Fluid 与云原生存储 CubeFS 联合应用实战

原创 顾 荣 Fluid开源项目 2023-08-29 14:53 发表于江苏

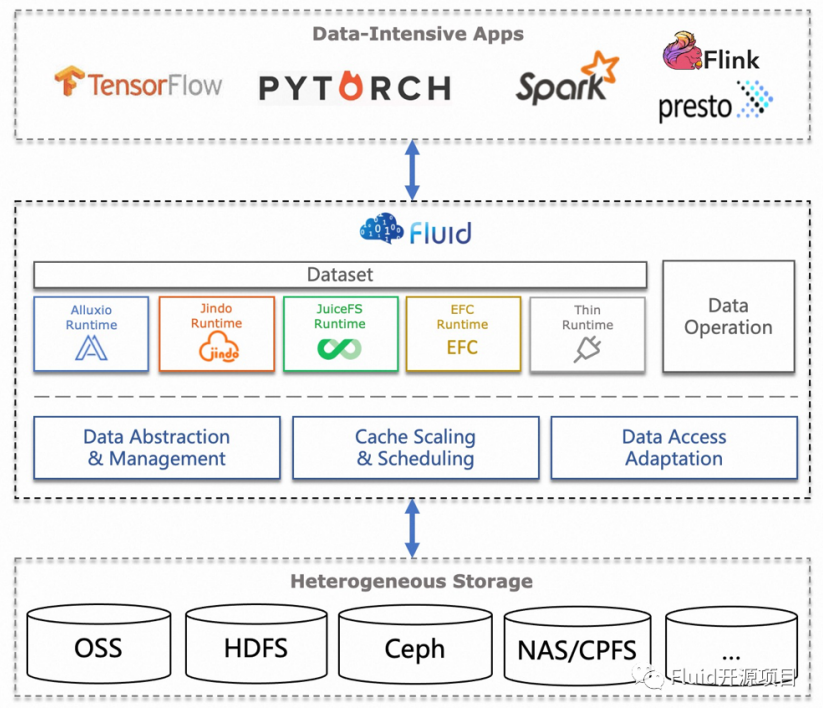
Fluid 介绍

1. 什么是 Fluid?

Fluid 是一个开源的 Kubernetes 原生的分布式数据集编排和加速引擎，主要服务于云原生场景下的数据密集型应用，例如大数据应用、AI应用等。

Fluid 对"计算作业使用的数据模型与对象"进行抽象，提出了弹性数据集 Dataset 的概念，并作为一等公民在 Kubernetes 中实现。Fluid 围绕弹性数据集 Dataset，创建了数据编排与加速系统，来实现 Dataset 管理（CRUD操作）、权限控制和访问加速等能力。具体来说，Fluid 具有以下功能：

- **数据集抽象原生支持**：将数据密集型应用所需基础支撑能力云原生化，实现数据高效访问，并降低多维管理复杂性。
- **云上数据预热与加速**：Fluid 通过使用分布式缓存引擎（Alluxio/JuiceFS/JindoFS/GooseFS 等）为云上应用提供数据预热与缓存加速，同时可以保障缓存数据的可观测性、可迁移性和自动化的水平扩展。
- **数据应用协同编排**：在云上调度应用和数据时，同时考虑两者特性与位置，实现协同编排，提升应用运行性能。
- **多命名空间管理支持**：用户可以创建和管理不同 Namespace 的数据集，从而灵活地实现隔离性和访问权限管理控制。
- **异构数据源管理**：一站式统一访问不同来源的底层数据（对象存储，HDFS 和 Ceph 等存储），适用于混合云等多数据源场景。



2. 什么是 Fluid ThinRuntime?

此外，Fluid 已经通过 Runtime Plugin 的方式，扩展兼容多种分布式缓存引擎，包括阿里云 EMR 的 JindoFS，开源的 Alluxio，Juicefs 等缓存引擎等。这极大地方便了用户通过上述缓存系统与底层存储系统交互，打通数据访问链路。

我们发现，除了上述已经集成的缓存引擎外，用户在一些云场景下存在访问自建存储系统的需求。为满足这个需求，用户须自行完成自建存储系统与 Kubernetes 环境的对接工作。这就需要用户根据自建存储系统的特性，设计实现自定义的 CSI 插件或编写 Runtime Controller 与Fluid 对接。以上这些方式都要求用户具备 Kubernetes 的相关知识，增加了存储提供者的工作成本。

