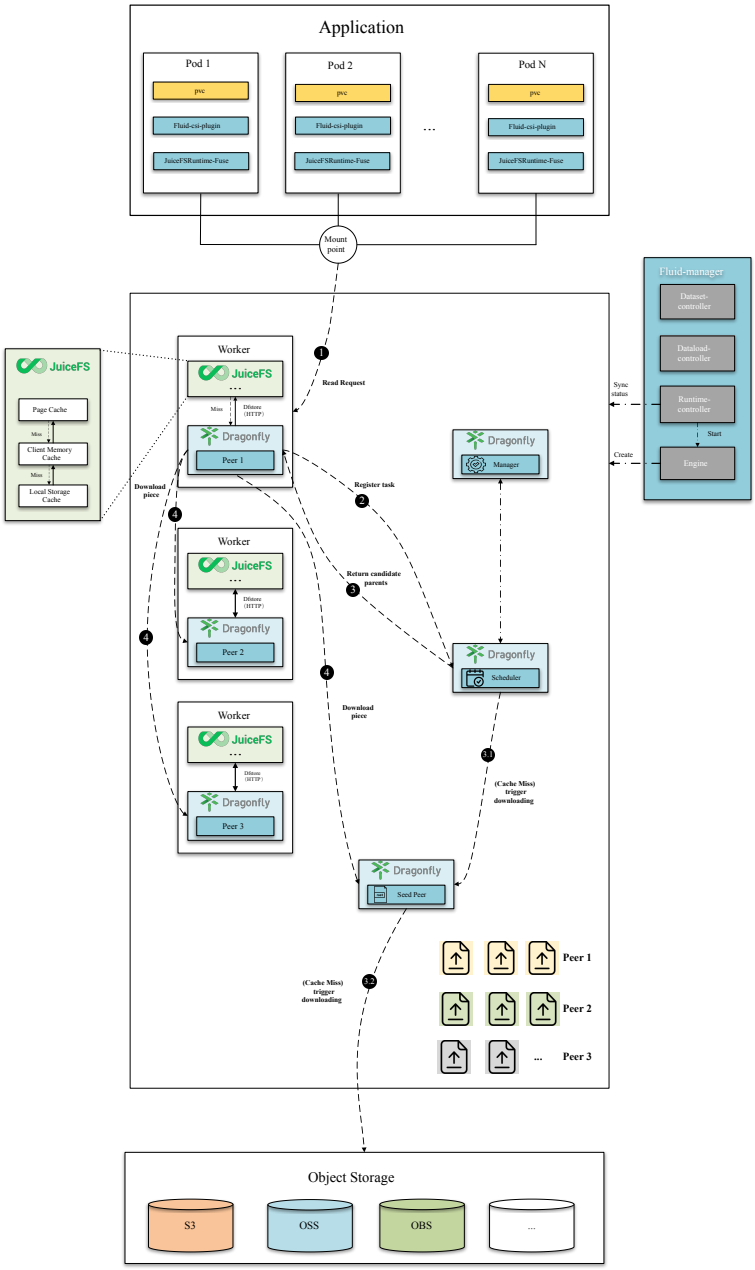
Fluid & JuiceFS & Dragonfly-EN V2

This document will help you experience how to use dragonfly with Fluid & JuiceFS Runtime.It introduces how to install Fluid & JuiceFS Runtime with Dragonfly and performance

testing.

Fluid supports Alluxio Runtime, JuiceFS Runtime, Thin Runtime, etc. However, Fluid still has scope for optimization for large file downloads at the same time. For example, Facebook LLaMa 65B which is 121.62 GB. If use the memory cache or disk cache on a single node, It is a hard work for the storage and network in the machine. Dragonfly Provides efficient, stable, secure file distribution and image acceleration based on p2p technology. It is designed to improve the efficiency and speed of large-scale file distribution. **JuiceFS depend on Dragonfly for object storage, and realizes the integration of Fluid and Dragonfly through JuiceFS Runtime..**

Architecture



Dragonfly becomes a new cache between JuiceFS and object storage. There are optimizations in the reading and writing. When reading, if there is no hit in the JuiceFS cache, the traffic will be forwarded to Dragonfly Peer. It can be used to eliminate the bandwidth limit of the object storage through P2P technology, thereby accelerating file downloading.

Install JuiceFS with Dragonfly

Dragonfly Kubernetes Cluster Setup

Setup Kubernetes Cluster

Kind is recommended if no Kubernetes cluster is available for testing.

