CNCF Webinar

Apache Flink on Kubernetes Operator

https://github.com/GoogleCloudPlatform/flink-on-k8s-operator/

Aniket Mokashi - Tech Lead Manager
Dagang Wei - Software Engineer
Cloud Dataproc, Google
February 2020

Outline

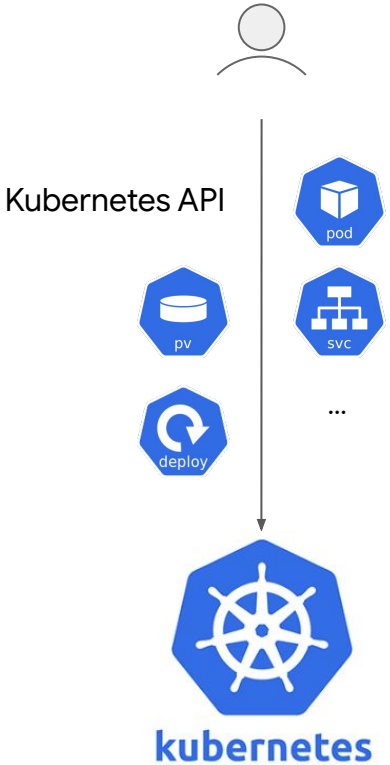
- Why we need the Flink Operator
- Architecture and features
- Beam on Flink Operator