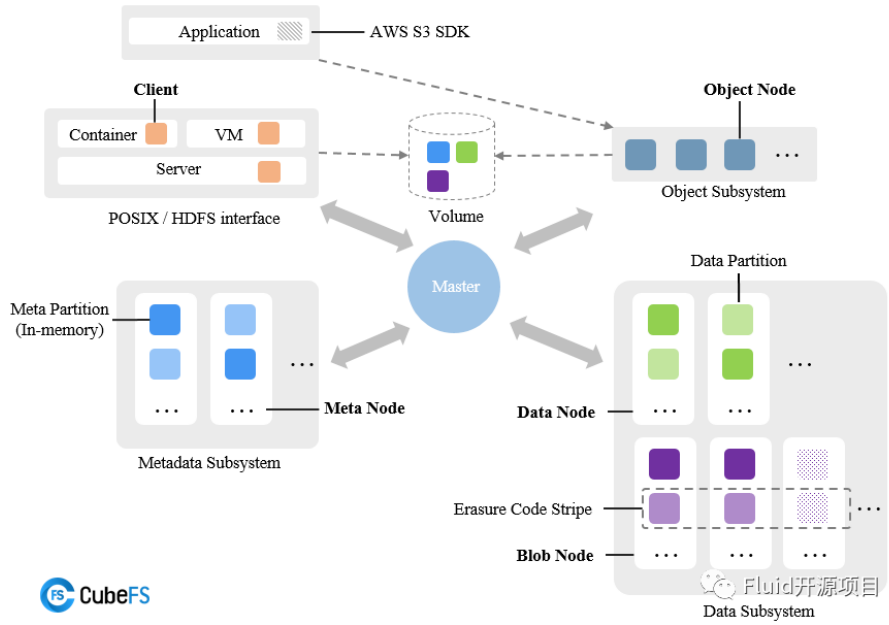
因此，Fluid ThinRuntime 被提出以支持通用存储系统快速接入，满足用户使用 Fluid 访问通用存储系统数据需求。与 Fluid 对接后，用户可复用 Fluid 已有的通用能力，如：

- 通用化 CSI Plugin 实现
- 通用化 Serverless 场景适用的 Fuse Sidecar 架构支持
- 数据集抽象管理

基于 ThinRuntime，用户可以通过编写 FUSE 客户端的方式，将自建存储系统对接到 Fluid，具体开发指南见社区(https://github.com/fluid-cloudnative/fluid/tree/master/addons)。如果有多个存储系统，或者存储系统版本出现不兼容的情况，Fluid 也可以很好的通过 ThinRuntime 的方式，为用户提供统一的 Kubernetes 标准 PVC 接入。

CubeFS 介绍

CubeFS (github：https://github.com/cubefs) 是新一代云原生开源存储产品，兼容 S3、POSIX、HDFS 等多种访问协议，支持多副本与纠删码两种存储引擎，为用户提供多租户、 多AZ部署、多级缓存加速等多种特性，广泛应用于大数据、AI、容器平台、数据库、中间件存算分离、数据共享以及数据保护等场景。CubeFS属云原生计算基金会（CNCF）托管的孵化阶段产品。



CubeFS具备以下特点：

- **可扩展**：元数据采用分片管理，具备强大的水平扩展能力；支持全量缓存，访问时延低；单集群可轻松支持EB级存储规模，单目录可支持亿级文件数。
- **强一致性**：根据不同的写入方式提供了不同的复制协议。当进行顺序写入时，使用NRW(N=W=3, R=1) 协议来保障吞吐。而在进行随机写入时，采用Multi-Raft的复制协议，以确保系统的性能和强一致性。
- **多协议支持**：兼容S3、POSIX、HDFS三种访问协议，不同协议共享底层的元数据及数据存储，有效提升数据复用效率；
- **多级缓存**：纠删码卷支持多级缓存加速，为热点数据提供更高的访问性能：
  - **本地缓存**：可以在Client机器上同机部署BlockCache组件，将本地磁盘作为本地缓存. 可以不经过网络直接读取本地Cache, 但容量受本地磁盘限制；
  - **全局缓存**：使用副本组件DataNode搭建的分布式全局Cache, 比如可以通过部署客户端同机房的SSD磁盘的DataNode作为全局cache, 相对于本地cache, 需要经过网络, 但是容量更大, 可动态扩缩容,副本数可调。
- **云原生支持**：基于CSI插件(https://cubefs.io/zh/docs/master/user-guide/k8s.html)可以快速地在Kubernetes上使用CubeFS，方便在多云上快速部署；同时CubeFS提供了丰富的可观测能力。