Create kind multi-node cluster configuration file `kind-config.yaml`, configuration content is as follows:

```Shell
kind: Cluster
apiVersion: kind.x-k8s.io/v1alpha4
nodes:
  - role: control-plane
  - role: worker
  - role: worker
```

Create a kind multi-node cluster using the configuration file:

```Shell
kind create cluster --config kind-config.yaml
```

Kind loads dragonfly image

Pull dragonfly latest images:

```Shell
docker pull dragonflyoss/scheduler:latest
docker pull dragonflyoss/manager:latest
docker pull dragonflyoss/dfdaemon:latest
```

Kind cluster loads dragonfly latest images:

```Shell
kind load docker-image dragonflyoss/scheduler:latest
kind load docker-image dragonflyoss/manager:latest
kind load docker-image dragonflyoss/dfdaemon:latest
```

Create dragonfly cluster based on helm charts

Create helm charts configuration file `charts-config.yaml` and set `dfdaemon.config.proxy.proxies` and `seedPeer.config.proxy.proxies` to math endpoint of the object storage, configuration content is as follows:

YAML

```yaml
scheduler:
  image: dragonflyoss/scheduler
  tag: latest
  replicas: 1
  metrics:
    enable: true
  config:
    verbose: true
    pprofPort: 18066

seedPeer:
  image: dragonflyoss/dfdaemon
  tag: latest
  replicas: 1
  metrics:
    enable: true
  config:
    verbose: true
    pprofPort: 18066
    proxy:
      defaultFilter: 'Expires&Signature&ns'
      security:
        insecure: true
        tlsVerify: false
      tcpListen:
        # # Listen address.
        # listen: 0.0.0.0
        # Listen port, daemon will try to listen,
        # when this port is not available, daemon will try next port.
        port: 65001
        namespace: ""
      proxies:
        # Proxy all http download requests of the s3.
        - regx: s3.*amazonaws.com.*
        # Proxy all http download requests of the oss.
        - regx: oss.*aliyuncs.com.*
        # Proxy all http download requests of the obs.
        - regx: obs.*myhuaweicloud.com.*

dfdaemon:
  image: dragonflyoss/dfdaemon
  tag: latest
  metrics:
    enable: true
  config:
```

```yaml
    verbose: true
    pprofPort: 18066
    proxy:
      defaultFilter: 'Expires&Signature&ns'
      security:
        insecure: true
        tlsVerify: false
      tcpListen:
        # # Listen address.
        # listen: 0.0.0.0
        # Listen port, daemon will try to listen,
        # when this port is not available, daemon will try next port.
        port: 65001
        namespace: ""
      proxies:
        # Proxy all http download requests of the s3.
        - regx: s3.*amazonaws.com.*
        # Proxy all http download requests of the oss.
        - regx: oss.*aliyuncs.com.*
        # Proxy all http download requests of the obs.
        - regx: obs.*myhuaweicloud.com.*

manager:
  image: dragonflyoss/manager
  tag: latest
  replicas: 1
  metrics:
    enable: true
  config:
    verbose: true
    pprofPort: 18066

jaeger:
  enable: true
```

Create a dragonfly cluster using the configuration file:

```shell
$ helm repo add dragonfly https://dragonflyoss.github.io/helm-charts/
$ helm install --wait --create-namespace --namespace dragonfly-system dragonfly dragonfly/dragonfly -f charts-config.yaml
NAME: dragonfly
LAST DEPLOYED: Thu Sep 28 17:35:49 2023
NAMESPACE: dragonfly-system
STATUS: deployed
REVISION: 1
TEST SUITE: None
NOTES:
1. Get the scheduler address by running these commands:
  export SCHEDULER_POD_NAME=$(kubectl get pods --namespace dragonfly-system -l "app=dragonfly,release=dragonfly,component=scheduler" -o jsonpath={.items[0].metadata.name})
  export SCHEDULER_CONTAINER_PORT=$(kubectl get pod --namespace dragonfly-system $SCHEDULER_POD_NAME -o jsonpath="{.spec.containers[0].ports[0].containerPort}")
  kubectl --namespace dragonfly-system port-forward $SCHEDULER_POD_NAME 8002:$SCHEDULER_CONTAINER_PORT
  echo "Visit http://127.0.0.1:8002 to use your scheduler"

2. Get the dfdaemon port by running these commands:
  export DFDAEMON_POD_NAME=$(kubectl get pods --namespace dragonfly-system -l "app=dragonfly,release=dragonfly,component=dfdaemon" -o jsonpath={.items[0].metadata.name})
  export DFDAEMON_CONTAINER_PORT=$(kubectl get pod --namespace dragonfly-system $DFDAEMON_POD_NAME -o jsonpath="{.spec.containers[0].ports[0].containerPort}")
  You can use $DFDAEMON_CONTAINER_PORT as a proxy port in Node.

3. Configure runtime to use dragonfly:
  https://d7y.io/docs/getting-started/quick-start/kubernetes/


4. Get Jaeger query URL by running these commands:
  export JAEGER_QUERY_PORT=$(kubectl --namespace dragonfly-system get services dragonfly-jaeger-query -o jsonpath="{.spec.ports[0].port}")
  kubectl --namespace dragonfly-system port-forward service/dragonfly-jaeger-query 16686:$JAEGER_QUERY_PORT
  echo "Visit http://127.0.0.1:16686/search?limit=20&lookback=1h&maxDuration&minDuration&service=dragonfly to query download events"
```

