OpenKasugai Demo

v1.1.0

Contents

0. Introduction

This document describes the demonstration procedure of OpenKasugai-Controller v1.1.0. It is assumed that the demo execution environment has been configured according to the saparate "OpenKasugai-Controller-InstallManual”.

1. Demonstration procedure of basic function #1

This chapter demonstrates one of the basic features of OpenKasugai-Controller v1.1.0, which automatically writes Bitstreams (child bs) to FPGA devices during DataFlow deployment. This demonstration shows the procedure for an example in which an FPGA based DataFlow is deployed to the initial system and a second DataFlow is deployed after the deployment is complete so that both DataFlow use the same deployment area (Lane) of the same FPGA device.

1.1 Advance preparation

1.1.1 Deployment of test scripts

Target : K8s control plane, all K8s nodes

Deploy the scripts used in this document to simplify the evaluation procedures (various shell scripts and YAML files for deploying video streaming and receiving tools).

1. Copy the test script directory (script) in the acquired material "CRC Source Code & Sample Data" to the home directory.

```
$ cd ~/
$ cp -rf ~/controller/test/script .
```

2. If necessary, modify the following variables in each script appropriately for the environment.

Table 1 List of Scripts and Variables

File name	Target	Variable name	Setting value
run_controllers.sh	control plane	YAML_DIR K8S_SOFT_DIR	・absolute path of yaml/ ・absolute path of controller/
delete_all.sh		YAML_DIR K8S_SOFT_DIR	・absolute path of yaml/ ・absolute path of controller/
reset_ph3_fpga.sh	node with FPGA card	BIT_DIR	・Absolute path of the directory containing BIT files

*In the above reset_ph3_fpga.sh, Bitstream is written using the mcap tool, but the -x (PCI device ID specification) and -s (BDF specification) options in the mcap command are environment-dependent values, so it is necessary to change them according to the environment to be built.

The values of -x and -s for each environment (K8s node) can be checked with the lspci command. In the execution example of lspci below, the information of the number of FPGA cards is displayed, and xx: xx.x (e.g., 1f: 0.0) is the BDF, and 903f is the PCI device ID. The blue letters are the -s value, and the red letters are the -x value..

```
$ lspci |grep Xilinx
xx:xx.x Processing accelerators: Xilinx Corporation Device 903f
 . . .

$ vi ~/script/reset_ph3_fpga.sh
 . . .
BIT_DIR=$HOME/hardware-design/example-design/bitstream
cd $BIT_DIR
$ sudo mcap -x 903f -s xx:xx.x -p OpenKasugai-fpga-example-design-1.0.0-2.bit
 . . .
```

1.1.2 Editing DataFlow yaml

Target：K8s control plane

For each DataFlow yaml deployed in Section 1.3.1, change the network information (IP address and port number) used by Pod of each processing module in DataFlow as necessary.
If the IP address set for each server's 100G NIC is different from the IP address shown in Figure 1 in Section 1.1 in the separate document "OpenKasugai-Controller-Install Manual", change the IP address.
For instructions on how to make the changes, please refer to the sheet "2 (Supplement). DataFlow yaml settings" in the separate document "OpenKasugai-Controller-InstallManual_Attachment1".

Table 2 List of DataFlow yaml deployed in Section 1.3.1

DataFlow Types	yaml to be edited
1st FPGA F/R DF (df-test-1-1)	~/controller/test/sample-data/sample-data-demo/yaml/dataflows/test-1/df-test-1-1.yaml
2nd FPGA F/R DF (df-test-1-2)	~/controller/test/sample-data/sample-data-demo/yaml/dataflows/test-1/df-test-1-2.yaml

* When changing DataFlow name (metadata.name value) in the above sample data, the name must be 14 characters or less. Refer to the attached "OpenKasugai-Controller_Attachment1 (CR/CM Specification)" for details on the character limit.

1.2 Environment initialization

*When redeploying DataFlow and streaming video, be sure to perform "1.3.4 Video stream and reception stop" and "1.4 Stop Environment" before starting from the following steps.

1.2.1 Write FPGA Bitstream and auto collect & rerun CM creation tool

Target：K8s node with FPGA card

(1) Writing FPGA Bitstream

Execute the following script to write the Bitstream to the FPGA and reset it.

```
$ cd ~/script
$ ./reset_ph3_fpga.sh
```

* Writing the Bitstream takes several tens of seconds. If the writing is successful, "FPGA Configuration Done!!" will be displayed in the log.

When the FPGA is initialized, it is necessary to delete the used runtime directory/all Hugpages and directories, so perform the following steps:

```
$ sudo rm -rf /var/run/dpdk/*
$ sudo rm -rf /dev/hugepages/*
```

(2) Running the automatic collection & CM creation tool

Run the automatic collection & CM creation tool on the target K8s node to restore the ConfigMap and CustomResource associated with the FPGA to their initial state.

```
$ export
  PKG_CONFIG_PATH=${PKG_CONFIG_PATH}:$HOME/controller/src/submodules/fpga-
  software/lib/build/pkgconfig
$ cd ~/controller/src/tools/InfoCollector/
$ ln -s ../../fpgadb/test/bitstream_id-config-table.json bitstream_id-config-
  table.json *1
$ make all *2
```

*1. Skip "ln -s" command if you've already executed it in step3 of Section 8.6 of the separate "OpenKasugai -Controller-Install Manual".

*2. In case of K8s Node without NVIDIA GPU, since DCGM is not installed, the following error message is output in the InfoCollector log, but it can be ignored.

"INFO infocollect/infocollect.go:405 dcgm.Init() Error but Maybe there are NOT any GPU. {"error": "libdcgm.so not Found"}

1.2.2 CRC start up

Target : K8s control plane

(1) Running each CRC

Execute the following command to start each CRC and apply the function catalog yaml.

```
$ cd ~/script
$ ./run_controllers.sh
```

(2) Checking CRC startup and CRD installation.

Enter the following command to check that each CRC pod is running, all CRDs (20 types) are installed, and the FunctionChain of cpu-decode-filter-resize-high-infer-chain is Ready.

If the check result is different from the output shown below, identify the cause by checking the log of each Pod or the result of kubectl describe, and then execute "1.4 Environmental shutdown". After taking the corrective action, start again from "1.2 Environment initialization"

```
$ kubectl get pod
NAME                                        READY    STATUS       RESTARTS     AGE
crc-cpufunction-daemon-gh4gb                1/1      Running      0            15s
crc-cpufunction-daemon-xzf9s                1/1      Running      0            15s
crc-deviceinfo-daemon-qgv5w                 1/1      Running      0            18s
crc-deviceinfo-daemon-rwmqh                 1/1      Running      0            18s
crc-ethernetconnection-daemon-h67pj         1/1      Running      0            15s
crc-ethernetconnection-daemon-t2hnm         1/1      Running      0            15s
crc-fpgafunction-daemon-ts8lt               1/1      Running      0            15s
crc-fpgafunction-daemon-zxxrs               1/1      Running      0            15s
crc-gpufunction-daemon-mzkwk                1/1      Running      0            15s
crc-gpufunction-daemon-r596t                1/1      Running      0            15s
crc-pcieconnection-daemon-hvrg6             1/1      Running      0            15s
crc-pcieconnection-daemon-tg79x             1/1      Running      0            15s

$ kubectl  get pod -n whitebox-k8s-flowctrl-system
NAME                                                           READY    STATUS     RESTARTS    AGE
whitebox-k8s-flowctrl-controller-manager-74b469d866-c8q7f      2/2      Running    0           5m47s
$ kubectl get pod -n wbfunction-system
NAME                                                           READY    STATUS     RESTARTS    AGE
wbfunction-controller-manager-6df5bf76d6-wvj65                 2/2      Running    0           5m40s
$ kubectl get pod -n wbconnection-system
NAME                                                           READY    STATUS     RESTARTS    AGE
wbconnection-controller-manager-78b44dd58b-lbpnq               2/2      Running    0           6h44m
$ kubectl get crd |grep example.com
childbs.example.com                                  2024-11-11T00:18:56Z
computeresources.example.com                         2024-11-11T07:29:35Z
connectiontargets.example.com                        2024-11-11T07:29:35Z
connectiontypes.example.com                          2024-11-11T07:29:35Z
cpufunctions.example.com                             2024-11-08T08:29:56Z
dataflows.example.com                                2024-11-11T07:29:36Z
deviceinfoes.example.com                             2024-11-10T22:56:32Z
ethernetconnections.example.com                      2024-11-11T07:29:36Z
fpgafunctions.example.com                            2024-11-11T00:18:56Z
fpgareconfigurations.example.com                     2025-02-05T06:10:39Z
fpgas.example.com                                    2024-11-11T00:18:56Z
functionchains.example.com                           2024-11-11T07:29:36Z
functiontargets.example.com                          2024-11-11T07:29:36Z
functiontypes.example.com                            2024-11-11T07:29:36Z
gpufunctions.example.com                             2024-11-11T00:25:38Z
pcieconnections.example.com                          2024-11-11T07:29:36Z
schedulingdata.example.com                           2024-11-11T07:29:36Z
topologyinfos.example.com                            2024-11-11T07:29:36Z
wbconnections.example.com                            2024-11-11T07:29:36Z
wbfunctions.example.com                              2024-11-11T07:29:36Z
$ kubectl get functionchains.example.com -A
NAMESPACE        NAME                                              STATUS
chain-imgproc    cpu-decode-filter-resize-high-infer-chain    Ready
```

1.3 Video stream

1.3.1 DataFlow deployment

Target：K8s control plane

This section shows the flow of DataFlow deployment and video stream.

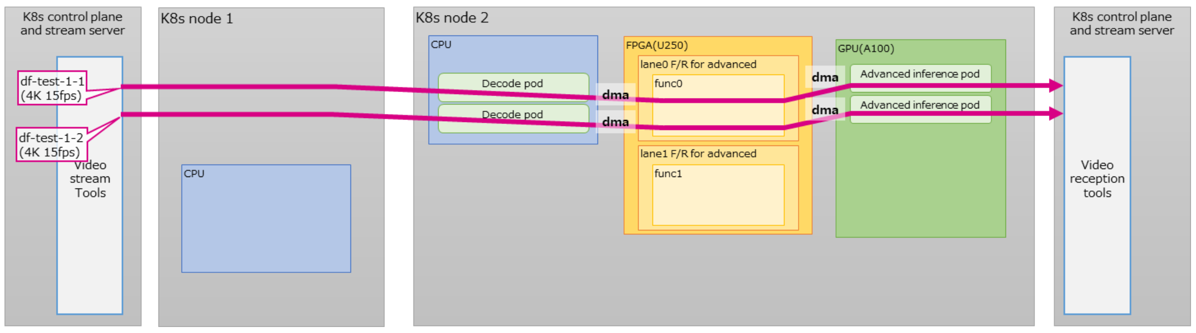

Figure 1 Deployment Image of Two DataFlow for the Demo in this Chapter

(1) Deploying DataFlow

The following describes how to deploy the DataFlow (DF) of the pattern A shown at the beginning of the chapter.

Apply the DF yaml of the sample data df-test-1-1 and df-test-1-2 one by one.

```
$ cd ~/controller/test/sample-data/sample-data-demo/yaml/dataflows/test-1/
$ kubectl apply -f df-test-1-1.yaml
$ kubectl get dataflow -A
(Wait about 30 seconds for STATUS to be Deployed)

$ kubectl apply -f df-test-1-2.yaml
```

(2) Checking deployment results

Check the results (STATUS) of the kubectl get command to ensure that each custom resource is generated normally and each Pod is deployed normally.

Check the following from the command execution result.