beam

Apache Flink

kubernetes

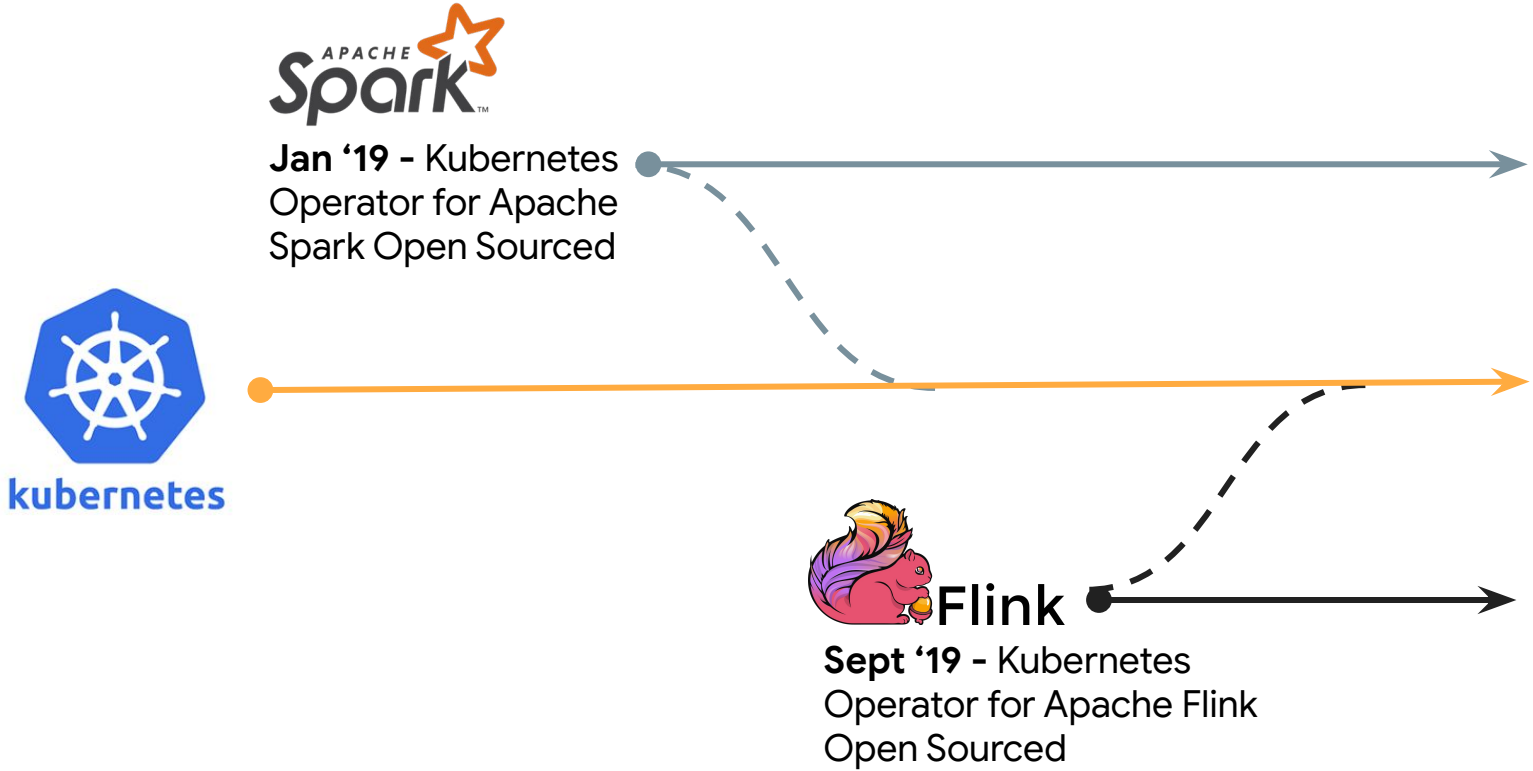
Kubernetes



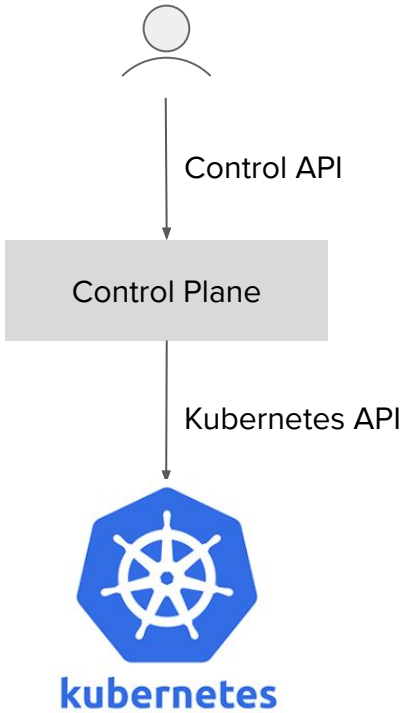
- Kubernetes
  - a general-purpose cluster manager for all sorts of containerized microservice applications.
  - makes it easy to develop distributed systems through its constructs such as Pod, Service, PersistentVolume, Deployment...

The trend of running OSS on Kubernetes

History

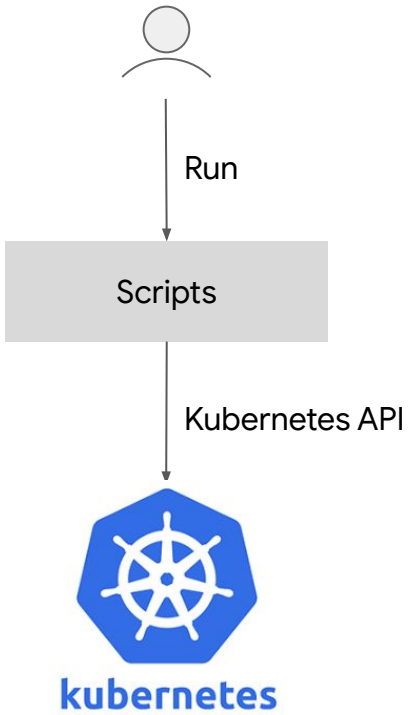

APACHE
Spark
Jan ‘19 - Kubernetes Operator for Apache Spark Open Sourced
kubernetes
Flink
Sept ‘19 - Kubernetes Operator for Apache Flink Open Sourced

