Docker+K8S+DevOps 微服务架构师

学神 IT 教育：从零基础到实战，从入门到精通！

联系方式：

学神 IT 教育官方网站: http://www.xuegod.cn

学神 K8S 精英学习 11 群 QQ 群: 957231097

学习顾问：小语老师　　学习顾问：边边老师　　学神微信公众号

微信扫码添加学习顾问微信，同时扫码关注学神公众号了解最新行业动态，获取更多学习资料及答疑就业服务！

第 9 章：k8s 配置管理中心 Configmap 和 Secret

本节所讲内容：

9.1 Configmap 概述

9.2 Configmap 创建方法

9.3 使用 Configmap

实战 1：基于 Istio 的灰度发布

实战 2：卸载 istio 集群

实战 3：istio 核心功能演示

实战 1：基于 Istio 的灰度发布

什么是灰度发布?

灰度发布也叫金丝雀部署 ，是指通过控制流量的比例，实现新老版本的逐步更替。比如对于服务 A 有 version1、 version2 两个版本 ， 当前两个版本同时部署，但是 version1 比例 90% ， version2 比例 10% ，看运行效果，如果效果好逐步调整 流量占比 80～20 ， 70～30 ……10～90 ， 0，100 ， 最终 version1 版本下线。

灰度发布的特点

1）新老板共存

2）可以实时根据反馈动态调整占比

3）理论上不存在服务完全宕机的情况。

4）适合于服务的平滑升级与动态更新。

实战演练-使用 istio 进行金丝雀发布

（1）创建金丝雀服务

```
[root@xuegod64 ~]# docker load -i canary-v1.tar.gz
[root@xuegod64 ~]# docker load -i canary-v2.tar.gz
```

[root@xuegod63 ~]# cat deployment.yaml，内容如下：

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: appv1
  labels:
    app: v1
spec:
  replicas: 1
  selector:
    matchLabels:
      app: v1
      apply: canary
  template:
    metadata:
```

```
        labels:
          app: v1
          apply: canary
      spec:
        containers:
        - name: nginx
          image: xuegod/canary:v1
          ports:
          - containerPort: 80
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: appv2
  labels:
    app: v2
spec:
  replicas: 1
  selector:
    matchLabels:
      app: v2
      apply: canary
  template:
    metadata:
      labels:
        app: v2
        apply: canary
    spec:
      containers:
      - name: nginx
        image: xuegod/canary:v2
        ports:
        - containerPort: 80
```

更新：

```
[root@xuegod63 ~]# kubectl apply -f deployment.yaml
```

（2）创建 service

[root@xuegod63 ~]# cat service.yaml 文件，内容如下：

```
apiVersion: v1
kind: Service
metadata:
  name: canary
  labels:
```

```
    apply: canary
spec:
  selector:
    apply: canary
  ports:
    - protocol: TCP
      port: 80
      targetPort: 80
```

更新 service.yaml 文件

```
[root@xuegod63 ~]# kubectl apply -f service.yaml
```

（3）创建 gateway

[root@xuegod63 ~]# cat gateway.yaml 文件，内容如下：

```
apiVersion: networking.istio.io/v1alpha3
kind: Gateway
metadata:
  name: canary-gateway
spec:
  selector:
    istio: ingressgateway
  servers:
  - port:
      number: 80
      name: http
      protocol: HTTP
    hosts:
    - "*"
```

更新 gateway.yaml

```
[root@xuegod63 ~]# kubectl apply -f gateway.yaml
```

（4）创建 virtualservice

[root@xuegod63 ~]# cat virtual.yaml，内容如下：

```
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: canary
spec:
  hosts:
  - "*"
  gateways:
  - canary-gateway
  http:
```

```
  - route:
    - destination:
        host: canary.default.svc.cluster.local
        subset: v1
      weight: 90
    - destination:
        host: canary.default.svc.cluster.local
        subset: v2
      weight: 10
---
apiVersion: networking.istio.io/v1alpha3
kind: DestinationRule
metadata:
  name: canary
spec:
  host: canary.default.svc.cluster.local
  subsets:
  - name: v1
    labels:
      app: v1
  - name: v2
    labels:
      app: v2
```

更新 virtual.yaml 文件

```
[root@xuegod63 ~]# kubectl apply -f virtual.yaml
```

（5）获取 Ingress_port:

```
kubectl -n istio-system get service istio-ingressgateway -o
jsonpath='{.spec.ports[?(@.name=="http2")].nodePort}'
```

显示结果是 31362

（6）验证金丝雀发布效果：

