OpenKasugai Demo

v1.1.0

目次

0. はじめに

本書では、OpenKasugai-Controller v1.1.0 のデモ手順を記載する。なお、前提として別紙「OpenKasugai-Controller-InstallManual」に沿ってデモの実行環境を構築していることを前提としている。

1. 基本機能#1 のデモ手順

本章では、OpenKasugai-Controller v1.1.0 の基本機能のうち、DataFlow 配備の際に自動的に FPGA デバイスに Bitstream(子 bs)を書込む機能のデモの手順を示す。本デモでは、初期状態のシステムに対して FPGA を用いる DataFlow の配備を行い、配備が完了した後に 2 本目の DataFlow を配備することで、2 本の DataFlow の両方が、同じ FPGA デバイスの同じ配備領域(Lane)を使用する例の手順を示す。

1.1 事前準備

1.1.1 試験スクリプトの配置

対象：K8s control plane、全ての K8s node

本書内で使用している評価手順簡略化のためのスクリプト（各種 shell スクリプトおよび映像配信・受信ツールの配備用 YAML ファイル）を配置する。

1. 取得済みの資材「CRC ソースコード&サンプルデータ」内にある試験スクリプトディレクトリ（script）をホームディレクトリにコピーする。

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

2. 必要があれば各スクリプトの以下変数を環境に応じて適切に修正する。

表 1：スクリプトと変数一覧

ファイル名	対象	変数名	設定値
run_controllers.sh	control plane	YAML_DIR K8S_SOFT_DIR	・yaml/の絶対パス ・controller/の絶対パス
delete_all.sh		YAML_DIR K8S_SOFT_DIR	・yaml/の絶対パス ・controller/の絶対パス
reset_ph3_fpga.sh	FPGA カードを搭載している node	BIT_DIR	・子 bs ファイルがあるディレクトリの絶対パス

※上記 reset_ph3_fpga.sh では、mcap ツールにて子 bs の書込みを行うが、mcap コマンドにおけるオプションの-x(PCI デバイス ID 指定)と-s(BDF 指定)は環境依存の値となるため、構築する環境に合わせて変更する必要がある。

各環境(K8s node)での-x や-s の値は lspci コマンドによって確認出来る。下記 lspci の実行例では、FPGA カードの情報が枚数分表示されており、xx:xx.x （例：1f:00.0）が BDF、903f が PCI デバイス ID にあたり、青文字が-s の値で赤文字が-x の値になる。

```
$ 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 DataFlow の yaml の編集

対象：K8s control plane

1.3.1 節で配備する各 DataFlow の yaml について、DataFlow の各処理モジュールの Pod が使用するネットワーク情報（IP アドレス・ポート番号）を必要に応じて変更する。

各サーバの 100G NIC に設定している IP アドレスが別紙「OpenKasugai-Controller-InstallManual」の 1.1 節の図 1 の IP アドレスと異なる場合は変更が必要となる。

変更方法は、別紙「OpenKasugai-Controller-InstallManual_Attachment1」のシート「2(補足).DataFlow yaml の設定」を参照すること。

表 2：1.3.1 節で配備する DataFlow の yaml 一覧

DataFlow の種類	編集対象のサンプルデータの yaml
FPGA F/R の DF 1 本目 (df-test-1-1)	~/controller/test/sample-data/sample-data-demo/yaml/dataflows/test-1/df-test-1-1.yaml
FPGA F/R の DF 2 本目 (df-test-1-2)	~/controller/test/sample-data/sample-data-demo/yaml/dataflows/test-1/df-test-1-2.yaml

※上記サンプルデータの DataFlow の名前（metadata.name の値）を変更する場合は、名前を 14 文字以下にすること。文字数制限の詳細は、別紙「OpenKasugai-Controller_Attachment1（CR/CM 仕様書）」を参照すること。

1.2 環境初期化

※再度 DataFlow の配備・映像配信を行う際は、必ず「1.3.4 映像配信・受信停止」「1.4 環境停止」を実施した上で、以下の手順から開始すること。

1.2.1 Bitstream(子 bs)の書込みと自動収集&CM 作成ツールの再実行

対象：FPGA カードを搭載している全ての K8s node

(1) 子 bs の書込み

以下のスクリプトを実行して、FPGA に子 bs ファイルの書き込みとリセットを実施する。

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

※子 bs の書き込みは数十秒かかる。
書き込みが成功した場合はログ中に FPGA Configuration Done!!と表示される。

FPGA の初期化に伴い、使用済みのランタイムディレクトリ／全 Hugepage およびディレクトリの削除が必要となるため、以下の手順を実施する。

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

(2) 自動収集&CM 作成ツールの実行

FPGA に関連する ConfigMap と CustomResource を初期状態に戻すため、対象の K8s node にて自動集&CM 作成ツールを実行する。

```
$ 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. ln -s コマンドは、別紙「OpenKasugai-Controller-InstallManual」の 8.6 節の手順 3 にて既に実施済みであればスキップする

※2. NVIDIA GPU 未搭載の K8s Node の場合、DCGM をインストールしていないので、InfoCollector のログに以下の様なエラーメッセージが出力されているが無視して良い。

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

1.2.2 CRC 起動

対象：K8s control plane

(1) 各 CRC の起動

下記コマンドを実行し、各 CRC の起動とファンクションカタログの yaml 適用を行う。

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

(2) CRC 起動と CRD インストールの確認

下記コマンドを入力し、各種 CRC の Pod が RUNNING となっていること、各 CRD（20 種）がインストールされていること、cpu-decode-filter-resize-high-infer-chain の FunctionChain が Ready になっていることを確認する。

確認結果が下記の出力と異なる場合は、各 Pod のログや kubectl describe の結果を確認するなどで原因特定を行った後に「1.4 環境停止」を実施し、対処後に「1.2 環境初期化」からやり直すこと。

```
$ 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 映像配信

1.3.1 DataFlow 配備

対象： K8s control plane

本節では DataFlow の配備と映像配信の流れを示す。

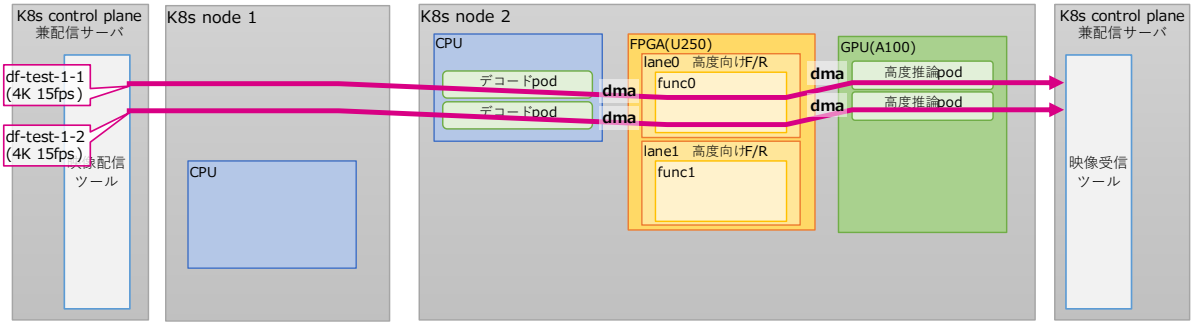

図 1： 本章のデモにおける DataFlow(2 本)の配備イメージ

(1) DataFlow の配備

以下では章の冒頭で示したパターン A の DataFlow(DF)の配備方法を示す。

サンプルデータの df-test-1-1 と df-test-1-2 の DF の yaml を 1 本ずつ適用する。

```
$ cd ~/controller/test/sample-data/sample-data-demo/yaml/dataflows/test-1/
$ kubectl apply -f df-test-1-1.yaml
$ kubectl get dataflow -A
（STATUS が Deployed になるまで 30 秒程度待つ）

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

(2) 配備結果の確認

各カスタムリソースの正常生成と各 Pod の正常配備を kubectl get コマンドの投入結果（STATUS）で確認する。

コマンド実行結果から以下を確認する。

・各 DataFlow の STATUS が Deployed であること

・各 CPUFunction、GPUFunction 、FPGAFunction、PCIeConnection の STATUS が Running であること

・各 PCIeConnection の FROMFUNC_STATUS と TOFUNC_STATUS が OK であること

・各 EthernetConnection についてはリソースが作成されていないこと（"No resources found"が表示されること）

・各 Pod の STATUS が Running であること

```
$ 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 映像受信起動

対象：K8s control plane

(1) 映像受信の起動方法

① 映像受信ツールの配備と受信用スクリプト（高度推論結果受信×2）の作成
（初回のみ実施、2 回目以降は手順①は不要）

```
$ cd ~/controller/sample-functions/utils/rcv_video_tool/
$ kubectl create ns test
$ kubectl apply -f rcv_video_tool.yaml
$ kubectl get pod -n test       ※映像配信ツールの pod 名を確認する
$ kubectl exec -n test -it rcv-video-tool-xxx -- bash   ※xxx の部分は上記で確認した pod 名に合わせて変更
# vi start_test1.sh   ※以下を貼り付けて保存する
```

```
#!/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
```

② 新規ウィンドウで受信ツール用にターミナルを開き、Pod に乗り込んで映像受信を起動する。

```
$ kubectl exec -n test -it rcv-video-tool-xxx -- bash   ※pod 名は環境に合わせて変更
# ./start_test1.sh
```

（初回のみは CRITICAL／WARNING ログが出力されるが Enter 押下でプロンプトに戻って良い。次手順でプロセスの起動が確認できれば問題ない）

③ コンテナ内で映像受信ツールの起動確認

※ps auxwwf で COMMAND が¥_ gst-launch-1.0 から始まるプロセスが 2 個動いていることを確認する。