Building control planes for Kubernetes applications



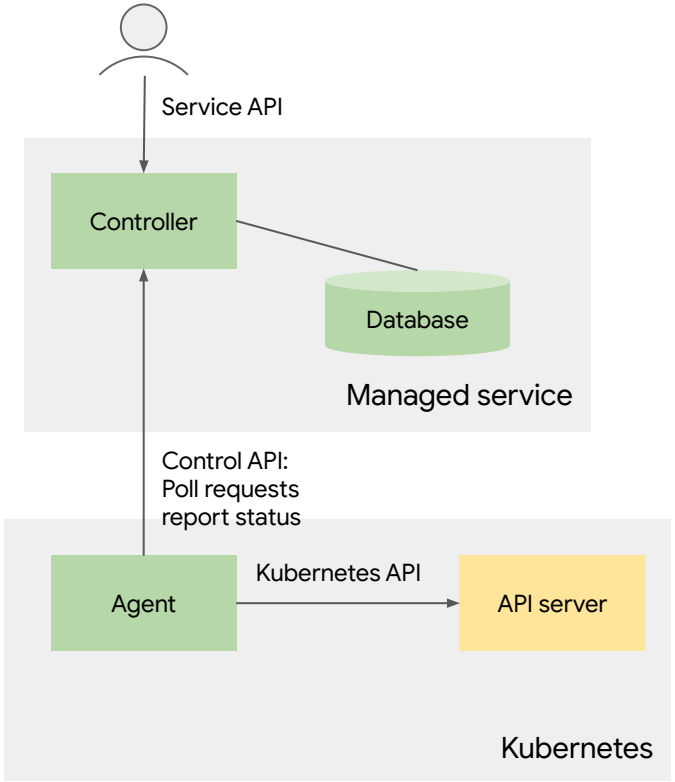
- There is a need for control planes when running complex application on top of Kubernetes.
  - Provide higher-level APIs
  - Manage application state
  - Manage the lifecycle of Kubernetes resources
- Problem: no standard
  - Developers solve same problems in different ways: programming languages, databases, APIs, CLIs, monitoring, logging, etc.
  - Users get different experiences

Scripts as control planes



- Simple, but insufficient for complex applications.

Managed services: control planes in the cloud



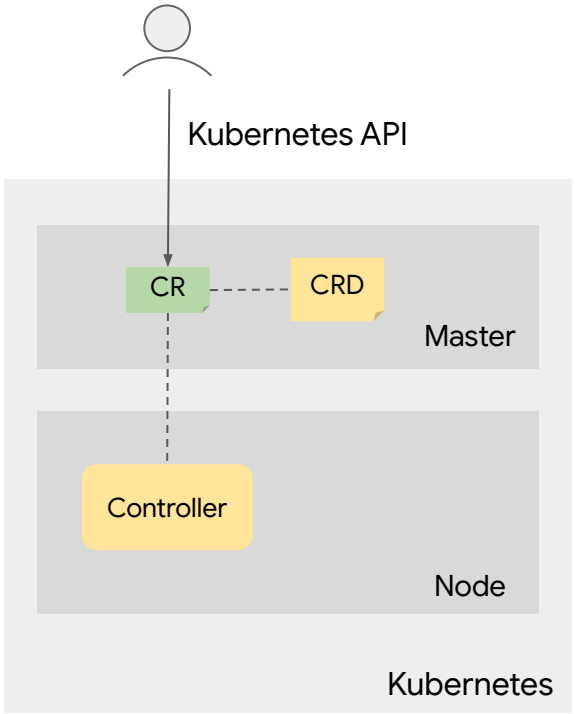
- Cloud providers traditionally help users run OSS with managed services.
  - Example: Google Cloud Dataproc
  - Runs on the internal infrastructure of the cloud provider
  - Controller provides a service API, implements the control logic
  - Database stores resource metadata (spec / status)
  - Agent runs in Kubernetes, polls requests from controller, translates them to Kubernetes API calls, and reports status back

Two Alternatives

	Scripts	Managed services	???
Open source	No	No	
Features	Poor	Rich	
Dev cost	Low	High	
Support	Self	Cloud provider	
Price	Free	Not free	
Portable	Yes	No	

Is there a third alternative?

