** :web规模的流量模式挑战应用程序保持容错和弹性。
- **厂商锁定** :厂商级别的平台和基础设施抽象可以阻止您移植应用程序。
- **不灵活的资源** :您的资源可能局限于一组特定的计算、存储或网络产品。

异构部署可以帮助解决这些挑战，但是必须使用编程的和确定性的流程和过程来构建它们。一次性的或特别的部署过程可能导致部署或流程脆弱，无法容忍失败。特别进程可能会丢失数据或丢失流量。好的部署过程必须是可重复的，并且使用经过验证的方法来管理供应、配置和维护。

异构部署的三个常见场景是多云部署、前置本地数据和持续集成/持续交付(CI/CD)流程。

下面的练习实践了异构部署的一些常见用例，以及使用Kubernetes和其他基础设施资源的良好架构方法来完成它们。

了解 deployment 对象

让我们从部署开始。首先，让我们看一下Deployment对象。kubectl中的explain命令可以告诉我们关于Deployment对象的信息。

```
kubectl explain deployment
```

我们还可以使用 `--recursive` 选项查看所有字段。

```
kubectl explain deployment --recursive
```

在进行实验时，您可以使用explain命令来帮助您理解Deployment对象的结构和理解各个字段的作用。

```
kubectl explain deployment.metadata.name
```

创建一个 deployment

更新配置文件 `deployments/auth.yaml`：

```
vi deployments/auth.yaml
```

修改 `image`：

```
...
containers:
- name: auth
 image: "kelseyhightower/auth:1.0.0"
...
```

保存 `auth.yaml`：

```
:wq
```

(Output)

```
apiVersion: apps/v1
kind: Deployment
metadata:
 name: auth
spec:
 replicas: 1
 selector:
 matchLabels:
 app: auth
 template:
 metadata:
 labels:
 app: auth
 track: stable
 spec:
 containers:
 - name: auth
 image: "kelseyhightower/auth:1.0.0"
 ports:
 - name: http
 containerPort: 80
 - name: health
 containerPort: 81
...
```

注意部署是如何创建一个副本的，并且它使用的是版本1.0.0的认证容器。

当您运行"kubectl create"命令来创建认证部署时，它将创建一个符合deployment清单中的数据的pod。这意味着我们可以通过改变"副本"字段中指定的数量来缩放Pods的数量。

通过 `kubectl create` 创建:

```
kubectl create -f deployments/auth.yaml
```

检查:

```
kubectl get deployments
```

一旦创建了部署，Kubernetes将为部署创建一个（ReplicaSet）副本集。我们可以验证为我们的部署创建了一个副本集:

```
kubectl get replicasets
```

我们应该看到一个名为auth-xxxxxxx的副本集

最后，我们可以查看作为部署的一部分创建的Pods。单个Pod是在创建replicset时由Kubernetes创建的。

```
kubectl get pods
```

现在是为我们的身份验证部署创建服务的时候了。您已经看到了服务清单文件，因此我们在这里不再深入讨论细节。使用kubectl create命令创建auth服务。

```
kubectl create -f services/auth.yaml
```

现在，执行同样的操作来创建和公开hello Deployment。

```
kubectl create -f deployments/hello.yaml
kubectl create -f services/hello.yaml
```

再一次创建和公开前端部署。

```
kubectl create secret generic tls-certs --from-file tls/
kubectl create configmap nginx-frontend-conf --from-file=nginx/frontend.conf
kubectl create -f deployments/frontend.yaml
kubectl create -f services/frontend.yaml
```

```
kubectl get services frontend
```

为您的服务填充External-IP字段可能需要几秒钟的时间。这很正常。只需每隔几秒钟重新运行上述命令，直到填充该字段。

```
curl -ks https://<EXTERNAL-IP>
```

然后你会得到hello响应。

你也可以使用输出模板特性' kubectl '来使用curl作为一行代码:

```
curl -kshttps://`kubectl get svc frontend -o=jsonpath="{.status.loadBalancer.ingress[0].ip}"`
```