Fluid 结合 CubeFS

CubeFS 由于其稳定性、扩展性、易运维性、高性能等优点，南京大学云原生科研团队近年来在研究项目中其作为计算与存储分离的持久化存储方案。Fluid 与 CubeFS 配合使用可以提供如下几个优点：

1. 在同一个 K8s 集群，甚至同一个应用(Pod)中访问来自不同CubeFS存储的数据。
2. 解耦存储和 CSI 插件，从而降低了 Kubernetes运维的复杂性。
3. 以 FUSE Sidecar 的通用模式支持 Serverless 容器使用。例如，可以在阿里云 ASK 上运行。
4. 将 CubeFS 客户端独立运行在Pod中，从而使其处于 Kubernetes 的管控下，可观测性更强；同时以 Pod 为单元进行管理也可以增强隔离性，可以单独设置客户端的计算资源配额（CPU/Memory）等

场景案例分享

使用 Fluid ThinRuntime 为 CubeFS 提供应用 Serverless 访问能力，可以在同一个应用（Pod）访问 CubeFS 2.4 和 3.2 两个版本集群的数据。主要实施步骤如下：

1.安装CubeFS

1）CubeFS2 安装

通过 helm 安装 CubeFS2, cubefs2.yaml如下：

```
# 这里配置需要安装的组件，如果只安装服务端的话保持下方配置即可，如果要安装客户端的话，
component:
  master: true
  datanode: true
  metanode: true
  objectnode: false
  client: false
  csi: false
  provisioner: false
  monitor: false
  ingress: true

image:
  server: cubefs/cfs-server:2.5.2
  client: cubefs/cfs-client:2.5.2
  csi_driver: cubefs/cfs-csi-driver:2.4.0.110.4
  csi_provisioner: quay.io/k8scsi/csi-provisioner:v1.6.0
```

```
  driver_registrar: quay.io/k8scsi/csi-node-driver-registrar:v1.3.0
  csi_attacher: quay.io/k8scsi/csi-attacher:v2.0.0
  consul: consul:1.6.1
  pull_policy: "IfNotPresent"

# path.data: Master、MetaNode 的元数据存储路径，会以 hostPath 的方式存储在宿主机上
# path.log: 所有组件的日志在宿主机上的存储路径
path:
  data: /var/lib/cubefs2
  log: /var/log/cubefs2

master:
  replicas: 1
  nodeSelector:
    "component.cubefs2.io/master": "enabled"
  # port: 17012
  # prof: 17022
  # exporter_port: 9502

metanode:
  nodeSelector:
    "component.cubefs2.io/metanode": "enabled"
  total_mem: "10737418240"

datanode:
  nodeSelector:
    "component.cubefs2.io/datanode": "enabled"
  disks:
    - /data1/cubefs2:104857600

consul:
  port: 8502
  replicas: 1
  external_address: ""

provisioner:
  kubelet_path: /usr/bin/kubelet

```

执行以下命令安装 CubeFS2：

```
$ helm upgrade --install cubefs2 ./cubefs -f ./cubefs2.yaml -n cubefs2 --crea

$ kubectl get pods -n cubefs2
NAME              READY   STATUS    RESTARTS   AGE
datanode-nt252    1/1     Running   0          149m
datanode-xbdvz    1/1     Running   0          149m
master-0          1/1     Running   0          149m
metanode-b954z    1/1     Running   0          149m
metanode-z2vz8    1/1     Running   0          149m

$ ./cfs-cli config set --addr <IP:Port>
```