```
[root@xuegod63 ~]# for i in `seq 1 100`; do curl 192.168.1.63:31362;done > 1.txt
```

打开 1.txt 可以看到结果有 90 次出现 v1，10 次出现 canary-v2,符合我们预先设计的流量走向。

实战 2：卸载 istio 集群

暂时不执行，记住这个命令即可

```
[root@xuegod63 ~]# istioctl manifest generate --set profile=demo | kubectl delete -f -
```

实战 3：istio 核心功能演示

1 断路器

官网：

https://istio.io/latest/zh/docs/tasks/traffic-management/circuit-breaking/

断路器是创建弹性微服务应用程序的重要模式。断路器使应用程序可以适应网络故障和延迟等网络不良影响。

测试断路器：

1、在 k8s 集群创建后端服务

```
[root@xuegod63 ~]# cd istio-1.8.1
[root@xuegod63 istio-1.8.1]# cat samples/httpbin/httpbin.yaml
apiVersion: v1
kind: ServiceAccount
metadata:
  name: httpbin
---
apiVersion: v1
kind: Service
metadata:
  name: httpbin
  labels:
    app: httpbin
    service: httpbin
spec:
  ports:
  - name: http
    port: 8000
    targetPort: 80
  selector:
    app: httpbin
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: httpbin
spec:
  replicas: 1
  selector:
    matchLabels:
      app: httpbin
      version: v1
  template:
```

```
    metadata:
      labels:
        app: httpbin
        version: v1
    spec:
      serviceAccountName: httpbin
      containers:
      - image: docker.io/kennethreitz/httpbin
        imagePullPolicy: IfNotPresent
        name: httpbin
        ports:
        - containerPort: 80
```

#把 httpbin.tar.gz 上传到 xuegod64 节点，手动解压：

```
[root@xuegod64 ~]# docker load -i httpbin.tar.gz
[root@xuegod63 istio-1.10.1]# kubectl apply -f samples/httpbin/httpbin.yaml
```

#该 httpbin 应用程序充当后端服务。

2、配置断路器

#创建一个目标规则，在调用 httpbin 服务时应用断路器设置：

```
[root@xuegod63 istio-1.8.1]# vim destination.yaml
apiVersion:  networking.istio.io/v1beta1
kind: DestinationRule
metadata:
  name: httpbin
spec:
  host: httpbin
  trafficPolicy:
    connectionPool:
```

#连接池（TCP | HTTP）配置，例如：连接数、并发请求等

```
      tcp:
        maxConnections: 1
```

#TCP 连接池中的最大连接请求数，当超过这个值，会返回 503 代码。如两个请求过来，就会有一个请求返回 503。

```
      http:
        http1MaxPendingRequests: 1
```

#连接到目标主机的最大挂起请求数，也就是待处理请求数。这里的目标指的是 virtualservice 路由规则中配置的 destination。

```
        maxRequestsPerConnection: 1
```

#连接池中每个连接最多处理 1 个请求后就关闭，并根据需要重新创建连接池中的连接

```
    outlierDetection:
```

#异常检测配置，传统意义上的熔断配置，即对规定时间内服务错误数的监测

```
      consecutiveGatewayErrors: 1
```

#连续错误数 1，即连续返回 502-504 状态码的 Http 请求错误数

```
        interval: 1s
    #错误异常的扫描间隔 1s，即在 interval（1s）内连续发生 consecutiveGatewayErrors（1）个错误，则触发服务熔断
        baseEjectionTime: 3m
    #基本驱逐时间 3 分钟，实际驱逐时间为 baseEjectionTime*驱逐次数
        maxEjectionPercent: 100
    #最大驱逐百分比 100%

    [root@xuegod63 istio-1.10.1]# kubectl apply -f destination.yaml
    destinationrule.networking.istio.io/httpbin created
```

3、添加客户端访问 httpbin 服务

创建一个客户端以将流量发送给 httpbin 服务。该客户端是一个简单的负载测试客户端，Fortio 可以控制连接数，并发数和 HTTP 调用延迟。使用此客户端来“跳闸”在 DestinationRule 中设置的断路器策略。

```
    #通过执行下面的命令部署 fortio 客户端：
    #把 fortio.tar.gz 上传到 xuegod64 节点，手动解压：
    [root@xuegod64 ~]# docker load -i fortor.tar.gz
    [root@xuegod63 istio-1.10.1]# kubectl apply -f  samples/httpbin/sample-client/fortio-deploy.yaml

