目录

Kubernetes迁移

1.1 迁移背景

进入云架构时代以来，随着公有云成为越来越多客户的选择，云迁移成为必不可少的服务。
在当下，容器具有其跨平台、资源利用率高、敏捷、具有可持续部署和测试等诸多优势，被越来越多的企业接纳。容器结合公有云的弹性和云网络的优势，可谓是完美的结合，越来越多的企业会考虑在云上使用容器集群，本文就以企业自建kubernetes上云迁移至金山云kubernetes集群为例，进行容器迁移上云的方案介绍。

1.2 迁移流程

在迁移前，需确定数据传输方式，如果采用专线或VPN传输，需提前做好专线施工、网络调试，本文主要介绍源端和目标端网络连通后的迁移过程。具体过程介绍如下：

1. 在金山云部署kubernetes集群，并进行集群资源配置。
2. 数据迁移：数据主要包括数据库数据、文件存储、容器镜像等。
    - 对于数据库数据迁移，金山云提供数据传输服务（DTS），通过DTS可实现全量、增量同步。
    - 对于文件存储迁移，金山云提供KS3Up-tool工具，可实现PB级文件数据迁移至金山云对象存储（KS3）上，KS3Up-tool支持将金山云KS3、阿里云OSS、腾讯云COS、百度云、七牛云和 AWS S3 设置为迁移源。
    - 对于容器镜像迁移，如果使用的是Harbor，Harbor提供了迁移能力，可以实现自动迁移到金山云容器镜像仓库；如果镜像规模较小，也可以通过pull/push进行迁移。
3. 应用及配置迁移，该部分迁移主要通过velero工具实现，金山云通过开源velero工具进行二次研发，可实现源端集群配置顺利迁移至金山云kubernetes集群。关于Velero的详细介绍：https://velero.io/。
4. 迁移后进行应用回归测试，验证相关服务是否启动运行正常。
5. 灰度切量，完成业务流量逐步切换至金山云容器集群。
6. 业务在新环境运行稳定后，可逐步下线源端集群，完成迁移。

2.1 应用及配置迁移方案概述

具有容器服务集群备份或者迁移需求的场景下，一般考采用Velero集成Restic工具实现，该工具适用以下场景：

- 集群升级前，做集群备份。
- 为保障集群高可靠，做定期备份。
- 开发、测试、生产环境之间，通过备份实现集群的迁移。
- 外部Kubernetes集群迁入至金山云容器服务。

Velero是一个提供 Kubernetes 集群和持久卷的备份、迁移以及灾难恢复等的开源工具。金山云开发了针对Velero的插件以及Velero集成Restic方案，可以实现容器服务的备份、迁移。本文主要介绍Velero集成Restic方案实现Kubernetes集群迁移。
Velero支持集成Restic工具来备份和还原存储卷，Restic除了支持块存储之外，还支持更多类型的存储，比如NFS，Emptydir、Local以及其他不支持快照的任何存储类型。当前Restic不支持HostPath类型的存储，支持Local类型的PV存储卷。

2.2 使用须知

- 在做迁移使用时，建议Kubernetes同版本下迁移(未来Velero新版本会支持跨版本迁移)。
- 推荐排除备份velero和kube-system命名空间下的资源，因为kube-system下都是系统服务不需要备份；velero命名空间下是部署的 velero 服务，也不需要备份。
- Velero的Restic集成需要Kubernetes [MountPropagation功能]，该功能在Kubernetes v1.10.0和更高版本中默认启用。
- 跨厂商迁移使用Velero的Restic集成，由于不同的云厂商，后端的存储基础设施是不一样的，因此在金山云kubernetes集群恢复时，需要先创建一个相同名称的。StorageClass来屏蔽对底层存储基础设施差异的感知。