・Status of each DataFlow is Deployed
・The STATUS of each CPUFunction, GPUFunction, FPGAFunction, EthernetConnection, and PCIeConnection is Running.
・ FROMFUNC_STATUS and TOFUNC_STATUS of each EthernetConnection and PCIeConnection are OK.
・No resources have been created for each EthernetConnection ("No resources found" is displayed)

・The STATUS for each Pod is Running

```
$ kubectl get dataflows.example.com -A
$ kubectl get cpufunctions.example.com -A
$ kubectl get gpufunctions.example.com -A
$ kubectl get fpgafunctions.example.com -A
$ kubectl get pcieconnections.example.com -A
$ kubectl get ethernetconnections.example.com -A
$ kubectl get pod -n test01 -o wide
```

1.3.2 Start video reception

Target : K8s control plane

(1) Method for starting video reception

i. Deployment of video reception tool and creation of reception script (advanced inference result reception ×2).

(Only perform the first time, Step i is unnecessary from the second time onward.)

```
$ cd ~/controller/sample-functions/utils/rcv_video_tool/
$ kubectl create ns test
$ kubectl apply -f rcv_video_tool.yaml
$ kubectl get pod -n test    *Checking the pod name of video stream tool
$ kubectl exec -n test -it rcv-video-tool-xxx -- bash  *The xxx part is
changed according to the pod name confirmed above.
# vi start_test1.sh  *Paste the following and save it.
```

```
#!/bin/bash -x

for i in `seq -w 01 02`
do
   gst-launch-1.0 -e udpsrc buffer-size=21299100 mtu=8900 port=20${i} !
'application/x-rtp, media=(string)video, clock-rate=(int)90000, encoding-
name=(string)RAW, sampling=(string)BGR, depth=(string)8, width=(string)1280,
height=(string)1280, payload=(int)96' ! rtpvrawdepay ! queue ! videoconvert !
'video/x-raw, format=(string)I420' ! openh264enc ! 'video/x-h264, stream-
format=byte-stream, profile=(string)high' ! perf name=stream${i} !
h264parse ! qtmux ! filesink location=/tmp/output_st${i}.mp4 sync=false >
/tmp/rcv_video_tool_st${i}.log &
done
```

```
# chmod +x start_test1.sh
# exit
```

ii. In a new window, open a terminal for the receiving tool, climb into the Pod and start receiving video.

```
$ kubectl exec -n test -it rcv-video-tool-xxx -- bash  *Change pod name to
suit your environment
# ./start_test1.sh
```

(CRITICAL/WARNING logs are output only on the first time, but you can return to the prompt by pressing Enter. If the process startup can be confirmed in the next step, there is no problem.)

iii. Check the startup of the video reception tool inside the container.

* Verify that there are two processes starting with COMMAND as ¥_ gst-launch-1.0 running using ps auxwwf.

```
# ps auxwwf
USER         PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root         971  5.0  0.0   6048  2836 pts/49   Rs+  10:03   0:00 ps auxwwf
...
root         954  0.0  0.0 317188 13380 pts/48   Sl+  09:57   0:00  ¥_ gst-
launch-1.0 -e udpsrc buffer-size=21299100 mtu=8900 port=2008 !
application/x-rtp, media=(string)video, clock-rate=(int)90000, encoding-
name=(string)RAW, sampling=(string)BGR, depth=(string)8, width=(string)1280,
height=(string)1280, payload=(int)96 ! rtpvrawdepay ! queue ! videoconvert !
video/x-raw, format=(string)I420 ! openh264enc ! video/x-h264, stream-
format=byte-stream, profile=(string)high ! queue ! perf name=stream08 !
h264parse ! qtmux ! filesink location=/tmp/output_st08.mp4 sync=1
root         943  0.0  0.0 317188 13392 pts/47   Sl+  09:57   0:00  ¥_ gst-
launch-1.0 -e udpsrc buffer-size=21299100 mtu=8900 port=2007 !
application/x-rtp, media=(string)video, clock-rate=(int)90000, encoding-
name=(string)RAW, sampling=(string)BGR, depth=(string)8, width=(string)1280,
height=(string)1280, payload=(int)96 ! rtpvrawdepay ! queue ! videoconvert !
video/x-raw, format=(string)I420 ! openh264enc ! video/x-h264, stream-
format=byte-stream, profile=(string)high ! queue ! perf name=stream07 !
h264parse ! qtmux ! filesink location=/tmp/output_st07.mp4 sync=1
...
```

1.3.3 Start of video stream

Target：K8s control plane

(1) Method for starting video distribution

i. Deployment of video streaming tools and creation of streaming scripts (two 4K 15fps streams for advanced inference)
(Execute only the first time, Step i is unnecessary from the second time onward)

```
$ cd ~/controller/sample-functions/utils/send_video_tool/
$ kubectl apply -f send_video_tool.yaml
$ kubectl get pod -n test    *Checking the pod name of video stream tool
$ kubectl exec -n test -it send-video-tool-xxx -- bash  *Change pod name to
suit your environment
# vi start_test1.sh    *Paste and save the following
#!/bin/bash

sleep_time=$1

./start_gst_sender.sh /opt/video/input_4K_15fps.mp4  192.174.90.101  5004  1
${sleep_time:-3}

./start_gst_sender.sh /opt/video/input_4K_15fps.mp4  192.174.90.102  5004  1
${sleep_time:-3}



# chmod +x start_test1.sh
# exit
```

Table 3: Explanation of arguments for start_gst_sender.sh in the above script

nth	Description	About Values
1	Video file path	Specifies the video file to be delivered. Change the settings according to the movie file to be used. (The above example video was created in section 7.8 of the "OpenKasugai-Controller-InstallManual".) Use 4 K 15fps video files. The length of the video is preferably about 30 to 60 seconds.
2	Destination IP	Specify the destination IP address. Change it to match the IP address assigned to the CPU decoding described in the DataFlow yaml.
3	Destination Port	Specify the destination port number. Change the port number to match the reception port number set for the CPU decoding described in the DataFlow yaml.
4	Number of startup processes	Specify the number of delivery processes to generate (If you specify 2 or more, start the specified number of processes that distribute the same video file to the same destination IP. Destination port number is incremented each time the delivery process is started)

5	Streaming Delay Interval	Video streaming start delay [seconds] (interval for starting streaming processes when 2 or more is specified for the fourth argument)

ii. In a new window, you open a terminal for the distribution tool, climb into the Pod, and start streaming.

```
$ kubectl exec -n test -it send-video-tool-xxx -- bash  *Change pod name to suit your environment
# ./start_test1.sh
```

(A CRITICAL log may be output only the first time, but you can return to the prompt by pressing Enter.There is no problem if the process startup can be confirmed in the next step.)

iii. Confirm the startup of the video streaming tool inside the container.
*Verify that two gst-launch runs on ps aux.

```
# ps aux | grep gst-launch
root       2335  0.0  0.0 309636 12272 pts/3    Sl   11:17   0:00 gst-launch-1.0 filesrc location=/mnt/input_4K_15fps.mp4 ! qtdemux ! video/x-h264 ! h264parse ! rtph264pay config-interval=-1 seqnum-offset=1 ! udpsink host=192.174.91.81 port=5004 buffer-size=2048
root       2336  0.0  0.0 309636 12272 pts/3    Sl   11:17   0:00 gst-launch-1.0 filesrc location=/mnt/input_4K_15fps.mp4 ! qtdemux ! video/x-h264 ! h264parse ! rtph264pay config-interval=-1 seqnum-offset=1 ! udpsink host=10.38.119.67 port=5004 buffer-size=2048

. . .
```

(2) Video input/output check of DataFlow

By following this procedure, video input from the video distribution tool is processed by the deployed DataFlow, and the resulting video is delivered to the video receiving tool. The video file of the distributed video will stop streaming once the video time elapses, but you can stop the distribution at any time by following the steps in the next section.

You can check if the video is being received by monitoring the mp4 files under/tmp in the video receiving tool container.

```
$ kubectl exec -n test -it rcv-video-tool-xxx -- bash  *Change pod name to suit your environment
# cd /tmp/
# ls -l
(Check the increase in size of each mp4 file.)
```

1.3.4 Video stream and reception stop

Target：K8s control plane

1. Stop within the container of the video distribution tool started in Section 1.3.3 and confirm that it has stopped.

```
$ kubectl exec -n test -it send-video-tool-xxx -- bash  *Change pod name to suit
your environment
# ./stop_gst_sender.sh
# ps aux
    (Make sure that gst-launch is not running)
```

2. Stop within the container of the video reception tool as follows and confirm that it has stopped.

```
$ kubectl exec -n test -it rcv-video-tool-xxx -- bash  *Change pod name to suit
your environment
# pgrep gst-launch-1.0 | xargs kill -2 > /dev/null
# ps aux
    (Make sure that gst-launch is not running)
```

*The following error message may be output, but it can be ignored.
[OpenH264] this = 0x0x7fccc40058c0, Error:CWelsH264SVCEncoder::EncodeFrame(), cmInitParaError.

3. Copy each output video file to the host.

```
$ kubectl -n test cp  rcv-video-tool-xxx:/tmp/output_stXX.mp4 ./output_stXX.mp4
    --retries 10
 (XX:01~02)
```

*Execute the command with the --retries 10 option because the copy operation may fail.
*Even if the following message is output, it can be ignored if the video file is copied.
"tar: Removing leading `/' from member names"

4. The video output is a video in which the Bounding Box(BB) for the detected person is drawn.
*In the case of an DF that includes the configuration of "FPGA F/R (PCIe connection) to GPU advanced inference" deployed in Section 1.3.1, a blank image is shown for the first approximately one minute of the output video, and then a BB image is shown, which is a normal result. The blank image is the dummy input data when no data has yet come to GPU advanced inference.

1.4 Environmental shutdown

1.4.1 Delete DataFlow, Delete various CRs, Stop each CRC

Target: K8s control plane

Execute the following script to delete DataFlow, delete various CRs, and stop each CRC.

```
$ cd ~/script
$ ./delete_all.sh
```

After executing the above script, it is normal that some CRD and ConfigMap registered in the separate document "OpenKasugai-Controller-InstallManual" procedure remain as shown below, so there is no need to manually delete them.

```
$ kubectl get crd |grep example.com
childbs.example.com                              2024-11-11T00:18:56Z
cpufunctions.example.com                         2024-11-08T08:29:56Z
deviceinfoes.example.com                         2024-11-10T22:56:32Z
ethernetconnections.example.com                  2024-11-11T23:30:33Z
fpgafunctions.example.com                        2024-11-11T00:18:56Z
fpgas.example.com                                2024-11-11T00:18:56Z
gpufunctions.example.com                         2024-11-11T00:25:38Z
pcieconnections.example.com                      2024-11-11T23:29:56Z


$ kubectl get cm
NAME                                     DATA     AGE
Connectionkindmap                        1        6d11h
cpufunc-config-copy-branch               1        6d11h
cpufunc-config-decode                    1        6d11h
cpufunc-config-filter-resize-high-infer  1        6d11h
cpufunc-config-filter-resize-low-infer   1        6d11h
cpufunc-config-glue-fdma-to-tcp          1        6d11h
deployinfo                               1        22d
filter-resize-ch                         1        22d
fpgareconfigurations                     1        6d11h
fpgafunc-config-filter-resize-high-infer 1        6d11h
fpgafunc-config-filter-resize-low-infer  1        6d11h
function-unique-info                     1        6d11h
functionkindmap                          1        6d11h
gpufunc-config-high-infer                1        6d11h
gpufunc-config-low-infer                 1        6d11h
infrastructureinfo                       1        22d
```

*When deploying DataFlow and distributing video again, start from "1.2 Initializing the environment."