Check that dragonfly is deployed successfully:

```shell
$ kubectl get po -n dragonfly-system
NAME                                 READY   STATUS    RESTARTS        AGE
dragonfly-dfdaemon-65rz7             1/1     Running   5 (6m17s ago)   8m43s
dragonfly-dfdaemon-rnvsj             1/1     Running   5 (6m23s ago)   8m43s
dragonfly-jaeger-7d58dfcfc8-qmn8c    1/1     Running   0               8m43s
dragonfly-manager-6f8b4f5c66-qq8sd   1/1     Running   0               8m43s
dragonfly-mysql-0                    1/1     Running   0               8m43s
dragonfly-redis-master-0             1/1     Running   0               8m43s
dragonfly-redis-replicas-0           1/1     Running   0               8m43s
dragonfly-redis-replicas-1           1/1     Running   0               7m33s
dragonfly-redis-replicas-2           1/1     Running   0               5m50s
dragonfly-scheduler-0                1/1     Running   0               8m43s
dragonfly-seed-peer-0                1/1     Running   3 (5m56s ago)   8m43s
```

Expose Dragonfly Peer Proxy service port

Create the `proxy.yaml` configuration to expose the port on which the Dragonfly Peer Proxy listens. The default port is `65001` and set `targetPort` to `65001`.

```yaml
kind: Service
apiVersion: v1
metadata:
  name: proxy
spec:
  selector:
    app: dragonfly
    component: dfdaemon
    release: dragonfly

  ports:
  - protocol: TCP
    port: 65001
    targetPort: 65001

  type: NodePort
```

Create service:

```
kubectl --namespace dragonfly-system apply -f proxy.yaml
```

Forward request to Dragonfly Peer Proxy:

```shell
kubectl --namespace dragonfly-system port-forward service/proxy 65001:65001
```

Install JuiceFS

For detailed installation documentation, please refer to JuiceFS document. For Linux and macOS systems, you can use a one-click installation script that automatically downloads and installs the latest version of the JuiceFS client based on your hardware architecture.

```shell
# 默认安装到 /usr/local/bin
curl -sSL https://d.juicefs.com/install | sh -
```

After installation, you can specify the use of Dragonfly as the object storage when executing commands such as `juicefs format` and `juicefs config`:

```shell
juicefs format \
    --storage dragonfly \
    --access-key ABCDEFGHIJKLMNopqXYZ \
    --secret-key ZYXwvutsrqpoNMLkJiHgfeDCBA \
    --bucket "https://myjfs.oss-cn-shanghai.aliyuncs.com?proxy=http://127.0.0.1:65001&backendStorage=oss" \
    redis://192.168.1.6:6379/1 \
    myjfs
```

`--storage`: Set the object storage type, the value is `dragonfly`.

`--bucket`: Set the object storage endpoint.

`--access-key`: Set the object storage access key.

`--secret-key`: Set the object storage secret key.

Dragonfly Peer Proxy address and Backend object storage type are added to the query string in the `--bucket`. The details of parameter is as follows:

Param	Type	Describe	Required
backendStorage	string	Backend object storage type, supports s3, oss and obs.	Y
proxy	string	Draognfly Peer Proxy address.	Y

Verify the created file system status:

```shell
$ juicefs status redis://192.168.1.6:6379/1
2023/10/17 19:09:35.738635 juicefs[2273224] <INFO>: Meta address: redis://localhost:6379/1 [interface.go:498]
2023/10/17 19:09:35.739344 juicefs[2273224] <WARNING>: AOF is not enabled, you may lose data if Redis is not shutdown properly. [info.go:84]
2023/10/17 19:09:35.739407 juicefs[2273224] <INFO>: Ping redis latency: 22.384µs [redis.go:3572]
{
  "Setting": {
    "Name": "myjfs",
    "UUID": "316d39df-a7ba-4cde-8cc7-5568a7a0f745",
    "Storage": "dragonfly",
    "Bucket": "https://myjfs.oss-cn-shanghai.aliyuncs.com?proxy=http://127.0.0.1:65001\u0026backendStorage=oss",
    "BlockSize": 4096,
    "Compression": "none",
    "EncryptAlgo": "aes256gcm-rsa",
    "TrashDays": 1,
    "MetaVersion": 1,
    "MinClientVersion": "1.1.0-A",
    "DirStats": true
  },
  "Sessions": [],
  "Statistic": {
    "UsedSpace": 0,
    "AvailableSpace": 1125899906842624,
    "UsedInodes": 0,
    "AvailableInodes": 10485760
  }
}

```

When using other JuiceFS commands, you can also specify Dragonfly as the object storage. For detailed JuiceFS commands documentation, plese refer to document.