控制 Deployment 副本伸缩

既然已经创建了一个Deployment，我们就可以扩展它了。通过更新spec.replicas字段来实现这一点。您可以再次使用kubectl explain命令查看该字段的解释。

```
kubectl explain deployment.spec.replicas
```

使用kubectl scale命令可以最容易地更新replicas字段:

```
kubectl scale deployment hello --replicas=5
```

注意:它可能需要一分钟左右的所有新的pod启动。

更新部署后，Kubernetes将自动更新相关的副本集，并启动新的Pods，使pod的总数等于5。

确认现在有5个hello Pods正在运行:

```
kubectl get pods | grep hello- | wc -l
```

现在缩小应用程序:

```
kubectl scale deployment hello --replicas=3
```

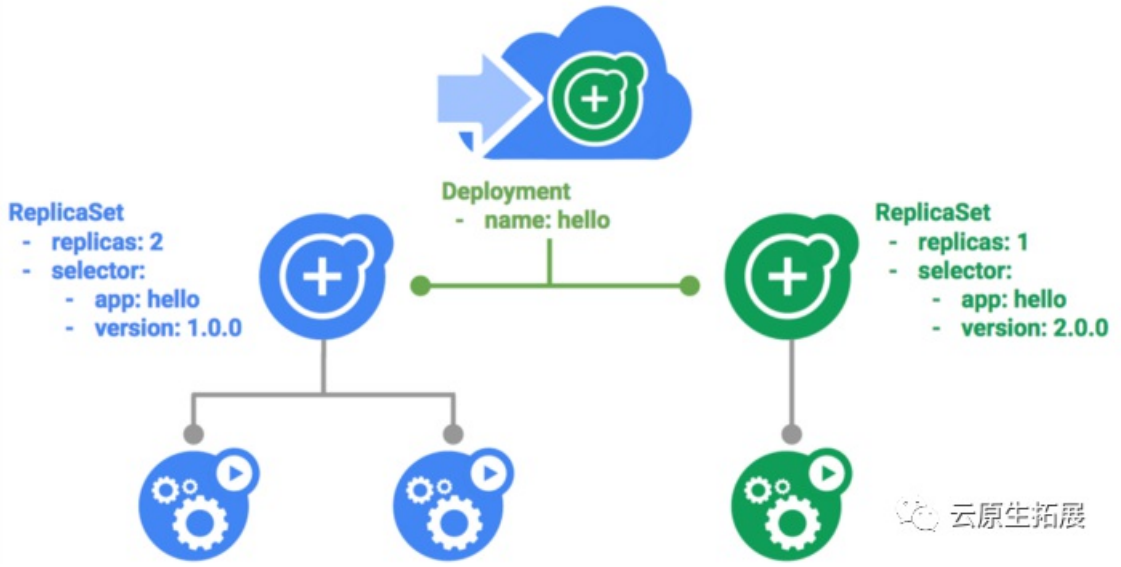
再次，验证您的pod数量是否正确。

```
kubectl get pods | grep hello- | wc -l
```

您了解了Kubernetes部署以及如何管理和扩展一组Pods。

滚动更新

部署支持通过滚动更新机制将镜像更新到新版本。当使用新版本更新部署时，它会创建一个新的副本集，并在减少旧副本集中的副本的同时，缓慢增加新副本集中的副本数量。



触发滚动更新

要更新您的Deployment，运行以下命令:

```
kubectl edit deployment hello
```

将部署中的容器部分中的镜像更改为以下内容:

```
...
containers:
 image: kelseyhightower/hello:2.0.0
...
```

保存并退出。

在编辑器外保存之后，更新的Deployment将被保存到您的集群中，Kubernetes将开始滚动更新。

查看Kubernetes创建的新副本集:

```
kubectl get replicaset
```

您还可以在rollout历史中看到一个新的条目:

```
kubectl rollout history deployment/hello
```

暂停滚动更新

如果检测到正在运行的rollout有问题，请暂停它以停止更新。现在就试试吧:

```
kubectl rollout pause deployment/hello
```