2. Demo procedure for basic function #2

OpenKasugai-Controller v1.1.0 adds the ability to recover various resources in the FPGA used by DataFlow to be deleted when removing DataFlow. With the addition of this function, the recovered resources in the FPGA can be reused except for some resources in the FPGA (resources in the Function).
As for the "channel" which is the resource inside the Function (the resource inside filter/resize circuit in Bitstream (child bs) for filter/resize), the state and setting information may remain depending on the logic of the Function, so it is not covered by clear. However, a plurality of channels exist and unused channels can be used. Therefore, by managing used channels and allocating unused channels during DataFlow deployment, DataFlow can be deployed without rewriting child bs. (For details, refer to the basic concept (8) in the separate document "Openkasugai-Controller"))
This chapter shows the steps to demonstrate DataFlow's delete feature. This demonstration shows the procedure for deploying a DataFlow (same configuration as DataFlow deployed in Chapter 1) using an FPGA to the initial system, deleting of said DataFlow, and then deploying a new DataFlow to the same FPGA.

2.1 Advance preparation

2.1.1 Placing test scripts

Target: K8s control plane, all K8s node

Implement Section 1.1.1. Skip if already performed.

2.1.2 Edit yaml of DataFlow

Target: K8s control plane

For each DataFlow yaml deployed in Section 2.3.1, change the network information (IP address and port number) used by Pod of each processing module in DataFlow as necessary.
If the IP address set for each server's 100G NIC is different from the IP address shown in Figure 1 in Section 1.1 in the separate document "OpenKasugai-Controller-InstallManual", change the IP address. For the method of change, refer to the sheet "2 (Supplement). DataFlow yaml Configuration" in the separate document "OpenKasugai-Controller-InstallManual_Attachment1".

Table 4: List of yaml in DataFlow to be deployed in Section 2.3.1

Types of DataFlow	yaml of the sample data to be edited
1. FPGA F/R's 1st DF (df-test-2-1)	~/controller/test/sample-data/sample-data-demo/yaml/dataflows/test-2/df-test-2-1.yaml
2. FPGA F/R's 2nd DF (df-test-2-2)	~/controller/test/sample-data/sample-data-demo/yaml/dataflows/test-2/df-test-2-2.yaml

*When changing DataFlow name (metadata.name value) in the above sample data, the name must be 14 characters or less. For details on the character limit, refer to the separate document "OpenKasugai-Controller_Attachment1 (CR/CM Specification)".

2.2 Environment initialization

Implement Section 1.2.

2.3 Video stream to the first DataFlow

2.3.1 DataFlow deployment

Target: K8s control plane

This section shows the flow of the first DataFlow deployment and video stream.

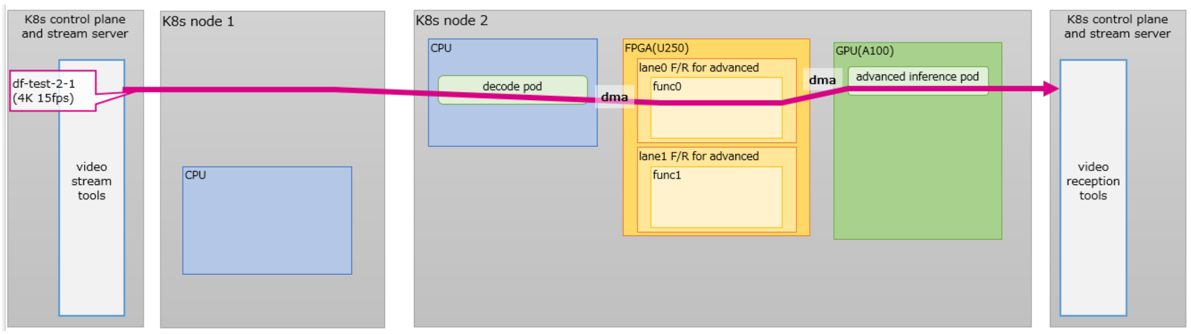

Figure 2 First DataFlow Deployment Image for the Demo in this Chapter

(1) DataFlow deployment

Apply yaml of sample data df-test-2-1.

```
$ cd ~/controller/test/sample-data/sample-data-demo/yaml/dataflows/test-2/
$ kubectl apply -f df-test-2-1.yaml
```

(2) Reviewing Deployment Results

Verify the successful generation of each custom resource and successful deployment of each Pod by performing (2) in Section 1.3.1

Also check which FPGA devices the deployed DataFlow is using.

Since the FPGA device is used in FPGAFunction CR in DataFlow, check the information of FPGAFunction CR of said DataFlow.

```
$ kubectl get fpgafunctions.example.com -A -o yaml
```

An example of the output is: Refer to status.dataFlowRef to see if it is FPGAFunction of said DataFlow or status.acceleratorStatuses.partionName.statsuses.acceleratorID to see which FPGA device is being used.

```
apiVersion: v1
items:
- apiVersion: example.com/v1
  kind: FPGAFunction
  metadata: (Omitted)
  spec: (Omitted)
  status:
    acceleratorStatuses:
    - partitionName: "0"
      statuses:
      - acceleratorID: /dev/xpcie_21320621V01L          Device file path for your FPGA
        status: OK
    childBitstreamName: 0100001c
    dataFlowRef:
      name: df-test-2-1                                  DataFlow Name
      namespace: test01                                  Namespace  Name
    (Omitted)
```

2.3.2 Video reception activation

Target: K8s control plane

(1) Method for starting video reception

i. Deployment of Video stream tool and creation of reception script (1 Advanced Inference Result Receiving)

(Perform only the first time, do not perform step (i) after the second time)

```
$ cd ~/controller/sample-functions/utils/rcv_video_tool/
$ kubectl create ns test
$ kubectl apply -f rcv_video_tool.yaml
$ kubectl get pod -n test        *Checking the pod name of Video stream tool
$ kubectl exec -n test -it rcv-video-tool-xxx -- bash    *The xxx part is changed according to the pod name confirmed above.
# vi start_test2.sh        *Paste and save the following
```

```
#!/bin/bash -x

for i in `seq -w 01 01`
do
    gst-launch-1.0 -e udpsrc buffer-size=21299100 mtu=8900 port=20${i} ! 'application/x-rtp, media=(string)video, clock-rate=(int)90000, encoding-name=(string)RAW, sampling=(string)BGR, depth=(string)8, width=(string)1280, height=(string)1280, payload=(int)96' ! rtpvrawdepay ! queue ! videoconvert ! 'video/x-raw, format=(string)I420' ! openh264enc ! 'video/x-h264, stream-format=byte-stream, profile=(string)high' ! perf name=stream${i} ! h264parse ! qtmux ! filesink location=/tmp/output_st${i}.mp4 sync=false > /tmp/rcv_video_tool_st${i}.log &
done
```

```
# chmod +x start_test2.sh
# exit
```

ii. Open a terminal for your reception tool in a new window, get into your Pod and start receiving video.

```
$ kubectl exec -n test -it rcv-video-tool-xxx -- bash    *Change pod name to suit your environment
# ./start_test2.sh
```

iii. Check the start of the Video reception tool in the container
  *ps auxwwf to see that 1 process with COMMAND starting at ¥_ gst-launch-1.0 is running

```
# ps auxwwf
USER        PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root        971  5.0  0.0   6048  2836 pts/49   Rs+  10:03   0:00 ps auxwwf
...
root        954  0.0  0.0 317188 13380 pts/48   Sl+  09:57   0:00  ¥_ gst-
launch-1.0 -e udpsrc buffer-size=21299100 mtu=8900 port=2008 ! application/x-
rtp, media=(string)video, clock-rate=(int)90000, encoding-name=(string)RAW,
sampling=(string)BGR, depth=(string)8, width=(string)1280,
height=(string)1280, payload=(int)96 ! rtpvrawdepay ! queue ! videoconvert !
video/x-raw, format=(string)I420 ! openh264enc ! video/x-h264, stream-
format=byte-stream, profile=(string)high ! queue ! perf name=stream08 !
h264parse ! qtmux ! filesink location=/tmp/output_st08.mp4 sync=1
...
```

2.3.3 Start of video stream

Target: K8s control plane

(1) Method for starting Video stream

i. Deploy Video stream tool and create a delivery script (1 x 4 K 15fps for advanced inference) (Perform this procedure only for the first time. This procedure (1) is not necessary for the second and subsequent times.)

```
$ cd ~/controller/sample-functions/utils/send_video_tool/
$ kubectl apply -f send_video_tool.yaml
$ kubectl get pod -n test    *Checking the pod name of Video stream tool
$ kubectl exec -n test -it send-video-tool-xxx -- bash  *The xxx part is changed
according to the pod name confirmed above.
# vi start_test2.sh    *Paste and save the following
```

```
#!/bin/bash

sleep_time=$1

./start_gst_sender.sh   /opt/video/input_4K_15fps.mp4   192.174.90.101   5004   1
${sleep_time:-3}
```

```
# chmod +x start_test2.sh
# exit
```

*See Table 3 (1) in section 1.3.3 , for arguments to start_gst_sender.sh.

ii. Open a terminal for stream tool in a new window, board Pod and start streaming.

```
$ kubectl exec -n test -it send-video-tool-xxx -- bash   *Change pod name to suit your environment
# ./start_test2.sh
```

(CRITICAL log may be output only for the first time, but press Enter to return to the prompt. If the process starts in the next step, there is no problem.)

iii Check the start of the Video stream tool in the container
*Verify that one gst-launch is running on ps aux.

```
# ps aux | grep gst-launch
root       2335  0.0  0.0 309636 12272 pts/3    Sl   11:17   0:00 gst-launch-1.0
filesrc location=/mnt/input_4K_15fps.mp4 ! qtdemux ! video/x-h264 ! h264parse !
rtph264pay  config-interval=-1  seqnum-offset=1  !  udpsink  host=192.174.91.81
port=5004 buffer-size=2048

 . . .
```

(2) Video input/output check of DataFlow
Implement (2) section 1.3.3

2.3.4 Video stream/reception stop

Target: K8s control plane

Implement section 1.3.4.

2.4 Delete the first DataFlow

2.4.1 Delete the DataFlow

Target: K8s control plane

Delete DataFlow deployed in Section 2.3.1.

```
$ kubectl delete dataflow df-test-2-1 -n test01
```

2.4.2 Confirmation whether the DataFlow was deleted

Target: K8s control plane

This section is executed when you'd like to confirm that various custom resource (CR) constituting DataFlow and Pod for processing module have been deleted. In the case of a deleted DataFlow (df-test-2-1), the resources to be deleted are as follows:
DataFlow, WBFunction and WBConnection (components of DataFlow),
CPUFunction, GPUFunction and FPGAFunction (WBFunction sub-resources),
Pod (a sub-resource of CPUFunction and GPUFunction)
PCIeConnection (a sub-resource of WBConnection)

To confirm the deletion of CR, execute kubectl_get.sh as shown below to confirm that the above-mentioned resources to be deleted have disappeared.

```
$ cd ~/script
$ ./kubectl_get.sh > ~/script/logs/deleted_kubectl_get.log
```

The result of kubectl_get.sh (In the above case it is output to ~/script/logs/deleted_kubectl_get.log.) is:. Verify that there are no CR or Pod listings for DataFlow you deleted (df-test-2-1).

```
(Omitted)
+ kubectl get dataflows.example.com -A
NAMESPACE   NAME         STATUS     FUNCTIONCHAIN
No resources found
+ kubectl get deviceinfoes.example.com -A
No resources found
+ kubectl get ethernetconnections.example.com -A
No resources found
+ kubectl get fpgafunctions.example.com -A
No resources found
+ kubectl get fpgas.example.com -A
No resources found
+ kubectl get gpufunctions.example.com -A
No resources found
+ kubectl get pcieconnections.example.com -A
No resources found
+ kubectl get wbconnections.example.com -A
No resources found
+ kubectl get wbfunctions.example.com -A
No resources found
+ kubectl get cpufunctions.example.com -A
No resources found
+ kubectl get pod -A -o wide
No resources found
+ kubectl get schedulingdata.example.com -A
No resources found
(Omitted)
```

2.5 Video stream to the second DataFlow

2.5.1 DataFlow deployment

Target: K8s control plane

This section shows the flow of the deployment and video stream of the second DataFlow.

