文档名称：ZDNS-Zcloud-kubernetes 本地存储-持久化-半动态分配—实施方案
文档分类：ZDNS-Zcloud-kubernetes 文档
服务等级：
文档编号：
作　　者: 余春云
联系电话：18811483030
电子邮件：yuchunyun@zdns.cn

kubernetes 本地存储-持久化-半动态分配—实施方案

文档变更历史记录：

变更日期	变更人	变更内容摘要	组长确认
2019-03-12	余春云	初次建立文档	

目录

目的

本地持久化卷允许用户通过标准 PVC 接口以简单、便携的方式访问本地存储

说明/局限

外部配置器（provisioner）可用于帮助简化本地存储管理。

但是，本地存储配置器与大多数配置器不同，并且尚不支持动态配置。相反，它要求管理员预先配置每个节点上的本地卷，并且这些卷应该是：

1. Filesystem volumeMode（默认）PV—— 将它们挂载到发现目录下。
2. Block volumeMode PV——在发现目录下为节点上的块设备创建一个符号链接。

配置器将通过为每个卷创建和清除 PersistentVolumes 来管理发现目录下的卷。

原理

Provisioner 本身其并不提供 local volume，但它在各个节点上的 provisioner 会去动态的“发现”挂载点（discovery directory）。当某 node 的 provisioner 在 discovery directory 下发现有挂载点时，会创建 PV，该 PV 的 local.path 就是挂载点，并设置 nodeAffinity 为该 node。当 Pod 结束并删除了使用的 PVC 后，provisioner 将自动清理该 PV 的文件、然后删除该 PV，最后重新创建 PV。

为了以后方便扩容，挂载点建议使用 LVM 的 lvextend 为特定容器使用的存储卷进行扩容。

前提

1.10 之前的 Kubernetes 版本需要几个附加 feature gate，因为持久的本地卷和其他功能处于 alpha 版本。

Kubernetes API server、controller manager、scheduler 和所有 kubelet:

feature-gates=”PersistentLocalVolumes=true,VolumeScheduling=true,MountPropagation=true,BlockVolume=true”

Filesystem 实施

注意

发现挂载点：直接去创建目录是行不通的，会提示“is not an actual mountpoint”，因为 provsioner 希望 PV 是隔离的，例如 capacity，io 等。正确的做法是是加硬盘、格式化、mount 到挂载点。

当同一个分区被两次 mount 到发现目录下后，provisioner 会认为是两个挂载点，就会生成两个相同 storageclass 的 PV，可被两个 pod 同时使用（可做共享）

当一个分区被 mount 到 filesystem 类型的 provisioner 后，如果该分区同时也在 block 类型的 provisioner 的发现目录，两个 provisioner 会生成两个不同 storageclass 的 pv，但只有一个 pod（filesystem 类型的 provisioner）能使用

配置步骤

1：Kubernetes 集群配置

目录映射

kubelet 需要将主机目录映射到容器中，修改 zke 生成的集群配置文件，

```
services: kubelet: extra_binds: - /data:/data:rshared
```

2：宿主机创建挂载点

安装 lvm

```
yum install lvm* -y
```

创建系统分区

```
fdisk /dev/sdb（注意修改分区模式为 LVM 格式）
```

创建 PV

```
pvcreate /dev/sdb1
```

创建 VG

```
vgcreate vg0 /dev/sdb1
```

创建 LV

```
lvcreate -L 8G -n lv1 vg0
```

格式化 lv

```
mkfs.ext4 /dev/vg0/lv1
```

创建挂载目录

```
mkdir -pv /data/local/
```

创建挂载点

```
mkdir /data/local/lv1
mount /dev/vg0/lv1 /data/local/lv1
```

3：部署 Provisioner

RBAC 配置

```
apiVersion: v1
kind: ServiceAccount
metadata:
```

```
  name: local-volume-dynamic-file-admin
  namespace: default
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: local-volume-provisioner-pv-binding
  namespace: default
subjects:
- kind: ServiceAccount
  name: local-volume-dynamic-file-admin
  namespace: default
roleRef:
  kind: ClusterRole
  name: system:persistent-volume-provisioner
  apiGroup: rbac.authorization.k8s.io
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: local-volume-provisioner-node-clusterrole
  namespace: default
rules:
- apiGroups: [""]
  resources: ["nodes", "persistentvolume"]
  verbs: ["get"]
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: local-volume-provisioner-node-binding
  namespace: default
subjects:
- kind: ServiceAccount
  name: local-volume-dynamic-file-admin
  namespace: default
roleRef:
  kind: ClusterRole
  name: local-volume-provisioner-node-clusterrole
  apiGroup: rbac.authorization.k8s.io
```

StorageClass 配置

```
kind: StorageClass
apiVersion: storage.k8s.io/v1
```

```
metadata:
  name: local-volume-dynamic-file
provisioner: kubernetes.io/no-provisioner
volumeBindingMode: WaitForFirstConsumer
```