2.3 迁移过程

2.3.1 环境准备

请参考以下步骤在自建kubernetes集群及金山云kubernetes集群中部署velero。

2.3.1.1 下载安装velero客户端

```
wget https://ks3-cn-beijing.ksyun.com/velero/velero-v1.2.0-linux-amd64.tar.gz
```

解压:

```
tar -zxvf velero-v1.2.0-linux-amd64.tar.gz
```

将 velero二进制文件移到PATH环境变量定义的目录下：

```
cp velero /bin
```

验证是否安装成功：

```
velero -h
```

2.3.1.2. 创建对象存储bucket

金山云KS3上创建2个bucket。

2.3.1.3. 获取ak/sk

获取ak/sk，ak/sk的主要目的是实现对金山云对象存储、块存储、块存储快照的操作权限。

2.3.1.4. 配置集群的config信息

连接本地集群无需单独指定，如果在本地执行目标端velero安装，则进行配置。容器服务的config文件可以通过**集群管理** > **集群详情页** > **集群基本信息** > **获取集群config**获得。
说明：velero默认会从$HOME/.kube目录下查找文件名为 config 的文件，如果将在目标端安装，则需要指定 config的路径。命令中添加--kubeconfig ，即为config的路径。

2.3.1.5. 安装velero服务

创建credentials-velero文件并设置ak/sk值。

```
vim /data/credentials-velero
```

[default]

```
ksyun_access_key_id = <your access_key>
ksyun_access_key_secret = <your secret_key>
```

velero服务安装:

```
velero install --provider ksyun \
       --image hub.kce.ksyun.com/ksyun/velero:v1.3.0-beta.2 \
       --plugins hub.kce.ksyun.com/ksyun/velero-plugin-ksyun:v1.2.0 \
       --kubeconfig <KUBECONFIG> \
       --bucket <YOUR_BUCKET> \
       --backup-location-config region=<YOUR_REGION>,resticRepoPrefix=<resticRepoPrefix> \
       --secret-file ./credentials-velero \
       --use-volume-snapshots=false \
       --use-restic
```

运行安装示例，注：示例中的两处bucket不能一样：

```
velero install --provider ksyun \
--image hub.kce.ksyun.com/ksyun/velero:v1.3.0-beta.2 \
--plugins hub.kce.ksyun.com/ksyun/velero-plugin-ksyun:v1.2.0 \
--kubeconfig /data/soukubeconfig \
--bucket test-b \
--backup-location-config region=cn-beijing-6,resticRepoPrefix=ks3:ks3-cn-beijing.ksyun.com/test-a \
--secret-file /data/credentials-velero \
--use-volume-snapshots=false \
--use-restic
```

2.3.1.6. 验证

执行：

```
kubectl logs deployment/velero -n velero
```

查看是否有错误。执行kubectl get pod -n velero -o wide查看namespace为velero的pod是否都启动并running正常。当restic-xxx pod显示CrashLoopBackOff，需要修改其中的存储卷位置信息：

```
kubectl get daemonset -n velero
```

执行:

```
kubectl edit ds restic -n velero
```

把 volumes 下的 kubelet 目录配置由 /var/lib/kubelet/pods 改成 /data/kubelet/pods，再次看pod运行是否正常。

2.3.2. 自建kubernetes集群备份

2.3.2.1. 设置 annotate

使用restic备份时，需要明确指定需要备份的存储卷pod。velero通过pod的 annotation 来过滤需要备份存储卷的pod。对于需要备份存储卷的pod，我们需要执行如下命令设置 annotate。

```
kubectl -n YOUR_POD_NAMESPACE annotate pod/YOUR_POD_NAME backup.velero.io/backup-volumes=YOUR_VOLUME_NAME_1,YOUR_VOLUME_NAME_2
,...
```

示例：

```
kubectl get pod -o yaml
```

查看volumes的名字。

```
kubectl -n default annotate pod/nginx-7bb7cd8db5-c9fjf backup.velero.io/backup-volumes=default-token-bs7q8
```