```
# 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 映像配信開始

対象：K8s control plane

(1) 映像配信の開始方法

① 映像配信ツールの配備と配信スクリプト（4K 15fps(高度推論用)×2 本）の作成
（初回のみ実施、2 回目以降は本手順①は不要）

```
$ cd ~/controller/sample-functions/utils/send_video_tool/
$ kubectl apply -f send_video_tool.yaml
$ kubectl get pod -n test    ※映像配信ツールの pod 名を確認する
$ kubectl exec -n test -it send-video-tool-xxx -- bash  ※xxx の部分は上記で確認した pod 名に合わせて変更
# vi start_test1.sh    ※以下を貼り付けて保存する
```

```
#!/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
```

表 3：上記スクリプトに書く start_gst_sender.sh の引数説明

n 個目	説明	値について
1	動画ファイルパス	配信する動画ファイルを指定。使用する動画ファイルに応じて変更すること。(上記の例の動画は「OpenKasugai-Controller-InstallManual」の 7.8 節で作成したもの) 動画ファイルは 4K 15fps の動画を使用すること。また、動画の長さは 30～60 秒程度が望ましい。
2	送信先 IP	送信先 IP アドレスを指定。DataFlow の yaml に記載した CPU デコードに割り当てる IP アドレスに合わせて変更すること。
3	送信先ポート	送信先ポート番号を指定。DataFlow の yaml に記載した CPU デコードに設定する受信ポート番号に合わせて変更すること
4	起動プロセス数	生成する配信プロセス数を指定（2 以上を指定した場合に同一の動画ファイルを同一の送信先 IP に対して配信するプロセスを指定した数だけ起動する。送信先ポート番号は配信プロセスを起動するたびにインクリメントされる）
5	配信遅延間隔	動画の配信開始遅延[秒]（4 個目の引数にて 2 以上を指定した場合の配信プロセスの起動間隔）

② 新規ウィンドウで配信ツール用にターミナルを開き、Pod に乗り込んで映像配信を開始する。

```
$ kubectl exec -n test -it send-video-tool-xxx -- bash  ※pod 名は環境に合わせて変更
# ./start_test1.sh
```

（初回のみ CRITICAL ログが出力される場合があるが Enter 押下でプロンプトに戻って良い。
次手順でプロセスの起動が確認できれば問題ない）

③　コンテナ内で映像配信ツールの起動確認
※ps aux で 2 本分 gst-launch が動いていることを確認。

```
# 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)　DataFlow の映像入出力確認

本手順により、配備した DataFlow に映像配信ツールから映像が入力され DataFlow で処理した結果映像が映像受信ツールに届く。配信映像の動画ファイルは動画時間が経過すると、映像配信が停止するが、任意のタイミングで次節の手順を実施することで配信が停止される。

映像が受信されているかの確認は、映像受信ツールコンテナ内の/tmp 配下の mp4 ファイルを監視することで可能。

```
$ kubectl exec -n test -it rcv-video-tool-xxx -- bash　※pod 名は環境に合わせて変更
# cd /tmp/
# ls -l
（各 mp4 ファイルのサイズ増加を確認）
```

1.3.4 映像配信・受信停止

対象：K8s control plane

1. 1.3.3 節で開始した映像配信ツールのコンテナ内で以下で停止して、停止したことを確認。

```
$ kubectl exec -n test -it send-video-tool-xxx -- bash　※pod 名は環境に合わせて変更
# ./stop_gst_sender.sh
# ps aux
　　（※gst-launch が動いていないことを確認）
```

2. 映像受信ツールのコンテナ内で以下で停止して、停止したことを確認。

```
$ kubectl exec -n test -it rcv-video-tool-xxx -- bash　※pod 名は環境に合わせて変更
# pgrep gst-launch-1.0 | xargs kill -2 > /dev/null
# ps aux
　　（※gst-launch が動いていないことを確認）
```

※以下の様なエラーメッセージが出力される場合があるが無視して良い。
[OpenH264] this = 0x0x7fccc40058c0, Error:CWelsH264SVCEncoder::EncodeFrame(), cmInitParaError.

3. 出力された動画ファイルを 1 つずつホストにコピー。

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

※コピー処理に失敗する場合があるため、--retries 10 のオプションをつけてコマンドを実行する
※以下の様なメッセージが出力されても動画ファイルがコピーされていれば無視して良い。
"tar: Removing leading `/' from member names"

4. 出力される動画は、検出された人物に対する Bounding Box(BB)が描画された動画となる。
※1.3.1 節で配備する「FPGA F/R —(PCIe 接続)→ GPU 高度推論」の構成を含む DF の場合は、出力動画の最初の約 1 分間真っ白な映像が流れ、その後 BB が描画された映像が流れるが、これは正常な結果である。真っ白な画像は、GPU 高度推論にデータがまだ来ていない時のダミーの入力データである。

1.4 環境停止

1.4.1 DataFlow 削除と各種 CR の削除、各 CRC の停止

対象：K8s control plane

以下のスクリプトを実行し、DataFlow の削除と各種 CR の削除、各 CRC 停止をする。

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

上記のスクリプト実行後、別紙「OpenKasugai-Controller-InstallManual」の手順で登録した一部の CRD と ConfigMap が以下の様に残っているのは正常であるため、手動で削除する必要はない。

```
$ 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
kube-root-ca.crt                              1        83d
```

※再度 DataFlow の配備・映像配信を行う場合は、「1.2 環境初期化」から実施すること

2. 基本機能#2 のデモ手順

OpenKasugai-Controller v1.1.0 より DataFlow を削除する際に、削除する DataFlow が使用していた FPGA 内の各種リソースを回収する機能が追加された。この機能の追加により FPGA 内の一部のリソース(Function 内部のリソース)を除き、回収された FPGA 内リソースを再利用可能となった。
Function 内部のリソース(フィルタリサイズ用の Bitstream(子 bs)ならフィルタリサイズ回路内のリソース)となる「チャネル」に関しては Function のロジック依存で状態や設定情報が残る可能性があるため回収の対象外となる。しかし、チャネルは複数存在しており未使用のチャネルは使用可能である。そのため、使用済みチャネルを管理して DataFlow 配備時には未使用チャネルを割当てることで、子 bs の再書込みを行うことなく DataFlow を配備することが可能となった。(詳細については別紙「Openkasugai-Controller」の基本コンセプト⑧を参照)
本章では DataFlow の削除機能に関するデモの手順を示す。本デモでは、初期状態のシステムに対して FPGA を用いる DataFlow(0 章で配備した DataFlow と同じ構成)の配備を行い、当該 DataFlow 削除を行った後、同じ FPGA に対して新たな DataFlow を配備する手順を示す。

2.1 事前準備

2.1.1 試験スクリプトの配置

対象：K8s control plane、全ての K8s node

1.1.1 節を実施すること。既に実施している場合はスキップする。

2.1.2 DataFlow の yaml の編集

対象：K8s control plane

2.3.1 節で配備する各 DataFlow の yaml について、DataFlow の各処理モジュールの Pod が使用するネットワーク情報（IP アドレス・ポート番号）を必要に応じて変更する。
各サーバの 100G NIC に設定している IP アドレスが別紙「OpenKasugai-Controller-InstallManual」の 1.1 節の図 1 の IP アドレスと異なる場合は変更が必要となる。
変更方法は、別紙「OpenKasugai-Controller-InstallManual_Attachment1」のシート「2(補足).DataFlow yaml の設定」を参照すること。

表 4：2.3.1 節で配備する DataFlow の yaml 一覧

DataFlow の種類	編集対象のサンプルデータの yaml
1. FPGA F/R の DF 1 本目 (df-test-2-1)	~/controller/test/sample-data/sample-data-demo/yaml/dataflows/test-2/df-test-2-1.yaml
2. FPGA F/R の DF 2 本目 (df-test-2-2)	~/controller/test/sample-data/sample-data-demo/yaml/dataflows/test-2/df-test-2-2.yaml

※上記サンプルデータの DataFlow の名前（metadata.name の値）を変更する場合は、名前を 14 文字以下にすること。文字数制限の詳細は、別紙「OpenKasugai-Controller_Attachment1（CR/CM 仕様書）」を参照すること。

2.2 環境初期化

1.2 節を実施する。

2.3 DataFlow(1 本目)の映像配信

2.3.1 DataFlow 配備

対象：K8s control plane

本節では 1 本目の DataFlow の配備と映像配信の流れを示す。

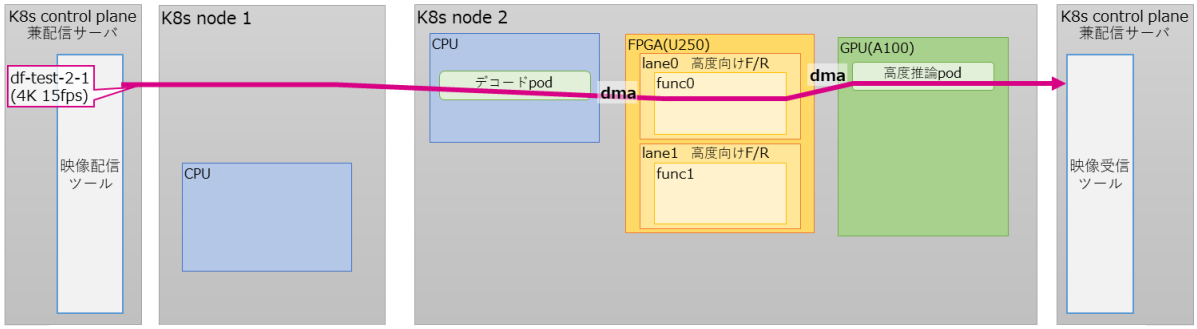

図 2： 本章のデモにおける 1 本目の DataFlow の配備イメージ

(1) DataFlow の配備

サンプルデータの df-test-2-1 の yaml を適用する。

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

(2) 配備結果の確認

各カスタムリソースの正常生成と各 Pod の正常配備を 1.3.1 節の(2)を実施して確認する。

また、配備された DataFlow が使用している FPGA デバイスを確認する。
FPGA デバイスは DataFlow のうち FPGAFunction CR にて使われるため、当該 DataFlow の FPGAFunction CR の情報を確認する。

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

出力結果の例は以下になる。当該 DataFlow の FPGAFunction かどうかを確認するには status.dataFlowRef を、使用している FPGA デバイスを確認するには status.acceleratorStatuses.partionName.statsuses.acceleratorID を参照する。