验证rollout的当前状态:

```
kubectl rollout status deployment/hello
```

你也可以直接在Pods上验证这一点:

```
kubectl get pods -o jsonpath --template='{range .items[*]}{.metadata.name}{"\t"}{"\t"}{.spec.containers[0]
```

恢复滚动更新

rollout 被暂停，这意味着一些pod处于新版本，而一些pod处于旧版本。我们可以使用resume命令继续展开。

```
kubectl rollout resume deployment/hello
```

完成rollout后，运行status命令时应该会看到以下内容。

```
kubectl rollout status deployment/hello
```

(Output)

```
deployment "hello" successfully rolled out
```

回滚更新

假设在新版本中检测到一个错误。由于新版本被认为有问题，任何连接到新Pods的用户都会遇到这些问题。

您可能想要回滚到以前的版本，这样您就可以研究一下，然后发布一个正确修复的版本。

使用rollout命令回退到上一个版本:

```
kubectl rollout undo deployment/hello
```

在历史记录中验证回滚:

```
kubectl rollout history deployment/hello
```

最后，验证所有的Pods已经回滚到它们以前的版本:

```
kubectl get pods -o jsonpath --template='{range .items[*]}{.metadata.name}{"\t"}{"\t"}{.spec.containers[0]
```

太棒了!您了解了Kubernetes部署的滚动更新，以及如何在不停机的情况下更新应用程序。

金丝雀部署

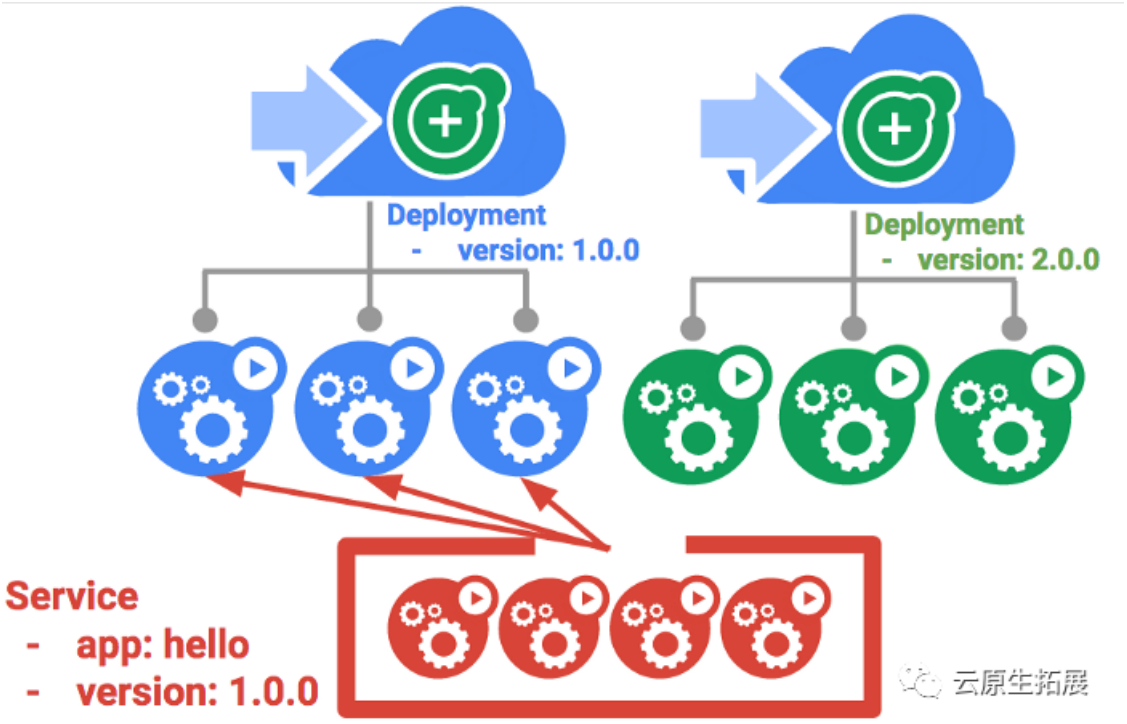
当您希望用您的用户子集在生产环境中测试新部署时，请使用canary部署。Canary部署允许您向一小部分用户发布更改，以降低与新版本相关的风险。