注：-n指定namespace，backup-volumes=volumes中的name。

2.3.2.2. 创建backup

```
velero backup create NAME OPTIONS...
```

示例：

```
velero backup create nginxbk20200313 --wait
```

2.3.3. 金山云kubernetes集群恢复

2.3.3.1. 设置restic helper

恢复资源时，会启动一个 restic helper 容器来帮助数据的恢复，可以通过configmap来自定义配置该容器的配置，如需要配置则需要在执行恢复操作之前先部署如下 ConfigMap。

```
apiVersion: v1
kind: ConfigMap
metadata:
  # any name can be used; Velero uses the labels (below)
  # to identify it rather than the name
  name: restic-restore-action-config
  # must be in the velero namespace
  namespace: velero
  # the below labels should be used verbatim in your
  # ConfigMap.
  labels:
    # this value-less label identifies the ConfigMap as
    # config for a plugin (i.e. the built-in restic restore
    # item action plugin)
    velero.io/plugin-config: ""
    # this label identifies the name and kind of plugin
    # that this ConfigMap is for.
    velero.io/restic: RestoreItemAction
data:
  # The value for "image" can either include a tag or not;
  # if the tag is *not* included, the tag from the main Velero
  # image will automatically be used.
  image: hub.kce.ksyun.com/ksyun/velero-restic-restore-helper:v1.2.0

  # "cpuRequest" sets the request.cpu value on the restic init containers during restore.
  # If not set, it will default to "100m". A value of "0" is treated as unbounded.
  cpuRequest: 200m

  # "memRequest" sets the request.memory value on the restic init containers during restore.
  # If not set, it will default to "128Mi". A value of "0" is treated as unbounded.
  memRequest: 128Mi

  # "cpuLimit" sets the request.cpu value on the restic init containers during restore.
  # If not set, it will default to "100m". A value of "0" is treated as unbounded.
  cpuLimit: 200m

  # "memLimit" sets the request.memory value on the restic init containers during restore.
  # If not set, it will default to "128Mi". A value of "0" is treated as unbounded.
  memLimit: 128Mi
```

2.3.3.2. 创建 restore

```
velero restore create --from-backup BACKUP_NAME OPTIONS...
```

示例：

```
velero restore create restore-20200313 --from-backup bk20200313 --wait
velero restore get restore-20200313
```

注： a、不指定create名，系统会自动生成，建议按照自己的规范命名，后面方便查看恢复的结果。 b、此处验证的是完全备份和完全恢复，恢复过程可能会提示部分失败，原因为目标端已经存在，可忽略。

2.3.3.3. 验证

```
kubectl get pod -n default -o wide
```

发现已经恢复并运行，可以通过访问pod应用进行验证。至此kubernetes应用及配置迁移完成。

2.3.4. 清理删除velero服务（如需删除重新安装时使用）

```
kubectl delete namespace/velero clusterrolebinding/velero
kubectl delete crds -l component=velero
```

通过Nginx-ingress实现灰度发布

在对服务进行版本发布或版本升级场景中，常会用到灰度发布、蓝绿发布等发布方式。本文将介绍如何在金山云容器服务中通过Nginx-ingress服务实现应用的灰度发布。

前提条件

集群内完成nginx-ingress-controller的部署，并通过金山云的负载均衡服务暴露到集群外。部署方式可参考Nginx-ingress使用。

Ingress配置策略

Ingress controller支持通过配置ingress annotations实现不同场景下的灰度发布和测试。Nginx annotations支持以下四种灰度发布（Canary）规则：

