RUN_SCENARIO(all-in-one) English

This document describes the steps to execute scenarios for the all-in-one included in the OpenKasugai-controller project. The execution steps for the scenarios may vary slightly depending on the commands executed during cluster construction. Please refer to the OpenKasugai-controller documentation below about details on each scenario, how to build images, and architecture.

- https://github.com/openkasugai/controller/tree/main/docs/Demonstrations/jp/OpenKasugai-Demo-for-All-in-One.pdf

Prerequisites

- The all-in-one cluster construction must be completed.
- The following images must be registered in a local container registry (referred to as `harbor` in this document) that is accessible from the kind-host. The configuration for the local registry is `DCI_REGISTRY_ADDR_ALT` in the Makefile.
  - Four arithmetic operation scenarios
    - `cpufunc_calcapp:1.0.0`
  - Video inference scenarios
    - `cpufunc_gst:1.0.0`
    - `gpufunc_dsa:1.0.0`

Steps to Execute the Four arithmetic Operation Scenario

When the cluster is built using all-in-one, the `./build` directory is created. This directory contains the files used during the cluster construction. Files related to the arithmetic operation scenario are placed in the following directory.

- `./build/openkasugai-controller/test/sample-data/sample-data-for-all-in-one/calc-func/`

Applying Various Resources

If you performed the `Procedure for manually registering resources` during the installation, you will need to manually register various resources necessary for executing the DataFlow. The files required for executing the data flow, their roles, and the order of application are as follows.

Application Order	File Name	K8s Resource Type	Description
1	cm-cpufunc-config.yaml	ConfigMap	Stores detailed configuration information necessary for function execution
2	functioninfo.yaml	ConfigMap	Stores information about the type of accelerator to execute functions and the connection methods supported by the function
3	functiontype.yaml	FunctionType	Defines the functions available on OpenKasugai from the Config and Info of the function
4	functionchain.yaml	FunctionChain	Defines the combinations of functions
5	df-cpufunc.yaml	DataFlow	Deploys the DataFlow specifying the FunctionChain

Applying the Data Flow

If you performed the `Bulk build procedure` during the installation, the application of the DataFlow is executed using the make command.

Make Command	Used Resources	Overview
make dataflow-calcapp-basic	cpu	Executes the arithmetic operation scenario using the cluster IP
make df-dataflow-calcapp-sriov	cpu, sr-iov NIC	Executes the arithmetic operation scenario using a virtual NIC via SR-IOV
make dataflow-<PatternName>-multi-worker	-	Distributes the functions of the above DataFlow across kind-worker and kind-worker2

Verifying DataFlow Operation

Verification of the arithmetic operation scenario is performed using the data sending function and data receiving function within the DataFlow.

Data Input

Data input for the arithmetic operation scenario is performed by executing the application from the Pod of the data sending function.

```
sudo kubectl exec -it -n cpufunc-calcapp df-calcapp-wbfunction-send-cpu-pod -- /calcapp 1 2 3 4 5
```

Checking Execution Results

The results of the scenario execution are verified by checking the standard output of the receiving function's Pod.

```
sudo kubectl logs -n cpufunc-calcapp df-calcapp-wbfunction-rcv-cpu-pod -c cpu-container0
```

This is an example of the standard output result below.

> `2024-12-10T06:58:35.585Z INFO workspace/main.go:312 start call Do {"req": "inputs:1 inputs:2 inputs:3 inputs:4 inputs:5 results:{operator:\"pluse\" value:15} results:{operator:\"minus\" value:-13} results:{operator:\"multiply\" value:120} results:{operator:\"divide\" value:0.008333333333333333} results: {operator:\"average\" value:30.502083333333335}"}`

Steps to Execute the Video Inference Scenario

Files related to the video inference scenarios are placed in the following directory.

- `./build/openkasugai-controller/test/sample-data/sample-data-for-all-in-one/cpugpu-func`

Applying Various Resources

If you performed the `Procedure for manually registering resources` during the installation, you will need to manually register various resources necessary for executing the DataFlow. The roles of each file and the order of resource application are the same as in the arithmetic operation scenarios.

Applying the Data Flow

If you performed the `Bulk build procedure` during the installation, the application of the DataFlow is executed using the make command.

Make Command	Used Resources	Overview
make dataflow-p1c1	cpu	Internal NW connection pattern
make dataflow-p1c2	cpu	External NW connection pattern
make dataflow-p2c1	cpu,gpu	GPU collaboration internal NW connection pattern
make dataflow-p2c2	cpu,gpu	GPU collaboration internal NW connection pattern
make dataflow-p2c3	cpu,gpu,sr-iov NIC	GPU collaboration external NW SR-IOV connection pattern
make dataflow-p2c4	cpu,gpu	GPU collaboration internal NW multiple connection patterns
make dataflow-p3c1	cpu,sr-iov NIC	SR-IOV connection pattern
make dataflow-<PatternName>-multi-worker	-	Distributes the functions of the above DataFlow across kind-worker and kind-worker2

Verifying DataFlow Operation

Data input to the video inference data flow is performed by sending video data from the video distribution Pod to the DataFlow. The video distribution Pod is automatically created when the cluster is created. The processing results from the data flow are verified by playing the mp4 file inside the video receiving container's Pod.

Data Input

Identify the IP address to send video from the video distribution Pod. The destination should be the `EXTERNAL-IP` of the first function in the DataFlow The first function can be confirmed by referring to `functionchain.yaml`.

```
sudo kubectl get service -n cpufunc-sample df-cpu-pXcY-wbfunction-pip-service
```

```
NAME                          TYPE        CLUSTER-IP     EXTERNAL-IP     PORT(S)        AGE
df-cpu-pXcY-wbfunction-pip-service   LoadBalancer   10.103.16.154   172.18.9.0      5678:30751/UDP   6m6s
```

Execute bash on the pods receiving video.

```
sudo kubectl exec -it -n test send-video-tool -- /bin/bash
```

Execute the distribution command. Specify the IP address identified above for `udpsink host`.

This example is that `sample.mp4` is located in `./work/DATA/video/` . The `./work/DATA/video` directory is mounted as `/opt/video` in the container.

```
gst-launch-1.0 -e -v filesrc location=/opt/video/sample.mp4  \
 ! qtdemux  \
 ! video/x-h264  \
 ! h264parse  \
 ! rtph264pay config-interval=-1 seqnum-offset=1  \
 ! udpsink host=172.18.9.0 port=5678 buffer-size=2048
```

Checking Execution Results

To obtain the video received by the video receiving function, execute the following command.

```
sudo kubectl cp -n cpufunc-sample -c cpu-container0 df-cpu-pXcY-wbfunction-rcv-cpu-pod:rcv_video.mp4 rcv_video.mp4
```