蓝绿部署

滚动更新是理想的，因为它允许您以最小的开销、最小的性能影响和最小的停机时间缓慢地部署应用程序。在某些情况下，只有在新版本完全部署之后才修改负载均衡器以指向该新版本是有益的。在这种情况下，蓝绿色部署是可行的。

Kubernetes通过创建两个独立的部署来实现这一点;一个是旧的"蓝色"版本，一个是新的"绿色"版本。对于"蓝色"版本，使用现有

的“hello”部署。部署将通过充当路由器的Service进行访问。一旦新的“绿色”版本启动并运行，您将通过更新服务切换到使用该版本。



蓝绿部署的一个主要缺点是，您需要在集群中至少拥有托管应用程序所需的2倍资源。在同时部署应用程序的两个版本之前，请确保您的集群中有足够的资源。

关于service

使用现有的hello服务，但更新它，使它有一个选择器' app:hello '， ' version: 1.0.0 '。选择器将匹配现有的“蓝色”部署。但是它不会匹配“绿色”部署，因为它将使用不同的版本。

首先，更新服务:

```
kubectl apply -f services/hello-blue.yaml
```

注意:忽略“ `resource service/hello is missing` ”的警告，这是自动修补的。

使用蓝绿部署进行更新

为了支持蓝绿的部署风格，我们将为新版本创建一个新的“绿色”部署。绿色部署更新版本标签和镜像路径。

```
apiVersion: apps/v1
kind: Deployment
metadata:
 name: hello-green
spec:
 replicas: 3
 selector:
 matchLabels:
 app: hello
 template:
 metadata:
 labels:
 app: hello
 track: stable
 version: 2.0.0
 spec:
 containers:
 - name: hello
 image: kelseyhightower/hello:2.0.0
 ports:
 - name: http
 containerPort: 80
 - name: health
 containerPort: 81
 resources:
 limits:
 cpu: 0.2
 memory: 10Mi
 livenessProbe:
 httpGet:
 path: /healthz
 port: 81
 scheme: HTTP
 initialDelaySeconds: 5
 periodSeconds: 15
 timeoutSeconds: 5
 readinessProbe:
 httpGet:
 path: /readiness
 port: 81
 scheme: HTTP
 initialDelaySeconds: 5
 timeoutSeconds: 1
```

创建deployment:

```
kubectl create -f deployments/hello-green.yaml
```

一旦你有了绿色部署并且它已经正常启动，验证当前版本的1.0.0仍然在使用:

```
curl -kshttps://`kubectl get svc frontend -o=jsonpath="{.status.loadBalancer.ingress[0].ip}"`/version
```

现在，更新服务以指向新版本:

```
kubectl apply -f services/hello-green.yaml
```

当服务更新时，将立即使用“绿色”部署。现在可以验证新版本是否一直在使用。

```
curl -kshttps://`kubectl get svc frontend -o=jsonpath="{.status.loadBalancer.ingress[0].ip}"`/version
```

蓝绿回滚

如果需要，可以用相同的方法回滚到旧版本。当“蓝色”部署仍在运行时，只需将服务更新回旧版本。

```
kubectl apply -f services/hello-blue.yaml
```

一旦您更新了服务，您的回滚就成功了。再次确认现在使用的版本是否正确:

```
curl -kshttps://`kubectl get svc frontend -o=jsonpath="{.status.loadBalancer.ingress[0].ip}"`/version
```

你做到了!您了解了蓝绿部署以及如何将更新部署到需要一次性切换所有版本的应用程序。

欢迎关注我的公众号“ **云原生拓展** ”，原创技术文章第一时间推送。