Kubernetes Operators: extending the Kubernetes control plane



- Kubernetes Operator
  - **Extends** the Kubernetes control plane with Custom Resource Definitions and Custom Resource Controllers
  - **Reuses** the Kubernetes infrastructure (resource model, API, reconciliation loop)
- Benefits
  - **Developers:** easy to develop
  - **Users:** work with higher-level of abstraction (e.g., FlinkCluster) while preserving the native Kubernetes experience (e.g., kubectl, monitoring)

Kubernetes Operators: a good trade-off

	Scripts	Managed services	Operators
Open source	No	No	Yes
Features	Poor	Rich	Rich
Dev cost	Low	High	Medium
Support	Self	Cloud provider	Community
Price	Free	Not free	Free
Portable	Yes	No	Yes

Outline

- Why we need the Flink Operator
- Architecture and features
- Beam on Flink Operator

beam

Apache Flink

kubernetes

Apache Spark, Flink and Beam

a general-purpose distributed computing engine (mainly) for **batch** data processing.

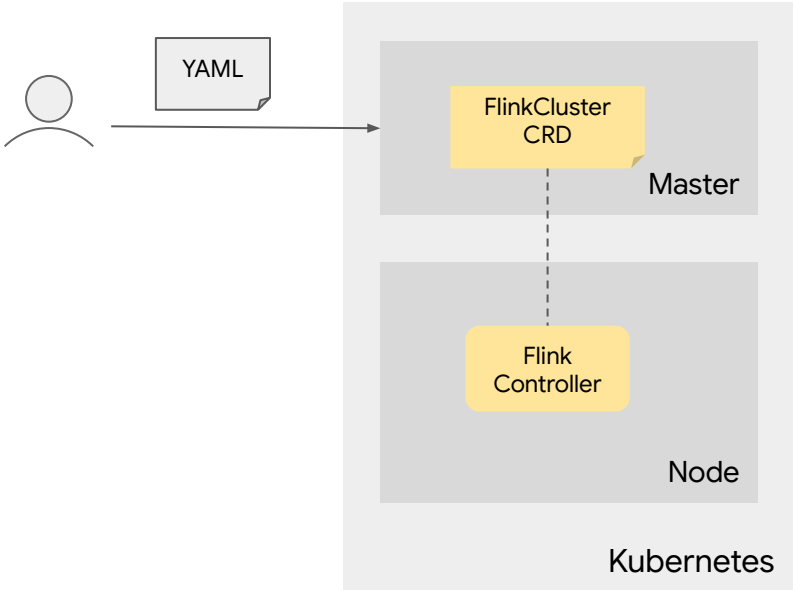
a distributed stateful computing engine over unbounded and bounded data **streams**.

a unified programming **model** for batch and streaming processing, which supports various programming **languages** and runs on any execution **engine**.

Flink Operator: managing Flink applications and more

- Goals
  - Managing the lifecycle of Flink applications
  - Running Beam Python jobs
  - Integrating with major cloud services

Quickstart: installation




```
kubectl apply -f flink-operator-v1beta1.yaml
```

Components

- **CRD**: defines the FlinkCluster custom resource
- **Controller**: watches the CR events, runs the reconciliation loop to continuously drive the observed state to the desired state.

Quickstart: running a Flink job

```
apiVersion: flinkoperator.k8s.io/v1beta1
kind: FlinkCluster
metadata:
  name: my-flinkjobcluster
spec:
  image:
    name: flink:1.8.2
  jobManager:
    resources:
      limits:
        memory: "1024Mi"
        cpu: "200m"
  taskManager:
    replicas: 2
    resources:
      limits:
        memory: "1024Mi"
        cpu: "200m"
  job:
    jarFile: ./examples/streaming/WordCount.jar
    className:
org.apache.flink.streaming.examples.wordcount.WordCount
    args: ["--input", "./README.txt"]
    parallelism: 2
  flinkProperties:
    taskmanager.numberOfTaskSlots: "1"
```