    #通过 kubectl 执行下面的命令，使用 fortio 客户端工具调用 httpbin：
    [root@xuegod63 istio-1.10.1]# kubectl get pods
    NAME                              READY   STATUS    RESTARTS   AGE
    appv1-77b5cbd5cc-bmch2            2/2     Running   0          28m
    appv2-f78cb577-n7rhc              2/2     Running   0          28m
    details-v1-847c7999fb-htd2z       2/2     Running   0          28m
    fortio-deploy-576dbdfbc4-z28m7    2/2     Running   0          3m32s
    httpbin-74fb669cc6-hqtzl          2/2     Running   0          15m
    productpage-v1-5f7cf79d5d-6d41x   2/2     Running   0          28m
    ratings-v1-7c46bc6f4d-sfqnz       2/2     Running   0          28m
    reviews-v1-549967c688-pr8gh       2/2     Running   0          28m
    reviews-v2-cf9c5bfcd-tn5z5        2/2     Running   0          28m
    reviews-v3-556dbb4456-dxt4r       2/2     Running   0          28m

    [root@xuegod63 istio-1.10.1]# kubectl exec  fortio-deploy-576dbdfbc4-z28m7   -c fortio -- /usr/bin/fortio curl  http://httpbin:8000/get
    #显示如下：
    HTTP/1.1 200 OK
    server: envoy
    date: Mon, 03 May 2021 02:28:06 GMT
    content-type: application/json
    content-length: 622
    access-control-allow-origin: *
```

```
access-control-allow-credentials: true
x-envoy-upstream-service-time: 2

{
  "args": {},
  "headers": {
    "Content-Length": "0",
    "Host": "httpbin:8000",
    "User-Agent": "fortio.org/fortio-1.11.3",
    "X-B3-Parentspanid": "4631e62a6cd0b167",
    "X-B3-Sampled": "1",
    "X-B3-Spanid": "6d20afff1671aa89",
    "X-B3-Traceid": "6f4ddb61363d04d54631e62a6cd0b167",
    "X-Envoy-Attempt-Count": "1",
    "X-Forwarded-Client-Cert":
"By=spiffe://cluster.local/ns/default/sa/httpbin;Hash=498edf0dcb7f6e74f40735869a9912eca62d61fb21dbc190943c1c19dbf01c18;Subject=\"\";URI=spiffe://cluster.local/ns/default/sa/default"
  },
  "origin": "127.0.0.1",
  "url": "http://httpbin:8000/get"
}
```

4、触发断路器

在 DestinationRule 设置中，指定了 maxConnections: 1 和 http1MaxPendingRequests: 1。这些规则表明，如果超过一个以上的连接并发请求，则 istio-proxy 在为进一步的请求和连接打开路由时，应该会看到下面的情况 。

以两个并发连接（-c 2）和发送 20 个请求（-n 20）调用服务：

```
[root@xuegod63 istio-1.10.1]# kubectl exec -it fortio-deploy-576dbdfbc4-z28m7  -c fortio -- /usr/bin/fortio load  -c 2 -qps 0 -n 20 -loglevel Warning http://httpbin:8000/get
```

#显示如下：

```
02:31:00 I logger.go:127> Log level is now 3 Warning (was 2 Info)
Fortio 1.11.3 running at 0 queries per second, 6->6 procs, for 20 calls: http://httpbin:8000/get
Starting at max qps with 2 thread(s) [gomax 6] for exactly 20 calls (10 per thread + 0)
02:31:00 W http_client.go:693> Parsed non ok code 503 (HTTP/1.1 503)
02:31:00 W http_client.go:693> Parsed non ok code 503 (HTTP/1.1 503)
02:31:00 W http_client.go:693> Parsed non ok code 503 (HTTP/1.1 503)
02:31:00 W http_client.go:693> Parsed non ok code 503 (HTTP/1.1 503)
02:31:00 W http_client.go:693> Parsed non ok code 503 (HTTP/1.1 503)
02:31:00 W http_client.go:693> Parsed non ok code 503 (HTTP/1.1 503)
02:31:00 W http_client.go:693> Parsed non ok code 503 (HTTP/1.1 503)
02:31:00 W http_client.go:693> Parsed non ok code 503 (HTTP/1.1 503)
02:31:00 W http_client.go:693> Parsed non ok code 503 (HTTP/1.1 503)
```