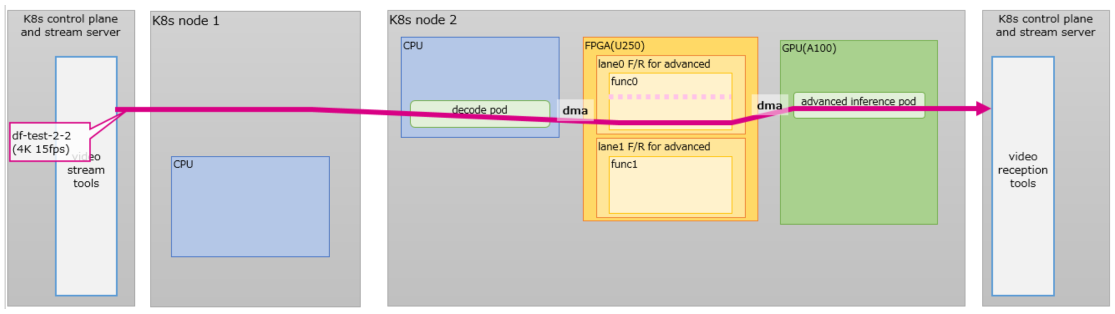

Figure 3 Second DataFlow Deployment Image for the Demo in this Chapter

(1) DataFlow deployment

Apply yaml of sample data df-test-2-2.

```
$ cd ~/controller/test/sample-data/sample-data-demo/yaml/dataflows/test-2/
$ kubectl apply -f df-test-2-2.yaml
```

(2) Reviewing Deployment Results

Verify the successful generation of each custom resource and successful deployment of each Pod by performing (2) in Section 1.3.1

Also check which FPGA devices the deployed DataFlow is using.
Since the FPGA device is used in FPGAFunction CR in DataFlow, check the information of FPGAFunction CR in DataFlow.

```
$ kubectl get fpgafunctions.example.com -A -o yaml
```

An example of the output is: Refer to status.dataFlowRef to see if it is FPGAFunction of DataFlow or status.acceleratorStatuses.partionName.statsuses.acceleratorID to see which FPGA device is being used.
Because we removed the first DataFlow in Section 2.4, the newly deployed DataFlow in this section should be using the same FPGA device that the first DataFlow was using.
Verify that the acceleratorID value is the same as the first DataFlow verified in (2) of section 2.3.1

```
apiVersion: v1
items:
- apiVersion: example.com/v1
  kind: FPGAFunction
 metadata: (Omitted)
 spec: (Omitted)
 status:
   acceleratorStatuses:
   - partitionName: "0"
     statuses:
     - acceleratorID: /dev/xpcie_21320621V01L          Device file path for your FPGA
       status: OK
   childBitstreamName: 0100001c
   dataFlowRef:
     name: df-test-2-2                                      DataFlow Name
     namespace: test01                                          Namespace
   (Omitted)
```

2.5.2 Video reception activation

Target: K8s control plane

Implement section 2.3.2.

2.5.3 Start of video stream

Target: K8s control plane

Implement Section 2.3.3.

2.5.4 Video stream/reception stop

Target: K8s control plane

Implement Section 1.3.4.

2.6 Delete the second DataFlow

2.6.1 Delete the DataFlow

Target: K8s control plane

Remove DataFlow deployed in Section 2.5.1.

```
$ kubectl delete dataflow df-test-2-2 -n test01
```

2.6.2 Confirmation whether the DataFlow was deleted

Target: K8s control plane

Perform this procedure to verify that no CR or Pod is listed for a deleted DataFlow (df-test-2-2). See Section 2.4.2 for details.

*Do not shut down the environment when performing the demonstration in Chapter 3. (If the demonstration in Chapter 3 is not performed, implement Section 1.4 to shut down the environment.)

3. Demo procedure for basic function #3

Using the FPGAReconfigurationTool, which was added from OpenKasugai-Controller v1.1.0, it is now possible to write Bitstream (child bs) without shutting down or initializing the environment. This chapter demonstrates the reset function of the FPGAReconfigurationTool (FPGA reset and Child Bitstream reset). In this demonstration, first, FPGA reset returns the FPGA used in Chapter 2 to child bs unwritten state (the same state as immediately after environment initialization). Then, DataFlow using the FPGA (same configuration as DataFlow deployed in Chapter 1) is repeatedly deployed and deleted to use up the resources (channels) inside the Function of the FPGA, and then Child Bitstream reset is executed to make the FPGA reusable (a new DataFlow is deployed).

3.1 Advance preparation

3.1.1 Deploying Test Scripts

Target: K8s control plane, all K8s node

Implement Section 1.1.1. Skip if already performed.

3.1.2 Editing yaml of DataFlow

Target: K8s control plane

For each DataFlow yaml deployed in Section 3.3, change the network information (IP address and port number) used by Pod of each processing module in DataFlow as necessary.

If the IP address set for each server's 100G NIC is different from the IP address shown in Figure 1 in Section 1.1 of the separate "OpenKasugai-Controller-Install Manual," change the IP address.

For the method of change, refer to the sheet "2 (Supplement)" in the separate document "OpenKasugai-Controller-InstallManual_Attachment1".

Table 3: List of yaml in DataFlow to be deployed in Sections 3.3 and 3.4

Types of DataFlow	yaml of the sample data to be edited
1. FPGA F/R's 1st DF (df-test-3-1)	~/controller/test/sample-data/sample-data-demo/yaml/dataflows/test-3/df-test-3-1.yaml
2. FPGA F/R's 2nd DF (df-test-3-2)	~/controller/test/sample-data/sample-data-demo/yaml/dataflows/test-3/df-test-3-2.yaml
3. FPGA F/R's 3rd DF (df-test-3-3)	~/controller/test/sample-data/sample-data-demo/yaml/dataflows/test-3/df-test-3-3.yaml
4. FPGA F/R's4th DF (df-test-3-4)	~/controller/test/sample-data/sample-data-demo/yaml/dataflows/test-3/df-test-3-4.yaml
5. FPGA F/R's5th DF (df-test-3-5)	~/controller/test/sample-data/sample-data-demo/yaml/dataflows/test-3/df-test-3-5.yaml
6. FPGA F/R's 6th DF (df-test-3-6)	~/controller/test/sample-data/sample-data-demo/yaml/dataflows/test-3/df-test-3-6.yaml
7. FPGA F/R's 7th DF (df-test-3-7)	~/controller/test/sample-data/sample-data-demo/yaml/dataflows/test-3/df-test-3-7.yaml
8. FPGA F/R's 8th DF (df-test-3-8)	~/controller/test/sample-data/sample-data-demo/yaml/dataflows/test-3/df-test-3-8.yaml
9. FPGA F/R's 9th DF (df-test-3-9)	~/controller/test/sample-data/sample-data-demo/yaml/dataflows/test-3/df-test-3-9.yaml
10. FPGA F/R's 10th DF (df-test-3-10)	~/controller/test/sample-data/sample-data-demo/yaml/dataflows/test-3/df-test-3-10.yaml

11. FPGA F/R's 11th DF (df-test-3-11)	~/controller/test/sample-data/sample-data-demo/yaml/dataflows/test-3/df-test-3-11.yaml
12. FPGA F/R's 12th DF (df-test-3-12)	~/controller/test/sample-data/sample-data-demo/yaml/dataflows/test-3/df-test-3-12.yaml
13. FPGA F/R's 13th DF (df-test-3-13)	~/controller/test/sample-data/sample-data-demo/yaml/dataflows/test-3/df-test-3-13.yaml
14. FPGA F/R's 14th DF (df-test-3-14)	~/controller/test/sample-data/sample-data-demo/yaml/dataflows/test-3/df-test-3-14.yaml
15. FPGA F/R's 15th DF (df-test-3-15)	~/controller/test/sample-data/sample-data-demo/yaml/dataflows/test-3/df-test-3-15.yaml
16. FPGA F/R's 16th DF (df-test-3-16)	~/controller/test/sample-data/sample-data-demo/yaml/dataflows/test-3/df-test-3-16.yaml
17. FPGA F/R's 17th DF (df-test-3-17)	~/controller/test/sample-data/sample-data-demo/yaml/dataflows/test-3/df-test-3-17.yaml

*When changing DataFlow name (metadata.name value) in the above sample data, the name must be 14 characters or less. Refer to the separate document "OpenKasugai-Controller_Attachment1 (CR/CM Specification)" for details on the character limit.

3.2 FPGA reset

Target: All K8s nodes equipped with FPGA cards on which DataFlow is deployed.

Confirm in advance that DataFlow is not deployed to the target FPGA (all deployed DataFlow have been deleted). Note that this is the state (DataFlow has all been deleted) when running as a continuation of the demo in Chapter 2.

Here, an example of performing FPGA reset on the FPGA (device file path is "/dev/xpcie_21320621V01L") installed in K8s node ("server2") shown in FIG. 1 in Section 1.1 in the separate document "OpenKasugai-Controller-InstallManual" is shown.

```
$ cd ~/controller/src/tools/FPGAReconfigurationTool/
$ ./FPGAReconfigurationTool server2 /dev/xpcie_21320621V01L -reset FPGA
```

*For more information on how to use FPGAReconfigurationTool, see/src/tools/FPGAReconfigurationTool/ReadMe.txt in the controller repository (v1.1.0) on github (OpenKasugai).

3.3 Use up resources (channels) inside FPGA functions

Target: K8s control plane

By repeatedly deploying and deleting DataFlow, resources (channels) inside the FPGA function are used up. This section describes the procedure for deploying a DataFlow that performs advanced inference in a hypothetical environment (The environment shown in Fig. 1 in Section 1.1 in the separate document "OpenKasugai-Controller-InstallManual" and the FPGA circuit described in Section 0.4 are used.).

In this case, 8 channels can be used in both lane0 and lane1, which are each deployment area (Lane). One channel is allocated to one DataFlow. Therefore, it is necessary to deploy 16 DataFlows to use up channels in one FPGA device. On the other hand, due to the capacity limit of each FPGA Lane, only two DataFlow can be deployed simultaneously.

Thus, to use up channels in one FPGA, it is necessary to repeat the deployment and deletion of 2 DataFlows 8 times.

The flow of DataFlow deployment and deletion is shown below.

i. Deployment, stream, and deletion of the 1st and the 2nd DataFlows

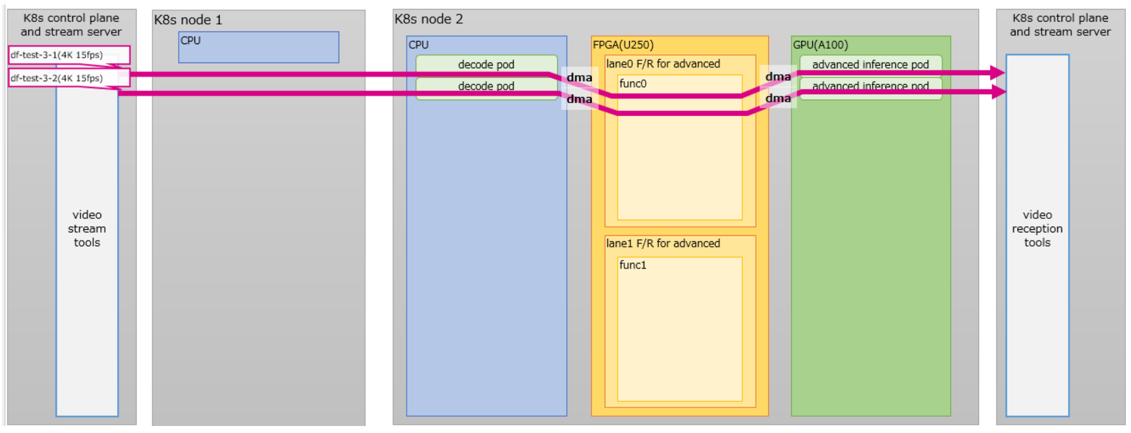

Figure 4 Image of the 1st and the 2nd DataFlow deployment for the Demo in this chapter

- DataFlow deployment

  Apply one sample df-test-3-1 and one sample df-test-3-2 of DF's yaml.

```
$ cd ~/controller/test/sample-data/sample-data-demo/yaml/dataflows/test-3/
$ kubectl apply -f df-test-3-1.yaml
$ kubectl get dataflow -A
(Wait about 30 seconds for STATUS to be Deployed)

$ kubectl apply -f df-test-3-2.yaml
```

  To check the deployment results, perform (2) in Section 1.3.1

- video stream

  Perform Section 1.3.2 to activate video reception. Then, Section 1.3.3 is executed to start video stream. After the video input/output is confirmed, Section 1.3.4 is executed to stop video reception and video stream.