ConfigMap 配置

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: local-volume-dynamic-file-config
  namespace: default
data:
  storageClassMap: |
    local-volume-dynamic-file:              #前面定义的 storageclass 的名称
      hostDir: /data/local                  #配置发现目录
      mountDir:    /data/local
      blockCleanerCommand:
        - "/scripts/shred.sh"
        - "2"
      volumeMode: Filesystem
      fsType: ext4
```

DaemonSet 配置

```
apiVersion: extensions/v1beta1
kind: DaemonSet
metadata:
  name: local-volume-dynamic-file-provisioner
  namespace: default
  labels:
    app: local-volume-dynamic-file-provisioner
spec:
  selector:
    matchLabels:
      app: local-volume-dynamic-file-provisioner
  template:
    metadata:
      labels:
        app: local-volume-dynamic-file-provisioner
    spec:
      serviceAccountName: local-volume-dynamic-file-admin
      containers:
        - image: "quay.io/external_storage/local-volume-provisioner "
          imagePullPolicy: "Always"
          name: local-volume-dynamic-file-provisioner
```

```
            securityContext:
              privileged: true
            env:
            - name: MY_NODE_NAME
              valueFrom:
                fieldRef:
                  fieldPath: spec.nodeName
            volumeMounts:
              - mountPath: /etc/provisioner/config
                name: provisioner-config
                readOnly: true
              - mountPath:    /data/local
                name: local-storage
                mountPropagation: "HostToContainer"
      volumes:
        - name: provisioner-config
          configMap:
            name: local-volume-dynamic-file-config
        - name: local-storage
          hostPath:
            path: /data/local
```

4：创建 pvc 和 pod 使用

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: local-volume-dynamic-file-pvc
spec:
  accessModes:
  - ReadWriteOnce
  resources:
    requests:
      storage: 3Gi
  storageClassName: local-volume-dynamic-file
---
kind: Pod
apiVersion: v1
metadata:
  name: local-volume-dynamic-file-pod
spec:
  containers:
    - name: client
```

```
      image: ikubernetes/myapp:v3
      volumeMounts:
      - mountPath: "/data"
        name: data
  volumes:
    - name: data
      persistentVolumeClaim:
        claimName: local-volume-dynamic-file-pvc
```

5：pv 扩容

Lv 增加容量

```
lvextend -L +1000M /dev/mapper/vgapps-lv_test
```

文件系统扩容

```
resize2fs /dev/mapper/vgapps-lv_test
```

此时 pod 里挂载分区容量已经扩大（k8s 里 pv 显示依然没有变化）

Block 实施

说明

为了省去在宿主机上手动创建挂载点/链接，我们将把/dev/mapper（lvm 默认链接目录）作为 provisioner 的发现目录

配置步骤

1：部署 Provisioner

RBAC 配置

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: local-volume-dynamic-block-admin
  namespace: default
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: local-storage:provisioner-pv-binding
  namespace: default
subjects:
```

```
- kind: ServiceAccount
  name: local-volume-dynamic-block-admin
  namespace: default
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: system:persistent-volume-provisioner
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: local-storage:provisioner-node-binding
  namespace: default
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: system:node
subjects:
- kind: ServiceAccount
  name: local-volume-dynamic-block-admin
  namespace: default
```

StorageClass 配置

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
 name: local-volume-dynamic-block
provisioner: kubernetes.io/no-provisioner
volumeBindingMode: WaitForFirstConsumer
```

ConfigMap 配置

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: local-volume-dynamic-block-config
data:
     storageClassMap: |
           local-volume-dynamic-block:
                 hostDir:    /dev/mapper
                 mountDir: /dev/mapper
                 volumeMode: Block
```

DaemonSet 配置

```
apiVersion: extensions/v1beta1
kind: DaemonSet
metadata:
  name: local-volume-dynamic-block-provisioner
```