```
02:31:00 W http_client.go:693> Parsed non ok code 503 (HTTP/1.1 503)
02:31:00 W http_client.go:693> Parsed non ok code 503 (HTTP/1.1 503)
02:31:00 W http_client.go:693> Parsed non ok code 503 (HTTP/1.1 503)
02:31:00 W http_client.go:693> Parsed non ok code 503 (HTTP/1.1 503)
02:31:00 W http_client.go:693> Parsed non ok code 503 (HTTP/1.1 503)
02:31:00 W http_client.go:693> Parsed non ok code 503 (HTTP/1.1 503)
02:31:00 W http_client.go:693> Parsed non ok code 503 (HTTP/1.1 503)
Ended after 69.506935ms : 20 calls. qps=287.74
Aggregated Function Time : count 20 avg 0.0054352091 +/- 0.01077 min 0.000474314 max
0.04968864 sum 0.108704183
# range, mid point, percentile, count
>= 0.000474314 <= 0.001 , 0.000737157 , 35.00, 7
> 0.001 <= 0.002 , 0.0015 , 50.00, 3
> 0.002 <= 0.003 , 0.0025 , 65.00, 3
> 0.004 <= 0.005 , 0.0045 , 75.00, 2
> 0.005 <= 0.006 , 0.0055 , 85.00, 2
> 0.007 <= 0.008 , 0.0075 , 90.00, 1
> 0.016 <= 0.018 , 0.017 , 95.00, 1
> 0.045 <= 0.0496886 , 0.0473443 , 100.00, 1
# target 50% 0.002
# target 75% 0.005
# target 90% 0.008
# target 99% 0.0487509
# target 99.9% 0.0495949
Sockets used: 16 (for perfect keepalive, would be 2)
Jitter: false
Code 200 : 4 (20.0 %)
Code 503 : 16 (80.0 %)
#只有 20%成功了，其余的都断开了

Response Header Sizes : count 20 avg 46 +/- 92 min 0 max 230 sum 920
Response Body/Total Sizes : count 20 avg 292.8 +/- 279.6 min 153 max 852 sum 5856
All done 20 calls (plus 0 warmup) 5.435 ms avg, 287.7 qps
```

2 超时

在生产环境中经常会碰到由于调用方等待下游的响应过长，堆积大量的请求阻塞了自身服务，造成雪崩的情况，通过通过超时处理来避免由于无限期等待造成的故障，进而增强服务的可用性，Istio 使用虚拟服务来优雅实现超时处理。

下面例子模拟客户端调用 nginx，nginx 将请求转发给 tomcat。nginx 服务设置了超时时间为 2 秒，如果超出这个时间就不在等待，返回超时错误。tomcat 服务设置了响应时间延迟 10 秒，任何请求都需要等待 10 秒后才能返回。client 通过访问 nginx 服务去反向代理 tomcat 服务，由于 tomcat 服务需要 10 秒后才能返回，但 nginx 服务只等待 2 秒，所以客户端会提示超时错误。

```
#把 busybox.tar.gz、 nginx.tar.gz、 tomcat-app.tar.gz 上传到 xuegod64 节点，手动解压：
[root@xuegod64 ~]# docker load -i nginx-appine.tar.gz
[root@xuegod64 ~]# docker load -i busybox.tar.gz
[root@xuegod64 ~]# docker load -i tomcat-app.tar.gz

[root@xuegod63 ~]# mkdir /root/timeout
[root@xuegod63 ~]# cd /root/timeout/
[root@xuegod63 timeout]# cat nginx-deployment.yaml
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx
  labels:
    server: nginx
    app: web
spec:
  replicas: 1
  selector:
    matchLabels:
      server: nginx
      app: web
  template:
    metadata:
      name: nginx
      labels:
        server: nginx
        app: web
    spec:
      containers:
      - name: nginx
        image: nginx:1.14-alpine
        imagePullPolicy: IfNotPresent
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: tomcat
  labels:
    server: tomcat
    app: web
spec:
  replicas: 1
  selector:
```

```
    matchLabels:
      server: tomcat
      app: web
  template:
    metadata:
      name: tomcat
      labels:
        server: tomcat
        app: web
    spec:
      containers:
      - name: tomcat
        image: docker.io/kubeguide/tomcat-app:v1
        imagePullPolicy: IfNotPresent

[root@xuegod63 timeout]# cat nginx-tomcat-svc.yaml
---
apiVersion: v1
kind: Service
metadata:
  name: nginx-svc
spec:
  selector:
    server: nginx
  ports:
  - name: http
    port: 80
    targetPort: 80
    protocol: TCP
---
apiVersion: v1
kind: Service
metadata:
  name: tomcat-svc
spec:
  selector:
    server: tomcat
  ports:
  - name: http
    port: 8080
    targetPort: 8080
    protocol: TCP