Verify

Execute the command:

```shell
juicefs objbench \
    --storage dragonfly \
    --access-key ABCDEFGHIJKLMNopqXYZ \
    --secret-key ZYXwvutsrqpoNMLkJiHgfeDCBA \
    https://myjfs.oss-cn-shanghai.aliyuncs.com?proxy=http://127.0.0.1:65001&backendStorage=oss
```

The endpoint is `https://myjfs.oss-cn-shanghai.aliyuncs.com?proxy=http://127.0.0.1:65001&backendStorage=oss`. It should pass the unit test for `teststorage`.

```
+----------+--------------------+-----------------------------------------
----------+
| CATEGORY |        TEST        |                    RESULT
          |
+----------+--------------------+-----------------------------------------
----------+
|    basic |     create a bucket |
     pass |
|    basic |       put an object |
     pass |
|    basic |       get an object |
     pass |
|    basic |       get non-exist |
     pass |
|    basic |  get partial object | failed to get object with the offset ou
t of r... |
|    basic |      head an object |
     pass |
|    basic |    delete an object |
     pass |
|    basic |    delete non-exist |
     pass |
|    basic |        list objects |
     pass |
|    basic |         special key |
     pass |
|     sync |    put a big object |
     pass |
|     sync | put an empty object |
     pass |
|     sync |    multipart upload |
     pass |
|     sync |  change owner/group |                                      no
t support |
|     sync |   change permission |                                      no
t support |
|     sync |        change mtime |                                      no
t support |
+----------+--------------------+-----------------------------------------
----------+
```

Multi-Node Read Performance Testing

JuiceFS

Test the caching performance of JuiceFS. The configured object storage needs to be the same as in Dragonfly.

```shell
juicefs format \
    --storage oss \
    --bucket https://myjfs.oss-cn-shanghai.aliyuncs.com \
    --access-key ABCDEFGHIJKLMNopqXYZ \
    --secret-key ZYXwvutsrqpoNMLkJiHgfeDCBA \
    redis://192.168.1.6:6379/2 \
    myjfs
```

Mount the file system using the `juicefs mount` command:

```shell
juicefs mount redis://192.168.1.6:6379/2  /mnt/jfs
```

Creat a 1GB file in the mounted directory:

```shell
$ time dd if=/dev/zero of=/mnt/jfs/test.txt bs=1M count=1000
1000+0 records in
1000+0 records out
1048576000 bytes (1.0 GB, 1000 MiB) copied, 10.7013 s, 98.0 MB/s
dd if=/dev/zero of=/mnt/jfs/test.txt bs=1M count=1000  0.00s user 0.33s sys
tem 3% cpu 10.711 tota
```

For the first read, JuiceFS triggers **back-to-source download** and it takes **11.356 seconds.**

```shell
$ time cp /mnt/jfs/test.txt /dev/null

cp /mnt/jfs/test.txt /dev/null  0.00s user 0.29s system 2% cpu 11.356 total
```

Clear the page cache and read again. Hit **JuiceFS's cache**, and it takes **0.347 seconds**.

```shell
$ sync && echo 3 > /proc/sys/vm/drop_caches
$ time cp /mnt/jfs/test.txt /dev/null
cp /mnt/jfs/test.txt /dev/null  0.00s user 0.30s system 86% cpu 0.347 total
```

JuiceFS & Dragonfly

Test the performance of Dragonfly cache and hit local peer cache and remote peer cache. Expose Draogngfly Peer's `65001` port.

```shell
export dragonfly_dfdaemon_name=$(kubectl get po -n dragonfly-system | grep dragonfly-dfdaemon- | tail -n 1 | awk '{print $1}')
kubectl --namespace dragonfly-system port-forward $dragonfly_dfdaemon_name 65001:65001
```

Initialize the file system based on Dragonfly:

```shell
juicefs format \
    --storage dragonfly \
    --access-key ABCDEFGHIJKLMNopqXYZ \
    --secret-key ZYXwvutsrqpoNMLkJiHgfeDCBA \
    --bucket "https://myjfs.oss-cn-shanghai.aliyuncs.com?proxy=http://127.0.0.1:65001&backendStorage=oss" \
    redis://192.168.1.6:6379/1 \
    myjfs
```

Mount the file system and disable JuiceFS's cache:

```shell
juicefs mount redis://192.168.1.6:6379/1  /mnt/jfs --cache-size=0
```

Create a 1GB file in the mounted directory:

```shell
$ time dd if=/dev/zero of=/mnt/jfs/test.txt bs=1M count=1000
1000+0 records in
1000+0 records out
1048576000 bytes (1.0 GB, 1000 MiB) copied, 10.2689 s, 102 MB/s
dd if=/dev/zero of=/mnt/jfs/test.txt bs=1M count=1000  0.00s user 0.38s system 3% cpu 10.271 total
```

For the first read. No cache hits for JuiceFS and Dragonfly, and it triggers **back-to-source download**, taking **11.147 seconds.**

```shell
$ time cp /mnt/jfs/test.txt /dev/null
cp /mnt/jfs/test.txt /dev/null  0.00s user 0.30s system 2% cpu 11.147 total
```