`kubectl apply -f <my-flinkjobcluster.yaml>`

Spec

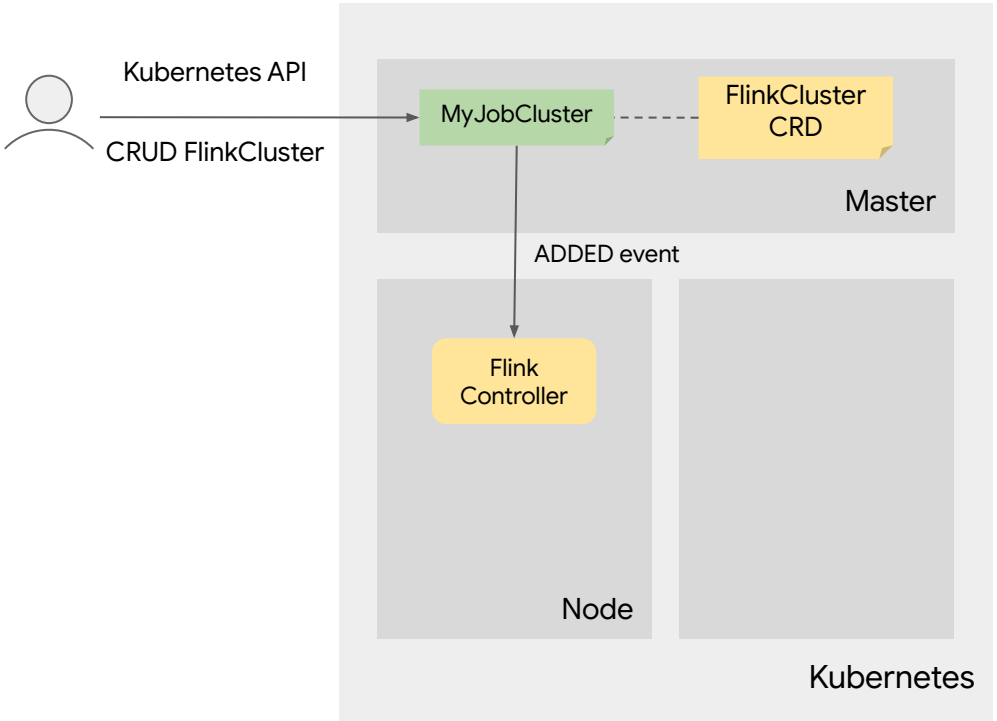
- Resource: number of replicas, memory and CPU
- Software: Flink image, JAR file, class name, arguments
- Config: Flink properties

Quickstart: checking job status and events

```
Name:         my-flinkjobcluster
Metadata:
  ...
Spec:
  ...
Status:
  Components:
    Config Map:
      Name:   flinkjobcluster-sample-configmap
      State:  Deleted
    Job:
      Id:     9076d259c3bb9002b52b3b4a9a4d5790
      Name:   flinkjobcluster-sample-job
      State:  Succeeded
    Job Manager Deployment:
      Name:   flinkjobcluster-sample-jobmanager
      State:  Deleted
    Job Manager Service:
      Name:   flinkjobcluster-sample-jobmanager
      State:  Deleted
    Task Manager Deployment:
      Name:          flinkjobcluster-sample-taskmanager
      State:         Deleted
  Last Update Time:  2020-02-10T23:20:33Z
  State:             Stopped
Events:
  ...
```

`kubectl describe flinkclusters <name>`

Flink Operator Architecture (1/3)



