Fluid Serverless 离线任务生命周期管理

一、背景

Fluid 在 0.7.0 版本通过 sidecar 注入 FUSE 容器的方式支持 Serverless 的在线服务场景。但在离线任务场景中，以 Kubernetes 原生的 Batch Job 和 Kubeflow 社区的 TFJob，MPIJob 为例，当 Pod 的业务容器完成任务并退出后，通常希望注入的 FUSE 容器也可以主动退出，从而使 Kubernetes 的 Job Controller 能够正确判断 Pod 所处的完成状态。然而，FUSE容器并没有提供主动退出机制。为了解决这类问题，Fluid 引入了新的组件能力来保证 Fuse Sidecar 安全退出。

二、方案设计

方案概述

需要引入FluidApp Contoller，该Controller会根据label标识选中对应的sidecar container，并结合user container 容器的状态来选择合适的时机将这些sidecar fuse container退出。

Controller逻辑

实现原理

事件监听

FluidApp Controller只会监听Pod的Update事件，并进一步进行事件过滤：

1. 如果 Pod 不是 Serverless pod，则忽略该事件；
   判断依据：pod 是否有 done.sidecar.fluid.io/inject: "true" label
2. 判断Pod的 **`restartPolicy`**，如果为 **`Always`**，则忽略该事件；
3. 判断本次 Update 是否判断其他非 Fluid-fuse Container 的容器是否都执行完毕，若没有执行完毕，则忽略该事件；
   判断依据：容器的退出码为 0（containerStatus.State.Terminated.ExitCode==0）
4. 判断Fluid-fuse Container是已经退出，若已退出，则忽略该事件；
   判断依据同上

经过以上条件过滤通过后的事件，过滤通过的事件会通知 Controller，触发相应的 Controller 逻辑。

事件处理

FluidApp Controller通过`kubectl exec`应用Pod Fluid-fuse Container执行`umount`命令退出。

问题

上述方案的问题：当集群中 pod 数量规模相当大的时候，FluidApp Controller 对 pod 的 listwatch 机制会给 APIServer 造成相当大的压力。