```
spec:
  template:
    metadata:
      labels:
        app: local-volume-dynamic-block-provisioner
    spec:
      containers:
      - env:
        - name: MY_NODE_NAME
          valueFrom:
            fieldRef:
              apiVersion: v1
              fieldPath: spec.nodeName
        - name: MY_NAMESPACE
          valueFrom:
            fieldRef:
              apiVersion: v1
              fieldPath: metadata.namespace
        - name: VOLUME_CONFIG_NAME
          value: local-volume-dynamic-block-config
        image: quay.io/external_storage/local-volume-provisioner
        imagePullPolicy: "Always"
        name: local-volume-dynamic-block-provisioner
        securityContext:
          privileged: true
          runAsUser: 0
          seLinuxOptions:
            level: "s0:c0.c1023"
        volumeMounts:
        - mountPath: /dev/mapper
          name: local-storage
          mountPropagation: HostToContainer
        - mountPath: /etc/provisioner/config
          name: provisioner-config
          readOnly: true
      serviceAccountName: "local-volume-dynamic-block-admin"
      volumes:
      - hostPath:
          path: /dev/mapper
        name: local-storage
      - configMap:
          name: local-volume-dynamic-block-config
        name: provisioner-config
```

2：创建 pvc 和 pod 使用

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: local-volume-dynamic-block-pvc
spec:
  storageClassName: local-volume-dynamic-block
  accessModes:
    - ReadWriteOnce
  volumeMode: Block
  resources:
    requests:
      storage: 2Gi
---
kind: Pod
apiVersion: v1
metadata:
  name: local-volume-dynamic-block-pod
  annotations:
    container.apparmor.security.beta.kubernetes.io/client: unconfined
spec:
  containers:
    - name: client
      image: centos
      imagePullPolicy: Never
      command: ["/bin/sh", "-c"]
      args: ["tail -f /dev/null"]
      securityContext:
        capabilities:
          add: ["SYS_ADMIN"]
      volumeDevices:
        - name: device
          devicePath: /dev/xvde
  volumes:
    - name: device
      persistentVolumeClaim:
        claimName: local-volume-dynamic-block-pvc
```

3：pv 扩容

Lv 增加容量

```
lvextend -L +1000M /dev/vgapps/lv_test
```

文件系统扩容

resize2fs /dev/mapper/vgapps-lv_test（pod 里需要先 umount）

但是，pv 和 pod 里都看不到扩容，比较奇怪

PV 回收策略（原始数据是否保留）

动态生成的 PV 的 persistentVolumeReclaimPolicy 回收策略会继承 StorageClass. reclaimPolicy 的设置。local volume storageclass 使用 kubernetes.io/no-provisioner 时，StorageClass. reclaimPolicy 支持两种回收策略。

Retain

pvc 删除后 pv 会一直处于 Released 状态（被原命名的 pvc 所声明使用）

如果是 Filesystem 类型，如果还想继续使用该 pv 和里面的数据文件，步骤如下：

1：删除 pv（对应的实际存储空间里文件内容不会消失）

2：利用 storageclass 重新创建 pv

3：pod 通过 pvc 申请命中到该 pv 后，可以看见原来的文件（pod/pvc 的命名可与之前不同）

如果是 Block 类型，如果还想继续使用该 pv 和里面的数据文件，步骤如下：

1：删除 pv（对应的实际块设备里的文件内容不会消失）

2：在宿主机上将原始块设备挂载，然后将里面的文件拷贝出来

3：利用 storageclass 重新创建 pv

4：pod 通过 pvc 申请命中到该 pv 后，由于需要 mkfs 格式化，所以块设备里的原文件会删除

5：将之前拷贝出来的文件再重新复制过来到宿主机挂载目录，此时 pod 里可以看见原文件了

Delete

pvc 删除后，对应的 pv 和后端实际存储空间里的文件一起删除

第三种在 kubernetes.io/no-provisioner 不支持

Recycle：基本的删除操作（“rm -rf /thevolume/*”）

StatefulSet 实战

前提

使用 Filesystem 类型存储，需要提前手动将分区/磁盘挂载到发现目录下
建议将 StorageClass 的 reclaimPolicy 设置为 Retain

配置 StatefulSet

```
apiVersion: v1
kind: Service
metadata:
  name: stateful-svc-myapp
  namespace: default
spec:
  clusterIP: None
  selector:
    app: myapp-pod
  ports:
  - port: 80
    name: web
---
apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: statefulset-myapp
spec:
  serviceName: stateful-svc-myapp
  replicas: 3
  selector:
    matchLabels:
      app: myapp-pod
  template:
    metadata:
      name: statefulset-myapp-temp
      labels:
        app: myapp-pod
    spec:
      tolerations:
      - key: "ceph"
        operator: "Equal"
        value: "true"
        effect: "NoSchedule"
      containers:
      - name: statefulset-myapp-temp-c1
        image: ikubernetes/myapp:v1
        imagePullPolicy: IfNotPresent
        ports:
```

```
        - name: web
          containerPort: 80
        volumeMounts:
        - name: myappdata
          mountPath: /usr/share/nginx/html
  volumeClaimTemplates:
  - metadata:
      name: myappdata
    spec:
      storageClassName: local-volume-dynamic-file#此处填写对应的 Storageclass 名称
      accessModes: ["ReadWriteOnce"]
      resources:
        requests:
          storage: 1Gi
```

测试结果

每个 pod 都会自动生成一个 pvc（以 statefulSet. spec. volumeClaimTemplates. metadata. name+pod 名称+编号命名）和绑定一个对应的 PV

通过 kubectl patch sts statefulset-myapp -p '{"spec":{"replicas":2}}'控制副本数量，当数量由大变小时，pod 会按顺序被删除。但其对应的 PVC 并不会被删除，PV 也就依然处于 Bound 状态。当下次增大副本数量时，pod 还会使用之前自己的那个 pvc 和相应的 PV，这样就保证了数据完整性和数据一致性。