```
apiVersion: v1
items:
- apiVersion: example.com/v1
  kind: FPGAFunction
  metadata: (略)
  spec: (略)
  status:
    acceleratorStatuses:
      - partitionName: "0"
        statuses:
        - acceleratorID: /dev/xpcie_21320621V01L        使用している FPGA のデバイスファイルパス
          status: OK
      childBitstreamName: 0100001c
      dataFlowRef:
        name: df-test-2-1                               DataFlow 名
        namespace: test01                               名前空間名
      (略)
```

2.3.2 映像受信起動

対象：K8s control plane

(1) 映像受信の起動方法

① 映像配信ツールの配備と受信用スクリプト（高度推論結果受信×1）の作成
（初回のみ実施、2 回目以降は手順①は不要）

```
$ cd ~/controller/sample-functions/utils/rcv_video_tool/
$ kubectl create ns test
$ kubectl apply -f rcv_video_tool.yaml
$ kubectl get pod -n test　　※映像配信ツールの pod 名を確認する
$ kubectl exec -n test -it rcv-video-tool-xxx -- bash　※xxx の部分は上記で確認した pod 名に合わせて変更
# vi start_test2.sh　　※以下を貼り付けて保存する
```

```
#!/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
```

② 新規ウィンドウで受信ツール用にターミナルを開き、Pod に乗り込んで映像受信を起動する。

```
$ kubectl exec -n test -it rcv-video-tool-xxx -- bash　※pod 名は環境に合わせて変更
# ./start_test2.sh
```

③ コンテナ内で映像受信ツールの起動確認

※ps auxwwf で COMMAND が¥_ gst-launch-1.0 から始まるプロセスが 1 個動いていることを確認する。

```
# 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 映像配信開始

対象：K8s control plane

(1) 映像配信の開始方法

① 映像配信ツールの配備と配信スクリプト（4K 15fps(高度推論用)×1 本）の作成

（初回のみ実施、2 回目以降は本手順①は不要）

```
$ cd ~/controller/sample-functions/utils/send_video_tool/
$ kubectl apply -f send_video_tool.yaml
$ kubectl get pod -n test    ※映像配信ツールの pod 名を確認する
$ kubectl exec -n test -it send-video-tool-xxx -- bash  ※xxx の部分は上記で確認した pod 名に合わせて変更
# vi start_test2.sh    ※以下を貼り付けて保存する
```

```
#!/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
```

※start_gst_sender.sh の引数については 1.3.3 節の(1)の表 3 を参照

② 新規ウィンドウで配信ツール用にターミナルを開き、Pod に乗り込んで映像配信を開始する。

```
$ kubectl exec -n test -it send-video-tool-xxx -- bash  ※pod 名は環境に合わせて変更
# ./start_test2.sh
```

（初回のみ CRITICAL ログが出力される場合があるが Enter 押下でプロンプトに戻って良い。
次手順でプロセスの起動が確認できれば問題ない）

③　コンテナ内で映像配信ツールの起動確認
※ps aux で 1 本分 gst-launch が動いていることを確認。

```
# 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)　DataFlow の映像入出力確認
1.3.3 節の(2)を実施する。

2.3.4　映像配信・受信停止

対象：K8s control plane

1.3.4 節を実施する。

2.4 DataFlow(1 本目)の削除

2.4.1　DataFlow の削除

対象：K8s control plane

2.3.1 節で配備した DataFlow を削除する。

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

2.4.2　DataFlow の削除確認

対象：K8s control plane

本節は DataFlow を構成する各種カスタムリソース(CR)や処理モジュール用の Pod が削除されていることを確認したい場合に実施する。削除した DataFlow(df-test-2-1)の場合に削除されるリソースは以下になる。
DataFlow、(DataFlow の構成要素である)**WBFunction** と **WBConnection**、
(WBFunction のサブリソースである)**CPUFunction** と **GPUFunction** と **FPGAFunction**、
(CPUFunction や GPUFunction のサブリソースである)**Pod**
(WBConnection のサブリソースである)**PCIeConnection**

CR の削除の確認方法は以下の様に kubectl_get.sh を実行して、上記の削除対象リソースが無くなっていることを確認する。

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

kubectl_get.sh の結果(上記の場合は~/script/logs/deleted_kubectl_get.log に出力される)は以下の様になる。削除した DataFlow(df-test-2-1)に関する CR や Pod が表示されていないことを確認する。

```
(略)
+ 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
(略)
```

2.5 DataFlow(2 本目)の映像配信

2.5.1 DataFlow 配備

対象：K8s control plane

本節では 2 本目の DataFlow の配備と映像配信の流れを示す。

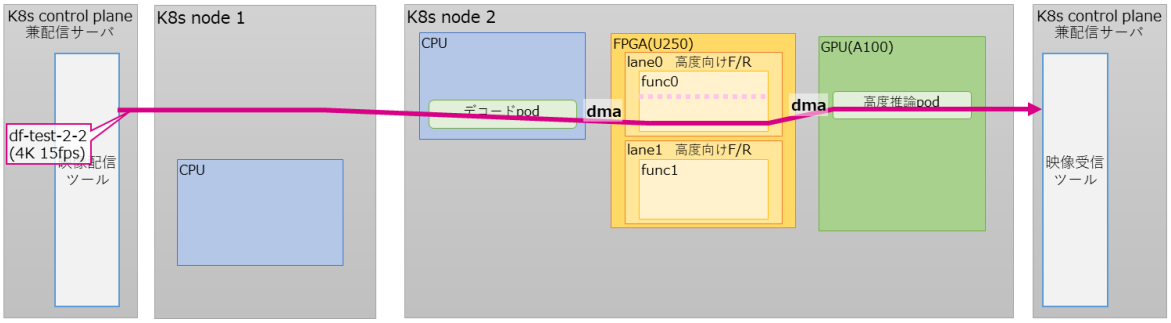

図 3： 本章のデモにおける 2 本目の DataFlow の配備イメージ

(1) DataFlow の配備

サンプルデータの df-test-2-2 の yaml を適用する。

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

(2) 配備結果の確認

各カスタムリソースの正常生成と各 Pod の正常配備を 1.3.1 節の(2)を実施して確認する。

また、配備された DataFlow が使用している FPGA デバイスを確認する。
FPGA デバイスは DataFlow のうち FPGAFunction CR にて使われるため、当該 DataFlow の FPGAFunction CR の情報を確認する。

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

出力結果の例は以下になる。当該 DataFlow の FPGAFunction かどうかを確認するには status.dataFlowRef を、使用している FPGA デバイスを確認するには status.acceleratorStatuses.partionName.statsuses.acceleratorID を参照する。
2.4 節で 1 本目の DataFlow を削除しているため、本節で新たに配備した DataFlow は 1 本目の DataFlow が使用していた FPGA デバイスと同じ FPGA デバイスを使用しているはずなので、acceleratorID の値が 2.3.1 節の(2)で確認した 1 本目の DataFlow と同じであることを確認する。