- Delete DataFlow

  Delete the deployed DataFlow.

```
$ kubectl delete dataflow df-test-3-1 -n test01
$ kubectl delete dataflow df-test-3-2 -n test01
```

When checking the deletion results, refer to Section 2.4.2 to confirm that there is no CR or Pod related to the deleted DataFlow.

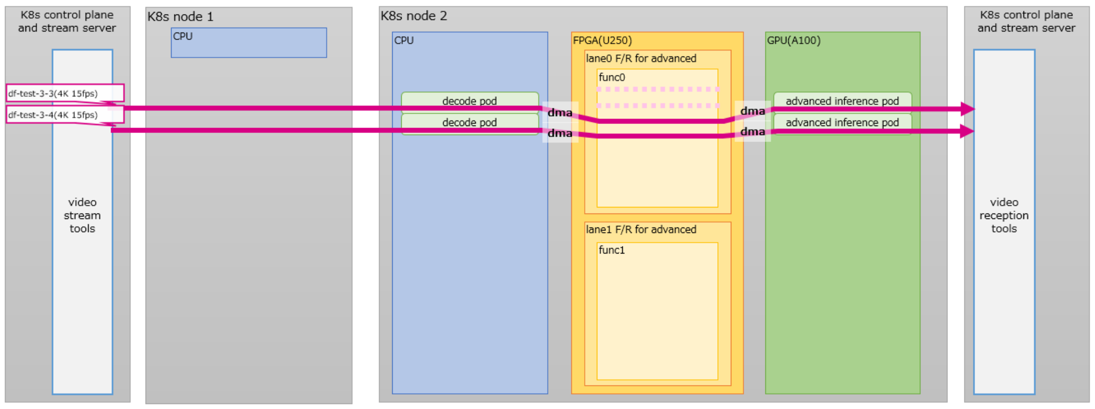

Figure 5: Image of the 3rd and the 4th DataFlow deployment for the demonstration in this chapter

ii. Deployment, stream, and deletion of the 3rd and the 4th DataFlow

・DataFlow deployment

Apply one sample df-test-3-3 and one sample df-test-3-4 of DF's yaml.

```
$ kubectl apply -f df-test-3-3.yaml
(Wait about 20 seconds for STATUS to be Deployed)
$ kubectl apply -f df-test-3-4.yaml
```

To check the deployment results, perform (2) in Section 1.3.1

・video stream

Perform Section 1.3.2 to activate video reception. Then, Section 1.3.3 is executed to start video stream. After the video input/output is confirmed, Section 1.3.4 is executed to stop video reception and video stream.

・Delete DataFlow

Delete the deployed DataFlow.

```
$ kubectl delete dataflow df-test-3-3 -n test01
$ kubectl delete dataflow df-test-3-4 -n test01
```

When checking the deletion results, refer to Section 2.4.2 to confirm that there is no CR or Pod related to the deleted DataFlow.

iii. Deployment, stream, and deletion of DataFlow (5th, 6th)

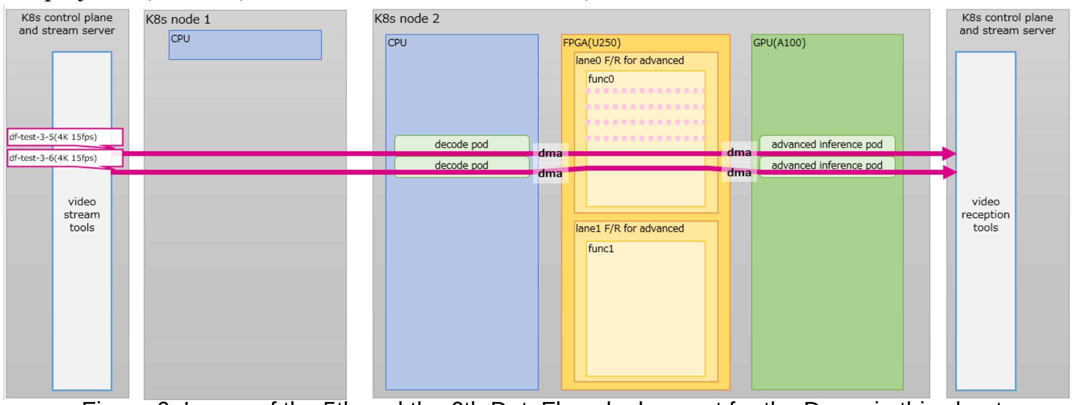

Figure 6: Image of the 5th and the 6th DataFlow deployment for the Demo in this chapter

・DataFlow deployment

Apply one sample df-test-3-5 and one sample df-test-3-6 of DF's yaml.

```
$ kubectl apply -f df-test-3-5.yaml
(Wait about 20 seconds for STATUS to be Deployed)
$ kubectl apply -f df-test-3-6.yaml
```

To check the deployment results, perform (2) in Section 1.3.1

・video stream

Perform Section 1.3.2 to activate video reception. Then, Section 1.3.3 is executed to start video stream. After the video input/output is confirmed, Section 1.3.4 is executed to stop video reception and video stream.

・Delete DataFlow

Delete the deployed DataFlow.

```
$ kubectl delete dataflow df-test-3-5 -n test01
$ kubectl delete dataflow df-test-3-6 -n test01
```

When checking the deletion results, refer to Section 2.4.2 to confirm that there is no CR or Pod related to the deleted DataFlow.

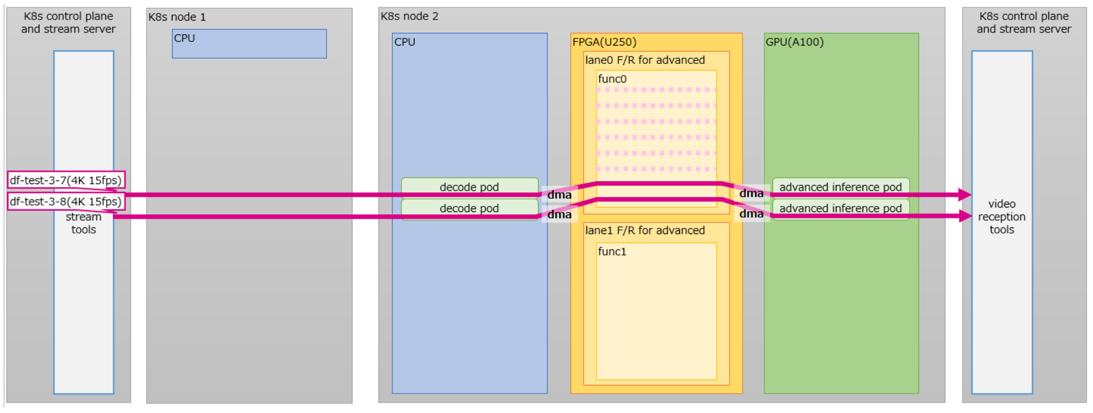

Figure 7: Image of the 7th and the 8th DataFlow deployment for the demonstration in this chapter

iv. Deployment, stream, and deletion of the 7th and the 8th DataFlow

・DataFlow deployment

Apply one sample df-test-3-7 and one sample df-test-3-8 of DF's yaml.

```
$ kubectl apply -f df-test-3-7.yaml
(Wait about 20 seconds for STATUS to be Deployed)
$ kubectl apply -f df-test-3-8.yaml
```

To check the deployment results, perform (2) in Section 1.3.1

・video stream

Perform Section 1.3.2 to activate video reception. Then, Section 1.3.3 is executed to start video stream. After the video input/output is confirmed, Section 1.3.4 is executed to stop video reception and video stream.

・Delete DataFlow

Delete the deployed DataFlow.

```
$ kubectl delete dataflow df-test-3-7 -n test01
$ kubectl delete dataflow df-test-3-8 -n test01
```

When checking the deletion results, refer to Section 2.4.2 to confirm that there is no CR or Pod related to the deleted DataFlow.

v. Deployment, stream, and deletion of the 9th and the 10th DataFlow

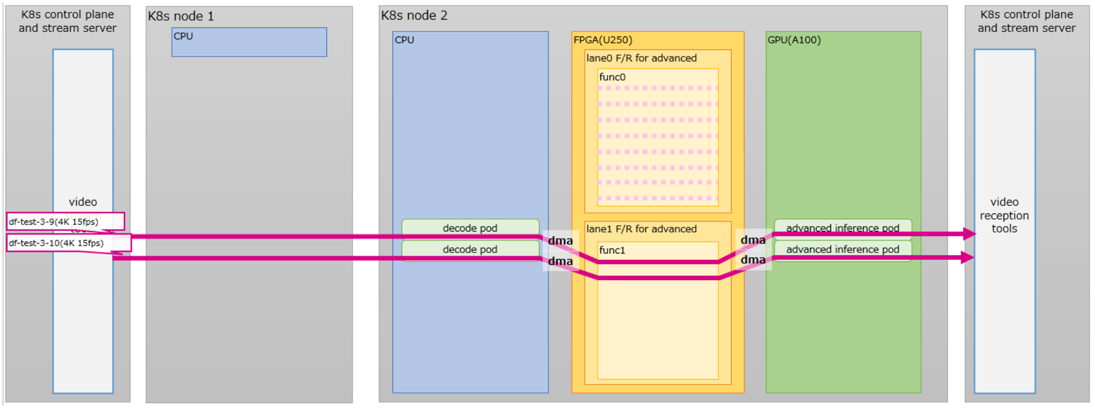

Figure 8: Image of the 9th and the10th DataFlow deployment for the Demo in this chapter

· DataFlow deployment

Apply one sample df-test-3-9 and one sample df-test-3-10 of DF's yaml.

```
$ kubectl apply -f df-test-3-9.yaml
(Wait about 30 seconds for STATUS to be Deployed)
$ kubectl apply -f df-test-3-10.yaml
```

To check the deployment results, perform (2) in Section 1.3.1

・ video stream

Perform Section 1.3.2 to activate video reception. Then, Section 1.3.3 is executed to start video stream. After the video input/output is confirmed, Section 1.3.4 is executed to stop video reception and video stream.