- nginx.ingress.kubernetes.io/canary-by-header：基于 Request Header 的流量切分，适用于灰度发布以及 A/B 测试。当 Request Header 设置为 always时，请求将会被一直发送到 Canary 版本；当 Request Header 设置为 never时，请求不会被发送到 Canary 入口；对于任何其他 Header 值，将忽略 Header，并通过优先级将请求与其他 Canary 规则进行优先级的比较。
- nginx.ingress.kubernetes.io/canary-by-header-value：要匹配的 Request Header 的值，用于通知 Ingress 将请求路由到 Canary Ingress 中指定的服务。当 Request Header 设置为此值时，它将被路由到 Canary 入口。该规则允许用户自定义 Request Header 的值，必须与上一个 annotation（即：canary-by-header）一起使用。
- nginx.ingress.kubernetes.io/canary-weight：基于服务权重的流量切分，适用于蓝绿部署，权重范围 0 - 100 按百分比将请求路由到 Canary Ingress 中指定的服务。权重为 0 意味着该金丝雀规则不会向 Canary 入口的服务发送任何请求。权重为 100 意味着所有请求都将被发送到 Canary 入口。
- nginx.ingress.kubernetes.io/canary-by-cookie：基于 Cookie 的流量切分，适用于灰度发布与 A/B 测试。用于通知 Ingress 将请求路由到 Canary Ingress 中指定的服务的 cookie。当 cookie 值设置为 always时，它将被路由到 Canary 入口；当 cookie 值设置为 never时，请求不会被发送到 Canary 入口；对于任何其他值，将忽略 cookie 并将请求与其他金丝雀规则进行优先级的比较。

  注意：当同时使用多个Canary规则时，按如下优先顺序进行排序： canary-by-header - > canary-by-cookie - > canary-weight

部署示例

以下示例中将会部署helloworld服务的v1和v2版本，将v2版本作为canary版本，在ingress中设置canary规则，实现基于服务权重的流量切分。

创建测试应用

helloworld-v1.yaml如下：

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: hello-world
spec:
  replicas: 1
  selector:
    matchLabels:
      app: hello-world
```

```
      version: v1
  template:
    metadata:
      labels:
        app: hello-world
        version: v1
    spec:
      containers:
      - name: hello-world
        image: hub.kce.ksyun.com/kingsoft/hello-world:v1
---
apiVersion: v1
kind: Service
metadata:
  labels:
    app: hello-world
  name: hello-world-svc
spec:
  ports:
  - port: 80
    protocol: TCP
    targetPort: 80
  selector:
    app: hello-world
    version: v1
  type: ClusterIP
```

helloworld-v2.yaml如下：

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: hello-world-v2
spec:
  replicas: 1
  selector:
    matchLabels:
      app: hello-world
      version: v2
  template:
    metadata:
      labels:
        app: hello-world
        version: v2
    spec:
      containers:
      - name: hello-world
        image: hub.kce.ksyun.com/kingsoft/hello-world:v2
---
apiVersion: v1
kind: Service
metadata:
  labels:
    app: hello-world
  name: hello-world-svc-v2
spec:
  ports:
  - port: 80
    protocol: TCP
    targetPort: 80
  selector:
    app: hello-world
    version: v2
  type: ClusterIP
```

创建并验证v1,v2版本服务的部署情况：

```
[root@vm10-0-11-201 ~]# kubectl apply -f helloworld-v2.yaml
deployment.apps/hello-world-v2 created
service/hello-world-svc-v2 created
[root@vm10-0-11-201 ~]# kubectl get deploy
NAME             READY   UP-TO-DATE   AVAILABLE   AGE
hello-world      1/1     1            1           15s
hello-world-v2   1/1     1            1           9s
[root@vm10-0-11-201 ~]# kubectl get svc
NAME                 TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)   AGE
hello-world-svc      ClusterIP   10.254.200.211   <none>        80/TCP    19s
hello-world-svc-v2   ClusterIP   10.254.14.130    <none>        80/TCP    13s
kubernetes           ClusterIP   10.254.0.1       <none>        443/TCP   7d
[root@vm10-0-11-201 ~]# curl 10.254.200.211
Hello World v1!
[root@vm10-0-11-201 ~]# curl 10.254.14.130
Hello World v2!
```