Clear the cache of the file system and read again. Hit the cache of **Dragonfly's Local Peer** and it takes **1.554 seconds.**

```shell
$ sync && echo 3 > /proc/sys/vm/drop_caches
$ time cp /mnt/jfs/test.txt /dev/null
cp /mnt/jfs/test.txt /dev/null  0.00s user 0.32s system 20% cpu 1.554 total
```

Test the cache speed of the hit Dragonfly Remote Peer, delete the Peer:

```shell
$ export dragonfly_dfdaemon_name=$(kubectl get po -n dragonfly-system --sort-by=.metadata.creationTimestamp | grep "dragonfly-dfdaemon-" | awk '{print $1}' | tail -n 1)
$ kubectl delete po $dragonfly_dfdaemon_name -n dragonfly-system
$ kubectl get po -n dragonfly-system
NAME                                 READY   STATUS    RESTARTS   AGE
dragonfly-dfdaemon-5q4r8             1/1     Running   0          30s # new pod
dragonfly-dfdaemon-nhzcc             1/1     Running   0          19m
dragonfly-jaeger-c7947b579-q4hr4     1/1     Running   0          19m
dragonfly-manager-5dc5fbf548-zrf7d   1/1     Running   0          19m
dragonfly-mysql-0                    1/1     Running   0          19m
dragonfly-redis-master-0             1/1     Running   0          19m
dragonfly-redis-replicas-0           1/1     Running   0          19m
dragonfly-redis-replicas-1           1/1     Running   0          18m
dragonfly-redis-replicas-2           1/1     Running   0          18m
dragonfly-scheduler-0                1/1     Running   0          19m
dragonfly-seed-peer-0                1/1     Running   0          19m
```

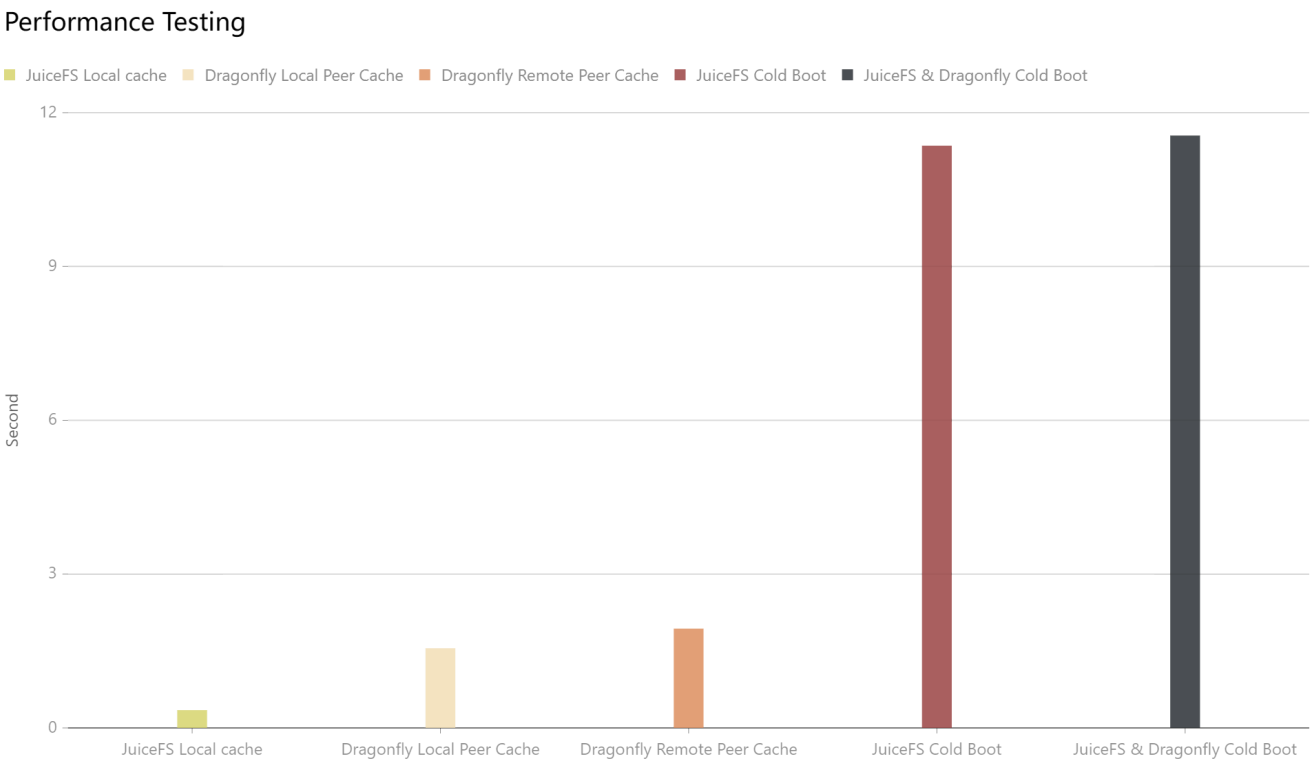
Recreate the pod:

```shell
kubectl --namespace dragonfly-system port-forward $dragonfly_dfdaemon_name 65001:65001
```