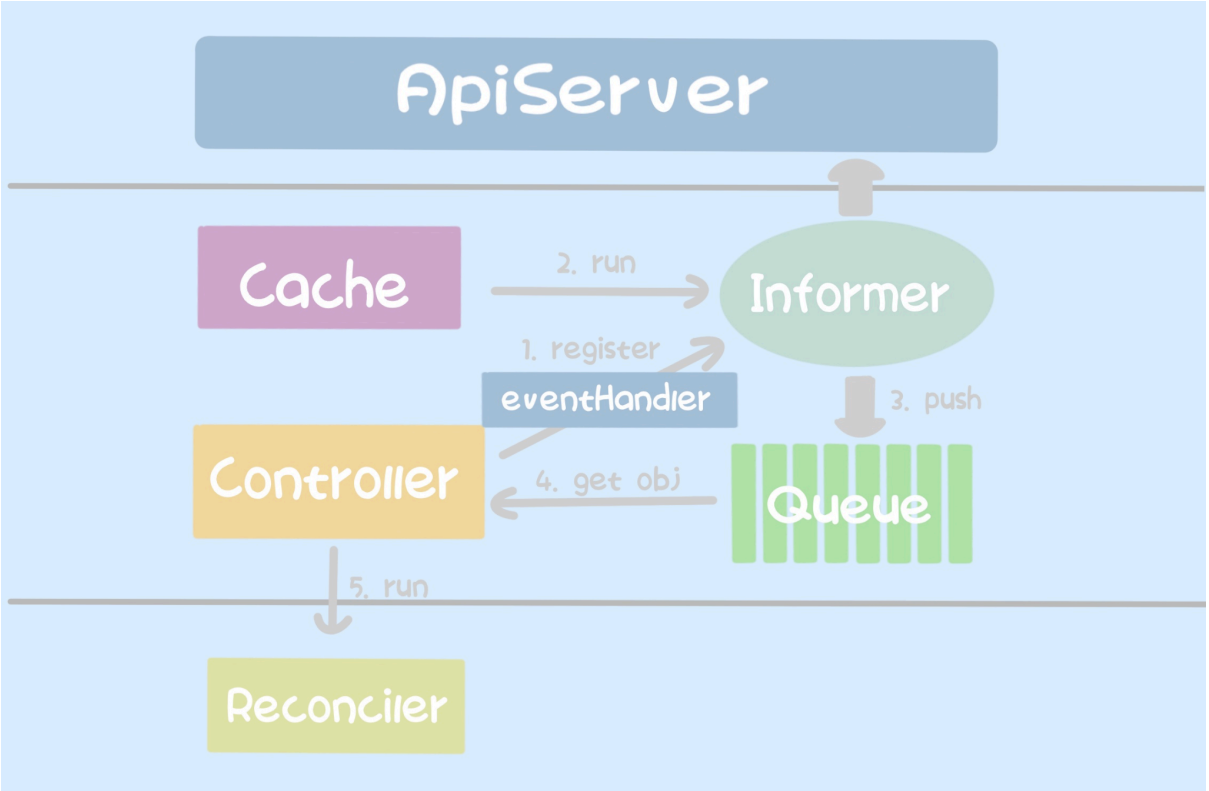
性能优化

1. cache selctor

controller-runtime 在 v0.11.0 版本允许在初始化 Controller 的时候自定义 cache selector

FluidApp Controller 初始化时加上 cache selector：done.sidecar.fluid.io/inject: "true"

设计文档：
https://github.com/kubernetes-sigs/controller-runtime/blob/master/designs/use-selectors-at-cache.md



Controller 在启动时，会初始化 informer 来访问 ApiServer，会调用两个接口，一个是 list 接口，一个是 watch 接口；然后将得到的资源推入队列，Controller 再从队列中消费资源并传给 Reconciler 执行。具体工作原理参考文章

2. 接口 protobuf 序列化格式

API 的默认序列化格式是 `application/json`，但 `application/vnd.kubernetes.protobuf` 序列化格式在大规模集群下有更好的性能。

在 Controller 架构中，用到的接口有 list pod 和 watch pod（informer），controller runtime 在 v0.11.0 版本中，将系统自带资源的接口序列化格式优化成了 protobuf：
https://github.com/kubernetes-sigs/controller-runtime/pull/1149/commits/7c26bc082e53eb784746b32b7067d4d1974f064f

三、使用流程

1 为 namespace 开启 webhook

Fluid webhook 提供了在 Serverless 场景中为 pod 注入 FUSE Sidecar 的功能，为了开启该功能，需要将对应的 namespace 打上 `fluid.io/enable-injection=true` 的标签。操作如下：

```
$ kubectl patch ns default -p '{"metadata": {"labels":
{"fluid.io/enable-injection": "true"}}}'
namespace/default patched
$ kubectl get ns default --show-labels
NAME      STATUS   AGE     LABELS
default   Active   4d12h
fluid.io/enable-injection=true,kubernetes.io/metadata.name=default
```

2. 创建 dataset 和 runtime

针对不同类型的 runtime 创建相应的 Runtime 资源，以及同名的 Dataset。这里以 JuiceFSRuntime 为例，具体可参考 文档，如下：

```
$ kubectl get juicefsruntime
NAME      WORKER PHASE   FUSE PHASE   AGE
jfsdemo   Ready          Ready        2m58s
$ kubectl get dataset
NAME      UFS TOTAL SIZE   CACHED   CACHE CAPACITY   CACHED PERCENTAGE
PHASE   AGE
jfsdemo   [Calculating]    N/A                       N/A
Bound   2m55s
```

3 创建 Job 资源对象

在 Serverless 场景使用 Fluid，需要在应用 Pod 中添加 `serverless.fluid.io/inject: "true"` label。如下：

```
$ cat<<EOF >sample.yaml
apiVersion: batch/v1
kind: Job
metadata:
  name: demo-app
spec:
  template:
    metadata:
      labels:
        serverless.fluid.io/inject: "true"
    spec:
      containers:
        - name: demo
          image: busybox
          args:
            - -c
            - echo $(date -u) >> /data/out.txt
          command:
            - /bin/sh
          volumeMounts:
            - mountPath: /data
              name: demo
      restartPolicy: Never
      volumes:
        - name: demo
          persistentVolumeClaim:
            claimName: jfsdemo
  backoffLimit: 4
EOF
$ kubectl create -f sample.yaml
job.batch/demo-app created
```

4 查看 job 是否完成

```
$ kubectl get job
NAME       COMPLETIONS   DURATION   AGE
demo-app   1/1           14s        46s
$ kubectl get po
NAME                READY   STATUS      RESTARTS      AGE
demo-app-wdfr8      0/2     Completed   0             25s
jfsdemo-worker-0    1/1     Running     0             14m
```

可以看到，job 已经完成，其 pod 有两个 container，均已完成。