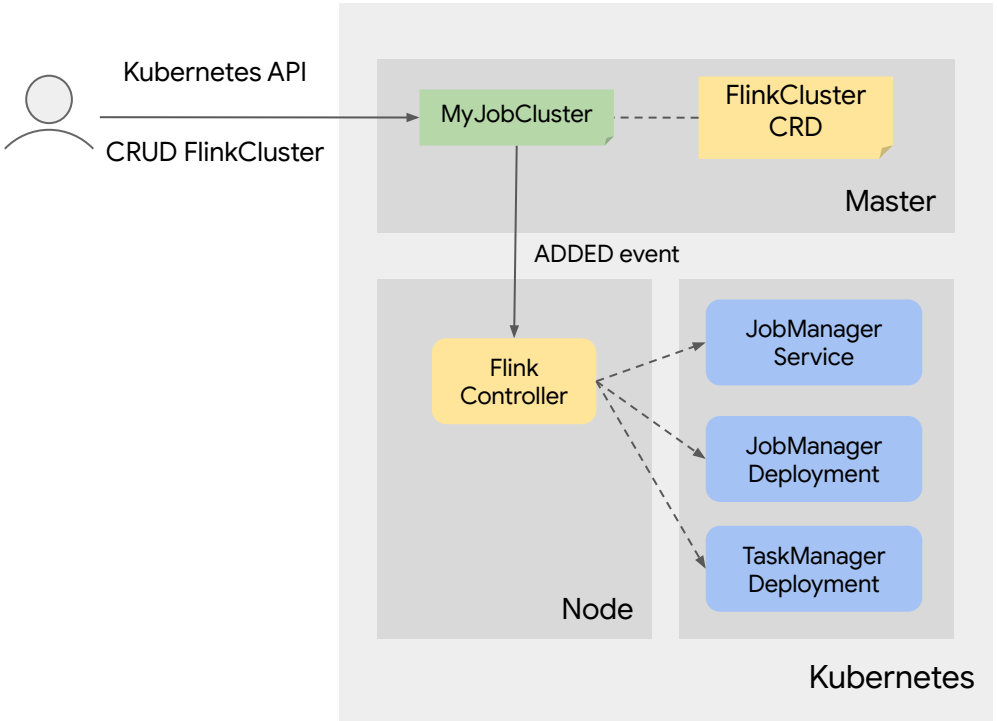
0. The Flink Operator (including CRD and Controller) has been deployed in the cluster.

1. The user runs `kubectl apply -f myjobcluster.yaml` which sends a `FlinkCluster` spec to the API server.

2. API server validates the spec against on the CRD, then creates a `FlinkCluster` CR and stores it in etcd.

3. A `FlinkCluster` ADDED event is triggered by Kubernetes and dispatched to the Flink Controller.

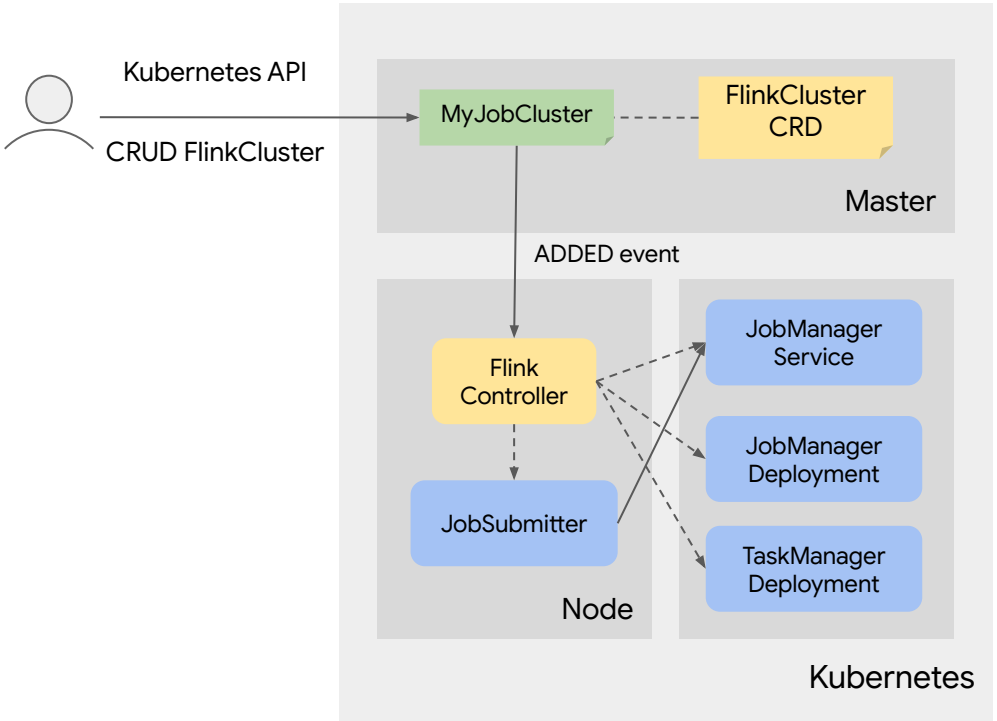
Flink Operator Architecture (2/3)



4. The Flink Controller analyzes the `FlinkCluster` CR, then calls the API server to create the underlying primitive resources (JobManager service, JobManager deployment, TaskManager deployment).