配置Ingress

基于服务权重进行流量切分

对v1版本服务进行Ingress配置，创建helloworld-ingress.yaml：

```
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: hello-world
  annotations:
    kubernetes.io/ingress.class: nginx
spec:
  rules:
  - host: hello.world.test
    http:
      paths:
      - backend:
          serviceName: hello-world-svc
          servicePort: 80
```

对v2版本的canary规则进行配置，创建weight-ingress.yaml：

```
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  annotations:
    kubernetes.io/ingress.class: nginx
    nginx.ingress.kubernetes.io/canary: "true"
    nginx.ingress.kubernetes.io/canary-weight: "50"
  name: helloworld-weight
spec:
  rules:
  - host: hello.world.test
    http:
      paths:
      - backend:
          serviceName: hello-world-svc-v2
          servicePort: 80
```

创建Ingress规则：

```
[root@vm10-0-11-201 ~]# kubectl apply -f helloworld-ingress.yaml
ingress.extensions/hello-world created
[root@vm10-0-11-201 ~]# kubectl apply -f weight-ingress.yaml
ingress.extensions/helloworld-weight created
[root@vm10-0-11-201 ~]# kubectl get ingress
NAME                CLASS    HOSTS              ADDRESS   PORTS   AGE
hello-world         <none>   hello.world.test             80      41s
helloworld-weight   <none>   hello.world.test             80      27s
```

验证访问情况

通过以下命令获取EXTERNAL-IP及访问服务：

```
[root@vm10-0-11-201 ~]# kubectl get svc -n ingress-nginx
NAME            TYPE           CLUSTER-IP     EXTERNAL-IP     PORT(S)                      AGE
nginx-ingress   LoadBalancer   10.254.28.54   120.92.xx.xx   80:31741/TCP,443:32754/TCP   3h19m
[root@vm10-0-11-201 ~]# for i in $(seq 1 10); do curl -H "Host: hello.world.test"  http://120.92.xx.xx; done;
Hello World v2!
Hello World v2!
Hello World v1!
Hello World v2!
Hello World v2!
Hello World v1!
Hello World v1!
Hello World v1!
Hello World v2!
Hello World v2!
```

多次访问能发现约50%的流量会被分发到v2版本服务中。

在KCE集群中对Pod进行带宽限速

金山云容器服务原生支持对Pod进行带宽限速。本文档讲介绍如何在KCE集群设置Pod带宽限速。

备注

- 对于创建时间在2021-3-16之后的集群，默认支持对Pod带宽进行限速；对于创建时间在2021-3-16之前的集群，如需要使用Pod带宽限速的能力，请联系您的商务申请
- 集群网络插件Flannel和Canal均支持Pod带宽限速的能力

使用方式

新建Pod，通过annotations的方式设置Pod出入带宽

- kubernetes.io/egress-bandwidth:定义Pod的出向带宽，如240M，这里单位是Mbit，
- kubernetes.io/ingress-bandwidth：定义Pod的入向带宽，如400M，这里单位是Mbit

Yaml示例如下：

```
apiVersion: v1
kind: Pod
metadata:
  annotations:
    kubernetes.io/egress-bandwidth: 240M
    kubernetes.io/ingress-bandwidth: 400M
  name: iperf3-4
  namespace: default
spec:
  containers:
  - args:
    - iperf3
    - -s
    image: networkstatic/iperf3
    imagePullPolicy: Always
    name: iperf3-4
    resources: {}
  dnsPolicy: ClusterFirst
```

验证

你可以通过以下两种方式验证：

- 登陆pod所在的节点，执行：

  ```
  tc qdisc show
  ```

  返回如下示例代表设置成功

  ```
  qdisc tbf 1: dev vethbab31be6 root refcnt 2 rate 400Mbit burst 256Mb lat 25.0ms
  qdisc ingress ffff: dev vethbab31be6 parent ffff:fff1 ----------------
  qdisc tbf 1: dev bwp91fff0f8f685 root refcnt 2 rate 240Mbit burst 256Mb lat 25.0ms
  ```