Clear the cache of the file system and read again. The created Pod has no cache, and it hits the cache of the Remote Peer, it takes **1.937 seconds.**

```shell
$ sync && echo 3 > /proc/sys/vm/drop_caches
$ time cp /mnt/jfs/test.txt /dev/null
cp /mnt/jfs/test.txt /dev/null  0.01s user 0.32s system 16% cpu 1.937 total
```

Analysis



Test results show JuiceFS and Dragonfly integration. It can effectively reduce the file download time. Due to the influence of the network environment of the machine itself, the actual download time is not important, but the ratio of the increase in the download time in different scenarios is very important.

Install Fluid & JuiceFS Runtime with Dragonfly

Dragonfly Kubernetes Cluster Setup

Setup Kubernetes Cluster

Kind is recommended if no Kubernetes cluster is available for testing.

Create kind multi-node cluster configuration file `kind-config.yaml`, configuration content is as follows:

```shell
kind: Cluster
apiVersion: kind.x-k8s.io/v1alpha4
nodes:
  - role: control-plane
  - role: worker
  - role: worker
```

Create a kind multi-node cluster using the configuration file:

```shell
kind create cluster --config kind-config.yaml
```

Kind loads dragonfly image

Pull dragonfly latest images:

```shell
docker pull dragonflyoss/scheduler:latest
docker pull dragonflyoss/manager:latest
docker pull dragonflyoss/dfdaemon:latest
```

Kind cluster loads dragonfly latest images:

```shell
kind load docker-image dragonflyoss/scheduler:latest
kind load docker-image dragonflyoss/manager:latest
kind load docker-image dragonflyoss/dfdaemon:latest
```

Create dragonfly cluster based on helm charts

Create helm charts configuration file `charts-config.yaml` and set `dfdaemon.config.proxy.proxies` and `seedPeer.config.proxy.proxies` to math endpoint of the object storage, configuration content is as follows:

YAML

```yaml
scheduler:
  image: dragonflyoss/scheduler
  tag: latest
  replicas: 1
  metrics:
    enable: true
  config:
    verbose: true
    pprofPort: 18066

seedPeer:
  image: dragonflyoss/dfdaemon
  tag: latest
  replicas: 1
  metrics:
    enable: true
  config:
    verbose: true
    pprofPort: 18066
    proxy:
      defaultFilter: 'Expires&Signature&ns'
      security:
        insecure: true
        tlsVerify: false
      tcpListen:
        # # Listen address.
        # listen: 0.0.0.0
        # Listen port, daemon will try to listen,
        # when this port is not available, daemon will try next port.
        port: 65001
        namespace: ""
      proxies:
        # Proxy all http download requests of the s3.
        - regx: s3.*amazonaws.com.*
        # Proxy all http download requests of the oss.
        - regx: oss.*aliyuncs.com.*
        # Proxy all http download requests of the obs.
        - regx: obs.*myhuaweicloud.com.*

dfdaemon:
  image: dragonflyoss/dfdaemon
  tag: latest
  metrics:
    enable: true
  config:
```

```yaml
    verbose: true
    pprofPort: 18066
    proxy:
      defaultFilter: 'Expires&Signature&ns'
      security:
        insecure: true
        tlsVerify: false
      tcpListen:
        # # Listen address.
        # listen: 0.0.0.0
        # Listen port, daemon will try to listen,
        # when this port is not available, daemon will try next port.
        port: 65001
        namespace: ""
      proxies:
        # Proxy all http download requests of the s3.
        - regx: s3.*amazonaws.com.*
        # Proxy all http download requests of the oss.
        - regx: oss.*aliyuncs.com.*
        # Proxy all http download requests of the obs.
        - regx: obs.*myhuaweicloud.com.*

manager:
  image: dragonflyoss/manager
  tag: latest
  replicas: 1
  metrics:
    enable: true
  config:
    verbose: true
    pprofPort: 18066

jaeger:
  enable: true
```

