73Kubernetes 系列（六十六）零停机发布模式

说明

正如软件的本质一样，总有一天需要部署错误修复或新功能。任何运营团队都会证明发布充满担忧，这是因为执行软件发布可能会引入新的错误，甚至导致计划外中断。发布软件与运营团队的目标有些适得其反，即维护平台的稳定性。多年来，我从同事那里听到过一些短语（甚至是我自己说的），例如：

> “此版本是否经过测试”

> “回滚过程是什么”

> “我们能多快修补这个”

对于在 Kubernetes 集群中运行的工作负载，我将描述一种方式，您可以在其中发布更新的容器镜像，同时最大限度地降低上述风险。

背景

蓝/绿部署策略是一种在不停机的情况下更新应用程序的方法。其背后的想法是拥有一组匹配的基础设施，一个当前处于活动状态（“蓝色”），另一个正在运行更新的代码并且不主动接收流量（“绿色”）。为避免混淆， blue 始终引用当前的实时基础设施; green 是为更新的非实时基础设施保留的。

虽然能够创建不同的基础设施是执行零停机发布的关键，但我们仍然需要知道如何允许外部客户端访问更新的应用程序，而无需客户端在每次发布时重新配置。一种选择可能是更新DNS并指向新的IP地址，但是由于缓存，TTL和DNS传播需要应对，这将有些问题。

我建议解决这个问题的是使用 Kubernetes Ingress 并利用路由规则将流量定向到蓝色和绿色应用程序的单独后端服务。

Ingress

Ingress 是一种 Kubernetes 资源，它将 HTTP（s） 路由从集群外部公开到集群内的服务，流量路由由 ingress 资源上定义的规则控制。必须先在群集内运行 ingress 控制器，然后才能创建 ingress 对象。对于此示例，我们将使用 Nginx Ingress 控制器。

方案

成功安装 Nginx Ingress 控制器后，您需要创建一个 Ingress，该 Ingress 将启用我们讨论过的蓝/绿策略：

```yaml
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  annotations:
    nginx.ingress.kubernetes.io/rewrite-target: /$2
    nginx.ingress.kubernetes.io/use-regex: "true"
  labels:
    app: nginx
  name: nginx
  namespace: default
spec:
  ingressClassName: nginx
  rules:
  - host: example.com
    http:
      paths:
      - backend:
          service:
            name: nginx-v1.22.0
            port:
              number: 80
        path: /*
        pathType: ImplementationSpecific
      - backend:
          service:
            name: nginx-v1.24.0
            port:
              number: 80
        path: /green(/|$)(.*)
        pathType: ImplementationSpecific
```

有了这个清单，我们说的是，创建一个 Ingress，并为任何发往路径 `/*` 的流量直接发送到名为 `nginx-v1.22.0` 的服务（这是我们当前的活动版本）;发往 `/green(/|$)(.*)` 的任何流量都直接流向名为 `nginx-v1.24.0` 的服务（这是我们的入站版本）。在第二条路径中，我们使用正则表达式捕获组首先匹配行尾字符或 `/`，第二个捕获组 `(.*)` 是简单的通配符匹配。第二个捕获组捕获的任何字符都将分配给占位符 `$2`，然后用作 `rewrite-target` 注释中的参数。

例如：

- `example.com/green` 重写为 `example.com/`
- `example.com/green/` 重写为 `example.com/`
- `example.com/green/new` 重写为 `example.com/new`

这样做的好处是开发团队不需要在应用程序中添加任何其他路由来处理此行为，因为请求是在入口点重写的。

我们现在需要创建与入口规则对应的后端服务和部署

```yaml
---
apiVersion: v1
kind: Service
metadata:
  labels:
    app: nginx
  name: nginx-v1.22.0
  namespace: default
spec:
  ports:
  - name: nginx-80
    port: 80
    protocol: TCP
    targetPort: 80
  selector:
    app: nginx
    revision: v1.22.0
  sessionAffinity: None
  type: ClusterIP
---
apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    app: nginx
    revision: v1.22.0
  name: nginx-v1.22.0
  namespace: default
spec:
  replicas: 1
  selector:
    matchLabels:
      app: nginx
      revision: v1.22.0
  strategy:
    rollingUpdate:
      maxSurge: 25%
      maxUnavailable: 25%
    type: RollingUpdate
  template:
    metadata:
      labels:
        app: nginx
        revision: v1.22.0
      name: nginx
      namespace: default
    spec:
      automountServiceAccountToken: true
      containers:
      - image: nginx:1.22.0
```

```
        name: nginx
        volumeMounts:
        - mountPath: /usr/share/nginx/html/index.html
          name: config
          subPath: index.html
      serviceAccount: nginx
      volumes:
      - configMap:
          defaultMode: 420
          items:
          - key: blue
            path: index.html
          name: nginx-config
          optional: false
        name: config
---
apiVersion: v1
kind: Service
metadata:
  labels:
    app: nginx
    revision: v1.24.0
  name: nginx-v1.24.0
  namespace: default
spec:
  ports:
  - name: nginx-80
    port: 80
    protocol: TCP
    targetPort: 80
  selector:
    app: nginx
    revision: v1.24.0
  sessionAffinity: None
  type: ClusterIP
---
apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    app: nginx
    revision: v1.24.0
  name: nginx-v1.24.0
  namespace: default
spec:
  replicas: 1
  selector:
    matchLabels:
      app: nginx
      revision: v1.24.0
  strategy:
```

```
    rollingUpdate:
      maxSurge: 25%
      maxUnavailable: 25%
    type: RollingUpdate
  template:
    metadata:
      labels:
        app: nginx
        revision: v1.24.0
      name: nginx
      namespace: default
    spec:
      automountServiceAccountToken: true
      containers:
      - image: nginx:1.24.0
        name: nginx
        volumeMounts:
        - mountPath: /usr/share/nginx/html/index.html
          name: config
          subPath: index.html
      serviceAccount: nginx
      volumes:
      - configMap:
          defaultMode: 420
          items:
          - key: green
            path: index.html
          name: nginx-config
          optional: false
        name: config
```

工作负载的蓝色和绿色版本现在彼此并排运行，单独的服务对象将流量定向到各自部署中的 Pod。我们可以导航到 example.com/foo 访问蓝色版本，导航到 example.com/green/foo 访问绿色版本。这提供了在不中断实时流量的情况下执行任何功能测试或回归测试的能力，从而在将流量切换到更新的工作负载之前建立对候选发布的信心。

测试完成后，您有 2 个可能的步骤，要么测试取得了巨大的成功，您可以提升发布或测试失败，并且您需要更多的开发时间来修复发生的任何错误/回归。在后者的情况下，没有遭受服务中断，从而使运营团队满意

要协调系统，只需更新 Ingress 资源以指向更新的服务即可，即

```yaml
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  annotations:
    nginx.ingress.kubernetes.io/rewrite-target: /$2
    nginx.ingress.kubernetes.io/use-regex: "true"
  labels:
    app: nginx
  name: nginx
  namespace: default
spec:
  ingressClassName: nginx
  rules:
  - host: example.com
    http:
      paths:
      - backend:
          service:
            name: nginx-v1.24.0
            port:
              number: 80
        path: /*
        pathType: ImplementationSpecific
```

总结

虽然蓝/绿部署策略可有效降低计划外中断的风险并保持正常运行时间统计信息，但它并不适合所有工作负载。数据库增加了复杂性，您应确保任何数据库迁移都完全向后兼容。