2）CubeFS3 安装

通过 helm 安装 CubeFS3, cubefs3.yaml如下：

```
# Select which component to install
component:
  master: true
  datanode: true
  metanode: true
  objectnode: false
  client: false
  provisioner: false
  monitor: false
  ingress: false

image:
  server: cubefs/cfs-server:v3.2.0
  client: cubefs/cfs-client:v3.2.0

  # CSI related images
  csi_driver: ghcr.io/cubefs/cfs-csi-driver:3.2.0.150.0
  csi_provisioner: registry.k8s.io/sig-storage/csi-provisioner:v2.2.2
  csi_attacher: registry.k8s.io/sig-storage/csi-attacher:v3.4.0
  csi_resizer: registry.k8s.io/sig-storage/csi-resizer:v1.3.0
  driver_registrar: registry.k8s.io/sig-storage/csi-node-driver-registrar:v2.
```

```
  grafana: grafana/grafana:6.4.4
  prometheus: prom/prometheus:v2.13.1
  consul: consul:1.6.1
  pull_policy: "IfNotPresent"

# store data,log and other data, these directory will be
#  mounted from host to container using hostPath
path:
  data: /var/lib/cubefs3
  log: /var/log/cubefs3

master:
  # Master 组件实例数量
  replicas: 1
  nodeSelector:
    "component.cubefs3.io/master": "enabled"
  port: 17013
  prof: 17023
  exporter_port: 9503

datanode:
  nodeSelector:
    "component.cubefs3.io/datanode": "enabled"
  port: 17313
  prof: 17323
  raft_heartbeat: 17333
  raft_replica: 17343
  exporter_port: 9523
  disks:
    - /data1/cubefs3:21474836480

metanode:
  nodeSelector:
    "component.cubefs3.io/metanode": "enabled"
  port: 17213
  prof: 17223
  raft_heartbeat: 17233
  raft_replica: 17243
  exporter_port: 9513
```

```
  # MetaNode 可以使用的总内存，单位字节，建议设置为机器可以内存的 80%，也可以按需减少
  total_mem: "10737418240"

consul:
  port: 8503
  replicas: 1
  external_address: ""

provisioner:
  kubelet_path: /usr/bin/kubelet
```

执行以下命令安装CubeFS3：

```
$ helm upgrade --install cubefs3 ./cubefs -f ./cubefs3.yaml -n cubefs3 --crea

$ kubectl get pods -n cubefs3
NAME              READY   STATUS    RESTARTS   AGE
datanode-kxc2f    1/1     Running   0          139m
datanode-thntv    1/1     Running   0          139m
master-0          1/1     Running   0          139m
metanode-qtlgh    1/1     Running   0          139m
metanode-r6j77    1/1     Running   0          139m

$ ./cfs-cli config set --addr <IP:Port>
$ ./cfs-cli volume create cubefs3-volume root --capacity 1
# 或者使用以下命令创建volume
# curl -v "http://<MasterIP:Port>/admin/createVol?name=cubefs3-volume&capaci
```

注意：创建卷时，需要指定卷的 owner 是 root，要不然无法挂载卷。

2.安装 Fluid

1.将 Fluid 仓库添加到 Helm 仓库中：

```
$ helm repo add fluid https://fluid-cloudnative.github.io/charts
```

2.更新以获取来自 Fluid 仓库的最新 Chart 信息：

```
$ helm repo update
```

3.搜索可用的版本：

```
helm search repo fluid
```

4.安装可用版本的 Fluid：

```
$ helm install fluid fluid/fluid
NAME: fluid
LAST DEPLOYED: Fri Sep  2 19:03:56 2022
NAMESPACE: default
STATUS: deployed
REVISION: 1
TEST SUITE: None
```

3.创建对应 Fluid Dataset 和 ThinRuntime

1) 为 CubeFS2 存储卷创建 Dataset 和 ThinRuntime

使用以下命令创建并部署 ThinRuntimeProfile 资源：

```
$ cat <<EOF > runtime-profile.yaml
apiVersion: data.fluid.io/v1alpha1
kind: ThinRuntimeProfile
metadata:
  name: cubefs2.4
spec:
  fileSystemType: cubefs
  fuse:
    image: fluidcloudnative/cubefs_v2.4
    imageTag: v0.1
```

```
    imagePullPolicy: IfNotPresent
    command:
      - "/usr/local/bin/entrypoint.sh"
EOF

$ kubectl apply -f runtime-profile.yaml

```

使用以下命令创建 Dataset 和 ThinRuntime：

```
$ cat <<EOF > dataset.yaml
apiVersion: data.fluid.io/v1alpha1
kind: Dataset
metadata:
  name: cubefs2
spec:
  mounts:
    - mountPoint: <IP:Port>
      name: cubefs2-volume
---
apiVersion: data.fluid.io/v1alpha1
kind: ThinRuntime
metadata:
  name: cubefs2
spec:
  profileName: cubefs2.4
EOF

$ kubectl apply -f dataset.yaml
```