Create a dragonfly cluster using the configuration file:

```shell
$ helm repo add dragonfly https://dragonflyoss.github.io/helm-charts/
$ helm install --wait --create-namespace --namespace dragonfly-system dragonfly dragonfly/dragonfly -f charts-config.yaml
NAME: dragonfly
LAST DEPLOYED: Thu Sep 28 17:35:49 2023
NAMESPACE: dragonfly-system
STATUS: deployed
REVISION: 1
TEST SUITE: None
NOTES:
1. Get the scheduler address by running these commands:
  export SCHEDULER_POD_NAME=$(kubectl get pods --namespace dragonfly-system -l "app=dragonfly,release=dragonfly,component=scheduler" -o jsonpath={.items[0].metadata.name})
  export SCHEDULER_CONTAINER_PORT=$(kubectl get pod --namespace dragonfly-system $SCHEDULER_POD_NAME -o jsonpath="{.spec.containers[0].ports[0].containerPort}")
  kubectl --namespace dragonfly-system port-forward $SCHEDULER_POD_NAME 8002:$SCHEDULER_CONTAINER_PORT
  echo "Visit http://127.0.0.1:8002 to use your scheduler"

2. Get the dfdaemon port by running these commands:
  export DFDAEMON_POD_NAME=$(kubectl get pods --namespace dragonfly-system -l "app=dragonfly,release=dragonfly,component=dfdaemon" -o jsonpath={.items[0].metadata.name})
  export DFDAEMON_CONTAINER_PORT=$(kubectl get pod --namespace dragonfly-system $DFDAEMON_POD_NAME -o jsonpath="{.spec.containers[0].ports[0].containerPort}")
  You can use $DFDAEMON_CONTAINER_PORT as a proxy port in Node.

3. Configure runtime to use dragonfly:
  https://d7y.io/docs/getting-started/quick-start/kubernetes/


4. Get Jaeger query URL by running these commands:
  export JAEGER_QUERY_PORT=$(kubectl --namespace dragonfly-system get services dragonfly-jaeger-query -o jsonpath="{.spec.ports[0].port}")
  kubectl --namespace dragonfly-system port-forward service/dragonfly-jaeger-query 16686:$JAEGER_QUERY_PORT
  echo "Visit http://127.0.0.1:16686/search?limit=20&lookback=1h&maxDuration&minDuration&service=dragonfly to query download events"
```

Check that dragonfly is deployed successfully:

```shell
$ kubectl get po -n dragonfly-system
NAME                                 READY   STATUS    RESTARTS        AGE
dragonfly-dfdaemon-65rz7             1/1     Running   5 (6m17s ago)   8m43s
dragonfly-dfdaemon-rnvsj             1/1     Running   5 (6m23s ago)   8m43s
dragonfly-jaeger-7d58dfcfc8-qmn8c    1/1     Running   0               8m43s
dragonfly-manager-6f8b4f5c66-qq8sd   1/1     Running   0               8m43s
dragonfly-mysql-0                    1/1     Running   0               8m43s
dragonfly-redis-master-0             1/1     Running   0               8m43s
dragonfly-redis-replicas-0           1/1     Running   0               8m43s
dragonfly-redis-replicas-1           1/1     Running   0               7m33s
dragonfly-redis-replicas-2           1/1     Running   0               5m50s
dragonfly-scheduler-0                1/1     Running   0               8m43s
dragonfly-seed-peer-0                1/1     Running   3 (5m56s ago)   8m43s
```

Expose Dragonfly Peer Proxy service port

Create the `proxy.yaml` configuration to expose the port on which the Dragonfly Peer Proxy listens. The default port is `65001` and set `targetPort` to `65001`.

```yaml
kind: Service
apiVersion: v1
metadata:
  name: proxy
spec:
  selector:
    app: dragonfly
    component: dfdaemon
    release: dragonfly

  ports:
  - protocol: TCP
    port: 65001
    targetPort: 65001

  type: NodePort
```

Create service:

```
kubectl --namespace dragonfly-system apply -f proxy.yaml
```

Forward request to Dragonfly Peer Proxy:

```shell
kubectl --namespace dragonfly-system port-forward service/proxy 65001:65001
```

Install Fluid

Create Fluid cluster based on helm charts

For detailed installation documentation, please refer to document.

Create namespace:

```shell
$ kubectl create ns fluid-system
```

Create a dragonfly cluster:

```shell
$ helm repo add fluid https://fluid-cloudnative.github.io/charts
$ helm repo update
$ helm install fluid fluid/fluid
NAME: fluid
LAST DEPLOYED: Thu Oct 12 21:54:34 2023
NAMESPACE: default
STATUS: deployed
REVISION: 1
TEST SUITE: None
```

Check that Fluid is deployed successfully:

```shell
$ kubectl get po -n fluid-system
NAME                                  READY   STATUS    RESTARTS   AGE
csi-nodeplugin-fluid-nq65p            2/2     Running   0          7m12s
csi-nodeplugin-fluid-nrwbt            2/2     Running   0          7m12s
csi-nodeplugin-fluid-q565r            2/2     Running   0          7m12s
dataset-controller-5f5f46d969-lpsc7   1/1     Running   0          7m11s
fluid-webhook-75f489c7b5-whzjz        1/1     Running   0          7m12s
fluidapp-controller-54975849ff-w272h  1/1     Running   0          7m12s
```