[root@xuegod63 timeout]# cat virtual-tomcat.yaml
```

```
---
apiVersion: networking.istio.io/v1beta1
kind: VirtualService
metadata:
  name: nginx-vs
spec:
  hosts:
  - nginx-svc
  http:
  - route:
    - destination:
        host: nginx-svc
    timeout: 2s
---
apiVersion: networking.istio.io/v1beta1
kind: VirtualService
metadata:
  name: tomcat-vs
spec:
  hosts:
  - tomcat-svc
  http:
  - fault:
      delay:
        percentage:
          value: 100
        fixedDelay: 10s
    route:
    - destination:
        host: tomcat-svc
```

#virtual-tomcat.yaml 资源清单重点知识讲解

第一：故障注入：

```
http:
- fault:
    delay:
      percentage:
      value: 100
   fixedDelay: 10s
```

该设置说明每次调用 tomcat-svc 的 k8s service，都会延迟 10s 才会调用。

第二：调用超时：

```
hosts:
- nginx-svc
  http:
```

```
    - route:
      - destination:
        host: nginx-svc
      timeout: 2s
```

该设置说明调用 nginx-svc 的 k8s service，请求超时时间是 2s。

#部署 tomcat、nginx 服务

需要对 nginx-deployment.yaml 资源文件进行 Istio 注入，将 nginx、tomcat 都放入到网格中。可以采用手工注入 Istio 方式。

```
[root@xuegod63 timeout]# kubectl apply -f nginx-deployment.yaml
```

执行成功后，通过 kubectl get pods 查看 Istio 注入情况：

```
[root@xuegod63 timeout]# kubectl get pods
NAME                               READY   STATUS             RESTARTS   AGE
nginx-tomcat-7dd6f74846-48g9f      2/2     Running    0          6m36s
tomcat-86ddb8f5c9-h6jdl            2/2     Running             0          53s
```

#部署 nginx 和 tomcat 的 service

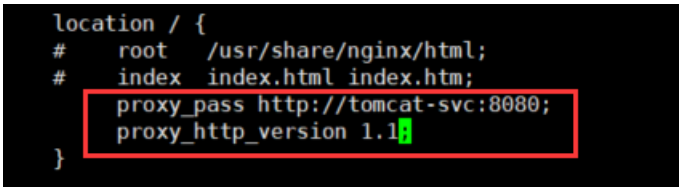
```
[root@xuegod63 timeout]# kubectl apply -f nginx-tomcat-svc.yaml
```

#部署虚拟服务

```
[root@xuegod63 timeout]# kubectl apply -f virtual-tomcat.yaml
```

#设置超时时间

```
[root@xuegod63 timeout]# kubectl exec -it nginx-tomcat-7dd6f74846-48g9f --  sh
# apt-get update
# apt-get install vim -y
/ # vim /etc/nginx/conf.d/default.conf
```




```
proxy_pass http://tomcat-svc:8080;
proxy_http_version 1.1;
```

编辑完后，再执行如下语句验证配置和让配置生效：

```
/ # nginx -t
/ # nginx -s reload
```

这样，整个样例配置和部署都完成了。

#验证超时

登录 client，执行如下语句：

```
[root@xuegod63 timeout]# kubectl run busybox --image busybox:1.28 --restart=Never --rm -it busybox -- sh
/ # time wget -q -O - http://nginx-svc
```

```
wget: server returned error: HTTP/1.1 408 Request Timeout
Command exited with non-zero status 1
real  0m 2.02s
user  0m 0.00s
sys   0m 0.00s

/ # while true; do wget -q -O - http://nginx-svc; done
wget: server returned error: HTTP/1.1 504 Gateway Timeout
wget: server returned error: HTTP/1.1 504 Gateway Timeout
wget: server returned error: HTTP/1.1 504 Gateway Timeout
wget: server returned error: HTTP/1.1 504 Gateway Timeout
wget: server returned error: HTTP/1.1 408 Request Timeout
```

每隔 2 秒，由于 nginx 服务的超时时间到了而 tomcat 未有响应，则提示返回超时错误。

验证故障注入效果，执行如下语句：

```
/ # time wget -q -O - http://tomcat-svc
wget: server returned error: HTTP/1.1 503 Service Unavailable
Command exited with non-zero status 1
real  0m 10.02s
user  0m 0.00s
sys   0m 0.01s
```

执行之后 10s 才会有结果

3 故障注入和重试

Istio 重试机制就是如果调用服务失败，Envoy 代理尝试连接服务的最大次数。而默认情况下，Envoy 代理在失败后并不会尝试重新连接服务，除非我们启动 Istio 重试机制。

下面例子模拟客户端调用 nginx，nginx 将请求转发给 tomcat。tomcat 通过故障注入而中止对外服务，nginx 设置如果访问 tomcat 失败则会重试 3 次。