将上述 mountPoint修改为您需要使用的 CuebFS2 集群Master 的地址，name 修改为需要挂载的存储卷的名字，此例中为 cubefs2-volume。

2) 为 CubeFS3 存储卷创建 Dataset 和 Thinruntime

使用以下命令创建并部署 ThinRuntimeProfile 资源：

```
$ cat <<EOF > runtime-profile.yaml
```

```
apiVersion: data.fluid.io/v1alpha1
kind: ThinRuntimeProfile
metadata:
  name: cubefs3.2
spec:
  fileSystemType: cubefs
  fuse:
    image: fluidcloudnative/cubefs_v3.2
    imageTag: v0.1
    imagePullPolicy: IfNotPresent
    command:
      - "/usr/local/bin/entrypoint.sh"
EOF

$ kubectl apply -f runtime-profile.yaml
```

使用以下命令创建 Dataset 和 ThinRuntime：

```
$ cat <<EOF > dataset.yaml
apiVersion: data.fluid.io/v1alpha1
kind: Dataset
metadata:
  name: cubefs3
spec:
  mounts:
    - mountPoint: <IP:Port>
      name: cubefs3-volume
---
apiVersion: data.fluid.io/v1alpha1
kind: ThinRuntime
metadata:
  name: cubefs3
spec:
  profileName: cubefs3.2
EOF

$ kubectl apply -f dataset.yaml
```

上述 mountPoint 修改为您需要使用的 CubeFS3 集群 Master 的地址，name 修改为需要挂载的存储卷的名字，此例中为 cubefs3-volume。

4.创建对应 Fluid Dataset 和 ThinRuntime

在 Serverless 场景下，用户只需要在应用 podSpec 或者 podTemplateSpec 中的 label 中配置 **serverless.fluid.io/inject: "true"** ，便可为 Serverless 应用提供数据访问能力。

使用以下命令创建数据访问应用示例：

```
$ cat <<EOF > app.yaml
apiVersion: v1
kind: Pod
metadata:
  name: nginx
  labels:
     serverless.fluid.io/inject: "true"
spec:
  containers:
    - name: nginx
      image: nginx:latest
      imagePullPolicy: IfNotPresent
      command: ["bash"]
      args:
        - -c
        - sleep 9999
      volumeMounts:
        - mountPath: /data2
          name: cubefs2
        - mountPath: /data3
          name: cubefs3
  volumes:
    - name: cubefs2
      persistentVolumeClaim:
        claimName: cubefs2
    - name: cubefs3
      persistentVolumeClaim:
        claimName: cubefs3
EOF

```

```
$ kubectl apply -f app.yaml
```

运行以上命令后，应用 Pod 将会自动创建 Fuse Pod，并将其调度到与应用相同的节点上。用户可以通过以下命令检查 Pod 状态：

```
$ kubectl get pods
NAME                  READY   STATUS    RESTARTS      AGE
nginx                 3/3     Running   0             2m21s
```

用户可以通过以下命令检查远程文件系统已被挂载到 nginx pod 的 /data2, /data3 目录下。

```
$ kubectl exec -it nginx -c nginx bash
root@nginx:/# df -h
...
cubefs-cubefs2-volume  1.0G     0  1.0G   0% /data2
cubefs-cubefs3-volume    1.0G     0  1.0G   0% /data3
...

root@nginx:/# ls -l /data2
total 0
-rw-r--r--. 1 root root 0 Apr 18 06:57 cubefs2-volume-access

root@nginx:/# ls -l /data3
total 0
-rw-r--r--. 1 root root 0 Apr 18 08:54 cubefs3-volume-access
```

总结

从上面的示例可以看出，通过使用 Fluid 能够简化 CubeFS 在 Kubernetes 场景下的数据访问，增强其对于 Serverless 场景支持，使数据访问更简单和可靠。此外用户无需过多关注存储或者 CSI 插件升级问题。感谢 Mingwei Gong（CubeFS committer，github id：GongWilliam）等人对本文技术方案与实践给予的支持与建议。

相关链接

更多 Fluid 和 CubeFS 相关介绍请参考以下链接：

- [1] Fluid：
  https://github.com/fluid-cloudnative/fluid
- [2] CubeFS：
  https://github.com/cubefs

👇👇 **戳原文，直达Fluid开源项目 GitHub 地址！**

阅读原文