```
apiVersion: v1
items:
- apiVersion: example.com/v1
  kind: FPGAFunction
  metadata: (略)
  spec: (略)
  status:
    acceleratorStatuses:
      - partitionName: "0"
        statuses:
        - acceleratorID: /dev/xpcie_21320621V01L        使用している FPGA のデバイスファイルパス
          status: OK
      childBitstreamName: 0100001c
      dataFlowRef:
        name: df-test-2-2                               DataFlow 名
        namespace: test01                               名前空間
      (略)
```

2.5.2 映像受信起動

対象：K8s control plane

2.3.2 節を実施する。

2.5.3 映像配信開始

対象：K8s control plane

2.3.3 節を実施する。

2.5.4 映像配信・受信停止

対象：K8s control plane

1.3.4 節を実施する。

2.6 DataFlow(2 本目)の削除

2.6.1 DataFlow の削除

対象：K8s control plane

2.5.1 節で配備した DataFlow を削除する。

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

2.6.2 DataFlow の削除確認

対象：K8s control plane

削除した DataFlow(df-test-2-2)に関する CR や Pod が表示されていないことを確認したい場合に実施する。実施内容は 2.4.2 節を参照。

※3 章のデモを実施する場合は環境停止を実施しない。(3 章のデモを実施しない場合は 1.4 節を実施して環境停止を行う。)

3. 基本機能#3 のデモ手順

OpenKasugai-Controller v1.1.0 より追加された FPGAReconfigurationTool を用いることで、環境停止や環境初期化を行うことなく Bitstream(子 bs)の書込みが可能となった。本章では FPGAReconfigurationTool のリセット機能（FPGA のリセットと子 bs のリセット）に関するデモの手順を示す。本デモでは、まず FPGA のリセットによって 2 章で使用した FPGA を子 bs 未書込み状態(環境初期化実施直後と同じ状態)に戻す。その後 FPGA を用いる DataFlow(0 章で配備した DataFlow と同じ構成)の配備と削除を繰り返して FPGA の Function 内部のリソース(チャネル)を使い切ったうえで子 bs のリセットを実施して、当該 FPGA を再活用可能にする(新たな DataFlow を配備する)。

3.1 事前準備

3.1.1 試験スクリプトの配置

対象：K8s control plane、全ての K8s node

1.1.1 節を実施すること。既に実施している場合はスキップする。

3.1.2 DataFlow の yaml の編集

対象：K8s control plane

3.3 節で配備する各 DataFlow の yaml について、DataFlow の各処理モジュールの Pod が使用するネットワーク情報（IP アドレス・ポート番号）を必要に応じて変更する。

各サーバの 100G NIC に設定している IP アドレスが別紙「OpenKasugai-Controller-InstallManual」の 1.1 節の図 1 の IP アドレスと異なる場合は変更が必要となる。

変更方法は、別紙「OpenKasugai-Controller-InstallManual_Attachment1」のシート「2(補足).DataFlow yaml の設定」を参照すること。

表 5：3.3 節と 3.4 節で配備する DataFlow の yaml 一覧

DataFlow の種類	編集対象のサンプルデータの yaml
1. FPGA F/R の DF 1 本目 (df-test-3-1)	~/controller/test/sample-data/sample-data-demo/yaml/dataflows/test-3/df-test-3-1.yaml
2. FPGA F/R の DF 2 本目 (df-test-3-2)	~/controller/test/sample-data/sample-data-demo/yaml/dataflows/test-3/df-test-3-2.yaml
3. FPGA F/R の DF 3 本目 (df-test-3-3)	~/controller/test/sample-data/sample-data-demo/yaml/dataflows/test-3/df-test-3-3.yaml
4. FPGA F/R の DF 4 本目 (df-test-3-4)	~/controller/test/sample-data/sample-data-demo/yaml/dataflows/test-3/df-test-3-4.yaml
5. FPGA F/R の DF 5 本目 (df-test-3-5)	~/controller/test/sample-data/sample-data-demo/yaml/dataflows/test-3/df-test-3-5.yaml
6. FPGA F/R の DF 6 本目 (df-test-3-6)	~/controller/test/sample-data/sample-data-demo/yaml/dataflows/test-3/df-test-3-6.yaml
7. FPGA F/R の DF 7 本目 (df-test-3-7)	~/controller/test/sample-data/sample-data-demo/yaml/dataflows/test-3/df-test-3-7.yaml
8. FPGA F/R の DF 8 本目 (df-test-3-8)	~/controller/test/sample-data/sample-data-demo/yaml/dataflows/test-3/df-test-3-8.yaml
9. FPGA F/R の DF 9 本目 (df-test-3-9)	~/controller/test/sample-data/sample-data-demo/yaml/dataflows/test-3/df-test-3-9.yaml
10. FPGA F/R の DF 10 本目 (df-test-3-10)	~/controller/test/sample-data/sample-data-demo/yaml/dataflows/test-3/df-test-3-10.yaml

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

※上記サンプルデータの DataFlow の名前（metadata.name の値）を変更する場合は、名前を 14 文字以下にすること。文字数制限の詳細は、別紙「OpenKasugai-Controller_Attachment1（CR/CM 仕様書）」を参照すること。

3.2 FPGA のリセット

対象：DataFlow を配備した FPGA カードを搭載している全ての K8s node

事前に対象の FPGA に DataFlow が配備されていない(配備していた DataFlow が全て削除済み)ことを確認しておく。なお、2 章のデモの続きとして実施している場合はこの状態(DataFlow は全て削除済み)になっている。

ここでは別紙「OpenKasugai-Controller-InstallManual」の 1.1 節の図 1 の K8s node(“server2”)に搭載されている FPGA(デバイスファイルパスは”/dev/xpcie_21320621V01L”)に対して FPGA のリセットを行う場合の例を示す。

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

※なお、FPGAReconfigurationTool の使用方法については github(OpenKasugai)の controller リポジトリ(v1.1.0)の/src/tools/FPGAReconfigurationTool/ReadMe.txt を参照。

3.3 FPGA の Function 内部のリソース(チャネル)の使い切り

対象：K8s control plane

DataFlow の配備や削除を繰り返すことで FPGA の Function 内部のリソース(チャネル)を使い切る。本節では、想定環境(別紙「OpenKasugai-Controller-InstallManual」の 1.1 節の図 1 の環境と 0.4 節に記載の FPGA の bit 回路を使用)において高度推論を行う DataFlow を配備する場合の手順を示す。

この場合使用可能なチャネルは各配備領域(Lane)である lane0, lane1 ともに 8 つである。また 1 本の DataFlow に対して 1 つのチャネルを割当てる。従って、1 枚の FPGA のチャネルを使い切るには 16 本の DataFlow を配備する必要がある。一方で、FPGA の各 Lane のキャパシティの上限により、同時に配備可能な DataFlow は 2 本である。

上記により、1 枚の FPGA を使い切るために、DataFlow2 本の配備と削除を 8 回繰り返す必要がある。

以下で DataFlow の配備と削除を繰り返す流れを示す。

① DataFlow(1～2 本目)の配備、映像配信、削除

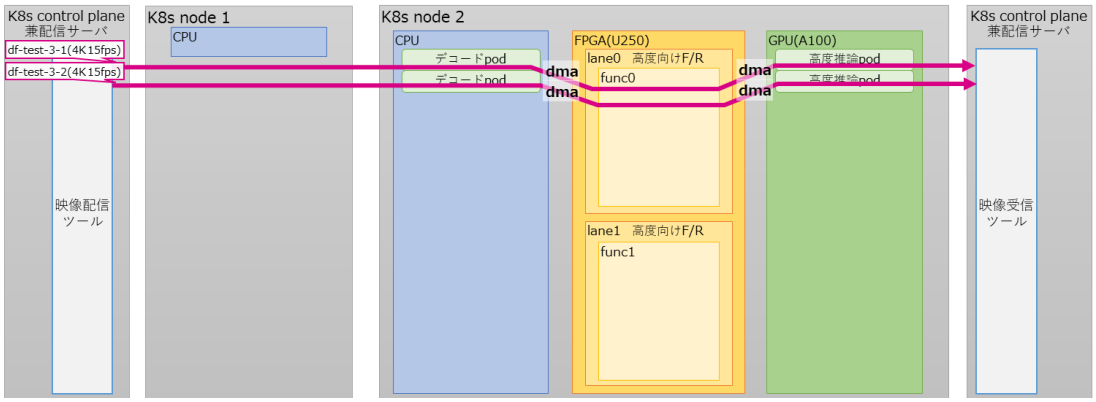

図 4： 本章のデモにおける 1～2 本目の DataFlow の配備イメージ

・DataFlow の配備

サンプルデータの df-test-3-1 と df-test-3-2 の DF の yaml を 1 本ずつ適用する。

```
$ cd ~/controller/test/sample-data/sample-data-demo/yaml/dataflows/test-3/
$ kubectl apply -f df-test-3-1.yaml
$ kubectl get dataflow -A
（STATUS が Deployed になるまで 30 秒程度待つ）

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

配備結果の確認を行う場合は 1.3.1 節の(2)を実施する。

・映像配信

映像受信を起動するため 1.3.2 節を実施する。その後映像配信を開始するため 1.3.3 節を実施する。

映像の入出力を確認出来た後、映像受信と映像配信を停止するため 1.3.4 節を実施する。

・DataFlow の削除

配備した DataFlow を削除する。

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

削除結果の確認を行う場合は 2.4.2 節を参照して、削除した DataFlow に関する CR や Pod が無いことを確認する。

② DataFlow(3～4 本目)の配備、映像配信、削除

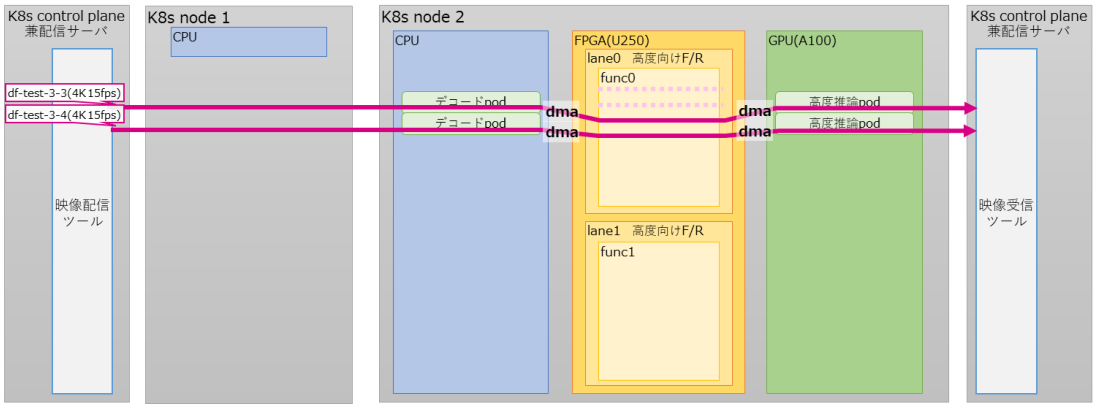

図 5： 本章のデモにおける 3～4 本目の DataFlow の配備イメージ

・DataFlow の配備

サンプルデータの df-test-3-3 と df-test-3-4 の DF の yaml を 1 本ずつ適用する。

```
$ kubectl apply -f df-test-3-3.yaml
（STATUS が Deployed になるまで 20 秒程度待つ）
$ kubectl apply -f df-test-3-4.yaml
```

配備結果の確認を行う場合は 1.3.1 節の(2)を実施する。

・映像配信

映像受信を起動するため 1.3.2 節を実施する。その後映像配信を開始するため 1.3.3 節を実施する。
映像の入出力を確認出来た後、映像受信と映像配信を停止するため 1.3.4 節を実施する。