```
[root@xuegod63 attemp]# cd /root/timeout/
[root@xuegod63 timeout]# kubectl delete -f .
[root@xuegod63 timeout]# kubectl apply -f nginx-deployment.yaml
[root@xuegod63 timeout]# kubectl apply -f nginx-tomcat-svc.yaml
[root@xuegod63 ~]# kubectl get pods
NAME                              READY   STATUS    RESTARTS   AGE
busybox                           2/2     Running   0          55m
nginx-7f6496574c-zbtqj            2/2     Running   0          10m
tomcat-86ddb8f5c9-dqxcq           2/2     Running   0          35m

[root@xuegod63 timeout]# cat virtual-attempt.yaml
---
apiVersion: networking.istio.io/v1beta1
kind: VirtualService
```

```
metadata:
  name: nginx-vs
spec:
  hosts:
  - nginx-svc
  http:
  - route:
    - destination:
        host: nginx-svc
    retries:
      attempts: 3
      perTryTimeout: 2s
---
apiVersion: networking.istio.io/v1beta1
kind: VirtualService
metadata:
  name: tomcat-vs
spec:
  hosts:
  - tomcat-svc
  http:
  - fault:
      abort:
        percentage:
          value: 100
        httpStatus: 503
    route:
    - destination:
        host: tomcat-svc
[root@xuegod63 timeout]# kubectl apply -f virtual-attempt.yaml
```

虚拟服务资源清单解读：

第一：故障注入。该虚拟服务的作用对象就是 tomcat-svc。使用此故障注入后，在网格中该 tomcat 就是不可用的。

```
abort:
        percentage:
          value: 100
        httpStatus: 503
```

abort 是模拟 tomcat 服务始终不可用，该设置说明每次调用 tomcat-svc 的 k8s service，100%都会返回错误状态码 503。

第二：调用超时：

```
hosts:
```

```
  - nginx-svc
    http:
    - route:
      - destination:
          host: nginx-svc
      reties:
        attempts: 3
        perTryTimeout: 2s
```

该设置说明调用 nginx-svc 的 k8s service，在初始调用失败后最多重试 3 次来连接到服务子集，每个重试都有 2 秒的超时。

```
[root@xuegod63 timeout]# kubectl exec -it nginx-tomcat-7dd6f74846-rdqqf -- /bin/sh
# apt-get update
# apt-get install vim -y
/ # vi /etc/nginx/conf.d/default.conf
```

```
location / {
    #    root   /usr/share/nginx/html;
    #    index  index.html index.htm;
     proxy_pass http://tomcat-svc:8080;
     proxy_http_version 1.1;

}
```

```
/ # nginx -t
/ # nginx -s reload
```

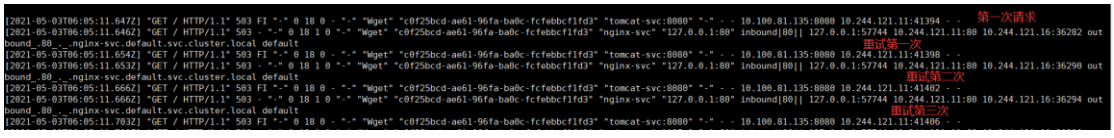
#验证重试是否生效

```
[root@xuegod63 timeout]#  kubectl run busybox --image busybox:1.28 --restart=Never --rm -it busybox -- sh
```