5. The controller implements the reconciliation loop: watches the status changes of the primitive resources, updates the status field of the CR accordingly, continuously take actions to drive the observed state to the desired state when needed.

Flink Operator Architecture (3/3)



6. When JobManager deployment, JobManager service, TaskManager deployment are all ready, the controller creates a Flink job submitter which submits the job to Flink REST API through the JobManager service.

7. The JobSubmitter keeps polling the job status from the Flink REST API, finishes itself when the job is completed or failed.

8. After the job is done, the controller deletes all the resources (JM, TM) for the job, but the job cluster metadata is kept.

Feature: session cluster and job cluster

```
apiVersion: flinkoperator.k8s.io/v1beta1
kind: FlinkCluster
metadata:
  name: flinkjobcluster-sample
spec:
  image:
    name: flink:1.8.2
  jobManager:
    ports:
      ui: 8081
    resources:
      limits:
        memory: "1024Mi"
        cpu: "200m"
  taskManager:
    replicas: 2
    resources:
      limits:
        memory: "1024Mi"
        cpu: "200m"
  job:
    jarFile: ./examples/streaming/WordCount.jar
    className:
org.apache.flink.streaming.examples.wordcount.WordCount
    args: ["--input", "./README.txt"]
    parallelism: 2
```

- Job spec is optional
  - Session cluster: only cluster spec, no job spec
  - Job cluster: cluster spec + job spec

Feature: init containers and remote job JAR

```
apiVersion: flinkoperator.k8s.io/v1beta1
kind: FlinkCluster
metadata:
  name: flinkjobcluster-gcs
spec:
  image:
    name: flink:1.8.2
  jobManager:
    ...
  taskManager:
    ...
  job:
    jarFile: /cache/wordcount.jar
    volumes:
      - name: cache-volume
        emptyDir: {}
    volumeMounts:
      - mountPath: /cache
        name: cache-volume
    initContainers:
      - name: gcs-downloader
        image: google/cloud-sdk
        command: ["gsutil"]
        args:
          - "cp"
          - "gs://my-bucket/wordcount.jar"
          - "/cache/wordcount.jar"
...
```

- Flink job JAR could be downloaded from a remote storage with an init container. This allows reusing the Flink image, no need to rebuild just to include the job JAR.

Feature: taking savepoints automatically

```
apiVersion: flinkoperator.k8s.io/v1beta1
kind: FlinkCluster
metadata:
  name: flinkjobcluster-sample
spec:
  image:
    name: flink:1.8.2
  jobManager:
    ...
  taskManager:
    ...
  job:
    autoSavepointSeconds: 300
    savepointsDir: gs://my-bucket/savepoints/
    ...
```

- If you specify `autoSavepointSeconds` and `savepointsDir`, the operator could take savepoints automatically for you.
- The locations of the saved savepoints are recorded in the status field.

Feature: restarting jobs from the lastest savepoint

```
apiVersion: flinkoperator.k8s.io/v1beta1
kind: FlinkCluster
metadata:
  name: flinkjobcluster-sample
spec:
  ...
  job:
    autoSavePointSeconds: 300
    savepointsDir: gs://my-bucket/savepoints/
    restartPolicy: FromSavepointOnFailure
    ...
```

- Long-running jobs may fail for various reasons, if you specify `restartPolicy` to `FromSavepointOnFailure`, the operator can automatically restart failed jobs from the latest savepoint.

Feature: sidecar containers

```
apiVersion: flinkoperator.k8s.io/v1beta1
kind: FlinkCluster
metadata:
  name: beam-flink
spec:
  image:
    name: flink:1.8.1
  jobManager:
    resources:
      limits:
        memory: "1Gi"
  taskManager:
    replicas: 2
    resources:
      limits:
        memory: "2Gi"
    sidecars:
      - name: beam-sdk-worker
        image: apachebeam/python3.7_sdk:2.18.0
        args: ["--worker_pool"]
  flinkProperties:
    taskmanager.numberOfTaskSlots: "1"
```

- You can run sidecar containers along with TM containers to provide services or proxies for your job.
- This is the enabling feature for Beam on Flink Operator.