・DataFlow の削除

配備した DataFlow を削除する。

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

削除結果の確認を行う場合は 2.4.2 節を参照して、削除した DataFlow に関する CR や Pod が無いことを確認する。

③ DataFlow(5～6 本目)の配備、映像配信、削除

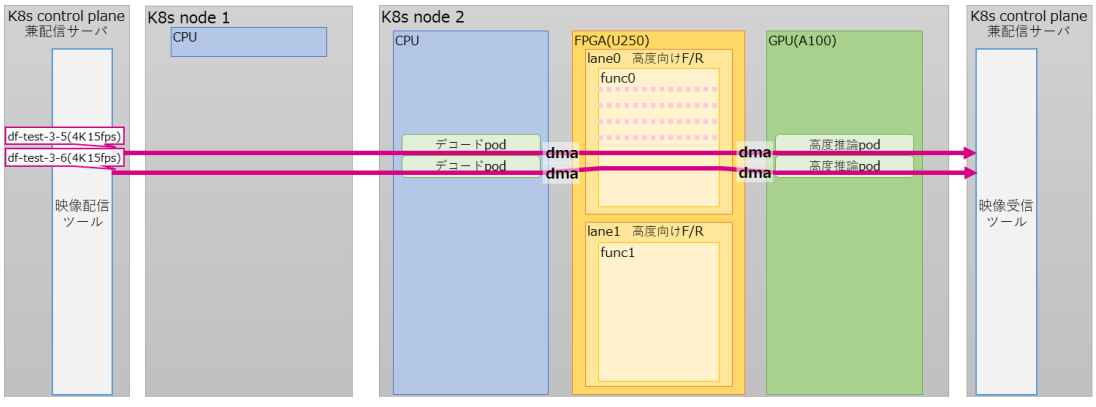

図 6： 本章のデモにおける 5～6 本目の DataFlow の配備イメージ

・DataFlow の配備

サンプルデータの df-test-3-5 と df-test-3-6 の DF の yaml を 1 本ずつ適用する。

```
$ kubectl apply -f df-test-3-5.yaml
（STATUS が Deployed になるまで 20 秒程度待つ）
$ kubectl apply -f df-test-3-6.yaml
```

配備結果の確認を行う場合は 1.3.1 節の(2)を実施する。

・映像配信

映像受信を起動するため 1.3.2 節を実施する。その後映像配信を開始するため 1.3.3 節を実施する。
映像の入出力を確認出来た後、映像受信と映像配信を停止するため 1.3.4 節を実施する。

・DataFlow の削除

配備した DataFlow を削除する。

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

削除結果の確認を行う場合は 2.4.2 節を参照して、削除した DataFlow に関する CR や Pod が無いことを確認する。

④ DataFlow(7～8 本目)の配備、映像配信、削除

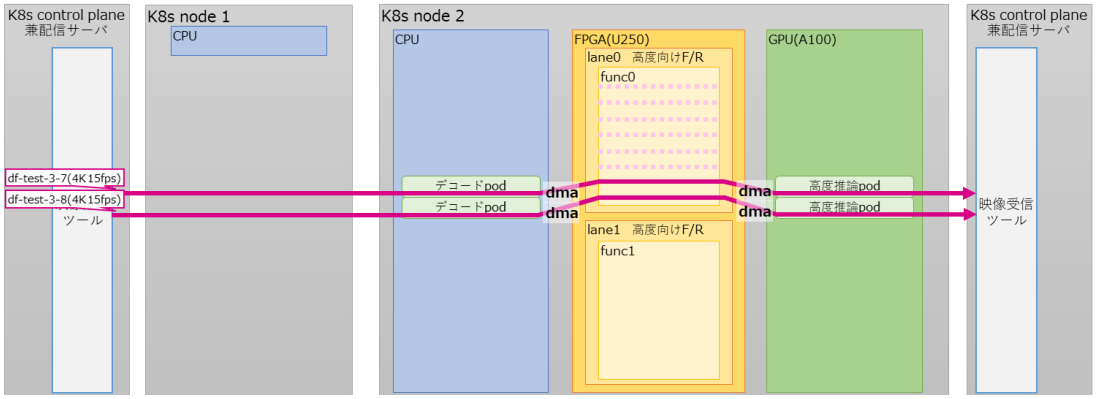

図 7： 本章のデモにおける 7～8 本目の DataFlow の配備イメージ

・DataFlow の配備

サンプルデータの df-test-3-7 と df-test-3-8 の DF の yaml を 1 本ずつ適用する。

```
$ kubectl apply -f df-test-3-7.yaml
（STATUS が Deployed になるまで 20 秒程度待つ）
$ kubectl apply -f df-test-3-8.yaml
```

配備結果の確認を行う場合は 1.3.1 節の(2)を実施する。

・映像配信

映像受信を起動するため 1.3.2 節を実施する。その後映像配信を開始するため 1.3.3 節を実施する。
映像の入出力を確認出来た後、映像受信と映像配信を停止するため 1.3.4 節を実施する。

・DataFlow の削除

配備した DataFlow を削除する。

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

削除結果の確認を行う場合は 2.4.2 節を参照して、削除した DataFlow に関する CR や Pod が無いことを確認する。

⑤ DataFlow(9～10 本目)の配備、映像配信、削除

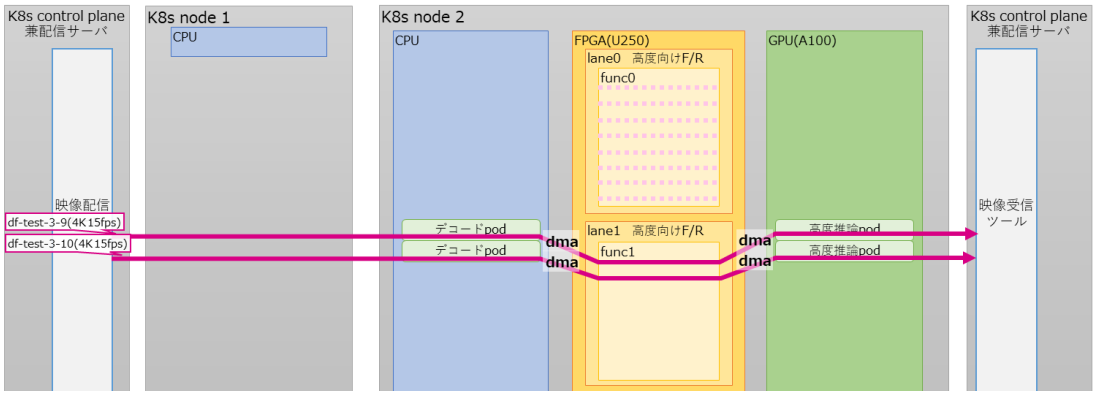

図 8： 本章のデモにおける 9～10 本目の DataFlow の配備イメージ

・DataFlow の配備

サンプルデータの df-test-3-9 と df-test-3-10 の DF の yaml を 1 本ずつ適用する。

```
$ kubectl apply -f df-test-3-9.yaml
（STATUS が Deployed になるまで 30 秒程度待つ）
$ kubectl apply -f df-test-3-10.yaml
```

配備結果の確認を行う場合は 1.3.1 節の(2)を実施する。

・映像配信

映像受信を起動するため 1.3.2 節を実施する。その後映像配信を開始するため 1.3.3 節を実施する。
映像の入出力を確認出来た後、映像受信と映像配信を停止するため 1.3.4 節を実施する。

・DataFlow の削除

配備した DataFlow を削除する。

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

削除結果の確認を行う場合は 2.4.2 節を参照して、削除した DataFlow に関する CR や Pod が無いことを確認する。

⑥ DataFlow(11～12 本目)の配備、映像配信、削除

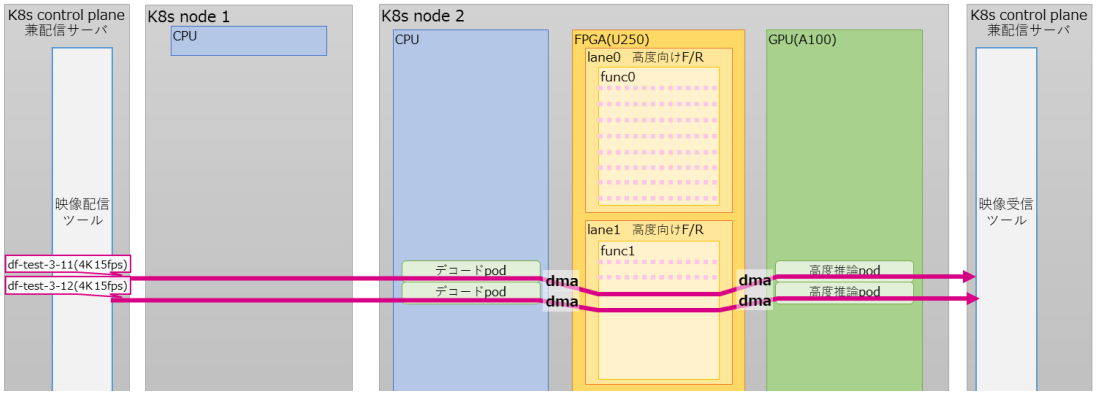

図 9： 本章のデモにおける 11～12 本目の DataFlow の配備イメージ

・DataFlow の配備

サンプルデータの df-test-3-11 と df-test-3-12 の DF の yaml を 1 本ずつ適用する。

```
$ kubectl apply -f df-test-3-11.yaml
（STATUS が Deployed になるまで 20 秒程度待つ）
$ kubectl apply -f df-test-3-12.yaml
```

配備結果の確認を行う場合は 1.3.1 節の(2)を実施する。

・映像配信

映像受信を起動するため 1.3.2 節を実施する。その後映像配信を開始するため 1.3.3 節を実施する。
映像の入出力を確認出来た後、映像受信と映像配信を停止するため 1.3.4 節を実施する。