- 使用iperf3工具验证

  ```
  iperf3 -c <服务 IP> -i 1
  ```

  结果如下：

  ```
  Server listening on 5201
  -----------------------------------------------------------
  Accepted connection from 10.0.11.85, port 45690
  [  5] local 10.0.11.207 port 5201 connected to 10.0.11.85 port 45692
  [ ID] Interval           Transfer     Bandwidth
  [  5]   0.00-1.00   sec   243 MBytes  2.03 Gbits/sec
  [  5]   1.00-2.00   sec  49.2 MBytes   413 Mbits/sec
  [  5]   2.00-3.00   sec  27.3 MBytes   229 Mbits/sec
  [  5]   3.00-4.00   sec  27.3 MBytes   229 Mbits/sec
  [  5]   4.00-5.00   sec  27.4 MBytes   230 Mbits/sec
  [  5]   5.00-6.00   sec  27.3 MBytes   229 Mbits/sec
  [  5]   6.00-7.00   sec  27.3 MBytes   229 Mbits/sec
  [  5]   7.00-8.00   sec  27.3 MBytes   229 Mbits/sec
  [  5]   8.00-9.00   sec  27.3 MBytes   229 Mbits/sec
  [  5]   9.00-10.00  sec  27.3 MBytes   229 Mbits/sec
  [  5]  10.00-10.06  sec  1.49 MBytes   225 Mbits/sec
  ```

- [ID] Interval Transfer Bandwidth [5] 0.00-10.06 sec 0.00 Bytes 0.00 bits/sec sender [5] 0.00-10.06 sec 512 MBytes 427 Mbits/sec receiver

如何选择Containerd和Docker

容器运行时简介

容器运行时（Container Runtime）是kubernetes最重要的组件之一，负责管理镜像和容器的生命周期。Kubelet通过Container Runtime Interface (CRI) 与容器运行时交互，来管理镜像和容器。

如何选择Containerd和Docker？

容器服务支持Containerd和Docker两种运行时，您可根据实际需求选择不同的运行时：

- Containerd运行时：调用链更短、组件更少、更稳定、占用节点资源更少，建议您选择Containerd运行时。
- 若您有以下使用场景，建议您选择Docker运行时：
  - 使用 docker in docker。
  - 在节点使用docker build/push/save/load等命令。
  - 调用docker API。
  - 需要docker compose 或 docker swarm。

Containerd和Docker常用命令对比

> 说明：Containerd 不支持 docker API 和 docker CLI，若您有该需求，可以通过 cri-tool 命令实现类似的功能。

镜像相关功能

命令	Docker	Containerd
查看镜像列表	docker images	crictl images
下载镜像	docker pull	crictl pull
上传镜像	docker push	-
删除镜像	docker rmi	crictl rmi
查看镜像详情	docker inspect	crictl inspect

容器相关功能

命令	Docker	Containerd
查看容器列表	docker ps	crictl ps
创建容器	docker create	crictl create
启动容器	docker start	crictl start
停止容器	docker stop	crictl stop
删除容器	docker rm	crictl rm
查看容器详情	docker inspect	crictl inspect
挂载容器	docker attach	crictl attach
容器内执行命令	docker exec	crictl exec
查看容器日志	docker logs	crictl logs
显示容器资源使用情况	docker stats	crictl stats

POD相关功能

命令	Docker	Containerd
查看pod列表	-	crictl pods
查看pod详情	-	crictl inspectp
运行pod	-	crictl runp
停止pod	-	crictl stopp

调用链对比

容器运行时	调用链
Docker	kubelet -> docker shim -> dockerd -> containerd -> containerd-shim -> runC容器
Containerd	kubelet -> containerd -> containerd-shim -> runC容器