Outline

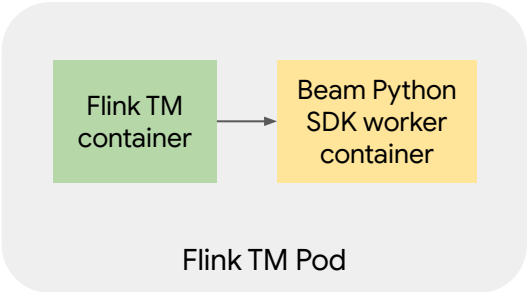
- Why we need the Flink Operator
- Architecture and features
- Beam on Flink Operator

beam

Apache Flink

kubernetes

How Flink runs Beam UDFs in Python



- Beam Python job -> Flink Java job + Python UDFs
- Beam runners are run by Flink TMs.
- UDFs in Python are run by Beam Python SDK workers:
  - Process mode
    - Beam runner in each Flink TM will automatically launch a Beam SDK worker process.
    - This requires a custom Flink image with Beam SDK builtin.
  - Docker mode
    - Beam runner in each Flink TM will automatically launch a Beam SDK worker container.
    - This requires running Docker in Docker on Kubernetes.
  - External mode
    - Beam runner doesn't launch Beam SDK worker by itself, but sends a request to an external WorkerPool service to launch one.
    - **This is the mode we choose** to run Beam Python jobs with the operator.

Beam on Flink Operator (1/3): creating a Flink session cluster

```
apiVersion: flinkoperator.k8s.io/v1beta1
kind: FlinkCluster
metadata:
  name: beam-flink
spec:
  image:
    name: flink:1.8.1
  jobManager:
    resources:
      limits:
        memory: "1Gi"
  taskManager:
    replicas: 2
    resources:
      limits:
        memory: "2Gi"
    sidecars:
      - name: beam-sdk-worker
        image: apachebeam/python3.7_sdk:2.18.0
        args: ["--worker_pool"]
  flinkProperties:
    taskmanager.numberOfTaskSlots: "1"
```

`kubectl apply -f <flinksessioncluster.yaml>`

- Create a Flink session cluster which run Beam Python SDK workers as sidecar containers with Flink TM containers.

Beam on Flink Operator (2/3): creating a Beam job submitter

```
apiVersion: batch/v1
kind: Job
metadata:
  name: beam-wordcount-py
spec:
  template:
    metadata:
      labels:
        app: beam-wordcount-py
    spec:
      containers:
        - name: beam-wordcount-py
          image: apachebeam/python3.7_sdk:2.18.0
          command: ["python3"]
          args:
            - "-m"
            - "apache_beam.examples.wordcount"
            - "--runner=FlinkRunner"
            - "--flink_master=beam-flink-jobmanager:8081"
            - "--flink_submit_uber_jar"
            - "--environment_type=EXTERNAL"
            - "--environment_config=localhost:50000"
            - "--input"
            - "/etc/ucf.conf"
            - "--output"
            - "/tmp/output"
```

`kubectl apply -f <beam-job.yaml>`

- Create a Kubernetes job which submits the Beam Python job to the Flink session cluster.

Beam on Flink Operator (3/3): checking job status and logs

```
Name:           beam-wordcount-py
Namespace:      default
Selector:
controller-uid=71e8ab00-d343-4928-8c92-ed80ed36a170
Labels:         app=beam-wordcount-py

controller-uid=71e8ab00-d343-4928-8c92-ed80ed36a170
                job-name=beam-wordcount-py
Annotations:    ...
Parallelism:    1
Completions:    1
Start Time:     Tue, 11 Feb 2020 15:22:47 -0800
Pods Statuses:  1 Running / 0 Succeeded / 0 Failed
Pod Template:
  ...
Events:
  Type    Reason            Age    From            Message
  ----    ------            ----   ----            -------
  Normal  SuccessfulCreate  4m53s  job-controller  Created
pod: beam-wordcount-py-86fwp
```

kubectl describe jobs <name>

kubectl logs jobs/<name>

Q & A

beam

Apache Flink

kubernetes