・DataFlow の削除

配備した DataFlow を削除する。

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

削除結果の確認を行う場合は 2.4.2 節を参照して、削除した DataFlow に関する CR や Pod が無いことを確認する。

⑦ DataFlow(13～14 本目)の配備、映像配信、削除

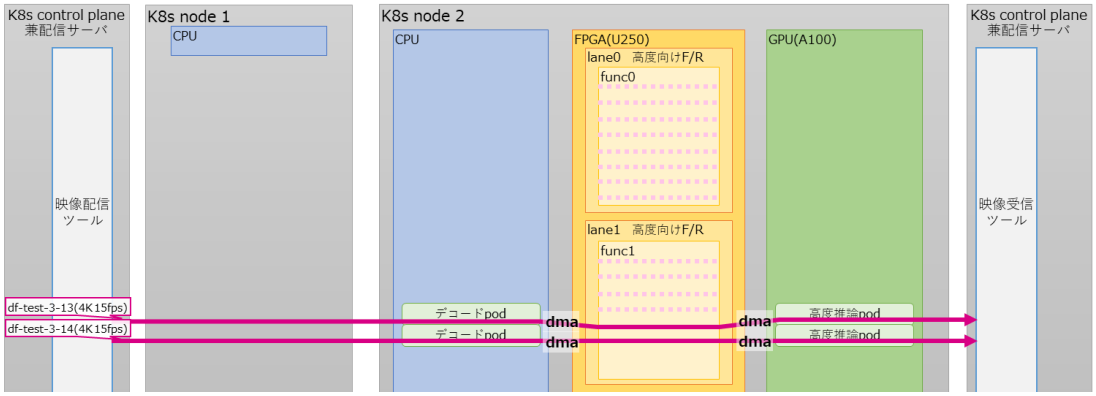

図 10： 本章のデモにおける 13～14 本目の DataFlow の配備イメージ

・DataFlow の配備

サンプルデータの df-test-3-13 と df-test-3-14 の DF の yaml を 1 本ずつ適用する。

```
$ kubectl apply -f df-test-3-13.yaml
（STATUS が Deployed になるまで 20 秒程度待つ）
$ kubectl apply -f df-test-3-14.yaml
```

配備結果の確認を行う場合は 1.3.1 節の(2)を実施する。

・映像配信

映像受信を起動するため 1.3.2 節を実施する。その後映像配信を開始するため 1.3.3 節を実施する。
映像の入出力を確認出来た後、映像受信と映像配信を停止するため 1.3.4 節を実施する。

・DataFlow の削除

配備した DataFlow を削除する。

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

削除結果の確認を行う場合は 2.4.2 節を参照して、削除した DataFlow に関する CR や Pod が無いことを確認する。

⑧ DataFlow(15～16 本目)の配備、映像配信、削除

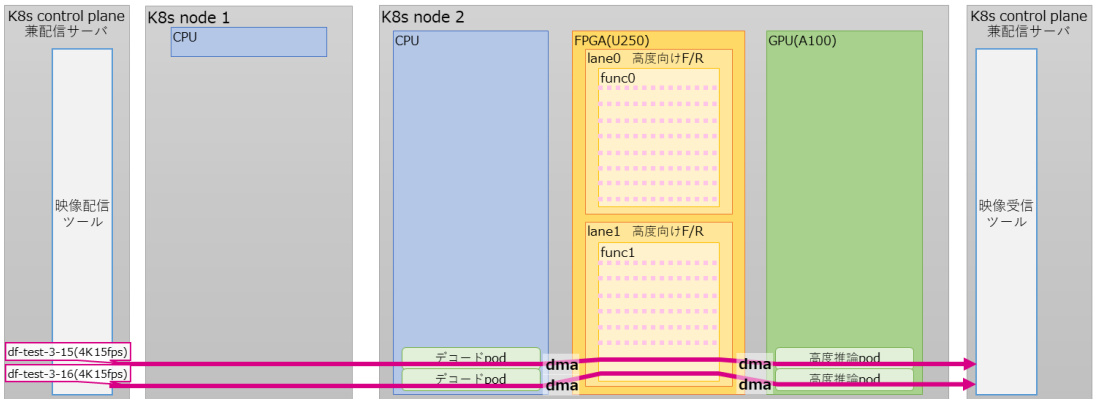

図 11： 本章のデモにおける 15～16 本目の DataFlow の配備イメージ

・DataFlow の配備

サンプルデータの df-test-3-15 と df-test-3-16 の DF の yaml を 1 本ずつ適用する。

```
$ kubectl apply -f df-test-3-15.yaml
（STATUS が Deployed になるまで 20 秒程度待つ）
$ kubectl apply -f df-test-3-16.yaml
```

配備結果の確認を行う場合は 1.3.1 節の(2)を実施する。

・映像配信

映像受信を起動するため 1.3.2 節を実施する。その後映像配信を開始するため 1.3.3 節を実施する。

映像の入出力を確認出来た後、映像受信と映像配信を停止するため 1.3.4 節を実施する。

・DataFlow の削除

配備した DataFlow を削除する。

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

削除結果の確認を行う場合は 2.4.2 節を参照して、削除した DataFlow に関する CR や Pod が無いことを確認する。

3.4 DataFlow の配備(失敗)とリソース使用状況の確認

3.3 節を実施後にさらに新たな DataFlow を配備を要求する。が、以下の様に配備に失敗する。

```
$ kubectl apply -f df-test-3-17.yaml
（5 秒程度待つ）
$ 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
```

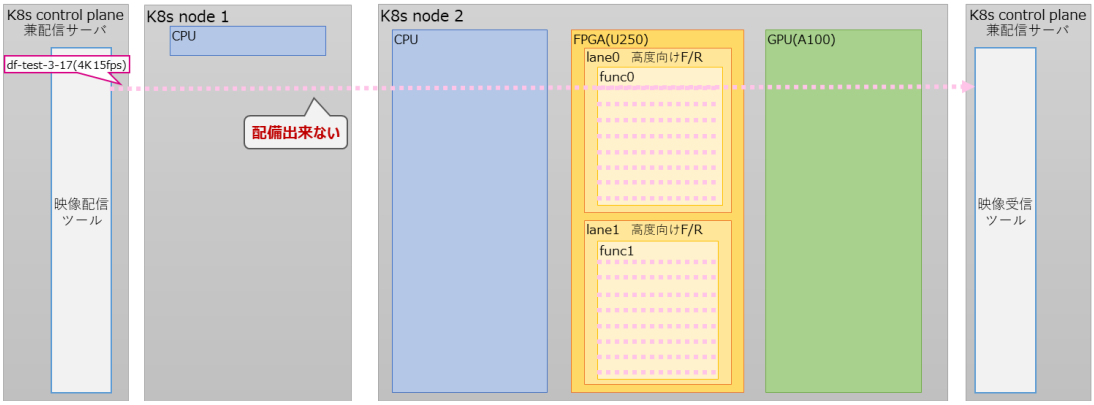

図 12： 16 本分配備・削除した後の 17 本目の DataFlow の配備イメージ

DataFlow の配備に失敗するのは、FPGA の全チャネルを使い切ったことで FPGA のリソースが枯渇しているからである。FPGA の全チャネルが使い切ったことを確認するには、以下の様に当該 FPGA に対応した ChildBs CR から確認出来る。

ChildBs CR の各 regions における modules.deploySpec.intraResourceMgmtMap において、各チャネルの使用状況が記録されている。Map の key がチャネルの ID、value が当該チャネルの状態を表す。

削除済みの DataFlow に割当てられていたチャネルの場合、value における”available”が”false”、”functionCRName”は値無しとなる。従って 3.3 節実施後では、全てのチャネルについて”available”が”false”、”functionCRName”は値無しとなる。

※ChildBs CR の詳細については別紙「Openkasugai-Controller_Attachment1（CR/CM 仕様書）」の「ChildBs」を参照のこと。

ChildBs の確認方法の例を示す。

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

出力結果の例は以下になる。lane0, lane1 の全てのチャネルにおいて available が”false”になっていることを確認する。

```
apiVersion: v1
items:
- apiVersion: example.com/V1
  kind: ChildBs
  metadata: (略)
  spec: (略)
  status:
    regions:
    - maxCapacity: 40                                          lane0 の状態に関する記録
      maxFunctions: 1
      modules:
        (略)
        functions: (略)
        - deploySpec: (略)
          id:0
          intraResourceMgmtMap:
            “0”:                          削除した Dataflow に割当てられていたチャネル ID0 に関する情報
              available: false                                 available が”false”,
            “1”:                          削除した Dataflow に割当てられていたチャネル ID1 に関する情報
              available: false                                 available が”false”,
            (略)
            “7”:                          削除した Dataflow に割当てられていたチャネル ID7 に関する情報
              available: false                                 available が”false”,
        (略)
      name: lane0
    - maxCapacity: 40                                          lane1 の状態に関する記録
      maxFunctions: 1
      modules:
        (略)
        functions: (略)
        - deploySpec: (略)
          id:0
          intraResourceMgmtMap:
            “0”:                          削除した Dataflow に割当てられていたチャネル ID0 に関する情報
              available: false                                 available が”false”,
            “1”:                          削除した Dataflow に割当てられていたチャネル ID1 に関する情報
              available: false                                 available が”false”,
            (略)
            “7”:                          削除した Dataflow に割当てられていたチャネル ID7 に関する情報
              available: false                                 available が”false”,
        (略)
      name: lane1
    state: Ready
    status: Ready
```

3.5 子 bs のリセット

対象：DataFlow を配備した FPGA カードを搭載している全ての K8s node

リソースを使い切った FPGA に対して子 bs のリセットを行い子 bs 書込み直後の状態に戻す。
ここでは別紙「OpenKasugai-Controller-InstallManual」の 1.1 節の図 1 の K8s node(“server2”)に搭載されている FPGA(デバイスファイルパスは”/dev/xpcie_21320621V01L”)に対して FPGA のリセットを行う場合の例を示す。