・ Delete DataFlow

Delete the deployed DataFlow.

```
$ kubectl delete dataflow df-test-3-9 -n test01
$ kubectl delete dataflow df-test-3-10 -n test01
```

When checking the deletion results, refer to Section 2.4.2 to confirm that there is no CR or Pod related to the deleted DataFlow.

vi. Deployment, stream, and deletion of the 11th and the 12th DataFlow

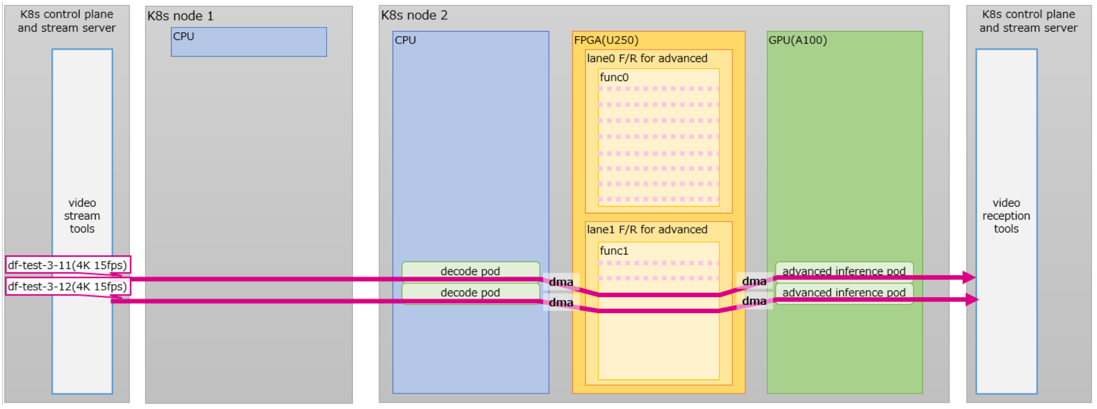

Figure 9: Image of the 11th and the12th DataFlow deployment for the demonstration in this chapter

・DataFlow deployment

Apply one sample df-test-3-11 and one sample df-test-3-12 of DF's yaml.

```
$ kubectl apply -f df-test-3-11.yaml
(Wait about 20 seconds for STATUS to be Deployed)
$ kubectl apply -f df-test-3-12.yaml
```

To check the deployment results, perform (2) in Section 1.3.1

・video stream

Perform Section 1.3.2 to activate video reception. Then, Section 1.3.3 is executed to start video stream. After the video input/output is confirmed, Section 1.3.4 is executed to stop video reception and video stream.

・Delete DataFlow

Delete the deployed DataFlow.

```
$ kubectl delete dataflow df-test-3-11 -n test01
$ kubectl delete dataflow df-test-3-12 -n test01
```

When checking the deletion results, refer to Section 2.4.2 to confirm that there is no CR or Pod related to the deleted DataFlow.

vii. Deployment, stream, and deletion of the 13th and the 14th DataFlow

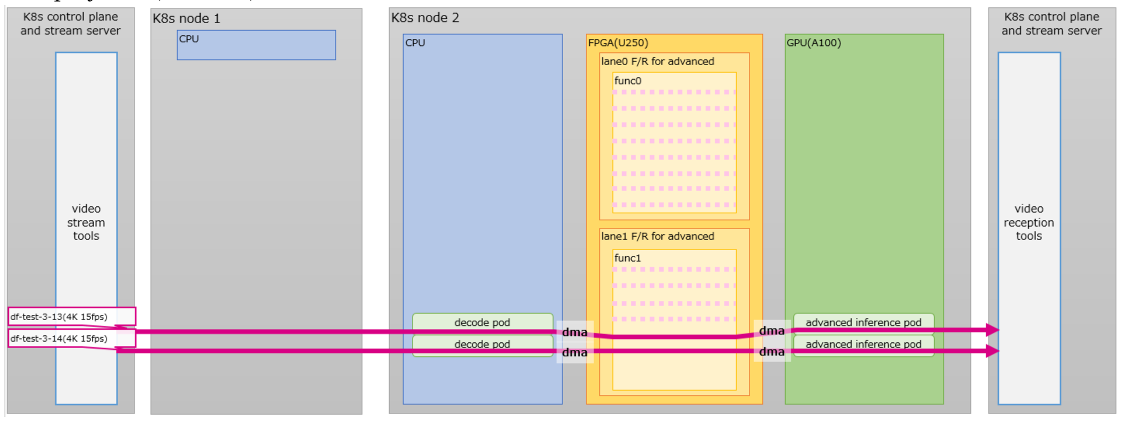

Figure 10: Image of the 13th and the14th DataFlow deployment for the demonstration in this chapter

・DataFlow deployment

Apply one sample df-test-3-13 and one sample df-test-3-14 of DF's yaml.

```
$ kubectl apply -f df-test-3-13.yaml
(Wait about 20 seconds for STATUS to be Deployed)
$ kubectl apply -f df-test-3-14.yaml
```

To check the deployment results, perform (2) in Section 1.3.1

・video stream

Perform Section 1.3.2 to activate video reception. Then, Section 1.3.3 is executed to start video stream. After the video input/output is confirmed, Section 1.3.4 is executed to stop video reception and video stream.

・Delete DataFlow

Delete the deployed DataFlow.

```
$ kubectl delete dataflow df-test-3-13 -n test01
$ kubectl delete dataflow df-test-3-14 -n test01
```

When checking the deletion results, refer to Section 2.4.2 to confirm that there is no CR or Pod related to the deleted DataFlow.

viii. Deployment, stream, and deletion of the 15th and the 16th DataFlow

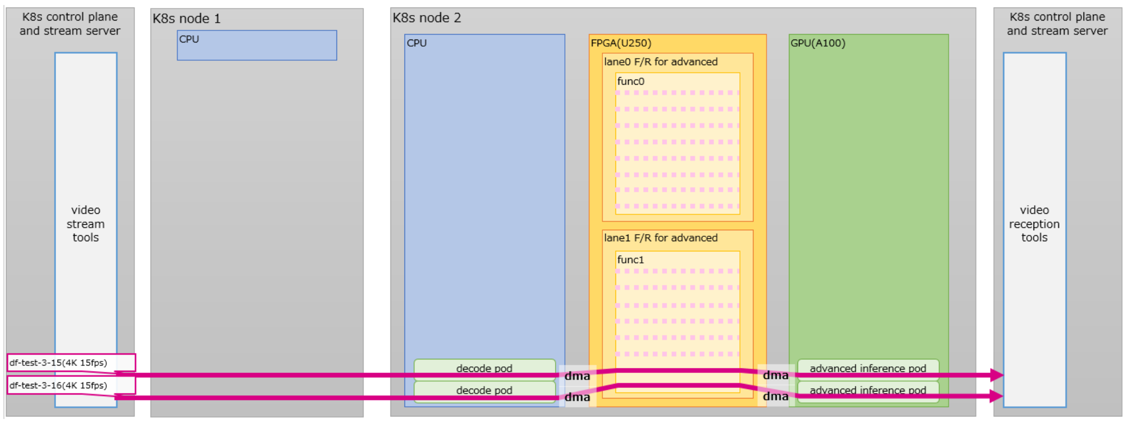

Figure 11: Image of the 15th and the16th DataFlow deployment for the demonstration in this chapter

・DataFlow deployment

Apply one sample df-test-3-15 and one sample df-test-3-16 of DF's yaml.

```
$ kubectl apply -f df-test-3-15.yaml
(Wait about 20 seconds for STATUS to be Deployed)
$ kubectl apply -f df-test-3-16.yaml
```

To check the deployment results, perform (2) in Section 1.3.1

・video stream

Perform Section 1.3.2 to activate video reception. Then, Section 1.3.3 is executed to start video stream. After the video input/output is confirmed, Section 1.3.4 is executed to stop video reception and video stream.

・Delete DataFlow

Delete the deployed DataFlow.

```
$ kubectl delete dataflow df-test-3-15 -n test01
$ kubectl delete dataflow df-test-3-16 -n test01
```

When checking the deletion results, refer to Section 2.4.2 to confirm that there is no CR or Pod related to the deleted DataFlow.

3.4 Check DataFlow Deployment (Failed) and Resource Usage

We demand the deployment of a new DataFlow after the implementation of Section 3.3. However, deployment fails as follows:

```
$ kubectl apply -f df-test-3-17.yaml
(Wait for about 5 seconds.)
$ kubectl get dataflows.example.com -A
NAMESPACE    NAME         STATUS                  FUNCTIONCHAIN
test01       df-test-3-17 Scheduling in progress  cpu-decode-filter-resize-low-infer-chain
```

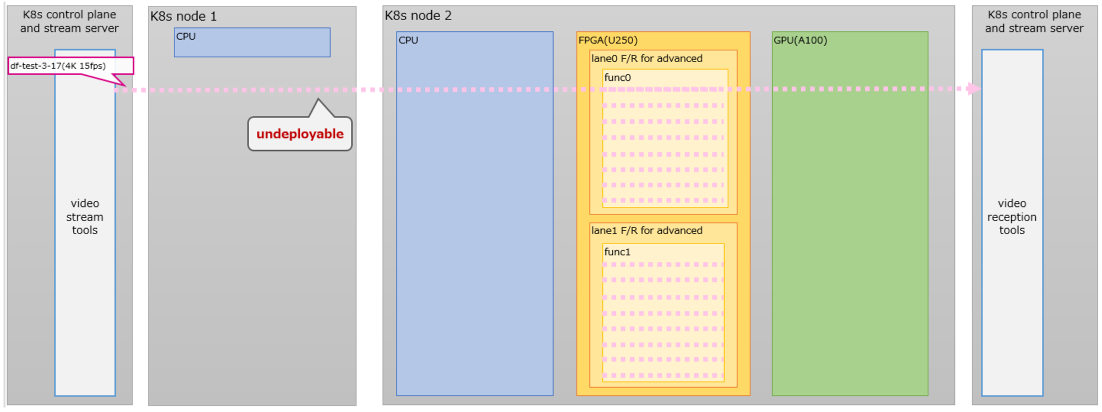

Figure 12: Image of the 17th DataFlow deployments after deployments and deletions the 16 DataFlows

DataFlow fails to deploy because all FPGA channels have been used up and the resources of the FPGA are depleted. To confirm that all channels of the FPGA have been used up, it can be confirmed from the ChildBs CR corresponding to the FPGA as follows.
The usage of each channel is recorded in the modules.deploySpec.intraResourceMgmtMap for each region of the ChildBs CR. Map key indicates the channel ID, and value indicates the state of the channel.
In the case of a channel assigned to a deleted DataFlow, "available" in value is "false" and "functionCRName" is empty. Therefore, after the implementation of Section 3.3, "available" will be "false" and "functionCRName" will have no value for all channels.
*For more information on the ChildBs CR, see the separate document "Openkasugai-Controller_Attachment1 (CR/CM Specification)".

Here is an example of how to check ChildBs:

```
$ kubectl get childbs.example.com -A -o yaml
```

An example of the output is following. Make sure that value of “available” field is “false” for all channels of lane0 and lane1.

```
apiVersion: v1
items:
- apiVersion: example.com/V1
  kind: ChildBs
  metadata: (Omitted)
  spec: (Omitted)
  status:
    regions:
    - maxCapacity: 40                                                  Record of the state of lane0
      maxFunctions: 1
      modules:
         (Omitted)
        functions: (Omitted)
        - deploySpec: (Omitted)
          id:0
          intraResourceMgmtMap:
           “0”:                   Information about channel ID0 that was assigned to the deleted dataflow
             available: false                                                    available is "false"
           “1”:                   Information about channel ID1 that was assigned to the deleted dataflow
             available: false                                                    available is "false"
           (Omitted)
           “7”:                   Information about channel ID7 that was assigned to the deleted dataflow
             available: false                                                    available is "false"
        (Omitted)
      name: lane0
    - maxCapacity: 40                                                  Record of the state of lane1
      maxFunctions: 1
      modules:
         (Omitted)
        functions: (Omitted)
        - deploySpec: (Omitted)
          id:0
          intraResourceMgmtMap:
           “0”:                   Information about channel ID0 that was assigned to the deleted dataflow
             available: false                                                    available is "false"
           “1”:                   Information about channel ID1 that was assigned to the deleted dataflow
             available: false                                                    available is "false"
           (Omitted)
           “7”:                   Information about channel ID7 that was assigned to the deleted dataflow
             available: false                                                    available is "false"
        (Omitted)
      name: lane1
    state: Ready
    status: Ready
```

3.5 Child Bitstream reset

Target: All K8s nodes equipped with FPGA cards on which DataFlow is deployed.