Create Dataset

Create the `secret.yaml` :

```shell
apiVersion: v1
kind: Secret
metadata:
  name: jfs-secret
type: Opaque
stringData:
  name: dragonfly
  metaurl: redis://127.0.0.1:6379/3
  token: ${JUICEFS_TOKEN}
  access-key: ${ACCESS_KEY}
  secret-key: ${SECRET_KEY}
```

Create a secret using the configuration:

```shell
$ kubectl create -f secret.yaml
```

Create `dataset.yaml` :

```yaml
$ cat<<EOF >dataset.yaml
apiVersion: data.fluid.io/v1alpha1
kind: Dataset
metadata:
  name: jfsdemo
spec:
  mounts:
    - name: dragonfly
      mountPoint: "juicefs:///"
      options:
        bucket: "'https://myjfs.oss-cn-shanghai.aliyuncs.com?proxy=http://127.0.0.1:65001&backendStorage=oss'"
        storage: "dragonfly"
      encryptOptions:
        - name: metaurl
          valueFrom:
            secretKeyRef:
              name: jfs-secret
              key: metaurl
        - name: token
          valueFrom:
            secretKeyRef:
              name: jfs-secret
              key: token
        - name: access-key
          valueFrom:
            secretKeyRef:
              name: jfs-secret
              key: access-key
        - name: secret-key
          valueFrom:
            secretKeyRef:
              name: jfs-secret
              key: secret-key
EOF
```

Where:

- `mountPoint` : The directory where users store data in the JuiceFS file system, starting

with `juicefs://`. For example, `juicefs:///demo` is a subdirectory `/demo` of the JuiceFS file system.

- `storage`: Set the object storage type, the value is `dragonfly`.
- `bucket`: Set the object storage endpoint.

Dragonfly Peer Proxy address and Backend object storage type are added to the query string in the `--bucket`. The details of parameter is as follows:

Param	Type	Describe	Required
backendStorage	string	Backend object storage type, supports s3, oss and obs.	Y
proxy	string	Draognfly Peer Proxy address.	Y

Create a Dataset:

```shell
$ kubectl create -f dataset.yaml
dataset.data.fluid.io/jfsdemo created
```

Create JuiceFS Runtime

Create `runtime.yaml`:

```shell
$ cat<<EOF >runtime.yaml
apiVersion: data.fluid.io/v1alpha1
kind: JuiceFSRuntime
metadata:
  name: jfsdemo
spec:
  replicas: 1
  fuse:
    image: dragonflyoss/juicefs-fuse
    imageTag: 0.1.0
    imagePullPolicy: IfNotPresent
  juicefsVersion:
    image: dragonflyoss/juicefs-fuse
    imageTag: 0.1.0
    imagePullPolicy: IfNotPresent
  tieredstore:
    levels:
      - mediumtype: MEM
        path: /dev/shm
        quota: 40Gi
        low: "0.1"
EOF
```

Create JuiceFS Runtime:

```shell
$ kubectl create -f runtime.yaml
juicefsruntime.data.fluid.io/jfsdemo created
```

JuiceFS Runtime to start successfully:

```shell
$ kubectl get po |grep jfs
jfsdemo-worker-0                                          1/1     Running
 0          4m2s
```

Check the dataset status, it has been bound to JuiceFS Runtime:

```shell
$ kubectl get dataset jfsdemo
NAME      UFS TOTAL SIZE   CACHED   CACHE CAPACITY   CACHED PERCENTAGE   PHASE   AGE
jfsdemo   0.00B            0.00B    40.00GiB         0.0%                Bound   21h
```

Fluid has created PV and PVC with the same name as the dataset.

```shell
$ kubectl get pv | grep jfs
default-jfsdemo       100Pi      ROX            Retain           Bound    default/jfsdemo                                  fluid                    16h
$ kubectl get pvc
NAME      STATUS   VOLUME            CAPACITY   ACCESS MODES   STORAGECLASS   AGE
jfsdemo   Bound    default-jfsdemo   100Pi      ROX            fluid          21h
```

Verify

Create an application to use the dataset:

```shell
$ cat<<EOF >app.yaml
apiVersion: v1
kind: Pod
metadata:
  name: demo-app
spec:
  containers:
    - name: demo
      image: nginx
      volumeMounts:
        - mountPath: /demo
          name: demo
  volumes:
    - name: demo
      persistentVolumeClaim:
        claimName: jfsdemo
EOF
```

Create application:

```shell
$ kubectl create -f app.yaml
```

Check that the Pod has been created:

```shell
$ kubectl get pod
NAME                READY   STATUS    RESTARTS   AGE
demo-app            1/1     Running   0          40s
jfsdemo-fuse-vfqgt  1/1     Running   0          40s
```

The Pod has been created successfully and JuiceFS's FUSE component has also started successfully.

Reference

1. JuiceFS Install
2. JuiceFS Performance Testing
3. JuiceFS Command Reference

4. Dragonfly Quick Start
5. Dragonfly Helm chart configuration file
6. How to Use JuiceFS in Fluid