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

※なお、FPGAReconfigurationTool の使用方法については github(OpenKasugai)の controller リポジトリ(v1.1.0)の/src/tools/FPGAReconfigurationTool/ReadMe.txt を参照。

子 bs のリセットに成功すると、FPGA 内のチャネルも全て使用可能になるため、3.4 節で配備に失敗した DataFlow(df-test-3-17)についても再スケジューリングされ配備に成功する。

```
$ 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
```

(df-test-3-17 を 3.4 節で配備したまま削除していない場合を想定している。もし子 bs のリセットを行う前に当該 DataFlow を削除していた場合は改めて配備を再実施すること。)

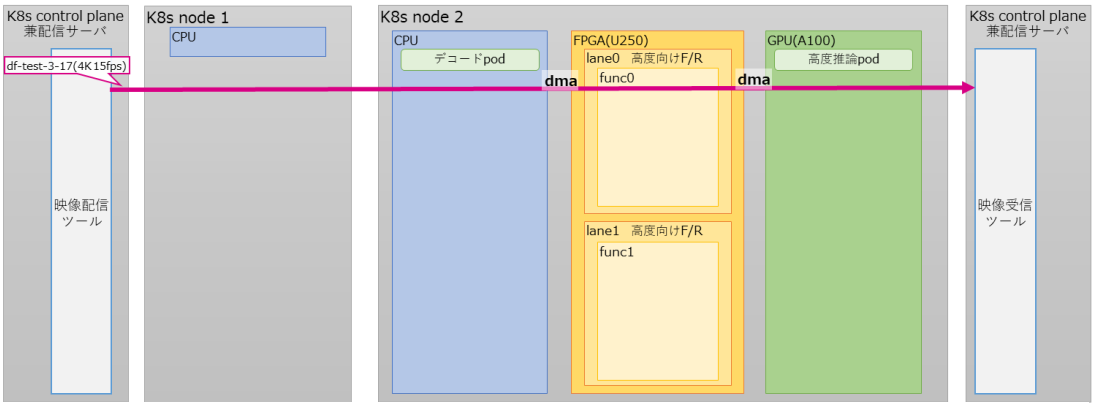

図 13： 子 bs のリセット実施後の DataFlow の配備イメージ

また、当該 ChildBs CR の状態も更新されている。

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

出力結果の例は以下になる。df-test-3-17 に割当てられたチャネル以外は(lane0 のチャネルも lane1 のチャネルも)全て available が”true”になっていることを確認する。

```
apiVersion: v1
items:
- apiVersion: example.com/V1
  kind: ChildBs
  metadata: (略)
  spec: (略)
  status:
    regions:
    - maxCapacity: 40                                   lane0 の状態に関する記録
      maxFunctions: 1
      modules:
        (略)
        functions: (略)
        - deploySpec: (略)
          id:0
          intraResourceMgmtMap:
            “0”:                                        ID0 のチャネルに関する情報(df-test-3-17 に割当て中)
              available: false                          available が”false”,
              functionCRName: df-test-3-17              functionCRName は当該 DataFlow 名
              rx: (略)
              tx: (略)
            “1”:                                        ID1 のチャネルに関する情報
              available: true                           available が”true”
            (略)
            “7”:                                        ID7 のチャネルに関する情報
              available: true                           available が”true”
        (略)
      name: lane0
    - maxCapacity: 40                                   lane0 の状態に関する記録
      maxFunctions: 1
      modules:
        (略)
        functions: (略)
        - deploySpec: (略)
          id:0
          intraResourceMgmtMap:
            “0”:                                        ID0 のチャネルに関する情報
              available: true                           available が”true”
            (略)
            “7”:                                        ID7 のチャネルに関する情報
              available: true                           available が”true”
        (略)
      name: lane1
    state: Ready
    status: Ready
```

3.6 環境停止

対象：K8s control plane

1.4 節を実施する。

4. 基本機能#4 のデモ手順

本章では FPGAReconfigurationTool の子 bs 書込み機能（子 bs 未書込み状態の FPGA に子 bs を書込む）に関するデモの手順を示す。本デモでは、初期状態のシステムに対して子 bs の書込みを行い、その後 FPGA を用いる DataFlow(1 章で配備した DataFlow と同じ構成)の配備を行う。

4.1 事前準備

4.1.1 試験スクリプトの配置

対象：K8s control plane、全ての K8s node

1.1.1 節を実施すること。既に実施している場合はスキップする。

4.1.2 DataFlow の yaml の編集

対象：K8s control plane

4.4.1 節で配備する各 DataFlow の yaml について、DataFlow の各処理モジュールの Pod が使用するネットワーク情報（IP アドレス・ポート番号）を必要に応じて変更する。

各サーバの 100G NIC に設定している IP アドレスが別紙「OpenKasugai-Controller-InstallManual」の 1.1 節の図 1 の IP アドレスと異なる場合は変更が必要となる。

変更方法は、別紙「OpenKasugai-Controller-InstallManual_Attachment1」のシート「2(補足).DataFlow yaml の設定」を参照すること。

表 6：4.4.1 節で配備する DataFlow の yaml 一覧

DataFlow の種類	編集対象のサンプルデータの yaml
1. FPGA F/R の DF 1 本目 (df-test-4-1)	~/controller/test/sample-data/sample-data-demo/yaml/dataflows/test-4/df-test-4-1.yaml

4.2 環境初期化

1.2 節を実施する。

4.3 子 bs の書込み

対象：手動での子 bs 書込みの書込み先となる FPGA カードを搭載している K8s node

FPGAReconfigurationTool を用いた場合は、FPGA 内の複数の配備領域(Lane)で実行する処理モジュール(フィルタリサイズなど)に対し、Lane 毎に異なるパラメータを設定することも可能である。（DataFlow 配備時に自動で子 bs を書込む場合は全ての Lane で同じパラメータを設定することしか出来ない）。
ここでは別紙「OpenKasugai-Controller-InstallManual」の 1.1 節の図 1 の K8s node(“server2”)に搭載されている FPGA(デバイスファイルパスは”/dev/xpcie_21320621V01L”)を書込み先とし、Lane 毎に異なるパラメータを設定する場合を例に示す。具体的には、2 つの Lane(lane0, lane1)のうち、lane0 には高度推論向けフィルタリサイズ用のパラメータを、lane1 には軽量推論向けフィルタリサイズ用のパラメータを設定する。

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

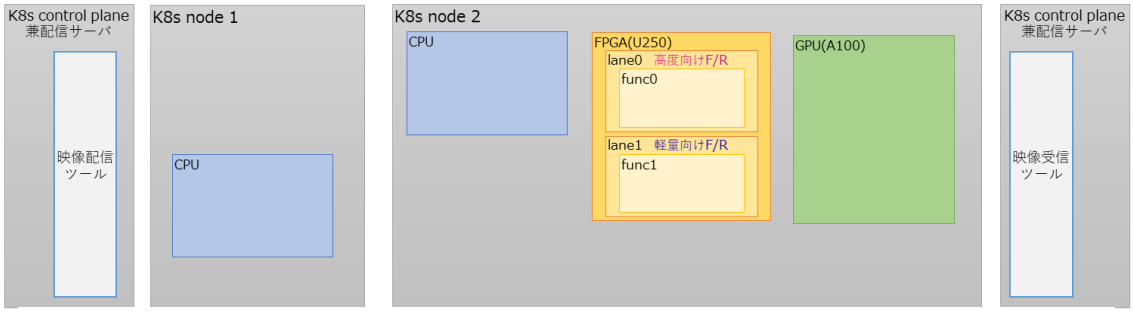

図 14： FPGAReconfigurationTool を用いた子 bs の書込み(Lane 毎に異なるパラメータを設定)

4.4 映像配信

4.4.1 DataFlow 配備

対象：K8s control plane

本節では DataFlow の配備と映像配信の流れを示す。

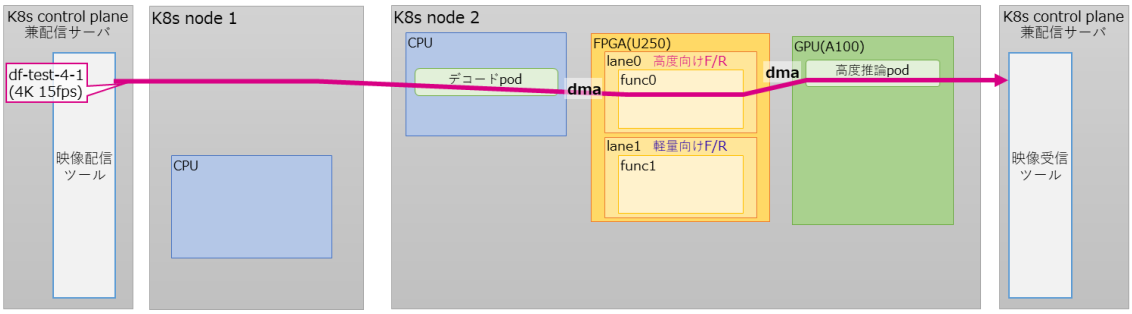

図 15： 本章のデモにおける DataFlow の配備イメージ

(1) DataFlow の配備

サンプルデータの df-test-4-1 の yaml を適用する。

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

(2) 配備結果の確認

1.3.1 節の(2)を実施する。

4.4.2 映像受信起動

対象：K8s control plane

2.3.2 節を実施する。

4.4.3 映像配信開始

対象：K8s control plane

2.3.3 節を実施する。

4.4.4 映像配信・受信停止

対象：K8s control plane

1.3.4 節を実施する。

4.4.5 DataFlow の削除

対象：K8s control plane

4.4.1 節で配備した DataFlow を削除する。

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

削除結果の確認を行う場合は 2.4.2 節を参照して、削除した DataFlow に関する CR や Pod が無いことを確認する。

4.5 環境停止

対象：K8s control plane

1.4 節を実施する。

5. [付録] 拡張機能のデモ手順

本章では、OpenKasugai コントローラの拡張機能のデモ手順として、コピー分岐の DataFlow と Glue の DataFlow の配備からデータ疎通確認までの手順を示す。