Child Bitstream reset is performed on the FPGA that has used up the resources, and the state immediately after child bs write is restored.
Here, an example of performing FPGA reset on the FPGA (device file path is "/dev/xpcie_21320621V01L") installed in K8s node ("server2") shown in FIG. 1 in Section 1.1 in the separate document "OpenKasugai-Controller-InstallManual" is shown.

```
$ cd ~/controller/src/tools/FPGAReconfigurationTool/
$ ./FPGAReconfigurationTool server2 /dev/xpcie_21320621V01L -reset ChildBs
```

*For more information on how to use FPGAReconfigurationTool, see/src/tools/FPGAReconfigurationTool/ReadMe.txt in the controller repository (v1.1.0) on github (OpenKasugai).

If Child Bitstream reset is successful, all the channels in the FPGA become available, so even DataFlow (df-test-3-17) that failed to be deployed in Section 3.4 is rescheduled and deployed successfully.

```
$ kubectl get dataflow df-test-3-17 -n test01
NAMESPACE   NAME           STATUS     FUNCTIONCHAIN
test01      df-test-3-17   Deployed   cpu-decode-filter-resize-high-infer-chain
```

(This assumes that df-test-3-17 was deployed in Section 3.4 and not removed. If DataFlow was deleted prior to Child Bitstream reset, re-deploy again.)

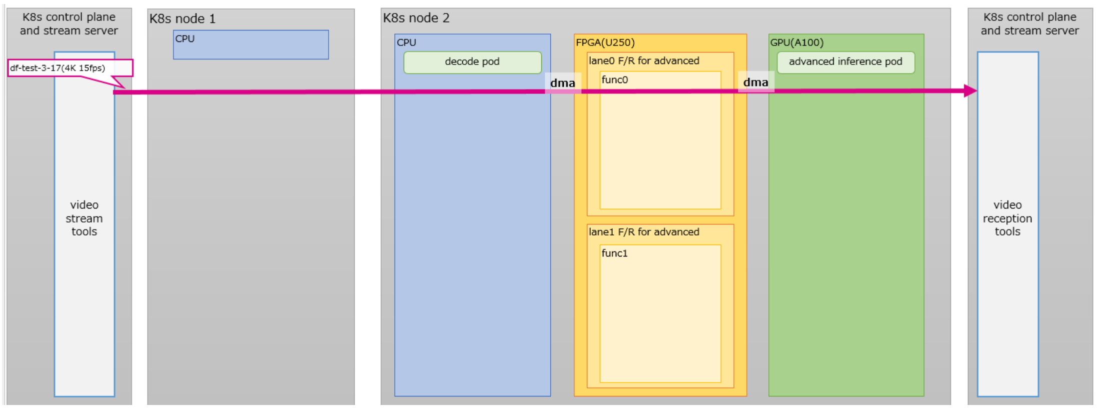

Figure 5: Image of DataFlow deployment after Child Bitstream reset implementation

The status of the ChildBs CR is also updated.

```
$ kubectl get childbs.example.com -A -o yaml
```

An example of the output is following. Make sure that value of “available” field is “true” for all channels of lane0 and lane1 except those assigned to df-test-3-17.

```
apiVersion: v1
items:
- apiVersion: example.com/V1
  kind: ChildBs
  metadata: (Omitted)
  spec: (Omitted)
  status:
    regions:
    - maxCapacity: 40                                  Record of the state of lane0
      maxFunctions: 1
      modules:
         (Omitted)
        functions: (Omitted)
        - deploySpec: (Omitted)
          id:0
          intraResourceMgmtMap:
           “0”:              Information about channel ID0 (assigned to df-test-3 -17)
             available: false                                     available is "false",
             functionCRName: df-test-3-17            functionCRName is DataFlow name
             rx: (Omitted)
             tx: (Omitted)
           “1”:                                Information about the channel with ID1
             available: true                                        available is "true"
           (Omitted)
           “7”:                                  Information about channels with ID7
             available: true                                        available is "true"
        (Omitted)
      name: lane0
    - maxCapacity: 40
      maxFunctions: 1
      modules:                                         record of the state of lane1
         (Omitted)
        functions: (Omitted)
        - deploySpec: (Omitted)
          id:0
          intraResourceMgmtMap:
           “0”:                                        Information about channel ID0
             available: false                                       available is "true"
           (Omitted)
           “7”:                                               Information about ID7
             available: false                                       available is "true"
        (Omitted)
      name: lane1
    state: Ready
    status: Ready
```

3.6 Environmental shutdown

Target: K8s control plane

Section 1.4 is implemented.

4. Demo procedure for basic function #4

This chapter shows the procedure for the demonstration of the FPGAReconfiguration Tool's child bs write function (writing child bs to an FPGA in a child bs unwritten state). In this demo, we perform FPGA Bitstream write on the initial system and then deploy an FPGA-powered DataFlow (the same configuration as DataFlow deployed in Chapter 1.

4.1 Advance preparation

4.1.1 Deploying test scripts

Target: K8s control plane, all K8s node

Implement Section 1.1.1. Skip if already performed.

4.1.2 Editing yaml in DataFlow

Target: K8s control plane

For each DataFlow yaml deployed in Section 4.4.1, change the network information (IP address and port number) used by Pod of each processing module in DataFlow as necessary.
If the IP address set for each server's 100G NIC is different from the IP address shown in Figure 1 in Section 1.1 in the separate document "OpenKasugai-Controller-InstallManual", change the IP address.
For the method of change, refer to the sheet "2 (Supplement)" in the separate document "OpenKasugai-Controller-InstallManual_Attachment1".

Table 4: List of yaml in DataFlow to be deployed in Section 4.4.1

Types of DataFlow	yaml of the sample data to be edited
1. FPGA F/R's 1st DF (df-test-4-1)	~/controller/test/sample-data/sample-data-demo/yaml/dataflows/test-4/df-test-4-1.yaml

4.2 environment initialization

Implement Section 1.2.

4.3 FPGA Bitstream write

Target: K8s node with an FPGA card to which a manual child bs write is written

In the case of using the FPGAReconfigurationTool, it is also possible to set different parameters for each Lane for a processing module (such as filter/resize) executed in a plurality of deployment areas (Lane) in the FPGA. (If you want to automatically write child bs at DataFlow deployment, you can only set the same parameters on all Lane).

Here, as an example, the FPGA installed in K8s node shown in FIG. 1 in Section 1.1 in the separate document "OpenKasugai-Controller-InstallManual" (the device file path is "/ dev/ xpcie_21320621V01L") is set as the write destination, and different parameters are set for each Lane. Specifically, of the two Lane (lane0, lane1), the parameters for filter/resize for advanced inference are set in lane0, and the parameters for filter/resize for lightweight inference are set in lane1.

```
$ cd ~/controller/src/tools/FPGAReconfigurationTool/
$ ./FPGAReconfigurationTool server2 /dev/xpcie_21320621V01L -l0 fpgafunc-config-filter-resize-high-infer -
l1 fpgafunc-config-filter-resize-low-infer
```

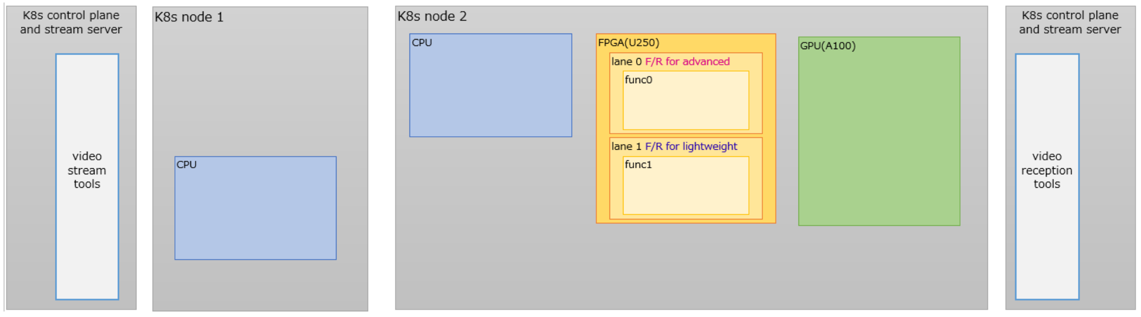

Figure 6: FPGA Bitstream write with FPGAReconfigurationTool (different parameters for each Lane)

4.4 video stream

4.4.1 DataFlow deployment

Target: K8s control plane

This section shows the flow of DataFlow deployment and video stream.

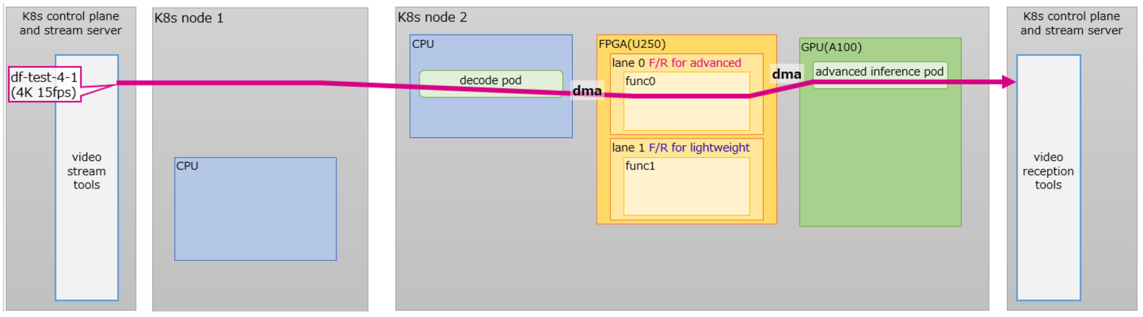

Figure 7: Image of the DataFlow deployment for the demonstration in this chapter

(1) DataFlow deployment

Apply yaml of sample data df-test-4-1.

```
$ cd ~/controller/test/sample-data/sample-data-demo/yaml/dataflows/test-4/
$ kubectl apply -f df-test-4-1.yaml
```

(2) Reviewing Deployment Results

Implement (2) Section 1.3.1.

4.4.2 video reception activation

Target: K8s control plane

Implement Section 2.3.2.

4.4.3 Start of video stream

Target: K8s control plane

Implement Section 2.3.3..

4.4.4 Video stream/reception stop

Target: K8s control plane

Implement Section 1.3.4.

4.4.5 Delete DataFlow

Target: K8s control plane

Delete DataFlow deployed in Section 4.4.1.

```
$ kubectl delete dataflow df-test-4-1 -n test01
```

When checking the deletion results, refer to Section 2.4.2 to confirm that there is no CR or Pod related to the deleted DataFlow.

4.5 environmental shutdown

Target: K8s control plane

Section 1.4 is implemented.

5. [Appendix] Extension demonstration procedure

In this chapter, as a demonstration procedure of the OpenKasugai controller extension function, the procedure from the deployment of copy branch DataFlow and Glue DataFlow to the confirmation of data communication is shown.