```
/ # wget -q -O - http://nginx-svc
```

```
[root@xuegod63 timeout]# kubectl logs -f nginx-tomcat-7dd6f74846-rdqqf  -c istio-proxy
```

#执行结果如下：



由上图可知，重试设置生效。

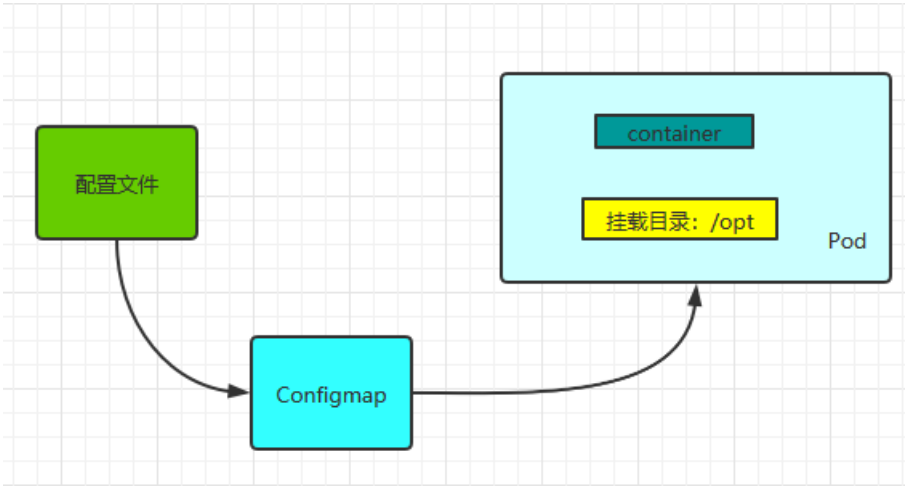
9.1 Configmap 概述

9.1.1 什么是 Configmap?

Configmap 是 k8s 中的资源对象，用于保存非机密性的配置的，数据可以用 key/value 键值对的形式保存，也可通过文件的形式保存。

9.1.2 Configmap 能解决哪些问题?

我们在部署服务的时候，每个服务都有自己的配置文件，如果一台服务器上部署多个服务：nginx、tomcat、apache 等，那么这些配置都存在这个节点上，假如一台服务器不能满足线上高并发的要求，需要对服务器扩容，扩容之后的服务器还是需要部署多个服务：nginx、tomcat、apache，新增加的服务器上还是要管理这些服务的配置，如果有一个服务出现问题，需要修改配置文件，每台物理节点上的配置都需要修改，这种方式肯定满足不了线上大批量的配置变更要求。 所以，k8s 中引入了 Configmap 资源对象，可以当成 volume 挂载到 pod 中，实现统一的配置管理。



1、Configmap 是 k8s 中的资源， 相当于配置文件，可以有一个或者多个 Configmap；

2、Configmap 可以做成 Volume，k8s pod 启动之后，通过 volume 形式映射到容器内部指定目录上；

3、容器中应用程序按照原有方式读取容器特定目录上的配置文件。

4、在容器看来，配置文件就像是打包在容器内部特定目录，整个过程对应用没有任何侵入。

9.1.3 Configmap 应用场景

1、使用 k8s 部署应用，当你将应用配置写进代码中，更新配置时也需要打包镜像，configmap 可以将配置信息和 docker 镜像解耦，以便实现镜像的可移植性和可复用性，因为一个 configMap 其实就是一系列配置信息的集合，可直接注入到 Pod 中给容器使用。configmap 注入方式有两种，一种将 configMap 做为存储卷，一种是将 configMap 通过 env 中 configMapKeyRef 注入到容器中。另一种是做成 volume 卷，挂载到容器里

2、使用微服务架构的话，存在多个服务共用配置的情况，如果每个服务中单独一份配置的话，那么更新配置就很麻烦，使用 configmap 可以友好的进行配置共享。

9.1.4 局限性

ConfigMap 在设计上不是用来保存大量数据的。在 ConfigMap 中保存的数据不可超过 1 MiB。如果你需要保存超出此尺寸限制的数据，可以考虑挂载存储卷或者使用独立的数据库或者文件服务。

9.2 Configmap 创建方法

9.2.1 命令行直接创建

直接在命令行中指定 configmap 参数创建，通过--from-literal 指定参数

```
[root@xuegod63 ~]# kubectl create configmap tomcat-config --from-
literal=tomcat_port=8080 --from-literal=server_name=myapp.tomcat.com
[root@xuegod63 ~]# kubectl describe configmap tomcat-config
Name:             tomcat-config
Namespace:        default
Labels:           <none>
Annotations:      <none>
Data
====
server_name:
----
myapp.tomcat.com
tomcat_port:
----
8080
Events:   <none>
```

9.2.2 通过文件创建

通过指定文件创建一个 configmap，--from-file=<文件>

```
[root@xuegod63 ~]# vim nginx.conf
server {
   server_name www.nginx.com;
   listen 80;
   root /home/nginx/www/
}
#定义一个 key 是 www，值是 nginx.conf 中的内容
[root@xuegod63 ~]# kubectl create configmap www-nginx --from-
file=www=./nginx.conf
[root@xuegod63 ~]# kubectl describe configmap www-nginx
Name:             www-nginx
Namespace:        default
Labels:           <none>
Annotations:      <none>

Data
====
www:
----
```

```
server {
  server_name www.nginx.com;
  listen 80;
  root /home/nginx/www/
}
```

9.2.3 指定目录创建 configmap