1. コピー分岐を用いて途中で複数ファンクションに分岐する DF を 1 本配備
   - df-test-ext-1-1：CPU デコード —(Ethernet 接続)→ CPU F/R —(Ethernet 接続)→ CPU コピー分岐 —(Ethernet 接続)→ GPU 高度推論×2

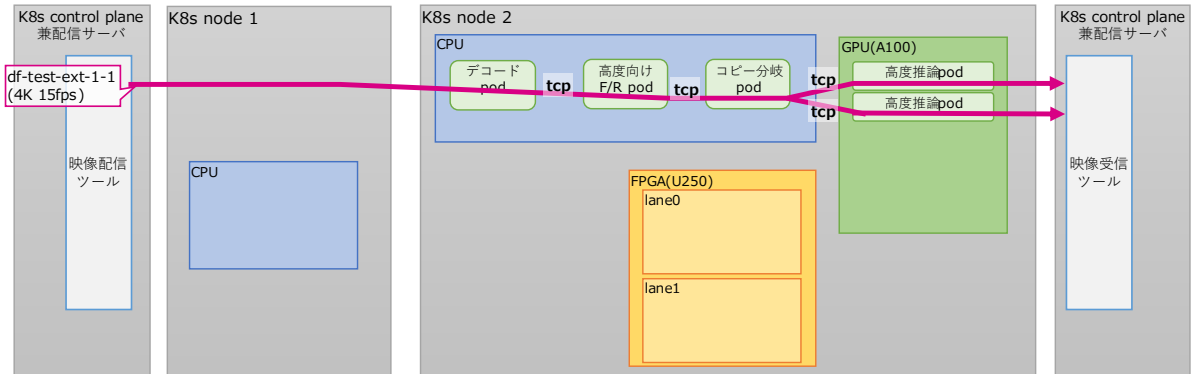

図 16：複数ファンクションに分岐する DF の配備イメージ

2. 接続種別の変換処理（Glue）を用いて接続種別が異なるファンクション同士を繋いだ DF を 1 本配備
   - df-test-ext-2-1：CPU デコード —(PCIe 接続)→ FPGA F/R —(PCIe 接続)→ CPU Glue —(Ethernet 接続)→ GPU 高度推論

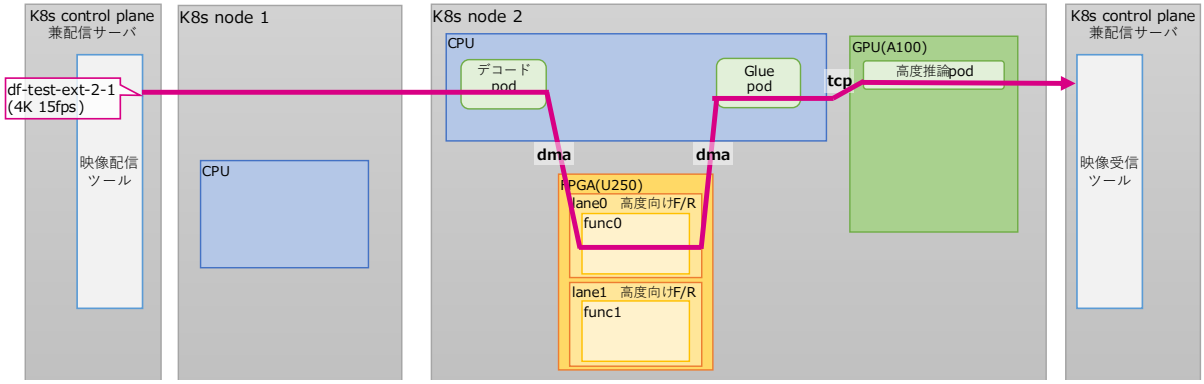

図 17：接続種別が異なるファンクション同士を繋いだ DF の配備イメージ

5.1 事前準備

5.1.1 試験スクリプトの配置

対象：K8s control plane、全ての K8s node

1.1.1 項を実施すること。既に実施している場合はスキップする。

5.1.2 DataFlow の yaml の編集

対象：K8s control plane

5.3.1 節で配備する各 DataFlow の yaml について、DataFlow の各処理モジュールの Pod が使用するネットワーク情報（IP アドレス・ポート番号）を必要に応じて変更する。
各サーバの 100G NIC に設定している IP アドレスが別紙「OpenKasugai-Controller-InstallManual」の 1.1 節の図 1 の IP アドレスと異なる場合は変更が必要となる。
変更方法は、別紙「OpenKasugai-Controller-InstallManual_Attachment1 のシート「2(補足).DataFlow yaml の設定」を参照すること。

表 7：5.3.1 節で配備する DataFlow の yaml 一覧

DataFlow の種類	編集対象のサンプルデータの yaml
1. コピー分岐の DF (df-test-ext-1-1)	~/controller/test/sample-data/sample-data-demo/yaml/dataflows/test-ext-1/df-test-ext-1-1.yaml
2. Glue の 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 環境初期化

1.2 節を実施する。

5.3 映像配信

5.3.1 DataFlow 配備

対象： K8s control plane

(1) DataFlow の配備

以下では 2 章の冒頭で示した DataFlow の配備方法を示す。

①コピー分岐の DF の場合

サンプルデータの df-test-ext-1-1 の DataFlow の yaml を適用する。

```
$ 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
（Deployed になるまで 20 秒ほど待つ）
```

②Glue 変換の DF の場合

サンプルデータの df-test-ext-2-1 の DataFlow の yaml を適用する。

```
$ 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
（Deployed になるまで 20 秒ほど待つ）
```

(2) 配備結果の確認

各カスタムリソースの正常生成と各 Pod の正常配備を kubectl get コマンドの投入結果（STATUS）で確認する。

①コピー分岐の DF の場合

コマンド実行結果から以下を確認する。

・各 DataFlow の STATUS が Deployed であること

・各 CPUFunction、GPUFunction 、EthernetConnection の STATUS が Running であること

・各 EthernetConnection の FROMFUNC_STATUS と TOFUNC_STATUS が OK であること

・各 Pod の STATUS が Running であること

```
$ 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
```

②Glue 変換の DF の場合

コマンド実行結果から以下を確認する。

・各 DataFlow の STATUS が Deployed であること

・各 CPUFunction、GPUFunction 、FPGAFunction、PCIeConnection、EthernetConnection の STATUS が Running であること

・各 PCIeConnection、各 EthernetConnection の FROMFUNC_STATUS と TOFUNC_STATUS が OK であること

・各 Pod の STATUS が Running であること

```
$ 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 映像受信起動

対象：K8s control plane

(1) 映像受信の起動方法

①コピー分岐の DF の場合

1.3.2 節と同じ手順を実施する。

②Glue 変換の DF の場合

映像配信ツールの配備と受信用スクリプト（高度推論結果受信×1）の作成

（初回のみ実施、2 回目以降は手順①は不要）

```
$ cd ~/controller/sample-functions/utils/rcv_video_tool/
$ kubectl create ns test
$ kubectl apply -f rcv_video_tool.yaml
$ kubectl get pod -n test    ※映像配信ツールの pod 名を確認する
$ kubectl exec -n test -it rcv-video-tool-xxx -- bash  ※xxx の部分は上記で確認した pod 名に合わせて変更
# vi start_test2.sh    ※以下を貼り付けて保存する
```

```
#!/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
```

新規ウィンドウで受信ツール用にターミナルを開き、Pod に乗り込んで映像受信を起動する。

```
$ kubectl exec -n test -it rcv-video-tool-xxx -- bash  ※pod 名は環境に合わせて変更
# ./start_test2.sh
```

映像受信ツールの起動確認

※ps auxwwf で COMMAND が¥_ gst-launch-1.0 から始まるプロセスが 1 個動いていることを確認する。

```
$ 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 映像配信開始

対象：K8s control plane

(1) 映像配信の開始方法

※コピー分岐の DF と Glue 変換の DF 共通

① 映像配信ツールの配備と配信スクリプト（4K 15fps(高度推論用)×1 本）の作成

（初回のみ実施、2 回目以降は本手順①は不要）

```
$ cd ~/controller/sample-functions/utils/send_video_tool/
$ kubectl apply -f send_video_tool.yaml
$ kubectl get pod -n test    ※映像配信ツールの pod 名を確認する
$ kubectl exec -n test -it send-video-tool-xxx -- bash  ※xxx の部分は上記で確認した pod 名に合わせて変更
# vi start_test2.sh     ※以下を貼り付けて保存する
```

```
#!/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
```

✧ 上記スクリプトに書く start_gst_sender.sh の引数説明

- ※1.3.3 節の(1)を参照

② 新規ウィンドウで配信ツール用にターミナルを開き、Pod に乗り込んで映像配信を開始する。

```
$ kubectl exec -n test -it send-video-tool-xxx -- bash  ※pod 名は環境に合わせて変更
# ./start_test2.sh
```

（初回のみ CRITICAL ログが出力される場合があるが Enter 押下でプロンプトに戻って良い。次手順でプロセスの起動が確認できれば問題ない）

③ 映像配信ツールのコンテナ内で起動確認

※ps aux で 1 本分 gst-launch が動いていることを確認

```
# 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) DataFlow の映像入出力確認

1.3.3 節の(2)と同じ手順で確認する。

5.3.4 映像配信・受信停止

1.3.4 節を実施する。

5.4 環境停止

1.4 節を実施する。