1. Deploy one DF that branches into multiple functions in the middle using copy branch
   - df-test-ext-1-1: CPU Decode —(Ethernet connection)→ CPU F/R —(Ethernet connection)→ CPU Copy Branch —(Ethernet connection)→ GPU Advanced Inference ×2

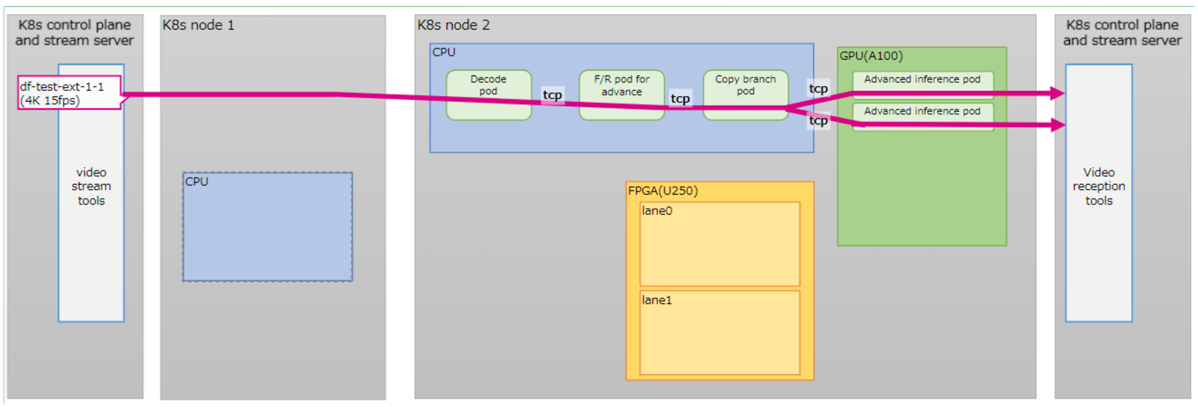

Figure 16　DF deployment image branching into multiple functions

2. Use connection type conversion processing (Glue) to deploy a single DF that connects functions with different connection types.
   - df-test-ext-2-1: CPU Decode - (PCIe connection) → FPGA F/R - (PCIe connection) → CPU Glue -(Ethernet connection) → GPU advanced inference

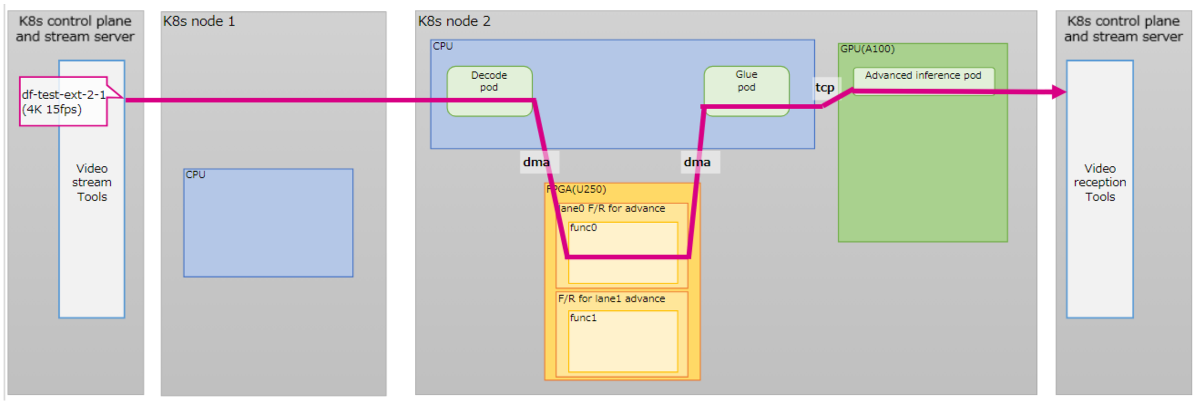

Figure 17　DF deployment image connecting functions of different connection types

5.1 Advance preparation

5.1.1 Deploying test scripts

Targe: K8s control plane, all K8s nodes

Perform step 1.1.1. If already performed, skip this step.

5.1.2 Editing DataFlow yaml

Target : K8s control plane

For each DataFlow yaml deployed in Section 5.3.1, change the network information (IP address and port number) used by Pod of each processing module in DataFlow as necessary.
If the IP address set for each server's 100G NIC is different from the IP address shown in Figure 1 in Section 1.1 in the separate document "OpenKasugai-Controller-Install Manual" change the IP address.
For the method of change, refer to the sheet "2 (Supplement) DataFlow yaml settings" in the separate document "OpenKasugai-Controller-InstallManual_Attachment1".

Table 7: List of DataFlow yaml to deploy in Section 5.3.1

DataFlow Types	yaml to edit
3. Copy Branch DF (df-test-ext-1-1)	~/controller/test/sample-data/sample-data-demo/yaml/dataflows/test-ext-1/df-test-ext-1-1.yaml
4. Glue DF (df-test-ext-2-1)	~/controller/test/sample-data/sample-data-demo/yaml/dataflows/test-ext-2/df-test-ext-2-1.yaml

5.2 Environment initialization

Execute section 1.2.

5.3 Video stream

5.3.1 DataFlow deployment

Target：K8s control plane

(1) Deploying DataFlow

The following describes how to deploy DataFlow as shown at the beginning of Chapter 2.

i. For copy branch DF

Apply the DataFlow yaml in the sample data df-test-ext-1-1.

```
$ cd ~/controller/test/sample-data/sample-data-demo/yaml/dataflows/test-ext-1/
$ kubectl apply -f df-test-ext-1-1.yaml
$ kubectl get dataflow -A
(Wait about 20 seconds until it becomes Deployed)
```

ii. For DF with Glue conversion

Apply the DataFlow yaml in the sample data df-test-ext-2-1.

```
$ cd ~/controller/test/sample-data/sample-data-demo/yaml/dataflows/test-ext-2/
$ kubectl apply -f df-test-ext-2-1.yaml
$ kubectl get dataflow -A
(Wait about 20 seconds until it becomes Deployed)
```

(2) Checking deployment results

Check the results (STATUS) of the kubectl get command to ensure that each custom resource is generated normally and each Pod is deployed normally.

i. In the case of copy branch DF:

Check the following from the command execution result:

・Each DataFlow has a STATUS of Deployed
・Status is Running for each CPUFunction, GPUFunction, and EthernetConnection
・FROMFUNC_STATUS and TOFUNC_STATUS are OK for each EthernetConnection
・The STATUS for each Pod is Running

```
$ kubectl get dataflows.example.com -A
$ kubectl get cpufunctions.example.com -A
$ kubectl get gpufunctions.example.com -A
$ kubectl get ethernetconnections.example.com -A
$ kubectl get pod -n test01 -o wide
```

ii. In the case of Glue conversion DF

Check the following from the command execution result:

・Each DataFlow has a STATUS of Deployed
・Status is Running for each CPUFunction, GPUFunction, FPGAFunction, PCIeConnection, and EthernetConnection
・ FROMFUNC_STATUS and TOFUNC_STATUS for each PCIeConnection and each EthernetConnection must be OK.
・The STATUS for each Pod is Running

```
$ kubectl get dataflows.example.com -A
$ kubectl get cpufunctions.example.com -A
$ kubectl get gpufunctions.example.com -A
$ kubectl get ethernetconnections.example.com -A
$ kubectl get fpgafunctions.example.com -A
$ kubectl get pcieconnections.example.com -A
$ kubectl get pod -n test01 -o wide
```

5.3.2 Start video reception

Target：K8s control plane

(1) Method for starting video reception

i. For copy branch DF

Execute the same procedure as in section 1.3.2.

ii. For DF with Glue conversion

Deployment of the video distribution tool and creation of a receiving script (1 for receiving advanced inference results).

(Execute only the first time, step i is unnecessary from the second time onward)

```
$ cd ~/controller/sample-functions/utils/rcv_vide_tool/
$ kubectl create ns test
$ kubectl apply -f rcv_video_tool.yaml
$ kubectl get pod -n test    *Checking the pod name of video stream tool
$ kubectl exec -n test -it rcv-video-tool-xxx -- bash   *The xxx part is
changed according to the pod name confirmed above.
# vi start_test2.sh    *Paste and save the following
```

```
#!/bin/bash -x

for i in `seq -w 01 01`
do
    gst-launch-1.0 -e udpsrc buffer-size=21299100 mtu=8900 port=20${i} !
'application/x-rtp, media=(string)video, clock-rate=(int)90000, encoding-
name=(string)RAW, sampling=(string)BGR, depth=(string)8, width=(string)1280,
height=(string)1280, payload=(int)96' ! rtpvrawdepay ! queue ! videoconvert !
'video/x-raw, format=(string)I420' ! openh264enc ! 'video/x-h264, stream-
format=byte-stream, profile=(string)high' ! perf name=stream${i} ! h264parse !
qtmux ! filesink location=/tmp/output_st${i}.mp4 sync=false >
/tmp/rcv_video_tool_st${i}.log &
done
```

```
# chmod +x start_test2.sh
# exit
```

In a new window, open a terminal for the receiving tool, climb into the Pod and start receiving video.

```
$ kubectl exec -n test -it rcv-video-tool-XXX -- bash  *Change pod name to
suit your environment
# ./start_test2.sh
```

Check the start of the video reception tool.

*ps auxwwf to see that 1 process with COMMAND starting at ¥_ gst-launch-1.0 is running.

```
$ ps auxwwf
USER         PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root         971  5.0  0.0   6048  2836 pts/49   Rs+  10:03   0:00 ps auxwwf
...
root         954  0.0  0.0 317188 13380 pts/48   Sl+  09:57   0:00  ¥_ gst-
launch-1.0 -e udpsrc buffer-size=21299100 mtu=8900 port=2008 !
application/x-rtp, media=(string)video, clock-rate=(int)90000, encoding-
name=(string)RAW, sampling=(string)BGR, depth=(string)8,
width=(string)1280, height=(string)1280, payload=(int)96 ! rtpvrawdepay !
queue ! videoconvert ! video/x-raw, format=(string)I420 ! openh264enc !
video/x-h264, stream-format=byte-stream, profile=(string)high ! queue !
perf name=stream08 ! h264parse ! qtmux ! filesink
location=/tmp/output_st08.mp4 sync=1
...
```

5.3.3 Start of video stream

Target : K8s control plane

(1) Method for starting video stream

*Copy Branch DF and Glue Transform DF Common

i. Deployment of video stream tool and creation of stream script
(4K 15fps (-for advanced inference) × 1 stream)

(Perform this procedure only for the first time. This procedure (1) is not necessary for the second and subsequent times.)

```
$ cd ~/controller/sample-functions/utils/send_video_tool/
$ kubectl apply -f send_video_tool.yaml
$ kubectl get pod -n test    *Checking the pod name of video stream tool
$ kubectl exec -n test -it send-video-tool-xxx -- bash  *The xxx part is
changed according to the pod name confirmed above.
# vi start_test2.sh    *Paste and save the following
```

```
#!/bin/bash

sleep_time=$1

./start_gst_sender.sh /opt/video/input_4K_15fps.mp4 192.174.90.101 5004 1
${sleep_time:-3}
```

```
# chmod +x start_test2.sh
# exit
```

✧ Explanation of arguments for start_gst_sender.sh written in the above script.

- *See paragraph section 1.3.3 (1).

ii. In a new window, you open a terminal for the distribution tool, climb into the Pod, and start streaming.

```
$ kubectl exec -n test -it send-video-tool-XXX -- bash  *Change pod name to
suit your environment
# ./start_test2.sh
```

(A CRITICAL log may be output only on the first time, but you can return to the prompt by pressing Enter. If the process startup can be confirmed in the next step, there is no problem.)

iii. Confirm start within the video stream tool's container.

*Verify that one gst-launch is running on ps aux.

```
# ps aux | grep gst-launch
root       2335  0.0  0.0 309636 12272 pts/3    Sl   11:17   0:00 gst-launch-
1.0  filesrc  location=/mnt/input_4K_15fps.mp4  !  qtdemux  !  video/x-h264  !
h264parse   !   rtph264pay   config-interval=-1   seqnum-offset=1   !   udpsink
host=192.174.90.101 port=5004 buffer-size=2048
```

(2) Video input/output check of DataFlow

Confirm using the same procedure as in section 1.3.3 (2).

5.3.4 Video stream/reception stop

Check section 1.3.4.

5.4 Environmental shutdown

Check section 1.4.