```
[root@xuegod63 ~]# mkdir test-a
[root@xuegod63 ~]# cd test-a/
[root@xuegod63 test-a]# cat my-server.cnf
server-id=1
[root@xuegod63 test-a]# cat my-slave.cnf
server-id=2
#指定目录创建 configmap
[root@xuegod63 test-a]# kubectl create configmap mysql-config --from-
file=/root/test-a/
#查看 configmap 详细信息
[root@xuegod63 test-a]# kubectl describe configmap mysql-config
Name:         mysql-config
Namespace:    default
Labels:       <none>
Annotations:  <none>
Data
====
my-server.cnf:
----
server-id=1
my-slave.cnf:
----
server-id=2
Events:  <none>
```

9.2.4 编写 configmap 资源清单 YAML 文件

```
[root@xuegod63 mysql]# cat mysql-configmap.yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: mysql
  labels:
    app: mysql
data:
  master.cnf: |
    [mysqld]
```

```
    log-bin
    log_bin_trust_function_creators=1
    lower_case_table_names=1
  slave.cnf: |
    [mysqld]
    super-read-only
    log_bin_trust_function_creators=1
```

9.3 使用 Configmap

9.3.1 通过环境变量引入：使用 configMapKeyRef

```
#创建一个存储 mysql 配置的 configmap
[root@xuegod63 ~]# cat mysql-configmap.yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: mysql
  labels:
    app: mysql
data:
    log: "1"
    lower: "1"
[root@xuegod63 ~]# kubectl apply -f mysql-configmap.yaml
#创建 pod，引用 Configmap 中的内容
[root@xuegod63 ~]# cat mysql-pod.yaml
apiVersion: v1
kind: Pod
metadata:
  name: mysql-pod
spec:
  containers:
  - name: mysql
    image: busybox
    command: [ "/bin/sh", "-c", "sleep 3600" ]
    env:
    - name: log_bin     #定义环境变量 log_bin
      valueFrom:
        configMapKeyRef:
          name: mysql       #指定 configmap 的名字
          key: log #指定 configmap 中的 key
    - name: lower     #定义环境变量 lower
      valueFrom:
        configMapKeyRef:
```

```
            name: mysql
            key: lower
  restartPolicy: Never
#更新资源清单文件
[root@xuegod63 ~]# kubectl apply -f mysql-pod.yaml
[root@xuegod63 ~]# kubectl exec -it mysql-pod -- /bin/sh
/ # printenv
log_bin=1
lower=1
```

9.3.2 通过环境变量引入：使用 envfrom

```
[root@xuegod63 ~]# cat mysql-pod-envfrom.yaml
apiVersion: v1
kind: Pod
metadata:
  name: mysql-pod-envfrom
spec:
  containers:
  - name: mysql
    image: busybox
    command: [ "/bin/sh", "-c", "sleep 3600" ]
    envFrom:
    - configMapRef:
        name: mysql       #指定 configmap 的名字
  restartPolicy: Never

#更新资源清单文件
[root@xuegod63 ~]# kubectl apply -f mysql-pod-envfrom.yaml
[root@xuegod63 ~]# kubectl exec -it mysql-pod-envfrom -- /bin/sh
/ # printenv
lower=1
log=1
```

9.3.3 把 configmap 做成 volume，挂载到 pod

```
[root@xuegod63 ~]# cat mysql-configmap.yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: mysql
  labels:
    app: mysql
data:
    log: "1"
    lower: "1"
    my.cnf: |
```

```
        [mysqld]
        Welcome=xuegod
[root@xuegod63 ~]# kubectl apply -f mysql-configmap.yaml
[root@xuegod63 ~]# cat mysql-pod-volume.yaml
apiVersion: v1
kind: Pod
metadata:
  name: mysql-pod-volume
spec:
  containers:
  - name: mysql
    image: busybox
    command: [ "/bin/sh","-c","sleep 3600" ]
    volumeMounts:
    - name: mysql-config
      mountPath: /tmp/config
  volumes:
  - name: mysql-config
    configMap:
      name: mysql
  restartPolicy: Never
[root@xuegod63 ~]# kubectl apply -f mysql-pod-volume.yaml
[root@xuegod63 ~]# kubectl exec -it mysql-pod-volume -- /bin/sh
/ # cd /tmp/config/
/tmp/config # ls
log      lower     my.cnf
```

总结：

实战 1：基于 Istio 的灰度发布

实战 2：卸载 istio 集群

实战 3：istio 核心功能演示

9.1　Configmap 概述

9.2　Configmap 创建方法

9.3　使用 Configmap